JAEA-Review
2008-038

JAEA-Review

Systemization of Burnup Sensitivity Analysis Code (II)
(Contract Research)

Masahiro TATSUMI* and Hideaki HYOUDOU*

Reactor Physics Analysis and Evaluation Group
Advanced Nuclear System Research and Development Directorate

August 2008

Japan Atomic Energy Agency

日本原子力研究開発機構

独立行政法人日本原子力研究開発機構　研究技術情報部　研究技術情報課
〒319-1195　茨城県那珂郡東海村白方白根 2 番地 4
電話 029-282-6387, Fax 029-282-5920, E-mail:ird-support@jaea.go.jp

JAEA-Review 2008-038

Systemization of Burnup Sensitivity Analysis Code (II)
（Contract Research）

Masahiro TATSUMI* and Hideaki HYOUDOU*

FBR System Technology Development Unit
Advanced Nuclear System Research and Development Directorate
Japan Atomic Energy Agency
Oarai-machi, Higashiibaraki-gun, Ibaraki-ken

(Received May 28, 2008)

Towards the practical use of fast reactors, it is a very important subject to improve prediction accuracy for neutronic properties in LMFBR cores from the viewpoint of improvements on plant economic efficiency with rationally high performance cores and that on reliability and safety margins.

A distinct improvement on accuracy in nuclear core design has been accomplished by the development of adjusted nuclear library using the cross-section adjustment method, in which the results of critical experiments of JUPITER and so on are reflected. In the design of large LMFBR cores, however, it is important to accurately estimate not only neutronic characteristics, for example, reaction rate distribution and control rod worth but also burnup characteristics, for example, burnup reactivity loss, breeding ratio and so on. For this purpose, it is desired to improve prediction accuracy of burnup characteristics using the data widely obtained in actual core such as the experimental fast reactor "JOYO".

The analysis of burnup characteristics is needed to effectively use burnup characteristics data in the actual cores based on the cross-section adjustment method. So far, a burnup sensitivity analysis code, SAGEP-BURN, has been developed and confirmed its effectiveness. However, there is a problem that analysis sequence become inefficient because of a big burden to users due to complexity of the theory of burnup sensitivity and limitation of the system. It is also desired to rearrange the system for future revision since it is becoming difficult to implement new functions in the existing large system.

It is not sufficient to unify each computational component for the following reasons: the computational sequence may be changed for each item being analyzed or for purpose such as interpretation of physical meaning. Therefore, it is needed to systemize the current code for burnup sensitivity analysis with component blocks of functionality that can be divided or constructed on occasion. For this purpose, the burnup sensitivity analysis code has synthesized with an object-oriented scripting language.

---

*Nuclear Fuel Industries, Ltd.
This document is a translation of JNC TJ9410 2004-002 published in Feburary 2005.
The translation was carried out with support from OECD/NEA.

In the present study, an examination was conducted for the two-layer controlling model of the conventional system using Python, the object-oriented scripting language. On the basis of the result in the examination, a new analysis system for burnup sensitivity, PSAGEP (Python-wrapped SAGEP-burn), was implemented. It was confirmed the effectiveness of the reconstruction method based on the two-layer controlling model of conventional system.

Keywords: Fast Reactor, Burnup Sensitivity Analysis, SAGEP-BURN, PSAGEP, Object-oriented, Python

燃焼感度解析コードのシステム化整備(II)

（委託研究）

日本原子力研究開発機構　次世代原子力システム研究開発部門
FBR要素技術ユニット

巽 雅洋*、兵頭 秀昭*

（2008年5月28日受理）

高速炉の実用化に向けて、高速炉実機炉心の核特性予測精度を向上させることは、合理的で高性能な炉心を設計してプラントの経済性の向上を図る上でも、信頼性および安全性の裕度をより高める上でも、極めて重要な研究課題となっている。

これまでの研究では、炉定数調整法を適用することにより、JUPITER等の臨界実験の成果を最大限有効に反映した統合炉定数を開発し、核設計精度の大幅な向上を達成している。一方、高速炉の炉心設計に於いては、臨界性、反応率、制御棒価値等のいわゆる静核特性だけでなく、燃焼反応度損失、増殖比といった燃焼核特性の精度良い評価も重要である。このためには、高速実験炉「常陽」等の豊富な実機燃焼データを有効に活用して、燃焼核特性の精度向上を図る必要がある。

炉定数調整法により、実機燃焼データを活用するためには、燃焼核特性の感度解析（以下、燃焼感度解析と呼ぶ）を行う必要がある。これまでに、燃焼感度解析を実施するためのコード（SAGEP-BURN）の開発が行われ、その有効性が確認されている。しかしながら、この燃焼感度の理論の複雑さと、システム上の制限から、ユーザへの負担が大きく解析作業が極めて非効率的であるという問題があった。また、システムの巨大化により機能の拡張が難しくなっているため、今後の機能拡張のために整理・統合が必要となっている。

一方、解析対象によって計算ステップが変わることや、物理的意味を分析する際には計算ステップを分解する必要があること等から、各計算機能を単純に統合するだけでは不十分である。各計算ステップは部品として保持したまま、必要に応じて部品を組み立てたり分解したりできるようにして、現在の燃焼感度解析コードをシステム化する必要がある。

本報告書は2005年2月に発行された報告書（JNC TJ9410 2004-002）の翻訳である。
この翻訳はOECD/NEAの支援を受けて行われたものである。
大洗研究開発センター（駐在）：〒311-1393 茨城県東茨城郡大洗町成田町4002
*原子燃料工業（株）

このため、オブジェクト指向とスクリプト言語の技術を利用して、燃焼感度解析コードのシステム化作業を実施した。
本研究では、オブジェクト指向スクリプト言語Pythonを用いた既存システムの二階層制御モデルについて検討を行った。その成果を元に、新燃焼感度解析システムPSAGEP (Python-wrapped SAGEP-burn)を実装した。これより、二階層制御モデルによる既存システムの再構築手法が、有効であることが確認された。

Contents

目 次

Table List

Figure List

This is a blank page.

1. Introduction

For the practical implementation of fast reactors, prediction accuracy must be improved for neutronic properties in LMFBR cores. This accuracy improvement will enhance plant efficiency, reliability and safety margins.

Accuracy improvements in nuclear core design have been accomplished with the development of an adjusted nuclear library using the cross-section adjustment method, in which the results of critical experiments of JUPITER and so on are reflected. In the design of large LMFBR cores, however, it is important to accurately estimate not only neutronic characteristics (criticality, reaction rate distribution and control rod worth), but also burnup characteristics (burnup reactivity loss, breeding ratio, etc). Prediction accuracy of burnup characteristics can be improved by using the data widely obtained in an actual core such as the experimental fast reactor "JOYO".

The intense analysis of burnup characteristics is required to effectively utilize burnup characteristics data in the power reactors based on the cross-section adjustment method. An existing burnup sensitivity analysis code, SAGEP-BURN, has been developed and confirmed in its effectiveness. However, complications exist with the analysis sequence because of its cumbersome nature, which arises from the complexity of the theory of burnup sensitivity, as well as inherent limitations of the code. Consequently, the system should be analyzed for further revision since it is becoming difficult to implement new functions in the existing environment.

Merely merging computational components is not sufficient since the computational sequence may vary depending on the burnup characteristics being analyzed, or when analyzing the consistency of resultant parameters. Therefore, it is necessary to establish a new burnup sensitivity analysis code that retains current calculation steps as components while combining necessary additional components for each problem.

In conjunction with the challenge of developing a more efficient analysis system, JNC has the task of orchestrating and managing the migration of the existing system. The initial concept of a next generation analysis system includes a two-layer controlling model: that is, a control-layer and a calculation-layer, employing the application of a multipurpose object-oriented scripting language. As a part of the multi-year project mentioned above, this year's project has two purposes: (1) the systemization of burnup sensitivity analysis code, and (2) planning the migration of the existing system to the subsequent system.

In the work last year, the burnup sensitivity analysis code was systemized using

the object-oriented scripting language, Python. The control layer for the existing Fortran program was designed and implemented. As a result, the Python language layer can control all the functions of existing SAGEP-BURN and the flexibility of input / output operation has been widely improved.

This year, to enhance accessibility of user interface, the input format and implementation of analysis scenarios were researched. PSAGEP (Python-wrapped SAGEP-burn) was developed based on the research results. It is apparent that the new system design offers a higher flexibility in analysis than the previous system.

2. PSAGEP Design Details

This chapter explains: (1) abstract of burnup sensitivity analysis code SAGEP-BURN systemization, (2) results of design planning and improvement from work done last year, (3) sharing input information among modules as well as analysis cases, and (4) the principle of system operation.

2.1. Abstract of Burnup Sensitivity Analysis Code SAGEP-BURN Systemization

SAGEP-BURN is a large system that consists of system modules for burnup sensitivity analysis shown in Table 2-1. **System modules** are also called the **calculation codes in the backend** of PSAGEP while the control layers are in the front end. It is necessary to designate input files for each independent system module because each system module is independent. Information exchange between modules is done through file I/O and the entire system is controlled by a shell script. A sample of the calculation flow of the breeding ratio is shown in Fig. 2-1 as an example. As seen in this figure, versatility of analysis flow causes a complicated structure. This introduces difficulties in creating calculation script files. It is troublesome to modify shell scripts to analyze different problems because calculation flows differ by the burnup characteristics analyzed. Hence, the disadvantages of the existing SAGEP-BURN were studied, and an improvement of input/output file formats and system module controls was planned. It was concluded that the most promising avenue was to introduce a two-layer controlling model using Python (that is an object-oriented scripting language). The two-layer controlling model is one of systemization methods. It consists of an encapsulated layer that operates system modules from Python and a control-layer that operates encapsulated system modules.

System development was carried out in phases: phase 1 (fiscal year of 2003) and phase 2 (FY 2004). In phase 1, design and implementation of the encapsulated layer using Python was done. Twenty-two system classes and corresponding test classes were implemented. The Agile development method was introduced to improve work efficiency. This method is an iterative and incremental development cycle that has gained recognition recently. It applies a testing strategy called “Test First Programming”. Test First Programming is the method of making a test program before implementing the main program. Test programs implemented in this development were: “Unit test” (those that test operation of each class), and “Join test” (those that test correlation between

objects). In phase 1, program validation was verified by comparing results of the breeding ratio burnup sensitivity analysis on a reference problem run on both the new and the existing systems.

The Agile method was used in phase 2, also. A control layer was implemented, and a control method of analysis sequence, as well as management of input meta data were studied before implementing the control layer. This newly designed PSAGEP liberates users from the burden of operating complicated assemblies. The following sections describe items researched in phase 2.

Table 2-1 Breakdowns of SAGEP-BURN System Modules

System Module	Function
START-UP	Generates macro cross sections.
CITATION	Diffusion calculation code; derives neutron flux and adjoint neutron flux.
FIRE-1	Burnup calculation code; derives power-normalization factor of neutron flux, solves burnup formula to derive number density time change, and solves adjoint burnup formula temporally adversely to derive adjoint number density.
FIRE-2	Burnup calculation code; solves burnup equation by inputting output-normalized factor from FIRE-1 to derive number density of assembly of interest.
FIRE-3	Burnup calculation code; solves adjoint burnup equation by inputting output-normalized factor from FIRE-1 to derive adjoint number density when deriving sensitivity coefficient of number density.
NS-INI	Initiates adjoint number density when deriving number density sensitivity coefficient.
NS-INI2	Initiates adjoint number density when deriving effective multiplication factor sensitivity coefficient.
NS-INI3	Initiates adjoint number density when deriving burnup reactivity loss sensitivity coefficient. This module is used when considering discontinuity by refueling.
NS-INI4	Initiates adjoint number density when deriving breeding ratio sensitivity coefficient.
NS-INI5	Initiates adjoint number density when deriving control rod worth sensitivity coefficient.
NS-INI6	Initiates adjoint number density when deriving reaction rate sensitivity coefficient.
NS-JUMP	Calculates adjoint number density discontinuity between the end of time step and the beginning of subsequent time step to derive adjoint number density in subsequent step.
REFUEL	Controls number density including refueling.
REFUEL2	Controls adjoint number density including refueling.
REFUEL3	Controls adjoint number density including refueling while considering adjoint number density discontinuity on burnup reactivity loss.
SCGIVE	Calculates adjoint reactor power P* to derive second term (number density) and fifth term (reactor power) of sensitivity coefficient. Calculates source term to derive adjoint generalized neutron flux.
SAGEP93	Calculates first term (direct term) and third term (neutron flux) of sensitivity coefficient. Calculates adjoint reactor power and source term to derive breeding ratio sensitivity coefficient and reaction rate sensitivity coefficient.

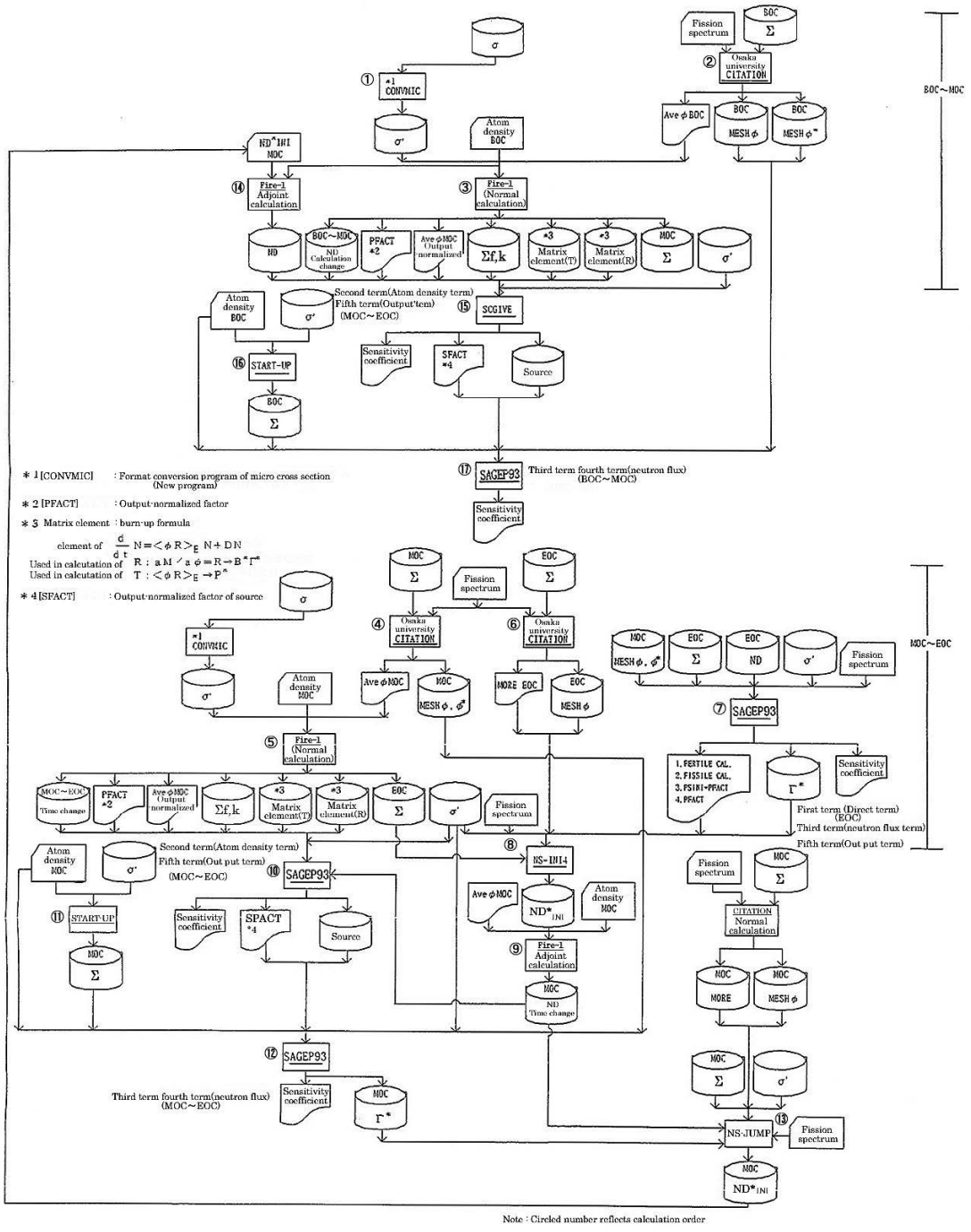


Fig. 2-1 Calculation flow of breeding ratio sensitivity coefficient (extracted from reference 2).

2.2. Review of Phase 1 Design Proposal

The encapsulated layer was designed and implemented in phase 1. Features of the control layer in phase 2 were studied when phase 1 was completed. A diagram of PSAGEP system classes was made, as seen in Fig. 2-2. This diagram was made using the following concepts:

- Input information being grouped and classified
- Classified groups being generalized into *Info* classes
- *InfoSet* being defined as something that stores instances of *Info* classes
- *InfoSetManager* storing instances of *InfoSet* classes

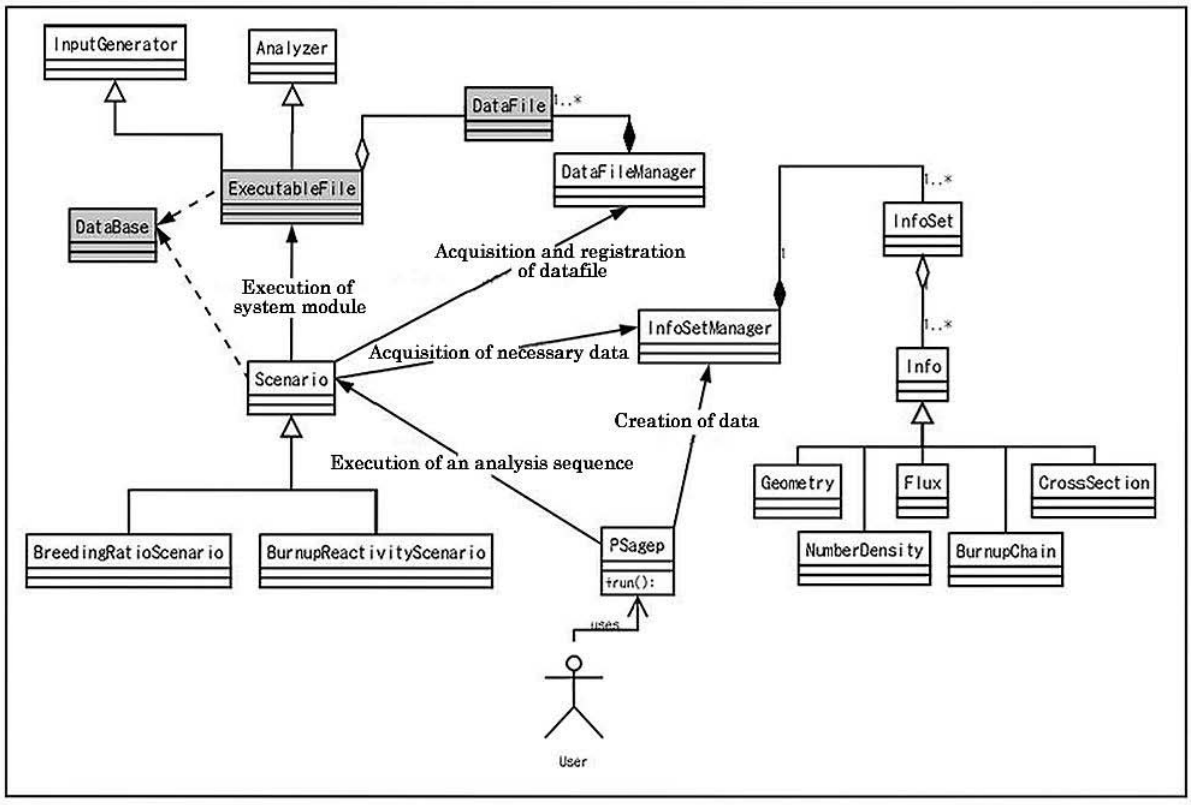


Fig. 2-2 Phase 2 design proposal (class diagram) from Phase 1 report in 2003.

Phase 2 development of this project started and encountered the following retrospective questions:

- Is it necessary to group the *Info* class instances as *InfoSet*?

- Is it possible to use *InputGenerator* as a generalized *ExecutableFile*? Or, is it better to separate *InputGenerator* as a "management class" like *DataFileManager*?
- Can the *Analyzer* class that processes calculation results and *InputGenerator* class be on a similar position venue?

To answer these questions, the relationships between classes were reviewed. It was concluded that class diagrams should be modified with the following policies:

- Omit *InfoSet*. Have *InfoManager* directly control all information.
- Separate *InputGenerator* and *Analyzer* as independent "control classes".

As the result, object classes could be categorized into 3 types, i.e. meta information class (*Info* system), file-related class (*File* system), and management class (*Manager* system). Table 2-2 shows the typical classes of each category.

Table 2-2 Object Class Category and Typical Classes

Class Category	Class Name	Function
Meta Information-related Classes	Info	Abstract data class
	Nuclide	Expression of nuclide
	Material	Nuclide and number density
	Region	Region information
	Geometry	Geometry information
	BurnupChain	Burnup chain
	SensitivityOption	Sensitivity options
	ConvergenceCondition	Convergence conditions
File Classes	DataFile	General data file
	ExecutableFile	Encapsulated executable module
	DataBase	Database
Management Classes	Scenario	Analysis scenario
	InfoManager	Input information management
	InputGenerator	Creates input to system module
	DataFileManager	Data file control
	OutputManger	Generates results file

The class diagram shown in Fig. 2-3 was created as a result of reviewing the class relationship on a standard analysis sequence. Based on this diagram, each class in the control layer was designed in detail and implemented.

Meta information-related classes store various calculation conditions. Specific content is defined in the input file supplied by users. File-related classes encapsulate data files and executable files as objects. Management classes deal with objects of different information, files, and control analysis processes. According to the two-layer controlling model, meta information-related classes and file-related classes correspond to the encapsulated layer, while the management classes correspond to the control layer.

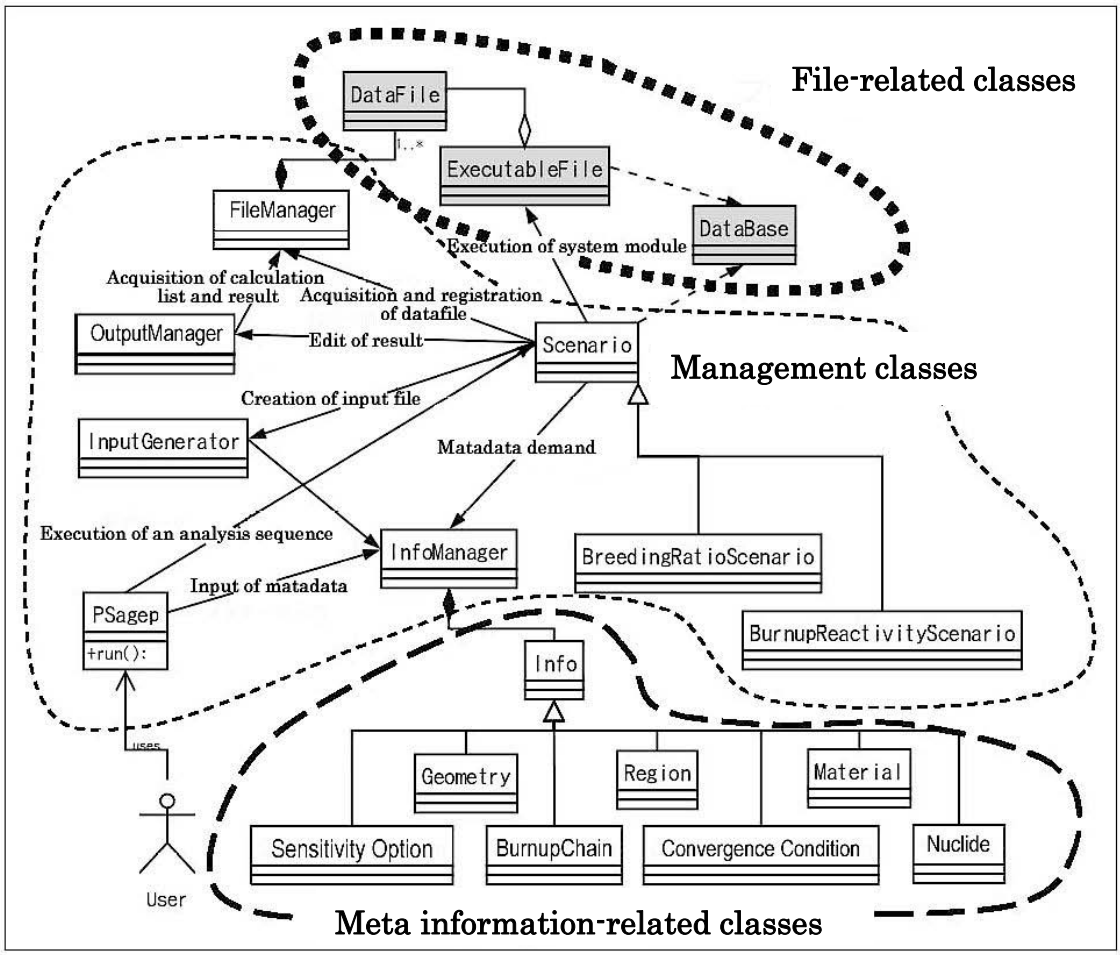


Fig. 2-3 Class diagram of phase 2 system after design review.

2.3. Operation Principle

It is important to hide the existing operation principles completely in phase 2. This means that users do not have to see the input / output detail of the existing system. Users can employ the phase 2 system simply by possessing knowledge of the methodology of burnup sensitivity analysis. Hence, it was necessary to design a new data input format independent from the existing system.

In this section, (bearing the former assertion in mind) necessary input information was studied for each burnup sensitivity analysis using actual analysis cases. Commonalities of input information to system modules between calculation codes and also between analysis cases were studied.

2.3.1. Input Meta Information

For calculation control, it is necessary to have a structure that automatically processes analysis condition information given by users; this allows each subsystem to recognize these analysis conditions. Initially, information for breeding ratio analysis scenarios was studied. Table 2-3 shows results of the study. Refer to reference No. 3 for more information about codes in the table.

Input information users should designate is hereafter referred to as “input meta information”. Table 2-3 suggests the following as input meta information necessary for analysis:

- Designation of nuclide (type and number of nuclide(s), and PDS file name)
- Number of region (sub-region, number of grid in axial, and number of refueling assembly)
- Material (number density, and number of material region)
- Calculation conditions (mesh nodes, number of mesh divisions, boundary conditions, convergence conditions, and number of energy group)
- Burnup conditions (number of cycles, number of burnup nuclides, cooling period, decay constant, burnup chain matrix, reactor power, burnup period, burnup sub-step, and number of steps)
- Others (code specific option, and analysis case specific option)

In the existing system, it was necessary to generate separate input files for each system module, each analysis case, and each burnup cycle. For this reason, users must modify all files that contain the necessary information. Modifying many files was

troublesome. Thus, the new system introduces central controls to manage input meta information so that modified information is automatically distributed to related modules. Moreover, users do not have to witness the information transfer in the new system.

Table 2-3 Input Meta Information to Calculation Codes (Breeding Ratio Analysis Case)

Item ＼ Code	convmic	pdsedit	refuel	startup	citation	fire1	fire2	nsini4	nsjump	refue12	scgive	sagep93
Number of nuclides	○ 33		○ 33	○ 33								
Nuclide ID (JENDL)	○ 925 *3		○ 925					○ 925				
Number of regions		○ 16	○ 16	○ 16		○ 16	←				○ 16	○ 16
Number of sub- regions			○ 4									
Numbers of axially- regions			○ 4									
Number of assemblies (refuel) of regions			○ 1							○ 1.0 *3		
Number of energy groups		○ 18		○ 18		○ 18	←				○ 18	○ 18
PDS file name		○ IC01 *3								○ 1 *3		
Setting of the number of cycle			○ 1 *3									
Atom density			○ *4									
Number of burn-up nuclides			○ 21			○ 21	←	○ 21	○ 21		○ 21	
Decay constant			○ 2E-09							○ 2E-09		
Days of cooling			○ 42							○ 42		
Specific option of Calculation codes												
Number of iterations					○ *1			○ *1	○ *1			○ *2 Γ
Convergence conditions					○ *1			○ *1	○ *1			○ *2 Γ
Number of material regions					○ *1			○ *1	○ *1			
Number of mesh intervals					○ *1			○ *1	○ *1			○ *2
Mesh width					○ *1			○ *1	○ *1			○ *2
Material map					○ *1			○ *1	○ *1			○ *1
Boundary conditions					○ *1			○ *1	○ *1			
Burn-up matrix information						○	←	△ *5				○
Heat output						○ 4E-06	←		○ 4E-06			○
Burn-up period						○ 5E-07	←				○ 5E-07	
Number of Burn-up steps							←					
Number of Burn-up sub-steps												
Number of steps of interest												
Option for sensitivity calculation												○

○: Supplying Information is necessary. (An example is shown as a remark).

*1 Possible to share. *2 Different in form. *3 And others. *4 Each assemlby. *5 Partially applied

2.3.2. Managing Input Meta Information and Generating Input File

The system has been designed so that input meta information mentioned in the previous chapter is managed exclusively by the *InfoManager* instance. As seen in Fig. 2-3, input files to each calculation code are generated by the *InputGenerator* instance using input meta information. That is, the *InputGenerator* instance extracts necessary information from *InputManager* and outputs it in a format specific to each calculation code.

Even in the same calculation code, contents of the input files are not the same when analyzing different parameters. However this is no longer a problem because the *Scenario* class will control the *InputGenerator* instance. For example, precise management is possible by designating the operation mode: i.e., the normal calculation mode and adjoint calculation mode when booting up the *InputGenerator* class method.

2.3.3. Managing Analysis Scenarios

In burnup sensitivity analysis, analysis sequences vary when neutronic characteristics differ. That is, the type of calculation codes and their execution orders differ depending on analysis cases. All analysis sequences cannot be combined because the type and content of connecting input / output files between different calculation codes will vary. Hence, this system is designed to have the *Scenario* class manage analysis scenarios. Actual analysis sequences are defined in subclasses under the *Scenario* class.

Under the *Scenario* class (actually its subclass), the analysis sequence runs the following procedure:

- Booting up *InputGenerator* method; this generates the input file.
- Obtaining the file from *FileManager* to connect to the calculation code.
- Booting up the wrapper class run() method; and run() method running the calculation code.
- Registering the output file derived by the calculation into *FileManager*.

This procedure corresponds to the executable shell script in the existing analysis procedure. However, it is a significant labor saver in managing input / output files because the dependency of input / output between different codes is automatically solved.

2.3.4. Similarity of Input between Analysis Cases

As mentioned, in the existing system it was necessary to provide the input file specific to each calculation code, as well as each cycle and the beginning / end of burnup. Input information is contained in different files but most of the parts are commonly used by different calculation codes. *InputGenerator* enables a dynamic generation of input files by extracting necessary information from different input meta information files.

The similarity of input files between different analysis cases in each calculation code is studied here, using breeding ratio burnup sensitivity analysis (case BR) as an example. The input file format is categorized and provided for implementation of *InputGenerator*.

Figures 2-4 to 2-12 show the results of considering this similarity. Notations are as follows -- B: beginning of burnup (BOC); E: end of burnup (EOC); adj: adjoint calculation; and numbers: cycle numbers, respectively. Symbol "×" denotes there were not similarities; "△","○","◎" denote there were similarities. Investigations of case combinations were terminated when rules were found instead of investigating all the combinations. The results breakdown is shown below:

Table 2-4 Input Format Similarities between Analysis Cases in Each Calculation Code

Calculation Code	Input Format Similarity between Cycles
Refuel	- Originally, cycle-1 and cycle-2 have different input format. - Refueling and burnup cycles have different input format. - BOC case → Refuel: IOPT = 0; EOC case → Burnup: IOPT = 1
Citation	- Unique region labels to edit at most. - Only adjoint calculation option is unique between Forward and Adjoint calculations.
Fire1	- Unique region labels to edit at most. - No particular difference between Forward and Adjoint.
Fire2	- Unique region labels to edit at most.
Startup	- Available without any modification.
Nsini4	- (Only one input file).
Scgive	- Unique region labels to edit at most.
Sagep	- Unique region labels to edit at most.
Refuel2	- Cycle number information from last cycle is unique. - There are similarities between BOC of each cycle and between EOC of each cycle. - Exception: EOC of Cyc04 differs from that of other cycles. (Additional block)

2.3.5. Input Similarity between Analysis Scenarios

Following the analysis case similarity investigation, the analysis scenario similarities were investigated. Figures 2-13 to 2-23 manifest results of similarities between analysis scenarios of each calculation code. Notations in these figures are explained in Table 2-5 and the symbols utilized are the same as those in the previous investigation.

Table 2-5 Analysis Scenarios and Endpoints

Scenarios	Endpoints
Br	Breeding ratio
Burnup	Burnup reactivity loss
keff_eoc	Effective multiplication factor (at EOC)
nav_eoc	Sodium void reactivity (at EOC)
nd-focused	Number density of interested nuclide
rr_eoc	Reaction rate (at EOC)

Table 2-6 Input Format Similarities between Analysis Scenarios in Each Calculation Code

Calculation Code	Input Format Similarity between Scenarios
Refuel	・ Almost completely shared. ・ Only Cyc01-BOC has different **nd-focused**.
Startup	・ Independent from scenarios and completely shared,.
Citation	・ Buckling calculation option is on for **keff**, **nav**, **nd**, **rr**. ・ However, it is assumed that there is no damage essentially even when sharing all.
Fire1	・ Identical.
Fire1(adj)	・ Identical. ・ However, nd-focused does not perform adjoint calculation.
Fire2	・ Identical.
Nsini4	・ (Only one input file).
Refuel2	・ Almost identical.
Scgive	・ Almost identical. ・ However, Designation of several parameters differ in nd-focused.
Sagep-flx	・ Convergence condition differ in nd-focused. ・ However, it is possible to unify.
Sagep-direct	・ Basically, identical even if some region labels differ.

The conclusion from the above-mentioned investigation is that similarities between input formats of analysis scenarios are high in each calculation code. Hence, further

investigation needs to focus on typical scenarios first, followed by investigation of other scenarios. As an example, the breeding ratio (BR) analysis scenario was studied here.

Code: **Refuel**

Notes about input:

- Originally, cycle-1 and cycle-2 have different input formats.
- Refueling and burnup cycles have different input formats.
- BOC case→refueling　　EOC case→burnup

	B-1	E-1	B-2	E-2	B-3	E-3	B-4	E-4
B-1	■	×	△	×	△	×	△	×
E-1		■	×	○	×	○	×	○
B-2			■	×	○	×	○	×
E-2				■				
B-3					■			
E-3						■		
B-4							■	
E-4								■

Difference exists only in 1cycle.

◎ Available without any modification
○ One part of the parameters is unique
△ There are common parts except a block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-4 Similarity between analysis cases of input file to Refuel code.

Code: **CITATION**

Notes about input:

- Unique region labels to edit.
- Only the adjoint calculation option is unique between Forward and Adjoint calculations.

	B-1	E-1	B-2	E-2	B-3	E-3	B-4	E-4
B-1	■	○	○	○	○	○	○	○
E-1		■						
B-2			■					
E-2				■				
B-3					■			
E-3						■		
B-4							■	
E-4								■

```
*** CIT-cyc01-BOC     Thu Jun 24 16:00:03 2004
--- CIT-cyc04-EOC     Thu Jun 24 16:00:03 2004
**************
*** 33.38 ****
     0.0   0.4692   0.4692   0.4692   0.0     0.0   0.0
      17      17
      16
! KEFF 01B
! AVFX 01B
! VOL  01B
--- 33.38 ----
     0.0   0.4692   0.4692   0.4692   0.0     0.0   0.0
      17      17
      16
! KEFF 04E
! AVFX 04E
! VOL  04E
```

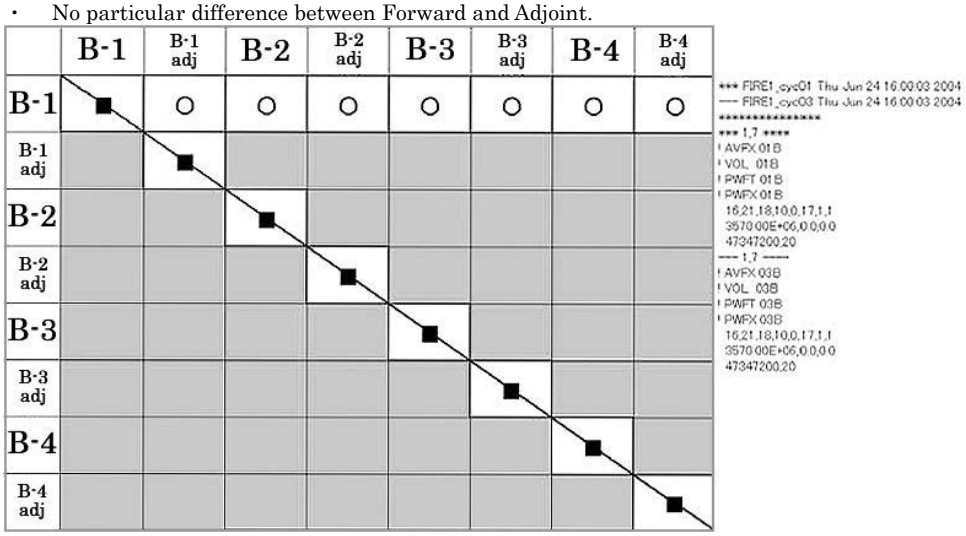
◎ Available without any modification
○ One part of the parameters is unique
△ There are common parts except the block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-5 Similarity between analysis cases of input file to Citation code.

Code: **FIRE1**

Notes about input:

- Unique region labels to edit.
- No particular difference between Forward and Adjoint.



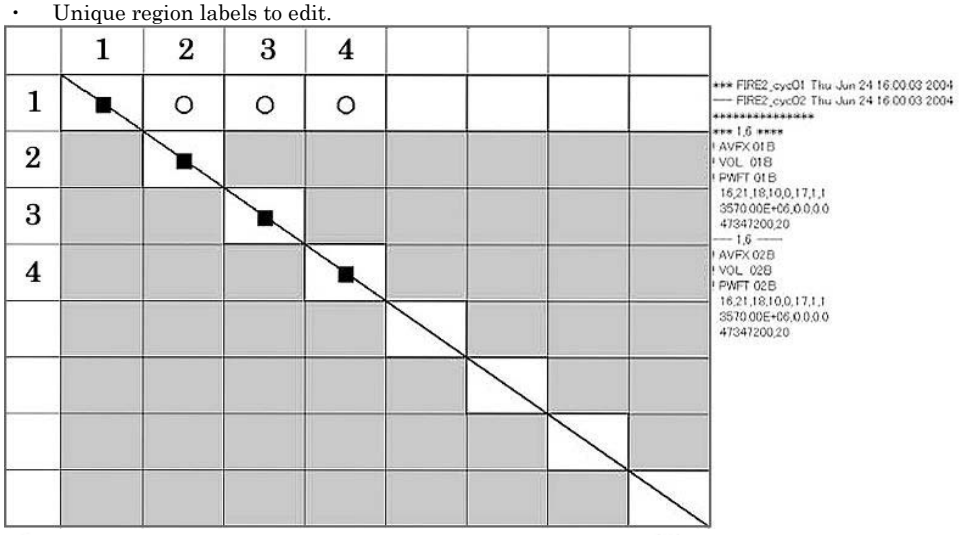
◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-6 Similarity between analysis cases of input file to Firel code.

Code: **FIRE2**

Notes about input:

- Unique region labels to edit.



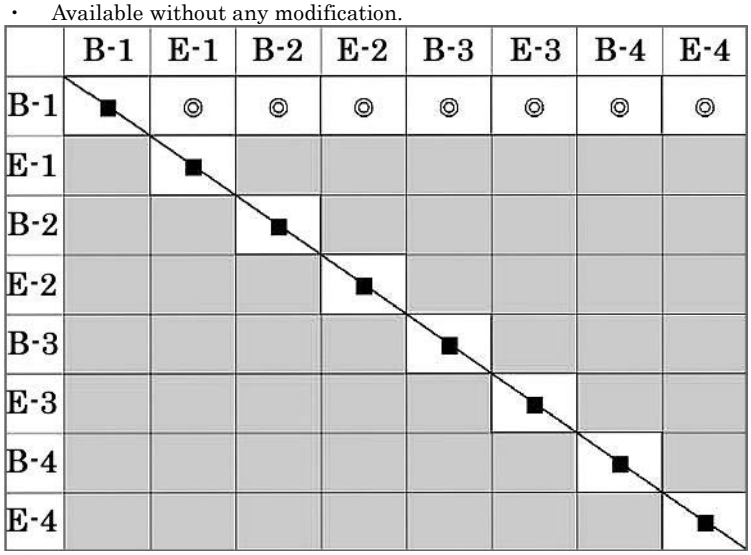
◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-7 Similarity between analysis cases of input file to Fire2 code.

Code: **STARTUP**

Notes about input:

・ Available without any modification.



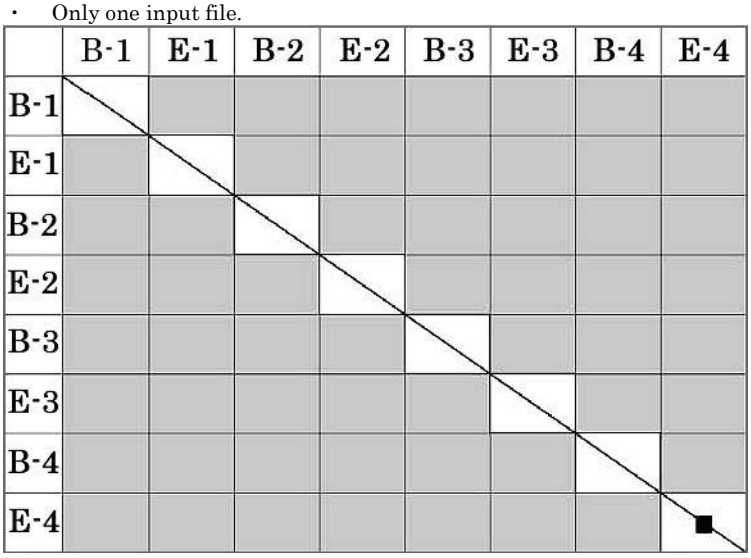
◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-8 Similarity between analysis cases of input file to Startup code.

Code: **NSINI4**

Notes about input:

・ Only one input file.



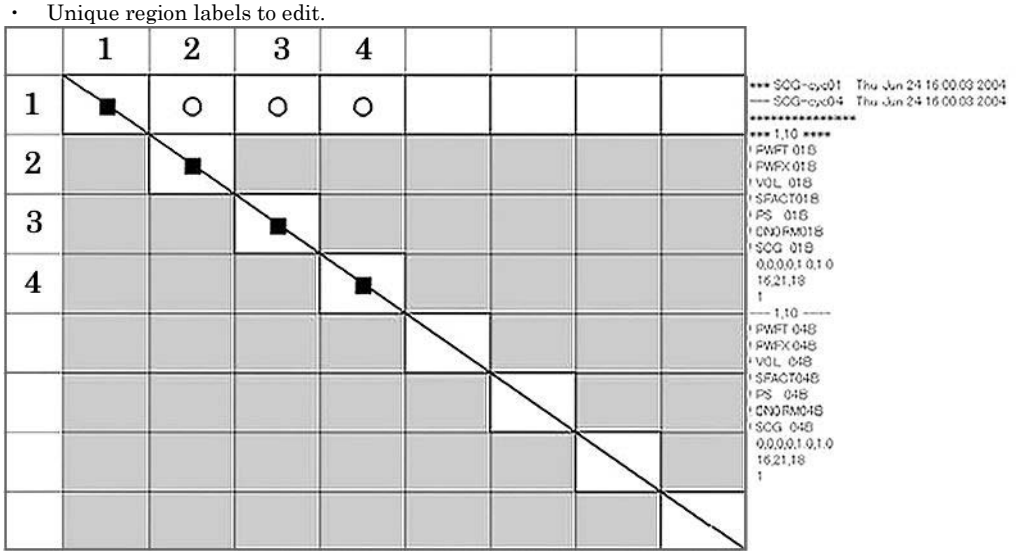
◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-9 Similarity between analysis cases of input file to Nsini4 code.

Code: **SCGIVE**

Notes about input:

・ Unique region labels to edit.



◎ Available without any modification

○ One part of parameters is unique

△ There are common parts except block is unique.

× Widely different

■ Be defined

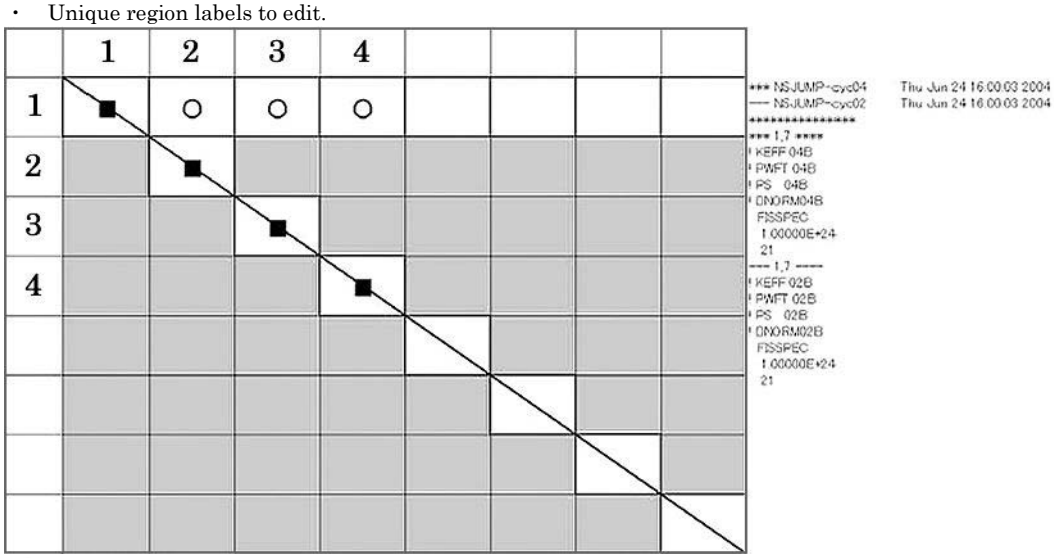
Investigation is omitted.

Fig. 2-10 Similarity between analysis cases of input file to Scgive code.

Code: **SAGEP**

Notes about input :

・ Unique region labels to edit.



◎ Available without any modification

○ One part of parameters is unique

△ There are common parts except block is unique.

× Widely different

■ Be defined

Investigation is omitted.

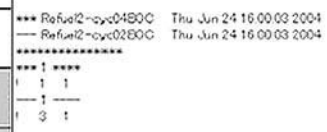
Fig. 2-11 Similarity between analysis cases of input file to Sagep code.

Code: **REFUEL2**

Notes about input:

- Cycle number information from last cycle is unique.
- There are similarities between BOC of each cycle and between EOC of each cycle.
- Exception: EOC of Cyc04 differs from that of other cycles.(Additional block)

	B-1	E-1	B-2	E-2	B-3	E-3	B-4	E-4
B-1	■							
E-1		■						
B-2			■					
E-2				■				
B-3					■			
E-3		○		○		■		
B-4		×	○	×	○	×	■	
E-4	×	×	×	×	×	×	×	■



◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-12 Similarity between analysis cases of input file to Refuel2 code.

Code: **Refuel**

Notes about input:

- Only Cyc01-BOC has different nd-focused.

	br	burnup	keff_ecc	nav_ecc	nd-focused	rr_eoc
br	■	◎	◎	◎	△	◎
burnup		■				
keff_ecc			■			
nav_ecc				■		
nd-focused					■	
rr_eoc						■

◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-13 Similarity between analysis scenarios of input file to Refuel code.

Code: **STARTUP**

Notes about input:

- Identical

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	◎	◎	◎	◎
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■

◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-14 Similarity between analysis scenarios of input file to Startup code.

Code: **CITATION**

Notes about input:

- Buckling calculation option is on for keff, nav, nd, rr.
- However, sharing is possible.

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	○	○	○	○
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■

◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-15 Similarity between analysis scenarios of input file to Citation code.

Code: **FIRE1** (Forward)
Notes about input:
・ Identical

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	◎	◎	◎	◎
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused						
rr_eoc						■

◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-16 Similarity between analysis scenarios of input file to Fire1 code (Normal calculation).

Code: **FIRE1** (Adjoint)
Notes about input:
・ Identical
・ However, nd-focused does not perform adjoint calculation

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	◎	◎		◎
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused						
rr_eoc						■

◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-17 Similarity between analysis scenarios of input file to Fire1 code (Adjoint calculation).

Code: **FIRE2**

Notes about input:

・ Identical

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	◎	◎	◎	◎
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■

◎ Available without any modification　　× Widely different

○ One part of parameters is unique　　■ Be defined

△ There are common parts except block is unique.　　Investigation is omitted.

Fig. 2-18 Similarity between analysis scenarios of input file to Fire2 code.

Code: **NSINI4**

Notes about input:

・ Only one input file.

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br						
burnup						
keff_eoc						
nav_eoc						
nd-focused						
rr_eoc						

◎ Available without any modification　　× Widely different

○ One part of parameters is unique　　■ Be defined

△ There are common parts except block is unique.　　Investigation is omitted.

Fig. 2-19 Similarity between analysis scenarios of input file to Nsini4 code.

Code: **REFUEL2**

Notes about input:

・ Identical

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■					
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc	◎	◎	◎	◎	◎	■

◎ Available without any modification　　× Widely different

○ One part of parameters is unique　　■ Be defined

△ There are common parts except block is unique.　　Investigation is omitted.

Fig. 2-20 Similarity between analysis scenarios of input file to Refuel2 code.

Code: **SCGIVE**

Notes about input:

・ Almost identical.

・ However, designation of several parameters differs in nd-focused.

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	◎	◎	○	◎
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■

```
*** br/Burn/SCG-cyc04  2004-06-24
16:00:03.000000000 +0900
--- nd-focused/Burn/SCG-cyc04  2004-06-24
16:00:02.000000000 +0900
**********
*** 5,11 ****
 PS  04B
 DNORM04B
 SCG  04B
 0,0,0,0,1.0,1.0
 16,21,18
 1
 47347200,20,3570.00E+06
--- 5,11 ----
 PS  04B
 DNORM04B
 SCG  04B
 1,949,  7,  1,1.0,1.0
 16,21,18
 1
 47347200,20,3570.00E+06
```

◎ Available without any modification　　× Widely different

○ One part of parameters is unique　　■ Be defined

△ There are common parts except block is unique.　　Investigation is omitted.

Fig. 2-21 Similarity between analysis scenarios of input file to Scgive code.

Code: **SAGEP-Flx**

Notes about input:

- Convergence conditions differ in nd-focused.
- However, it is possible to unify.

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	◎	◎	○	◎
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■

◎ Available without any modification

○ One part of parameters is unique

△ There are common parts except block is unique.

× Widely different

■ Be defined

Investigation is omitted.

Fig. 2-22 Similarity between analysis scenarios of input file to Sagep code (Neutron term calculation).

Code: **SAGEP-Direct**

Notes about input:

- Basically, identical even some region labels differ.

	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	○	○	○	○	○
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■

```
** ../../burnup/Burn/SGP-cyc04-EOC-direct     Thu Jun 24 15:59:52 2004
--- ../../keff_eoc/Burn/SGP-cyc04-EOC-direct   Thu Jun 24 15:59:59 2004
***************
*** 1,12 ****
  FISSPEC
  KEFF 04E
  SFACT
! DAMMY04E
! DAMMY04E
! DAMMY04E
! DAMMY04E
! DAMMY04E
! DAMMY04E
  H14 NA-MOX Breeding Ratio  cyc04 EOC DIRECT
    0 1  0  0 18 16  0
    1  0  0  0
--- 1,12 ----
  FISSPEC
  KEFF 04E
  SFACT
! PFACT04E
! BRFET04E
! BRFIS01E
! BRFI204E
! PSIN804E
! SGPFX04E
  H14 NA-MOX Breeding Ratio  cyc04 EOC DIRECT
    0 1  0  0 18 16  0
    1  0  0  0
***************
*** 64,68 ****
  RDS1      SLAROM
    16
  RDS2      SLAROM
! H13 NA-MOX Breeding Ratio  cyc04 EOC DIRECT
     -1    3    0
--- 64,68 ----
  RDS1      SLAROM
    16
  RDS2      SLAROM
  H13 NA-MOX KEFF          cyc04 EOC DIRECT
     -1    3    0
```

◎ Available without any modification
○ One part of parameters is unique
△ There are common parts except block is unique.
× Widely different
■ Be defined
Investigation is omitted.

Fig. 2-23 Similarity between analysis scenarios of input file to Sagep Code (Direct Term Calculation).

3. Implementation of PSAGEP System

Based on design policies made in the previous chapter, phase 2 was implemented. In this chapter, PSAGEP system structure is shown followed by an explanation of class functions developed in phases 1 and 2.

3.1. Module Package

The PSAGEP system consists of several function units similar to the standard Python system, and managed as module package. Modules are managed as layers (See Fig. 3-1).

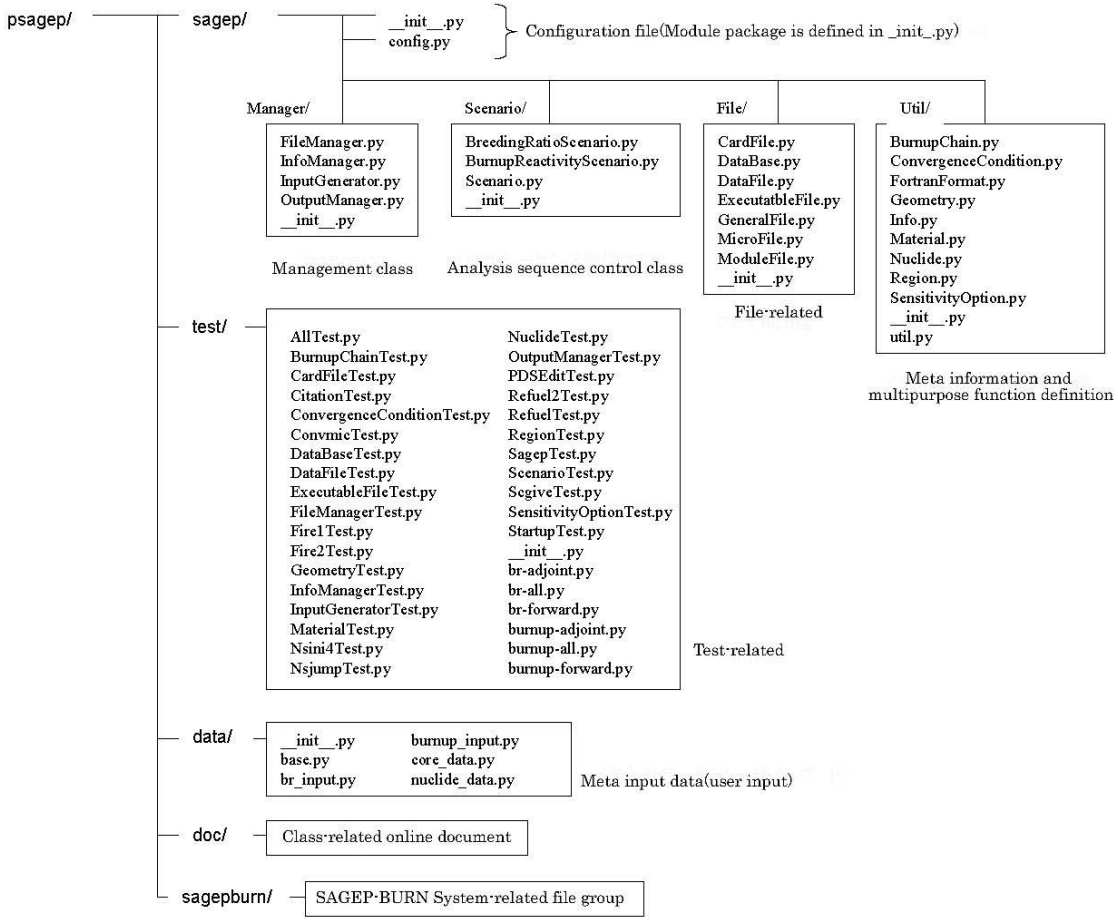


Fig. 3-1 Module package of PSAGEP system.

Module layers are consistent with the installed directory structure. Modules are stored in sub packages by the functions to be managed. (See Table 3-1).

Table 3-1 Module Package Name and Function

Package Name/Sub package		Function
Sagep	sagep.Manager	Management Class
	sagep.Scenario	Control class of analysis sequence
	sagep.File	File-related class
	sagep.Util	Meta information class and multipurpose function
Data		Definition of input meta information for analysis
Test		Test-related class

In order to use these packages, setting up Python is required. (Refer to Chapter 4 for details)

3.2. Content of Each Class

Each module package is composed of several class- defined files. Figure 3-2 shows the relationship between each class that composes PSAGEP. The gray highlighted portion represents areas developed during phase 2.

The following sections describe the content of each class.

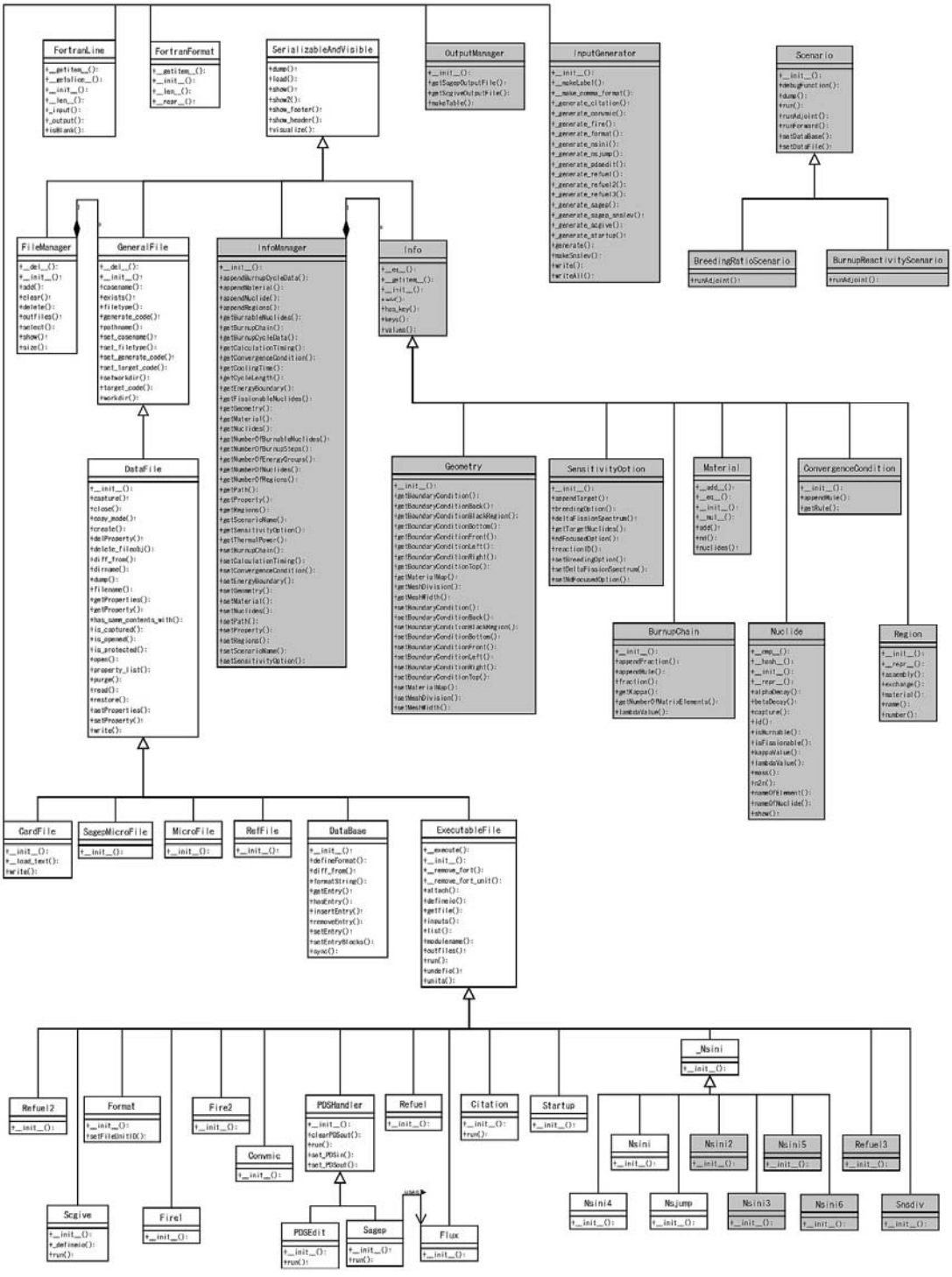


Fig. 3-2 Class group for composing PSAGEP system (class diagram).

3.2.1. Management Class

The management class controls the functional unit of the system. The list of manager class is shown in Table 3-2 and the content of each class is explained below:

- *FileManager* class: This class serves as a container for saving the file outputted by the *ExecutableFile* class. The file object is extracted with search condition of attribution (a generated code, a file type, case name) defined by the *GeneralFile* class.
- *InfoManager* class: This class can manage the Info object that is input metadata, and store various kinds of information.
- *InputGenerator* class: This class acquires meta information from the *InfoManager* class, and automatically generates the input file for the backend code to calculate.
- *OutputManager* class: This class extracts the sensitivity coefficient information from the output file (generated automatically by the backend code in the *FileManager* class), and arranges the result for increased accessibility and usage.

3.2.2. Analysis Sequence (Scenario) Control Class

The analysis sequence control class is also called the scenario class. It is prepared as a different class for each neutronic characteristic. Table 3-3 shows the list of the analysis scenario control class.

In this class, a series of analysis sequences that include backend codes are described. Therefore, users who want to create a new analysis sequence (scenario) are required to have a basic knowledge of backend codes. However, since wrapper definition information on each code and *InputGenerator* can automatically solve file connections of input files to each code and between codes, operation is easier compared to the former method. Contents of each class are shown as follows:

- *Scenario* class is the base class of each scenario. The forward calculation sequence independent of the analysis scenario and some common methods are defined.
- *BreedingRatioScenario* and *BurnupReactivitySenario* are classes for sensitivity analysis for breeding ratio and burnup reactivity loss, respectively. Here, the Adjoint calculation sequence is defined.

3.2.3. File-Related Class

As shown in Fig. 3-2, file-related classes have an inheritance relation. Accordingly the content of each class is shown. Table 3-4 shows the list of file-related classes.

- *GeneralFile* is the base class of the file-related classes, where basic methods are defined. It only manages pointer information to the "*File*" which exists on a system.
- *DataFile* has a function of *GeneralFile*, and converts files (i.e. input files for backend calculation codes and data files outputted from calculation codes) into objects. *DataFile* can read and write data file contents, compare file contents, and create different information for files.
- *CardFile* is a class which specializes in creating a fixed-formatted file. It is mainly used in the creation of an input file for the backend calculation code.
- *ExecutableFile* is the base class of the wrapper for each backend calculation code. It can set up dependency information for file connection with the Fortran I/O unit, solve dependency of files automatically, and perform calculation codes, and so on.
- Other file-related classes that are in *ModuleFile.py* define their input-output file information. The dependence of the file connection can be solved automatically based on this definition.

3.2.4. Meta Information and Others

Meta information class is a container object that maintains meta information specified by users. Table 3-5 shows meta information and lists of other classes. This table also contains classes needed for defining meta information. For showing hierarchical structure, the classes are introduced here (even though there is no common method). The content of each class is shown as follows:

- *Nuclide* class treats nuclides as abstraction data objects and manages JFS ID, physical properties value information and so on.
- *Geometry* class contains system information (mesh width, number of meshes) for the backend calculation code and information on boundary conditions and a material map.
- *Material* class contains number density information for each nuclide. Moreover, it also supports the calculation of *Material* object (the sum between *Materials and* the product with a scalar value), and the realization of material weighted

averaging.

- *Region* class contains information that connects the region name to material.
- *BurnupChain* class contains the information of burnup chain. This is easily set by using the parent nuclide, the daughter nuclide and the fraction (yield). From the specified set of chain rules, each burnup matrix for backend calculation codes is automatically generated.
- *ConvergenceCondition* class stores judgment conditions of the neutron Flux derived by backend calculation codes (SAGEP and CITATION).
- *SensitivityOption* class stores input information for analysis -- such as target reaction type that is specified to the backend calculation code (SAGEP and SCGIVE).

3.2.5. Test-Related Class

Test-related classes can be categorized into 2 types: unit test class and join test class. The unit test class is for performing target unit tests and testing method actions. The join test class is for investigating the combination of relation and action between classes. In fact, based on the actual analysis sequence, a series of analyses are carried out by operating the wrapper object of the backend calculation code. In this way, consistency between the results and reference calculations is confirmed.

Table 3-2 Function of Each Management-Related Class

Class name	Function
FileManager	Input-output management of data file
InfoManager	Management of Meta input infromaiton
InputGenerator	Generation of input file for calculation code from meta input information
OutputManager	Generation of sensitivity coefficient table from output file

*Gray part shows what implemented in phase 2 (this notation is the same in the following tables).

Table 3-3 Function of Each Analysis Sequence (Scenario) Control-Related Class

Class name	Function
Scenario	Base class of analysis scenario
BreedingRatioScenario	Analysis scenario class of breeding ratio
BurnupReactivitySenario	Analysis scenario class of burnup reactivity loss

*Class name indent shows inheritance relation.

Table 3-4 Function of Each File-Related Class

Class name	Function
GeneralFile	Base class of file operation
Data file	Deal with data file
CardFile	Deal with user card (fort.5)
ExecutableFile	Base class for load module wrapper
Convmic	Convmic module wrapper
Citation	Citation module wrapper
Fire1	Fire1 module wrapper
Fire2	Fire2 module wrapper
PDSHandler	Base class for module that deals with PDS
PDSEdit	PDSEdit module wrapper
Sagep	Sagep93 module wrapper
Flux	Flux module wrapper
_Nsini	Base class for Ns system module
Nsini	Nsini module wrapper
Nsini2	Nsini2 module wrapper
Nsini3	Nsini3 module wrapper
Nsini4	Nsini4 module wrapper
Nsini5	Nsini5 module wrapper
Nsini6	Nsini6 module wrapper
Nsjump	Nsjump module wrapper
Refuel	Refuel1 module wrapper
Refuel2	Refuel2 module wrapper
Refuel3	Refuel3 module wrapper
Snsdiv	Snsdiv module wrapper
Scgive	Scgive (scgiveber) module wrapper
Startup	Startup module wrapper

*Class name indent shows inheritance relation.

Table 3-5 Funtion of Each Class and Others

Class name	Function
Info	Base class of meta information
Geometry	Managemet class of geometry information
Nuclide	Managemet class of nuclide information
Material	Management class of material information
Region	Management class o f region information (refueling assemblies) of regions
BurnupChain	Management class of burnup chain
ConvergenceCondition	Management class of convergence condition
SensitivityOption	Management class of information of nuclide etc.to be analyzed
SerializableAndVisible	Display of object internal state
PSagep	Control class of the whole Psagep system
FortranFormat, FortranLine	Reading/writing of plan text by Fotran form format (function enhancement of ScientificPython)

*Class name indent shows inheritance relation. Gray part shows what implemented in phase 2 (this notation is the same in the following tables).

3.3. Investigation

As mentioned in section 2.1, the system development was carried out based on the idea of "Test first programming". That is, primarily, describing test codes that defined the action of an object class, and then implementing the PSAGEP system that executes the test. As Section 3.2.5 showed, the test code is prepared for all classes. *PyUnit*, which is a testing framework, was used for describing these test codes.

Test codes can be categorized into 2 types, i.e. unit test and join test. Unit test is designed for testing each method action in this class. Join test is intended for testing the relation between objects, such as file delivery.

In these tests, a verified set of input and output files for breeding ratio sensitivity calculation case was used for checking the operation.

3.3.1. Unit Test

Table 3-6 shows the list of the unit test. As seen in this table, the name of unit classes consist of class names to be examined, plus “Test”. For example, the unit test class for the *BurnupChain* class is *BurnupChainTest.py*. Testing wrapper classes of backend codes confirmed that input/output files and calculation lists are consistent with those of accepted standards.

3.3.2. Join test

Table 3-7 shows the list of the join test. The connection relations for the analysis scenario files were tested. These are tests about Forward calculation and Adjoint calculation for sensitivity analysis of breeding ratio and burnup reactivity loss. Moreover, ScenarioTest serves as an integration test that is used in all modules of the PSAGEP system. This system's validity was confirmed by maintaining consistency between the results from these tests and the reference calculation.

Table 3-6 Script File Name of Unit Test-Related and Class to Test

File name	Class to test
AllTest.py	Executing all the following tests
BurnupChainTest.py	BurnupChain
CardFileTest.py	CardFile
CitationTest.py	Citation
ConvergenceConditionTest.py	ConvergenceCondition
ConvmicTest.py	Convmic
DataBaseTest.py	DataBase
Data fileTest.py	Data file
ExecutableFileTest.py	ExecutableFile
Fire1Test.py	Fire1
Fire2Test.py	Fire2
GeometryTest.py	Geometry
InfoManagerTest.py	InfoManager
InputGeneratorTest.py	InputGenerator
MaterialTest.py	Material
Nsini4Test.py	Nsini4
NsjumpTest.py	Nsjump
NuclideTest.py	Nuclide
OutputManagerTest.py	OutputManager
PDSEditTest.py	PDSEdit
Refuel2Test.py	Refuel2
RefuelTest.py	Refuel
SagepTest.py	Sagep
ScgiveTest.py	Scgive
SensitivityOptionTest.py	SensitivityOption
StartupTest.py	Startup

Table 3-7 Join test-Related Script File Name and Test Contents

File name	Test contents
br-forward.py	Forward calculation of breeding ratio
br-adjoint.py	Adjoint calculation of breeding ratio
br-all.py	Forward and Adjoint calculation of breeding ratio
burnup-forward.py	Forward calculation of burnup reactivity loss
burnup-adjoint.py	Adjoint calculation of burnup reactivity loss
burnup-all.py	Adjoint and Forward calculation of burnup reactivity loss
ScenarioTest.py	general function test of analysis Scenario

4. Analysis Using PSAGEP System

This chapter explains PSAGEP analysis procedure and examples.

4.1. Preparation for Analysis

This section explains installation of PSAGEP. It is assumed that version 2.3 or later of a Python language is installed.

4.1.1. Installation Procedure

The following files are stored in CD-ROM.

- psagep.tgz

Psagep is installed in a directory of the user's choice. In this case, "/usr/local/" is used as an example.

```
# cd /usr/local/
# cp {CD-ROM}/psagep.tgz ./psagep.tgz
# tar zxvf psagep.tgz
```

"tar zxvf {filename}.tgz"

```
# gunzip { filename }.tgz
# tar xvf { filename }.tar
```

This may be substituted by "gunzip {filename}.tgz" or "tar xvf {filename}.tar". Next (if necessary), the PSAGEP system environment is set up using "/usr/local/psagep/sagep/config.py" file. When the directory to install differs from /usr/local, the value of source_dir is changed from /usr/local/psagep/.

Then, the SAGEP-BURN execution module is compiled.

```
# cd /usr/local/psagep/sagepburn/CODE/sagep-burn/src
# csh MAKE
# cd /usr/local/psagep/sagepburn/CODE
# csh mklall.sh
```

PSAGEP is installed completely when executable modules are successfully compiled.

4.1.2. Setting Up Environment

Next, the user environment is set up. The “cshrc” is opened using a suitable editor and it sets up PATH and PYTHONPATH.

- The following is added into cshrc:

  set path=(/usr/local/psagep $path)

  setenv PYTHONPATH “/usr/local/psagep:/usr/local/lib/python2.3”

- User environment is set up by executing the following commands:

```
# source .cshrc  (Reflection of change)
# which psagep.py  (Confirmation of path)
/usr/local/psagep/psagep.py ← OK if this line is displayed
>> echo $PYTHONPATH  (Confirmation of Python path)
/usr/local/psagep:/usr/local/lib/python2.3
```

4.2. Overview

The environment was established in the previous section. This section explains the procedure for performing a series of analyses using the PSAGEP system.

4.2.1. Scenario

Target parameters and the number of cycles to analyze require corresponding analysis sequences. Conversely, analysis conditions basically do not require modification of analysis sequences. The analysis sequence that was generalized is called the scenario.

4.2.2. Output File

When calculation is completed, the file SNS_{an analysis scenario name} is generated as the final result of PSAGEP.

After the **Forward** calculation is completed, a file called {scenario name}-**Forward**.dat is created. This file stores all restart files, including a cross section and a neutron flux created by each calculation in Forward calculation. After **Adjoint** calculation is completed, a file called {scenario name}-**Adjoint**.dat is generated. A database file is written out in detail by the name DataBase-Forward-{analysis scenario name}. (Note: “Forward” is replaced with “Adjoint” for the Adjoint calculation).

Analysis scenarios are identical between different Forward calculations therefore different analysis scenarios can use the same restart files and database files.

4.2.3. Intermediate (Temporary) File

The directory psagep-{user name}-{process ID} is created by executing the psagep under the tmp_dir specified by config.py. This directory executes all of the calculations. With the exception of the restart file and the final result (SNS file), all of the intermediate products and the necessary input/ output to each SAGEP-BURN calculation and so on are generated in this work directory.

4.2.4. Creating Reference Result for Test

The test code compares the input file for SGAEP-BURN and the input file created by PSAGEP. The test code also compares results from both systems.

Therefore, it is necessary to have an analysis result of the existing SAGEP-BURN system that is used as a reference while executing the test code. The existing system is performed as follows. An example of the breeding ratio test is shown below.

```
# cd /usr/local/psagep/sagepburn/br/Burn
# sh Burn-normal.sh
# sh Burn-adj.sh
```

After the calculation is completed correctly, a file called {input file name}.dmp should be generated to each calculation code. Next, the calculation result is processed.

```
# cd ../table
# sh mk_shs.sh
```

After the above calculation is completed, a file called SNS is generated. This file is equivalent to the final product of PSAGEP.

4.3. Creating User Input

The PSAGEP input is roughly divided into three parts:

1. Data that relates to nuclides such as nuclide names, burnup chains and energy group structure. (nuclide_data.py)
2. Data that relates to core structure such as core geometry, and material included in the area. (core_data.py)
3. Data, such as burnup step, convergence conditions, and sensitivity calculation options, which are changed frequently by user. (br_input.py etc.)

This data is compiled in /usr/local/psagep/work/psagep/data. Refer to appendix 1 for details of the input file.

The following section explains Python grammar to create the user input file and each parameter in the sample input file.

4.3.1. Explaining Python Grammar

All the user inputs of PSAGEP are described in Python format. This section explains Python grammar necessary to describe the input.

(1) **Substituting Variables**

Since Python is a scripting language, a variable can be used without a declaration statement.

That is,

Variable = formula

(2) **List**

Generally a list is called an array. Python lists/arrays are described in writing as that which consists of the comma-delimited values surrounded by parenthesis [ ]. All the list elements need not be of the same type. Arguments in the list start from zero.

```
% python
Python 2.3.2 (#1, Dec 4 2003, 11:29:08)
[GCC 2.96 20000731 (Red Hat Linux 7.3 2.96-113)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> A = [ 'a', 'b', 10, 100 ]
>>> print A
[ 'a', 'b', 10, 100 ]
>>> print A[1], A[2]
'b', 10
>>> print A[0]
'a'
```

(3) **Dictionary**

The dictionary is called an associative array. Since the dictionary consists of the pair of key and value (Key: value) without sequence, the key (in the dictionary) should

be unique.

```
>>> B = {'a' : 1, 'b' : 2 }
>>> print B
{'a' : 1, 'b' : 2 }
>>> print B['a']
1
```

(4) **String**

Items in quotes are treated as strings.

(5) **Comment**

In Python, the contents from # to the end of the line (line feed code) are comments.

(6) **Method**

The method is equivalent to the function of the C language. In the user input of PSAGEP, data is added or overwritten into the input dataset. In accepted rules governing method names, the method that starts with "append" will add data and the method name the starts with "set" will overwrite data.

4.3.2. nuclide_data.py

This file contains nuclide-related data. Corresponding data is shown as follows:

- Energy group structure
- Nuclide name used in SAGEP-BURN
- Burnup chain
- Reaction type of the nuclide to derive sensitivity coefficient

Input form is explained in the order from the energy group structure.

(1) **Energy Group Structure**

Energy group structure is inputted in list format. The following table shows the 18-energy group structure. The number of data is 19 (= 18 +1) because lower and upper data are required for each group of 18 groups.

```
### Energy group Info.
jfs18g = [ 1.00000000e+07, 6.06530700e+06, 3.67879400e+06, 2.23130200e+06,
1.35335300e+06, 8.20850000e+05, 3.87742094e+05, 1.83156406e+05,
8.65169531e+04, 4.08677109e+04, 1.93045391e+04, 9.11882031e+03,
4.30742480e+03, 2.03468396e+03, 9.61116516e+02, 4.53999298e+02,
2.14454102e+02, 1.01300903e+02, 9.99999975e-06]
info_man.setEnergyBoundary( jfs18g )
```

Fig. 4-1 Input example of 18-energy group structure.

(2) Nuclide Name Used in SAGEP-BURN

Nuclide name is inputted in list format. A hyphen is required in the description of the nuclide name. For example: u-235, U-235, AM-242M, u-235fp, and PU-235fp. Refer to Nuclide class (psagep/sagep/Util/Nuclide.py) for the available nuclide names.

Since the order of nuclides must be consistent with that in the micro cross-section file generated with JOINT, it is necessary to match the nuclide order in the micro cross section file and in *InfoManager*. The number of nuclides must be the same in both the micro cross-section file and *InfoManager*.

```
### Nuclide Set
# The order of nuclides must be consistent with that in the micro XS file generated with JOINT
nuclide_names = [ 'U-235', 'U-236', 'Np-237', 'U-238', 'Np-239', 'Pu-238', 'Pu-239',
'Pu-240', 'Pu-241', 'Pu-242', 'Am-241', 'Am-242m', 'Am-243', 'Cm-242',
'Cm-243', 'Cm-244', 'Cm-245', 'U-235FP', 'U-238FP', 'Pu-239FP', 'Pu-241FP',
'O-16', 'Na-23', 'Cr-nat.', 'Mn-55', 'Fe-nat.', 'Ni-nat.', 'Mo-nat.',
'Nd-143', 'W-nat.', 'B-10', 'B-11', 'C-12' ]
nuclide_set = [ Nuclide(name=x) for x in nuclide_names ]


info_man.setNuclides( nuclide_set )


#print nuclide_set
```

Fig. 4-2 Input example of nuclide name.

(3) **Burnup Chain**

The burnup chain is defined by using an append Rule method. The input form is shown as follows:

- appendRule (*parent nuclide name, the reaction type 1, the reaction type 2...*).

That is, the first argument is the parent nuclide name, and second and later are reaction types. Input forms of reaction types are string (e.g. 'U-235') and list (e.g. ['Fission', 'U-235FP']. Figure 4-3 explains how to designate the burnup chain with an example.

The reaction type is basically entered by list form such as **reaction type, daughter nuclide, and yield**. The input of the reaction rate can be omitted if the reaction rate equals 1. When the daughter nuclide is uniquely determined with the reaction type (for example, when U-236 is generated at the capture reaction of U-235), the input of the daughter nuclide can be omitted. When the daughter nuclide is omitted, the input is specified by string, and not by list. As noted, if the daughter nuclide is not registered in PSAGEP (when not registered in the previous part of the input explained in the section 4.3.2 (2)), an error is raised.

```
### Burnup Chain
bc = BurnupChain()
#          Fertile Name, Reaction Type, [ Reaction Type, Daughter], [ Reaction Type, Daughter, Fraction ]
#          If Reactin Type is Fission, you need to input Daughter.
bc.appendRule( 'U-235', 'Capture', ['Fission', 'U-235FP'] )
bc.appendRule( 'U-236', ['Capture', 'Np-237'], ['Fission', 'U-235FP'], 'N2N' )
bc.appendRule( 'Np-237', ['Capture', 'Pu-238'], ['Fission', 'U-238FP'] )
bc.appendRule( 'U-238', ['Capture', 'Np-239'], ['Fission', 'U-238FP'], ['N2N', 'Np-237'] )
bc.appendRule( 'Np-239', ['Fission', 'Pu-239FP'], 'BetaDecay' )
bc.appendRule( 'Pu-238', 'Capture', ['Fission', 'U-238FP'], ['N2N', 'Np-237'] )
bc.appendRule( 'Pu-239', 'Capture', ['Fission', 'Pu-239FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Pu-240', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Pu-241', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'BetaDecay' )
bc.appendRule( 'Pu-242', ['Capture', 'Am-243'], ['Fission', 'Pu-241FP'], 'N2N' )
bc.appendRule( 'Am-241', ['Capture', 'Pu-242', 0.1384], ['Capture', 'Am-242m', 0.2000], ¥
             ['Capture', 'Cm-242', 0.6616], ['Fission', 'Pu-241FP'], ['N2N', 'Pu-240'], 'AlphaDecay' )
bc.appendRule( 'Am-242m', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', ['Decay', 'Pu-242', 0.173],
             ['Decay', 'Cm-242', 0.827] )
bc.appendRule( 'Am-243', ['Capture', 'Cm-244'], ['Fission', 'Pu-241FP'] )
bc.appendRule( 'Cm-242', 'Capture', ['Fission', 'Pu-241FP'], ['N2N', 'Am-241'], 'AlphaDecay' )

info_man.setBurnupChain(bc)
```

Fig. 4-3 Input example of burnup chain.

(4) Specifying Reaction Type of Nuclide Concerning Sensitivity Calculation

When generating a sensitivity calculation of each nuclide, the amount of fluctuation of the fission spectrum and the calculating reaction type are specified.

Initially, the amount of change of the fission spectrum for calculating the sensitivity coefficient is entered. Later, the nuclide name and reaction type for calculating the sensitivity coefficient are inputted. The input form is to enter the nuclide name in the 1st argument and the reaction type in 2nd and later arguments.

Users can use this form without alteration.

```
### Sensitivity Calculation Option
so = SensitivityOption()

delta_xi = [-1.00, -0.90, -0.80, -0.70, -0.60, -0.50, -0.40, -0.30, -0.20, -0.10, -0.05,
            0.05, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00]
so.setDeltaFissionSpectrum( delta_xi )

# Capture, Nu, Transport, Fission, TotalScattering, ElasticScattering, InelasticScattering
# Mu, N2N, InelasticCrossSection N
so.appendTarget( 'U-235', 'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering',
                 'Mu', 'N2N' )
so.appendTarget( 'U-236', 'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering',
                 'Mu', 'N2N' )
so.appendTarget( 'U-237', 'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering',
                 'Mu', 'N2N' )
so.appendTarget( 'U-238', 'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering',
                 'Mu', 'N2N' )

info_man.setSensitivityOption(so)
```

Fig. 4-4 Input example about fission spectrum fluctuation of sensitivity calculation and reaction type of each nuclide.

4.3.3. core_data.py

This file contains core data that are shown as follows:

- Material information
- Region information
- Core geometry
- Core structure (Material configuration)
- Boundary condition

Input form is hereafter explained in order from material information.

(1) **Material Information**

This defines material data to be assigned on each region of the evaluating system. Material data defines sets of nuclide and number density that compose the material in dictionary form.

{ *nuclide name: number density …* (as necessary number) }

Create material information for the number of materials that exist in the evaluation.

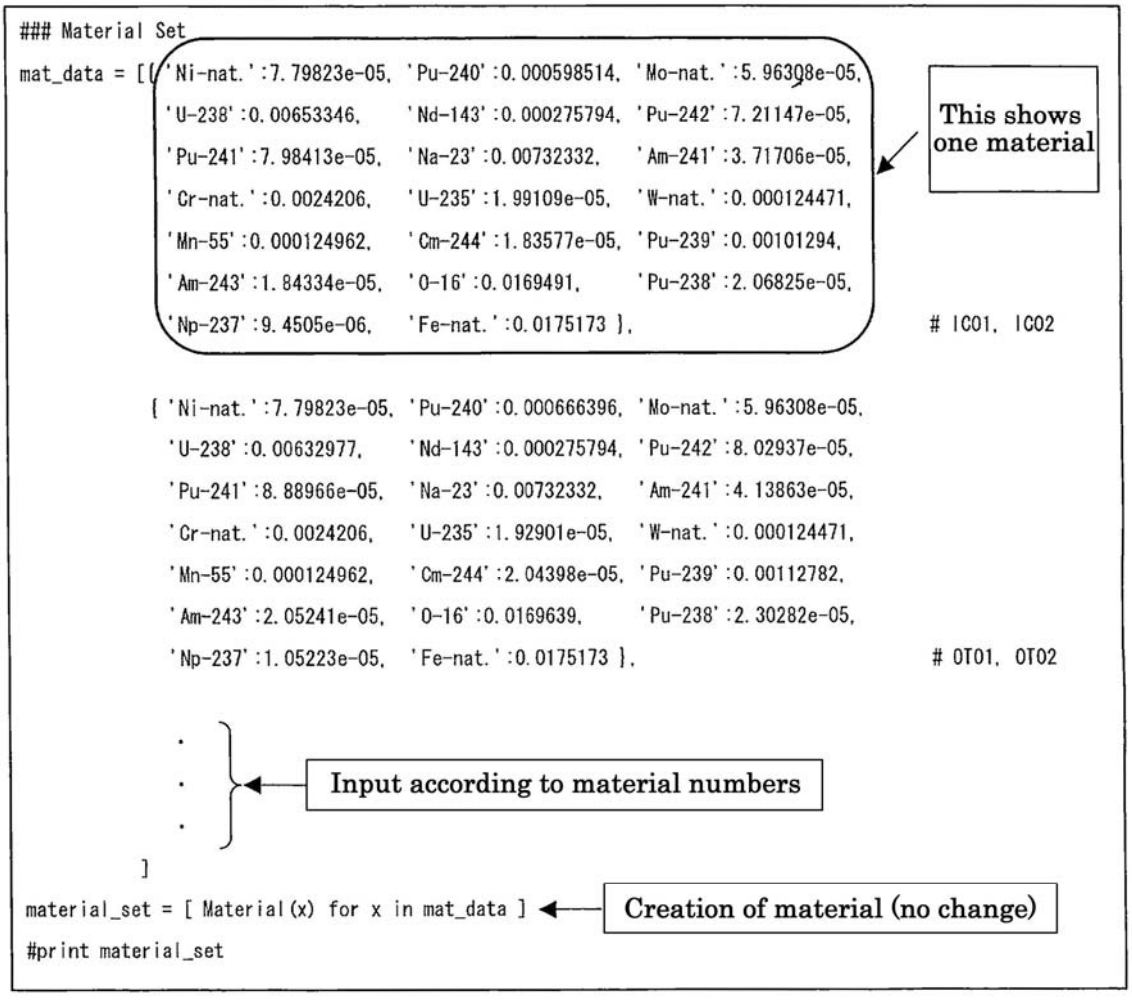


Fig. 4-5 Input example of material data set.

(2) Region Information

Here, the region information in the calculation system is created. The followings are mentioned as data required for region information:

1. Region ID
2. Region name
3. Material
4. Number of assemblies contained in each subzone
5. Refueling option（YES/NO：1.0/0.0）

Material data that was created before is entered as region information: Material.

```
### Region Set
 region_data = [ [ 1, "IC01", material_set[0], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 2, "IC02", material_set[0], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 3, "OT01", material_set[1], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 4, "OT02", material_set[1], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 5, "ABLU", material_set[2], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 6, "ABLL", material_set[2], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 7, "RBLU", material_set[3], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 8, "RBLM", material_set[3], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 9, "RBLL", material_set[3], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                 [ 10, "NAFO", material_set[4], [ 1.0 ], 0.0],
                 [ 11, "GPL1", material_set[5], [ 1.0 ], 0.0],
                 [ 12, "GPL2", material_set[6], [ 1.0 ], 0.0],
                 [ 13, "AXSU", material_set[7], [ 1.0 ], 0.0],
                 [ 14, "AXSL", material_set[8], [ 1.0 ], 0.0],
                 [ 15, "RDS1", material_set[9], [ 1.0 ], 0.0],
                 [ 16, "RDS2", material_set[10], [ 1.0 ], 0.0]
                 ]
 region_set = [ Region( no=x[0], name=x[1], material=x[2], assembly=x[3], exchange=x[4] ) ¥
 for x in region_data ]
 info_man.setRegions(region_set)
 #print region_set
```

Fig. 4-6 Input example of region information.

Material information and region information do not always need to follow the above-mentioned setup. PSAGEP input is in Python script, and very user-friendly. For instance, it is possible to create material objects and region objects expressly as follows, and it is possible to carry out additional registration in InfoManager.

```
### Material Set
mat1 = Material( { 'Ni-nat.':7.79823e-05, 'Pu-240':0.000598514, 'Mo-nat.':5.96308e-05,
'U-238':0.00653346, 'Nd-143':0.000275794, 'Pu-242':7.21147e-05,
'Pu-241':7.98413e-05, 'Na-23':0.00732332, 'Am-241':3.71706e-05,
'Cr-nat.':0.0024206, 'U-235':1.99109e-05, 'W-nat.':0.000124471,
'Mn-55':0.000124962, 'Cm-244':1.83577e-05, 'Pu-239':0.00101294,
'Am-243':1.84334e-05, 'O-16':0.0169491, 'Pu-238':2.06825e-05,
'Np-237':9.4505e-06, 'Fe-nat.':0.0175173 } )
reg1 = Region( no=1, name=" IC01" , material = mat1, assembly = [ 1.0, 1.0, 1.0, 1.0 ], exchange = 1)
info_man.appendRegions(reg1)


# The following is same.
```

Fig. 4-7 Another example on specification of material information and region information.

(3) Core Geometry

Here, the region number in mesh and the mesh width that are used in CITATION and SAGEP, etc. are inputted. In the first half, the region number in each mesh is entered in two-dimensional list form. (Refer to the manual of CITATION about the relation between mesh and area.)

```
### Geometry
geom = Geometry()

### Number of division of each mesh

div = [
      [ 2, 3, 3, 3, 2, 4, 4, 3, 2, 5, 4, 4, 2, 3, 2, 2, 4, 4, 4, 5, 3 ], # X AXIS
      [ 5, 6, 4, 3, 3, 3, 4, 3, 3, 3, 4, 16, 5],                         # Y AXIS
      [ 1 ]                                                              # Z AXIS
      ]

geom.setMeshDivision( div )
```

Fig. 4-8 Input example of region number in mesh.

Next, the width of each mesh is inputted in two-dimensional list form.

```
### Width of each mesh
width = [
        [ 9.55043, 15.71764, 16.36130, 16.46364, 5.97324, 18.64292, 17.91088, 12.78505,
          4.72722, 21.25165, 17.83599, 18.10268, 4.62130, 13.20161, 10.12392, 3.55860,
         18.97454, 17.49924, 17.37424, 32.60152, 17.22164 ],
       [ 30.0000, 30.0000, 20.0000, 15.0000, 15.0000, 15.0000, 20.0000, 15.0000,
         15.0000, 15.0000, 20.0000, 100.000, 30.0000 ],
       [ 1.0000 ]
       ]
geom.setMeshWidth( width )
```

Fig. 4-9 Input example of mesh width.

(4) **Core Structure (Material Configuration)**

Here, the material ID which is assigned to each mesh is inputted in two-dimensional list form. The 1st argument of Region Set corresponds to this ID.

```
### material number of each mesh
material_map = [[ 10,13,13,13,10,13,13,13,10,13,13,13,10,13,13,10,13,13,13,15,16 ],
                [ 10,11,11,11,10,11,11,11,10,11,11,11,10,11,11,10,11,11,12,15,16 ],
                [ 10, 5, 5, 5,10, 5, 5, 5,10, 5, 5, 5,10, 5, 5,10, 5, 5, 7,15,16 ],
                [ 10, 5, 5, 5,10, 5, 5, 5,10, 5, 5, 5,10, 5, 5,10, 5, 5, 7,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 6, 6, 6,10, 6, 6, 6,10, 6, 6, 6,10, 6, 6,10, 6, 6, 9,15,16 ],
                [ 10, 6, 6, 6,10, 6, 6, 6,10, 6, 6, 6,10, 6, 6,10, 6, 6, 9,15,16 ],
                [ 10,11,11,11,10,11,11,11,10,11,11,11,10,11,11,10,11,11,12,15,16 ],
                [ 14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,16 ]
                ]

 geom.setMaterialMap( material_map )
```

Fig. 4-10 Input example about material in core.

(5) **Boundary Condition**

Here, boundary conditions of an external boundary that are used in CITATION and SAGEP, etc. are inputted. "REFLECTIVE", "VACUUM", and "DUMMY" are prepared as global variables in the input of boundary conditions. Users should input using these variables. (It is possible to input numeric data directly, but using these variables is recommended for ease of comprehensibility.)

There are two kinds of input methods. One is a method (Refer to the Fig. 4-11, 4th and 5th line) to input by list form. Another one is (from 6th line in the figure below) to input for each direction using methods such as setBoundaryConditionLeft.

```
# XYZ: Left:X- Top:Y- Right:X+ Bottom:Y+ Front:Z- Back:Z+ Black_Region
# RZ: Left:X- Top:Y- Right:X+ Bottom:Y+ Dummy(0.0) Dummy(0.0) Black_Region
# dir: BC_LEFT BC_TOP BC_RIGHT BC_BOTTOM BC_FRONT BC_BACK BC_BLACK
#boundary = [ REFLECTIVE, VACUUM, VACUUM, VACUUM, REFLECTIVE, REFLECTIVE, DUMMY ]
#geom.setBoundaryCondition( boundary )

geom.setBoundaryConditionLeft( REFLECTIVE )
geom.setBoundaryConditionTop( VACUUM )
geom.setBoundaryConditionRight( VACUUM )
geom.setBoundaryConditionBottom( VACUUM )
geom.setBoundaryConditionFront( REFLECTIVE )
geom.setBoundaryConditionBack( REFLECTIVE )
geom.setBoundaryConditionBlackRegion( DUMMY )
info_man.setGeometry( geom )
```

Fig. 4-11 Input example of boundary conditions.

4.3.4. User Input

As an argument of PSAGEP execution, inputs specified by users are a file type, such as br_input.py and burnup_input.py. This will differ for users' analysis in the respective analysis scenario.

The following information is included in the above-mentioned files:

- Burnup step information
- Convergence condition
- Scenario name
- Path information regarding cross-section file and reference for test
- Sensitivity calculation option

Following, the input form is explained in sequence beginning with burnup step information.

(1) Burnup Step Information

Thermal power, cooling period, and full power day are inputted for each burnup step. The number and order of steps entered here will be calculated.

```
### Burnup Cycles
#                    Cooling Operation Sub-Step
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 0*DAYS, operating = 548*DAYS,
                                    substep = 20 ) # CYC-01
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS,
                                    substep = 20 )  # CYC-02
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS,
                                    substep = 20 )  # CYC-03
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS,
                                    substep = 20 )  # CYC-04
```

Fig. 4-12 Input example of burnup step.

(2) Convergence Conditions

Convergence conditions used by the calculation of SAGEP-BURN system (CITATION, SAGEP) are inputted here. When changing input values, change “value”, but do not change “code” and “type” (See Fig. 4-13).

```
### Convergence Condition
cc = ConvergenceCondition()
cc.appendRule( code = 'Citation', type = 'MaxOuterIterations', value = 300 )
cc.appendRule( code = 'Citation', type = 'InnerCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Citation', type = 'OuterCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Sagep', type = 'MaxInnerIterations', value = 1000 )
cc.appendRule( code = 'Sagep', type = 'MaxOuterIterations', value = 200 )
cc.appendRule( code = 'Sagep', type = 'MaxCalculationTime', value = 2000 )
cc.appendRule( code = 'Sagep', type = 'InnerCriterion', value = 1.0E-04 )
cc.appendRule( code = 'Sagep', type = 'CutOffCondition', value = 5.0E-02 )


info_man.setConvergenceCondition(cc)
```

Fig. 4-13 Input example of convergence condition.

(3) Scenario Name

When specifying an analysis scenario name, the scenario name includes the following:

- BreedingRatio (Breeding ratio)
- BurnupRecactivity (Burnup reactivity loss)
- K-Effective (Effective multiplication factor: unsupported)
- NumberDensity (Number density: unsupported)
- ReactionRate (Reaction rate: unsupported)
- NaVoidReactivity (Sodium void reactivity value: unsupported)

Calculation timing (BOC or EOC) should be specified when calculating the sensitivity coefficients of the effective multiplication factor, reaction rate, and reactivity worth. Refer to comment of Fig. 4-14 for this input case.

```
### Scenario Name
scenario_name = 'BreedingRatio'
#calculation_timing =  'BOC'
info_man.setScenarioName(scenario_name)
#info_man.setCalculationTiming(calculation_timing)
```

Fig. 4-14 Input example of analysis scenario.

(4) Path Information for Cross-Section File and Reference for Test

Paths to the following data are specified here.

- cross_section_file_path: path which saves effective micro cross section
- data_base_path: Path which saves initial database of calculation
- reference_path:　Path which saves reference for test

```
### Path Infomation
cross_section_file_path = '/project/psagep/sagepburn/br/WORK/'
data_base_path = '/project/psagep/work/hyd/psagep/data/'
reference_path = '/project/psagep/sagepburn/br/Burn/'

info_man.setPath( target = 'cross_section', path = cross_section_file_path )
info_man.setPath( target = 'database', path = data_base_path )
info_man.setPath( target = 'reference', path = reference_path )
```

Fig. 4-15 Input example of path specification.

(5) Sensitivity Calculation Option

Necessary inputs are carried out in cases doing sensitivity calculations. Specification methods of fertile nuclide, fissile nuclide and other nuclides, which are needed in calculating breeding ratio, are shown in Fig. 4-16 as an example. Currently, only breeding ratio calculation is supported. (Specific input to sensitivity calculation is not needed when calculating burnup reactivity loss.)

```
### Calculation Breeding Ratio Option
so = info_man.getSensitivityOption()
fertile = ['U-238', 'Pu-240', 'Pu-238']
daughter = ['Pu-239', 'Pu-241']
other = ['U-235', 'U-236', 'Pu-242', 'Am-241', 'Am-242m', 'Am-243', 'Np-237', 'Np-239', 'Cm-242']
so.setBreedingOption(fertile, daughter, other)

info_man.setSensitivityOption(so)
```

Fig. 4-16 Input example of sensitivity analysis option.

4.4. Executing Analysis

The explanation of input format is in the previous section. This section explains the execution of the analysis procedure.

4.4.1. Execution Method of Analysis

First, create the directory for execution. It is assumed that a directory called psagep-works is created under the home directory.

```
# mkdir ~/psagep-works
# cd ~/psagep-works
```

Create the input file by copying the sample input under psagep/data directory and modifying it.

```
# cp /usr/local/psagep/work/psagep/data/br-input.py ./br-input.py
```

Try the execution of the psagep command. A brief explanation of the command is displayed when executing without arguments.

```
# psagep.py
usage: psagep.py [options] <input>
options:
    --debug, -d debug mode
    --all, -A All calculation is executed(default)
    --forward, -f Only Forward Calculation is executed
    --adjoint, -a Only Forward Calculation is executed

    --edit, -e Edit mode.(edit only)

    input: user input file name (eg. br_input,burnup_input)
```

Explanations of options are shown below.

Table 4-1 Explanation of Psagep Option

Option	Explanation
--debug -d	Debug option. Compares calculation results between reference directories specified by input and each step of PSAGEP, and confirms function.
--all -A	Executes all calculations of Forward and Adjoint
--forward -f	Only Forward calculation is executed.
--adjoint –a	Only Adjoint calculation is executed. Requires restart file and DataBase file that were created by Forward calculation in case of calculation. Uses these files to redo only Adjoint calculation by changing conditions.
--edit -e	Only edit of calculation result is performed. Requires restart file created by Adjoint calculation. Other options are disregarded and calculations are not performed if this option is chosen.

Try the execution of the sample input as it is to confirm whether it works in the debug mode of PSAGEP system.

After the calculation is completed normally, the calculation result is created in the directory (directory with user input) that performed the calculation.

```
# psagep.py -d -A br_input
# ls
Adjoint_BreedingRatio.dat   DataBase        DataBase-Adjoint-BreedingRatio
DataBase-Forward-BreedingRatio                  Forward_BreedingRatio.dat
SNS_BreedingRatio    br_input.py    br_input.pyc
```

Confirmation that PSAGEP works correctly is expressed as complete agreement between SNS_BreedingRatio and the reference directory.

4.5. Assessment of Analysis Result

The SNS{analysis case} file, which is the result of analysis of PSAGEP, can be created in the execution directory after analysis is completed. This file includes nuclide, reaction type, and sensitivity coefficients for each energy group. A part of the SNS file is shown in Fig. 4-17 as an example.

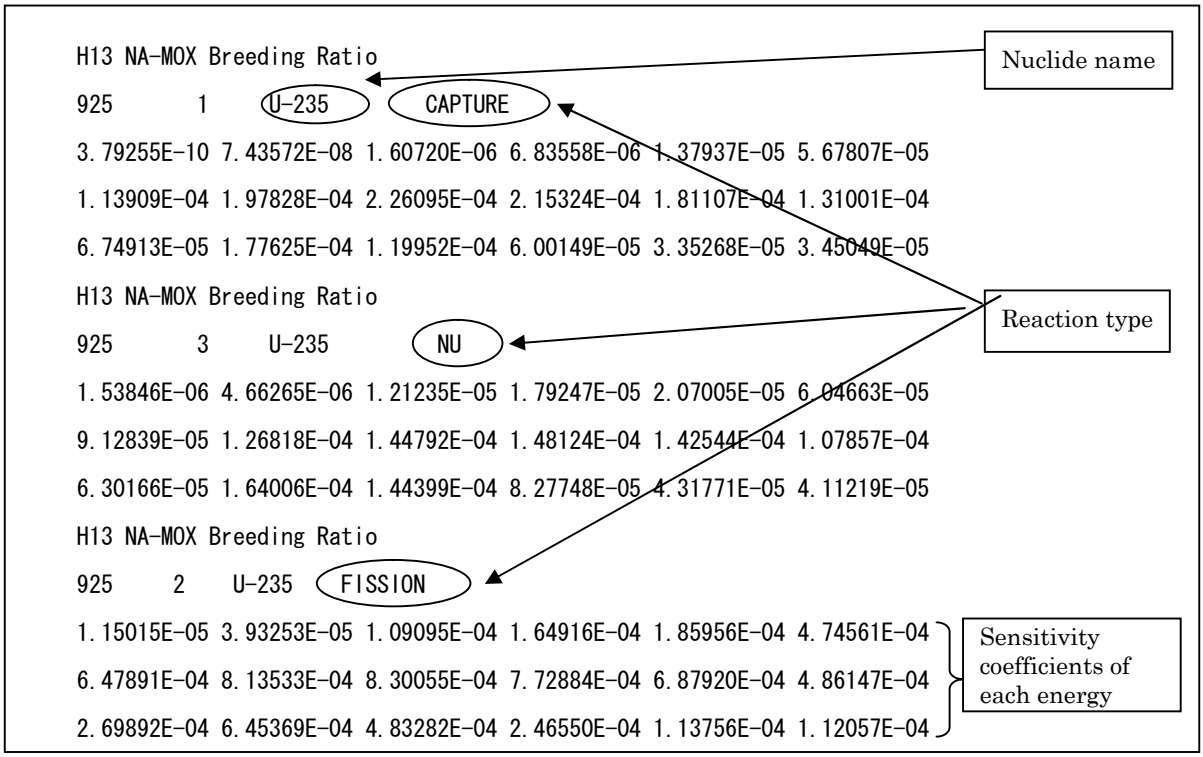


Fig. 4-17 Example of sensitivity calculation results.

5. Concluding Remarks

Burnup sensitivity analysis is important for realizing fast reactors; and the effectiveness of SAGEP-BURN performing burnup sensitivity analysis has been well established. Unfortunately, the analysis sequence becomes inefficient because of the relative complexity of the theory of burnup sensitivity, as well as inherent limitations of the system. The current burnup sensitivity analysis system required updating for the following reason: the computational sequence needed to be changed when validating the consistency of the resultant parameters. For this purpose, burnup sensitivity analysis code was systemized using the technology of object-orientation and scripting languages.

In phase 1 of the development (carried out in fiscal year of 2003), existing problems of the SAGEP-BURN system were identified through process analysis, and alternative options were investigated to determine the most desirable solution. Based on the investigation results, a two-layer controlling model consisting of a control layer and an encapsulation layer of the existing system were designed, and the encapsulation layer was implemented; this layer uses the object-oriented scripting language, Python. In this implementation, all functions of the existing SAGEP-BURN migrated to control by the Python language layer which resulted in widely improved flexibility of the input / output operation. In addition, without changing the system module and calculation logic, consistent analysis results were obtained and verified with accepted standards of existing analysis.

In phase 2 of the development (carried out in fiscal year of 2004), detail design and implementation of the control layer were done and PSAGEP (Python -wrapped SAGEP-burn) was completed as a system. In PSAGEP, it is possible for users to analyze burnup sensitivity without being aware of the backend calculation code. In this development, two analysis scenarios, i.e. breeding ratio and burnup reaction loss, are implemented, and implementing a new scenario has become more convenient.

In addition, (with the exception of classes that are strongly dependent scenario classes, such as PSAGEP,) Python class libraries developed here are independent from PSAGEP. Those class libraries can be utilized as reconstruction platforms of existing code. Through the development process, it was confirmed that the method of system-development based on encapsulation was efficient and effective as a method to shift the exiting analysis code to a new system. Internal contents of existing code remain when the existing analysis code is encapsulated using this method. Also, the internal contents can be gradually modified as far as the test program is well established and fixed.

References

1. H.Hanaki, S.Sawada and T.Sanda: “Preparation of Computer Codes for Analyzing Sensitivity Coefficients of Burnup Characteristics”, PNC TJ9124 93-009(1993). [in Japanese]
2. H.Hanaki, T.Sanda and M.Ohashi: “Preparation of Computer Codes for Analyzing Sensitivity Coefficients of Burnup Characteristics(II)”, PNC TJ9124 94-007(1994). [in Japanese]
3. M.Tatsumi and H.Hyoudou: “Systemization of Burnup Sensitivity Analysis Code”, JNC TJ9400　2003-012(2004). [in Japanese]
4. http://www.python.org
5. K. Yokoyama and H.Hosogai, et al.: “R & D of the Object-Integrated Code System for Fast Reactors – Needs and Fundamental System Concept -,” Atomic Energy Society of Japan, 2003 Fall Meeting, E64 (2003). [in Japanese]
6. K. Yokoyama and H.Hosogai, et al.:”Development of the Next Generation Code System, As an Engineering Modeling Language (I)”,　JNC TN9420 2002-004(2002). [in Japanese]
7. K. Yokoyama and H.Hosogai, et al.: “Development of the Next Generation Code System, As an Engineering Modeling Language (II) – Study with Prototyping”, JNC TN9400 2003-021(2003). [in Japanese]
8. Japan XP User Group, “eXtereme Programming, Test giho, xUnit de hajimeru jissen XP programming”, Shoeisha (2001). [in Japanese]
9. T. Takemasa:”Hajimete manabu UML”, Natsumesha (2003). [in Japanese]
10. ScientificPython, http://starship.python.net/~hinsen/ScientificPython/

A.1 Input Examples of PSAGEP System

This is a blank page.

```
from sagep.config import *
from sagep.Util.Nuclide import *
from sagep.Util.Material import *
from sagep.Util.Region import *
from sagep.Util.Geometry import *
from sagep.Util.SensitivityOption import *
from sagep.Util.ConvergenceCondition import *
from sagep.Manager.InfoManager import *

### InfoManager
info_man = InfoManager()
```

Fig. A.1-1 base.py.

```
from data.core_data import *

### Burnup Cycles
# Thermal-Power Cooling Operation Sub-Step
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 0*DAYS, operating = 548*DAYS, substep = 20 ) # CYC-01
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 ) # CYC-02
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 ) # CYC-03
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 ) # CYC-04

### Convergence Condition
cc = ConvergenceCondition()
cc.appendRule( code = 'Citation', type = 'MaxOuterIterations', value = 300 )
cc.appendRule( code = 'Citation', type = 'InnerCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Citation', type = 'OuterCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Sagep', type = 'MaxInnerIterations', value = 1000 )
cc.appendRule( code = 'Sagep', type = 'MaxOuterIterations', value = 200 )
cc.appendRule( code = 'Sagep', type = 'MaxCalculationTime', value = 2000 )
cc.appendRule( code = 'Sagep', type = 'InnerCriterion', value = 1.0E-04 )
cc.appendRule( code = 'Sagep', type = 'CutOffCondition', value = 5.0E-02 )

info_man.setConvergenceCondition(cc)

### Scenario Name
scenario_name = 'BreedingRatio'
info_man.setScenarioName(scenario_name)

### Path Infomation
cross_section_file_path = '/project/psagep/sagepburn/br/WORK/'
data_base_path = '/project/psagep/work/hyd/psagep/data/'
reference_path = '/project/psagep/sagepburn/br/Burn/'
info_man.setPath( target = 'cross_section', path = cross_section_file_path )
info_man.setPath( target = 'database', path = data_base_path )
info_man.setPath( target = 'reference', path = reference_path )

### Calculation Breeding Ratio Option
so = info_man.getSensitivityOption()

fertile = ['U-238', 'Pu-240', 'Pu-238']
daughter = ['Pu-239', 'Pu-241']
other = ['U-235', 'U-236', 'Pu-242', 'Am-241', 'Am-242m', 'Am-243', 'Np-237', 'Np-239', 'Cm-242']
so.setBreedingOption(fertile, daughter, other)

info_man.setSensitivityOption(so)

print "InfoManager is built for a default input set.¥n"
```

Fig. A.1-2 br-input.py.

```
from data.core_data import *

### Burnup Cycles
# Thermal-Power Cooling Operation Sub-Step
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 0*DAYS, operating = 548*DAYS, substep = 20 ) # CYC-01
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 ) # CYC-02
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 ) # CYC-03
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 ) # CYC-04

### Convergence Condition
cc = ConvergenceCondition()
cc.appendRule( code = 'Citation', type = 'MaxOuterIterations', value = 300 )
cc.appendRule( code = 'Citation', type = 'InnerCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Citation', type = 'OuterCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Sagep', type = 'MaxInnerIterations', value = 1000 )
cc.appendRule( code = 'Sagep', type = 'MaxOuterIterations', value = 200 )
cc.appendRule( code = 'Sagep', type = 'MaxCalculationTime', value = 2000 )
cc.appendRule( code = 'Sagep', type = 'InnerCriterion', value = 1.0E-04 )
cc.appendRule( code = 'Sagep', type = 'CutOffCondition', value = 5.0E-02 )

info_man.setConvergenceCondition(cc)

### Scenario Name
scenario_name = 'BurnupReactivity'

info_man.setScenarioName(scenario_name)

### Path Infomation
cross_section_file_path = '/project/psagep/sagepburn/burn-up/WORK/'
data_base_path = '/project/psagep/work/hyd/psagep/data/'
reference_path = '/project/psagep/sagepburn/burn-up/Burn/'

info_man.setPath( target = 'cross_section', path = cross_section_file_path )
info_man.setPath( target = 'database', path = data_base_path )
info_man.setPath( target = 'reference', path = reference_path )

print "InfoManager is built for a default input set.¥n"
```

Fig. A.1-3 burnup-input.py.

```
from nuclide_data import *

### Material Set
mat_data = [{ 'Ni-nat.':7.79823e-05, 'Pu-240':0.000598514, 'Mo-nat.':5.96308e-05,
              'U-238':0.00653346, 'Nd-143':0.000275794, 'Pu-242':7.21147e-05,
              'Pu-241':7.98413e-05, 'Na-23':0.00732332, 'Am-241':3.71706e-05,
              'Cr-nat.':0.0024206, 'U-235':1.99109e-05, 'W-nat.':0.000124471,
              'Mn-55':0.000124962, 'Cm-244':1.83577e-05, 'Pu-239':0.00101294,
              'Am-243':1.84334e-05, 'O-16':0.0169491, 'Pu-238':2.06825e-05,
              'Np-237':9.4505e-06, 'Fe-nat.':0.0175173 }, # IC01, IC02

            { 'Ni-nat.':7.79823e-05, 'Pu-240':0.000666396, 'Mo-nat.':5.96308e-05,
              'U-238':0.00632977, 'Nd-143':0.000275794, 'Pu-242':8.02937e-05,
              'Pu-241':8.88966e-05, 'Na-23':0.00732332, 'Am-241':4.13863e-05,
              'Cr-nat.':0.0024206, 'U-235':1.92901e-05, 'W-nat.':0.000124471,
              'Mn-55':0.000124962, 'Cm-244':2.04398e-05, 'Pu-239':0.00112782,
              'Am-243':2.05241e-05, 'O-16':0.0169639, 'Pu-238':2.30282e-05,
              'Np-237':1.05223e-05, 'Fe-nat.':0.0175173 }, # OT01, OT02

            { 'Ni-nat.':7.79823e-05, 'Mo-nat.':5.96308e-05, 'Na-23':0.00732332,
              'U-235':2.4628e-05, 'U-238':0.00808131, 'W-nat.':0.000124471,
              'Fe-nat.':0.0175173, 'O-16':0.0162119, 'Cr-nat.':0.0024206,
              'Mn-55':0.000124962 }, # ABLU, ABLL

            { 'Ni-nat.':6.4778e-05, 'Mo-nat.':4.95338e-05, 'Na-23':0.00661882,
              'U-235':3.3389e-05, 'U-238':0.0109561, 'W-nat.':0.000103395,
              'Fe-nat.':0.0145512, 'O-16':0.021979, 'Cr-nat.':0.00201073,
              'Mn-55':0.000103803 }, # RBLU, RBLM, RBLL

            { 'Ni-nat.':0.00143655, 'Mo-nat.':0.00010992, 'Na-23':0.0203057,
              'Fe-nat.':0.00458722, 'Cr-nat.':0.00121654, 'Mn-55':0.000134329 }, # NAF0

            { 'Ni-nat.':0.00376282, 'Mo-nat.':0.000287732, 'Na-23':0.00722756,
              'Fe-nat.':0.0120114, 'Cr-nat.':0.00318543, 'Mn-55':0.000351733 }, # GPL1

            { 'Ni-nat.':0.0029681, 'Mo-nat.':0.000226961, 'Na-23':0.00685516,
              'Fe-nat.':0.00947455, 'Cr-nat.':0.00251265, 'Mn-55':0.000277445 }, # GPL2

            { 'Ni-nat.':0.00269258, 'Mo-nat.':0.000206026, 'Na-23':0.00722164,
              'B-11':0.0421443, 'B-10':0.0105361, 'C-12':0.0131701,
              'Cr-nat.':0.0022802, 'Mn-55':0.000251777, 'Fe-nat.':0.008598 }, # AXSU

            { 'Ni-nat.':0.0123677, 'Mo-nat.':0.000946329, 'Na-23':0.00555674,
              'Fe-nat.':0.0394927, 'Cr-nat.':0.0104735, 'Mn-55':0.00115647 }, # AXSL

            { 'Ni-nat.':0.0117553, 'Mo-nat.':0.000899472, 'Na-23':0.006383,
              'Fe-nat.':0.0375373, 'Cr-nat.':0.00995491, 'Mn-55':0.00109921 }, # RDS1

            { 'Ni-nat.':0.00342018, 'Mo-nat.':0.000261699, 'Na-23':0.00428467,
              'B-11':0.0488005, 'B-10':0.0122001, 'C-12':0.0152502,
              'Cr-nat.':0.00289636, 'Mn-55':0.000319813, 'Fe-nat.':0.0109214 }] # RDS2

### Region Set
material_set = [ Material(x) for x in mat_data ]
#print material_set
region_data = [ [ 1, "IC01", material_set[0], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 2, "IC02", material_set[0], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 3, "OT01", material_set[1], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 4, "OT02", material_set[1], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 5, "ABLU", material_set[2], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 6, "ABLL", material_set[2], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 7, "RBLU", material_set[3], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 8, "RBLM", material_set[3], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 9, "RBLL", material_set[3], [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                [ 10, "NAF0", material_set[4], [ 1.0 ], 0.0],
                [ 11, "GPL1", material_set[5], [ 1.0 ], 0.0],
                [ 12, "GPL2", material_set[6], [ 1.0 ], 0.0],
                [ 13, "AXSU", material_set[7], [ 1.0 ], 0.0],
                [ 14, "AXSL", material_set[8], [ 1.0 ], 0.0],
                [ 15, "RDS1", material_set[9], [ 1.0 ], 0.0],
                [ 16, "RDS2", material_set[10], [ 1.0 ], 0.0]
                ]

region_set = [ Region( no=x[0], name=x[1], material=x[2], assembly=x[3], exchange=x[4] ) for x in region_data ]
info_man.setRegions(region_set)
#print region_set
```

Fig. A.1-4 core_data.py.

```
### Geometry
geom = Geometry()

### Number of division of each mesh
div = [
          [ 2, 3, 3, 3, 2, 4, 4, 3, 2, 5, 4, 4, 2, 3, 2, 2, 4, 4, 4, 5, 3 ],   # X AXIS
          [ 5, 6, 4, 3, 3, 3, 4, 3, 3, 3, 4, 16, 5],                          # Y AXIS
          [ 1 ]                                                               # Z AXIS
          ]
geom.setMeshDivision( div )

### Width of each mesh
width = [
          [ 9.55043, 15.71764, 16.36130, 16.46364, 5.97324, 18.64292, 17.91088, 12.78505,
            4.72722, 21.25165, 17.83599, 18.10268, 4.62130, 13.20161, 10.12392, 3.55860,
           18.97454, 17.49924, 17.37424, 32.60152, 17.22164 ],
          [ 30.0000, 30.0000, 20.0000, 15.0000, 15.0000, 15.0000, 20.0000, 15.0000,
            15.0000, 15.0000, 20.0000, 100.000, 30.0000 ],
          [  1.0000 ]
          ]

geom.setMeshWidth( width )

### material number of each mesh
material_map = [[ 10,13,13,13,10,13,13,13,10,13,13,13,10,13,13,10,13,13,13,15,16 ],
                [ 10,11,11,11,10,11,11,11,10,11,11,11,10,11,11,10,11,11,12,15,16 ],
                [ 10, 5, 5, 5,10, 5, 5, 5,10, 5, 5, 5,10, 5, 5,10, 5, 5, 7,15,16 ],
                [ 10, 5, 5, 5,10, 5, 5, 5,10, 5, 5, 5,10, 5, 5,10, 5, 5, 7,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 6, 6, 6,10, 6, 6, 6,10, 6, 6, 6,10, 6, 6,10, 6, 6, 9,15,16 ],
                [ 10, 6, 6, 6,10, 6, 6, 6,10, 6, 6, 6,10, 6, 6,10, 6, 6, 9,15,16 ],
                [ 10,11,11,11,10,11,11,11,10,11,11,11,10,11,11,10,11,11,12,15,16 ],
                [ 14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,16 ]
                ]

geom.setMaterialMap( material_map )


### boundary condition
# XYZ: Left:X- Top:Y- Right:X+ Bottom:Y+ Front:Z- Back:Z+ Black_Region
# RZ: Left:X- Top:Y- Right:X+ Bottom:Y+ Dummy(0.0) Dummy(0.0) Black_Region
# dir: BC_LEFT BC_TOP BC_RIGHT BC_BOTTOM BC_FRONT BC_BACK BC_BLACK
#boundary = [ REFLECTIVE, VACUUM, VACUUM, VACUUM, REFLECTIVE, REFLECTIVE, DUMMY ]
#geom.setBoundaryCondition( boundary )

geom.setBoundaryConditionLeft( REFLECTIVE )
geom.setBoundaryConditionTop( VACUUM )
geom.setBoundaryConditionRight( VACUUM )
geom.setBoundaryConditionBottom( VACUUM )
geom.setBoundaryConditionFront( REFLECTIVE )
geom.setBoundaryConditionBack( REFLECTIVE )
geom.setBoundaryConditionBlackRegion( DUMMY )
info_man.setGeometry( geom )
```

Fig. A.1-4 core_data.py (cont'd).

```
from base import *

### Energy group Info.
jfs18g = [ 1.00000000e+07, 6.06530700e+06, 3.67879400e+06, 2.23130200e+06,
           1.35335300e+06, 8.20850000e+05, 3.87742094e+05, 1.83156406e+05,
           8.65169531e+04, 4.08677109e+04, 1.93045391e+04, 9.11882031e+03,
           4.30742480e+03, 2.03468396e+03, 9.61116516e+02, 4.53999298e+02,
           2.14454102e+02, 1.01300903e+02, 9.99999975e-06]
info_man.setEnergyBoundary( jfs18g )

### Nuclide Set
nuclide_names = [ 'U-235', 'U-236', 'Np-237', 'U-238', 'Np-239', 'Pu-238', 'Pu-239',
                  'Pu-240', 'Pu-241', 'Pu-242', 'Am-241', 'Am-242m', 'Am-243', 'Cm-242',
                  'Cm-243', 'Cm-244', 'Cm-245', 'U-235FP', 'U-238FP', 'Pu-239FP', 'Pu-241FP',
                  'O-16', 'Na-23', 'Cr-nat.', 'Mn-55', 'Fe-nat.', 'Ni-nat.', 'Mo-nat.',
                  'Nd-143', 'W-nat.', 'B-10', 'B-11', 'C-12' ]
nuclide_set = [ Nuclide(name=x) for x in nuclide_names ]

info_man.setNuclides( nuclide_set )

#print nuclide_set

### Burnup Chain
bc = BurnupChain()
#             Fertile Name, Reaction Type, [ Reaction Type, Daughter], [ Reaction Type, Daughter, Fraction ]
#             If Reactin Type is Fission, you need to input Daughter.
bc.appendRule( 'U-235', 'Capture', ['Fission', 'U-235FP'] )
bc.appendRule( 'U-236', ['Capture', 'Np-237'], ['Fission', 'U-235FP'], 'N2N' )
bc.appendRule( 'Np-237', ['Capture', 'Pu-238'], ['Fission', 'U-238FP'] )
bc.appendRule( 'U-238', ['Capture', 'Np-239'], ['Fission', 'U-238FP'], ['N2N', 'Np-237'] )
bc.appendRule( 'Np-239', ['Fission', 'Pu-239FP'], 'BetaDecay' )
bc.appendRule( 'Pu-238', 'Capture', ['Fission', 'U-238FP'], ['N2N', 'Np-237'] )
bc.appendRule( 'Pu-239', 'Capture', ['Fission', 'Pu-239FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Pu-240', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Pu-241', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'BetaDecay' )
bc.appendRule( 'Pu-242', ['Capture', 'Am-243'], ['Fission', 'Pu-241FP'], 'N2N' )
bc.appendRule( 'Am-241', ['Capture', 'Pu-242', 0.1384], ['Capture', 'Am-242m', 0.2000], ¥
               ['Capture', 'Cm-242', 0.6616], ['Fission', 'Pu-241FP'], ['N2N', 'Pu-240'], 'AlphaDecay' )
bc.appendRule( 'Am-242m', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', ['Decay', 'Pu-242', 0.173],
               ['Decay', 'Cm-242', 0.827] )
bc.appendRule( 'Am-243', ['Capture', 'Cm-244'], ['Fission', 'Pu-241FP'] )
bc.appendRule( 'Cm-242', 'Capture', ['Fission', 'Pu-241FP'], ['N2N', 'Am-241'], 'AlphaDecay' )
bc.appendRule( 'Cm-243', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Cm-244', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Cm-245', ['Fission', 'Pu-241FP'], 'N2N' )

info_man.setBurnupChain(bc)

### Sensitivity Calculation Option
so = SensitivityOption()

delta_xi = [-1.00, -0.90, -0.80, -0.70, -0.60, -0.50, -0.40, -0.30, -0.20, -0.10, -0.05,
             0.05, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00]
so.setDeltaFissionSpectrum( delta_xi )

# Capture, Nu, Transport, Fission, TotalScattering, ElasticScattering, InelasticScattering
# Mu, N2N, InelasticCrossSection N
so.appendTarget( 'U-235',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'U-236',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'U-237',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'U-238',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-238',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-239',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-240',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-241',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-242',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Am-241',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Am-242m',  'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Am-243',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Np-237',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Np-239',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Cm-242',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Cm-243',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Cm-244',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
```

Fig. A.1-5 nuclide_data.py.

```
so.appendTarget('O-16',     'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Na-23',    'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Cr-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Mn-55',    'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Fe-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Ni-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Zr-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Mo-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Nd-143',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('W-nat',    'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Pb-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('B-10',     'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('B-11',     'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('C-12',     'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Pu-241FP', 'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('U-235FP',  'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('U-238FP',  'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget('Pu-239FP', 'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )

info_man.setSensitivityOption(so)
```

Fig. A.1-5 nuclide_data.py (cont'd).

This is a blank page.

A.2 PSAGEP Development Manual

This is a blank page.

Table List

Figure List

This is a blank page.

A.2.1 Introduction

This document is intended for:

(1) The developer who expands the analysis scenario of PSAGEP
(2) The developer who encapsulates the existing code and carries out systematization with Python

In the 2nd chapter, the method of adding analysis scenario for PSAGEP will be explained. In reference to the analysis scenario that deals with the burnup sensitivity calculation of breeding ratio, the method of guiding the analysis scenario about burnup reactivity will be described in detail sequentially.

Chapter 3 explains encapsulation of calculation codes and data file conversion to the objects in order to control existing calculation codes from Python. Python systemization example is also shown in this chapter.

Examples of systemization using Python are also introdued.

Concerning the PSAGEP system, a mailing list has been established for information sharing. Inquiries should be directed to the following:

Mail:
Reactor Physics Engineering Group,
System Engineering Technology Division Section,
O-arai Engineering Center,
Japan Nuclear Cycle Development Institute
Kenji Yokoyama (e-mail: kyoko@oec.jnc.go.jp)

(Note) Current e-mail address: yokoyama.kenji09@jaea.go.jp

A.2.2 Adding New Scenarios

This section explains how to add analysis scenarios. In this chapter, it is assumed that readers have a knowledge of burnup sensitivity analysis and Python.

A.2.2.1 Workflow

The addition of a scenario is performed by adhering to the subsequent recommended procedure:

Initially, it is necessary to analyze the target scenario. Analysis of this scenario is conducted in relation to the following criteria:

(1) Analysis of the calculation flow
(2) Analysis of the peculiar input of a scenario (Input for each calculation code)
(3) Analysis of the peculiar calculation of a scenario (Calculation conditions for each calculation code)

In the following part, the description is based on the case that shell script has been prepared for SAGEP-BURN calculation. The description can also be referred to when creating a new shell script.

The above-mentioned scenario analysis is concerned about the files in the directories (e.g., br and burn-up etc.) for sensitivity analysis, which are included in the sagepburn directory. The commands of diff - bc and diff - y are used for investigating these differences. The comparison between breeding ratio (hereinafter br) and burnup reactivity loss (hereinafter burn-up) was done as an example.

First, the analysis of the calculation flow described in (1) is explained. Since the calculation flow of Forward do not change with the scenario, only the calculation flow of Adjoint is analyzed. With the comparison of the Burn-adj.sh that has already been installed with the one that is to be newly added, the difference of the calculation flow is extracted. The comparison result of br and burn-up is shown in Fig. A2-1. This figure is a result of command diff –y of Linux. As shown in this figure, symbols "<" and ">" indicate excess and lack of calculation, and symbol "|" indicates that the file name has been changed. This figure shows that there is an addition of a calculation in cycle 4 (the last cycle) and cycle 3 (last cycle -1), and that the shell name performed in cycle 3 has changed. Analysis results of a calculation flow are provided in these formats.

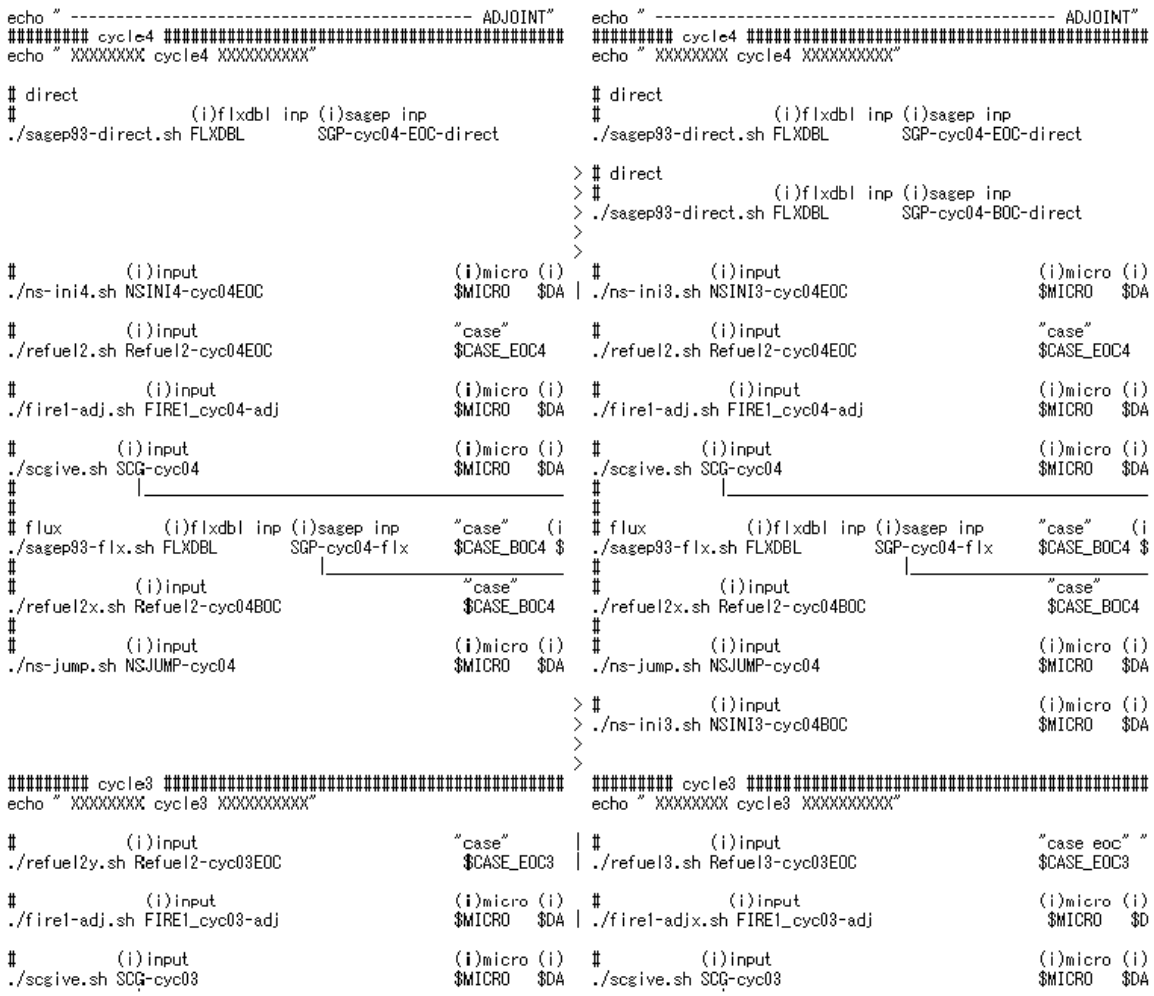


Fig. A.2-1 Comparison between Burn-adj.sh for br and Burn-adj.sh for burn-up.

Next, about the input file of the backend calculation code introduced in (2), the comparison between br scenario and burn-up scenario is done in the same way as described earlier. First, the excess and deficiency of the input file are investigated. Although there are many different methods for investigation, diff –y, which was used earlier, is utilized here. For each scenario, files that are arranged in order of file names are prepared. The results are shown in Fig. A2-2. Here, this example shows that the 4th cycle (i.e., the last cycle) of the calculation of SAGEP is used for both EOC and BOC. It also shows that the code is changed from NSINI4 to NSINI3.

```
PDSEDT-DATA                          PDSEDT-DATA
PDSEDT-NUCL                          PDSEDT-NUCL
SGP-cyc01-flx                        SGP-cyc01-flx
SGP-cyc02-flx                        SGP-cyc02-flx
SGP-cyc03-flx                        SGP-cyc03-flx
                                   > SGP-cyc04-BOC-direct
SGP-cyc04-EOC-direct                 SGP-cyc04-EOC-direct
SGP-cyc04-flx                        SGP-cyc04-flx
SAGEP-snslev                         SAGEP-snslev
NSINI4-cyc04EOC                    | NSINI3-cyc04BOC
                                   > NSINI3-cyc04EOC
NSJUMP-cyc02                         NSJUMP-cyc02
NSJUMP-cyc03                         NSJUMP-cyc03
NSJUMP-cyc04                         NSJUMP-cyc04
SCG-cyc01                            SCG-cyc01
SCG-cyc02                            SCG-cyc02
SCG-cyc03                            SCG-cyc03
SCG-cyc04                            SCG-cyc04
```

Fig. A.2-2 Excess and deficiency of the input of backend calculation.

Then, file contents are investigated. Although there are many different methods, the easiest method is to perform diff. In this case, the investigation is more efficient if performed as follows:

> diff –b br/Burn/(Input File) burn-up/Burn/(Input File) > diff_burn-up/inp/(Input File).diff

Merits of writing the result in re-direct are: the result is written in a file when there is a difference in the file, and an empty file is created when there is no difference in file. A different file can be searched out by executing ls -l according to the size of the file. The result is as follows:

```
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc01BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc01EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc02BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc02EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc03BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc03EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc04BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc04EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc01EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc02BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc02EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc03BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc03EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc04BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc04EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel3-cyc03EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SAGEP-snslev.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SCG-cyc01.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SCG-cyc02.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SCG-cyc03.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SCG-cyc04.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc01-flx.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc02-flx.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc03-flx.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc04-BOC-direct.diff
-rw-rw-r--    1 hyd      SE           718  4月 26 16:20 SGP-cyc04-EOC-direct.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc04-flx.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 STARTUP.diff
```

Fig. A.2-3 Changes of the input of backend calculation.

This result only shows the difference of the input of SAGEP calculation. The result of the excess and deficiency of the file ahead and the content of this file become the analysis results

of a peculiar input to the scenario.

Finally, for the execution shell of each calculation code of (3), whether the connecting file is changed or not is investigated. The procedure is the same as that mentioned in (2). The results are shown as follows:

```
cit-adj.sh                      cit-adj.sh
citation.sh                     citation.sh
convmic.sh                      convmic.sh
                              > differ.sh
fire1-adj.sh                    fire1-adj.sh
                              > fire1-adjx.sh
fire1.sh                        fire1.sh
fire2.sh                        fire2.sh
                              > ns-ini3.sh
ns-ini4.sh                      ns-ini4.sh
ns-jump.sh                      ns-jump.sh
pdsedit.sh                      pdsedit.sh
refuel.sh                       refuel.sh
refuel2.sh                      refuel2.sh
refuel2x.sh                     refuel2x.sh
refuel2y.sh                     refuel2y.sh
                              > refuel2z.sh
refuel3.sh                      refuel3.sh
sagep93-direct.sh               sagep93-direct.sh
sagep93-flx.sh                  sagep93-flx.sh
scgive.sh                       scgive.sh
```

Fig. A.2-4 Excess and deficiency of the execution shell.

```
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 cit-adj.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 citation.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 convmic.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 fire1-adj.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 fire1.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 fire2.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 ns-ini4.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 ns-jump.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 pdsedit.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 refuel.sh.diff
-rw-rw-r--    1 hyd      SE           122  4月 26 11:51 refuel2.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 refuel2x.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 refuel2y.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 refuel3.sh.diff
-rw-rw-r--    1 hyd      SE           400  4月 26 11:51 sagep93-direct.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 sagep93-flx.sh.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 11:51 scgive.sh.diff
-rw-rw-r--    1 hyd      SE           104  4月 26 11:51 start-up.sh.diff
```

Fig. A.2-5 Changes of the input of the execution shell.

Users can implement a new scenario by modifiying the copy of scenario class (BreedingRatioScenario.py) as necessary.

A.2.2.2 Modification of InputGenerator Class

InputGenerator class must contain an input generation block for a new scenario. As a result of the analysis of an input file (see (2) in previous section), the insufficiency in an input file name or a substantial difference in contents may correspond to the lack of an input generation block in InputGenerator. Basically, this can be solved by conditional branching with the use of the scenario_name variable. The portion actually used in the InputGenerator

class is shown below as an example. In the case of creating a special input for a scenario that is to be added, it is suitable to separate the conditions with the use of scenario names, as shown in the following figure.

```
body_format = []
body_value = []

scenario_name = self.info_man.getScenarioName()
so = self.info_man.getSensitivityOption()

# card 8: calculate option
if scenario_name == 'NumberDensity':
    calculation_option = 1
    target_nuclide = so.ndFocusedOption('nuclide')
    jendl_code_number = target_nuclide.id()
    num = 1
    for nuclide in self.info_man.getBurnableNuclides():
        if target_nuclide == nuclide:
            break
        num += 1
    number_of_burnup_chain_nuclide = num
    focused_region = so.ndFocusedOption('region')
else:
    calculation_option = 0
    jendl_code_number = 0
    number_of_burnup_chain_nuclide = 0
    focused_region = 0
```

Fig. A.2-6 A part of the InputGenerator class (conditional branching due to scenario name).

A.2.2.3 Modification of ModuleFile Class

It is necessary to modify a ModuleFile class when the definition of the calculation code to be used does not exist, or when the connection file is different. Please refer to the encapsulation section of Chapter 3 for the details about the ModuleFile.

A.2.2.4 Modification of SensitivityOption Class

This class stores and provides specific data for sensitivity calculation. Only the method of common data (the sensitivity and the perturbation quantity of the nuclear fission spectrum that calculates each nuclide) and the specific data of the breeding ratio are being implemented here.

Because this class is a container, it only stores and supplies data. Users must follow this rule when adding a new method.

A.2.2.5 Addition of a Scenario (Creation of Adjoint Scenario)

When adding a scenario, the contents of Burn-adj.sh should be rewritten using Python script. In the case of creating a scenario from the beginning, it is necessary to create a new scenario class using Burn-adj.sh. Therefore, since the scenarios of br and burn-up already exist, the scenarios to be added can be created by using the scenarios of br and burn-up as models. Scenario names are prepared as follows:

- K-Effective： Effective multiplication factor
- BurnupReactivity： Burn-up reactivity loss（Implemented）
- NumberDensity： Number density
- BreedingRatio： Breeding ratio（Implemented）
- ReactionRate： Reaction rate
- NaVoidReactivity： Na Void reactivity

These names are used in the InputGenerator and Scenario classes. Please use these names in creating a new analysis scenario as mentioned above.（NumberDensityScenario.py etc.）

Next, the fundamental structure of the scenario class is explained. A scenario class consists of the following four parts:

- Definition (calculation module and file manager, etc.)
- Control (control of calculation execution mode)
- Calculation execution (execution of Forward and Adjoint calculation)
- Debugging function

All of these parts except for the Adjoint calculation execution are included in the Scenario class, which is a super-class of each scenario class. Basically, these parts can be employed without any changes.

The calculation execution part consists of the main loop of the cycle and the execution of each calculation. When performing special calculations on a specific cycle (for example, the final cycle or the first cycle), the cycle is extracted by conditional branching.

An individual calculation execution consists of the following four steps:

- Creation of the definition of connection file
- File Connection
- Execution of calculation
- Selection of the calculation result

This example is shown in Fig. A.2-7.

Scenario constructions consist of the following two steps: (1) creation of the calculation execution part corresponding to execution shell of each calculation code and (2) arrangement of execution parts in sequence corresponding to Burn-adj.sh.

```
(Main loop: looping cycle)
for cy in range(total_cycle_number,0,-1):
(Defining connection file: created from InputGenerator)
sagep_direct_inp = self.ig.generate( code="sagep", cycle=cy, ・・・)
 (Difining connection file: taken out from FileManager)
flux_forward = self.fm.select( code="citation", type=・・・)
 (Connecting file)
self.sagep.input( [sagep_direct_inp, flux_forward, ・・・])
 (Executing calculation)
self.sagep.run( casename(cy, "EOC") )
 (Taking out calculation result from module class)
sagep_direct_outfiles = self.sagep.outfiles()
 (Storing calculation result in FileManager)
self.fm.add(sagep_direct_outfiles)
 (Executing debug: only when debug mode is specified)
if mode == "Debug"
    self.debugFunction(code='sagep', cycle=cy, ・・・)
```

Fig. A.2-7 Individual calculation execution in Scenario class (example: sagep code execution part).

A.2.2.6 Test Execution

User input of PSAGEP corresponding to the scenario, which has been added, is created in the psagep/data directory. Next, InputGeneratorTest.py class in psagep/test directory is modified. The following sentence is initially modified. Here, it is presumed that the *burnup_input* file has been created.

from data.br_input import * → from data.burnup_input import *

Secondly, the InputGeneratorTest class is modified for testing all the input appropriately. FigureFig. A.2-8 shows a part of the InputGeneratorTest class.

```
def testInputSAGEP(self):
for cy in [ 1, 2, 3, 4]:
  (Generating input by InputGenerator)
sagep_inp=self.ig.generate(code="sagep",cycle=cy,case="BOC",option="flux")
  (Specifying input name of reference for test)
refname=burn_dir[self.scenario_name] + "SGP-cyc%02d-flx" % cy
print refname
refname2=burn_dir[self.scenario_name] + "SAGEP-snslev"
os.system( "cat %s %s > /tmp/sagep.test " % (refname, refname2) )
  (Executing test)
assert sagep_inp.has_same_contents_with(RefFile("/tmp/sagep.test"), ¥
                'Breeding'),sagep_inp.diff_from( RefFile( "/tmp/sagep.test")
```

Fig. A.2-8 Test execution part of the InputGeneratorTest class (SAGEP code).

As shown in Fig. A.2-8, test execution part of InputGeneratorTest class consists of the following three steps:

- Creating input of backend code with InputGenerator class
- Specifying reference input of test
- Executing test

This is modified to cover all the input of the newly added scenario by combining cycle number, BOC, EOC and so on. After the test has executed, please confirm whether the data that is not expected to be modified, in input values of sample input, has been changed according to the cycle and calculation code.

After the confirmation that the input file is created correctly, the next step is to confirm whether the scenario class is functioned correctly or not. This is accomplished by executing psagep command in debug mode. Please execute the sample calculation beforehand at this time, and draw up a reference output for the test. The reference output can be created by executing Burn-normal.sh and Burn-adj.sh with sagepburn/burn-up/Burn directory. In addition, after all the calculations in the Burn directory are finished, the final result of SAGEP-BURN can be generated by executing mk_sns.sh with sagepburn/burn-up/table directory. After creation of a reference output is complete, the psagep command is executed in debug mode. The execution method is as follows:

```
>psagep.py –d (Input File)
```

After execution in debugging mode is successfully finished, the scenario creation is finished.

A.2.3 Encapsulation of An Existing Code

This chapter explains how to encapsulate the existing code with Python in tutorial form. To learn how to deal with this in Python language, please study the following example.

A.2.3.1 Merits of Encapsulation

The merit of encapsulation can be expressed by the phrase, "Simplification by abstraction". That is to say, users no longer have to wrestle with a complicated process. The system automatically creates a mechanism that solves this troublesome procedure. For example, users no longer have to be responsible for the logical unit for each input file, nor preprocessing of the calculation. The encapsulation concept improved the calculation procedure that was cumbersome when controlled by shell script. It has become a simple and flexible procedure when controlled by the Python layer.

Readers can experience the merits of encapsulation through the following example.

A.2.3.2 Example of Encapsulation

This shows the encapsulation an existing code. Here, the target is the FORTRAN code shown in Fig. A.2-9.

```
program MyFortCode
  read(5,*) imax, jmax
  do j=1, jmax
    do i=1, imax
      read(1,*) k
      write(6,*) i, j, k
      write(10,*) i, j, k
   enddo
  enddo
  end
```

Fig. A.2-9 MyFortCode.f of the encapsulation object.

The input / output files of the code are shown in Table A2-1. In addition, the input / outputs shown in parenthesis are automatically set up by the system.

Table A.2-1 Input/output files of MyFortCode

Unit Number	Input-Output	Creation Code	File Contents (Type)
1	Input	CODE1	COUNT_DATA
5	Input	-	(USER_CARD)
6	Output	-	(EDIT_LIST)
10	Output	(MYFORTCODE)	(COUNT_SUMMARY_DATA)

It is supposed that the compiled executable modules of MyFortCode shown in Fig. A.2-9 are /users/test/MyFortCode. In this case, the encapsulation class of MyFortCode is shown in Fig. A.2-10. It is assumed that MyModuleFile.py files, which include these definitions, are saved in ($prefix/sagep/File/) directory. MyModuleFile.py is also saved in the same directory. (Please save data in the above-mentioned directory when you actually work.)

Now, MyFortCode can be executed by using the Python interpreter.

First, boot up Python interpreter.

```
% python
Python 2.3.3 (#1, Apr 14 2004, 20:41:04)
[GCC 3.2.2 20030222 (Red Hat Linux 3.2.2-5)] on linux2
 Type "help", "copyright", "credits" or "license" for more information.
 >>>
```

Next, the definition of MyModuleFile.py that is mentioned earlier is read.

```
>>> from sagep.File.MyModuleFile import *
>>
```

If there is no problem, the prompt will be displayed as described above. Please check it again when an error message is shown here. A likely problem could be that there is a description mistake in MyModuleFile.py.

```
1:from sagep.File.ExecutableFile import *
2:from sagep.File.CardFile import *
3:from os.path import *
4:import string
5:
6:class MyFortCode(ExecutableFile):
7:  "Encapsulating class for MyFortCode"
8:  def __init__(self, case=""):
9:  ExecutableFile.__init__(self, "/users/test/MyFortCode", case )
10:
11:  # input file(s)
12:  self.defineio(unit=1, mode="i", generated="CODE1", type="COUNT_DATA", comment="Counting Data")
13:
14:  # output file(s)
15:  self.defineio(unit=10, mode="o", type = "COUNT_SUMMARY_DATA", comment="Counting Summary Data")
16:
17:  # user card and edit list
18:  self.defineio(unit=5, mode="i", type = "USER_CARD", comment="User Input Cards")
19:  self.defineio(unit=6, mode="o", type = "EDIT_LIST", comment="Editing List")
```

Fig. A.2-10 The encapsulation class MyFortCode.

(Please designate file name as MyModuleFile.py)
Row numbers are attached for convenience; they are not included in the actual file.

Next, the encapsulated object is substantiated. The substantiated object is called an instance. In the following example, the instance of calculation code object is created in code and name.

```
>>> code = MyFortCode()
>>>
```

Then, prepare the input user card (the 5th file) and the data file (the 1st file) for the calculation. In individual cases, input user card is created dynamically using the CardFile class.

```
>>> inp_text = [ "2 3" ]
>>> inp = CardFile( inp_text )
>>> inp.show()
===== CardFile (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = False
case =
contents =
data =
file <type 'file'> = <closed file '/tmp/psagep.test.22605/sagepCjmiEF',
mode 'w' at 0x400755e0>
fixed_path_flag <type 'bool'> = False
full_path_name = /tmp/psagep.test.22605/sagepCjmiEF
gen_code =
is_temporary <type 'bool'> = True
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = False
restore_path =
target_code =
type = USER_CARD
===== CardFile (END) =======
>>>
```

When creating a CardFile object, the string array of input card contents is specified as an argument when the constructor is invoked (the second line from the top). In the third line, all variables in an object are displayed by invoking the show() Method, and then the contents of the

variables are confirmed. In this case, the file has been automatically created temporarily in a temporary area (below /tmp directory).

Next, prepare the first file. This time, the existing file is the target. It is assumed that there is a file of /users/test/fort1.dat. The content is shown in Fig. A.2-11.

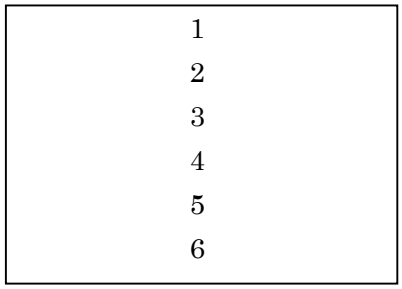


Fig. A.2-11 Contents of fort1.dat.

Then, encapsulate MyFortCode.inp with the DataFile object so that MyFortCode.inp can be abstracted.

```
>>> dat = DataFile("/users/test/fort1.dat")
>>> dat.show()
===== DataFile (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = False
case =
contents =
data =
fixed_path_flag <type 'bool'> = False
full_path_name = /users/test/MyFortCode.inp
gen_code =
is_temporary <type 'bool'> = False
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = True
restore_path =
target_code =
type =
===== DataFile (END) =======
>>>
```

Now, preparation of the input file is mostly finished. However, it is necessary to prepare a little more about the dat object.

As defined with the 12th line in Fig. A.2-10, the file that is connected to the 1st unit of MyFortCode requires the file type attribution of "COUNT_DATA" which is created in calculation module"CODE1". In other words, in the case of preparing one or more data files, the calculation code and files that have the same file type attribution with the files can be selected automatically. Therefore, the created calculation code and file type attribution are set to the dat object explicitly.

```
>>> dat.set_generate_code("CODE1")
>>> dat.set_filetype("COUNT_DATA")
>>> dat.show()
===== DataFile (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = False
case =
contents =
data =
fixed_path_flag <type 'bool'> = False
full_path_name = /users/test/fort1.dat
gen_code = CODE1
is_temporary <type 'bool'> = False
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = True
restore_path =
target_code =
type = COUNT_DATA
===== DataFile (END) =======
>>>
```

The gen_code and type which are internal variables of the dat object are respectively set in the above-mentioned example. (However, because these are internal variables, they are not generated directly by the user. Please use the method described such as set_generate_data ().)

The next step is to connect the file. The “inp” and “dat” objects are connected with code object.

```
>>> code.inputs([inp, dat])
>>>
```

Although no changes are shown here, many processes are performed regarding assignments to each file inside the object. Details are being omitted here. Following is an overview of the inside of the object. Though it is little complicated, the rough meaning will become clear by comparing the object variable name and its contents with the contents of Fig. A.2-10.

```
>>> code.show()
===== MyFortCode (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = False
case =
comment <type 'dict'> = {1: 'Counting Data', 10: 'Counting Summry Data',
5: 'User Input Cards', 6: 'Editing List'}
contents =
data =
       files <type 'dict'> = {1: <sagep.File.DataFile.DataFile instance
  at0x403aaa0c>, 5: <sagep.File.CardFile.CardFile instance at
0x403aa9ec>}
fixed_path_flag <type 'bool'> = False
full_path_name = /users/test/MyFortCode
gen_code =
generated <type 'dict'> = {1: 'CODE1', 10: '', 5: '', 6: ''}
hint <type 'dict'> = {1: '', 10: '', 5: '', 6: ''}
is_temporary <type 'bool'> = False
mod_name = MYFORTCODE
mode <type 'dict'> = {1: 'i', 10: 'o', 5: 'i', 6: 'o'}
omittable <type 'dict'> = {1: False, 10: False, 5: False, 6: False}
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = True
restore_path =
target <type 'dict'> = {1: '*', 10: '*', 5: '*', 6: '*'}
target_code =
     type <type 'dict'>={1: 'COUNT_DATA', 10: 'COUNT_SUMMRY_DATA', 5:
  'USER_CARD', 6: 'EDIT_LIST'}
  ===== MyFortCode (END) =======
  >>>
```

Dictionary variable *files* merit attention. This variable indicates that each file connects to the proper logical unit, and that instances of data file objects are saved.

Then, execute the calculation code.

```
>>> code.run("test_case")
=== EXECUTING MYFORTCODE for the case of TEST_CASE
0
>>>
```

The calculation case name is specified for the run() method that is executed. Any name will be fine. Here, consider the calculation case name as "test_case". After the run() method is invoked, the calculation terminiates normally with a message of “Executing“. The number displayed shows the error code and 0 means a normal termination.

Then, select the calculation result as follows: The list() and outfiles() methods are used respectively for selecting the calculation list (the 6th unit) and the output data file (units other than the 6th). When data is selected as a 1st object, data cannot be extracted as a file. To write the data into a specific file, the restore() method is used. The path name of the output re-direction is specified for the argument of the restore() method.

```
>>> lst = code.list()
>>> lst.show()
===== DataFile (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = True
case = TEST_CASE
contents =
data = [' 1 1 1¥n', ' 2 1 2¥n', ' 1 2 3¥n', ' 2 2 4¥n',
'1 3 5¥n', ' 2 3 6¥n']
fixed_path_flag <type 'bool'> = False
full_path_name =
gen_code = MYFORTCODE
is_temporary <type 'bool'> = True
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = False
restore_path =
target_code = *
type = EDIT_LIST
===== DataFile (END) =======
>>> lst.restore("/users/test/fort6.dat")
>>>
```

The No. 10 file is selected in the same way.　Here it is executed actually.

```
>>> out = code.outfiles()
>>> len(out)
1
>>> out[0].show()
===== DataFile (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = True
case = TEST_CASE
contents =
data = [' 1 1 1¥n', ' 2 1 2¥n', ' 1 2 3¥n', ' 2 2 4¥n',
' 1 3 5¥n', ' 2 3 6¥n']
fixed_path_flag <type 'bool'> = False
full_path_name =
gen_code = MYFORTCODE
is_temporary <type 'bool'> = True
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = False
restore_path =
target_code = *
type = COUNT_SUMMRY_DATA
===== DataFile (END) =======
>>> out[0].restore("/users/test/fort10.dat")
>>>
```

The output file is outputted as might have been expected.

```
% more /users/test/fort10.dat
1 1 1
2 1 2
1 2 3
2 2 4
1 3 5
2 3 6
%
```

This concludes the explanation of the encapsulation outline of how to use MyFortCode from the Python language.

A.2.3.3 File Management

As shown in the preceding section, the input card and the general file were processed as an abstracted file object. A FileManager class can be used to manage file objects.

FileManager is a kind of object database with the following functions. Please refer to the test code (FileManagerTest.py) for examples of each method.

- Addition of a file object (add method)
- Deletion of a file object (delete method)
- Acquisition of the number of objects (size method)
- Clearing of all objects (clear method)
- Selection of a file object (outfiles and select method)

Here, the merit of using FileManager is explained briefly. In the case of adding a file object to a FileManager object, object indices are attached with the following information:

(1) The calculation code of file object
(2) The filetype of file object
(3) The calculation case name

Additionally, an object can be selected by using the above-mentioned index as a key. For example, in the case of performing many calculation cases in the same calculation code, it is better to add an output file into the FileManager after setting a calculation case name appropriately. Later, the output file of a specific case can be selected by inquiring the FileManager with the case name as a key. In addition, in a Scenario class, FileManager is utilized as an object database by utilizing the selection function with this key.

Since FileManager class inherits SerializableAndVisible class (sagep.util module), FileManager class also supports serialization of an object. That is, preservation and restoration of a FileManager instance can be carried out. Preservaton and restoration functions can be used to restart calculations.

A.2.3.4 Additional Information

There are two ways to deal with the input of existing code. One is the method of using CardFile as explained in Section 3.2. The other is the method of preparing InfoManager and InputGenerator for concerning codes. When initializing a small-scale calculation code, it will not be necessary to utilize a complicated method like the latter, since small-scale calculation code can be solved with the former method.

Since the type and number of the files connected to code may change with the analysis scenario in PSAGEP, calculation codes are managed by Scenario class, which is the center of a control layer, and Psagep class which is located in higher layers. It will not be necessary to distinguish between Scenario class and Psagep class clearly in small-scale cases where calculation codes are not changed. Various applications can be considered by encapsulating the existing code. Those examples are shown in Table A.2-2. Please utilize this framework to substantiate individual situations.

Table A.2-2 PSAGEP Example of Encapsulation Framework

Use	Explanation
Parameter Survey	Several input files are automatically created from the Python layer and FileManager manages data files after executing the calculation job. If post-processing is also implemented by Python, post-processing can manage data consistently.
Coupling Calculation	Two or more calculation codes are connected by using the abstracted data file object. Since the control layer can be described with Python, algorithm development can be performed easily. The decentralized execution can also be implemented comparatively easily.
Prototype Implementation	Encapsulation is utilized in Prototype implementation, which is for examining the API (Application Program Interface) when the system is constructed. An encapsulated existing code is used as a stub for developing the system; hybrid development can be shifted to a scratch development smoothly by replacing the stub with a new development module.

This is a blank page.

国際単位系（SI）

表１．SI 基本単位

基本量	SI 基本単位	
	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光度	カンデラ	cd

表２．基本単位を用いて表されるSI組立単位の例

組立量	SI 基本単位	
	名称	記号
面積	平方メートル	m^2
体積	立法メートル	m^3
速さ，速度	メートル毎秒	m/s
加速度	メートル毎秒毎秒	m/s^2
波数	毎メートル	m-1
密度（質量密度）	キログラム毎立法メートル	kg/m^3
質量体積（比体積）	立法メートル毎キログラム	m^3/kg
電流密度	アンペア毎平方メートル	A/m^2
磁界の強さ	アンペア毎メートル	A/m
（物質量の）濃度	モル毎立方メートル	mol/m^3
輝度	カンデラ毎平方メートル	cd/m^2
屈折率	（数の）1	1

表５．SI 接頭語

乗数	接頭語	記号	乗数	接頭語	記号
10^{24}	ヨタ	Y	10^{-1}	デシ	d
10^{21}	ゼタ	Z	10^{-2}	センチ	c
10^{18}	エクサ	E	10^{-3}	ミリ	m
10^{15}	ペタ	P	10^{-6}	マイクロ	μ
10^{12}	テラ	T	10^{-9}	ナノ	n
10^{9}	ギガ	G	10^{-12}	ピコ	p
10^{6}	メガ	M	10^{-15}	フェムト	f
10^{3}	キロ	k	10^{-18}	アト	a
10^{2}	ヘクト	h	10^{-21}	ゼプト	z
10^{1}	デカ	da	10^{-24}	ヨクト	y

表３．固有の名称とその独自の記号で表されるSI組立単位

組立量	SI 組立単位			
	名称	記号	他のSI単位による表し方	SI基本単位による表し方
平面角	ラジアン(a)	rad		$m \cdot m^{-1}=1$(b)
立体角	ステラジアン(a)	sr(c)		$m^2 \cdot m^{-2}=1$(b)
周波数	ヘルツ	Hz		s^{-1}
力	ニュートン	N		$m \cdot kg \cdot s^{-2}$
圧力，応力	パスカル	Pa	N/m^2	$m^{-1} \cdot kg \cdot s^{-2}$
エネルギー，仕事，熱量	ジュール	J	N・m	$m^2 \cdot kg \cdot s^{-2}$
工率，放射束	ワット	W	J/s	$m^2 \cdot kg \cdot s^{-3}$
電荷，電気量	クーロン	C		s・A
電位差（電圧），起電力	ボルト	V	W/A	$m^2 \cdot kg \cdot s^{-3} \cdot A^{-1}$
静電容量	ファラド	F	C/V	$m^{-2} \cdot kg^{-1} \cdot s^4 \cdot A^2$
電気抵抗	オーム	Ω	V/A	$m^2 \cdot kg \cdot s^{-3} \cdot A^{-2}$
コンダクタンス	ジーメンス	S	A/V	$m^{-2} \cdot kg^{-1} \cdot s^3 \cdot A^2$
磁束	ウエーバ	Wb	V・s	$m^2 \cdot kg \cdot s^{-2} \cdot A^{-1}$
磁束密度	テスラ	T	Wb/m^2	$kg \cdot s^{-2} \cdot A^{-1}$
インダクタンス	ヘンリー	H	Wb/A	$m^2 \cdot kg \cdot s^{-2} \cdot A^{-2}$
セルシウス温度	セルシウス度(d)	℃		K
光束	ルーメン	lm	cd・sr(c)	$m^2 \cdot m^{-2} \cdot cd=cd$
照度	ルクス	lx	lm/m^2	$m^2 \cdot m^{-4} \cdot cd=m^{-2} \cdot cd$
（放射性核種の）放射能	ベクレル	Bq		s^{-1}
吸収線量，質量エネルギー分与，カーマ	グレイ	Gy	J/kg	$m^2 \cdot s^{-2}$
線量当量，周辺線量当量，方向性線量当量，個人線量当量，組織線量当	シーベルト	Sv	J/kg	$m^2 \cdot s^{-2}$

(a)ラジアン及びステラジアンの使用は、同じ次元であっても異なった性質をもった量を区別するときの組立単位の表し方として利点がある。組立単位を形作るときのいくつかの用例は表４に示されている。
(b)実際には、使用する時には記号rad及びsrが用いられるが、習慣として組立単位としての記号“１”は明示されない。
(c)測光学では、ステラジアンの名称と記号srを単位の表し方の中にそのまま維持している。
(d)この単位は、例としてミリセルシウス度m℃のようにSI接頭語を伴って用いても良い。

表４．単位の中に固有の名称とその独自の記号を含むSI組立単位の例

組立量	SI 組立単位		
	名称	記号	SI 基本単位による表し方
粘度	パスカル秒	Pa・s	$m^{-1} \cdot kg \cdot s^{-1}$
力のモーメント	ニュートンメートル	N・m	$m^2 \cdot kg \cdot s^{-2}$
表面張力	ニュートン毎メートル	N/m	$kg \cdot s^{-2}$
角速度	ラジアン毎秒	rad/s	$m \cdot m^{-1} \cdot s^{-1}=s^{-1}$
角加速度	ラジアン毎平方秒	rad/s^2	$m \cdot m^{-1} \cdot s^{-2}=s^{-2}$
熱流密度，放射照度	ワット毎平方メートル	W/m^2	$kg \cdot s^{-3}$
熱容量，エントロピー	ジュール毎ケルビン	J/K	$m^2 \cdot kg \cdot s^{-2} \cdot K^{-1}$
質量熱容量（比熱容量），質量エントロピー	ジュール毎キログラム毎ケルビン	J/(kg・K)	$m^2 \cdot s^{-2} \cdot K^{-1}$
質量エネルギー（比エネルギー）	ジュール毎キログラム	J/kg	$m^2 \cdot s^{-2} \cdot K^{-1}$
熱伝導率	ワット毎メートル毎ケルビン	W/(m・K)	$m \cdot kg \cdot s^{-3} \cdot K^{-1}$
体積エネルギー	ジュール毎立方メートル	J/m^3	$m^{-1} \cdot kg \cdot s^{-2}$
電界の強さ	ボルト毎メートル	V/m	$m \cdot kg \cdot s^{-3} \cdot A^{-1}$
体積電荷	クーロン毎立方メートル	C/m^3	$m^{-3} \cdot s \cdot A$
電気変位	クーロン毎平方メートル	C/m^2	$m^{-2} \cdot s \cdot A$
誘電率	ファラド毎メートル	F/m	$m^{-3} \cdot kg^{-1} \cdot s^4 \cdot A^2$
透磁率	ヘンリー毎メートル	H/m	$m \cdot kg \cdot s^{-2} \cdot A^{-2}$
モルエネルギー	ジュール毎モル	J/mol	$m^2 \cdot kg \cdot s^{-2} \cdot mol^{-1}$
モルエントロピー，モル熱容量	ジュール毎モル毎ケルビン	J/(mol・K)	$m^2 \cdot kg \cdot s^{-2} \cdot K^{-1} \cdot mol^{-1}$
照射線量（X線及びγ線）	クーロン毎キログラム	C/kg	$kg^{-1} \cdot s \cdot A$
吸収線量率	グレイ毎秒	Gy/s	$m^2 \cdot s^{-3}$
放射強度	ワット毎ステラジアン	W/sr	$m^4 \cdot m^{-2} \cdot kg \cdot s^{-3}=m^2 \cdot kg \cdot s^{-3}$
放射輝度	ワット毎平方メートル毎ステラジアン	$W/(m^2 \cdot sr)$	$m^2 \cdot m^{-2} \cdot kg \cdot s^{-3}=kg \cdot s^{-3}$

表６．国際単位系と併用されるが国際単位系に属さない単位

名称	記号	SI 単位による値
分	min	1 min=60s
時	h	1h =60 min=3600 s
日	d	1 d=24 h=86400 s
度	°	1 °=(π/180) rad
分	'	1 '=(1/60) °=(π/10800) rad
秒	"	1 "=(1/60) '=(π/648000) rad
リットル	l、L	1l=1 $dm^3=10^{-3}m^3$
トン	t	$1t=10^3$ kg
ネーパ	Np	1Np=1
ベル	B	1B=(1/2)ln10(Np)

表７．国際単位系と併用されこれに属さない単位で SI単位で表される数値が実験的に得られるもの

名称	記号	SI 単位であらわされる数値
電子ボルト	eV	$1eV=1.60217733(49) \times 10^{-19}J$
統一原子質量単位	u	$1u=1.6605402(10) \times 10^{-27}kg$
天文単位	ua	$1ua=1.49597870691(30) \times 10^{11}m$

表８．国際単位系に属さないが国際単位系と 併用されるその他の単位

名称	記号	SI 単位であらわされる数値
海里		1海里=1852m
ノット		1ノット=1海里毎時=(1852/3600)m/s
アール	a	$1a=1\ dam^2=10^2m^2$
ヘクタール	ha	$1ha=1\ hm^2=10^4m^2$
バール	bar	$1bar=0.1MPa=100kPa=1000hPa=10^5Pa$
オングストローム	Å	$1Å=0.1nm=10^{-10}m$
バーン	b	$1b=100fm^2=10^{-28}m^2$

表９．固有の名称を含むCGS組立単位

名称	記号	SI 単位であらわされる数値
エルグ	erg	$1\ erg=10^{-7}$ J
ダイン	dyn	$1\ dyn=10^{-5}N$
ポアズ	P	$1\ P=1\ dyn \cdot s/cm^2=0.1Pa \cdot s$
ストークス	St	$1\ St=1cm^2/s=10^{-4}m^2/s$
ガウス	G	1 G $\triangleq 10^{-4}$T
エルステッド	Oe	1 Oe $\triangleq (1000/4\pi)$A/m
マクスウェル	Mx	1 Mx $\triangleq 10^{-8}$Wb
スチルブ	sb	$1\ sb=1cd/cm^2=10^4cd/m^2$
ホト	ph	$1\ ph=10^4lx$
ガル	Gal	$1\ Gal=1cm/s^2=10^{-2}m/s^2$

表10．国際単位に属さないその他の単位の例

名称	記号	SI 単位であらわされる数値
キュリー	Ci	$1\ Ci=3.7 \times 10^{10}Bq$
レントゲン	R	$1\ R = 2.58 \times 10^{-4}C/kg$
ラド	rad	$1\ rad=1cGy=10^{-2}Gy$
レム	rem	$1\ rem=1\ cSv=10^{-2}Sv$
X線単位		$1X\ unit=1.002 \times 10^{-4}nm$
ガンマ	γ	$1\gamma=1\ nT=10^{-9}T$
ジャンスキー	Jy	$1\ Jy=10^{-26}W \cdot m^{-2} \cdot Hz^{-1}$
フェルミ		$1\ fermi=1\ fm=10^{-15}m$
メートル系カラット		1 metric carat = 200 mg = 2×10^{-4}kg
トル	Torr	1 Torr = (101 325/760) Pa
標準大気圧	atm	1 atm = 101 325 Pa
カロリー	cal	
ミクロン	μ	$1\ \mu=1\mu m=10^{-6}m$

（第７版，1998年改訂）