JAERI-M
9703

線形三重対角方程式の並列計算

1981年9月

石黒美佐子・原田裕夫・難波克光*
藤井　実・藤村統一郎・中村康弘

日本原子力研究所
Japan Atomic Energy Research Institute

JAERI-M 9703

線形三重対角方程式の並列計算

日本原子力研究所東海研究所計算センター
石黒美佐子・原田裕夫・難波克光*
藤井　実・藤村統一郎+・中村康弘

（１９８１年９月７日受理）

近年，大型プログラムの高速計算の必要性などから，科学技術計算に対する並列計算の適用が盛んになってきた。原研計算センターにおいても，アレイ・プロセッサF230-75 APUが並列計算の原子力コードへの適用を研究するために設置された。APUを使用して，科学計算でよく問題となる線形三重対角システムの解法について並列計算手法に関する最近の論文を参考にして，数値実験を行った。

解法は，ガウス消去法，パラレル・ガウス法，加速パラレル・ガウス法，ヤコビ法，リカーシブ・ダブリング法，サイクリック・リダクション法，チェビシェフ反復法，共役傾斜法の8種について調査された。

数値実験の結果にもとづいて，各解法の計算時間，精度の比較がなされた。

結果として，計算速度，精度の面でサイクリック・リダクション法と，逐次解法ではあるがガウスの消去法が勝れていることがわかった。

+　原子炉工学部

*　筑波大学

JAERI-M 9703

Parallel Computation for Solving the Tridiagonal Linear System of Equations

Misako ISHIGURO, Hiroo HARADA, Katsumi NANBA*
Minoru FUJII, Toichiro FUJIMURA+ and Yasuhiro NAKAMURA
Computing Center, Tokai Research Establishment,JAERI

(Received September 7, 1981)

Recently, applications of parallel computation for scientific calculations have increased from the need of the high speed calculation of large scale programs. At the JAERI computing center, an array processor FACOM 230-75 APU has installed to study the applicability of parallel computation for nuclear codes.

We made some numerical experiments by using the APU on the methods of solution of tridiagonal linear equation which is an important problem in scientific calculations.

Referring to the recent papers with parallel methods, we investigate eight ones. These are Gauss eliminaton method, Parallel Gauss method, Accelerated parallel Gauss method, Jacobi method, Recursive doubling method, Cyclic reduction method, Chebyshev iteration method, and Conjugate gradient method.

The computing time and accuracy were compared among the methods on the basis of the numerical experiments.

As the result, it is found that the Cyclic reduction method is best both in computing time and accuracy and the Gauss elimination method is the second one.

Keywords: Computer, Tridiagonal Equations, Parallel Computations, Recursive Doubling, Accelerated Gauss, Cyclic Reduction, Chebyshev Iteration, Conjugate Gradient Method, Computing Time, Comparative Evaluations, Accuracy

---

+) Division of Reactor Engineering, Tokai Research Establishment, JAERI
*) Tsukuba University

目　　次

CONTENTS

1. はじめに

1.1 数値実験の概要

近年，大型プログラムの高速計算の必要性などから，科学技術計算に対する並列計算処理の適用が盛んになってきた。原研計算センターにおいても，主要計算機Ｍ２００の他に，アレイプロセッサ（ＡＰＵ）付Ｆ２３０－７５計算機が使用可能になり（Fig.1，Fig.2，Table 1），技術計算でよく問題となる線形三重対角方程式の解法について，並列計算手法に関する論文等を参考にして，実際に並列計算機で数値実験を行った。

線形三重対角方程式を解く。ただし，係数行列は優対角すなわち

$$|d_i| \geqq |e_i| + |f_i|$$

とする。

線形三重対角システム

$$Ax = b$$

$$\begin{pmatrix} d_1 & f_1 & & & & \\ e_2 & d_2 & f_2 & & & \\ & e_3 & d_3 & f_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & e_{n-1} & d_{n-1} & f_{n-1} \\ & & & & e_n & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

Ａは優対角行列とする。

解法は，三重対角用ガウスの消去法[1]（ＧＥＴ），パラレル・ガウス法[2]（ＰＧ），加速パラレル・ガウス法[2]（ＡＰＧ），ヤコビ法[1]（ＪＣ），リカーシブ・ダブリング法[3]（ＲＤ），サイクリック・リダクション法[4]（ＣＲ），チェビシェフ反復法[5]（ＣＨ），共役傾斜法[6]（ＣＧ）の８種類について行った。

各解法毎に，Ｍ２００，Ｆ75CPU用の逐次処理プログラムとＦ75APU用の並列処理プログラムの２種類を作成した。並列処理プログラムはFORTRANにアレイやベクトル処理のための文が強化されているAP－FORTRANで書いた。AP－FORTRANにより記述されたプログラムの例をFig.3で示す。

係数行列の入力データは，次の５種類のデータで，各２種（優対角の度合の強いものと弱いもの）の合計１０種類で行った。ただし，対角要素は，すべて1.0とした。

(1)定数データ（例$e_i = f_i = 0.3$），(2)線形変化データ（例$e_i = -0.49 + \Delta e \cdot i$），(3)Givensデータ（例$e_i = f_i = -0.5$，$f_1 = -0.3333$，$e_n = -1.0$）（Fig.4），(4)乱数デー

タ（例 $0.2<e_i, f_i<0.4$ の一様乱数），(5)中性子拡散プログラムから抽出したデータ（例 $0.15<e_i, f_i<0.2$）

次数Nは，100，500，1000，5000 について行った。

1.2 解法の概要

数値実験された8種の解法の概要を以下に述べる。

(1) 三重対角用ガウスの消去法（GET）[1]

（Gauss Elimination Method for Tridiagonal Equation）

これは，いわゆる基本のガウスの消去法を三重対角行列用に改良したものであり，本質的に逐次的にしか計算できない解法である。

(2) パラレル・ガウス法（PG）[2]

（Parallel Gauss Method）

これは，ガウスの消去法をLDR分解を利用して，並列計算用に改良したものである。

(3) 加速パラレル・ガウス法（APG）[2]

（Accelerated Parallel Gauss Method）

これは，パラレル・ガウス法における収束計算を数個おきに計算し，次の計算にその結果を代入することにより収束を加速させる改良と，反復回数の上限を設定する改良を行ったものである。

(4) ヤコビ法（JC）[1]

（Jacobi Iteration Method）

これは，反復法の基本をなすものである。

(5) リカーシブ・ダブリング法（RD）[3]

これは，H.S.Stone によって発表された解法である。LU分解に基づく並列計算用の解法で，特にILLIAC Ⅳ用に作成されたものである。

(6) サイクリック・リダクション法（CR）[4]

（Cyclic Reduction Method）

これは，三重対角システムを解くための手法として，D. Heller によって，発表された解法である。三重対角行列の非対角要素を外側に追い出し，最後に対角行列にするという手法である。

(7) チェビシェフの反復法（CH）[5]

（Cyclic Chebyshev Polynomial Iteration Method）

この解法は，最適加速パラメータの近似値が不明のとき，反復法の中で最も速いとされている。

(8) 共役傾斜法（CG）[6]

（Conjugate Gradient Method）

反復法で真の解に逐次的に近づけてゆく方法である。残差ベクトル $r^{(i)}=b-Ax^{(i)}$ の直交性を利用している。丸め誤差がなければ高々n回で真の解に到達する。

これらの各解法についての大まかな特徴はFig.5に示される。

各種解法についての計算速度，精度などについては詳細は以下2章で論じられるが，Fig. 6により各解法の計算時間の一瞥を与える。サイクリック・リダクション法が一番速いことがわかる。

Table 1a. Characteristics and performances of FACOM 230-75 APU

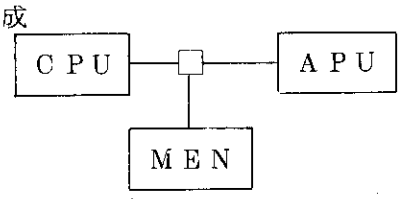
Items		Specifications
Machine cycle (Basic clock for pipeline)		90 ns
Elements	High-speed CML (Current Mode Logic)	3 ns
	NLT (Non Threshold Logic)	1 ns
	IC memory	14 ns or 35 ns
Buffers	Instruction buffer	8-word
	Data buffer	256-kiloword × 3
	Buffer memory	2-kiloword
Registers	Data register	256
	Vector register	1792-word
Operation pipeline		Addition pipeline Multiplication pipeline Logical operation pipeline
Vector operation time (Floating point, single precision)		Addition 22 MFLOPS Multiplication 11 MFLOPS Division 1.2 MFLOPS Inner product 22 MFLOPS Summation 22 MFLOPS

Table 1b. Characteristics and performances of FACOM 230-75 CPU

Items		Specifications	
CPU cycle time		90 ns	
Buffer memory	Size	2- or 4-kiloword	
	Access time	90 ns/2-word	
Core memory	Size	64- (min)/1024-kiloword (max)	
	Memory cycle time	1 μs/2-word	
Bit pattern		40-bit (36 data bits + 4 flag bits)	
		Fixed point	Floating point
Operation time	Addition-subtraction	108 ns	360 ns
	Multiplication	450 ns	540 ns
	Division	2250 ns	1350 ns
Gibson mix		267 ns	
Operating system		FACOM MONITOR VII	
Program language		FACOM FORTRAN-H	
Compiler		FACOM FORTRAN IV	

F230－75APUの特徴

1. パイプライン方式の並列計算機
2. 構　　成



3. 1語は36ビット
   （数値実験は単精度で計算）
4. プログラムは，AP－FORTRAN

Fig.1 Feature of F230－75 APU

並列計算プログラム

1. 並列化できる例

```
DO  10  I = 1 , N
10 A (I) = B (I) + C (I)*D (I)
```

⇓

```
INDEX  I／1 , N／
A (I) = B (I) + C (I)*D (I)
```

2. 並列化できない例（GETなど）

```
DO  20  J = 2 , N
20 B (J) = B (J) - E (J) *B (J-1)
```

Fig.3 Program by AP－FORTRAN

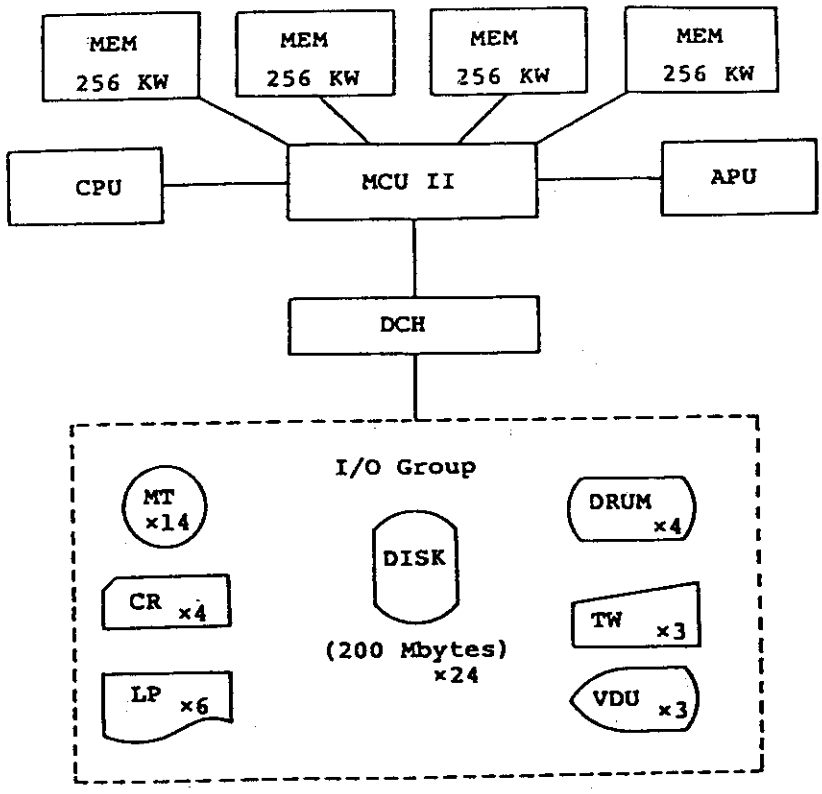


Fig. 2 System configuration of FACOM 230-75 APU/CPU

Givens のデータ

$$\begin{bmatrix} 1.0 & -0.3333 & & & & \\ -0.5 & 1.0 & -0.5 & & & \\ & -0.5 & 1.0 & -0.5 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & \cdot & \cdot \\ & & & -0.5 & 1.0 & -0.5 \\ & & & & -1.0 & 1.0 \end{bmatrix}$$

$d_i \geq e_i + f_i$

Fig. 4 Givens matrix

解　法	反復解法	並列計算向	三重対角用
GET			
PG	○	○	
APG	○	○	○
JC	○		
RD		○	○
CR		○	○
CH	○		
CG	○		

Fig. 5 Distinguishings of the methods

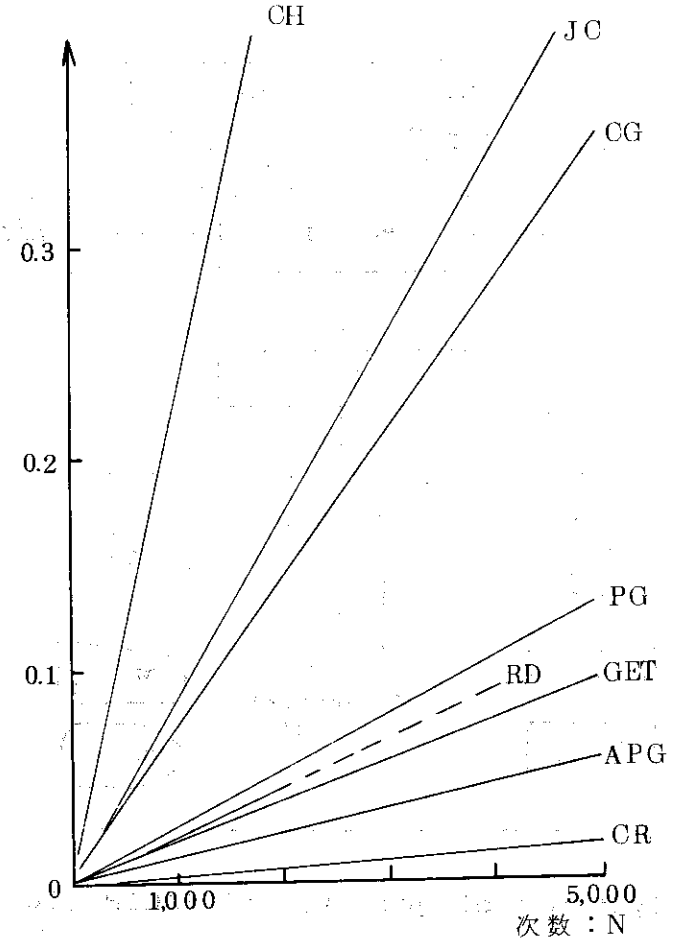


Fig. 6 Computing time on F230－75 APU (Random data)

2. 各解法の考察

2.1 三重対角用ガウスの消去法（ＧＥＴ）

2.1.1 解法

解くべき線形三重対角システムを

$$A\,x = b$$

とする。要素別に書くと，

$$\begin{cases} d_1 x_1 + f_1 x_2 = b_1 \\ e_2 x_1 + d_2 x_2 + f_2 x_3 = b_2 \\ e_3 x_3 + d_3 x_3 + f_3 x_4 = b_3 \\ \cdots\cdots\cdots \\ e_{n-1} x_{n-2} + d_{n-1} x_{n-1} + f_{n-1} x_n = b_{n-1} \\ e_n x_{n-1} + d_n x_n = b_n \end{cases}$$

となる。

解法は前進消去のステップで対角下側の要素を０とし，後退代入のステップで対角上側の要素を０にする。

(1) 前進消去ステップ

第１行目の式は

$$x_1 + \frac{f_1}{d_1} x_2 = \frac{b_1}{d_1}$$

ここで，

$$f_1' = \frac{f_1}{d_1} \quad , \quad b_1' = \frac{b_1}{d_1}$$

とおくと，

$$x_1 + f_1' x_2 = b_1'$$

となる。

第２行目の式は，

$$(d_2 - e_2 \cdot f_1')\, x_2 + f_2 x_2 = b_2 - e_2 \cdot b_1'$$

ここで，

$$d_2' = d_2 - e_2 \cdot f_1' \ , \ f_2' = \frac{f_2}{d_2'} \ , \ b_2' = \frac{b_2 - e_2 \cdot b_1'}{d_2'}$$

とおくと，

$$x_2 + f_2' x_3 = b_2'$$

となる。

以下同様にして，前進消去のステップを進めると，第k行目では，

$$(d_k - e_k \cdot f_{k-1}') x_k + f_k x_{k+1} = b_k - e_k \cdot b_{k-1}'$$

ここで

$$d_k' = d_k - e_k \cdot f_{k-1}' \quad , \quad f_k' = \frac{f_k}{d_k'} \quad , \quad b_k' = \frac{b_k - e_k \cdot b_{k-1}'}{d_k'}$$

とおくと，

$$x_k + f_k' x_{k-1} = b_k'$$

となる。

このようにして，前進消去のステップが第n行目まで終わると，

$$\begin{cases} x_1 + f_1' x_2 = b_1' \\ x_2 + f_2' x_3 = b_2' \\ x_3 + f_3 x_4 = b_3' \\ \\ x_{n-1} + f_{n-1}' x_n = b_{n-1}' \\ x_n = b_n' \end{cases}$$

の形になる。

(2) 後退代入ステップ

後退代入のステップでは，逆の順序で逐次代入していく。

$$x_{n-1} = b_{n-1}' - f_{n-1}' x_n$$

$$x_k = b_k' - f_k' x_{k+1}$$

$$x_1 = b_1' - f_1' x_2$$

2.1.2 プログラム

この解法をFORTRAN のプログラムにする。ただし，$d_1 = 1.0$ であること，除算より乗算の方が速いことを考慮に入れてプログラムすると次のようになる。

前進消去

```
DO  10  I=2, N
    D (I) =1.0／(D (I) －E (I) ＊F (I－1))
    F (I) =F (I) ＊D (I)
```

```
      B (I) = (B (I) －E (I) ＊B (I－1)) ＊D (I)
   10 CONTINUE
```

後退代入

```
   X (N) =B (N)
   DO 20 JB=1, N－1
      J=N－JB
      X (J) =B (J) －F (J) ＊X (J＋1)
   20 CONTINUE
```

2.1.3 メモリサイズと計算時間の推定値

この解法のプログラムに必要なメモリは，長さｎの１次元配列が５個である。

計算時間は，１回の加算（減算）時間をａ，乗算時間をｍ，除算時間をｄとすると，前進消去のステップでは，

$$2(n-1)a+4(n-1)m+(n-1)d$$

後退代入のステップでは，

$$2(n-1)a+(n-1)m$$

合計では，およそ

$$4na+5nm+nd$$

を要する。

2.1.4 特徴

この解法は，プログラムから見てもわかるように，非常に簡潔であり，計算速度も速く，解の精度についてもGivens のデータを除いて求める精度が得られた（Table 2参照）。

一方，この解法は，直接法で，完全な逐次計算解法であるため，並列計算は適用できない。

2.2 パラレル・ガウス法（ＰＧ）及び加速パラレル・ガウス法（ＡＰＧ）

2.2.1 概　要

ＡＰＧ（Accelerated Parallel Gauss）は，三重対角システムの繰返し解法の一種であり，Hellerらによって提案された。

文献[2]によると，ＩＬＬＩＡＣⅣ　ＳＴＡＲ－１００，ＡＳＣ，ＣＲＡＹ－１などの並列計算機の出現によって，非常に効率のよいベクトル演算を含んだ新しいアルゴリズムが次々に開発されてきた。三重対角システムに関しては，Stone が並列計算機への直接解法を，Traub が繰返し手法を初期に考案し，これらの手法の比較をLambiotte とVoigt がＳＴＡＲ－１００で行っている。

ＡＰＧは，Gauss の消去法を並列計算用に改良したTraub のParallel Gauss に以下の改良を行ったものである。

(1) 収束計算をｋ個おきに計算し，次の計算にその結果を代入することにより収束を加速させた。

(2) 収束回数の上限を設定した。

2.2.2 Gauss の消去法，Parallel Gauss ，ＡＰＧアルゴリズム

次の三重対角システムを考える。

$$A x = c \qquad (1)$$

ここで，便宜上Ａの対角要素は１にとる。

$$A = \begin{pmatrix} 1 & b_1 & & & & \\ a_2 & 1 & b_2 & & & \\ & a_3 & 1 & b_3 & & \\ & & & & & \\ & & a_{n-1} & 1 & b_{n-1} & \\ & & & a_n & 1 & \end{pmatrix} \qquad (2)$$

〔 Gauss の消去法 〕

(1) Ａ＝（Ｉ＋Ｌ）Ｄ（Ｉ＋Ｒ）に分解する。

$$d_1 = 1$$
$$d_j = 1 - a_j b_{j-1} / d_{j-1} \qquad (j = 2, 3, \cdots, n)$$

(2) （Ｉ＋Ｌ）ｆ＝ｃを解く。

$$f_1 = c_1$$
$$f_j = c_j - l_j f_{j-1} \qquad (j = 2, 3, \cdots, n)$$

ここで，$l_j = a_j / d_{j-1}$ 。

(3) （Ｉ＋Ｒ）$x = D^{-1} f$

$$x_n = g_n$$
$$= g_j - r_j x_{j+1} \qquad (j = n-1, n-2, \cdots, 1)$$

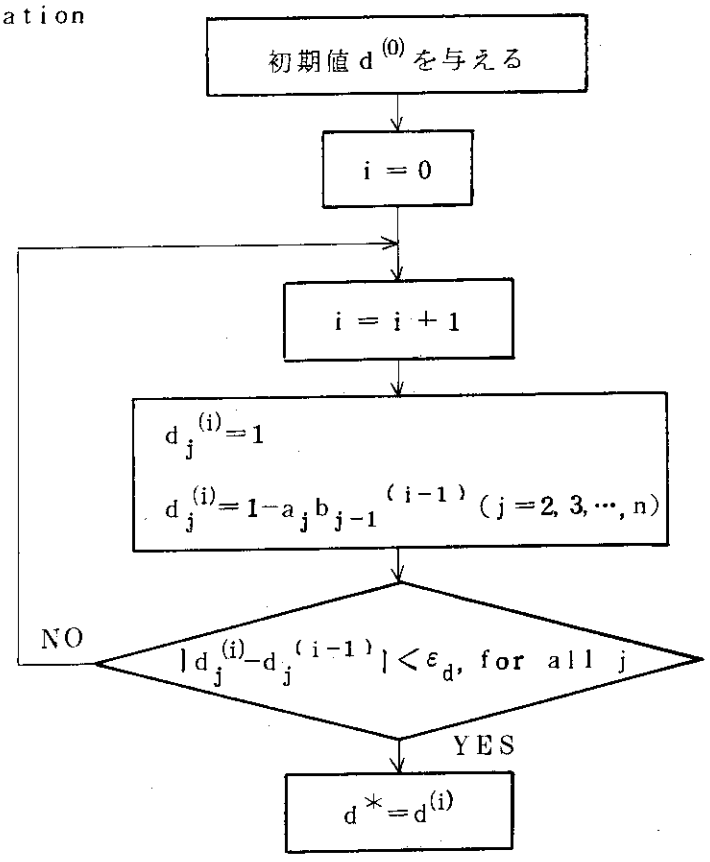
ここで，$r_j = b_j / d_j$ ，$g_j = f_j / d$ 。

このGaussの消去法は，各計算において直前の計算結果を用いているので，本質的に逐次計算手法である。

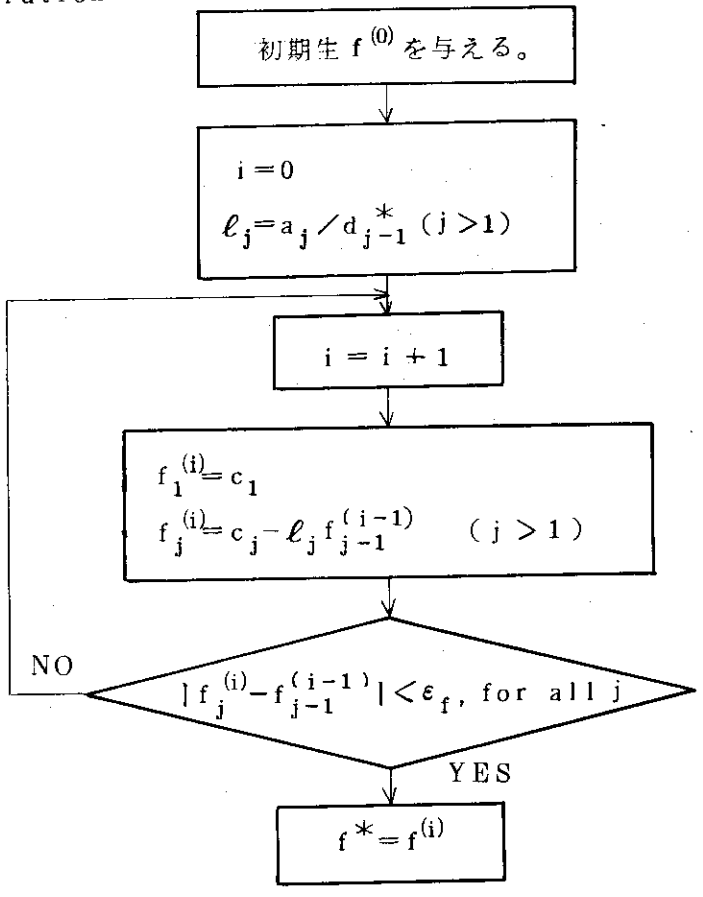
〔 Parallel Gauss 〕

Traub は，Gaussの消去法の各ステップをそれぞれ繰返し近似で求めることにより，並列計算向きに改良した。ε_d，ε_f，ε_x はそれぞれＤ，Ｆ，Ｇ－iteration の収束精度である。

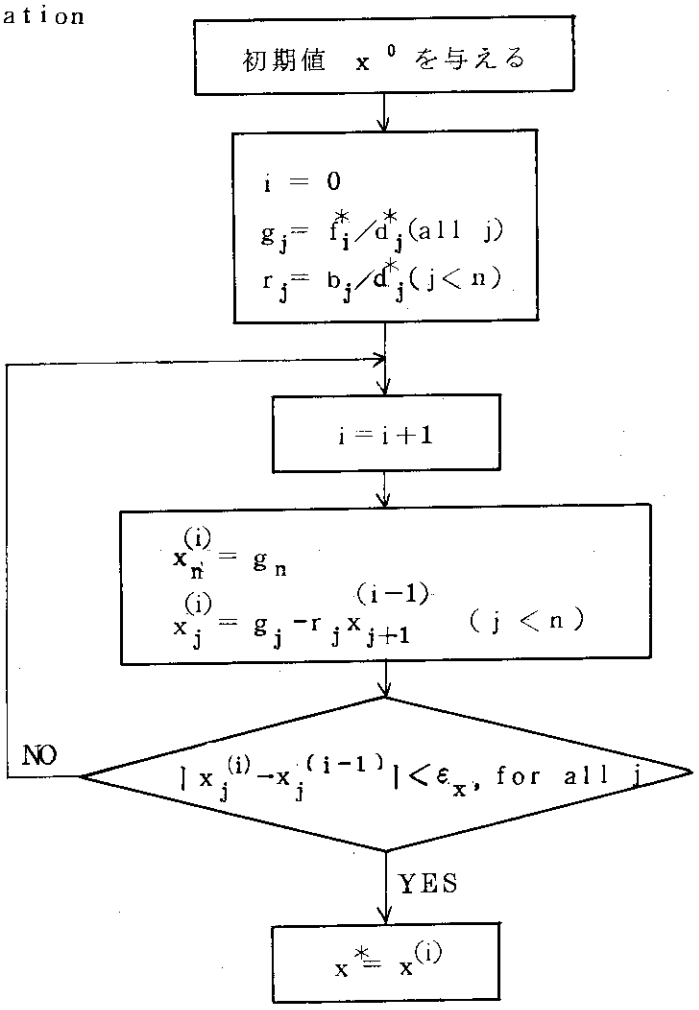
(1) D−iteration



(2) F−iteration



(3) G－iteration



〔APG〕

Hellerらは，Parallel Gaussの各繰返しがベクトル添字で直前の値しか使用していないことに着眼し，k個おきに繰返し計算を行わせ，収束をk倍速くさせるAPG手法を示した。同時に文献[2]では，並列計算機のあまり得意でない収束判定を除去するために，あらかじめ各繰返し計算の収束に必要な繰返し回数の算出も述べている。この手法のD, F, G－iterationの記述においてはk＝5（odd－even）の場合を示す。

(1) 繰返し回数ID，IF，IXの算出

① D, F, G－iteration の収束判定 ε_d, ε_f, ε_x を与える。

② $\lambda = \max_j |a_j b_j|$ を算出する。（$\lambda < 1$）

③ 収束率 $R = \dfrac{1-\sqrt{1-\lambda}}{1+\sqrt{1-\lambda}}$ を算出する。

④ ID，IF，IXを算出する。

$$R * \left(R^k\right)^{ID-1} < \varepsilon_d$$

より，　　$ID=\left[\left(\frac{\log_{10}\varepsilon_d}{\log_{10}R}+k-1\right)\Big/k\right]+1$

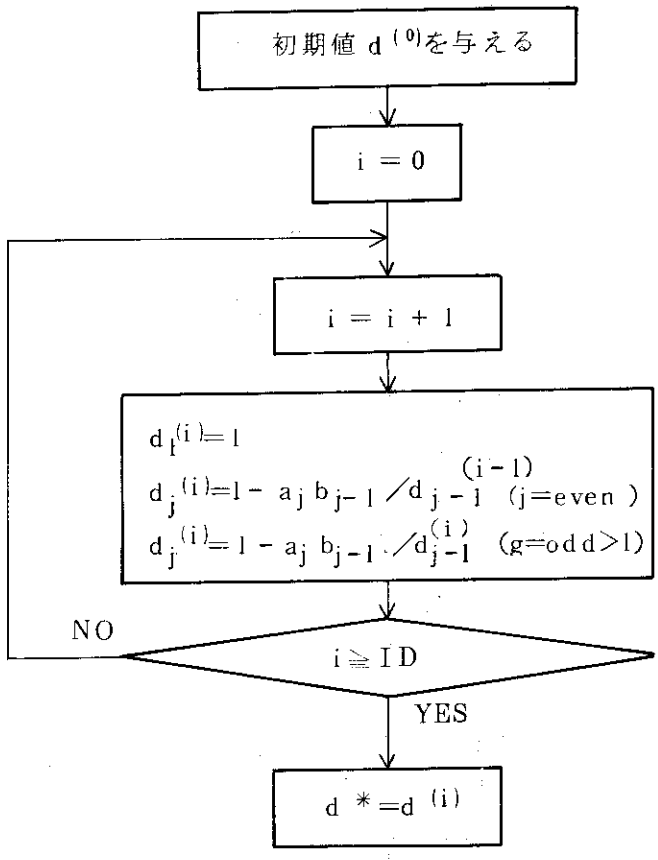
である。ここで〔 〕はガウス記号である。同様に，

$$IF=\left[\left(\frac{\log_{10}\varepsilon_f}{\log_{10}R}+k-1\right)\Big/k\right]+1$$

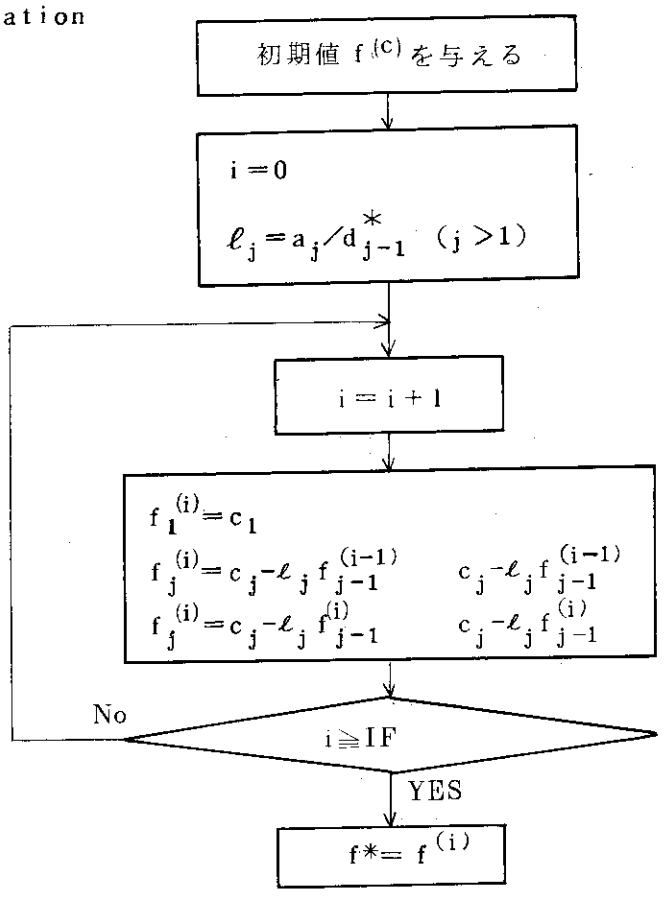
$$IX=\left[\left(\frac{\log_{10}\varepsilon_x}{\log_{10}R}+k-1\right)\Big/k\right]+1$$

である。

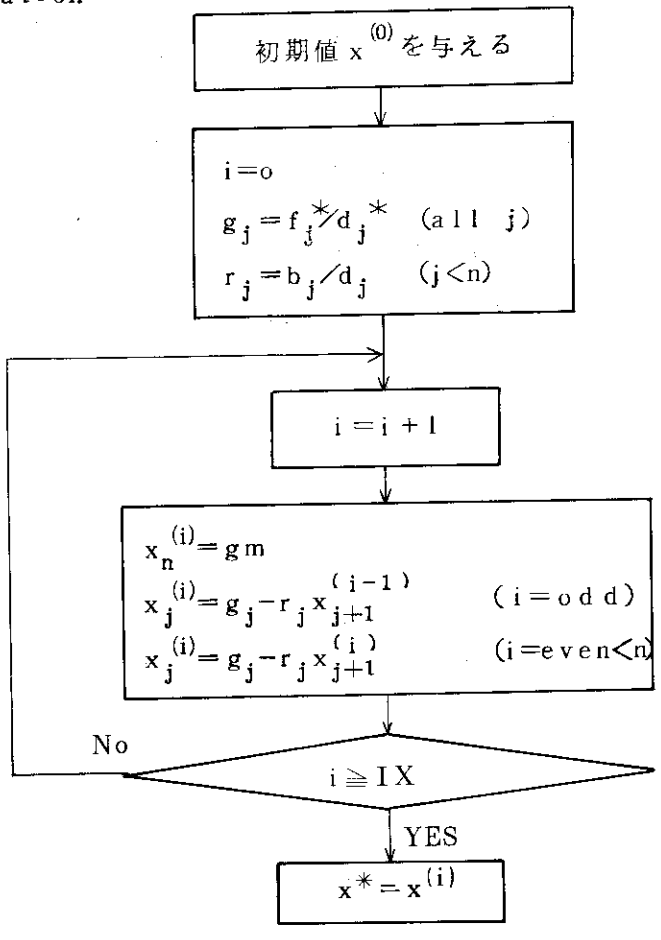
(2)　D－iteration



(3) F－iteration



(4) G－iteration



2.2.3　数値実験

ＡＰＧの数値実験は，ｋ＝１，ｋ＝５について行った。ただし，2.2.2で示したHellerらの繰返し回数の算出式は必ずしも正しくないので，ＡＰＧにおいても各繰返しにおける収束判定チェックをParallel Gaussのところで示したものと同様な方法で行っている。Ｄ，Ｆ，Ｇ－iterationのそれぞれの収束判定値は10^{-5}，10^{-4}，10^{-4}を使った。Givensのデータのみは10^{-7}，10^{-5}，10^{-5}を使った（Table 4参照）。

ｋ＝１のときはParallel Gaussと同じになるが，ＡＰＧ用の処理プログラムを使っているために実際のParallel Gaussをベクトル化したときよりＦ７５ＡＰＵの計算時間は多くかかっている（Table 3参照）。

ｋ＝５のときは，

最初に　　１，６，１１，１６，・・・・・，
次に　　２，７，１２，１７，・・・・・，
〃　　３，８，１３，１８，・・・・・，
〃　　４，９，１４，１９，・・・・・，
〃　　５，10，１５，２０，・・・・・

の順に計算されている。

Table 3に示すＦ７５ＡＰＵ計算時間の約2/3は収束判定にかかっている時間である。

2.3　ヤコビ法（ＪＣ）

2.3.1　アルゴリズム

2.2の(1)，(2)式より

$$\begin{aligned} x_1 + f_1 x_2 &= b_1 \\ e_2 x_1 + x_2 + f_2 x_3 &= b_2 \\ e_3 x_2 + x_3 + f_3 x_4 &= b_3 \\ &\cdots\cdots\cdots \\ e_n x_{n-1} + x_n &= b_n \end{aligned} \qquad (3)$$

(3)式を解くには１番目の式のx_1以外の未知数に近似値を代入してx_1（のさらによい近似値）を求め，２番目の式のx_2以外の未知数に近似値を代入してx_2を求める，というようにｉ番目の式のx_i以外の未知数に近似値を代入してx_iを求める，という方法を考える。

(3)式でｉ番目は，

$$e_i x_{i-1} + x_i + f_i x_{i+1} = b_i$$

より

$$x_i = b_i - e_i x_{i-1} - f_i x_{i+1}$$

である。

これをｉ＝１よりｉ＝ｎまで計算すれば，前よりよい近似値を得る。さらに精度を高め

るには，これを繰り返してやればよい。

$x_1, x_2, \cdots, x_n$ の第k近似値を，

$x_1^{(k)}, x_2^{(k)}, \quad , x_n^{(k)}$ とすれば，

第k＋1近似解を求める反復式は，

$$x_1^{(k+1)} = b_1 - f_1 x_2^{(k)}$$

$$x_2^{(k+1)} = b_2 - e_2 x_1^{(k)} - f_2 x_4^{(k)}$$

$$x_3^{(k+1)} = b_3 - e_3 x_2^{(k)} - f_3 x_4^{(k)}$$

$$----$$

$$x_n^{(k+1)} = b_n - e_n x_{n-1}^{(k)}$$

となる。

2.3.2 ヤコビ法の考察

ヤコビ法は最も簡単な繰り返し法である。そのために収束に時間がかかりすぎる。従ってヤコビ法の改良版のガウス・ザイデル法の方がすぐれている。ガウス・ザイデル法に収束をはやめる加速因子を導入したＳＯＲ法等が現在繰り返し法の主流となっている。

またＡＰＵを使っても時間が1/2程度にしかならないのは，ヤコビ法が，本質的に逐次処理解法のためである。

Givensデータで$e_i = f_j = -0.5$の時は収束しなかった。

計算速度はTable 5で示される。空欄は収束しなくて解が求められなかったことを示す。

2.4 リカーシブ・ダブリング法（ＲＤ）

2.4.1 はじめに

リカーシブ・ダブリング法は，H.S.Stone により発表された[3]。この方法は，三重対角方程式(1)の並列計算解法である。特にＩＬＬＩＡＣⅣ 用に作成された。

2.4.2 アルゴリズム

(1) ＬＵ分解

三重対角方程式(1)を解く。リカーシブ・ダブリング・アルゴリズムはＬＵ分解に基づいている。つまり，

$$LU = A$$

ここで，Ｌは対角成分が1のlower bidiagonal行列，Ｕはupper bidiagonal行列であるとする。

すなわち

$$L=\begin{pmatrix} 1 & & & \\ m_2 & 1 & 0 & \\ & m_3 & 1 & \\ & & & \\ 0 & & m_n & 1 \end{pmatrix}$$

$$U=\begin{pmatrix} u_1 f_1 & & & \\ & u_2 f_2 & & 0 \\ & & u_3 f_3 & \\ & 0 & & \\ & & & u_n \end{pmatrix}$$

以上より

$$u_1=d_1,\ u_i=d_i-(e_i f_{i-1}/u_{i-1})\quad(i>1) \tag{4}$$

$$\begin{aligned} m_2&=e_2/d_1 \\ &=e_i/(d_{i-1}-f_{i-2}\cdot m_{i-1}) \quad (i>2) \\ &=e_i/u_{i-1} \quad (i\geq 2) \end{aligned} \tag{5}$$

ＬＵ分解の後は，次の２つのことを行う。

$y=Ux$とおくと

$$Ax=LUx=Ly=b$$

$Ly=b$ を解く。

$$y_1=b_1,\ y_i=b_i-m_i y_{i-1}\quad(2\leq i\leq N) \tag{6}$$

$Ux=y$を解く。

$$x_N=y_N/u_N,\ x_i=(y_i-x_{i+1}f_i)\,U_i \tag{7}$$

(2)　前進消去と後退代入

(6)式より

$$y_1=b_1,\ y_i=b_i+(-m_i)y_{i-1}\quad(2\leq i\leq n)$$

$$\Rightarrow$$

$$y_i=\sum_{j=1}^{i} b \prod_{k=j+1}^{i}(-m_k)\quad(1\leq i\leq n) \tag{7}$$

ここでy_{2i}をy_iの関数で表わすことを考える。

はじめに次の関数$Y_i(j)$を導入する。

$$Y_{i+1}(j)=Y_1(j)+Y_i(j-1)\cdot(-m_j)\quad(i,\ j\geq 1)$$

$$Y_1(j)=b_j \quad (j\geq 1),\ Y_i(j)=0 \quad (j\leq 0),\ Y_i(j)=0 \quad (i\leq 0).$$

そうすれば

$$Y_{i+s}(j)=Y_s(j)+Y_i(j-s)\prod_{k=j-s+1}^{i}(-m_k) \qquad (i\geq 1,\ j\geq s) \tag{8}$$

$$Y_i(j)=\sum_{k=1}^{j}Y_1(k)\prod_{s=k+1}^{j}(-m_s) \qquad (i\geq j\geq 1) \tag{9}$$

$$i\geq j\geq 1,\ Y_i(j)=y_i \tag{10}$$

が成立する。

(7)式より，

$$Y_{2i}(j)=Y_i(j)+Y_i(j-i)\cdot\prod_{k=j-i+1}^{i}(-m_k) \qquad (i,\ j\geq 1) \tag{11}$$

ここで，$M_i(j)=\prod_{k=j-i+1}^{i}(-m_k) \quad (j\geq i)$

$$=\prod_{k=1}^{i}(-m_k) \quad (j<i) \tag{12}$$

とすれば，

(11)式に(12)式を代入すれば，

$$Y_{2i}(j)=Y_i(j)+Y_i(j-i)\cdot M_i(j),\quad (i,\ j\geq 1)$$

$$M_{2i}(j)=M_i(j)\cdot M_i(j-i),\quad (i,\ j\geq 1)$$

$$M_1(j)=-m_j \qquad (i\geq 1),$$

$$M_i(j)=1 \qquad (i\leq 0),$$

$$M_i(j)=1 \qquad (j\leq 0). \tag{13}$$

すなわち前進消去(7)式が(13)式に変換でき，これが求めるべきリカーシブ・ダブリングの形となる（Fig.7）。すなわち$Y_2(j)$, $Y_4(j)$, $Y_8(j)$の順に計算を行う。ここで$Y_2(2)$, $Y_2(3)\cdots Y_2(8)$は並列計算，$Y_4(3)\sim Y_4(8)$，$Y_8(5)\sim Y_8(8)$ は並列計算される。後退代入も前進消去と同様に計算できる。

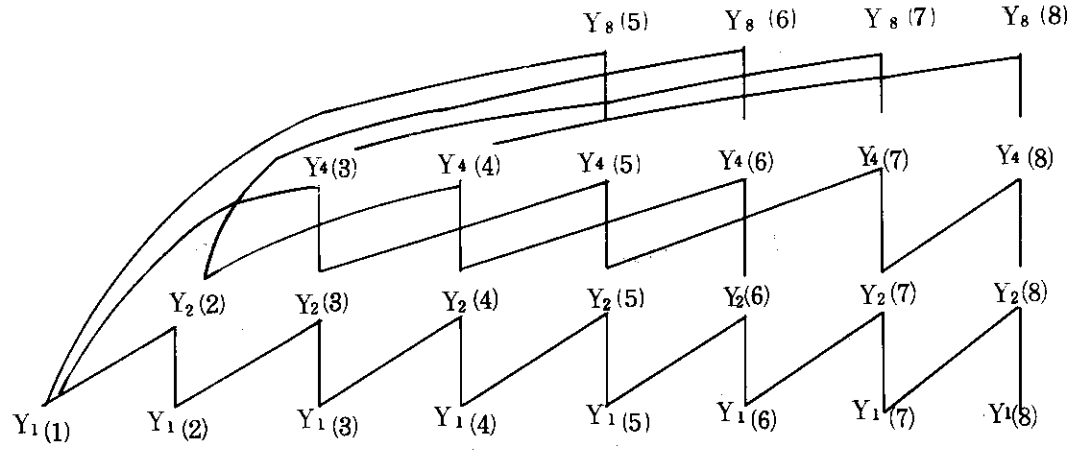


Fig.7 Relation between the values of Yi, when N=8

(3) LU分解のリカーシブ・ダブリング形

(1), (2)式をどのように並列に解くかを考える。

はじめに次の形の式を考えると,

$$q_i = d_i \cdot q_{i-1} - e_i f_{i-1} q_{i-2} \quad (i \geq 2)$$
$$q_0 = 1, \quad q_1 = d_1 \quad , \qquad (14)$$

$$u = q / q \quad (i \geq 1)$$

すなわち(1)式をパラレルに解くには(14)式を並列に解ければよい。

ここで次の関数を導入する。

$$Q_{i+s}(j) = Q_s(j) Q_j(j-s) - e_{j-s+1} f_{j-s} Q_{s-1}(j) \times Q_{i-1}(j-s-1), \quad (j \geq s, \ i \geq 1)$$

ここで $Q_1(j) = d_j \quad (j \geq 1)$, $Q_i(j) = 1 \quad (j \geq 0, \ i \leq 0)$,

$Q_i(j) = 1 \quad (j \leq 0, \ i \geq 0)$, $e_{j+1} f_j = 0 \quad (j \leq 0)$. (15)

ここで $Q_i(j)$は

$i \geq j \geq 1$ $Q_i(j) = q_i$である。

従って(14)式のリカーシブ・ダブリングは次の形となる。

$$Q_{2i}(j) = d_i Q_{2i-1}(j-1) + (-e_j f_{j-1}) Q_{2i-2}(j-2),$$
$$Q_{2i-1}(j) = Q_i(j) Q_{i-1}(j-i) + (-e_{j-i+1} f_{j-i}) \times Q_{i-1}(j) Q_{i-2}(j-i-1),$$
$$Q_{2i-2}(j) = Q_{i-1}(j) Q_{i-1}(j-i+1) + (-e_{j-i+2} f_{j-i+1}) Q_{i-2}(j) Q_{i-2}(j-i) \qquad (16)$$

ここでは, i に関して並列計算がなされる。

2.4.3 ストーンの方法の問題点と考察

計算速度についてはTable 6で示される。

(1) オーバフロー, アンダーフロー

リカーシブ・ダブリングはILLIAC Ⅳ用に特に作られたものである。ILLIAC Ⅳの浮動小数点の表示は64ビット(符号1ビット, 指数部15ビット, 仮数部48ビット)であり, F230-75 APUは36ビット(符号1ビット, 指数部9ビット, 仮数部26ビット)である。LU分解部分では$q_i = d_i q_{i-1} - e_i f_{i-1} q_{i-2}$を計算している。例えば, $d_i = 1$, $e_i = f_i = 0.3$ ($1 \leq i \leq N$)では$q_i = q_{i-1} - 0.09 q_{i-2}$を解くことになる。($q_0 = q_1 = 1$)すると$q_n = 9^n - 1 / (8 \cdot 10^{n-1})$となり,

$$q_{4096} = \frac{9^{4096} - 1}{8 \cdot 10^{4096}} \ll 10^{-77}$$

である。すなわちアンダーフローを起こす。$q_0 = q_1 = 10^{77}$としてもアンダーフローとなる。

ILLIAC Ⅳは指数部15ビットであるので今回のデーターではアンダーフローを起こさない。Table 6で空白は，アンダーフロー，オーバーフローの為にF230-75APUで行なえなかった部分である。

またGivensデータで$e_i = f_i = -0.5$の場合と一様乱数$0.485 < e_i, f_i < 0.495$の場合には予想以上に誤差が大きくなる。

(2) Givensデータの場合

LU分解で$q_0 = q_1 = 1$

$$q_i = \frac{2i+1}{6 \cdot 2^{i-2}} \qquad (2 \leq i \leq N-1)$$

$$q_N = \frac{2(N-1)+1}{6 \cdot 2^{N-3}} - \frac{1}{2} \frac{2(N-2)+1}{6 \cdot 2^{N-4}} = \frac{2}{6 \cdot 2^{N-3}}$$

ここで $U_i = q_i / q_{i-1}$で　$i = N$のとき

$U_N = q_N / q_{N-1}$で誤差が生じるように思われる。

$$U_N = \frac{2}{6 \cdot 2^{N-3}} \Big/ \frac{2(N-1)+1}{6 \cdot 2^{N-3}} = \frac{2}{2N+1}$$

なのでNが大きくなればU_Nの誤差が大きくなる。

精度の問題については，今回は考察を行わなかった。ただGivensのデータ，$e_i = f_i = -0.5$での誤差が$a \times 10^{-2}$ $(0 \leq a \leq 1)$であり，$|e_i| + |f_i| \ll |d_i|$では誤差が$a \times 10^{-6}$ $(0 \leq a \leq 1)$である。

(3) パイプライン計算機における処理速度

参考文献[7]には，パイプライン向けのmodified recursive doubling algorithmが示されている。今回では改良版アルゴリズムで試すことはしなかった。

思った程の時間改善が行えなかったが，

これは，①対角成分（$d_i = 1$, $1 \leq i \leq N$）が1であることが生かせなかった。

②パイプライン向きのアルゴリズムを使わなかった。

の2点のためであるだろう。

本数値実験で行なったストーンの方法[3]はプロセッサ・アレイ型計算機ILLIAC Ⅳ用に作られたものであり，パイプライン計算機向きのアルゴリズム[7]では行わなかった。APUはパイプライン方式なのでアレイ向きのアルゴリズムでは少し効率が落ちる。

ストーンの方法でプロセッサ・アレイ型並列計算機[3]を使っての計算時間は$\log_2 N$（Nは行列の大きさ）に比例することが知られている。

N	F75APU Time	理論値の時間
$2^6 = 64$	1	T
$2^7 = 128$	2	7／6T
$2^8 = 256$	5	8／6T
$2^9 = 512$	9	9／6T
$2^{10} = 1024$	20	10／6T
$2^{11} = 2048$	45	11／6T

（TはN＝６４の時の計算時間）

Ｆ７５ＡＰＵではＮ＝６４の時計算時間が１，Ｎ＝１２８の時計算時間が２のように，Ｎの大きさが２倍になると時間のほぼ２倍になっている。

理論値は理想化された計算機を使っての時間なので並列計算のためのオーバーヘッドは含まれていない。本実験ではパイプライン計算機でありベクトル計算のためのオーバーヘッドが，かなりかかるために理論値を上まわる計算時間となっている（Table 6）。

2.5 サイクリック・リダクション法（ＣＲ）

2.5.1 解法概略

Cyclic odd-even reductionアルゴリズムは，三重対角行列を解くための手法として文献で示された[4]。

Ｎ×Ｎ次のブロック三重対角行列の１次方程式Ａｘ＝ｖ，すなわ

$$e_j x_{j-1} + d_j x_j + f_j x_{j+1} = v_j, \quad j = 1, \cdots, N,$$

$$N = 2^{m+1} - 1, \quad m \geqq 1, \quad e_1 = f_N = 0 \qquad (17)$$

に対し，奇数インデックスを持つ変数の消去を行うと次式を得る。

$$(-e_{2j} d_{2j}^{-1} e_{j-1}) x_{2j-2}$$

$$+ (d_{2j} - e_{2j} d_{2j-1}^{-1} f_{2j-1} - f_{2j} d_{2j+1}^{-1} e_{2j+1}) x_{2j} + (-f_{2j} d_{2j+1}^{-1} f_{2j+1}) x_{2j+2}$$

$$= v_{2j} - e_{2j} d_{2j-1}^{-1} v_{2j-1} - f_{2j} d_{2j+1}^{-1} v_{2j+1}. \qquad (18)$$

新しい(18)式は再びブロック三重対角行列でその次数はＮ／２次元である。同様の操作をくり返していく。

$A^{(o)} = A$, $x^{(0)} = x$, $v^{(0)} = v$ とおき，$A^{(i)} x^{(i)} = v^{(i)}$

を計算する。ここで$A^{(i)}$は$N_i \times N_i$ ($N_i = 2^{m-i+1} - 1$) 次元の三重対角行列である。

$\log_2 N (=m)$ 回の操作で，1ブロックから成る $A^{(m)}$ が残り，

$$A^{(m)}\ x^{(m)} = v^{(m)} \qquad (19)$$

より $x^{(m)}$ が計算される。$x = x^{(0)}$ は $x^{(m)}$ を使用して後進代入により求める。

すなわち，

$i = 0, 1, \cdots, m-1$ ， $j = 1, 2, \cdots, N_{i+1}$ に対し，

$$e_j^{(i+1)} = -e_{2j}^{(i)} \left(d_{2j-1}^{(i)}\right)^{-1} e_{2j-1}^{(i)} ,$$

$$d_j^{(i+1)} = d_{2j}^{(i)} - e_{2j}^{(i)} \left(d_{2j-1}^{(i)}\right)^{-1} f_{2j-1}^{(i)} - f_{2j}^{(j)} \left(d_{2j+1}^{(i)}\right)^{-1} e_{2j+1}^{(i)} ,$$

$$f_j^{(i+1)} = -f_{2j}^{(i)} \left(d_{2j+1}^{(i)}\right)^{-1} f_{2j+1}^{(i)} ,$$

$$x_j^{(i+1)} = x_{2j}^{(i)} ,$$

$$v_j^{(i+1)} = v_{2j}^{(i)} - e_{2j}^{(i)} \left(d_{2j-1}^{(i)}\right)^{-1} v_{2j-1}^{(i)} - f_{2j}^{(i)} \left(d_{2j+1}^{(i)}\right)^{-1} v_{2j+1}^{(i)} . \qquad (20)$$

$y^{(m)} = x^{(m)}$ とおき，次の手順で $y^{(m-1)}$, $y^{(m-2)}$, $\cdots$, $y^{(0)}$ を求める。$y^{(0)}$ が求める x である。

$$y_{2j}^{(i)} = y_j^{(i+1)} , \quad j = 1, \cdots, N_{i+1} ,$$

$$y_1^{(i)} = \left(d_1^{(i)}\right)^{-1} \left(v_1^{(i)} - f_1^{(i)}\ y_2^{(i)}\right) ,$$

$$y_{2j-1}^{(i)} = \left(d_{2j-1}^{(i)}\right)^{-1} \left(v_{2j-1}^{(i)} - e_{2j-1}^{(i)} y_{2j-1}^{(i)} - f_{2j-1}^{(i)} y_{2j-1}^{(i)}\right) , \quad j = 2, \cdots, N_{i+1} ,$$

$$y_{Ni}^{(i)} = \left(d_{Ni}^{(i)}\right)^{-1} \left(v_{Ni}^{(i)} - e_{Ni}^{(i)}\ y_{Ni-1}^{(i)}\right) . \qquad (21)$$

(20), (21)式は j について並列計算される。

解法の概略は Fig.8（m＝3 の場合の例）で示すとおり，三重対角行列の非対角要素を外側に追い出し，最後に対角行列にするというものである。

非対角要素は1回のリダクション毎に $d_j^{-1} e_j$ または $d_j^{-1} f_j$ のオーダで小さくなる。したがって，リダクションを最後まで行う（つまり対角行列となる）前に，非対角要素が誤差の範囲以下になることがある。リダクション回数(k)が $k < m$ のとき，不完全サイクリック・リダクション（imcomplete cyclic reduction ）と呼び，$k = m$ のとき完全サイクリック・リダクション（complete cyclic reduction ）と呼ぶ。リダクション回数(k)は，誤差(ε)および優対角の度合(β)によって以下のように計算される。

$$k = \max\left(0, \min\left(m, \left\lceil \log_2 \left(\log_2 \varepsilon / \log_2 \beta\right) \right\rceil\right)\right) \qquad (22)$$

ここで，

$$\beta = \max_{j=1,\,2,\,\cdots,\,N} \left(\left| d_j^{-1} e_j \right| + \left| d_j^{-1} f_j \right| \right) \qquad (23)$$

βが1に近いときは，完全サイクリック・リダクションを行った方が良い。つまりｋ＝ｍとする。

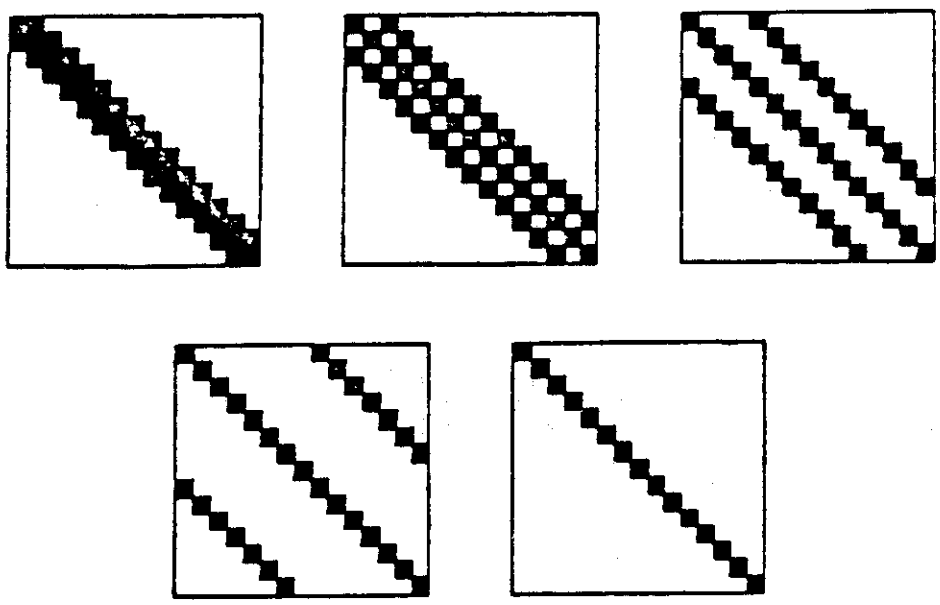


Fig.8　Cyclic reduction

2.5.2　計算結果に対する考察

数値実験の結果はTable 7で示される。

計算速度についてはデータによる差は少い。完全サイクリック・リダクションを行った場合（　印）も目立った処理時間の増加はない。M200，F75CPU，F75APU 間の速度比は1：2.3：0.3であり，F75CPUとＡＰＵでは7～8倍のスピード・アップが認められた。

計算はGivensデータの1種を除き4桁以上の精度を得た。

解法の欠点としては，前進消去時の各リダクションでの計算の途中結果を後退代入時に使用することから，（行列の次数）×（リダクション数）×3のメモリーを必要とすることが挙げられる。行列の次数が５０００の場合に約２００Ｋ語が必要となる。

2.6　チェビシェフ反復法

2.6.1　解法概要

(1)

$Ax = b$，$A = (a_{ij})$　　ｎ×ｎ実対称行列とする。　(24)

(2)

$x=(x_1, x_2, \cdots, x_n)^t$ の x_i を並べかえ

$n=2m$ のとき　$y=(\overbrace{x_1, x_3, \cdots, x_{n-1}}^{m}, \overbrace{x_2, x_4, \cdots, x_n}^{m})^t$

$n=2m+1$ のとき　$y=(\overbrace{x_1, x_3, \cdots, x_n}^{m+1}, \overbrace{x_2, x_4, \cdots, x_{n-1}}^{m})^t$ (25)

とする。

(3)

並びかえた式は，$y=(y_1, y_2)^t$ として

$$\begin{pmatrix} D_1 & B_1 \\ B_2 & D_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \quad (26)$$

と書ける。

〔例〕

$$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ & a_{32} & a_{33} & a_{34} & \\ & & a_{43} & a_{44} & a_{45} \\ & & & a_{54} & a_{55} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \\ \\ x_5 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \\ \\ b_5 \end{pmatrix} \quad \text{は，}$$

$$\left(\begin{array}{ccc|cc} a_{11} & & & a_{12} & \\ & a_{33} & & a_{32} & a_{34} \\ & & a_{55} & & a_{54} \\ \hline a_{21} & a_{23} & & a_{22} & \\ & a_{43} & a_{45} & & a_{44} \end{array}\right) \left(\begin{array}{c} x_1 \\ x_3 \\ x_5 \\ \hline x_2 \\ x_4 \end{array}\right) = \left(\begin{array}{c} b_1 \\ b_3 \\ b_5 \\ \hline b_2 \\ b \end{array}\right)$$

以下(24)式を

$$\begin{pmatrix} D_1 & B_1 \\ B_2 & D_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \text{ と書く。} \quad (27)$$

(4)

反復式

$$x_1^k=\sigma_1(v_1^k-x_1^{k-1})+x_1^{k-1} \quad \text{ここで} \quad D_1v_1^k=-B_1x_2^{k-1}+b_1$$

$$x_2^k=\sigma_2(v_2^k-x_2^{k-1})+x_2^{k-1} \quad 〃 \quad D_2v_2^k=-B_2x_1^k+b_2 \quad (28)$$

$$\sigma_{k,1} = \frac{1}{1-\frac{\rho^2}{4}\sigma_{k-1,2}} \quad , \quad \sigma_{k,2} = \frac{1}{1-\frac{\rho^2}{4}\sigma_{k,1}} \qquad (29)$$

ここにρは，$J = \begin{pmatrix} D_1 & \\ & D_2 \end{pmatrix}^{-1} \begin{pmatrix} & -B_1 \\ -B_2 & \end{pmatrix}$ のスペクトル半径で未知であり反復の過程で改善していく。

(5)

初期値

$x_1^0 = x_2^0 = 0$ or 1

$q_1 \sim q_6 = q_7 = \cdots\cdots$ 多項式の次数

$\gamma_1 \sim \gamma_7 = \gamma_8 = \cdots\cdots$ ρの上限の推定値

$\sigma_1 = \sigma_2 = 1$

(6)

$r_1^k = v_1 - x_1^{k-1}$

$r_2^k = v_2 - x_2^{k-1}$

(7)

チェビシェフ多項式

$T_0(t) = \ell$, $T_1(z) = z$, $T_{\ell+1}(z) = 2 \times z \times T_\ell(z) - T_{\ell-1}(z)$

(8) 必要なメモリーは$n^2 + 2n + c$ （nは行列長）

2.6.2 計算時間の考察

計算速度についてはTable 8－a，誤差についてはTable 8－bで示される。

(1) このChebyshev反復法はスカラー部分が多くベクトル化の効果が少ない。

(2) F75CPUとAPUとの比は1.3～1.4倍にすぎない。

(3) 今スカラー部分が2倍，ベクトルの部分が1/5倍計算時間がかかるとしたとき，プログラム構成から，スカラー部分が全体の$\frac{1}{4}$を占めるから，

$$\frac{1}{4} \times 2 + \frac{3}{4} \times \frac{1}{5} = 0.65, \quad \frac{1}{0.65} = 1.54 \quad ,$$

スカラー部　ベクトル部

即ち計算時間は0.65倍と想定される。

1.54倍が1.3～1.4倍に落ちているのはサブルーチンに飛込む回数が多いためと思われる。

(4) この解法は最適の加速パラメータの近似値が不明の場合，反復法の中で最も速いとされている。[1],[5]

(5) Aの分割が条件をみたさないとき，収束しかけてまた発散することがあるので要注意

例　反復回数　1 → 10 → 100

相対誤差　10^{-2} → 15^{-5} → 10^3

(6) 収束するときの誤差は殆んど単調減少

2.7 共役傾斜法（ＣＧ）

2.7.1 解法概要

ｎ元の三重対角線型システムＡｘ＝ｂ，すなわち，

$$\begin{pmatrix} d_1 & f_1 & & & \\ e_2 & d_2 & f_2 & & \\ & & & & f_{n-1} \\ & & & e_n & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

について考える。

ＣＧ法は共役傾斜法（method of conjugate gradients ）と呼ばれ，ガウス・ザイデル法と同様に反復法で逐次に真の解に近づいてゆく方法である。

今，第ｉ近似解ベクトル$x^{(i)}$に対する残差ベクトルを

$$r^{(i)} = b - A x^{(i)} \qquad (i = 0, 1, 2, \cdots\cdots)$$

で定義すると，ＣＧ法では

$$(r^{(i)}, r^{(j)}) = 0 \qquad (i \neq j)$$

が成り立つ。ＣＧ法はこのように残差ベクトルの直交性によって特徴づけることができる。

ｎ次のベクトル空間では互いに直交するベクトルはｎ個しか存在しないので，

ｍ≧ｎなるｍに対して，$r^{(m)} = 0$

でなければならない。したがって$x^{(m)}$が真の解であり，丸め誤差がなければ，高々ｎ回の後には解は真の値に到達することになる。

このように有限回で終結するという点ではＣＧ法はGaussの消去法と同じである。

Fig.9に三重対角線型システムの場合のＣＧ法のアルゴリズムを示す。

2.7.2 数値実験結果

ＣＧ法による数値実験結果の特徴について述べる（Table 9）。

まず計算精度については，Givens のデータを除いては，Ｍ２００，Ｆ７５ＣＰＵ，Ｆ７５ＡＰＵ共，収束判定条件$\varepsilon = 10^{-4}$でほぼ満足できる結果を得た。Givens のデータについては，ｎ＝１００の場合についてのみ有効数字２桁まで正しい値を得たが，ｎ＝５００以上については正しい解は得られなかった。

つぎに計算速度については，ＣＧ法が逐次近似法なので，他の並列化手法に比べて総じてかなり遅い。特にGivens のデータについては極端に計算時間がかかった。ＣＧ法のＦ７５ＣＰＵ，Ｍ２００，Ｆ７５ＡＰＵでの計算速度比は，Givens 以外のデータについて，ｎ＝１０００以上で，大体１：３：１０となっている。このようにＣＧ法自身はＦ７５ＡＰＵによるベクトル計算の効

果が大きいので三重対角方程式の解法以外のものに適用した場合の並列化による効果は期待できる。

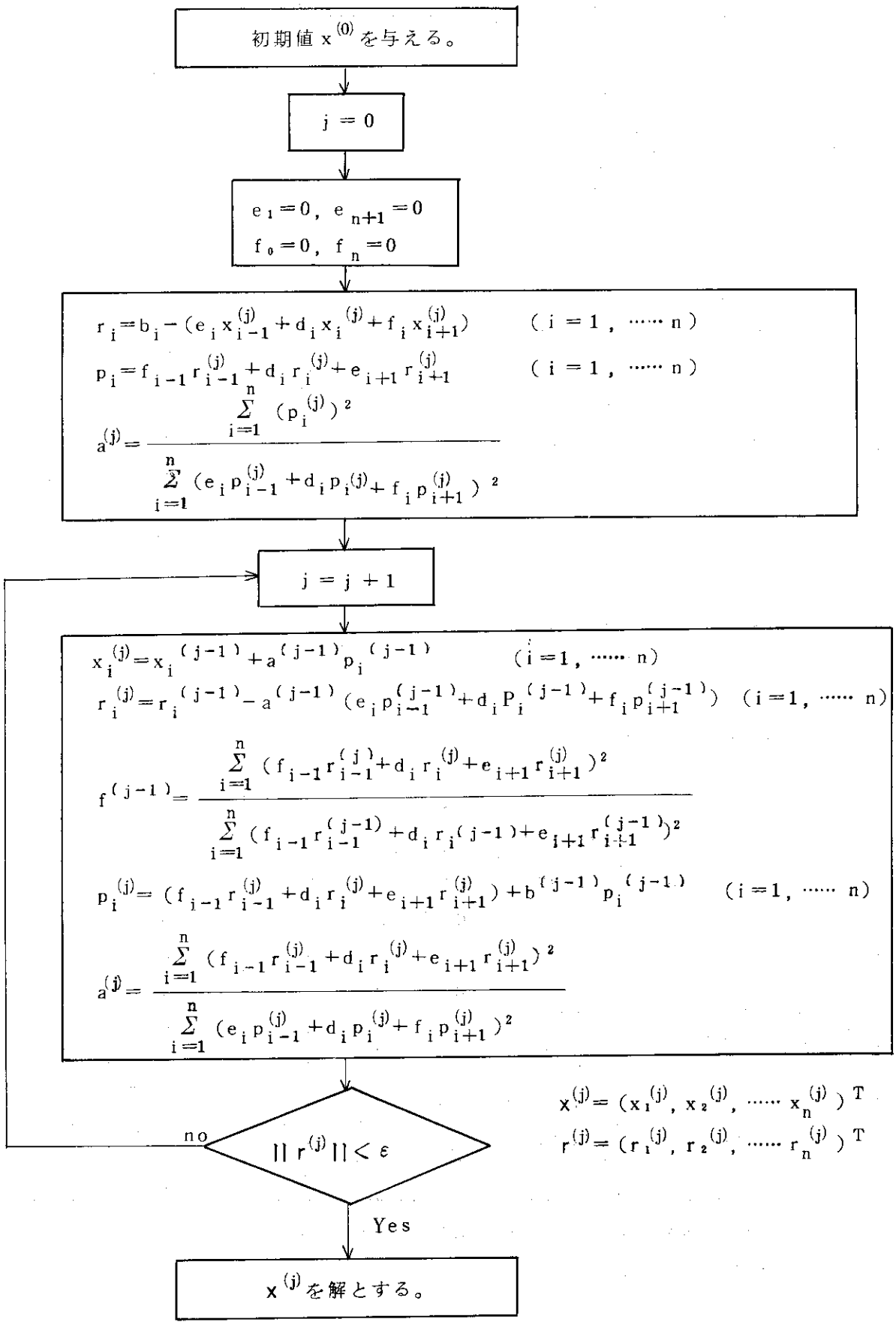


Fig. 9 CG algorithm for tridiagonal linear system

Table 2 Tridiagonal Gauss elimination method (GET)

Computing time (m sec)

input data \ matrix size	machine	100	500	1000	5000
$e_i = f_i = 0.3$ $x_i = 1.0$ for all i	M200	1	1	4	19
	F75 CPU	0.5*	4	9	47
	F75 APU	1	9	18	93
$e_i = f_i = 0.49$ $x_i = 1.0$ for all i	M200	0.5*	1	4	19
	F75 CPU	0.5*	4	9	47
	F75 APU	1	9	18	93
$e_i = -0.39 + \Delta e \cdot i$ $f_i = 0.3 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	0.5*	2	3	19
	F75 CPU	0.5*	4	9	47
	F75 APU	1	9	18	93
$e_i = -0.49 + \Delta e \cdot i$ $f_i = 0.45 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	0.5*	2	3	19
	F75 CPU	0.5*	4	9	47
	F75 APU	1	9	18	93
Givens Data $e_{i+1} = f_i = -0.5$ $x_i = 1.0$ (i : odd) 2.0 (i : even)	M200	1	1	4	19**
	F75 CPU	0.5*	4	9	47**
	F75 APU	1	9	18	93**
Givens Data $e_{i+1} = f_i = -0.4975$ $x_i = 1.0$ (i : odd) 2.0 (i : even)	M200	1	1	3	19
	F75 CPU	0.5*	4	9	48
	F75 APU	1	9	18	93
$0.2 < e_i, f_i < 0.4$ 一様乱数 $x_i = 1.0$	M200	1	2	4	19
	F75 CPU	0.5*	4	9	47
	F75 APU	1	9	18	93
$0.485 < e_i, f_i < 0.495$ 一様乱数 $x_i = 1.0$	M200	0.5*	2	4	20
	F75 CPU	0.5*	4	9	47
	F75 APU	1	9	18	93
実験データ e_i = ADCのC1係数 f_i = ADCのC2係数 $x_i = 1.0$	M200	0.5*	2	4	19
	F75 CPU	0.5*	4	9	46
	F75 APU	1	9	18	93
実験データ $e_{i+1} = f_i$ = ADCのC2係数 $x_i = 1.0$	M200	0.5*	2	4	18
	F75 CPU	0.5*	4	9	46
	F75 APU	1	9	18	93

注＊　CLOCKMによる計測では0 milli secと計測されている。

注＊＊　小数点以下2桁までしか正しく得られていない。

Table 3 Parallel Gauss method (PG)

Computing time (msec)

input data \ matrix size	machine	100	500	1000	5000
$e_i = f_i = 0.3$ $x_i = 1.0$ for all i	M200	2	8	15	78
	F75 CPU	3	19	38	211
	F75 APU	3	12	23	106
$e_i = f_i = 0.49$ $x_i = 1.0$ for all i	M200	2	7	14	78
	F75 CPU	3	19	39	211
	F75 APU	15	53	101	486
$e_i = -0.39 + \Delta e \cdot i$ $f_i = 0.3 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	2	8	14	74
	F75 CPU	3	18	37	203
	F75 APU	3	12	22	106
$e_i = -0.49 + \Delta e \cdot i$ $f_i = 0.45 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	2	7	14	77
	F75 CPU	3	19	38	210
	F75 APU	3	14	26	124
Givens Data $e_i = f_i = -0.5$ $x_i = 1.0$ (i : odd) 2.0 (i : even)	M200	2③	8①	14①	76②
	F75 CPU	3	19③	38	210①
	F75 APU	39	709②	2693②	50817②
Givens Data $e_i = f_i = -0.4975$ $x_i = 1.0$ (i : odd) 2.0 (i : even)	M200	2③	7③	15②	75③
	F75 CPU	3	19	38	210
	F75 APU	28③	104②	196②	905②
$0.2 < e_i, f_i < 0.495$ 一様乱数 $x_i = 1.0$	M200	2	8	15	76
	F75 CPU	3	19	39	210
	F75 APU	3	14	27	127
$0.485 < e_i, f_i < 0.495$ 一様乱数 $x_i = 1.0$	M200	2	7	15	78
	F75 CPU	3	19	38	209
	F75 APU	15	55	105	495
実験データ e_i = ADCのC1係数 f_i = ADCのC2係数 $x_i = 1.0$	M200	2	7	15	74
	F75 CPU	3	18	36	201
	F75 APU	2	8	16	79
実験データ $e_i = f_i$ = ADCのC2係数 $x_i = 1.0$	M200	2	8	14	75
	F75 CPU	3	18	37	201
	F75 APU	2	8	16	83

① の中の数字は収束精度で 10^{-i} を示す。

Table 4 Accelerated parallel Gauss method（APG）

Computing time （m sec）

matrix size / input data	machine	100	500	1000	5000
$e_i=f_i=0.3$ $x_i=1.0$ for all i	M200	2	11	22	119
	F75 CPU	5	26	53	391
	F75 APU	2	7	12	55
$e_i=f_i=0.49$ $x_i=1.0$ for all i	M200	6	23	47	247
	F75 CPU	12	55	109	904
	F75 APU	6	16	29	129
$e_i=-0.39+\Delta e\cdot i$ $f_i=0.3-\Delta f\cdot i$ $x_i=1.0+\Delta x\cdot i$	M200	3	10	20	105
	F75 CPU	5	25	51	382
	F75 APU	2	7	12	56
$e_i=-0.49+\Delta e\cdot i$ $f_i=0.45-\Delta f\cdot i$ $x_i=1.0+\Delta x\cdot i$	M200	3	10	21	107
	F75 CPU	5	26	52	389
	F75 APU	2	7	12	56
Givens Data $e_i=f_i=-0.5$ $x_i=1.0$ (i：odd) 2.0 (i：even)	M200	11③	238①	953①	21411⓪
	F75 CPU	26	599③	2386	94352①
	F75 APU	13	160②	565②	12232⓪
Givens Data $e_i=f_i=-0.4975$ $x_i=1.0$ (i：odd) 2.0 (i：even)	M200	8③	37③	73③	399③
	F75 CPU	20②	89②	176③	1458②
	F75 APU	10③	28③	48③	210③
$0.2<e_i,\ f_i<0.4$ 一様乱数 $x_i=1.0$	M200	2	11	21	107
	F75 CPU	5	26	53	390
	F75 APU	2	7	12	52
$0.485<e_i,\ f_i<0.495$ 一様乱数 $x_i=1.0$	M200	5	24	47	242
	F75 CPU	12	55	109	902
	F75 APU	6	16	29	128
実験データ e_i＝ADCのC1係数 f_i＝ADCのC2係数 $x_i=1.0$	M200	2	9	18	94
	F75 CPU	4	22	45	328
	F75 APU	2	6	11	50
実験データ $e_i=f_i$＝ADCのC2係数 $x_i=1.0$	M200	2	9	18	95
	F75 CPU	4	22	46	329
	F75 APU	2	6	11	50

Table 5 Jacobi method (J C)

Computing time (m sec)

input data \ matrix size	machine	100	500	1000	5000
$e_i = f_i = 0.3$ $x_i = 1.0$ for all i	M200	7	31	66	346
	F75 CPU	16	83	179	931
	F75 APU	7	32	64	324
$e_i = f_i = 0.49$ $x_i = 1.0$ for all i	M200	151	760	1531	8357
	F75 CPU	418	2043	4452	23182
	F75 APU	184	783	1532	7558
$e_i = -0.39 + \Delta e \cdot i$ $f_i = 0.3 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	5	28	55	292
	F75 CPU	14	71	156	809
	F75 APU	3	20	40	208
$e_i = -0.49 + \Delta e \cdot i$ $f_i = 0.45 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	21	104	207	1240
	F75 CPU	55	284	621	3464
	F75 APU	19	84	166	746
Givens Data $e_i = f_i = -0.5$ $x_i = 1.0$ (i : odd) 2.0 (i : even)	M200				
	F75 CPU				
	F75 APU				
Givens Data $e_i = f_i = -0.4975$ $x_i = 1.0$ (i : odd) 2.0 (i : even)	M200	544	2805	5615	31861
	F75 CPU	1610	7945	17271	89450
	F75 APU	926	3390	7722	37934
$0.2 < e_i, f_i < 0.4$ 一様乱数 $x_i = 1.0$	M200	8	41	82	450
	F75 CPU	15	81	180	990
	F75 APU	8	41	81	433
$0.485 < e_i, f_i < 0.495$ 一様乱数 $x_i = 1.0$	M200	145	765	1538	8538
	F75 CPU	332	1677	3644	19584
	F75 APU	189	830	1624	8289
実験データ e_i = ADCのC1係数 f_i = ADCのC2係数 $x_i = 1.0$	M200	2	12	24	150
	F75 CPU	5	33	71	421
	F75 APU	3	16	32	187
実験データ $e_i = f_i$ = ADCのC2係数 $x_i = 1.0$	M200	2	12	24	149
	F75 CPU	5	32	71	421
	F75 APU	3	16	32	185

空欄は収束しなくて解が求められなかったことを示す。

Table 6 Recursive doubling method (RD)

Computing time (m sec)

input data \ matrix size	machine	128	512	1024	4096
$e_i = f_i = 0.3$ $x_i = 1.0$ for all i	M200	5	32	73	
	F75 CPU	16	88	201	
	F75 APU	2	9	20	
$e_i = f_i = 0.49$ $x_i = 1.0$ for all i	M200	5	32		
	F75 CPU	16	87		
	F75 APU	2	9		
$e_i = -0.39 + \Delta e \cdot i$ $f_i = 0.3 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	5	32		
	F75 CPU	16	87		
	F75 APU	2	9		
$e_i = -0.49 + \Delta e \cdot i$ $f_i = 0.45 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	5	28	73	
	F75 CPU	16	87	199	
	F75 APU	2	9	20	
Givens Data $e_i = f_i = -0.5$ $x_i = 1.0$ (i:odd) 2.0 (i:even)	M200	5			
	F75 CPU	15			
	F75 APU	2			
Givens Data $e_i = f_i = -0.4975$ $x_i = 1.0$ (i:odd) 2.0 (i:even)	M200	5	26		
	F75 CPU	15	82		
	F75 APU	2	9		
$0.2 < e_i, f_i < 0.4$ 一様乱数 $x_i = 1.0$	M200	4	34	77	
	F75 CPU	15	82	185	
	F75 APU	2	9	20	
$0.485 < e_i, f_i < 0.495$ 一様乱数 $x_i = 1.0$	M200	5	11		
	F75 CPU	15	82		
	F75 APU	2	9		
実験データ e_i = ADCのC1係数 f_i = ADCのC2係数 $x_i = 1.0$	M200	2	25	79	346
	F75 CPU	14	78	177	926
	F75 APU	2	9	20	90
実験データ $e_i = f_i$ = ADCのC2係数 $x_i = 1.0$	M200	2	24	78	320
	F75 CPU	14	77	177	926
	F75 APU	2	9	20	91

- 空欄はLU分解でオーバフロー，アンダフローにより計算できなかったことを示す。
- 行列長を2のベキ乗のみ扱っているが，ストーンのアルゴリズムで効率が良いためである。

Table 7 Cyclic reduction Algorithm (CR)

Computing time (m sec)

input data \ matrix size	machine	100	500	1000	5000
$e_i = f_i = 0.3$ $x_i = 1.0$ for all i	M200	2	6	11	56
	F75 CPU	2	12	26	136
	F75 APU	1	2	4	17
$e_i = f_i = 0.49$ $x_i = 1.0$ for all i	(M200)	1	6	13	72
	(F75 CPU)	2	13	27	139
	(F75 APU)	1	3	4	19
$e_i = -0.39 + \Delta e \cdot i$ $f_i = 0.3 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	1	5	11	59
	F75 CPU	2	12	26	135
	F75 APU	1	2	4	16
$e_i = -0.49 + \Delta e \cdot i$ $f_i = 0.45 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	2	5	11	59
	F75 CPU	2	12	26	135
	F75 APU	1	2	4	16
Givens Data $e_i = f_i = -0.5$ $x_i = 1.0$ (i:odd) 2.0 (i:even)	(M200)	1	6	11	60
	(F75 CPU)	2	12	26	134
	(F75 APU)	1	3	4	19
Givens Data $e_i = f_i = -0.4975$ $x_i = 1.0$ (i:odd) 2.0 (i:even)	(M200)	1 ($\varepsilon = 10^{-4} \times 2$)	6 ($\varepsilon = 10^{-4} \times 2$)	11 ($\varepsilon = 10^{-4} \times 2$)	68 ($\varepsilon = 10^{-4} \times 2$)
	(F75 CPU)	2	13	27	140
	(F75 APU)	1	3	4	18
$0.2 < e_i, f_i < 0.4$ 一様乱数 $x_i = 1.0$	M200	1	5	12	57
	F75 CPU	2	12	24	125
	F75 APU	1	2	3	17
$0.485 < e_i, f_i < 0.495$ 一様乱数 $x_i = 1.0$	(M200)	2	6	12	66
	(F75 CPU)	2	12	25	132
	(F75 APU)	1	3	4	19
実験データ e_i = ADCのC1係数 f_i = ADCのC2係数 $x_i = 1.0$	M200	2	6	11	60
	F75 CPU	2	11	23	124
	F75 APU	1	2	3	16
実験データ $e_i = f_i$ = ADCのC2係数 $x_i = 1.0$	M200	1	5	11	60
	F75 CPU	2	11	23	123
	F75 APU	1	2	3	15

(　) Complete cyclic reduction

他のもの Imcomplete cyclic reduction

ε は相対誤差で，$\varepsilon > 10^{-4}$ についてのみ記す。

Table 8.a Cyclic Chebyshev polynomial iteration method (CH)

Computing time (m sec)

input data \ matrix size	machine	100		500		1000		5000	
		t	ms	t	ms	t	ms	t	ms
$e_i = f_i = 0.3$ $x_i = 1.0$ for all i	M200	8	8	8	33	8	65	8	316
	F75 CPU	8	22	8	91	8	178	8	870
	F75 APU	8	19	8	80	8	156	8	758
$e_i = f_i = 0.49$ $x_i = 1.0$ for all i	M200	50	57	48	209	48	402	47	1987
	F75 CPU	50	159	50	619	49	1158	49	5595
	F75 APU	50	131	50	511	49	974	49	4728
$e_i = -0.39 + \Delta e \cdot i$ $f_i = 0.3 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200								
	F75 CPU								
	F75 APU								
$e_i = -0.49 + \Delta e \cdot i$ $f_i = 0.45 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200								
	F75 CPU								
	F75 APU								
Givens Data $e_i = f_i = -0.5$ $x_i = 1.0$ (i:odd) 2.0 (i:even)	M200	50	57	50	219	50	424	50	2073
	F75 CPU	50	157	50	595	50	1120	50	5446
	F75 APU	50	130	50	510	50	992	50	4810
Givens Data $e_i = f_i = -0.4975$ $x_i = 1.0$ (i:odd) 2.0 (i:even)	M200	50	57	50	217	50	418	50	2070
	F75 CPU	50	158	50	614	50	1181	50	5677
	F75 APU	50	130	50	510	50	992	50	4812
$0.2 < e_i, f_i < 0.4$ 一様乱数 $x_i = 1.0$	M200	12	12	12	50	12	98	14	571
	F75 CPU	12	36	12	141	12	276	13	1450
	F75 APU	12	29	12	120	12	235	13	1242
$0.485 < e_i, f_i < 0.495$ 一様乱数 $x_i = 1.0$	M200								
	F75 CPU								
	F75 APU								
実験データ e_i=ADCのC1係数 f_i=ACCのC2係数 $x_i = 1.0$	M200								
	F75 CPU								
	F75 APU								
実験データ $e_{i+1} = f_i$=ADCのC2係数 $x_i = 1.0$	M200	5	5	6	23	6	48	7	270
	F75 CPU	5	12	6	66	6	131	7	759
	F75 APU	5	11	6	58	6	116	7	664

t 反復回数

Table 8.b Error of the cyclic Chebyshiv polynomial iteration method When not converged

$$\max_i \left| \frac{x_i - e_i}{e_i} \right|$$

input data \ matrix size	machine	100	500	1000	5000
$e_i = f_i = 0.3$ $x_i = 1.0$ for all i	M200				
	F75 CPU				
	F75 APU				
$e_i = f_i = 0.49$ $x_i = 1.0$ for all i	M200	7.6×10^{-6}			
	F75 CPU	2.3×10^{-5}	7.5×10^{-6}		
	F75 APU	2.4×10^{-5}	7.7×10^{-6}		
$e_i = -0.39 + \Delta e \cdot i$ $f_i = 0.3 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200				
	F75 CPU				
	F75 APU				
$e_i = -0.49 + \Delta e \cdot i$ $f_i = 0.45 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200				
	F75 CPU				
	F75 APU				
Givens Data $e_{i+1} = f_i = -0.5$ $x_i = 1.0$ (i : odd) 2.0 (i : even)	M200	1.9×10^{0}	2.0×10^{0}	2.0×10^{0}	2.0×10^{0}
	F75 CPU	〃 〃	〃 〃	〃 〃	〃 〃
	F75 APU	〃 〃	〃 〃	〃 〃	〃 〃
Givens Data $e_{i+1} = f_i = -0.4975$ $x_i = 1.0$ (i : odd) 2.0 (i : even)	M200	1.2×10^{-1}	1.1×10^{-1}	1.1×10^{-1}	9.7×10^{-2}
	F75 CPU	〃 〃	〃 〃	〃 〃	〃 〃
	F75 APU	〃 〃	〃 〃	〃 〃	〃 〃

n＝8ぐらいのときは$\varepsilon = 10^{-5}$で収束，n＝100では2よりわずかに減少，n≧500ではこのまゝ減らない。

Table 9 Conjugate gradient method (CG)

Computing time (m sec)

matrix size / input data	machine	100	500	1000	5000
$e_i = f_i = 0.3$ $x_i = 1.0$ for all i	M200	14(16)	72(16)	146(16)	826(16)
	F75 CPU	43(16)	235(16)	488(16)	2,496(16)
	F75 APU	9(16)	26(16)	49(16)	212(16)
$e_i = f_i = 0.49$ $x_i = 1.0$ for all i	M200	83(93)	568(132)	1,162(132)	6,513(132)
	F75 CPU	210(79)	1,877(132)	3,929(132)	20,110(132)
	F75 APU	46(82)	213(132)	388(132)	1,684(132)
$e_i = -0.39 + \Delta e \cdot i$ $f_i = 0.3 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	6(6)	29(6)	58(6)	327(6)
	F75 CPU	17(6)	91(6)	190(6)	973(6)
	F75 APU	3(6)	10(6)	19(6)	82(6)
$e_i = -0.49 + \Delta e \cdot i$ $f_i = 0.45 - \Delta f \cdot i$ $x_i = 1.0 + \Delta x \cdot i$	M200	6(6)	29(6)	65(7)	375(7)
	F75 CPU	17(6)	91(6)	219(7)	1,124(7)
	F75 APU	3(6)	10(6)	22(7)	95(7)
Givens Data $e_i = f_i = -0.5$ $x_i = 1.0$ (i:odd) 2.0 (i:even)	M200	1,056(1,171)	187,799(42,498)	716,885(81,185)	
	F75 CPU	1,837(683)	239,258(16,518)	1,283,564 (42,320)	
	F75 APU	400(715)	24,865(15,450)	121,246(41,386)	
Givens Data $e_i = f_i = -0.4975$ $x_i = 1.0$ (i:odd) 2.0 (i:even)	M200	476(519)	6,332(1,397)	13,235(1,469)	77,679(1,516)
	F75 CPU	1,092(412)	18,591(1,320)	42,756(1,451)	224,760(1,483)
	F75 APU	221(395)	2,236(1,328)	4,260(1,453)	18,794(1,483)
$0.2 < e_i, f_i < 0.4$ 一様乱数 $x_i = 1.0$	M200	19(20)	106(23)	234(25)	1,406(28)
	F75 CPU	51(19)	340(24)	715(24)	4,117(27)
	F75 APU	11(19)	39(24)	72(24)	350(27)
$0.485 < e_i, f_i < 0.495$ 一様乱数 $x_i = 1.0$	M200	115(128)	786(178)	1,763(199)	11,262(231)
	F75 CPU	317(118)	2,564(181)	5,962(202)	35,258(234)
	F75 APU	67(118)	293(181)	594(202)	3,027(234)
実験データ e_i = ADCのC1係数 f_i = ADCのC2係数 $x_i = 1.0$	M200	9(8)	44(9)	97(10)	664(12)
	F75 CPU	22(8)	132(9)	306(10)	1,864(12)
	F75 APU	5(8)	15(9)	31(10)	158(12)
実験データ $e_i = f_i$ = ADCのC2係数 $x_i = 1.0$	M200	9(8)	49(10)	101(10)	642(12)
	F75 CPU	22(8)	145(10)	304(10)	1,866(12)
	F75 APU	5(8)	17(10)	31(10)	161(12)

(　　) は反復回数

3. 数値実験結果の分析

各解法の核となる部分の計算時間を比較してみると，Fig.6に見られるようにサイクリック・リダクション法，加速パラレル・ガウス法，三重対角用ガウスの消去法，パラレル・ガウス法の４つは，次数５０００のケースにおいて，F75APUで0.2秒以内と速い。

各解法の計算時間は，Fig.6に見られるように次数nに比例している。

並列計算（F75APU）と逐次計算（F75CPU）の計算時間を比較してみると，Table１０に見られるようにサイクリック・リダクション法，加速パラレル・ガウス法，リカーシブ・ダブリング法，共役傾斜法の４つの解法は，並列化の効果がよく現われている。三重対角用の並列計算解法として開発されたリカーシブ・ダブリング法や加速パラレル・ガウス法は，逐次計算解法のガウスの消去法ほどは速くなかった。

反復解法の計算時間と精度は，優対角の度合に依存性が見られ，Givensのデータのように特に優対角の度合の弱い場合は，精度が落ちるがサイクリック・リダクションとガウスの消去法が使用に耐える（Table１１）。

反復解法に要する計算時間は，データの優対角の度合に依存する。加速パラレル・ガウス法に対する計算時間と反復回数についての結果をTable１２で示す。

全体として，計算速度，精度の面でサイクリック・リダクション法と逐次解法ではあるがガウスの消去法が勝れている。

Table１０ Comparison of the computing times

解　法	並列計算（ミリ秒）	逐次計算（ミリ秒）	$\frac{\text{F75CPU}}{\text{F75APU}}$
CR	3	24	6.6
APG	12	53	4.0
GET	18	9	0.5
RD	20	182	9.5
PG	27	39	1.5
CG	72	715	10.0
JC	81	180	2.5
CH	235	276	1.2

次数：N＝１０００

乱数データ　$0.2 < e_i, f_i < 0.4$

Table 11 Computing accuracy for Givens data

解法	Givens data $e_i, f_i = 0.5$	Givens data $e_i, f_i = 0.4975$
CR	○	2.0×10^{-4}
APG	1.0×10^{-2}	1.0×10^{-3}
GET	4.0×10^{-3}	○
RD	×	×
PG	1.0×10^{-2}	1.0×10^{-2}
CG	×	
JC	×	○
CH	2.0×10^{0}	1.1×10^{-1}

N＝1000, F75APU

Table 12 Speed of iteration method

	優対角の度合 e_i, f_i	加速パラレル・ガウス法	
		計算時間	反復回数
強	0.15～0.2	11	3
	0.3	12	4
	0.2～0.4	12	4
	0.485～0.495	29	10
	0.4975	48	18
弱	0.5	565	201
		（ミリ秒）	（回）

N＝1000

4. 結び

1. 並列解法はサイクリック・リダクション法が有利である。
   逐次解法はガウスの消去法が有利である。
2. 共役傾斜法の並列化は有効である。
3. リカーシブ・ダブリング法，加速パラレル・ガウス法はガウスの消去法より遅い。
4. 反復解法（APG, PG, JC, CH, CG）の計算時間は優対角性に依存する。

Table 11 Computing accuracy for Givens data

解　　法	Givens data $e_i, f_i = 0.5$	Givens data $e_i, f_i = 0.4975$
CR	○	2.0×10^{-4}
APG	1.0×10^{-2}	1.0×10^{-3}
GET	4.0×10^{-3}	○
RD	×	×
PG	1.0×10^{-2}	1.0×10^{-2}
CG	×	
JC	×	○
CH	2.0×10^{0}	1.1×10^{-1}

N＝1000, F75APU

Table 12 Speed of iteration method

	優対角の度合 e_i, f_i	加速パラレル・ガウス法	
		計算時間	反復回数
強	0.15～0.2	11	3
	0.3	12	4
	0.2～0.4	12	4
	0.485～0.495	29	10
	0.4975	48	18
弱	0.5	565	201
		（ミリ秒）	（回）

N＝1000

4. 結　　び

1. 並列解法はサイクリック・リダクション法が有利である。
   逐次解法はガウスの消去法が有利である。
2. 共役傾斜法の並列化は有効である。
3. リカーシブ・ダブリング法，加速パラレル・ガウス法はガウスの消去法より遅い。
4. 反復解法（APG, PG, JC, CH, CG）の計算時間は優対角性に依存する。

謝　　辞

三重対角方程式の並列解法については，計算センターのセミナで最近の手法について勉強する機会を得て，数値実験を行うに至った。本報告の6名の著者に加えて，茨城大学の儀田和男氏と，計算センター室長代理の浅井清氏に対してセミナにおける有益な討論に感謝します。

参　考　文　献

[1] 戸川隼人：マトリックスの数値計算，オーム社（1971）

[2] Heller, D.E., Stevenson, D.K., Traub, J.F. : Accelerated Iterative Methods for the Solution of Tridiagonal Systems on Parallel Computers, JACM Vol.23, No.4, pp.636－654（1976）

[3] Stone, H.S. : An Efficent Parallel Algorithm for the Solution of a Tridiagonal System of Equations, JACM, Vol.20, No.1, pp.27－38（1973）

[4] Heller, D.E. : Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems, SIAM J.Numr.Anal., Vol.13, No.4, pp.484－496（1976）

[5] Hageman, L.A. : The Estimation of Acceleration Parameters for the Chebyshev Polynomial and the Succesive Overrelaxation Iterative Methods, WAPD－TM－1038,（1972）

[6] 森口繁一，高田勝：数値計算法，岩波書店，（1958）

[7] Kogge, P.M. and Stone, H.S. : A Parallel Algorithm for the Efficient Solution of a General Class of Recursive Equations, IEEE Trans. Computers, Vol C－22, No.8（1973）

謝　　辞

三重対角方程式の並列解法については，計算センターのセミナで最近の手法について勉強する機会を得て，数値実験を行うに至った。本報告の６名の著者に加えて，茨城大学の磯田和男氏と，計算センター室長代理の浅井清氏に対してセミナにおける有益な討論に感謝します。

参　考　文　献

[1] 戸川隼人：マトリックスの数値計算，オーム社（1971）

[2] Heller, D.E., Stevenson, D.K., Traub, J.F. : Accelerated Iterative Methods for the Solution of Tridiagonal Systems on Parallel Computers, JACM Vol.23, No.4, pp.636－654（1976）

[3] Stone, H.S.: An Efficent Parallel Algorithm for the Solution of a Tridiagonal System of Equations, JACM, Vol.20, No.1, pp.27－38（1973）

[4] Heller, D.E.: Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems, SIAM J.Numr.Anal., Vol.13, No.4, pp.484－496（1976）

[5] Hageman, L.A.: The Estimation of Acceleration Parameters for the Chebyshev Polynomial and the Succesive Overrelaxation Iterative Methods, WAPD－TM－1038,（1972）

[6] 森口繁一，高田勝：数値計算法，岩波書店，（1958）

[7] Kogge, P.M. and Stone, H.S.: A Parallel Algorithm for the Efficient Solution of a General Class of Recursive Equations, IEEE Trans. Computers, Vol C－22, No.8（1973）