ILU 分解を用いたマルチレベル前処理

張　　臨　傑[†]　野　寺　　隆[††]

大型で疎な線形方程式の近似解を求める算法に，クリロフ部分空間法とマルチグリッド法がある．マルチグリッド法は大型の線形問題を解く場合には効率的であるが，その性能は PDE 問題に限られている．これに対して，クリロフ部分空間法は任意の疎行列問題に対して有効な算法である．しかし，問題のサイズが大きくなると効率が落ちる．近年，クリロフ部分空間法の前処理の 1 つとして，ILU 分解を用いたマルチレベル前処理が注目を集めている．この算法はマルチグリッドのアイデアを取り込んでおり，両者のメリットを持ち，一般の大型線形問題にも適用できる．本稿では，この ILU 分解を用いたマルチレベル前処理の実装について述べ，数値実験によってその性能を評価する．

Multilevel Preconditioning with ILU Decomposition

Linjie Zhang[†] and Takashi Nodera[††]

Solving large linear systems of equations, two kind of iterative methods are available: Krylov subspace method and multigrid method. Multigrid method has been known to be strong to the size of systems. But, its overall success still relies on an underlying PDE problem in theory. In contrast, Krylov method can be used for solving arbitrary sparse linear systems of equations. However, its efficiency is not good when the size of system becomes larger. Recently, a kind of multilevel preconditioning technique with ILU decomposition has been proposed. This technique makes use of multigrid's idea and hold both merits of multigrid and Krylov subspace method. In this paper, we will give some of implementation of multilevel preconditioning. Numerical experiments are shown for the efficiency of the multilevel preconditioning.

1. はじめに

理工学における様々な現象を記述した楕円型偏微分方程式の境界値問題などを有限要素法や有限差分法によって離散化すると，大型で疎な行列を係数に持つ連立 1 次方程式

$$A\boldsymbol{x} = \boldsymbol{b}, \quad A \in \boldsymbol{R}^{n\times n}, \quad \boldsymbol{x}, \boldsymbol{b} \in \boldsymbol{R}^{n} \qquad (1)$$

が得られる．

このような大型で疎な連立 1 次方程式を解くためには，反復法が良く使われる．反復法には，クリロフ部分空間法やマルチグリッド法がある．マルチグリッド法は，グリッドを物理的に荒いグリッドと細かいグリッドに分け，荒いグリッドの近似解を求めた後，細かいグリッドの近似解を解くことになる．しかし，実装する際には，マルチレベルのグリッドと特殊なチューニングが必要となる．マルチグリッド法は並列化しやすく，大型の問題に強いという利点を持っているが，その算法の効率は強く問題に依存する．これに対して，クリロフ部分空間法は任意の疎行列問題に対して有効な解法として提案された算法である．しかし，問題のサイズが大きくなると，効率が落ちる時もある．

近年，クリロフ部分空間法の収束を改善するための前処理技術が進んでいる．その中で，注目を集めているのは ILU 分解を用いたマルチレベル前処理である．例えば，ILUM[4]，BILUM[5]，ARMS[3] と ARMS の並列版 pARMS[1] などがある．これらの前処理はマルチグリッドの考え方を取り込んで，前処理行列を作る方法である．このような前処理を利用するクリロフ部分空間法は大規模な問題に強いと言われている．本稿では，Saad ら[3] が提案した ARMS (Algebraic Recursive Multilevel Solver) の一部を実装し，数値実験によってその性能を評価する．

まず，2 章で，マルチレベル前処理について述べる．3 章では，クリロフ部分空間法の 1 つ GMRES(m) 法を取り上げて，この前処理の実装について述べる．この前処理を施した GMRES(m) 法の数値実験の結果を 4 章で示す．最後に，5 章においてまとめと結論を述べる．

† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†† 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University

```
1.  Select an unmarked point a
2.  Set count = 0, tmp_set = {a}
3.  For each unmarked b in tmp_set do
        Add b to block independent set
        Mark it, count ++
4.  If count >= b_size, go to 7
5.  For each point c in tmp_set do
        Add unmarked neighbors of c to tmp_set
6.  Go to 3
7.  For each point c in tmp_set do
        Mark unmarked neighbors of c
8.  Go to 1
```

図 1 ブロック版の Greedy な算法

2. ILU 分解を用いたマルチレベル前処理

マルチレベル前処理の基本的な考え方は，グリッドの各点を二つの集合に分けて，reduced system を構成することである．従って，元のシステムを次元の小さい問題に変換して解くことができる．本稿では，すべての点をブロック独立集合に属する点の集合とそのほかの点の集合の二つに分ける．ここで述べる点は物理的な格子点とは違って，未知数のことを意味する．この章では，まず，ブロック独立集合について説明する．その後，ILU 分解を用いたマルチレベル分解について述べる．

2.1 ブロック独立集合

互いに直接に関連していない未知数の集合を独立集合[4] と呼ぶ．Saad ら[5] は，独立集合の定義を拡張し，ブロック独立集合を提案している．ブロック独立集合は，次の条件を満たす点の集合である

- **ブロックに属する点は，ほかのブロックに属する点と直接に関連しない**
- **同じブロックの中の点は関連しても良い**

独立集合はブロック独立集合の特殊なケースである．ブロック独立集合を生成する算法には様々なものがあるが，本稿では ARMS[3] に使われた重み付きブロック版の Greedy な算法を使用する．

まず，ブロック版の Greedy な算法について述べる．ブロック版の Greedy な算法は図 1 の手順で，ブロック独立集合を構成する．本稿ではブロックの最小サイズを b_{size} で表記する．

上記の処理は全ての点がマークされるまで行う．生成されるブロック独立集合の中のブロックのサイズは b_{size} 以上である．ただし，b_{size} を 1 にすると，ブロック独立集合は独立集合となる．更に，2.2 節で述べるブロック独立集合と対応するブロック対角行列を ILU 分解する際に起こるブレークダウンを防止するために，対角優位の行と対応する点だけを選んでブロック独立集合に加える．対角優位を表す数値を重みと言い，次

```
1. Set C = {}, F = {}
2. For each nod a
3.    If a is not marked
4.       Set count = 0
5.       If weight of a < ω_tol
6.          Add a to F, mark a
7.       Else
8.          Set tmp_set = {a}
9.          While count < b_size
10.            For each unmarked nod b in tmp_set
11.               If weight of b < ω_tol
12.                  Add b to F , mark b
13.               Else
14.                  Add b to C , mark b
15.                  count ++
16.               Endif
17.            Endfor
18.            Update tmp_set
19.         Endwhile
20.         For each nod b in tmp_set
21.            For each neighbor c of b
22.               If c is not marked
23.                  Add c to F, mark c
24.               Endif
25.            Endfor
26.         Endfor
27.   Endif
28. Endfor
```

図 2 重み付きの Greedy な算法

の式によって計算する．

$$\bar{\omega}(i) = \frac{|a_{ii}|}{\sum_{j=1}^{n} |a_{ij}|}$$

しかし，極端な対角優位，又はその逆の場合，各行の重みがほぼ同じになる恐れがあるので，ARMS[3] には相対重み

$$\omega(i) = \frac{\bar{\omega}(i)}{\max\limits_{j=1,\ldots,n} \bar{\omega}(j)}$$

を利用して，ブロック独立集合に加える点を選ぶことになる．また，重みの閾値を ω_{tol} で表す．

重み付きブロック版の Greedy な算法を図 2 で示す．ただし，ブロック独立集合を C とし，ブロック独立集合に属していない未知数の集合を F とする．

集合 *tmp_set* を更新する方法については，図 1 の STEP 5 を参照してほしい．

2.2 マルチレベル ILU 分解

マルチレベルの ILU 分解は，まずブロック独立集合に属する未知数が上位になるように，リオーダリングを行う．元の行列 A を A_0 と書くと，オーダリングした後の係数行列は

$$P_0 A_0 P_0^T = \begin{pmatrix} B_0 & F_0 \\ E_0 & C_0 \end{pmatrix} \quad (2)$$

となり，元の式 (1) は

$$\begin{pmatrix} B_0 & F_0 \\ E_0 & C_0 \end{pmatrix} \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{z} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{pmatrix} \quad (3)$$

となる．B_0 はブロック対角行列である．対角にあるブロックのサイズはブロック独立集合にあるブロックのサイズと一致し，b_{size} 以上である．$\boldsymbol{y}$ はブロック独立集合に属する未知数，$\boldsymbol{z}$ はその他の未知数を表す．次に，B_0 を LU（ILU）分解し，行列 A_0 は式 (4) のように分解する．

$$\begin{pmatrix} L_0 & 0 \\ G_0 & I \end{pmatrix} \times \begin{pmatrix} U_0 & W_0 \\ 0 & A_1 \end{pmatrix} \quad (4)$$

ただし，I は単位行列であり，L_0 と U_0 は B_0 を LU（ILU）分解して得られた下三角行列と上三角行列であり，$G_0 = E_0 U_0^{-1}$，$W_0 = L_0^{-1} F_0$ である．行列 $A_1 \approx C_0 - E_0 B_0^{-1} F_0$ は，Schur Complement と呼ばれている．行列 U_0, L_0 は三角行列なので，その逆行列を求めるのは，難しい問題ではない．従って，式 (3) を近似的に解くために，次元の小さい問題

$$A_1 \boldsymbol{z} = \boldsymbol{g} - G_0 L_0^{-1} \boldsymbol{f} \quad (5)$$

を解けば良い．$\boldsymbol{z}$ の近似解を求めた後，後退代入で $\boldsymbol{y}$ を求める．式 (5) は reduced System と呼ばれている．ここで，A_1 の次元がある程度小さくなったら，直接法，又は反復法を利用して方程式 (5) が解ける．そうでなければ，A_0 と同じように，A_1 に対してリオーダリングと ILU 分解を繰り返す．つまり，レベル l における行列 A_l を式 (6) のように並び換え，式 (7) の分解を行う．

$$P_l A_l P_l^T = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} \quad (6)$$

$$A_l \approx \begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix} \times \begin{pmatrix} U_l & W_l \\ 0 & A_{l+1} \end{pmatrix} \quad (7)$$

こうすると，元の係数行列 A_0 を式 (8) のように近似できる．

$$\bar{L}_0 \cdots \bar{L}_l \times \begin{pmatrix} I & 0 \\ 0 & A_{l+1} \end{pmatrix} \times \bar{U}_l \cdots \bar{U}_0 \quad (8)$$

ただし，

$$\bar{L}_i = \begin{pmatrix} L_i & 0 \\ G_i & I \end{pmatrix}, \quad \bar{U}_i = \begin{pmatrix} U_i & W_i \\ 0 & I \end{pmatrix}.$$

この処理は A_{l+1} の次元が十分に小さくなるまで，または，指定したレベルを達成するまで行う．マルチレベル ILU 分解の算法を図 3 で示す．

実装する際には，図 3 の第 4 行で得られる行列 $L_l, U_l, G_l, W_l, A_{l+1}$ を全部保存することになる．本稿では，A_{last_lev} に対して，完全 LU 分解を行う．得られる 2 つの三角行列は図 4 の 4 行目で使われる．

ここで，問題となるのは，A_{l+1} は A_l より，非ゼロ要素が多くなり，密行列になる可能性がある．このため，l の増大につれて，ブロック独立集合のブロックのサイズが大きくなる．ゆえに，式 (6) の B_l の ILU 分解は難しくなる．これは ILU 分解を用いたマルチレベル算法が実用的ではないと言われる原因の 1 つである．しかし，ここでマルチレベルの ILU 分解を使用する目的は，元の式 (1) を解くためではなく，係数行列 A の近似行列を作るためであるので，それほど正確なものを要求しない．従って，疎な性質を保持するために，行列 A_{l+1} の行ごとの非ゼロ要素の数を制限しても良い．更に，メモリと処理時間を考えると，行列 L_l, U_l, G_l, W_l の非ゼロ要素を制限することもできる．

1. If $lev \neq last\_lev$
2. Find an independent set purmutation P_{lev}
3. $A_{lev} = P_{lev}^T A_{lev} P_{lev}$
4. Decomposite A_{lev} as (7)
5. call $multi\_decomposition(A_{lev+1})$
6. Endif

図 3 $multi\_decomposition(A_{lev})$

3. マルチレベル ILU 前処理を用いるクリロフ部分空間法

3.1 前 処 理

前処理は右前処理と左前処理の 2 つがある．式 (1) の代りに，式 (9) を解く方法は右前処理と呼ばれる．

$$\begin{cases} AM^{-1}\boldsymbol{y} = b \\ \boldsymbol{x} = M^{-1}\boldsymbol{y} \end{cases} \quad (9)$$

これに対して，係数行列 A の左に行列 M をかける方法を左前処理と呼ぶ．

$$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b} \quad (10)$$

行列 M は前処理行列と呼ばれ，通常 A の近似行列を利用する．本稿では，右前処理について述べる．

実装する際には，AM^{-1} とベクトルの積を計算する必要がある．しかし，一般的には M^{-1} を計算せず，線形方程式 $M\boldsymbol{x} = \boldsymbol{y}$ を解くことによって，$M^{-1}\boldsymbol{y}$ を求める．

前章で述べたように，マルチレベル ILU 分解によって，行列 A を式 (8) で近似できる．即ち，式 (8) は A の近似行列であり，前処理行列として利用できる．M^{-1} とベクトルの積を図 4 のように，前進および後退代入の処理で求められることである．

本章では，GMRES(m) 法を取り上げて，マルチレベル ILU 分解とクリロフ部分空間法との実装について述べる．

3.2 ILU 分解を用いたマルチレベル前処理を適用した GMRES(m) 法

前処理つきの GMRES(m) 法のうち，FGMRES(Flexible GMRES) 法[6] はよく使われる．FGMRES 法の算法を図 5 で示す．

3.1 節で述べたように，算法の第 6 行のベクトル $\bar{\boldsymbol{v}}$

1. Solve $L_l y = f_l$
2. Compute $g_l = g_l - G_l y$
3. If $lev = last\_lev$
4. Solve $A_{lev} z = g_l$
5. Else
6. Call $multi\_solution(A_{lev+1}, g_l)$
7. Endif
8. Back-Substitute to get y_l

図 4 $multi\_solution(A_{lev}, b)$

1: Choose an intial guess $\boldsymbol{x}_0$
2: Start
3: Set $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\beta = \| \boldsymbol{r}_0 \|_2$,
4: $\boldsymbol{v}_1 = \boldsymbol{r_0}/\beta$
5: For $i = 1$ to m
6: $\bar{\boldsymbol{v}} = AM^{-1}\boldsymbol{v}_m$
7: For $i = 1, 2, \ldots, m-1$
8: $\bar{H}(i,m) = \boldsymbol{v}^H \bar{\boldsymbol{v}}$
9: $\bar{\boldsymbol{v}} = \bar{\boldsymbol{v}} - \bar{H}(i,m)\boldsymbol{v}_i$
10: End for
11: $\bar{H}(m+1,m) = \| \bar{\boldsymbol{v}} \|_2$
12: $\boldsymbol{v}_{m+1} = \frac{\bar{\boldsymbol{v}}}{\bar{H}(m+1,m)}$
13: End for
14: Compute $\boldsymbol{y} = \min_y \| \beta \boldsymbol{e}_1 - \bar{H}\boldsymbol{y} \|_2$
15: $\boldsymbol{x}_m = M^{-1} V_m \boldsymbol{y}_m$
16: $\boldsymbol{r}_m = b - A\boldsymbol{x}_m$
17: If $\| \boldsymbol{r}_m \|_2 < \epsilon$
18: Stop iteration
19: Endif
20: Go to start

図 5 FGMRES(m) 法

は式のまま計算することは少なく，一般的に，以下の順番で計算する．

1. 方程式 $M\boldsymbol{w} = \boldsymbol{v}_m$ を解いて，ベクトル $\boldsymbol{w}$ を求める

2. 次に，$\bar{\boldsymbol{v}} = A\boldsymbol{w}$ を計算する

ただし，方程式 $M\boldsymbol{w} = \boldsymbol{v}_m$ はどんな方法で解いても良い．ILU 分解を用いたマルチレベル前処理を使用する場合，図 4 で与えられた算法を使って方程式 $M\boldsymbol{w} = \boldsymbol{v}_m$ を解く．つまり，図 5 の 1 行目の前に，

Call $multi\_decomposition(A_0)$

を追加し，6 行目を以下のコードで入れ換える．

$\omega = multi\_solution(A_0, \boldsymbol{v}_m)$, $\bar{\boldsymbol{v}} = A\omega$

4. 数 値 実 験

本章では，2, 3 章で述べた ILU 分解を用いたマルチレベルの性能を評価するために，数値実験を以下の環境で行った．

- 計算機：Dell PowerEdge 1750
- OS：Red Hat Linux7.2
- CPU：3.00 GHz ×1 インテル (R) Xeon(R)
- メモリ：512MB
- 収束判定条件：$\| \boldsymbol{r}_m \|_2 / \| \boldsymbol{r}_0 \|_2 \leq 1.0 \times 10^{-12}$
- 重み閾値，ドロップ閾値：10^{-12}
- 初期値ベクトル：$\boldsymbol{x}_0 = (0, 0, \ldots, 0)^T$
- 最大反復回数：20000
- プログラム言語：C 言語
- 計算精度：倍精度

全ての数値例に対して，対角スケーリングを行う．GMRES 法の反復回数については，Arnoldi 過程の 1 反復につき 1 回と数えた．計算時間については Clock() 関数で求めた値を秒単位で表した．ブロック独立集合を生成する回数を $nlev$ で表記し，ブロック独立集合のブロックの大きさを b_size で表記する．分解して得られた行列 $A_{i+1}, L_i, U_i, G_i, W_i (i = 0, \ldots, nlev)$ の行ごとの非ゼロ要素の数を $lnum$ 個まで制限する．指定する $lnum$ より多い非ゼロ要素が存在すれば，その中から，絶対値大きいものから $lnum$ 個を選ぶ．

4.1 数 値 例 1

領域 $\Omega = [0,1] \times [0,1] \times [0,1]$ 上において，次の偏微分方程式の境界値問題を考える[7]．

$$a_1 u_{xx} + a_2 u_{yy} + a_3 u_{zz} + R(a_4 u_x + a_5 u_y + a_6 u_z) + a_7 u = g(x,y,z) \quad \text{on } \Omega$$

ただし，

$a_1 = 2 + \sin(2\pi x)\cos(2\pi y)\cos(2\pi z)$
$a_2 = 2 + \cos(2\pi x)\sin(2\pi y)\cos(2\pi z)$
$a_3 = 2 + \cos(2\pi x)\cos(2\pi y)\sin(2\pi z)$
$a_4 = \sin(4\pi x)$, $a_5 = \sin(4\pi y)$, $a_6 = \sin(4\pi z)$
$a_7 = \sin(2\pi x)\sin(2\pi y)\sin(2\pi z)$

である．右辺 $g(x,y,z)$ と境界条件は厳密解が

$u(x,y,z) = \sin(2\pi x)\cos(2\pi y)\sin(2\pi z)$

となるように定めた．この偏微分方程式は 3 次元移流拡散問題の 1 つである．この例では x, y, z 方向に流れがあり，場所によって流れの強さと拡散の強さが異なる．係数 R が大きくなると流れが強くなる．領域 Ω を $64 \times 64 \times 64$ の格子点に区切り，この偏微分方程式を 7 点中心差分によって離散化した．得られた連立 1 次方程式の次元は $64^3 = 262144$ である．R の値を 2^6 にした．係数行列 A が 7 点中心差分によって得られた行列なので，行ごとの非ゼロ要素の数は 7 以下である．このため，まず $lnum$ を 7 に固定して，$nlev$ と b_{size} を 1 から 4 まで変化して，数値実験を行った．結果を表 1 に示す．

表 1 から，マルチレベル ILU 分解を適用した FGMRES(50) は前処理なしの GMRES(50) と比べて，より少ない反復回数で速く収束したことが分かる．しかも，ILU 分解の処理時間は総処理時間と比べると，それほど大きくはない．本例において，$nlev$ の

表 1 数値例 1 の結果 （I : 反復回数， T:計算時間 (秒)，T_{pre} : ILU 分解の処理時間）

Method	nlev	b_{size}	I	T	T_{pre}
GMRES(50)	-	-	745	215	-
FGMRES(50)	1	1	238	99	5
FGMRES(50)	1	2	214	100	10
FGMRES(50)	1	3	281	133	8
FGMRES(50)	1	4	291	139	7
FGMRES(50)	2	1	240	103	5
FGMRES(50)	2	2	216	106	7
FGMRES(50)	2	3	286	141	6
FGMRES(50)	2	4	286	145	6
FGMRES(50)	3	1	224	97	4
FGMRES(50)	3	2	219	112	7
FGMRES(50)	3	3	282	144	6
FGMRES(50)	3	4	285	147	6
FGMRES(50)	4	1	224	100	5
FGMRES(50)	4	2	285	116	7
FGMRES(50)	4	3	285	149	6
FGMRES(50)	4	4	285	152	5

表 2 数値例 1 の結果 ブロック対角行列 B_i の次元

nlev	b_{size}	B_0	B_1	B_2	B_3
1	1	131072	-	-	-
1	2	93008	-	-	-
1	3	126780	-	-	-
1	4	140088	-	-	-
2	1	131072	31011	-	-
2	2	93008	58590	-	-
2	3	126780	46747	-	-
2	4	140088	43955	-	-
3	1	131072	31011	23187	-
3	2	93008	58590	38378	-
3	3	126780	46747	30120	-
3	4	140088	43955	26922	-
4	1	131072	31011	23187	16191
4	2	93008	58590	38378	24759
4	3	126780	46747	30120	19681
4	4	140088	43955	26922	17288

収束に対する影響は見られなかった．すべての $nlev$ において，$b_{size} = 1$ の際の収束は一番速かった．これを説明するために，生成したブロック対角行列 $B_i(i = 0, \ldots, nlev - 1)$ の大きさを表 2 で示す．

表 2 から，b_{size} を 1 にした場合，得られたブロック対角行列 B_i のサイズはそれほど小さくはない．これは係数行列 A が中心差分法で得られたためであると考えられる．しかも，b_{size} を 1 にすると，B_i が対角行列になるので，ILU 分解は簡単になり，計算時間を短くすることができる．

次に，行列 $A_{i+1}, L_i, U_i, G_i, W_i(i = 0, \ldots, nlev-1)$ の行ごとの非ゼロ要素の数を制限することの収束に対する影響を調べるために，$nlev1 = 4, b_{size} = 1$ に固定して，$lnum$ を 1 から 30 までに変化させて，数値実験を行った．数値実験の結果を図 6 と図 7 で示す．

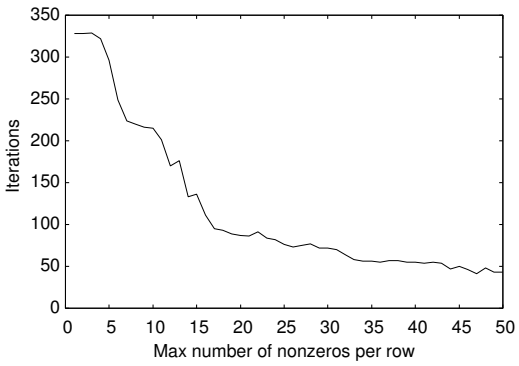

図 6 $lnum$ による反復回数の変化

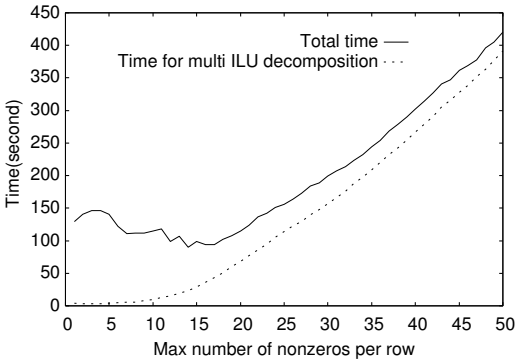

図 7 $lnum$ による処理時間の変化

図 6 から，$lnum$ が大きくなるにつれて，反復回数が大幅に減る傾向が見られる．これは前処理行列が係数行列 A をより近似できたためと考えられる．しかし，代価として，前処理行列の生成時間は大幅に増えている．これは図 7 から確認できる，$lnum$ を 7 程度にすると，処理時間は比較的に少ない．つまり，$lnum$ を元の係数行列の行の非ゼロ要素数とほぼ同じに設定すれば，適切であると思われる．$nlev$ と b_{size} を変えても同じ結果を得た．

4.2 数 値 例 2

この節では，数値例 1 と違う性質を持つ行列を使って数値実験を行う．Matrix Market[2] において提供されている実非対称正方行列 E30R0000 を考える．E30R0000 は次元 9661，306356 個の非ゼロ要素を持つ疎な実行列である．行ごとの非ゼロ要素の数は平均 32，最大 64，最小 7 である．行列 E30R0000 を連立 1 次方程式の係数行列とする．連立 1 次方程式の右辺も Matrix Market において提供されているデータを使用した．この問題は GMRES(50) 法では規定した最大反復回数内で収束しなかった．20000 回反復した時点での残差の log_{10} は約 −5.93 である．マルチレベル ILU 分解を適用した前処理つきの FGMRES(50) 法の反復回数と処理時間を表 3 で示す．ただし，$lnum$ を

表 3 数値例 2 の結果 (I:反復回数，T:計算時間 (秒)，T_{pre}:ILU 分解の処理時間)

$nlev$	1			2		
b_{size}	I	T	T_{pre}	I	T	T_{pre}
1	50	28	27	—	—	—
2	81	24	22	—	—	—
3	81	24	22	—	—	—
4	81	24	22	—	—	—

—: 最大反復回数で収束しなかった場合

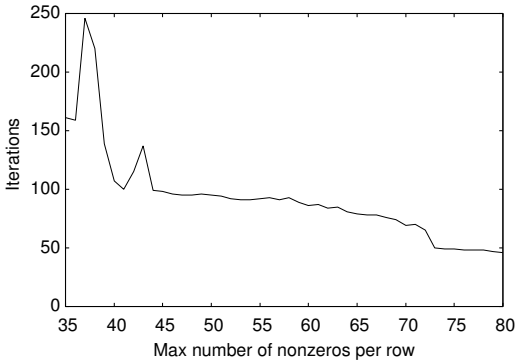

図 8 $lnum$ による反復回数の変化

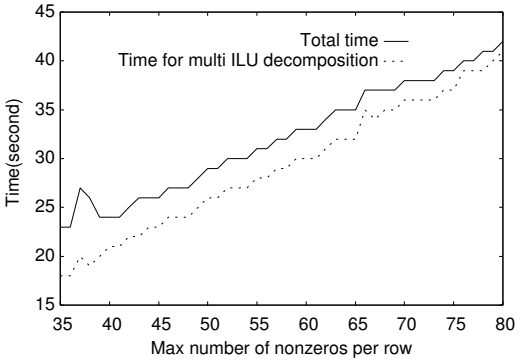

図 9 $lnum$ による処理時間の変化

係数行列 A の行ごとの非ゼロ要素の最大値 64 にした．

表 3 から，$nlev$ を 2 にした場合，最大反復回数内で収束しなかったことが分かる．又，$b_{size} = 2, 3, 4$ の場合，対角行列 B_0 の次元は 1800 であり，$b_{size} = 1$ の場合の 2 倍となる．即ち，構成した reduced system の次元は小さい．これは，$b_{size} = 2, 3, 4$ が $b_{size} = 1$ より速く収束した原因と考えられる．

次に，$nlev = 1, b_{size} = 2$ に固定して，$lnum$ を 32 から 80 まで変化させて，数値実験を行った．得られた数値実験の結果を図 8 と図 9 で示す．ただし，$lnum = 32, 33, 34$ の場合，ブレークダウンが発生したので，図 8 と図 9 には表示していない．

図 8 から，$lnum$ が大きくなるにつれて，反復回数が増えるケースが 2 つ存在しているが，全体的に減る傾向が見られる．また，総処理時間とマルチレベル ILU 分解の計算時間は $lnum$ の増大とともに，大きくなっていることを図 9 から確認できる．本例では，$lnum = 35$ の際に，一番速く収束したが，$lnum$ を 35 より小さくすると，ブレークダウンが発生するリスクがあるので，$lnum$ を係数行列 A の行ごとの非ゼロ要素の最大値にした方が良いと思われる．

5. おわりに

ILU 分解を用いたマルチレベル前処理は大規模な一般の線形問題にも適用でき，収束を大幅に改善できる可能性を持つ算法である．しかし，その性能は設定するパラメーターの値に大きく影響される．各パラメーターの適切な値を決定する方法はまだ提案されていない．

本稿では，三つのパラメーターを数値実験によって考察した．その中で，収束に最も大きく影響するのは，行ごとの非ゼロ要素の数 $lnum$ である．本稿の数値実験の結果から，$lnum$ を元の係数行列とほぼ同じに設定すれば，良い収束が得られることが分かる．

参考文献

1) Li, Z. and Saad, Y. and Sosonkina, M.: pARMS: a parallel version of the algebraic recursive multilevel solver, *Numer. Linear Algebra Appl.*, 10, 485-509, (2003).
2) Matrix Market: Collection of test matrices, available at: http://math.nist.gov/MatrixMarket/index.html
3) Saad, Y. and Suchomel, B.: ARMS: an algebraic recursive multilevel solver for general sparse linear systems, *Numer. Linear Algebra Appl.*, 9, 359-378, (2002).
4) Saad, Y.: ILUM: a multi-elimination ILU preconditioner for general sparse matrices, *SIAM J. Sci. Statist. Comput.*, 17(4), 830-116, (1996).
5) Saad, Y. and Zhang, J.: BILUM: Block versions of multi-elimination ILU preconditioner for general sparse linear systems, *SIAM J. Sci. Statist. Comput.*, 20, 2103-2121, (1999).
6) Saad, Y.: A Flexible Inner-outer Preconditioned GMRES Algorithm, *SIAM J. Sci. Stat. Comp.*, 14, 461–469, (1993).
7) Schnauer, W.: Scientific Computing on Vector Computers, North Holland, (1987).