社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－HPC－95 (5)
2003／8／4

Ajiz-Jenningsによる不完全分解前処理の改良

柿原正伸† 藤野 清次 ††

共役勾配法に対する前処理には通常の不完全コレスキー分解が広く用いられてきた．しかし，この不完全分解もあらゆる分野の問題に対して万能ではなく，苦手とする分野の問題も存在することがだんだんと知られてきた．本発表で扱う M. Ajiz と A. Jennings により考案された不完全コレスキー分解は，どんな行列に対しても安定して分解できることが保証されたロバストな分解法の1つである．本論文では，このロバストな不完全コレスキー分解の要素の棄却について着目し，それにある工夫を加えた改良版を提案する．その処理により，共役勾配法の反復における前処理行列の演算量を減少させ，全体の計算時間を減らすことができる．

An Improvement of Incomplete Factorization Preconditioner devised by Ajiz and Jennings

MASANOBU KAKIHARA † and SEIJI FUJINO ††

Standard Incomplete Cholesky factorization CG method has been widely applied to solution of a variety of realistic problems. The extension of this standard ICCG method led to the development of Robust Incomplete Cholesky factorization by M. Ajiz and A. Jennings. However, discarding part of the fill-in in the course of factorization process has room for improvement. In this article, a certain device for reduction of operations needed in the computation of preconditioner is described. Through numerical experiments effectiveness of this idea will be verified.

1. はじめに

前処理付き共役勾配 (Preconditioned Conjugagte Gradient: 以下，PCG と略す) 法は，偏微分方程式を有限要素法や有限差分法で離散化して得られる連立一次方程式が係数行列として大規模で疎な正値対称行列を行列を持つ場合の数値解法としてよく使用される．特に，流体や熱の解析の分野で生じる問題に対しては，fill-in を考慮しない不完全コレスキー分解つき CG 法が広く用いられる．しかし，構造解析や固体力学の分野で得られる H-行列でない行列を持つ問題に対しては，分解の途中で対角要素の値が負になり，分解過程の途中から分解が不可能になることが起こることが知られている.[3] これに対して，1984 年 Ajiz と Jennings により考案されたロバスト不完全コレスキー分解[1] は分解過程で発生する fill-in の影響を考慮し，すなわち要素を棄却するごとに対角要素に修正を施し，それによって対角要素の値が負になることを防ぐ斬新な分解の方法であった．このため，この分解法はどんな問題に対しても分解が途中で破綻することなく前処理行列を計算できる方法である．この事実は多くの数値実験で検証され，この不完全コレスキー分解はロバスト不完全コレスキー (Robust Incomplete Cholesky : 以下，**RIC** と略す) 分解と呼ばれてきた．

この他にも同様の不完全分解はいくつも提案されてきた．特に，I. Kaporin による不完全分解[4] が最近注目を集めている．この分解法は完全コレスキー分解に近く，より厳密な分解を行うことで CG 法が収束するまでの反復回数を少なくし，計算時間を短縮させるという戦略に基づいた分解法である．ここでも，分解の計算コストを抑えるために，上で述べた RIC 分解による要素の棄却と同じ方法で棄却を行なっている．そこで，本研究では，RIC 分解の計算が終了した後，さらに分解で得た要素を棄却することによって，CG 法の反復計算にかかる計算量を抑えるという戦略をとることにする．この考え方は，すでに M. Benzi と M. Tuma らによって提案されているが，まだ十分な議論と検証がされていないように思われる[2].

本論文では，この改良版の分解を説明する前に第 2 節で PCG 法の算法に簡単に触れる．第 3.1 小節で PCG 法で用いられる前処理行列を RIC 分解により計算する方法を述べ，さらに，第 3.2 小節では RIC 分

† 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†† 九州大学情報基盤センター
Computing and Communications Center, Kyushu University

解の重要な性質であるロバスト性について説明する．また，第3.3小節において，本論文の中心となるRIC分解の改良の仕方について述べる．第4節で前節で述べたRIC分解を実際にプログラムに実装する方法について述べる．最後に，第5節で数値実験した結果を報告し，RIC分解の改良版の有効性を検証する．

2. 前処理つき共役勾配法

本論文では，対称正定値行列を係数行列に持つ次の連立一次方程式を扱う．

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

ここで，A は大きさ $n \times n$ の正方行列，$\boldsymbol{x}$，$\boldsymbol{b}$ は大きさ n の解および右辺ベクトルとする．この連立一次方程式の解法として，PCG法を用いる．PCG法は方程式（1）の両辺に**前処理行列**と呼ばれる適当な正方行列 M^{-1} を掛け，次の連立一次方程式の求解に

$$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b} \tag{2}$$

に共役勾配法を適用する方法である．PCG法の算法は以下のように表される．

[PCG 法の算法]

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{p}_0 = M^{-1}\boldsymbol{r}_0$
for $k = 0, 1, \cdots$
　$\alpha_k = (\boldsymbol{r}_k, M^{-1}\boldsymbol{r}_k)/(\boldsymbol{p}_k, A\boldsymbol{p}_k)$
　$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
　$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$
　if $||\boldsymbol{r}_{k+1}||_2/||\boldsymbol{r}_0||_2 \leq \varepsilon$ **stop**
　$\beta_k = (\boldsymbol{r}_{k+1}, M^{-1}\boldsymbol{r}_{k+1})/(\boldsymbol{r}_k, M^{-1}\boldsymbol{r}_k)$
　$\boldsymbol{p}_{k+1} = M^{-1}\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$
end for

ここで，$\boldsymbol{x}_0$ は初期近似解，ε は収束判定用の微小な値である．PCG法における前処理行列 M の選び方により，CG法の収束性が大きく異なることがよく知られている．そこで，次の節では，どんな問題に対しても安定して前処理行列 M を求めることができるRIC分解について記述する．

3. RIC 分解

3.1 RIC 分解のアルゴリズム

RIC分解は，係数行列 A を

$$A = U^T U - R - R^T - D \tag{3}$$

と分解する方法である．ここで，行列 U は上三角行列，行列 U^T，R^T の上付き添え字 T は転置（Transpose）を表す．PCG法の前処理行列として用いる行列はこの上三角行列 U とその行列の転置行列 U^T の積である行列 $U^T U$ を用いる．行列 R は U の要素を分解中に棄却する（行列 U の値を0にする）ことを形式的に表す上三角行列である．行列 D は U の要素が棄却されたとき，行列 A の対角要素を修正することを表す対角行列である．行列 R と行列 D の具体的な行列は第3.2小節で示す．RIC分解の手順は次のように表せる．

(A)	非対角要素に対するコレスキー分解の計算を行う．
(B)	行列 R により，(A) で得られた要素の中から上三角行列 U に残す要素の個数を減らす．
(C)	要素が棄却された場合，行列 D により，棄却された要素に対応する行方向と列方向の行列 A の対角要素の修正を行う．
(D)	修正された行列 A の対角要素を用いて，行列 U の対角要素 u_{ii} を求める．
(D')	(D) で得られた U の対角要素を用いて，非対角要素 u_{ij} を求める．
(E)	修正された行列 A の対角要素を用いて，行列 U の対角要素に対するコレスキー分解の計算を行う．

以上より，RIC分解は以下のように書き表せる．ここで，a_{ij} は行列 A の要素を格納する配列，u_{ij} は行列 U の要素を格納する配列，d_i は行列 U の対角要素を計算するために必要となる配列，v_j は行列の要素を一時的に保存する配列を表す．また，tol-1は要素 u_{ij} を棄却するか否かの判定に使用する閾値（dropping tolerance value）である．

[RIC 分解]

for $i = 1, \cdots, n$
　$d_i = a_{ii}$
end for
for $i = 1, \cdots, n$
　for $j = i+1, \cdots, n$
　　$v_j = a_{ij}$
　end for
　for $k = 1, \cdots, i-1$
　　for $j = i+1, \cdots, n$
　　　$v_j = v_j - u_{ki}u_{kj}$　(A)
　　end for
　end for
　for $j = i+1, \cdots, n$
　　$\xi = |v_j|/\sqrt{d_i d_j}$
　　if $\xi \leq$ tol-1 **then**
　　　$v_j = 0$　(B)
　　　$d_i = (1+\xi)d_i$　(C)
　　　$d_j = (1+\xi)d_j$
　　end if
　end for
　$u_{ii} = \sqrt{d_i}$　(D)
　for $j = i+1, \cdots, n$
　　$u_{ij} = v_j/u_{ii}$　(D')
　end for
　for $j = i+1, \cdots, n$
　　$d_j = d_j - u_{ij}^2$　(E)

end for
end for

3.2 RIC 分解のロバスト性

RIC 分解が持つ重要な性質の 1 つにロバスト (頑強) 性がある．ここでロバストとは，いかなる問題に対しても，分解の途中で対角要素の値が負になり分解計算の続行が不可能になることがない，ことを指す．RIC 分解がロバストであることは，以下のように示される．初めに，行列 R を行列 U の 1 つの要素 u_{ij} を棄却する行列とすると，行列 R は

$$R = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & 0 & u_{ij} & \vdots \\ \vdots & & 0 & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{4}$$

と表される．このとき行列 A の対角要素を修正する行列 D は

$$D = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & \sqrt{\frac{d_i}{d_j}}|u_{ij}| & 0 & \vdots \\ \vdots & & \sqrt{\frac{d_j}{d_i}}|u_{ij}| & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{5}$$

と表される．ここで，D の要素は u_{ij} の絶対値に第 i 行には $\sqrt{\frac{d_i}{d_j}}$，第 j 行には $\sqrt{\frac{d_j}{d_i}}$ の重みの係数を掛ける．これは，対角要素の修正の大きさが，分解中の A の対角要素の大きさと同じ割合になるように調節するために行う．よって，行列 $S = R + R^T + D$ は

$$S = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & \sqrt{\frac{d_i}{d_j}}|u_{ij}| & u_{ij} & \vdots \\ \vdots & u_{ij} & \sqrt{\frac{d_j}{d_i}}|u_{ij}| & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{6}$$

となり，この行列の第 i 行の要素は第 j 行の要素の定数倍（この場合 $\sqrt{\frac{d_i}{d_j}}$ 倍）であり，かつすべての対角要素は非負であるので，この行列は非負定値となる．したがって，行列 A が正定値ならば行列 $A + S$ も正定値となる．正定値行列のコレスキー分解は計算中に対角要素の値が負になることはないので，分解が中断されることがない．以上の考察により，RIC 分解がロバストであることが示された．

3.3 post filtering

RIC 分解の過程で行われる要素の棄却は，分解過程における計算コストを減らすことを主目的にした処理であった．一方，ここで述べる post filtering の主目的は，RIC 分解の終了後に得られる行列 U の要素を棄却するにより，CG 法の反復計算の計算コストを減らすことにある．この方法は，分解の終了後，要素を棄却するという意味から **post filtering** と呼ばれる．また，従来の **RIC 分解**と区別するために，分解の後に post filtering 処理を施す RIC 分解のことを，以下では特に **pf つき RIC 分解**と呼ぶ．post filtering の処理は次のように表される．

[**post filtering の処理**]

for $i = 1, \cdots, n$
  for $j = i+1, \cdots n$
    if $|u_{ij}| <$ tol-2 **then**
      $u_{ij} = 0.0$
    end if
  end for
end for

ここで，閾値 tol-2 は RIC 分解の中で用いる閾値 tol-1 とは独立に定められる値である．また，tol-2 は分解後の行列 U の要素数をどのくらい棄却するかを決定づける重要なパラメータである．

RIC 分解では，棄却された要素に対して，対角要素の値が負になることを防ぐために，対角要素の修正を行なった．しかし，post filtering の処理で要素を棄却するときにはすでに分解の計算が終了しているため，対角要素の修正を行なわなくても対角要素の値は負になることはない．

4. RIC 分解の実装

この節では，RIC 分解のプログラムへの実装の方法について記述する．前節では，RIC 分解と post filtering の処理について記述した．その処理の記述は，非零要素だけなく零の要素も含めた形で表わされた．しかし，プログラムの実装について考察する場合，係数行列 A が疎であるという構造を考慮した，すなわち，非零要素のみを計算の対象とするような実装を行なわなければ，必要なメモリ量と計算コストの増大を招き好ましくない．そこで，実際のプログラム実装では，行列の非零要素のみを格納するデータ構造である，CCS（Compressed Column Storage）形式を用いている．ただし，ここでは第 3.1 小節の RIC 分解の算法との対応をわかり易くするために，以下では CRS（Compressed Row Storage）形式で説明する．

4.1 係数行列 A と配列の内容について

RIC 分解のプログラムを実装する上で，どのようにして分解の計算を行うかを知るために，図 1 に示すような大きさ 7×7 の行列 A の分解について考える．図 1 の配列 A には係数行列 A の非零要素が格納される．また，配列 colind には配列 A の要素が何列目に存在するかを表す列番号が格納され，配列 rowptr には各行の対角要素が存在する何番目の要素に相当するかを表す行の先頭ポインタが格納される．

ここで，RIC 分解の計算で行列 U が疎の構造を保ったまま計算する必要がある第 3.1 小節の RIC 分解の

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & & a_{17} \\ & a_{22} & & a_{24} & & a_{26} & \\ & & a_{33} & a_{34} & a_{35} & & a_{37} \\ & & & a_{44} & a_{45} & a_{46} & \\ & & & & a_{55} & a_{56} & \\ & & & & & a_{66} & a_{67} \\ & & & & & & a_{77} \end{bmatrix}$$

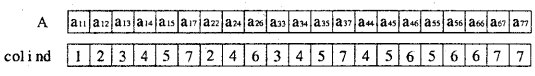

図 1 行列 A および 2 つの配列への値の収納例
Fig. 1 Example of matrix A and two arrays with values

算法 (A) の計算について説明する．行列 U の要素 u_{35} と u_{47} の計算を例として示す．要素 u_{35} の計算式は，

$$u_{35} = a_{35} - u_{13}u_{15} \tag{7}$$

のように表される．また，図 2 は要素 u_{35} の計算時点の行列 U と各配列の値を表す．ここで，配列 U には行列 U の非零要素が格納され，配列 U_colind，U_rowptr には，行列 U に対応した colind と rowptr の値がそれぞれ格納される．また，link には丸で囲った要素のポインタが格納され，next には丸で囲った要素よりも後に存在する要素のポインタが格納される．"*" 印は未知の値であることを表す．要素 u_{33} から要素 u_{37} を囲む四角は計算の対象となる行を表し，要素 u_{35} を囲む四角は現在計算している要素を表す．矢印は要素 u_{35} の計算で関係する要素を指している．

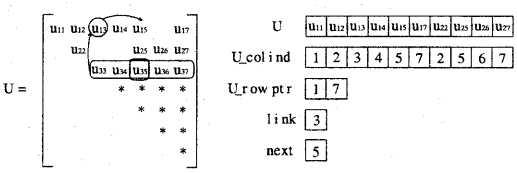

図 2 要素 u_{35} を計算するときの行列 U と各配列に収められた値
Fig. 2 Values stored in some arrays for entry u_{35}

同じように，要素 u_{47} の計算式は，

$$u_{47} = 0 - u_{14}u_{17} - u_{34}u_{37} \tag{8}$$

のように表される．また，図 3 は要素 u_{47} の計算時点の行列 U と各配列の値を表す．

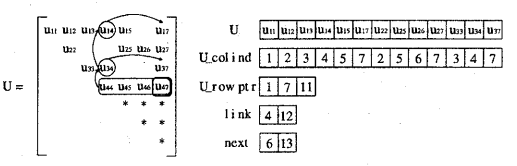

図 3 要素 u_{47} を計算するときの行列 U と各配列に収められた値
Fig. 3 Values stored in some arrays for entry u_{47}

4.2 RIC 分解の擬似プログラム

この節では，第 4.1 小節の計算の例を元に第 3.1 小節での RIC 分解の算法に対応する擬似プログラムを示す．第 3.1 小節での RIC 分解の処理 (A) に相当する擬似プログラムは

```
do k = 1, link の要素数
   if ( U_rowind(next(k)) == j ) then
      x = x - U(link(k)) * U(next(k))
      next(k) = next(k) + 1
   end if
end do
```

となる．ここで，j は計算する要素の列番号であり，x には計算する行列 U の要素に該当する場所の係数行列 A の要素が最初入っているとする．処理 (B)，(C) に相当する擬似プログラムは以下のようになる．

```
xi = abs(x) / sqrt(d(j) * d(i))
if ( xi < tol1 ) then
   d(j) = (1.0+xi)*d(j)
   d(i) = (1.0+xi)*d(i)
else
   u_nnz = u_nnz + 1
   U(u_nnz) = x
end if
```

ここで，d は対角要素を保存する配列であり，u_nnz は行列 U の非零要素数をカウントする変数である．また，sqrt は平方根を求める関数であり，tol1 は閾値を表す。第 3.1 小節の RIC 分解の処理 (D) に相当する擬似プログラムは，

```
uii = sqrt(d(i))
```

と表せ，処理 (D') に相当する擬似プログラムは

```
do px = pl+1, py-1
   U(px) = U(px) / uii
end do
```

と表せる．ここで，uii は行列 U の対角要素を一時的に保存する変数とする．また，pl は行列 U の計算の対象となる行の先頭を指すポインタで，py はその次の行の先頭の要素のポインタを表す．

最後に，第 3.1 小節の RIC 分解の処理 (E) に相当する擬似プログラムは，

```
do px = pl+1, py-1
   d(U_colind(px)) =
     d(U_colind(px)) - U(px) * U(px)
end do
```

と表せる．

5. 数値実験

5.1 計算機環境と計算条件

数値実験は Intel Pentium III (733MHz) の 1PE で行った．搭載されたメインメモリの大きさは 704MB である．プログラミング言語は Fortran90 を使用し，

コンパイルには Intel Fortran compiler version 5.0 を用い，最適化オプションは使用しなかった．計算はすべて倍精度演算で行った．CG 法の収束判定条件は相対残差 L_2 ノルム：$||\boldsymbol{r}_k||_2/||\boldsymbol{r}_0||_2$ の値が 10^{-8} 以下のときとした．右辺項は厳密解がすべて 1 となるように定め，初期値 $\boldsymbol{x}_0$ は全て 0 とした．最大反復回数は行列の次元数と同じ値とし，そこで計算を打ち切った．行列は全て対角項を 1 に正規化した．

5.2 テスト行列

テスト行列は主として構造解析の分野から選んだ．8 個のテスト行列の主な特徴などを表 1 に示す．出処の欄の M.M. は Matrix Market[6]，Florida はフロリダ大学の疎行列データベース[7]，Kouhia は R. Kouhia[5] の同データベースを各々表している．

表 1 数値実験に用いた行列の主な特徴
Table 1 Description of tested matrices.

行列	次元数	問題	出処
bcsstk25	15439	高層ビルの構造問題	M.M.
s3dkt3m2	90449	シリンダーシェルの有限要素解析	M.M.
s3rmt3m3	5357	シリンダーシェルの有限要素解析	M.M.
bcsstk35	30237	車の座席と車体の間の剛性行列	Florida
ct20stif	52329	エンジンブロックに関する剛性行列	Florida
tube1-2	21498	タイヤチューブの構造解析	Florida
smt	25710	トランジスタの熱応力	Kouhia
engine	143571	エンジンヘッドの解析	Kouhia

5.3 実験結果と考察

表 2 に対角スケーリングを施した CG 法（以下，SCG 法と呼ぶ）とフィルインを考慮しない不完全コレスキー分解つき CG 法（以下，ICCG 法と呼ぶ）の実験結果を示す．"∞" 印は各行列の次元数 n と同じ反復回数までで収束しなかったことを意味する．Diag. は SCG 法，IC は ICCG 法の結果である．表中において，precond. は前処理，itr. は反復回数，p-t は前処理の計算時間，itr-t は CG 法の収束までの計算時間，tot-t は前処理と CG 法の合計時間を各々表す．また，p-t の欄の "*" 印は 0 に近い微小な値を表す．計算時間の単位はすべて秒である．表からわかるように，ICCG 法はすべての問題に対して収束しなかった．SCG 法も 2 つの問題で収束しなかった．また，収束した場合でも収束までに多くの反復回数がかかった．

表 3 に，各行列に対して閾値：tol-1 と tol-2 を両方とも変化させたとき計算時間が最も短かった場合の RIC 分解つきの CG 法（以下，**RICCG 法**と略す）と pf つき RIC 分解を前処理とした CG 法（以下，**pf つき RICCG 法**と略す）の実験結果を示す．表中において，tol-1 は RIC 分解の閾値の値，tol-2 は post filtering を行なう際の閾値を tol-1 の倍率で表した値である．また，itr. は反復回数，CPU は前処理と CG 法の計算の合計時間で単位は秒，CPU-ra は **RICCG 法**の CPU 時間に対する **pf つき RICCG 法**の CPU 時間の比，Mem. は計算に要したメモリ量で単位は MB，Mem.-ra は **RICCG 法**のメモリ量に対する **pf つき RICCG 法**のメモリ量の比を表す．表から分かるように，**pf つき RICCG 法**は **RICCG 法**に比べて反復回数は若干多くなるあるが，計算時間は 25.7〜46.0%も短縮され，メモリ量も 25.6〜52.1%削減されたことが分かる．

表 2 SCG 法と ICCG 法の数値実験結果．
Table 2 Numerical results of SCG method and ICCG method without fill-in.

matrix	precond.	itr.	p-t	itr-t	tot-t
bcsstk25	Diag.	9289	*	89.9	89.9
	IC	∞	0.01	–	–
s3dkt3m2	Diag.	40579	*	5239	5239
	IC	∞	0.10	–	–
s3rmt3m3	Diag.	∞	*	–	–
	IC	∞	*	–	–
bcsstk35	Diag.	∞	*	–	–
	IC	∞	0.03	–	–
ct20stif	Diag.	26063	*	1392	1392
	IC	∞	0.07	–	–
tube1-2	Diag.	12119	*	367	367
	IC	∞	0.03	–	–
smt	Diag.	3387	*	375	375
	IC	∞	0.10	–	–
engine	Diag.	2869	*	508	508
	IC	∞	0.13	–	–

図 4 に行列 bcsstk25 に対する **pf つき RICCG 法**における閾値 tol-1 と tol-2（tol-1 に対する倍率）ごとの計算時間 tot-t（秒）の傾向を示す．図 4 より，**pf つき RICCG 法**では，tol-1 が大きいとき tol-2 を増やすと計算時間が増えるが，tol-1 が小さいとき tol-2 を大きくするのに伴い，計算時間が少なくなることが分かる．計算時間が最少になるのは図の閾値 tol-1=0.0005 と tol-2=7 のときである．閾値 tol-1 が最適な値のとき，閾値 tol-2 の値は閾値 tol-1 の値の 7〜10 倍の大きさのときであることが分かる．

図 5 は行列 smt に対する **RICCG 法**と **pf つき RICCG 法**の反復回数と相対残差（常用対数で表す）関係を各々表す．表 3 で示したように，行列 smt について，計算時間が最も短かったときの閾値 tol-1 は **RICCG 法**と **pf つき RICCG 法**で同じ値である．このような場合，反復回数と相対残差の関係は図 5 に示すようになり，**pf つき RICCG 法**の収束までの反復回数は **RICCG 法**よりも少し多くなるが計算時間は逆に少なくなる．

6. おわりに

RIC 分解が終了した後に，分解で得られた行列に対して，再度要素を棄却する post filtering という分解法の有効性を検証した．その結果，この post filtering

表 3 計算時間が最も少ないときの **RICCG 法** (上段) と **pf つき RICCG 法** (下段) の CPU 時間とメモリ量など

Table 3 CPU time (in sec.) and amount of memory in case of the least computation time.

matrix	tol-1	tol-2	itr.	CPU	CPU-ra	Mem.	Mem.-ra
bcsstk25	0.001	-	493	21.3	-	9.65	-
	0.0005	×7	399	14.3	0.671	7.18	0.744
s3dkt3m2	0.0001	-	1151	1028	-	205	-
	0.0001	×10	1196	587	0.571	103	0.502
s3rmt3m3	0.001	-	796	18.7	-	5.51	-
	0.0005	×10	575	10.1	**0.540**	3.79	0.687
bcsstk35	0.00005	-	283	110	-	72.6	-
	0.0001	×10	381	76.8	0.698	36.7	0.505
ct20stif	0.001	-	1984	677	-	77.9	-
	0.0005	×10	1701	426	0.629	52.0	0.667
tube1-2	0.001	-	796	89.7	-	24.5	-
	0.0005	×10	681	56.8	0.633	16.3	0.665
smt	0.005	-	965	252	-	56.6	-
	0.005	×7	1095	183	0.722	32.4	0.572
engine	0.0005	-	202	242	-	194	-
	0.001	×10	326	180	0.743	93.1	**0.479**
Average					0.650		0.602

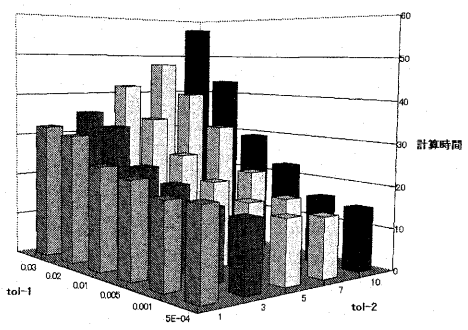

図 4 行列 bcsstk25 に対する **pf つき RICCG 法**の閾値 tol-1 と tol-2 (tol-1 に対する倍率) ごとのかかった計算時間 (秒) の分布.

Fig. 4 Distribution of computation time (in sec.) of post filtered RICCG method for matrix bcsstk25

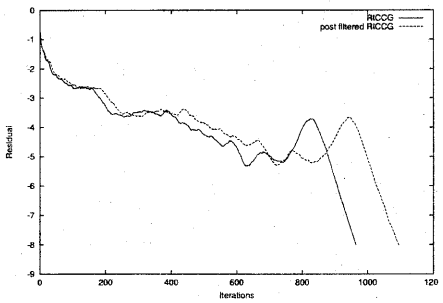

図 5 行列 smt に対する **RICCG 法**と **pf つき RICCG 法**の収束履歴.

Fig. 5 Convergence history of RICCG and post filtered RICCG methods for matrix smt.

つきの CG 法, すなわち **pf つき RICCG 法**は, 従来の **RICCG 法**に比べて, 平均計算時間で約 35%減少させ, 平均メモリ量で約 40%削減させることを実証した.

参 考 文 献

1) Ajiz, M. A., Jennings, A.: A robust incomplete Choleski-conjugate gradient algorithm, *International Journal for Numerical Methods in Engineering*, Vol. 20, pp. 949–966 (1984).
2) Benzi, M., Tuma, M.: A robust incomplete factorization preconditioner for positive definite matrices, *Numer. Lin. Alg. Appl.*, Vol. 99, pp. 1–20 (2001).
3) 池田優介 : Matrix Market における共役勾配法の収束性評価, 九州大学工学部卒業論文, 2002.
4) Kaporin, I. E.: High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU+U^TR+R^TU$-decomposition, *Numer. Lin. Alg. Appl.*, Vol. 5, pp. 483–509 (1998).
5) Kouhia, R. Sparse Matrices web page: http://www.hut.fi/~kouhia/sparse.html
6) Matrix Market web page: http://math.nist.gov/MatrixMarket/
7) University of Florida Sparse Matrix web page: http://www.cise.ufl.edu/research/sparse/matrices/