数　値　解　析 35－3
（1990. 12. 7）

連立一次方程式の高速解法について

長谷川秀彦1)、川端裕一2)、福井義成2)、吉田浩俊3)
1) 図書館情報大学、2) 東芝CAEシステムズ、3) 東芝

密行列を係数とする連立一次方程式の直接解法を高速化の観点から比較した。その結果、高速化のためにはメモリ参照を連続にしたうえでロード、ストア、ブランチの実行回数を減らすのが有効だった。

密行列を係数とする連立一次方程式を高速に解くにはk-j-i 形式の列ガウス、主記憶の制約が強い場合はたてブロックガウス、係数行列が主記憶に収まらない場合はI/O付きたてブロックガウス、対称正定値行列の場合は対称ガウスを使うのがよい。これらのアルゴリズムはメモリ参照が連続であり、アンローリングによって高速化が達成される。

最後に、並列実行が可能な並列たてブロックガウスのアルゴリズムについて述べる。

HIGH PERFORMANCE SOLUTIONS OF LINEAR EQUATIONS

Hidehiko Hasegawa1), Yuichi Kawabata2),
Yoshinari Fukui2), Hirotoshi Yoshida3)
1) University of Library and Information Science, 2) Toshiba CAE Systems,
3) Toshiba, Total Information & Systems Division

We compare some solutions of Linear Equations and evaluate its performance. For getting high performance, it is important that we reduce LOAD instructions, STORE instructions and BRANCH instructions and that the memory access is continuous.

In such conditions, we recommend k-j-i forms Column Gaussian elimination for general matrix, Column Blocked Gaussian elimination for general matrix in virtual memory environment, Column Blocked Gaussian elimination with Input/Output to files for very large general matrix, Symmetric Gaussian elimination for positive definite symmetric matrix.

Finally, we propose a parallel algorithm named Parallel Column Blocked Gaussian elimination.

1.はじめに

大規模な行列計算を各種コンピュータで高速に行うには、たとえば

①演算量を減少させる

②メモリの参照を連続的にする

③ロード、ストア、ブランチの実行回数を減らす

④機種固有の調整を行う

などの対策が必要である。①連立一次方程式の直接解法の場合には演算量が一定だが、反復解法などでは収束の速い解法が精度の点からも重要である。②はメモリ上のアドレス計算の手間を省き、スーパーコンピュータではバンクコンフリクトを防止する。仮想記憶方式のコンピュータではページスワップの影響を受けにくくする。③はレジスタやメモリ上のデータを有効に活用するためで、汎用コンピュータなどのパイプライン処理にも有効となる。④機種固有の調整を最後に行うのは、同じプログラムを異なるコンピュータで使うためである。コンピュータ毎に別々のプログラムを用意することも可能だが、ユーザが色々なコンピュータを使ったときは結果の比較が必要になるため、ここでは「同一のプログラムを各種コンピュータで使う」という方針を採用している。

対象としたのは、スーパーコンピュータ、汎用コンピュータとワークステーションである。最近のスーパーコンピュータは要素並列パイプライン方式を採用しているため、複数あるパイプラインが同一のベクトル命令で起動される。この点でS-810のような以前のスーパーコンピュータにおける高速化とは方針が少し異なっている。プログラムを機種に関係なく使用するため、言語はFORTRANを採用した。

2.ガウスの消去法

密行列を係数とする連立一次方程式の直接解法で基本となるのはガウスの消去法である。Ortega[1]の分類に従って3重ループの内側の演算を

$$a_{ij} = a_{ij} - a_{ik} \cdot a_{kj}$$

のように書けば、ループの制御変数i,j,kの6種類の組合せに対応した基本アルゴリズムができる。数値計算上の精度についてはどの算法も同程度なので、ここでは高速化の観点に注目する[2]。

表1　ループの順序による分類

	通称	演算	メモリ参照
i,j,k	クラウト	内積	改良可能
j,i,k	クラウト	内積	不連続
i,k,j	内積形式	積和	不連続
j,k,i	内積形式	積和	連続
k,i,j	行ガウス	積和	不連続
k,j,i	列ガウス	積和	連続

外側から順にijk、jikとすると、内積演算を主体にした、Wilkinsonの本[3]やNumerical RecipesのLCDCMP[4]などにあるクラウト法(ドゥリットル法)のアルゴリズムになる。内積演算が特別に速いコンピュータとか、混合演算を活用して精度を上げたりする場合を除けば、一般のスーパーコンピュー

タでは内積演算が遅いことと2重のループ内でのメモリ参照量が多いことから高速化が難しい。

ikj、jkiとすると、積和演算を主体にした内積形式と呼ばれるアルゴリズムになる。内積形式というのは線形計画法での積形式に対応したもので、演算が内積になるのではない[5]。文献5のLU22、文献6のSAXGLU、文献7のBG22がそのプログラムで、機械的なアンローリングが有効である。

この他に、クラウト法といいながら積和演算を併用したプログラムにNUMPACのLEQLUW[8]、文献6のCRLUS、文献7のクラウト法などがある。このあたりの区別は人による。これら4種類のアルゴリズムが狭義のLU分解である。

kij、kjiとすると、積和演算を主体にしたガウスの消去法のアルゴリズムになる。この算法を外積形式ともいう。純粋なガウスの消去法では行列Lを作らず、L^{-1}に相当する変換とUを作成する。このような広義のLU分解は、文献6のOGLU、LINPACKのSGEFA[9]、文献10、11のプログラムなどがある。また、このアルゴリズムを用いてLを作ることもできる。

行列の要素a_{ij}をそのままFORTRANの配列として格納した場合に、同一行方向のアクセスはメモリの不連続参照となるので、ikj、kijというアルゴリズムは実メモリ方式のスーコンピュータであっても好ましくない。クラウト法と内積形式はa_{ij}のストアが1回だけだが、外積形式では同じa_{ij}が何度もストアの対象になる。しかし、クラウト法と内積形式はストアの回数は少ないが、ロードによるメモリの参照量が多く、後述のたてブロックガウスのときにも問題がある。また、行交換を含めてメモリ参照を効率的に行い、右辺の前進消去についても高速化を考えるならkji型のガウスの消去法(列ガウス)を使うのがよい。列ガウスは

```
do k = 1, n
    部分軸選択と方程式の入れ換え
    do i = k+1, n
        αik = -aik / akk
    do j = k+1, n
        do i = k+1, n
            aij = aij + αik · akj
```

のアルゴリズムでAを上三角行列Uに変換し、右辺の計算では記憶しておいたα_{ik}を用いた前進消去とUに対する後退代入を行う。最適化能力の低いコンパイラでの使用を考えると、実際のプログラムではa_{kj}を定数として明示したり、α_{ik}を一次元配列に入れるなどの対策が重要である。このようにすると遅くなるコンピュータもあるが、その程度はわずかである。右辺の前進・後退代入はメモリの参照方向に応じて積和演算と内積演算の2種類を使い分けるとよい。

N=2000の問題をスーパーコンピュータS-820/80で解くと、純粋なクラウト法が17.08s、メモリ参照を改良したクラウト法が16.73s、さらに転置をしてページスワップを減少させたクラウト法が10.02s、行ガウスが12.00s、列ガウスが4.25sである。行ガウスと列ガウスにはコンパイラによって8重のアンローリングが行われている。これからもメモリ参照方法に

よって実行時間が大きく変化することがわかる。

3.アンローリング

スーパーコンピュータで高速化を図るには、演算やロード、ストアの回数を減らしたうえで、残った演算とロード、ストアをなるべく並列に実行させる必要がある。最近のコンパイラは自動的にアンローリングを施すこともあるが(陰アンローリング)、複雑なプログラムなどにはアンローリング(陽アンローリング)をしておくのがよい[11,12]。汎用コンピュータのコンパイラはアンローリングをしないが、アンローリングはロード、ストアの回数やループの判定回数が減るため汎用コンピュータに対しても効果がある。多様なコンピュータでの使用を前提とするなら、プログラムが能力の劣るコンパイラでも高性能を発揮するようにすべきである。

kji型のガウスの消去法(列ガウス)に対するアンローリングには

```
do k = 1, n, 2
  do j = k+2, n
    do i = k+2, n
      A(i,j)=A(i,j)+W1(i)·T+W2(i)·T1
```
(1)

と

```
do k = 1, n
  do i = k+2, n, 2
    do j = k+2, n
      A(i,j) = A(i,j) +T·W1(j)
      A(i+1,j)=A(i+1,j)+U·W1(j)
```
(2)

の2通りが考えられる。前者は段数kに対する2重のアンローリングで、2段同時という。2段同時は1回のストアに対して2回の積和演算が実行され、ストアが1/2、ロードが約3/4になる。後者は列jに対する2重のアンローリングで、2列同時という。2列同時は1つのループ中に複数の文を入れただけなので、ロードが3/4になるがストアは減らない。最近のコンパイラでは2重ループの外側のループに対する陰アンローリングを行うが、列ガウスに対する陰アンローリングは効果が小さい。jki型のLU分解ではコンパイラによる陰アンローリングが段数kについてとなるので効果が大きい。

スーパーコンピュータでは、列ガウスの主要部

```
do k = 1, n
  do j = k+1, n
    do i = k+1, n
      A(i,j) = A(i,j) + W(i)·T
```
(3)

の最内側ループが

```
  | VLD(ロード)
Δt| VLD
  ↓ VMAD(積和演算)
ΔtVSTD(ストア)
```

というベクトル命令になる。スーパーコンピュータS-820/80では、2つのロード命令またはロード命令とストア命令が演算と同時に実行可能なので、VLDからVMADまでは1単位時間(Δt)で実行可能であり、一連の処理は2Δtのうちに行われる。

2段同時の列ガウスの主要部

```
do k = 1, n, 2
  do j = k+2, n
    do i = k+1, n
      A(i,j)=A(i,j)+W1(j)·T1+W2(j)·T2
```
(4)

の最内側ループは

```
  | VLD
Δt| VLD
  |
```

```
  ↓VMAD
  |VLD
Δt|VMAD
  ↓VSTD
```

というベクトル命令になり、一連の処理が$2\Delta t$のうちに行われる。(3)の2倍の演算が同じ時間内に行われてループの実行回数が半分になるため、全体の実行時間は(3)の約1/2になる。対応する右辺の前進消去は2重ループの外側のループに関してのアンローリングになっている。後退代入は左辺と独立にアンローリング可能である。2段同時ではメモリの総参照ページ数が半減するので、メモリ制約がきつい汎用コンピュータやワークステーションなどでも有効である。

列ガウスでは、最も内側から一つ外側のループは列に関するループとなり、コンパイラによる陰アンローリングは2列同時となる。2段同時の列ガウスに対して、列について2重のアンローリングを行ったのが2段2列同時である。

2段2列同時の主要部

```
do k = 1, n, 2
  do j = k+2, n, 2
    do i = k+2, n                                  (5)
      A(i,j)=A(i,j)+W1(i)·T1+W2(i)·T2
      A(i,j+1)=A(i,j+1)+W1(i)·U1
                       +W2(i)·U2
```

の最内側ループ(5)が

```
  |VLD
Δt|VLD
  ↓VMAD
  |VLD
Δt|VMAD
  ↓VSTD
```

```
  |VLD
Δt↓VMAD
  |VMAD
Δt↓VSTD
```

というベクトル命令になる。この命令はどう並べ換えても$4\Delta t$はかかり、ロード命令の実行回数が若干減るだけなのでアンローリングの効果は小さい。

表2、表3にアンローリングによる命令の実行回数の変化を示す。表4には各種コンピュータでのCPUタイムを示す。このような線形計算を行うときのピーク性能はS-820/80が2GFLOPS、Y-MPは1CPUあたり333MFLOPSである。S-820/80では列ガウスに8重、2段同時に4重、2段2列同時に2重の列についての陰アンローリングが施されている。S-820/80でNUMPACのLEQLUWが1.08GFLOPSなのを考慮にいれると[8]、いずれのプログラムもかなりの好成績といえよう。表には示していないが、右辺の計算も約10%程度高速化される。Y-MPで2段2列が高速化されないのは、作業用のレジスタが不足するためと考えられる。このことから、アンローリングは単に多重度を増やしたのではいけないことがわかる。

アセンブラを使ってロードの回数と演算量をコントロールすれば、特定のコンピュータの限界性能近くまで高速化できる可能性があるが、FORTRANを使った汎用的なプログラムでは不可能である。

表2　左辺の命令数

	演算密度	ロード	ストア	ロード総数	ストア総数	分岐総数
列ガウス	1	2	1	$2n^3/3$	$n^3/3$	$n^3/3$
2段同時	2	3	1	$n^3/2$	$n^3/6$	$n^3/6$
2列同時	2	3	2	$n^3/2$	$n^3/3$	$n^3/6$
2段2列同時	4	4	2	$n^3/3$	$n^3/6$	$n^3/12$

表3　右辺の命令数

		前進消去			後退代入		
	演算密度	ロード総数	ストア総数	分岐総数	ロード総数	ストア総数	分岐総数
列ガウス	1	n^2	$n^2/3$	$n^2/2$	n^2	$n^2/3$	$n^2/2$
2段同時	2	$3n^2/4$	$n^2/6$	$n^2/4$	$3n^2/4$	$n^2/6$	$n^2/4$

表4　LU分解のCPUタイム(s)と演算性能(GFLOPS)

s (GFLOPS)	S-820/80			Y-MP 1CPU		
	列ガウス	2段同時	2段2列	列ガウス	2段同時	2段2列
N=500	0.094	0.061	0.061	0.460	0.373	0.427
1000	0.584 (1.14)	0.426 (1.58)	0.426 (1.56)	3.297 (0.20)	2.582 (0.25)	3.106 (0.21)
1500	2.821 (1.23)	1.403 (1.60)	1.401 (1.60)	10.74 (0.20)	8.275 (0.27)	10.12 (0.22)
2000	4.256 (1.25)	3.273 (1.62)	3.268 (1.63)	24.70 (0.21)	18.94 (0.28)	23.51 (0.22)

4.対称ガウス

係数行列Aが対称正定値ならば軸選択が不要で、消去後も対称性が保たれれるのでa_{ik}を記憶する必要がなく、メモリ容量と演算量が約半分ですむ。

このような場合には、内積演算中心のコレスキー分解よりも積和演算中心の対称ガウス[10,13]を使うのがよい。対称ガウスは軸選択をせず上三角部分だけで消去を行う行型(kij型)のガウスの消去法で、2段同時、2段2行同時などのアンローリングが適用できる。

対称性を利用した算法では上三角部分(または下三角部分)だけを用いればよいので、必要な部分がひも状の1次元配列になるように変換する。メモリ参照が不連続なアルゴリズムではループの変数に対応させて毎回添字式の値を計算する必要が生じ、ページスワップと添字の計算量が増えて大幅に遅くなる。メモリ参照を連続にするには、コレスキー分解では縦方向番地付け、対称ガウスは横方向番地付けにするのがよい。

5.たてブロックガウス

係数行列Aを複数の列(ベクトル)からなるブロックに分割し、ブロックごとに列ガウスを適用するのがたてブロックガウスである[10,11,12,14,15]。文献5のBGAUS、文献7のBG22Wとは方針が少し異なる。たてブロックガウスのアルゴリズムは

```
do kb = 1, n
  do kk = 1, kb-1
    kbブロックに対して
      kkブロックに対応した消去
  kbブロックの部分軸選択と消去
```

である。一般に仮想メモリ方式のコンピュータをマルチジョブ環境で利用すると、ページスワップの影響で演算に要するCPUタイムの数倍の実行時間が必要になる。たてブロックガウスでは消去に必要なkbブロックとkkブロックだけは実メモリに収まるようにしてやり、小さなワーキングセットで連続的なメモリ参照を行いページスワップの影響を最小限におさえて高速化する。個々の要素に対する演算順序と総演算量は列ガウスと同じであり、2段同時、2段2列同時などのアンローリングも適用できる。

仮想メモリ方式で実メモリの少ない環境で大規模な問題を解くときには、実行時間はページスワップに強く拘束されている。たとえばワークステーションなどでは、たてブロックガウスを用いても、列ガウスを2段同時にしても高速化が可能であり、高速化にはページ参照を減少させることが重要である。この点で②のメモリ参照の連続化が重要なポイントになる。

係数行列Aが主記憶に収まらない場合はI/O付きたてブロックガウスを使う[14,15]。I/O付きたてブロックガウスはたてブロックガウスの拡張で、補助記憶と主記憶の間で係数行列のブロックを入出力しながら消去を行う。アルゴリズムは

```
do kb = 1, n
  READ  kbブロック
  do kk = 1, kb-1
    READ  kkブロック
    kbブロックに対して
      kkブロックに対応した消去
  kbブロックの部分軸選択と消去
  WRITE  kbブロック
```

である。各ブロックが連続した領域として入出力できることが重要である。入出力はコンピュータの利用環境に依存する部分が大きいが、高速化には補助記憶と主記憶とのデータ転送を効率的に行えばよい。スーパーコンピュータの拡張記憶(半導体ディスク)や、非同期入出力や並行入出力を利用して入出力を改善すれば容易に高速化できる。

表5　たてブロックガウスのCPUタイム(s)と演算性能(GFLOPS)

s (GFLOPS) S-820/80	メモリ上			I/O付き		
		2段同時	2段2列		2段同時	2段2列
N＝1000	0.573 (1.16)	0.442 (1.50)	0.504 (1.32)	0.741 (0.89)	0.612 (1.08)	0.678 (0.98)
2000	3.865 (1.37)	3.345 (1.59)	3.660 (1.45)	4.964 (1.07)	4.452 (1.19)	4.788 (1.11)
3000	12.51 (1.43)	11.10 (1.62)	11.85 (1.51)	15.91 (1.13)	14.40 (1.24)	15.28 (1.17)
4000				36.57 (1.16)	34.07 (1.25)	35.95 (1.18)
5000				68.76 (1.21)	65.63 (1.26)	68.53 (1.21)

列ガウスとは異なり、たてブロックガウスの2段2列同時には陰アンローリングが行われず、列ガウスよりも10%程度性能が劣る。拡張記憶を使用したI/O付きたてブロックガウスの性能低下は30%程度である。1ブロックを120列に固定したため、オーバーヘッドが大きくなっている。いずれの場合もブロックサイズの決め方に注意が必要である。

I/O付きたてブロックガウスの場合、右辺の計算は補助記憶からの入出力1回に対する演算量が少ないので、解を1つだけを求めるときの実行時間は入出力の実行時間で決まり、高速化が難しい。複数のベクトルに対する右辺の計算を同時に実行すれば高速化が図れる。

6.並列たてブロックガウス

たてブロックガウスでは、kbブロックの部分軸選択と消去が終われば、kb+1からmブロックまでのブロックに対するkbブロックに対応した消去処理は並列に実行できる。非同期で並列にたてブロックガウスを実行するのが並列たてブロックガウスである。消去可能なブロックを待ち行列に入れて制御し、個々の要素に対する処理を列ガウスと同じになるようにする。アルゴリズム上は係数行列に制限がない。アルゴリズムは

```
when CPUが空いている
    QUEUEから1つタスクを実行

when タスクが終了
    終了したタスクに対応した
      新しいタスクをQUEUEに登録
    if 最終タスク then 終了
```

のようになる。この2種類の処理は非同期で並列に実行できる。高速化のためには、正しい演算順序を保ち、できるだけ多くの処理(タスク)を待ち行列に入れるのがよい。そのため

when kbの軸選択と消去が終了
kb+1, …, mの消去をQUEUEの後に登録
when kbに対してkb-1対応の消去が終了
kbの軸選択と消去をQUEUEの先頭に登録

とする。このアルゴリズムだと厳密にはFIFOでないためQUEUEとはいえないが、複数の純粋なQUEUEを用いて実現することもできる。

ブロック数をm、各ブロックの演算量が1、CPU台数を無限としたとき、処理すべきブロックの総数は

$$\sum_{k=1}^{m} k = (m+1)m/2,$$

並列実行時間は

$$1+2(m-1)=2m-1$$

なので、理論的な並列たてブロックガウスの並列化性能は

$$m(m+1)/2(2m-1) \fallingdotseq m/4$$

となる。CPUがp台ならば

$$\min(p, m/4)$$

が並列化性能の最大値となる。

汎用コンピュータ上で、たてブロックガウスの実測結果を用いて並列たてブロックガウスのシミュレーションを行った結果を以下に示す。

表6　CPU台数pと並列化性能

m \ p	2	4	8	16
10	1.96	3.67	4.44	4.44
15	1.98	3.83	6.08	6.14
20	1.99	3.93	7.22	8.75
25	1.99	3.95	7.57	10.9
50	1.99	3.99	7.91	15.0
100	2.00	3.99	7.98	15.8

CPUの台数に対してブロックが少ないとCPUが空いて効率が落ちるので、ブロック数はCPUの台数の4倍以上は必要である。並列化によるオーバーヘッドがないとしたシミュレーションなので、ブロック数を増やせば効率が上がるようにみえるが、実際はブロックが多すぎるとオーバーヘッドによって効率が低下する。アルゴリズム上ではCPUの台数やブロック数に制限がないため、マルチCPU用のプログラムが単一CPUのコンピュータでも実行できる。この点では汎用的なアルゴリズムといえる。また、2段同時などのアンローリングは並列化とは独立に適用が可能である。

実際にマルチCPUのコンピュータを使うときには、CPUの負荷分散、タスクQUEUEの排他制御、メモリの割り当てなどが問題になる。今後は、並列コンピュータ用のプログラムを作成し、実際のCPUで評価を行いたい。

7.まとめ

密行列を係数とした大規模な連立一次方程式の直接解法を高速化する際の問題点を検討し、実際のコンピュータ上でその効果を検証した。その結果、スーパーコンピュータ、汎用コンピュータともメモリ参照の問題が重要なことが明らかになった。メモリ参照を効率化するにはアルゴリズムから検討が必要である。

大規模な密行列を係数とした連立一次方程式の直接解法としては、一般の非対称行列ならばkji形式の列ガウス、主記憶の制約が大きい場合はたてブ

ロックガウス、主記憶に収まらない場合はI/O付きたてブロックガウス、対称正定値行列なら対称ガウスを使うのがよい。これらのアルゴリズムはメモリ参照が連続であり、アンローリングによって高速化が達成される。

マルチCPUのコンピュータでは、たてブロックガウスを非同期並列に実行する並列たてブロックガウスを使用するのがよいと考えている。しかし、現時点では並列たてブロックガウスの実測ができていないため、性能は未知数である。

謝辞

本論文の作成にあたり指導と討論をしてくださった村田健郎先生に深く感謝します。

参照文献

1) Ortega, J. M. Introduction to Parallel and Vector Solution of Linear Systems. New York, Plenum Press, 1988, 305p.

2) 長谷川秀彦. 密行列を係数とする連立一次方程式の解法(I). 図書館情報大学研究報告. Vol.6, No.1, p.13-44(1987)

3) Wilkinson, J.H.,Reinsch, C.R. Linear Algebra:Contri-bution I/7 by Bower, H. J.,Martin R.S., Peters, G. and Wilkinson, J.H. Chief editor: F.L.Bauer. Berlin, Springer-Verlag, 1971, p.93-110.(Handbook for Automatic Computation, v.2)

4) Press, H.W., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. Numerical Recipes. New York, Cambridge University Press, 1987, 808p.

5) 村田健郎, 小国力, 唐木幸比古. "4 LU分解法". スーパーコンピュータ. 東京, 丸善, 1985, p91-103.

6) 島崎眞昭. "5 連立一次方程式の直接型解法". スーパーコンピュータとプログラミング. 東京, 共立出版, 1988, p.109-139.

7) 津田孝夫. "3 連立1次方程式とLU分解". 数値処理プログラミング, 岩波講座ソフトウェア科学9. 東京, 岩波書店, 1988, p.88-150.

8) 二宮市三, 秦野　世. NUMPACスーパーコンピュータ版の性能. スーパーコンピューティングシンポジウム論文集. p.61-69(1989)

9) Dongarra, J. J., Moler, C. B., Bunch J. R., Stewart, G. W. LINPACK Users' Guide. Philadelphia, SIAM, 1979.

10) 村田健郎. 線形代数と線形計算法序説. 東京,サイエンス社, 1986, VII, 225p.

11) 村田健郎. "スーパーコンピュータと線形計算". コンピュータと数学5: コンピュータから生まれた新しい数学. 野崎昭弘, 廣瀬健編. 東京, 日本評論社, 1986, p.193-208(別冊数学セミナー)

12) 村田健郎, 小国力, 三好俊郎, 小柳義夫. "1.2 算法, Fortranプログラムとハードウェアとの関わり合い". 工学における数値シミュレーション. 東京, 丸善, 1988, p.19-36.

13) 長谷川秀彦. 対称正定値行列を係数とする連立一次方程式の解法の比較. 図書館情報大学研究報告. Vol.9, No.2 に投稿中

14) 村田健郎, 二村良彦, 門間三尚. 仮想記憶方式の下での大形行列計算技法. 情報処理. Vol.21, No.4, p.382-385(1980)

15) 長谷川秀彦. 密行列を係数とする連立一次方程式の解法(II). 図書館情報大学研究報告. Vol.7, No.2, p.45-63(1988)