2次元拡散方程式に対する数値解法の高速化

1994年3月

長谷川 秀彦

2次元拡散方程式に対する数値解法の高速化

1994年3月

長谷川 秀彦

目次

1. 序論

1.1 目的

熱や電磁気に現れる物理現象のうちで、拡散方程式という楕円型の偏微分方程式で表される現象に対して数値シミュレーションを行うことを考える。このとき、与えられた領域(場)上での方程式を、なんらかの離散化手法を用いて離散化して連立の一次方程式に表し、それをコンピュータで解く。数値シミュレーションでは日常的にこのような方程式を繰り返し解く必要があるため、精度がよく、安定で高速なプログラムに対する需要は多い。しかしコンピュータが高速大容量になるにつれ、コンピュータの高速化を実現するためのハードウェア技法が素直でないため、性能をひきだすためにアルゴリズムや高速化手法を見直す必要が生じている。

本研究では、2次元の矩形領域における拡散方程式を差分法で離散化して得られる連立一次方程式$Ax=b$を高速に解くにはどうすればよいかについて述べる。数学的には、対称正定値かつM-行列である規則的疎行列を係数とする連立一次方程式に対する数値解法についての研究ともいいかえられる。最終的な成果として、このような問題に適用可能な、精度がよく、汎用的でしかも高速なコンピュータプログラムを作成する。「高速化」の方法としては、個々のコンピュータに依存しない演算回数や反復回数などの面で高速化を行ない、ベクトルパイプライン方式のスーパーコンピュータに対する高速化手法を汎用性を失なわない範囲で導入する。誤差や精度の比較、ポータビリティなどの面やプログラムの保守で問題が生じるので、個々の計算機アーキテクチャに対応して個別にプログラムを作成するようなことはしない。

係数行列Aは対角要素、その1つとなりの列、対角からある値m1だけ離れた列というように、非零要素が5本のひも状の構造をしている。しかも、対称なので上三角または下三角の3本だけですべての情報を表すことができる。実際の数値シミュレーションでは扱う場と離散化の違いによって対象とする行列の形状や性質が変化するため、数値計算ライブラリとしては多種多様なプログラムが必要となるが、すべての条件に対応できるプログラムを用意することは非現実的である。そのため、ここでは対象をこのような問題に限定する。

連立一次方程式として表現された問題をコンピュータで解くには、ガウスの消去法に基づいた直接解法、または何らかの反復計算によって解に近づけていく反復解法を使う。

直接解法は対象とする問題の性質に影響を受けることが少ないため、安定で確実な方法であるが、行列の帯半幅と元数が大きくなるにつれてメモリ容量や演算回数が爆発的に大きくなって実行不可能となることもある。ここでは帯行列に対してガウスの消去法を適用した帯ガウス[74]、帯ガウスの右辺の計算時に生じる

メモリの不連続参照を改善したMartin-Wilkinsonの特殊ガウス[73]、対称正定値性を利用してピボッティングを省略し、消去範囲を最小限ですます対称帯ガウス[46]と帯コレスキー分解[46]について述べる。これらは行列の帯構造、対称性、正定値性などを利用してメモリ容量や演算回数を節約している。またこれらをコンピュータで高速に実行するには、演算の実行に必要なメモリ·アクセスの頻度を減らすことが重要である。そこで、必要なメモリ·アクセスを削減するための方法である2段同時、2段2行同時のアンローリング[49]について、問題点とその効果を実測値から明らかにする。またストリップマイニング[66]と呼ばれるデータのレジスタ常駐化の方法とその効果についても考察する。これら直接解法のプログラムは種々の反復解法のレファレンスとしても必要不可欠である。

一方、反復解法は必要なメモリ容量と、近似解が得られるまでの演算量が直接解法と比べて平均的には少ないため、大規模な連立一次方程式に対する解法としてよく使われるが、解が得られるまでに必要な反復回数が問題によって大きく変化する。ときには収束しないようなことも起こりうる。ここでは反復解法として、共役勾配法(Conjugate Gradient Method)[32]と前処理を施した方程式にCG法を適用する前処理付き共役勾配法(Preconditioned Conjugate Gradient Method)[42],[51]をとりあげる。CG法は理論的な収束が保証されている反復解法であるが、実際は誤差のために収束しないこともある。そこで、このような現象がどのようなときに生じるのか、また他の反復解法とはどのような関係にあるのかなどを実験を通して明らかにする。

まずはじめに固有値·固有ベクトルを用いてCG法の反復過程を精密に解析し、CG法の数値的な収束特性を明らかにする。この方法では固有値解析の制約から比較的小さな問題しか解析ができないが、数学的な道具建てを用いるため係数行列や右辺に対する一般的な議論が可能となる。ここでは場の物理的状況を変えることによって係数行列の固有値分布を変化させ、固有値分布の変化に対応してCG法の収束がどのように変化するかを数値実験によって明らかにする。このような方法によって、係数行列だけでなく右辺に関する収束特性の解析が可能になった。

CG法は丸め誤差の影響を受けやすいため、実際は前処理が必須とされている。前処理の有効性に対する説明として、固有値が密集すれば残差ベクトルが張る空間、したがって方向ベクトルが動きうる空間の次元が近似的に縮小して、少ない反復回数で近似解に到達するという解説がなされているが[51],[54]、固有値の密集度だけではPCG法の収束が説明できていない。そこでCG法に対する代表的な前処理であるMeijerinkの不完全コレスキー分解を用いた共役勾配法(Incomplete Cholesky Conjugate Gradient Method)[41],[42]とGustafsson流の変更を施した不完全コレスキー分解を用いた共役勾配法(Modified Incomplete Cholesky Conjugate Gradient Method)[18]をとりあげ、前処理の詳しさや修正

ファクターを変えることによって固有値·固有ベクトルの分布を変化させ、収束がどのように変わるかを数値実験を通して考察する。PCG法はAに近い行列$M=\tilde{U}^T\tilde{U}$を用いて、$\tilde{U}^{-T}A\tilde{U}^{-1}x=\tilde{U}^{-T}b$と変形した方程式に対するCG法なので、CG法の収束特性の解析から得られた知見から、どのような前処理がCG法にとって好ましい形の方程式になるかも明らかにできる。

このような解析には、対称正定値行列に対する一般固有値問題を解く必要があるが、一般固有値問題に内在する難しさのため、すべての条件に対する前処理について固有値·固有ベクトルが精度よく求められる保証はない[61]。ところがCG法に対する標準固有値問題については常に高精度な解析ができる。そのため、PCG法に対する一般固有値問題が解けなかった場合でもこのような解析方法は適用できる。

一方、ベクトルパイプライン方式のスーパーコンピュータでICCG法やMICCG法を実行させようとすると、不完全コレスキー分解$U^T DU$の作成と、前処理を作用させる部分$(U^T DU)^{-1}r$がベクトル化できないため、スーパーコンピュータとしての高速性が発揮できない。これを解決する方法として超平面法と呼ばれる方法が提案されている。超平面法は、なんらかの方法によって要素を同時実行可能なグループにわけ、ベクトル演算を可能にする方法である[39]。一般的には1重のループを2重のループに変換し、リストベクトルと呼ばれる一次元配列を用いたメモリの間接参照によってベクトル演算を行う。一般のコンピュータにおいては、ループ長が短くなったり、オーバーヘッドの増加による性能の劣化はわずかであるが、メモリの間接参照は大幅な速度低下をもたらす。

しかし、2次元矩形領域を差分法やバウンダリ·フィット法で離散化して得られた連立一次方程式の係数行列の場合は、問題に内在する幾何学的な性質から、メモリの間接参照を等間隔参照にできるため、リストベクトルが不要になる。スーパーコンピュータでは、この方法はリストベクトルを使用した従来の方法よりも著しく高速になる。汎用コンピュータにおいても、メモリの間接参照を行わないことにより、超平面法によるオーバーヘッドの増加を最小限にできる。これによって、スーパーコンピュータと汎用コンピュータに共通なプログラムが採用できるため、ライブラリの作成·管理の面でも価値がある。このような対称正定値疎行列を係数とする連立一次方程式を反復法で解きながら、同時に条件数を概算する方法についても述べる[29]。この方法は、きわめて簡単で、しかも正確である。

一方、使用するコンピュータによって前処理を変える方法も提案されているが[11],[62],[70]、

(1) 一般的な数値的安定性の議論がむづかしい

(2)コンピュータがかわると良い前処理が大幅にかわる

という理由から、このような方法は採用しない。さらに、超平面法を改良するこ

とで、ICCG法とMICCG法は安定で高速なプログラムに仕立てられるため、コンピュータによって前処理を変える必要性はうすれる。

結果として、2次元の矩形領域における拡散方程式を差分法で離散化して得られた対称正定値疎行列を係数とする連立一次方程式を、直接解法あるいは反復解法によって精度よく、安定かつ高速に解けるプログラムとして実現することができる。このプログラムはスーパーコンピュータ、汎用コンピュータをとわず汎用的に使用できる。その過程では、固有値と固有ベクトルの両方を用いたCG法とPCG法の収束特性の解析方法、反復解法を用いた簡単で高速な条件数の推定方法なども得られた。また、望ましい前処理に必要な条件、あるいは前処理の比較方法などが明らかになったので、これらは実用的にも十分に利用価値がある。

1.2 対象とする問題の特徴

コンピュータの進歩により、最近では物理現象をコンピュータ上の数値計算によって予測することが現実的にできるようになった。このような物理現象を予測するための数値シミュレーションは、偏微分方程式を解くことに帰着される。自然現象をモデル化した偏微分方程式を解くためには、対象とする場を離散化して得られる連立一次方程式の数値解法が最も基本的で重要である。対象とする物理現象は、図1.1のような2次元の矩形領域の温度分布uを求めることとする。これは(熱)拡散方程式とよばれる楕円型の偏微分方程式

$$\mathrm{div}(-k(x,y)\nabla u) = f(x,y) \qquad (1.1)$$

を解くことによって求められる。k(x,y)は拡散係数で、-k(x,y)∇uを拡散束という。拡散係数は場所によって異なるため、この式を一般化ポアソン方程式ともいう。一般の偏微分方程式は厳密な解析解が求められることはまれで、フーリエ展開などの近似を用いるか、数値計算によって数値解を求めることが行われている。また、応用場面でも厳密解が必要なのではなく、どういう解かを示す数値的な分布が分かれば充分なことが多い。また、数値シミュレーションの目的は、対象としている物理現象を与えられた精度で近似することによって、物理現象を予測することにもある[49],[51]。

このような拡散方程式を数値計算で解くためには、差分法や有限要素法などを用いて離散化を行い、連立一次方程式を作るのが一般的である。本研究では差分法による離散化だけを対象とするが、境界適合法(バウンダリ·フィット法)によっても同様な構造をもった行列が得られる。

差分法で離散化をするには、まずは矩形の領域全体に網目(メッシュ)を入れる。未知数としてメッシュの交点(節点)の温度分布uを求めるために、節点i上の値u_iに関する連立一次方程式を作成する。x方向の分割をMx、y方向の分割をMyとして、図1.1のようにy軸に沿って番号が続くように節点の番号付けをする。こ

の差分近似から作られる方程式の帯半幅m1はMy＋1、元数nは(a)がm1(m1＋1)、(b)がm1(2m1＋3)になる。帯半幅m1はメッシュ上の節点のとり方によって変わるが、元数nは節点の数によって決まる。境界条件は

Γ_0でディリクレ条件　　u＝0

Γ_1でノイマン条件　　$\nabla u \cdot n = 0$

とする。(b)ではメッシュの間隔をhとし、□ 部の拡散係数k(x,y) を1, ▨ 部の拡散係数k(x,y) をDF0、DF1、DF2、DF3とする。DF0は10^{-12}に固定した。拡散係数を変化させると条件数の異なる行列が得られる。DFは1、10^{-1}、10^{-3}、10^{-6}などと変化させ、拡散係数の値に段差をもたせることで、規則性をもった重複固有値が得られる。また、工学上の興味のある問題はこの程度の拡散係数の変化範囲内にあるといわれている[50]。

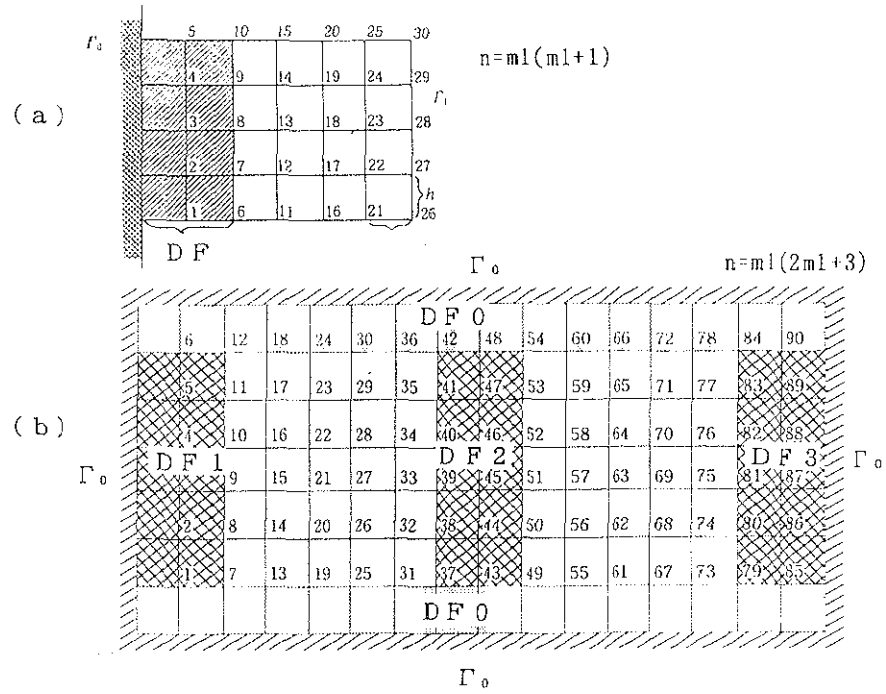
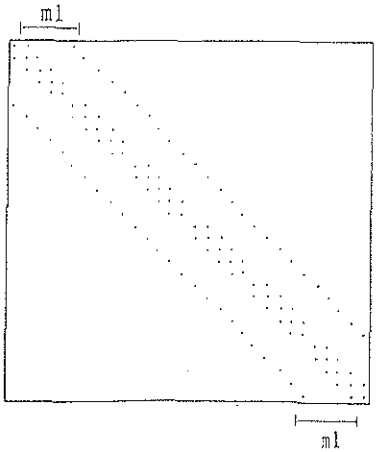


図１．１　テスト問題のための離散場



m1=5,　n=30
・印の部分に非零要素がある

図１．２　差分近似によって得られた行列

差分法の場合には(1.1)の一般化された積分形である

$$\partial/\partial t \int_\Omega u dx = -\int_\Gamma \Phi \cdot n ds + \int_\Omega f dx \qquad (1.2)$$

を基礎式とする[44]。uは物理量で単位は(gまたはerg)/cm³など、Ωは考えている場の体積、Γはその表面積である。(1.2)は古典物理学の保存則から導かれたものであり、dxは体積要素、dsは面積要素、nは外向き法線ベクトルである。(1.2)の表面積分項はガウスの発散定理によって、体積積分になおすことができる。ガウスの発散定理

$$\int_\Gamma \Phi \cdot n ds = \int_\Omega div\Phi dx \qquad (1.3)$$

を(1.2)に代入してまとめると、

$$\int_\Omega (\partial u/\partial t + div\Phi - f) dx \qquad (1.4)$$

となる。この式の( )の中を0とおいた式

$$\partial u/\partial t + div\Phi = f \qquad (1.5)$$

が保存則の微分形式である。

多くの場合、Φは拡散項 -k(x,y)∇u(または、-k(x,y)grad u)と移流項buからなる。本研究では移流項buがない拡散方程式を扱うが、移流項のある問題は移流拡散方程式とよばれ、流体力学、化学工学、生態学などのより広い応用範囲をもつ[51]。

拡散方程式には、

時間依存問題: $\partial u/\partial t + div(-k(x,y)\nabla u) = f$

定常問題: $div(-k(x,y)\nabla u) = f$

がある。このような拡散方程式を差分法や有限要素法によって離散化すると、時間依存問題からは常微分方程式系、定常問題からは連立一次方程式系が導かれる。差分法で離散化された連立一次方程式の数値解法として考えると、定常問題のほうが悪条件になりやすいため、本研究では定常問題だけを扱う。時間依存問題の場合は時間積分が必要になるが、対角優位性が強まるので連立一次方程式の数値解法としては定常問題よりも容易になる[47]。また、離散系の性質は、主として(1)境界条件、(2)場の物理定数、(3)離散化方法と網目の細かさによって決まる。

物体の内点u_iに対応する方程式を作るためには、座標(x,y)の位置にある点iを中心とした一辺がhの正方形を考え、基礎式(1.2)にもとづいて正方形の各面について(熱の)流入量を計算する。問題を簡単にするためメッシュの間隔hはどの方向についても同一とした。

内点i におけるu_i に関する方程式は、内点i を囲む一辺がhの正方形の4辺から外へ出ていく熱量が正方形内の発熱量に等しいことから、求める方程式は

$$\begin{aligned} &-k(x-h/2,y)u_{i-m1} - k(x,y-h/2)u_{i-1} \\ &\quad + (k(x-h/2,y) + k(x,y-h/2) + k(x,y+h/2) + k(x+h/2,y))u_i \\ &\quad - k(x,y+h/2)u_{i+1} - k(x+h/2,y)u_{i+m1} = f_i h^2 \end{aligned}$$

となる。f_iは点i に加えた熱量である。境界上の点についても同様な方法で方程式

を作ることができる。x軸方向とy軸方向でメッシュ間隔(アスペクト比)を変えたり、拡散係数に1以上の値を与えると行列の性質が変わるが、ここでは$\|A\|_1 \leqq 8$ である。

これから対称正定値疎行列Aを係数とする大規模な連立一次方程式

$$Au = f \tag{1.6}$$

が得られる。第i番方程式は

$$a_{i,i-m1}u_{i-m1} + a_{i,i-1}u_{i-1} + a_{i,i}u_i + a_{i,i+1}u_{i+1} + a_{i,i+m1}u_{i+m1} = f_i$$

と表され、離散化して得られる行列は非対角帯半幅m1の対称正定値疎行列になる。Aの非ゼロ要素は5本の対角部分で構成され、対称であることから上三角部分または下三角部分だけを考えればよい。ここでは上三角部分を考え、方程式番号iを固定したとき、対角要素をa_i、対角の右どなりの要素をb_i、対角からm1離れた要素をc_iで表し、実際のプログラムにおいても3本の一次元配列を用いる。図1.1(a)で、m1＝5、n＝m1(m1＋1)＝30 の場合について、差分法によって離散化して得られた係数行列Aの非ゼロ要素を図1.2に示す。3次元領域を離散化したときも、対角部分の本数は異なるが同様な帯行列が得られる。

連立一次方程式(1.6)は、次元nが小さければ教科書に書いてあるプログラムでも簡単に解けるが、次元nが大きくなるにつれて色々と面倒な問題が現れ、一筋縄ではいかなくなる。

1.3 関連する研究

直接解法のアルゴリズムは古くから知られているが[73],[74],[46]、LINPACKベンチマークなどに用いられる密行列に対する直接解法とは異なり、帯行列に対する直接解法は高速化があまり進められていない[60]。BLASのように基本的な演算を高速化することは行われているが[10]、連立一次方程式の数値解法全体を高速化するような試みはきかない。しかし、1.2章でみたように、物理現象を数値シミュレーションも用いて解こうとすると、帯行列を係数にもつ連立一次方程式が現れるため、帯行列に対する直接解法は非常に重要である。

大規模な問題に対しては反復法が重要だが、その収束特性は完全にわかっているわけではない。特に、PCG法がどのような収束特性を示すかについての研究は、特定の問題がどのような収束を示したかを述べたものがほとんどである[2],[7],[16],[36],[37],[40],[51]。そのため、自分が解こうとしている問題がどのような収束特性を示すかについての情報を事前に知っておきたいという目的には無力である。理論的に反復回数の上限を解析した研究もあるが、対象としている問題が理想的な問題であることと、多くの場合過大推定のため現実的ではない[15],[17],[59],[63],[65]。一般的に数学的な理論解析からは、何らかの式として評価が得られることが多いが、現実的な場を数学的な式だけから解析するのには無

理がある。一方、数値実験は現実的な問題に対しても適用できるが、式としての評価を得ることは難しく、多くの場合は定性的なことがわかるにすぎない。なかには、現実的な問題に対する固有値分布から解析した研究もあるが[3],[4],[6],[7],[51],[5]、ほとんどは係数行列に対する解析で終わっている。また反復解法の収束は右辺にも依存するが、このような解析方法では方程式の右辺や解に対する依存性は解析できない。

そこで、本研究では固有値と固有ベクトルの両方を活用した解析を行う。このような解析方法は固有値解析の難しさのためか、文献には現れていない。本来の目的は実用的なPCG法の収束特性の解析だが、固有値解析の困難がつきまとうので、まずはCG法の解析を精度よく行う。そのあとで、PCG法を前処理が施された行列に対するCG法とみなして解析する。以前からの固有値·固有ベクトルを用いた解析には、固有値分布、条件数、残差多項式などが使われていたが、これらからは実際の解析に役だつ情報はあまり得られなかった。そこで、ここでは理論と実際の数値実験がどこからどのように食い違うようになるかを明らかにすることを第一の課題とする。

考察すべき項目としては、場の物理的状況、問題の大きさ、メッシュの形状、メッシュの大きさなどがあるが、純粋に理論的考察が可能な問題だけでは実際の応用問題には不充分である。そのため、本研究では主として数値実験を行い、これらの条件を変えたときに収束がどのように変化するかを定性的に明らかにする。

さらに、本論文ではICCG法が収束に到るまでの反復回数を変化させずに高速化する方法についても議論する。これまでICCG法の高速化には超平面法が用いられ[39]、2次元問題ではICCG(1,1)のみが高速化の対象になっていた[39],[68]。ところが超平面法を実現するために用いられるリストベクトルは間接的なメモリ参照になるので、スーパーコンピュータ以外ではオーバーヘッドが大きかった。本研究では、オーバーヘッドを減らすことと、ICCG(1,1)以外のICCG法に対しても高速化することを試みる。その際、プログラムがスーパーコンピュータと汎用コンピュータの両方で汎用的に使えることを重視する。最適なプログラムは、反復回数の減少の効果と前処理の高速化の効果によって決まるので、汎用的な複数のプログラムを用意できれば問題に応じた選択が可能になる。

一方、使用するコンピュータの性能を劣化させないためには、コンピュータによって前処理を変えたほうがよいという意見もある[11],[62],[70]。しかし、このような方法は

(1) 結果や反復回数がコンピュータによって異なる
(2) 一般的な数値的安定性の議論がむづかしい
(3) コンピュータがかわると良い前処理が大幅にかわる

という問題点があるため、汎用的とはいいがたい。「精度の保証された一般的算

法で汎用かつ高速に実行する」ため、本論文ではICCG法の収束に必要な反復回数を変えない。

最後に、反復解法を用いて連立一次方程式を解くのと同時に条件数を概算する方法について述べる。直接解法の場合は連立一次方程式を解く際に条件数の概算する方法がいくつも報告されている[12],[33],[34],[35],[55],[77]。ところが反復解法の場合はそのような報告がほとんどないため、反復解法を用いて連立一次方程式を解く際にも条件数の概算を可能にする[29]。このとき付加的に必要なメモリやプログラム、実行に必要な演算量はわずかであることが必要とされる。

2. 直接解法

2次元矩形領域の場における拡散方程式を差分法を用いて離散化した結果は、x方向の分割をMx、y方向の分割をMyとすれば、おおむね元数n＝Mx×My、帯半幅m1(＝MyまたはMx)の帯構造の内側のみに非零要素をもった方程式となる。帯半幅m1はメッシュ上での格子点の順序づけ、元数nは格子点の数で決まり、節点の番号付けや境界条件によって若干の異なりはあっても、この帯構造に大きな変化はない。加えて拡散方程式の離散化によって得られた係数行列は対称正定値行列、M-行列でもある。これから、拡散方程式の数値解法として帯行列あるいは対称正定値帯行列を係数とする大規模な連立一次方程式の解法が必要になる。

このような問題はメモリ容量の制約から一般的な密行列用の算法では問題がある。例えばx方向の分割とy方向の分割を100にすると、帯半幅が100程度、次元nが10000の帯行列が得られる。この行列を8バイトの倍精度実数で格納しようとすると、密行列では800メガバイト、帯行列では16メガバイトを必要とする。制限三項演算

$$a_i = a_i + t \cdot a_i$$

を基準にして演算量を考えると、密行列に対するガウスの消去法では約$n^3/3$回、帯行列に対して後述の帯ガウスを用いると$2m1^2 \cdot n$ 回である。基本演算が10^{-6}秒(200MFLOPS)で計算できるコンピュータなら、CPU時間はそれぞれ45時間と1分になる。800メガバイトのメモリ容量が許されることは非現実的でも、1GFLOPSのスーパーコンピュータはきわめて現実的なので、問題はメモリネックである。

ここでは帯行列を係数とする連立一次方程式の直接解法の代表的なアルゴリズムである帯ガウス[74]、R.S.MartinとJ.H.Wilkinsonの特殊ガウス[73]、対称正定値帯行列を対象にした対称帯ガウス[46]と帯コレスキー分解[46]などとその高速化について述べる。いずれの算法も帯行列のなかの非ゼロ要素部分だけを計算の対象として、メモリ容量と計算量を削減している。

ここで、アルゴリズムがあることと、使えるプログラムがあることは別問題である。実際にはメモリ容量を節約することでプログラムが複雑になり、アルゴリズムのとおりに書いたプログラムと高速化を意識したプログラムでは実行時間に大きな差が生じる。帯行列の場合は、係数行列と作業領域を格納するのに必要なメモリ容量がどの位で済むかが一番の問題である。スーパーコンピュータの出現によって、メモリ容量に比べるとCPU時間はそれほど問題でなくなったが、それでも短いに越したことはない。そこで、これらの算法を高速化するのための問題点を明らかにする。

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)ワーキングセットを小さくする
(4)ロード、ストア、ブランチの実行回数を減らす
(5)機種固有の調整を行う

などの対策が必要である[28]。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、(1)の条件から、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となる。(2)と(3)は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し、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。(4)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。(5)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、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。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、言語はFORTRANとした。BLASのように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[1],[10]。

高速化の対象としたのは、要素並列ベクトルパイプライン方式のスーパーコンピュータ[53]、汎用コンピュータとワークステーションである。このように多様なコンピュータでの使用を前提にするなら、機能が劣るコンパイラでもプログラムを高速に実行できるようにする必要がある。そこで、帯行列を係数とした大規模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、仮想メモリ方式のコンピュータに限らず、実メモリ方式のスーパーコンピュータでもメモリ参照の効率化が重要であることが明らかになった。ところがメモリ参照の効率化にはアルゴリズムからの検討が必要である。これから述べるアルゴリズムはメモリ参照が連続で、アンローリングによって高速化できる。実際、何もアンローリングをしていないプログラムはロード、ストアの実行回数やメモリの参照量に問題がある[24]。

スーパーコンピュータのための高速化手法としては2段同時、2段2行同時のアンローリングが有効であるが[49]、これらの手法をパイプラインの制御が途切れることなく実行させることは簡単ではない。しかし、これらの対策がうまく効けば、ループ長が短いため性能がでにくい帯行列の場合にもよい結果がもたらされる。さらに2段同時、2段2行同時にアンローリングを施したプログラムは、汎用コンピュータでも高速に実行できる。

なお、一般の定義では帯半幅に対角要素を含めるが、ここでは対角要素を含めない。したがって、(ここでの)帯半幅m1=(一般の)帯半幅-1である。

2.1 直接解法のアルゴリズム

(1) 帯ガウス

密行列についての解法にはガウスの消去法とクラウト法によるLU分解が考えられ、精度とメモリ参照の問題を考えるとガウスの消去法がよかったが、クラウト法でも解けないことはなかった[20]。

ところが帯行列の場合には事情が異なる。部分軸選択を行う場合のクラウト法では、方程式の交換を表す行列Pを用いて

$$PA=LU \qquad L:\text{下三角行列、}\ U:\text{上三角行列}$$

と分解される。ここで帯半幅がm1のとき、第k番方程式と第k＋m1番方程式を交換すれば帯幅が4m1になる。このような方程式の交換を繰り返すと帯幅がどんどん広がり帯行列が密行列化し、最悪の場合には下三角部分は密で、上三角部分は帯半幅2m1の行列ができる。これでは問題を帯行列に制限してメモリ容量を節約した意味がない。ところが、帯行列に対するガウスの消去法では部分軸選択のために上三角行列に帯半幅の2倍が必要になるだけである[21]。したがって帯行列を係数とする連立一次方程式でピボット選択を必要とする問題に対する解法としては、クラウト法を用いたLU分解は採用できない。

帯行列を係数とする連立一次方程式にガウスの消去法を適用したときの第k段では、非対角帯半幅をm1とすると、k＋m1＋1行以降の方程式のx_k の係数a_{ik}はすべて0だから、その部分は消去する必要がない。またk番方程式の係数でx_{k+m1+1}以降はすべて0なので、方程式の交換がなければk＋m1列までが消去対象になる。

k段で第k番方程式とip(k) 番方程式との交換が起こったとする。下三角部分のk-1列まではすべて0なので、k列から上三角部分の非零要素の端までが交換対象となる。帯半幅がいちばん広がるのはip(k)＝k＋m1行と交換が起こった場合で、第k番方程式は非零要素がk＋2m1 列までとなる。このとき、k番方程式による消去を行うとk＋1 番からk＋m1番までの方程式のk＋2m1 列までは非零になる可能性がある。しかし、右帯半幅は常に2m1 以内に収まり、2m1を越えることはないし、部分軸選択ならびに消去の対象になる方程式の本数も変化しない。

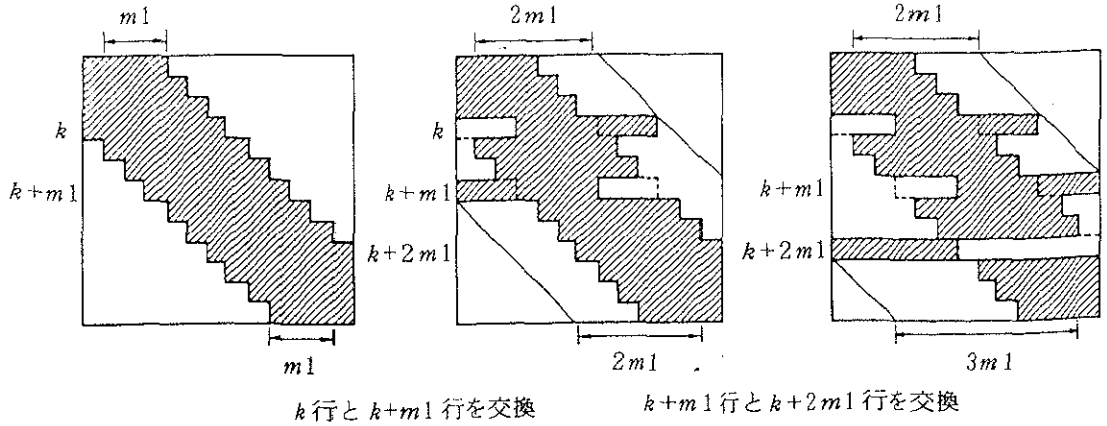


図２．１　クラウト法における方程式の入れ換え

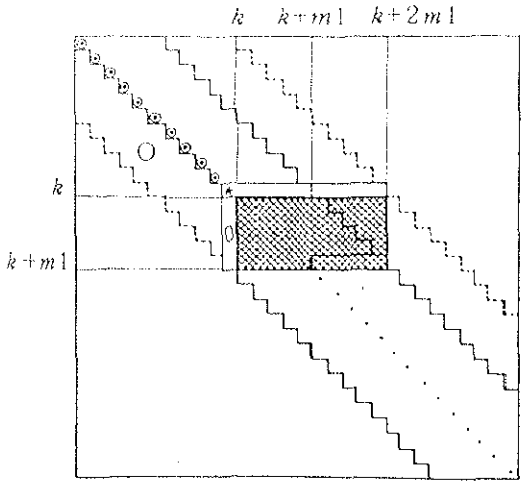


図２．２ 帯行列に対するガウスの消去法

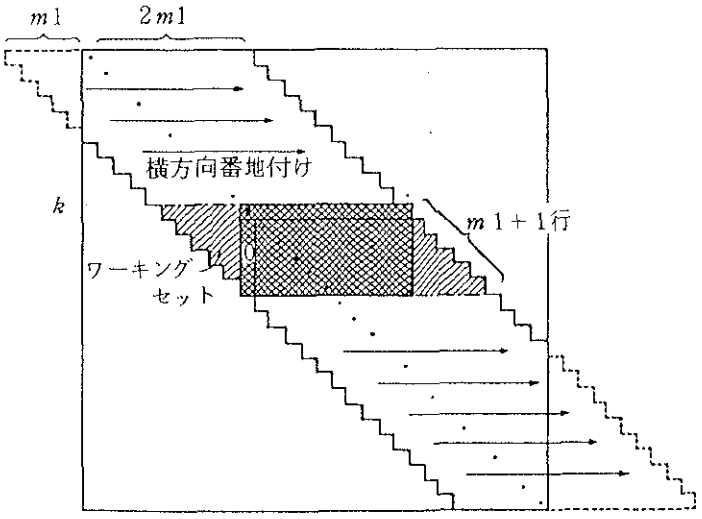


図２．３ 横方向番地付け

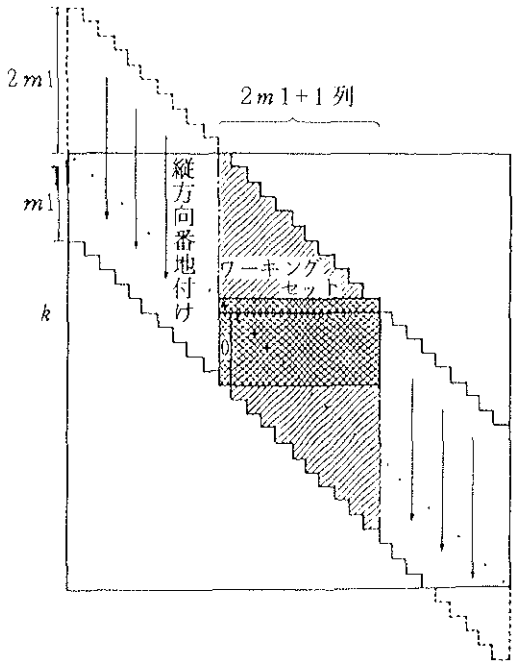


図２．４ 縦方向番地付け

帯行列を格納するのに必要なメモリ容量(3m1+1)×nダブルワードをメモリ上に格納する方法として、方程式毎メモリに格納する横方向番地付けと未知数毎メモリに格納する縦方向番地付けの二通りが考えられる。図2.3のように横方向に番地付けすると、行型ガウスではもっとも内側のループで連続的にメモリが参照されるが、列型ガウスでは3m1とびの参照になる。ワーキングセットと呼ばれる第k段の消去過程で実効的に必要とされるメモリ容量は行型ガウスの場合(m1+1)×(3m1+1)≈3m1²となる。図2.4のように縦方向に番地付けすると、列型ガウスではもっとも内側のループで連続的にメモリが参照されるが、行型ガウスでは3m1とびの参照になる。列型ガウスのワーキングセットは(3m1+1)×(2m1+1)≈6m1²となる。コンピュータの高速性をひきだすためには、メモリ参照が連続的でワーキングセットが小さいことが望ましいので、帯行列に対して方程式入れ換え方式の部分軸選択を行うガウスの消去法では、横方向に番地付けした行型ガウスがよい。

帯行列に対するこのようなガウスの消去法を帯ガウスといい[74],[46],[21]、アルゴリズムは以下のようになる。

```
do k = 1, n
    amax = | a_kk |;  ip(k) = k
    do i = k+1, min(k+m1,n)
        if | a_ik | > amax then
            amax = | a_ik |; ip(k) = i
    do j = k+1, min(k+2m1,n)
        a_k,j ↔ a_ip(k),j
    do i = k+1, min(k+m1,n)
        do j = k+1, min(k+2m1,n)
            a_ij = a_ij - a_ik·a_kj/a_kk
```

帯ガウスの第k段での演算量は制限三項演算を基準に考えて約2m1²回なので、全体の演算量は約2m1²n回となって、演算量も大幅に節約できる。密行列に対するガウスの消去法の演算回数n³/3回と比較すると、およそn>2.5m1 = √6m1 の場合、すなわち帯半幅が元数nの1/2.5よりも狭い場合には帯ガウスの演算量が少なくなる。

実際のプログラムでは係数行列を

$a'_{j-i,i} = a_{ij}$

と変換して

a'(-m1:2m1,n)

という(3m1+1)×nの配列に収める。メモリを節約したため、部分軸選択を行うときにはメモリ参照が不連続になる。

a_{ik}(実際はa(k-i,i)) にはスカラーtを用いてループ内で不変なことを明示しておく。a_{kj}(実際はa(j-k,k)) には一次元配列を用いて第k段の消去過程では不変なことを明示しておく。一次元アレイに格納するために2m1回の余計な代入が必要になるが、$2m1^2$ 回の繰り返しを高速化するためには有効である。プログラムふうに書くと

```
do k = 1, n
   amax = | a(0,k) |; ip(k) = k
   do i = k+1, min(k+m1,n)
      if| a(k-i,i) | > amax then
         amax = | a(k-i,i) |; ip(k) = i
   do j = k+1, min(k+2m1,n)
      a(k,j) ↔ a(ip(k),j)
   do j = k+1, min(k+2m1,n)
      wk(j) = a(j-k,k)
   do i = k+1, min(k+m1,n)
      a(k-i,i) = -a(k-i,i)/a(0,k)
      t = a(k-i,i)
      do j = k+1, min(k+2m1,n)
         a(j-i,i) = a(j-i,i)+t*wk(j)
```

となる。

プログラムの書き方を変えて汎用コンピュータM260H で実行したときのCPU時間は、アルゴリズム通りに書いたプログラムに比べて、スカラーと一次元アレイの両方を使ったプログラムは約20% も短くなることがわかっている[21]。

(2) R.S.Martin とJ.H.Wilkinsonの特殊ガウス

次に、帯行列を係数とした連立一次方程式の右辺の計算を重視した算法とその適用範囲について述べる。帯ガウスの右辺の前進消去過程に現れる

```
do i = k+1, min(k+m1,n)
   b(i) = b(i)+a(k-i,i)*b(k)
```

の処理は、配列aの第一添字と第二添字が同時に動くため不連続なメモリ参照となりページスワップの影響を受けやすい。そのため時間依存問題など、同一の係数行列をもった方程式の右辺だけを変化させて繰り返して解くような応用には向かない。そのため、上三角行列と消去変換に用いたαの領域をメモリ上に分離して格納し、帯ガウスと同一のメモリ容量でメモリ参照を連続にするのがR.S.MartinとJ.H.Wilkinsonの特殊ガウスである[73],[22]。この算法では局所的なメモリ参照が改善されるほか、ページスワップの対象になるメモリ量が帯ガウスの約半分になる。特に、同一の係数行列について右辺の計算を繰り返すと帯ガウスより

もCPU時間が有利になる。実際に仮想メモリ方式のコンピュータでマルチジョブ運転をすると、実行時間にはCPU時間以上の差が現れるため、適用範囲はここでの結果よりも広がる。以下、MartinとWilkinsonの特殊ガウスと略称する。

帯ガウスの前進消去過程は図2.2のようになるので、消去変換に用いるαを分離すれば、消去に必要な領域の帯幅は2m1+1である。そこで帯幅2m1+1の長方形領域で帯ガウスを実行するのがMartinとWilkinsonの特殊ガウスである。このアルゴリズムは帯ガウスの拡張であり、帯ガウスとはデータの格納方法が異なるが、演算順序は同じなので計算精度は同等である。

MartinとWilkinsonの特殊ガウスの場合には、図2.3のようなデータを横方向番地付けで格納し、消去変換に用いたαは別の配列に格納する。帯ガウスでは第k段の消去過程で0にされた部分にαをいれておいたが、MartinとWilkinsonの特殊ガウスでは消去後に結果を1列分だけ左にシフトして0となる部分を残さない。

実際は係数行列と消去後の上三角行列を横方向番地付けで、図2.5のように

a'(0:2m1,n)

の領域に格納する。データの格納方法をプログラム風に明示すると、

```
do i = 1, m1
   do j = max(i-m1,1), min(i+m1,n)
      a'j-1,i = ai,j
do i = m1+1, n
   do j = max(i-m1,1), min(i+m1,n)
      a'm1+j-i,i = ai,j
```

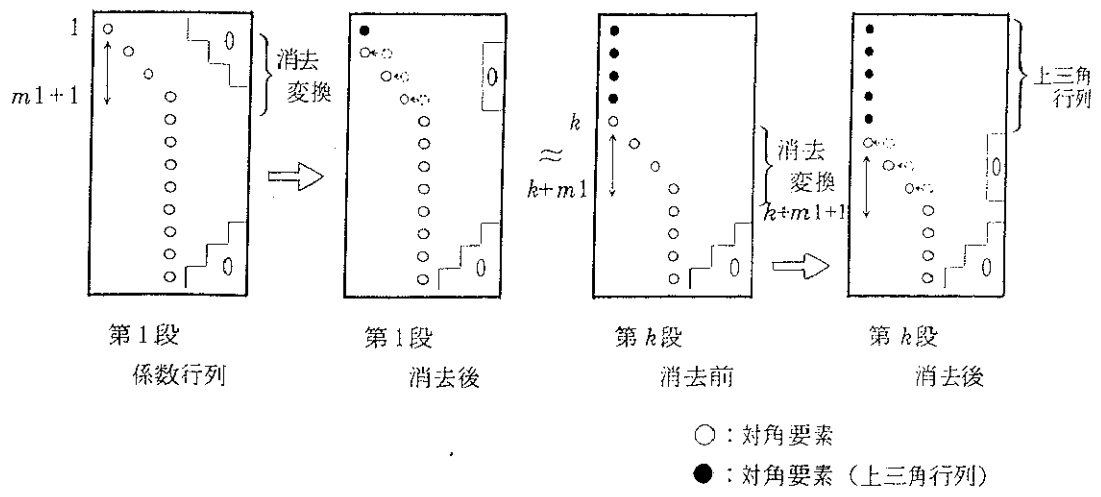


図2.5 R. S. Martin とJ. H. Wilkinson 流のデータ構造

となる。最初のm1行は第1列の要素を配列の上端(第0行)に合わせ、残りの行は帯の左端の要素を配列の上端に合わせ、対角要素を第m1列とする。MartinとWilkinsonの特殊ガウスの主要部分をプログラムふうに書けば

```
do i = k+1, min(k+m1,n)
   ac(i-k,k) = -a(0,i)/a(0,k)
   do j = k+1, min(k+2m1,n)
      a(j-k-1,i) = a(j-k,i)+ac(i-k,k)*a(j-k,k)
```

となる。消去後の値が1列分左にシフトされた結果、第k+1段の消去過程では

$a_{0,i}$ i = k+2, min(k+1+m1,n)

を0にすればよい。対応する右辺の前進消去過程は

```
do i = k+1, min(k+m1,n)
   b(i) = b(i)+ac(i-k,k)*b(k)
```

となり、メモリ参照が連続的になる。

データの格納方法を工夫したことにより添字が複雑になるので、実際のプログラムではループ内で不変な変数をスカラーを使って明示したり、第k段の消去過程で不変な配列領域を一次元配列を用いて明示したりする必要がある。その結果、プログラムは

```
do j = k+1, min(k+2m1,n)
   wk(j) = a(j-k,k)
do i = k+1, min(k+m1,n)
   ac(i-k,k) = -a(0,i)/a(0,k)
   t = ac(i-k,k)
   do j = k+1, min(k+2m1,n)
      a(j-k-1,i) = a(j-k,i)+t*wk(j)
```

となる。

メモリ参照の問題ではワーキングセットが重要であるが、MartinとWilkinsonの特殊ガウスでは配列を分離したことによって、ワーキングセットは

$(2m1+1)m1+m1 \approx 2m1^2$

で、帯ガウスの約2/3になる。前進消去過程のメモリ参照は上三角部分が(2m1+1)n、消去変換に用いたαを格納する部分がm1·n、合計は(3m1+1)nとなって帯ガウスと等しい。右辺の計算のメモリ参照は、前進消去過程がm1·n、後退代入過程が(2m1+1)n、合計は(3m1+1)nである。帯ガウスの場合には前進消去過程、後退代入過程とも(3m1+1)nなので、MartinとWilkinsonの特殊ガウスでは帯ガウスの約半分のメモリ参照で済み、しかもどの過程においてもメモリ参照が連続的になる。

このように右辺の計算を重視した算法について、同一の係数行列に対する右辺の計算をどのくらい繰り返すとCPU時間が有利になるかに注目する。Martinと

Wilkinsonの特殊ガウスは帯ガウスと演算量がほぼ同一でメモリ参照が改善されている。しかし、シフト操作のために左辺の計算が若干複雑になるため、左辺の計算に必要なCPU時間は帯ガウスのほうが短いが、右辺の計算に必要なCPU時間はメモリ参照の改善によってMartinとWilkinsonの特殊ガウスのほうが短いと考えられる。そのため方程式を繰り返して解いた場合、帯ガウスとMartinとWilkinsonの特殊ガウスのCPU時間が等しくなるような繰り返し回数が計算できる。この回数を平衡点と呼ぶ[22]。平衡点よりも繰り返し回数が多い場合はMartinとWilkinsonの特殊ガウスが有利となる。仮想メモリ方式のコンピュータをマルチジョブ環境で使用する場合には、CPU時間は実行に必要な時間の大小を決定するが、CPU時間の差がそのまま実行時間の差に現れるのではなく、実行時間にはシステムの環境に強く依存してCPU時間以上の差が現れる。

表2.1に要素並列ベクトルパイプライン方式のスーパーコンピュータS-820/80と汎用コンピュータM-880/310の結果を示す。以前の汎用コンピュータM260Hでは平衡点p_0が次元数の約5%から15%で、以前のベクトルパイプライン方式のスーパーコンピュータS-810/20では行列の次元が大きいときに次元数の10%程度だった[22]。ところが表2.1の結果のように[30]、最近のコンピュータではごく一部を除いてMartinとWilkinsonの特殊ガウスが速くなっている。したがって、MartinとWilkinsonの特殊ガウスはほとんどすべての場合に有効で、しかも次元数が大きい大規模計算のときにはその差が広がる。

(3)対称帯ガウス

係数行列が対称正定値行列の場合、部分軸選択が不要なことと消去の過程で対称性が保たれることから、一般にはコレスキー分解が用いられるが[46],[72]、対称ガウスを用いても容易に解くことができる[46],[23]。帯コレスキー分解は帯を保ったままで、対称正定値行列を$U^T DU$または$\tilde{U}^T \tilde{U}$に分解する算法である(Uは上三角行列、Dは対角行列)。古典的な正統コレスキー分解$A=\tilde{U}^T \tilde{U}$には平方根演算が必要なため、方程式を解く目的では改訂コレスキー分解(Modified Cholesky decomposition)を用いて$A=U^T DU$と分解するのが一般的である。改訂コレスキー分解を修正コレスキー分解ということもある[46]。帯行列に対する改訂コレスキー分解は

```
do j = 1, n
   do i = max(1,j-m1), j-1
      w_ij = a_ij - Σ u_ki w_kj
      u_ij = w_ij / d_i
   d_j = a_jj - Σ u_kj w_kj
```

となる。上三角行列の対角要素u_{jj}と対角行列の要素d_jの選び方には任意性がある

表2.1(a)　帯ガウスとMartin とWilkinsonの特殊ガウスの比較(s)　S-820/80

m1	n	帯ガウス	帯ガウス右辺	特殊ガウス	特殊ガウス右辺
50	2550	0.083	0.826	0.071	0.812
100	10100	0.621	3.622	0.484	3.579
200	40200	5.044	17.44	4.471	15.94

表2.1(b)　帯ガウスとMartin とWilkinsonの特殊ガウスの比較(s)　M-880/310

m1	n	帯ガウス	帯ガウス右辺	特殊ガウス	特殊ガウス右辺
50	2550	0.860	2.775	0.880	2.660
100	10100	14.77	21.96	14.62	20.97
200	40200	237.3	172.0	233.5	165.2

が、改訂コレスキー分解では$u_{jj}=1$とする。改訂コレスキー分解から得られたUとDを用いて$(\sqrt{D}U)^T(\sqrt{D}U)$とすれば正統コレスキー分解になる。

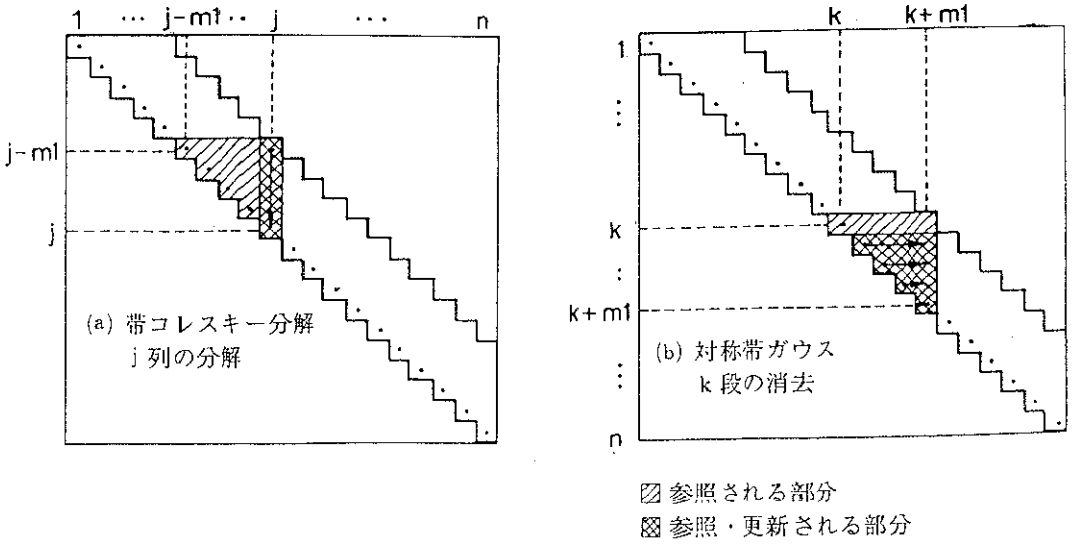


図２．６　メモリ参照の違い（m1=4, n=20）

スーパーコンピュータやパソコンなどでは乗算よりも除算が低速なことがあるので、除算 $1/d_i$ を乗算に直して高速化を図ることがある。ページスワップの影響が少なくするには、帯コレスキー分解の一番内側のループのメモリ参照を連続的にする。それには上三角行列Uを縦方向番地付けで格納するか、対称性を利用して下三角行列$L=U^T$についてのプログラムを作成してLを横方向番地付けで格納すればよい。右辺の計算を含めてメモリ参照を連続的にするためにはさらに改良が必要である。コレスキー分解は内積演算が中心なので、ストア命令の実行回数が少なく、高速化が可能だが、精度を得るためには演算順序に注意がいる。文献[23]によれば、汎用コンピュータでは帯コレスキー分解が対称帯ガウスよりも高速だが、要素並列ベクトルパイプライン方式のスーパーコンピュータでは対称帯ガウスが高速であり、また帯コレスキー分解に対しては高速化手法の適用が難しい。

対称帯ガウス[46],[23]は、帯ガウス[22]から部分軸選択を取り去り、対称性を利用して上(下)三角部分だけでガウスの消去法を行う。係数行列が正定値ならば軸選択が不要になることと、対称なら消去変換後も対称性が保たれるので[46]、消去範囲を上三角部分に限定すれば消去演算の回数は帯ガウスの約1/4になる。しかも消去変換に用いるαはa_{ik}と対称な位置にあるa_{ki}とa_{kk}から作ることができる。a_{ki}とa_{kk}はk段以降では更新されないからαを格納する必要がない。これから、対称帯ガウスのアルゴリズムは

```
do i = k+1, min(k+m1,n)
   t = -a_ki/a_kk
   do j = i, min(k+m1,n)
      a_ij = a_ij + t·a_kj
```

となる。消去後の上三角行列は$A=U^T DU$と分解したときのDUになる。連立一次方程式を解くには不要だが、コレスキー分解におけるUとDが必要ならば容易に作成できる。実際のプログラムではスカラーや一次元アレイを用いて高速化を図る。

対称帯ガウスの場合も帯ガウスと同様、与えられた帯行列を

$a'_{j-i,i} = a_{ij}$

と変換して、横方向番地付けで

a'(0:m1,n)

という (m1+1) ×nの配列に格納する。制限三項演算の回数は$m1^2n/2$回で帯ガウスの約1/4、必要なメモリ容量は(m1+1)×nワードで帯ガウスの約1/3になる。ワーキングセットは$m1^2$で、全体を通じたメモリ参照は前進消去過程、右辺の前進消去過程、後退代入過程ともm1·nであり、どの過程においても連続的である。

2.2 直接解法に対する高速化手法

アルゴリズムどおりのプログラムでもスーパーコンピュータを用いれば高速に実行される。しかし、スーパーコンピュータの性能を活用するにはなんらかの高速化手法の適用が必要である。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。文献[57]にはベクトルパイプライン方式のスーパーコンピュータ向けの高速化手法として

(a) ベクトル化制御行の利用
(b) ループ・アンローリング
(c) ベクトル長の増加
(d) コンパイラへのデータ構造の明示
(e) メモリ競合の防止

などがあげられている。(c)は消去範囲から自然に決まるし、(e)は連続的なメモリ参照ならば問題はない。(d)はプログラムを実際に書くときの注意事項で、(a)はコンパイラがうまく動かないときに使われるいわば「奥の手」であり、いずれも一般性に乏しい。ここでは(b)のループ・アンローリングを中心にメモリとベクトル演算器間のデータ転送に注意しながら高速化を図る。

最近のスーパーコンピュータは要素並列ベクトルパイプライン方式[53]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。このようなスーパーコンピュータで高速化を図るには、プログラムにアンローリングを施すのがよい。アンローリングはループ内の演算密度を上げ、並列に実行できる演算をふやして高速化を図る手法である[49],[48]。汎用コンピュータでは、アンローリングによってループオーバーヘッドが減少することにより高速化される。

最近のコンパイラは自動的にアンローリングを施して高速化を図ることもある(陰アンローリングとよぶ)。コンパイラによって最適なアンローリングがなされることが望ましいが、現時点でもコンパイラにはあまり期待できないので、複雑なプログラムにはあらかじめアンローリングを施してやる必要がある(陽アンローリングとよぶ)[49]。例えば、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。このように多様なコンピュータでの使用を前提とするなら、能力の劣るコンパイラでもプログラムが高性能になるようにすべきである。そこで「使用するコンパイラの違いによって大きな性能変化がないようなプログラム」を目的として、陽アンローリングを適用する。しかし、パイプラインの制御が途切れることなく実行させるためには、少しアレイを大きくとるなどの配慮が必要であり、そう簡単なことではない。しかも、プログラムの保守を考えると、アルゴリズムに忠実で読み

易いプログラムにしておく必要がある。結果として陽アンローリングはプログラムを複雑にするが、得られたプログラムはスーパーコンピュータと汎用コンピュータの両方で高速実行が可能となる。

アンローリングに際して重要な点はメモリとレジスタ間の無駄なデータのやりとりをなくすことで、この点ではJ.J.Dongarraらの方針と同じである[9],[11]。彼らは線形数値計算でよく使われる基本的な演算(例えばSAXPY)を高速化しているが[10],[1]、ここでは帯行列を係数とする連立一次方程式の解法全体を対象とすることによって、制限三項演算

$$a_i = a_i + t \cdot b_i$$

を

$$a_i = a_i + t \cdot b_i + u \cdot c_i$$

とするようなアンローリングが可能になった。影響がLU分解全体に及ぶため作業は大変なものの、メモリの参照回数を減らせるなどの特徴があり、高速化の割合も大きい。以下、スーパーコンピュータと汎用コンピュータの両方に好ましい高速化手法を帯行列に対する直接解法に適用した結果について述べ、その問題点を明らかにする。

ガウスの消去法は3重ループ内の演算なので、段k、行i、列jの3通りのループについてのアンローリングが可能である。要素並列ベクトルパイプライン方式のスーパーコンピュータでは同時実行が可能なベクトル演算器数と演算器にデータを供給するパイプライン数の関係が問題となる。汎用コンピュータではメモリとレジスタ間のロード･ストアとループの判定回数など、演算以外の命令数が主な問題となる。連続的なメモリ参照なのでページスワップの影響は考えなくても良いし、直接解法なので消去演算の数は一定である。議論を簡単にするためアンローリングは2重に限定し帯ガウスのみについて述べる。

要素並列ベクトルパイプライン方式のスーパーコンピュータでは、帯ガウスの主要部

```
do k = 1, n
   do i = k+1, min(k+m1,n)
      do j = k+1, min(k+2m1,n)
         a(j-i,i) = a(j-i,i) + t*wk(j)                    (2.1)
```

の最内側ループは

VLD(ロード)
VLD
VMAD(積和演算)
VSTD(ストア)

というベクトル命令になる[28]。例えば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 またはチャイムとよぶ[53])で実行可能であり、一連の処理は$2\Delta t$のうちに行われる。

段kについてのアンローリングを施して2段同時[49],[48]にすると

```
do k = 1, n, 2
   do i = k+1, min(k+m1,n)
      do j = k+1, min(k+2m1,n)
         a(j-i,i) = a(j-i,i)+t*wk(j)+t1*wk1(j)                    (2.2)
```

となる。最内側ループは

```
VLD
VLD
VMAD
VLD
VMAD
VSTD
```

というベクトル命令になり、S-820/80では一連の処理が$2\Delta t$のうちに行われる。(2.1)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ったことから、全体の実行時間は(2.1)の約1/2になる。2段同時では1回のストア命令に対して2回の積和演算が実行され、ストアが約1/2、ロードが約3/4になる。しかもいちばん外側のループkの実行回数が半減するので、メモリの総参照ページ数が半減する。そのため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。段に関するアンローリングは影響がプログラム全体に及ぶため、汎用性を失わないプログラムとしては2段程度が限度である。

2段同時に対応した右辺の前進消去過程は、2重ループの外側のループに関するアンローリングになっている。右辺の前進消去過程に最内側ループのループ長を減少させずにアンローリングを施すためには、左辺の消去過程を2段同時にする必要がある。逆に左辺の消去過程を2段同時にすると、自動的に対応する右辺の計算も高速化される。右辺の後退代入過程は左辺の処理とは独立にアンローリングが可能である。

行iについてのアンローリングを施して2行同時[49],[48]にすると

```
do k = 1, n
   do i = k+1, min(k+m1,n), 2
      do j = k+1, min(k+2m1,n)                                    (2.3)
         a(j-i,i) = a(j-i,i)+t*wk(j)
         a(j-i-1,i+1) = a(j-i-1,i+1)+u*wk(j)
```

となる。最内側ループは

```
VLD
VLD
```

VMAD
VSTD
VLD
VMAD
VSTD

というベクトル命令になり、一連の処理が3Δtのうちに行われる。(2.1)の2倍の演算が1.5倍の時間に行われるため、全体の実行時間は(2.1)の約2/3になる。2行同時では1つのループ内に演算とストア命令を2つずつ入れただけなので、ロードが約3/4になるだけである。2段同時と比べるとループ内の演算密度に比べて必要なパイプライン数が多いため、小さい倍率のアンローリングでパイプラインが不足する。最近のコンパイラは2重ループの外側のループについては陰アンローリングを行うから、帯ガウスではこれが行に関するアンローリングとなる。したがって、コンパイラによる陰アンローリングが期待できるのは行について、すなわち(2.3)相当のみである。

列jについてのアンローリングを施した2列同時[49],[48]

```
do k = 1, n
   do i = k+1, min(k+m1,n)
      do j = k+1, min(k+2m1,n), 2                    (2.4)
         a(j-i,i) = a(j-i,i)+t*wk(j)
         a(j+1-i,i) = a(j+1-i,i)+t*wk(j+1)
```

は、ループ長が半減し(c)の条件を満たさないため論外であろう。

2段同時に対して行について2重のアンローリングを行ったのが2段2行同時である[49],[48]。

```
do k = 1, n, 2
   do i = k+1, min(k+m1,n), 2
      do j = k+1, min(k+2m1,n)                       (2.5)
         a(j-i,i) = a(j-i,i)+t*wk(j)+t1*wk1(j)
         a(j-i-1,i+1) = a(j-i-1,i+1)+u*wk(j)+u1*wk1(j)
```

となる。この最内側ループ(2.5)が

VLD
VLD
VMAD
VLD
VMAD
VSTD
VLD
VMAD
VMAD

VSTD

というベクトル命令になる。S-820/80では、これらの命令をどう並べ換えても4Δt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く、2段同時とほぼ同様である。ところが、チェイニングによってベクトル命令のセットアップが並列に実行されるため、いくらかの性能向上が望まれる[45],[53]。しかし4行同時は5ベクトルロード、4ベクトルストアなのに比べると、2段2行同時は4ベクトルロード、2ベクトルストアなので必要なデータ移動命令がはるかに少ない。これから2段同時と2行同時を組み合わせることでロード･ストアパイプラインを有効に活用できることがわかる。

表2.2にアンローリングによる命令の実行回数の変化を示す[28]。

(1) 帯ガウス

2段同時は段数kについてのアンローリングのため、次元が奇数のときに見かけ上のランク落ちを生じないよう

$$a_{n+1,n+1} = 1$$

とする。また、k+m1+1行とk+2m1+1列にはk+1段の消去は行われるがk段の消去は行われないため、2段ずつ同時に消去を行うにはk段の消去が行われない状態を作る必要がある。そのため、右帯半幅を2m1+1,左帯半幅をm1+1とし、

$$a_{i,i-m1-1} = 0 \qquad (i=1,n+1)$$
$$a_{j,j+2m1+1} = 0 \qquad (i=1,n+1)$$

にすべて0を入れておくことによって、すべての消去演算を2段同時に実行する。

2行同時ではm1が奇数の場合に問題が起きる。本来k段とk+1段ではk+m1+1行までしか消去が行わないにもかかわらず、2行ずつ同時に消去を行うためk+m1+2行まで消去が及んでしまう。2段同時のように左帯半幅をm1+1として0を入れればk+m1+2行にはk+1段の消去はなされないが、k+m1+2行にk段の消去を行わないためには左帯半幅をさらに増やしてm1+2として、

$$a_{i,i-m1-2} = 0 \qquad (i=1,n+1)$$

とする必要がある。ところが横方向番地付けなので、いちばん左側の要素と次の行のいちばん右側の要素は同一視できる。結局、2段同時、2段2行同時に対応して、すべての演算を一様に実行するには、帯幅を拡張して

$$i-m1-1 \leqq j \leqq i+2m1+1$$

とすればよい。

表2.2(a)　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
帯ガウス	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行同時	2	3	2	$n^3/2$	$n^3/3$	$n^3/6$
2段2行同時	4	4	2	$n^3/3$	$n^3/6$	$n^3/12$

表2.2(b)　右辺の命令数

			前進消去			後退代入	
	演算密度1	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
帯ガウス	1	n^2	$n^2/3$	$n^2/2$	n^2	$n^2/3$	$n^2/2$
2段同時	2	$3n^2/4$	$n^2/6$	$n^2/4$	$3n^2/4$	$n^2/6$	$n^2/4$

(2) R.S.MartinとJ.H.Wilkinsonの特殊ガウス

MartinとWilkinsonの特殊ガウスのアンローリング方法も、基本的には帯ガウスと同じである[49](前述したように、R.S.MartinとJ.H.Wilkinsonの特殊ガウスをMartinとWilkinsonの特殊ガウスと略称する)。2段同時[49],[48]のプログラムの主要部分は

```
do k = 1, n, 2
   do i = k+2, min(k+1+m1,n)
      do j = k+2, min(k+1+2m1,n)
         a(j-k-2,i) = a(j-k,i)+t*wk(j)+t1*wk1(j)                    (2.6)
```

2段2行同時[49],[48]のプログラムの主要部分は

```
do k = 1, n, 2
   do i = k+2, min(k+1+m1,n), 2
      do j = k+2, min(k+1+2m1,n)
      a(j-k-2,i) = a(j-k,i)+t*wk(j)+t1*wk1(j)
      a(j-k-2,i+1) = a(j-k,i+1)+u*wk(j)+u1*wk1(j)                   (2.7)
```

となる。Martin とWilkinsonの特殊ガウスではシフト操作が本質的であるため、簡単なアンローリングを行ったのでは無関係な要素に副作用を及ぼす可能性が大きい。リストベクトルを使えばこのような問題は起こらないが、リストベ

クトルは間接的なメモリ参照のために汎用コンピュータ、スーパーコンピュータの双方にとって好ましくない。

2段同時では、2段分の消去変換を同時に作用させるため2列分のシフトが必要である。同時に2列分シフトするため、帯幅は列方向に1列余分に必要となる。また次元の奇数·偶数に関係なく第n+1行を消去する可能性があるため、行についても1行余分にとっておく必要がある。第n+1行の値をあらかじめ与え、消去の過程を通じてそのままの値に保つことも困難なので、次元が奇数のときに見かけ上のランク落ちを生じないようにするため、プログラムの終了時にランク落ちを表す変数を補正する。

MartinとWilkinsonの特殊ガウスのプログラムの2段同時では、シフト量が異なることと、第k+1段の部分軸選択の結果、方程式がどこと入れ換えられるかによっても消去のしかたが異なってくることから、場合分けが必要になる[22]。

2行同時にする際の問題は、行iに関するループのループ長が奇数の場合に起きる。MartinとWilkinsonの特殊ガウスではシフト操作が本質的なので、シフトまでを含めた無変換を作り出さない限り1行増やすことは許されない。したがって行に関するループのループ長が奇数の場合は、最後の1行を除いた残りを2行ずつを同時に消去し、最後は1行だけで消去する。ここでも場合分けが必要である。

2段2行同時の要点は、2段同時のループ(2.6)を2段2行同時となるようにアンローリングすることである。一般には行に関するループのループ長

loopi = min(k+1+m1,n)-(k+2)

などは

(k+1+m1)-(k+2)

が偶数であっても、min(k+1+m1,n)があるため段数kの値によっては奇数になりうる。そのため実際のloopiが偶数·奇数であるかを判定して場合分けをする必要が生じる。

loopiが奇数の場合には、最後の1行を特別扱いして

```
do k = 1, n, 2
   do i = k+2, min(k+1+m1,n)-1, 2
      do j = k+2, min(k+1+2m1,n)
         a(j-k-2,i) = a(j-k,i)+t*wk(j)+t1*wk1(j)
         a(j-k-2,i+1) = a(j-k,i+1)+u*wk(j)+u1*wk1(j)          (2.8)
   do i = min(k+1+m1,n), min(k+1+m1,n)
      do j = k+2, min(k+1+2m1,n)
         a(j-k-2,i) = a(j-k,i)+t*wk(j)+t1*wk1(j)
```

とすれば良いはずだが、これは正しくない。FORTRAN77とかPascalのようなプログラミング言語のDOループには始点、終点のチェック機能が備わってお

り、DO文を単なる繰り返しとみなすことはできない。たとえば、k=n のとき始点の値 k+2=n+2は終点の値nより大きいため (2.6)、(2.7) のループは何も実行されない。ところが(2.8)の後半のループ(あるいはループを取り去ってiの代わりにmin(k+1+m1,n) を代入したもの) は必ず実行されるため、アンローリングの前後でプログラムの同値性が保たれない。この問題はループ長loopi が負になるとき、すなわち元々のループがチェックのみで実行されない場合に生じる。このような場合はloopi が奇数でも、偶数扱いとして何も実行されないようにする必要がある。したがってMartinとWilkinsonの特殊ガウスの場合は機械的な陽アンローリングであってもそう簡単には行えない。

(3) 対称帯ガウス

帯改訂コレスキー分解$U^T DU$[46],[72]のアルゴリズム主要部は

$$u_{kj} = a_{kj} - \Sigma u_{ik} d_i u_{ij} / d_k$$

のような3つの項の積和になるため、2つのベクトルの要素毎の積を作ってから内積をとらなければならない。一方、正統コレスキー分解$\tilde{U}^T \tilde{U}$ではアルゴリズムの主要部が完全な内積演算になっている。改訂コレスキー分解の右辺の計算は除算がベクトル除算として高速に実行される。このため、スーパーコンピュータでは、左辺を重視すれば正統コレスキー分解、右辺を重視すれば改訂コレスキー分解がよいことになる。

コレスキー分解のような内積型の演算については

```
do j = 1, n
   do i = max(1,j-m1), j-1, 2
      w_ij = a_ij - Σ u_ki w_kj
      w_i+1,j = a_i+1,j - Σ u_k,i+1 w_kj
      u_ij = w_ij / d_i
      u_i+1,j = w_i+1,j / d_i+1
   dj = a_jj - Σ u_kj w_kj
```

のように、内積の並列実行が可能なだけで、WをDUに分解する部分があるため全体を通したアンローリングが困難である。命令の実行回数がそれほど減らせないうえ、スーパーコンピュータの内積演算は積和演算に比べて並列実行に難があり、しかも遅いとされている[26]。

対称帯ガウス[46],[23]に対するアンローリングも基本的には帯ガウスと同じだが、対称帯ガウスの場合は部分軸選択がないのではるかに簡単にできる。2段同時では、k+1+m1 行とk+1+m1 列にはk+1 段の消去変換だけを作用させ、その他にはk段とk+1 段の消去変換を同時に作用させる。場合分けをするとプログラムが複雑になるので、多少演算量は増えるが余分な領域をとってαを0とした無

変換を作り、常に2段ずつ同時に消去を行なう。それには列方向に帯幅を1つだけ広げて$a_{k,k+1+m1}=0$としておけばよい。また次元の奇数･偶数に関わらず第n+1行を消去する可能性があるため、行についても1行分余計に領域をとっておく。次元が奇数のときに見かけ上のランク落ちを生じないようにするため$a_{n+1,n+1}$には1を与える。

2段2行同時のアンローリングを施す際の問題点は、上三角行列の領域だけを扱っていることと、行iに関するループのループ長が奇数になることから生じる。上三角行列の領域だけを扱うことから、i行の対角要素a_{ii}に対する計算だけは特別に行い、i+1列からは2行ずつ同時に消去する。行に関するループのループ長が奇数のときは、消去領域からはみ出す1行の消去を無変換にするため、列方向にさらに帯幅を1つ広げて$a_{k,k+2+m1}=0$とする。k+m1+2行に対するk段の消去変換は$a_{k,k+2+m1}$、k+1段の消去変換は$a_{k+1,k+2+m1}$から決定されるが、これらは2列分余計に領域をとって0を与えておいた部分なので行に関するループのループ長が奇数であっても他の行の要素に影響は及ばない。

2.3 直接解法に対する高速化手法の効果

要素並列ベクトルパイプライン方式のスーパーコンピュータとしてS-820/80、汎用コンピュータとしてM-880/310、中間的なコンピュータとしてIAP(Integrated Array Processor)付のM-682Hを用いて実測を行った。計算はすべて倍精度で行った。結果を表2.3～表2.8にCPU時間を示す[30]。帯ガウスをBGLUx、MartinとWilkinsonの特殊ガウスをBHLUx、対称帯ガウスをBSLUxで表す。x=1は原形、x=2は2段同時のアンローリング、x=4は2段2行同時のアンローリングを意味する。右辺の計算は、BGSLVy、BHSLVy、BSSLVyで表し、y=1は原形、y=4は2重ループの外側ループについてのアンローリングを意味する。このような線形計算に対するスーパーコンピュータS-820/80のピーク性能は2GFLOPSである。

精度を調べるために残差ベクトルを計算し、残差ベクトルの1-ノルムと2-ノルムで比較を行った。帯ガウスとMartinとWilkinsonの特殊ガウスは全く同一だった。帯コレスキー分解と対称帯ガウスは、必要なメモリ容量と得られた解の精度に相違はなく、解の精度は部分軸選択を行う帯ガウスとも同等だった。これから、拡散方程式を離散化して得られる対称正定値帯行列を係数とするときは、理論だけではなく、現実的に部分軸選択は必要ないことが明らかになった。

スーパーコンピュータS-820/80では、帯ガウスとMartinとWilkinsonの特殊ガウスに8重、それぞれの2段同時に行について4重の陰アンローリングが施されている。スーパーコンピュータの場合、帯ガウスは2段同時、2段2行同時にすると約60%、Martin とWilkinsonの特殊ガウスは2段同時、2段2行同時にすると約

表2.3　帯行列に対するプログラムの性能　　S-820/80

s (GFLOPS)	BGLU1	BGLU2	BGLU4	BHLU1	BHLU2	BHLU4	BSLU1	BSLU2	BSLU4
M1＝50, N＝2550	0.083 (0.30)	0.056 (0.45)	0.053 (0.47)	0.071 (0.35)	0.049 (0.51)	0.050 (0.50)	0.082 (0.07)	0.039 (0.16)	0.037 (0.17)
M1＝100, N＝10100	0.621 (0.64)	0.379 (1.06)	0.377 (1.07)	0.484 (0.83)	0.345 (1.16)	0.347 (1.16)	0.655 (0.15)	0.298 (0.33)	0.251 (0.40)
M1＝200, N＝40200	5.044 (1.27)	3.867 (1.66)	3.842 (1.67)	4.741 (1.35)	3.994 (1.61)	4.010 (1.60)	5.475 (0.29)	2.661 (0.60)	2.132 (0.75)

表2.4　右辺の計算性能(100回繰り返し)　　S-820/80

s (GFLOPS)	BGSLV1	BGSLV4	BHSLV1	BHSLV4	BSSLV1	BSSLV4
M1＝50, N＝2550	0.826 (0.09)	0.633 (0.12)	0.812 (0.09)	0.640 (0.11)	0.811 (0.06)	0.570 (0.08)
M1＝100, N＝10100	3.622 (0.16)	2.976 (0.20)	3.579 (0.16)	2.870 (0.21)	3.543 (0.11)	2.569 (0.15)
M1＝200, N＝40200	17.44 (0.27)	13.89 (0.34)	15.94 (0.30)	13.46 (0.35)	15.78 (0.20)	11.91 (0.26)

表2.5　帯行列に対するプログラムの性能　　M-880/310

s (MFLOPS	BGLU1	BGLU2	BGLU4	BHLU1	BHLU2	BHLU4	BSLU1	BSLU2	BSLU4
M1＝50, N＝2550	0.860 (29)	0.707 (36)	0.692 (36)	0.880 (28)	0.713 (35)	0.698 (36)	0.242 (26)	0.199 (31)	0.190 (33)
M1＝100, N＝10100	14.77 (27)	11.26 (35)	11.00 (36)	14.62 (27)	11.29 (35)	10.96 (36)	3.560 (28)	2.912 (34)	2.819 (35)
M1＝200, N＝40200	237.3 (27)	178.2 (36)	175.7 (36)	233.5 (27)	180.6 (35)	175.0 (36)	58.95 (27)	46.21 (34)	44.75 (35)

70%、陰アンローリングの行われない対称帯ガウスでは、2段同時、2段2行同時にすると約40~50%のCPU時間で済む。右辺の計算は、帯ガウスとMartinとWilkinsonの特殊ガウスが約80%、対称帯ガウスが約70%のCPU時間で済む。帯

表2.6　右辺の計算性能(100回繰り返し)　　M-880/310

s (MFLOPS)	BGSLV1	BGSLV4	BHSLV1	BHSLV4	BSSLV1	BSSLV4
M1＝50, N＝2550	2.775 (27)	2.450 (31)	2.660 (28)	2.348 (32)	1.822 (27)	1.566 (32)
M1＝100, N＝10100	21.96 (27)	19.79 (30)	20.97 (28)	18.60 (32)	14.42 (28)	12.69 (31)
M1＝200, N＝40200	172.0 (28)	152.2 (31)	165.2 (29)	148.4 (32)	115.7 (27)	102.1 (31)

表2.7　帯行列に対するプログラムの性能　　M-682H IAP

s (MFLOPS)	BGLU1	BGLU2	BGLU4	BHLU1	BHLU2	BHLU4	BSLU1	BSLU2	BSLU4
M1＝50, N＝2550	0.458 (55)	0.437 (58)	0.430 (59)	0.793 (32)	0.763 (33)	0.753 (33)	0.204 (31)	0.209 (30)	0.206 (30)
M1＝100, N＝10100	8.824 (45)	7.240 (55)	7.227 (55)	12.95 (31)	12.96 (31)	13.27 (30)	2.087 (48)	2.105 (47)	2.082 (48)
M1＝150, N＝22650	52.93 (38)	38.25 (53)	38.05 (53)	73.95 (27)	73.28 (27)	74.21 (27)	8.747 (58)	9.122 (55)	8.957 (56)

表2.8　右辺の計算性能(100回繰り返し)　　M-682H IAP

s (MFLOPS)	BGSLV1	BGSLV4	BHSLV1	BHSLV4	BSSLV1	BSSLV4
M1＝50, N＝2550	3.302 (23)	3.407 (22)	3.334 (22)	3.264 (23)	2.172 (23)	2.193 (23)
M1＝100, N＝10100	23.30 (26)	23.98 (25)	23.46 (25)	23.18 (26)	15.47 (26)	15.80 (25)
M1＝150, N＝22650	73.72 (27)	77.28 (26)	75.87 (26)	75.03 (27)	48.88 (27)	50.65 (26)

行列の場合は最内側ループのループ長が2m1と比較的短いことを考えれば、2段同時、2段2行同時などのアンローリングは有効であるといえる。

アンローリングの結果、帯ガウスとMartinとWilkinsonの特殊ガウスの平衡点も変化してくる。原形ではメモリ参照の差が大きかったためにほとんど常にMartinとWilkinsonの特殊ガウスが有利だったが、2段同時などのアンローリン

グの効果が帯ガウスに著しいため、帯ガウスに有利な部分も生じてくる。しかし、全体的な傾向として繰り返し回数が多いときにはMartin とWilkinsonの特殊ガウスが有利なことに変わりはない。

対称帯ガウスは陰アンローリングが施されないために原形が非常に遅い。そのため、陽アンローリングの効果は大きいが、ピーク性能と比べるとその値は小さい。その原因は最内側ループのループ長が短いためで、対称帯ガウスが最大でm1なのに対して、帯ガウスとMartin Wilkinsonの特殊ガウスは2m1である。しかも対称帯ガウスでは最大ループ長こそm1だが、上三角部分しか消去を行わないため消去行がk行から遠ざかるにつれてループ長が減少して平均ループ長はm1/2になる。これは帯ガウスやMartin とWilkinsonの特殊ガウスの1/4である。

帯ガウスには8重の陰アンローリングが行われるため、演算量が対称帯ガウスの4倍あるのにもかかわらず、スーパーコンピュータでは対称ガウスのCPU時間よりも短いことがあるが、陽アンローリングを施すとそのような逆転は起こらない。

一般に、汎用コンピュータでは複数の演算を同時には実行できない。この場合、消去演算の回数は一定なので、主としてロード·ストア命令と分岐命令などの演算以外の命令の実行回数が減少することで高速化が図られる。

ロード·ストア命令(LD、STD)の実行回数が少なければ、メモリとレジスタ間で無駄なデータの移動を行わずに演算が行われる。分岐命令(BCT)の実行回数が少なければ、ループ内でより多くの演算が行われる。汎用コンピュータでもパイプライン処理が行われているため、分岐命令などのパイプライン処理が中断する命令は少ないに越したことはない[24]。

表2.2に示したように、2段同時ではロード·ストア命令の実行回数が減少し、2行同時ではロード命令の実行回数だけが減少する。分岐命令の実行回数はどのアンローリングの場合も半減する。2段2行同時にするとロード·ストア命令、分岐命令の実行回数ともさらに減少する。アンローリングによって副次的な処理は必要になるが、ロード·ストア命令の実行回数が減少し、ページ参照も半減するため2段2行同時にアンローリングしたプログラムは汎用コンピュータでも高速に実行できる。

その結果、汎用コンピュータM-880/310では2段同時、2段2行同時とも70~80%程度のCPU時間で済む。右辺の計算を2段同時にすると約90%のCPU時間となる。

IAP(Integrated Array Processor)付のM-682Hでは、アンローリングの効果は帯ガウスの左辺を除くとほんのわずかであるか、逆に遅くなるかである。帯ガウスだけにアンローリングが効果的というのは異常である。以前は、IAPのよう

に演算結果を常にメモリへ戻す場合にも2段同時などのアンローリングは有効だったことから、これはコンパイラの問題と考えられる。

このように、適切なアンローリングの程度は機種、あるいはコンパイラのバージョンによって異なる。多様なコンピュータでの使用を前提にするなら、機能が劣るコンパイラでも高速に実行できることが必要であり、この点で人手による陽アンローリングは欠かせない。しかし、より多重のアンローリングは高速化のメリットが少ない割にプログラムが複雑になる。一方アンローリングなしのプログラムはロード·ストアの実行回数や主記憶の参照量が多いため、スーパーコンピュータと汎用コンピュータの双方で問題である。高速化と保守の問題を併せて考えれば、並列に実行される演算に対して必要なロード·ストアが少ない2段2行同時程度のアンローリングが最適であろう。スーパーコンピュータ用のコンパイラは2重ループの外側のループに対して陰アンローリングを施すこともあるが、コンパイラに2段同時のアンローリングを期待するのは難しい。そのためスーパーコンピュータでコンパイラによる行についての陰アンローリングが期待できるなら、コンパイラには難しい2段同時の陽アンローリングだけを施すのがよい。2段同時のアンローリングは効果が大きいので充分な価値がある。また、これらの高速化手法は精度を悪化させず、しかもスーパーコンピュータと汎用コンピュータの双方で良い結果が得られる。

2.4 ストリップマイニングの効果

S-3800のような最近の要素並列ベクトルパイプライン方式のスーパーコンピュータでは[53]、演算パイプラインに見合うだけのロード·ストアパイプラインが用意されていない[38]。そのかわり、レジスタが比較的多めに用意されているので、2重のループまでを対象にしたロード·ストアの効率化ができればさらに高速化できる。このための方法として、最内側ループをベクトルレジスタの大きさに限定し、データをベクトルレジスタに常駐させることが行われる。この手法をストリップマイニングという(小片化ともいう)[66],[53],[9]。

例えば要素並列ベクトルパイプライン方式のスーパーコンピュータでは、帯ガウスの主要部

```
do k = 1, n
   do i = k+1, min(k+m1,n)
      do j = k+1, min(k+2m1,n)
         a(j-i,i) = a(j-i,i) + t*wk(j)
```

が

```
do k = 1, n
   do i = k+1, min(k+m1,n)
```

VLD(ロード)
VLD
VMAD(積和演算) (2.9)
VSTD(ストア)

のように実行される。ここでwkの部分は2重のループを通じて不変である。これを2重ループの外へ追いだして、wkをレジスタに1回だけコピーして繰り返し利用できるなら、主要部は

do k = 1, n
  VLD(ロード); レジスタ上に保持
  do i = k+1, min(k+m1,n)
    VLD
    VMAD(積和演算) (2.10)
    VSTD(ストア)

となる。積和演算とストアの回数は同じだが、ロードの回数は(2.9)では2n·m1、(2.10)ではn+n·m1となる。(2.10)ではn·(m1−1)回のベクトルロードが不要になると同時に、(2.9)では2単位時間(2Δt)かかっていた最内側ループがΔtで実行できるようになる。そのためには最内側のループ長がベクトルレジスタよりも小さいことが必要である。

現在の要素並列ベクトルパイプライン方式のスーパーコンピュータのベクトルレジスタサイズは512ダブルワード(4KB)なので[38]、小片の大きさは一般の帯行列に対しては帯半幅256以下、対称正定値帯行列に対しては帯半幅512以下の問題に相当する。同一の問題を複数のプログラムによって確認することも多いので、高速化のために帯半幅を256以下に限定する。もともと対象にしていた問題で帯半幅を250にすると元数はn＝503×250、1.25×10^5になる。このときに必要な領域は対称なら2.4×10^8、一般の帯行列ならば7.2×10^8ダブルワードになる。これは、それぞれ240MB、720MBであり、現時点ではほとんど利用不可能な値である。したがって、特別な計算環境が利用できる場合を除いて、帯半幅を256に限定しても大きな影響はない。これによって、大幅な性能向上が得られるなら、妥当な制約といえよう。

このように帯行列に対するガウスの消去法系統の算法では、帯半幅を制限するだけで自動的にストリップマイニングをしたことになるが、逆に、なんらかの方法で行に関するループを分割するのは難しい。

次にストリップマイニングの効果が著しくなるのはどのようなときかを考える。レジスタ常駐が有効なのはロード命令だけだから、まず1ストア命令に対するロード命令の実行回数が多いk段同時のアンローリングを考える。1単位時間にロード·ストアを合わせてp、積和演算をqできるスーパーコンピュータなら、CPU時間の短縮率の概算値は

レジスタ常駐/常駐なし

$= \max(2/p,k/q)/\max((k+2)/p,k/q)$

となる。S-3800の場合、p=q=2なので、分子はk≧2のときには第2項、k=1の

段数	VMAD	常駐なし VLD	VSTD	レジスタ常駐 VLD	VSTD
1	1	2	1	1	1
2	2	3	1	1	1
～	～	～	～	～	～
k	k	k+1	1	1	1

ときは第1項、分母は第1項のみがいきる。k≧2のときの短縮率の概算値はk/(k+2)になる。この値はkについての減少関数で、k=2のときに短縮率の概算値が最小値をとることから、2段同時に対してレジスタ常駐の効果が大きいことがわかる。また、ハードウエアの性能がp+2>qのとき、ガウスの消去法をk段同時(k=1を含む)に実行することを考える。この場合は常にロード･ストアが不足するため、演算パイプラインを完全に稼働させるためにはデータのレジスタ常駐が必要になる。これらロード･ストア命令の減り方、レジスタ常駐の効果からいっても、帯に対するガウスの消去法系統の解法では2段同時のアンローリングに注目するのがよい。

そこで実測値からその効果を検証する。ところが現在、S-3800用のコンパイラでは、帯ガウスとMartin とWilkinsonの特殊ガウスの2段同時のプログラムに対しては行に関する4重の陰アンローリングが施されるため、厳密な評価ができない。

そこで、同様な見積もりによって評価を行う。(2.2)に示した2段同時

```
do k = 1, n, 2
   do i = k+1, min(k+m1,n)
      do j = k+1, min(k+2m1,n)
         a(j-i,i) = a(j-i,i)+t*wk(j)+t1*wk1(j)
```

に4重の陰アンローリングが施されている場合、レジスタ常駐なしの最内側ループは

VLD	a(j-i,i)に対して 4	
	wk(j) とwk1(j)に対して2	
VSTD	a(j-i,i)に対して 4	
VMAD	8	(2.11)

となるはずである。2重ループ内で不変なデータのレジスタ常駐が図られると

VLD	a(j-i,i)に対して 4

VSTD　a(j-i,i)に対して4
VMAD　8　(2.12)

となるはずである。S-3800では(2.11)が(4+2+4)/2=5(Δt)、(2.12)が(4+4)/2=4(Δt)となる。レジスタ常駐を図ることによって、5Δtが4Δtになることから2段同時のプログラムでは約25%の高速化が期待できる。行に関して8重に陰アンローリングされた帯ガウスとMartinとWilkinsonの特殊ガウスを考えると、同様な議論によって9Δtが8Δtになることから、原形のプログラムではほとんど高速化が期待できない。実測結果を表2.9に示す。

MartinとWilkinsonの特殊ガウスの2段同時のプログラムでは期待どおりの結果が得られているが、帯ガウスは値、高速化率とも予想を下まわっている。S-3800は演算装置のピーク性能が8GFLOPS、演算装置とデータをやりとりするロード·ストアパイプラインのピーク性能が4GFLOPSである。帯ガウスはロード·ストアパイプラインの制約によるピーク性能の半分をも越えていないので、実際にレジスタ常駐が行われているかどうかは疑問である。コンパイラがレジスタ常駐を行うための条件が現時点のマニュアルに明確には記述されていないが、少なくともベクトル化指示文によってループ長が512以下であることを明示する必要がある。このような指示は他のコンピュータに対しては何の影響も及ぼさない。

レジスタ常駐は効果があることは明らかになったが、陰アンローリングの程度を含めて、陰アンローリングとレジスタ常駐の間にどのような関係があるかは、陰アンローリングなし/レジスタ常駐といったデータが測定できないため、はっきりしたことはわからない。また、性能はベクトル演算装置の立ち上がり時間の大小などにも大きく依存するため、純粋に理論的な考察はほとんど無意味であり、いくつかの実測値からここで述べたような簡単なモデルを作成し、よりよい高速化手法を選択するのがユーザに許された選択肢である。

この点で、これらのプログラムに対する高速化は充分な性能が得られている。したがって、帯行列を係数とする連立一次方程式の直接解法のアルゴリズムとしては帯ガウス、同一の係数行列に対して右辺の計算を繰り返す場合にはR.S.MartinとJ.H.Wilkinsonの特殊ガウス、係数行列が対称正定値ならば対称帯ガウスを使用するのが良い。これらのプログラムに対して人手によって2段のアンローリングを施し、コンパイラに行に関する陰アンローリングとレジスタ常駐を行わせたプログラムは、精度·性能ともスーパーコンピュータの性能を充分に活かしている。

表2.9　S-3800におけるストリップマイニングの効果

s:秒 (GFLOPS)	レジスタ常駐 あり	レジスタ常駐 なし
帯ガウス BGLU1	3.013 (1.07)	2.074 (1.07)
帯ガウス2段同時 BGLU2	1.188 (1.82)	1.232 (1.76)
特殊ガウス BHLU1	1.111 (1.95)	1.041 (2.08)
特殊ガウス2段同時 BHLU2	0.761 (2.85)	0.899 (2.41)

3. 反復解法

3.1 いろいろな反復解法

ガウスの消去法に代表される直接解法では、行列の性質によらずほぼ一定の演算量で厳密解が求められた。一方、反復解法は第1近似解から始めて、より良い近似解を作ることを繰り返して精度を高める解法である。反復解法では、直接解法よりも速く実用上十分な近似解が得られることもあるが、最悪の場合には収束しない可能性がある。問題によって反復解法の収束特性は異なるため、反復解法を利用する場合は解こうとしている問題に合った解法を用いる必要がある。特に大規模な疎行列を対象とした場合には、メモリの節約のために反復解法を使わざるをえないので、収束が速く安定な反復解法と実用的なプログラムが必要となる。

反復法の基本は、連立一次方程式$Ax=b$において、

$$A=A_0-R \tag{3.1}$$

と分離し、左辺の一部を右辺に移して

$$A_0x=Rx+b \tag{3.2}$$

とする。左辺のxを、すでに求められたxから計算するよう

$$A_0x_{k+1}=Rx_k+b \tag{3.3}$$

とする。これが反復法の原理である。(3.2)、(3.3)より

$$A_0(x_{k+1}-x)=R(x_k-x) \tag{3.4}$$

となるが、これは

$$x_{k+1}-x=A_0^{-1}R(x_k-x) \tag{3.5}$$

$$=(A_0^{-1}R)^k(x_0-x) \tag{3.6}$$

なるので、任意の初期ベクトルx_0に対してこの反復法が収束するための条件は反復行列$(A_0^{-1}R)$のスペクトル半径が1をこえない、すなわち

$$\rho(A_0^{-1}R)<1 \tag{3.7}$$

である。拡散方程式を差分法で離散化して得られた行列AはM-行列なので、Aの非対角要素の一部を省いたものをA_0とすれば、$\rho(A_0^{-1}R)<1$となって反復法の収束が保証される[69],[54]。

Vargaの本[69]には、基本的な反復法として点ヤコビ法、Gauss-Seidel法、SOR法、SLOR法などがあげられている。点ヤコビ法の反復式は

$$x_{k+1}=D^{-1}(L+U)x_k+D^{-1}b$$

と表される。ここで、Lは狭義下三角行列、Uは狭義上三角行列、Dは対角行列である。点ヤコビ法ではx_{k+1}の成分を計算する間、x_kの全成分を記憶しておく。

点ヤコビ法におけるx_{k+1}の更新過程で、積極的に新しいx_{k+1}の成分を使用するのがGauss-Seidel法である。Gauss-Seidel法の反復式は

$$x_{k+1}=(D-L)^{-1}Ux_k+(D-L)^{-1}b$$

と表される。$(D-L)$、Uとも三角行列になるため、x_{k+1}とx_kの両方を記憶する必要はない。

SOR法では、Gauss-Seidel法の反復式によって計算されるベクトルを補助的なベクトル$\tilde{x}_{k+1}$として、x_kと$\tilde{x}_{k+1}$の重み付き平均

$$x_{k+1}=(1-\omega)x_k+\omega\tilde{x}_{k+1}$$

を作る。$\omega=1$の場合はGauss-Seidel法である。$\tilde{L}=D^{-1}L$、$\tilde{U}=D^{-1}U$とおいて行列で書けば

$$x_{k+1}=(I-\omega\tilde{L})^{-1}\{(1-\omega)I+\omega\tilde{U}\}x_k+\omega(I-\omega\tilde{L})^{-1}D^{-1}b$$

となり、ωを緩和因子(relaxation parameter)という。$\omega>1$の場合を過大緩和(overrelaxation)といい、全体の方法をSOR法(successive overrlaxation method)とよぶ。SOR法の場合もx_{k+1}とx_kの両方を記憶する必要はない。ある仮定のもとで最適なωの決定方法も知られているが[69]、それには最大固有値λ_{max}の推定が必要になるので、実用上はωを変えてみて、適当なωを探すことが普通に行われている。

3重対角行列T、下三角行列E、上三角行列Fを導入して、$A=T+E+F$と分解して、

$$x_{k+1}=(T+E)^{-1}Fx_k+(T+E)^{-1}b$$

としたものを線反復法という。この行列に対して加速係数ωを導入し、

$$x_{k+1}=-(T+\omega E)^{-1}\{(\omega-1)T+\omega F\}x_k+\omega(T+\omega E)^{-1}b$$

としたのがSLOR法である。SLOR法の主要部分は方程式を解くことになるため、陰解法とよばれる[45]。

SLOR法を除いたこれらの古典的反復法は汎用性にかけ、現在では使用しないほうがよいとされている[45]。実際は、プログラムが簡単なことからかなり広く使われている。拡散方程式を離散化して作ったml=16、n=560の問題に対して、右辺として$\lambda_i v_i$を与えてSOR法とSLOR法でωを変えながら560回の反復を行ってみたが、所定の判定値(0.22×10^{-10})では収束しないことが多かった。そのため、拡散方程式を離散化して作った問題に対してはこれらの反復法の使用を考慮しない。

これらの方法は、いずれもある条件のもとで収束が保証されているが、実際に何回の反復で、あるいは最大何回の反復で収束するかは明らかでない。ところがCG法は、最悪でも有限回(n元ならばn回)の反復で厳密解に到達することが理論的に保証されている。そのため、CG法は発表当時から非常に魅力のある解法とされていた。

3.2 CG法(Conjugate Gradient Method)のアルゴリズム

CG法は共役勾配法(Conjugate Gradient Method)の略称で、1952年にHestenesとStiefelによって発表された対称正定値行列Aに関する連立一次方程式 $Ax=b$ の反復解法である[32]。

反復解法は、第1近似解から始めてより良い近似解を作ることを繰り返して精度を高める解法で、直接解法よりも速く実用上十分な解が得られることもあるが、最悪の場合には収束しない可能性がある。ところがCG法は、最悪でも有限回(n元ならばn回)の反復で厳密解に到達することが理論的に保証される。このことからCG法は発表当時から非常に魅力のある解法とされた[51],[67],[14]。

ところが、CG法は計算誤差に敏感で問題によっては収束しないこともあり、約20年後のPCG法までは実用化されることが少なかった。PCG法は、Aに近い対称正定値行列$\widetilde{U}^{T}\widetilde{U}$を用いて前処理を施した方程式

$$\widetilde{U}^{-T}A\widetilde{U}^{-1}y=\widetilde{U}^{-T}b,\quad x=\widetilde{U}^{-1}y$$

にCG法を適用するものである[51],[42],[41],[18]。

CG法の原理は「対称正定値行列Aを係数とする連立一次方程式 $Ax=b$ があるとき、解xは汎関数

$$f(x) = (x, Ax)-2(b, x) \tag{3.8}$$

を最小にするxである。」ことから導かれる[51],[25],[54]。

いま、反復k回目における解の近似をx_kとしてk+1回目の近似解x_{k+1}を

$$x_{k+1} = x_k + \alpha_k p_k \tag{3.9}$$

とおく。p_kはx_kの修正ベクトルで、α_kはp_kの大きさを決める。(3.9)を(3.8)に代入して

$$\begin{aligned} f(x_{k+1}) &= f(x_k + \alpha_k p_k) \\ &= f(x_k)-2\alpha_k(p_k, r_k)+\alpha_k^{2}(p_k, Ap_k) \end{aligned} \tag{3.10}$$

を得る。(3.10)をα_kの関数として最小にするためには

$$\alpha_k =(p_k, r_k) / (p_k, Ap_k) \tag{3.11}$$

とすればよい。r_kは残差ベクトル$b-Ax_k$である。修正ベクトルp_{k+1}の方向は

$A^{-1}b-x_{k+1}$とAp_kの直交性:

$$\begin{aligned} (Ap_k, A^{-1}b-x_{k+1}) &= (p_k, b-Ax_{k+1}) \\ &= (p_k, r_{k+1}) \\ &= (p_k, r_k - \alpha_k Ap_k) \\ &= (p_k, r_k) -\alpha_k (p_k, Ap_k) \\ &= 0 \end{aligned} \tag{3.12}$$

に着目して、解を望む方向$A^{-1}b-x_{k+1}$に選ぶ。すなわち

$$p_{k+1} = r_{k+1}+\beta_k p_k \tag{3.13}$$

がAp_kと直交するようにβ_kを決める。

$$(Ap_k, r_{k+1}+\beta_k p_k) = (Ap_k, r_{k+1}) + \beta_k (Ap_k, p_k) = 0$$

から

$$\beta_k = -(Ap_k, r_{k+1})/(Ap_k, p_k) \tag{3.14}$$

が得られる。$r_0, r_1, \ldots, r_n$ は互いに直交、$p_0, p_1, \ldots, p_n$ は互いにA-直交である[51],[67],[54]。

これから得られるCG法のアルゴリズムは次のようになる。

(I)

x_0:初期解を用意; $r_0 = b - Ax_0$:初期残差ベクトル
$p_0 = r_0$:初期修正ベクトル; k=0
while $\|r_k\| > \varepsilon$ do
  $\alpha_k = (p_k, r_k)/(p_k, Ap_k)$
  $x_{k+1} = x_k + \alpha_k p_k$; $r_{k+1} = r_k - \alpha_k Ap_k$
  $\beta_k = -(Ap_k, r_{k+1})/(Ap_k, p_k)$
  $p_{k+1} = r_{k+1} + \beta_k p_k$; k=k+1

この算法から、次の関係

$$(p_{k-1}, Ap_k) = 0 \tag{3.15}$$

$$(r_k, Ap_k) = (p_k, Ap_k) \tag{3.16}$$

$$(r_{k-1}, r_k) = 0 \tag{3.17}$$

は直ちに導かれる。

一方、よく使われているCG法のアルゴリズムは次のようなものである。

(II)

x_0:初期解を用意; $r_0 = b - Ax_0$:初期残差ベクトル
$p_0 = r_0$:初期修正ベクトル; k=0
while $\|r_k\| > \varepsilon$ do
  $\alpha_k = (r_k, r_k)/(p_k, Ap_k)$
  $x_{k+1} = x_k + \alpha_k p_k$; $r_{k+1} = r_k - \alpha_k Ap_k$
  $\beta_k = (r_{k+1}, r_{k+1})/(r_k, r_k)$
  $p_{k+1} = r_{k+1} + \beta_k p_k$; k=k+1

(II)はα_k, β_k の作り方が異なるだけで数学的には(I)と同じであることが、(3.15)、(3.16)、(3.17)から証明できる。

前にも述べたように、2次元の場を5点差分で離散化したとき、非対角帯半幅をm1、元数をnとすると、連立一次方程式$Au=f$の第i番方程式は

$$a_{i,i-m1}u_{i-m1} + a_{i,i-1}u_{i-1} + a_{i,i}u_i + a_{i,i+1}u_{i+1} + a_{i,i+m1}u_{i+m1} = f_i$$

と表される。Aの非ゼロ要素は5本の対角部分で構成され、Aが対称であることから上三角部分または下三角部分だけを考えればよい。以下では上三角部分を考え、方程式番号i を固定したとき、対角要素をa_i、対角の右どなりの要素をb_i、対

角からm1離れた要素をc_iで表し、実際のプログラムにおいても3本の一次元配列を用いる。

CG法に関する詳細な収束特性の解析は4章で述べる。

3.3 CG法の特徴

残差ベクトル$r_0,r_1,...,r_n$は互いに直交、$p_0,p_1,...,p_n$は互いにA-直交である[51],[67],[54]。これから、理論上は最悪でもn回(nは係数行列の次元)の反復で厳密解に収束する。そのためCG法は発表当時から非常に魅力的な算法とされた[51],[14]。

数式処理のような誤差の入らない計算が可能な環境では、CG法は有限回の反復で収束するため、直接解法のような取扱いが可能である。また、倍精度計算(64ビット)では誤差のため収束しなくても、4倍精度(128ビット)にすれば収束するかもしれない。そのため、精度と収束の関係が明らかになれば、より使いやすくなる。しかも、CG法では、内積、行列をベクトルにかける、ベクトルのスカラー倍を他のベクトルに加えるといった基本的な行列演算だけが必要であり、分散メモリ環境のような新しい計算環境で、これらの行列演算が高速･高精度に実行できればCG法の適用範囲ははるかに広がるはずである。

一般的な反復の停止条件として、$\|r_k\| \leqq \varepsilon$または$\|r_k\| / \|f\| \leqq \varepsilon$などが使われているが、誤差によって収束しない場合をモニターするためには、修正量α_kが小さくなったとか、$\|p_k\| \leqq \varepsilon$なども考えられる。

4. CG法の収束特性

対称正定値行列Aを係数とする連立一次方程式$Ax=b$を共役勾配法(CG法)で解くことを考える。PCG法では、前処理を施した行列$\widetilde{U}^{-T}A\widetilde{U}^{-1}$が単位行列に近づいたり、密集固有値が多く存在するときに収束が速いとされているが[51]、CG法自身の問題依存性などは調べられていない。ここでは、拡散方程式を差分法で離散化して得られた行列を対象にして、固有値·固有ベクトルなどの数学的性質に注目して検討を加える。また、右辺bや解xによっても収束の速さや精度は大きく左右されるが[25]、そのような右辺や解についての依存性はなかなか比較がしにくいため、このような解析も行われていない。そこで、主として解を固有ベクトルを用いて構成することによって、係数行列と解や右辺に依存してCG法の収束がどのように変化するかを明らかにする。

それには、あらかじめ係数行列の固有値·固有ベクトルを高精度で計算しておき、その固有ベクトルの組合せに基づいて解xを決める。右辺bは解xからAxによって作り、反復の初期ベクトルを$x=o$としてCG法を適用する。このような実験によって、個々の固有ベクトル成分に対する収束がどうなるかは完全に明らかになる。また、簡単な固有ベクトル成分の組み合わせたときに収束がどのようになるかも明らかになる。

4.1 収束特性解析の方法

CG法の原理は

$$f(x)=(x, Ax)-2(b, x)$$

の最小化だった[51],[25]。この式を残差$r=b-Ax$を用いて書き直すと

$$\begin{aligned} f(x) &= (A^{-1}(b-r), b-r)-2(b, A^{-1}b) \\ &= (A^{-1}r, r)-(A^{-1}b, b) \end{aligned}$$

となる。これから$f(x)$の最小化は$(A^{-1}r, r)$の最小化と同値であることがわかる。CG法のアルゴリズムで$p_0=r_0$として反復を始めると

$$\begin{aligned} r_1 &= r_0-\alpha_0 A p_0 \\ &= (I-\alpha_0 A) r_0 \\ p_1 &= r_1+\beta_0 p_0 \\ &= (I-\alpha_0 A) r_0+\beta_0 r_0 \\ &= ((1+\beta_0) I+\alpha_0 A) r_0 \end{aligned}$$

$R_0(A)=I$、$R_1(A)=I-\alpha_0 A$、$P_1(A)=(1+\beta_0)I+\alpha_0$とすると、

$$\begin{aligned} r_{k+1} &= r_k-\alpha_k A p_k \\ &= (R_k(A)-\alpha_k A P_k(A)) r_0 \\ &= R_{k+1}(A) r_0 \end{aligned}$$

同様に

$$p_{k+1}=r_k+\beta_k p_k$$
$$=(R_{k+1}(A)+\beta_k P_k(A))r_0$$
$$=P_{k+1}(A)r_0$$

と書ける。したがって、第k回の反復では

$$(A^{-1}r_k, r_k)=(A^{-1}R_k(A)r_0, R_k(A)r_0)$$

を最小化することになる。残差ベクトルr_0をAの固有ベクトルv_iを使って展開すると

$$\Sigma c_i^2 R_k^2(\lambda_i)/\lambda_i$$

の最小化となる[51],[58]。

4.2 固有値·固有ベクトルを用いた収束特性解析の特徴

固有ベクトルの組$\{v_1,v_2,...,v_n\}$はR^nの正規直交基底を構成し、対応する固有値は小さい順に$\lambda_1 \leqq \lambda_2 \leqq ... \leqq \lambda_n$のように番号付けられているとする。

初めに、$Av_i=\lambda_i v_i$となるような1本の固有ベクトルを解とするような方程式を考え、これに初期解$x_0=o$として共役勾配法(I)のアルゴリズムを適用する。CG法のアルゴリズムでは、$Av_i=\lambda_i v_i$となる1本のv_iを解とするような問題は理論上1回の反復で収束する[31]。

実際、$b=Av_i=\lambda_i v_i$なので、$r_0=b-Ax_0=\lambda_i v_i$、$p_0=r_0=\lambda_i v_i$である。1回目(k=0)の反復過程は

$$\alpha_0=(p_0, r_0)/(p_0,Ap_0)$$
$$=(\lambda_i v_i,\lambda_i v_i)/(\lambda_i v_i,\lambda_i^2 v_i)$$
$$=1/\lambda_i$$
$$x_1=x_0+\alpha_0 p_0$$
$$=o+\lambda_i v_i/\lambda_i$$
$$=v_i$$
$$r_1=r_0-\alpha_0 A p_0$$
$$=\lambda_i v_i-\lambda_i^2 v_i/\lambda_i$$
$$=o$$
$$\beta_0=-(Ap_0,r_1)/(Ap_0,p_0)$$
$$=-(\lambda_i^2 v_i,o)/(\lambda_i^2 v_i,\lambda_i v_i)$$
$$=0$$
$$p_1=r_1+\beta_0 p_0$$
$$=o+0(\lambda_i v_i)$$
$$=o$$

となって、1回の反復で収束する。したがって、理論上、1本の固有ベクトル成分

だけを解にもつような連立一次方程式はCG法で解くと必ず1回の反復で収束する。

つぎに、隣接した2本の固有ベクトル$v_{i-1}+v_i$を解にもつような問題を考える。$x_0=o$、$b=A(v_{i-1}+v_i)=\lambda_{i-1}v_{i-1}+\lambda_i v_i$なので、$r_0=b-Ax_0=\lambda_{i-1}v_{i-1}+\lambda_i v_i$、$p_0=r_0=\lambda_{i-1}v_{i-1}+\lambda_i v_i$である。k=0のときのCG法の反復過程は

$$\begin{aligned}\alpha_0&=(p_0,r_0)/(p_0,Ap_0)\\&=(\lambda_{i-1}v_{i-1}+\lambda_i v_i,\lambda_{i-1}v_{i-1}+\lambda_i v_i)/(\lambda_{i-1}v_{i-1}+\lambda_i v_i,\lambda_{i-1}^2v_{i-1}+\lambda_i^2v_i)\\&=(\lambda_{i-1}^2+\lambda_i^2)/(\lambda_{i-1}^3+\lambda_i^3)\end{aligned}$$

となる。

$$\begin{aligned}x_1&=x_0+\alpha_0p_0\\&=o+\alpha_0(\lambda_{i-1}v_{i-1}+\lambda_i v_i)\\&=\lambda_{i-1}\cdot(\lambda_{i-1}^2+\lambda_i^2)/(\lambda_{i-1}^3+\lambda_i^3)v_{i-1}+\lambda_i\cdot(\lambda_{i-1}^2+\lambda_i^2)/(\lambda_{i-1}^3+\lambda_i^3)v_i\\r_1&=r_0-\alpha_0Ap_0\\&=\lambda_{i-1}v_{i-1}+\lambda_i v_i-\alpha_0(\lambda_{i-1}^2v_{i-1}+\lambda_i^2v_i)\\&=\lambda_{i-1}\lambda_i(\lambda_i^2-\lambda_{i-1}\lambda_i)/(\lambda_{i-1}^3+\lambda_i^3)v_{i-1}+\lambda_{i-1}\lambda_i(\lambda_{i-1}^2-\lambda_{i-1}\lambda_i)/(\lambda_{i-1}^3+\lambda_i^3)v_i.\end{aligned}$$

CG法の性質から、残差ベクトル$r_0,r_1,\ldots,r_n$は互いに直交するので、この方程式は2回で収束する。$v_{i-1}+v_i$の場合を含めて、隣接する2本の固有ベクトルの線形結合を解にもつような方程式は2回の反復で収束する。このとき、固有値$\lambda_{i-1}=\lambda_i$ならば、残差ベクトルr_1の式からも明らかなように、1回の反復で収束する。

同様な方法で、Σv_iを考える。固有値λ_kと$\lambda_{k'}$が等しいなら、$v=v_k+v_{k'}$とおいた新しいベクトルvをつくる。vはv_k、$v_{k'}$以外の固有ベクトルとは直交する。したがって、線形結合Σv_iからv_kと$v_{k'}$を取り去って、vを使うことにすれば1本少ない固有ベクトルによって線形結合Σv_iが形成されるとみなせる。これから、重複固有値に対応した固有ベクトル成分を含む場合は、重複の分だけ反復回数が少なくて済むはずである。

実験のために次のような5通りの解を用意する。ここでも、固有ベクトルの組は正規直交基底を形成し、固有値は小さい順に並べられているものとする。

(a) v_k
(b) Σv_i (i=k-1,k)
(c) Σv_i (i=k-2,k)
(d) Σv_i (i=1,k)
(e) Σv_i (i=k,n)

(a)は固有ベクトル1本だけの場合で、理論との比較ならびに固有値の違いによる変化をみることを目的にしている。(b)と(c)はそれぞれ、固有ベクトル2本の場合、固有ベクトル3本の場合であり、重複ならび密集の効果をみるための組合せである。(d)は低次モード(小さな固有値に対応)の影響をみるための組合せで、(e)は高次モード(大きな固有値に対応)の影響をみるための組合せである。(d)、(e)に

対しては

$\Sigma\lambda_i v_i$

$\Sigma v_i/\lambda_i$

という一次結合の係数を大きく変化させた2種類の線形結合を加えた。これらは、一次結合の係数の大きさのオーダーが大きく変化するような問題と、各係数が1の場合とで収束がどのように変化するかをみるためのものである。

4.3 固有値解析の方法

すべての固有値·固有ベクトルを求めるためにはn^2ワード以上の領域が必要であること、高精度な固有値解析に必要となるCPU時間の制約から今回の固有値解析はm1=16、n=560に限定した。

行列Aは対称正定値で、上(下)三角行列の非零要素を3本の対角線上にもつ規則的疎行列である。解析にあたっては、このような行列の固有値·固有ベクトルを極めて精度良く求める必要がある。このような標準固有値問題に使えるプログラムとして、RutishauserのJACOBI[52],[75]、LINPACK中のQL法(TRED2+TQL2)[8]、行列計算ソフトウエアの村田法MURA2[52]、2分法·同時逆反復法EIGV3[52]などをとりあげた。このうち、MURA2、EIGV3は帯行列を対象にしているため、演算時間とメモリ容量の節約が可能である。これらはすべて倍精度用のプログラムなので、4倍精度用にはパラメータの変更が容易にできたJACOBIとQL法のみを使った。

実際は、これらのプログラムを使って対象とする問題すべてを汎用コンピュータ上で解き、得られた固有値·固有ベクトルの精度を検討していちばん精度のよいものを用いた。精度の比較には、(1)残差ノルム$\|A v_i-\lambda_i v_i\|$、(2)直交性$v_i^T v_j$、(3)混じり分$|x_i^T r_j|/(\lambda_i-\lambda j)$を用いた。ここで、$x_i$はi番目の近似固有ベクトル、$r_j$はj番目の残差ベクトルである。

後述の(646)のケースで倍精度計算のとき、一番精度がよかったのはJACOBIで、残差ノルムの最大値は0.291×10^{-12}、直交性の最大値は10^{-12}、混じり分の最大値は10^{-9}だった。MURA2、QL法、EIGV3は残差ノルムで1桁、直交性で2～7桁劣り、混じり分は同程度だった。汎用コンピュータM-660KでのCPU時間はJACOBIが44分、MURA2が84秒、EIGV3が133秒、QL法が255秒だった。

同じケースで4倍精度計算のとき、一番精度がよかったのはやはりJACOBIで、残差ノルムの最大値は0.586×10^{-29}、直交性の最大値は10^{-29}、混じり分の最大値は10^{-15}だった。JACOBIに比べると、QL法は残差ノルムと直交性で1桁、混じり分で5桁劣った。CPU時間はJACOBIが237分、QL法が28分だった。

4.4 収束特性の解析(数値実験)

1章で詳しく述べたように、2次元矩形領域における拡散方程式を差分法で離散化して得られた行列をテスト問題として用いる。y軸方向の分割をm1としたとき、元数n＝m1(2m1＋3)になる。図4.1で白い部分の拡散係数k(x,y)を1,その他の部分の拡散係数 k(x,y) をそれぞれ、DF0、DF1、DF2、DF3とし、DF0を10^{-12} に固定する。拡散係数DF1、DF2、DF3に小さな値を与えると、Aには与えた値程度の小さな固有値が3m1(48)個以上密集する。DF1、DF2、DF3を変化させ、拡散係数に段差をもたせると、固有値の密集の程度、条件数などが大きく変わる。

CG法の収束判定は、反復の間に更新されるr_kを用いた相対残差ノルムが$\|r_k\|/\|b\| < 0.22\times10^{-10}$となるまで行なった。計算は原則として倍精度とし、特別に精度を要求するところでは4倍精度とした。

実験に用いたDF1、DF2、DF3の組合せと、最大固有値λ_{min}、最小固有値λ_{max}、条件数Cond を表4.1に示す。以下、DF1＝10^{-6}、DF2＝10^{-4}、DF3＝10^{-6}の場合を(646)のような略称でよぶ。

表4.1　DF1、DF2、DF3とλ_{min}、λ_{max}、条件数Cond (m1＝16、n＝560)

略称	DF1	DF2	DF3	λ_{min}	λ_{max}	Cond
(000)	1	1	1	0.71309×10^{-2}	7.9514	0.11150×10^{4}
(323)	10^{-3}	10^{-2}	10^{-3}	0.27909×10^{-4}	7.9181	0.28370×10^{6}
(646)	10^{-6}	10^{-4}	10^{-6}	0.23978×10^{-7}	7.9181	0.28300×10^{9}
(969)	10^{-9}	10^{-6}	10^{-9}	0.27982×10^{-10}	7.9181	0.28296×10^{12}
(A8A)	10^{-10}	10^{-8}	10^{-10}	0.28015×10^{-11}	7.9181	0.28263×10^{13}

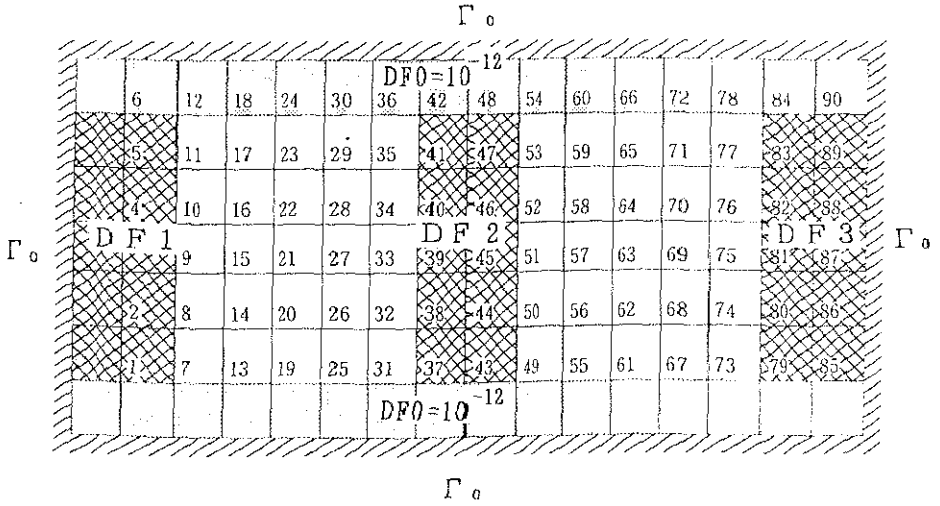


図４．１　テスト問題のための離散場　　n=m1(2m1+3)

(a) v_k

図4.2~4.5に(646)の結果を示す。横軸はlogスケールで固有値の存在するところを示す。縦軸はそのような固有値に対応した固有ベクトル成分を解とした問題に必要なCG法の反復回数を示す。反復は最大560回(=方程式の元数n)で打ち切った。(000)や(323)などの比較的条件数が小さい問題では、固有値解析とCG法を倍精度で計算すれば概ね理論どおり1回の反復で収束した。ところが(646)や(A8A)となると倍精度計算では理論どおりの結果が得られない。そこで、固有値解析とCG法に対して、倍精度計算と4倍精度計算を適用した結果が図4.3~4.5である。固有値解析とCG法が倍精度の場合は、10^{-3}以下の固有値に対応した解が1回では収束していない。図4.3のようにCG法だけを4倍精度にしてもほとんど効果はなく、固有値解析だけを4倍精度で行った図4.4は1回では収束しないものの反復回数はかなり減少している。図4.5のように、すべてを4倍精度で計算することによって始めて理論どおりの結果が得られる。

この原因はコンピュータの誤差にある。固有値解析を4倍精度で行うと反復回数が少なくなることから、CG法のような反復解法の収束特性を調べるとき、倍精度で計算して得られた残差ノルムで10^{-10}程度の精度をもつ固有値･固有ベクトルでは精度が不充分だということである。そして、条件数が10^{9}程度以上の問題を倍精度計算のCG法で解いた場合には、何らかの誤差によって収束しない可能性もありうるということである。

(A8A)のケースでは最小固有値$\lambda_{min}=0.28015\times10^{-11}$で、この固有値に対応した1本の固有ベクトルを解とする方程式は4倍精度でも37回の反復を必要とする。このとき、α_k、β_k、$\|r_k\|$、$\|p_k\|$がどのように変化するかを、図4.6、図4.7に示す。4倍精度の場合、α_0は理論通りおよそ$10^{12}(\approx\lambda_{min}^{-1})$なのに対し、倍精度の場合$10^{4}$程度にしかなっていない。そのため、2回目以降の反復で$\|r_k\|$、$\|p_k\|$が増大し、残差ノルムr_kは規則的な繰り返しで1から10^{5}の範囲を動き、560回の反復でも収束しない。

この例では、第1回目の反復過程で本来ならば0になるはずの残差ベクトルr_1、修正ベクトルp_1が0にならなかったためにこのような現象が生じている。固有ベクトルは正規直交なので、固有ベクトルには特別オーダーの隔たった値はないはずである。そうすると、原因はα_0を計算する部分で、分子(p_0,r_0)が約10^{-20}、分母(p_0,Ap_0)が約10^{-30}になって、倍精度計算の精度の限界をこえたことによると考えられる。最終的なα_0も決して妥当なオーダーの数ではない。固有値･固有ベクトルに含まれている誤差の関係もあるので一概にはいえないが、αの一般式は$\Sigma c_i^2\lambda_i^2/\Sigma c_i^2\lambda_i^3$なので、桁数の計算では$c_i$の大きさは1のオーダーとして無視できると仮定すれば、小さい固有値λ_iに対するλ_i^3がマシンイプシロンにふれないようにする必要がある。16進倍精度計算のマシンイプシロンは0.22×10^{-15}だから、有効桁数は最大でおよそ15桁で、少なくとも$\lambda_i/\lambda_{max}\geqq10^{-5}$の必要がある。マシンイプ

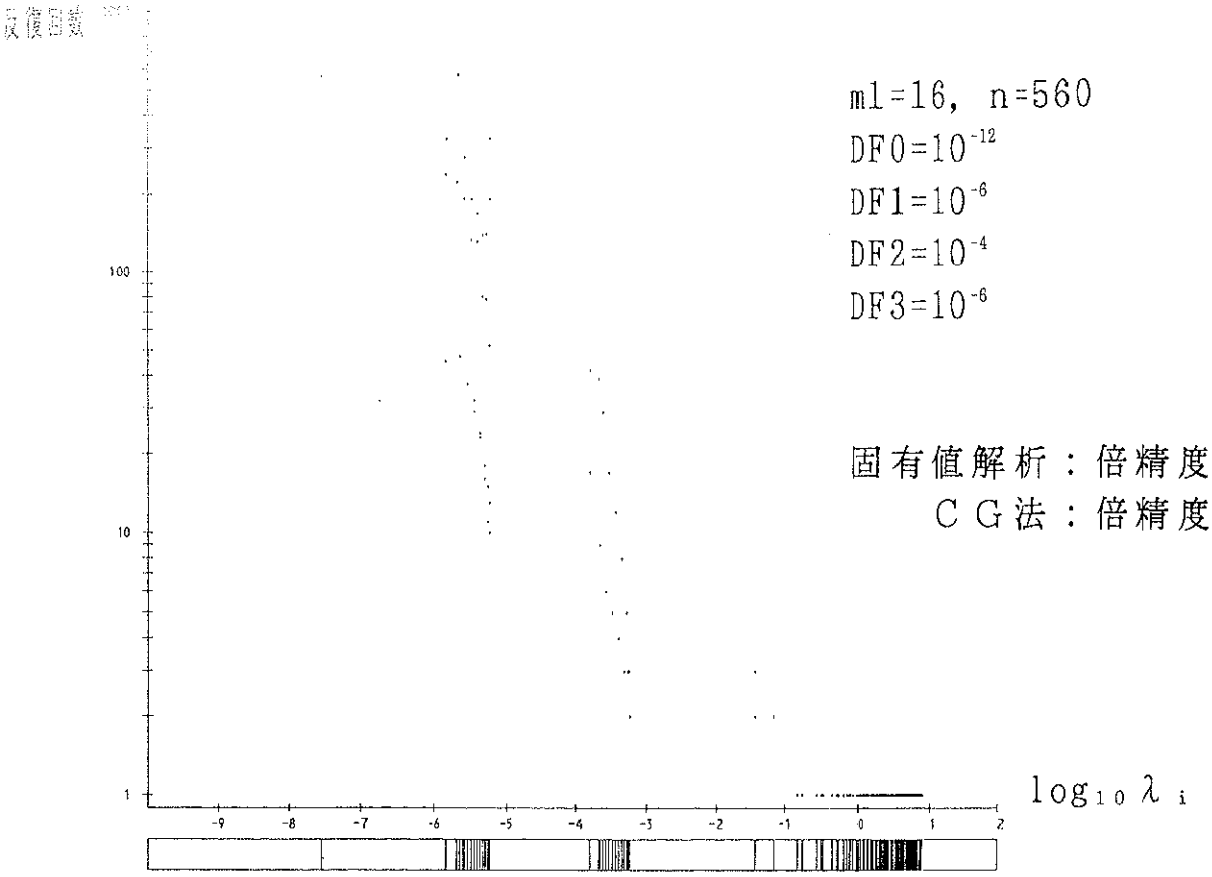


図４．２　固有値分布$\log_{10}\lambda_i$とv_iに対するＣＧ法の反復回数

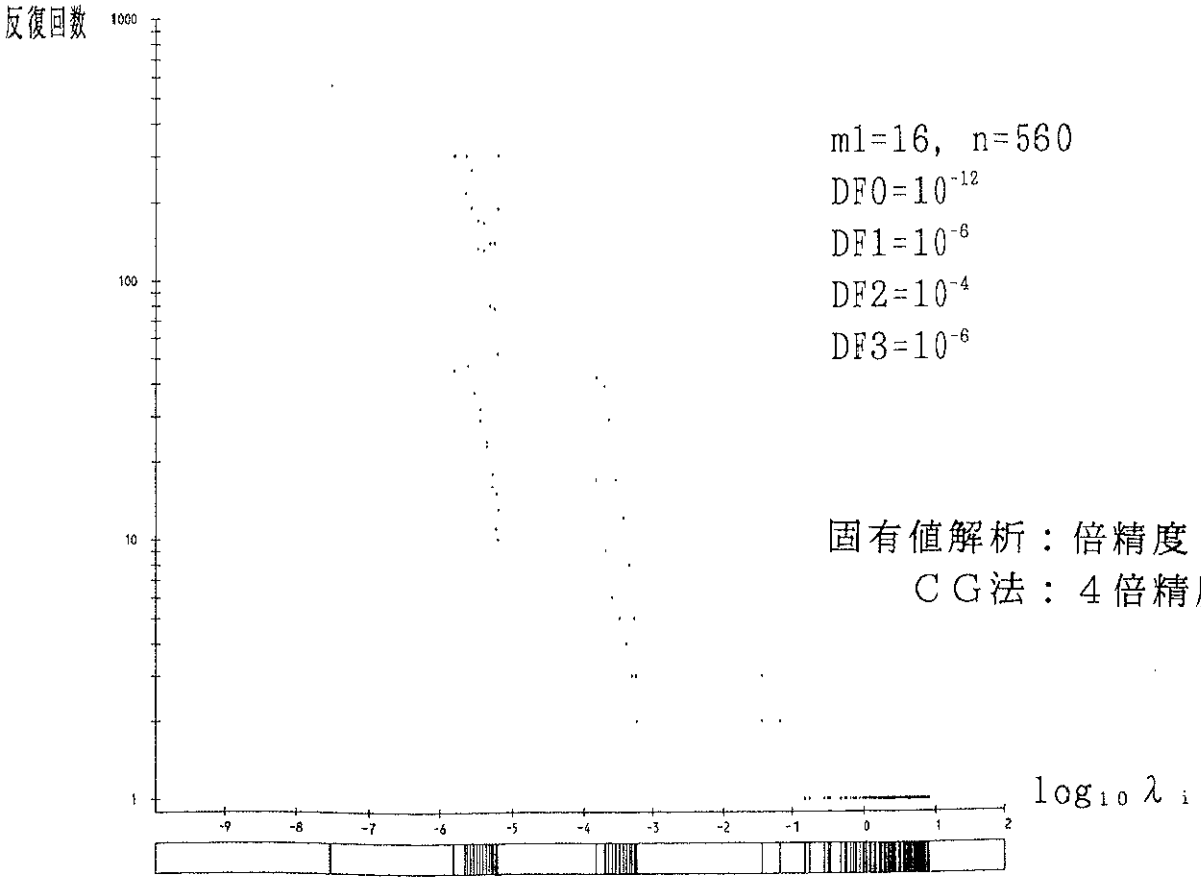


図４．３　固有値分布$\log_{10}\lambda_i$とv_iに対するＣＧ法の反復回数

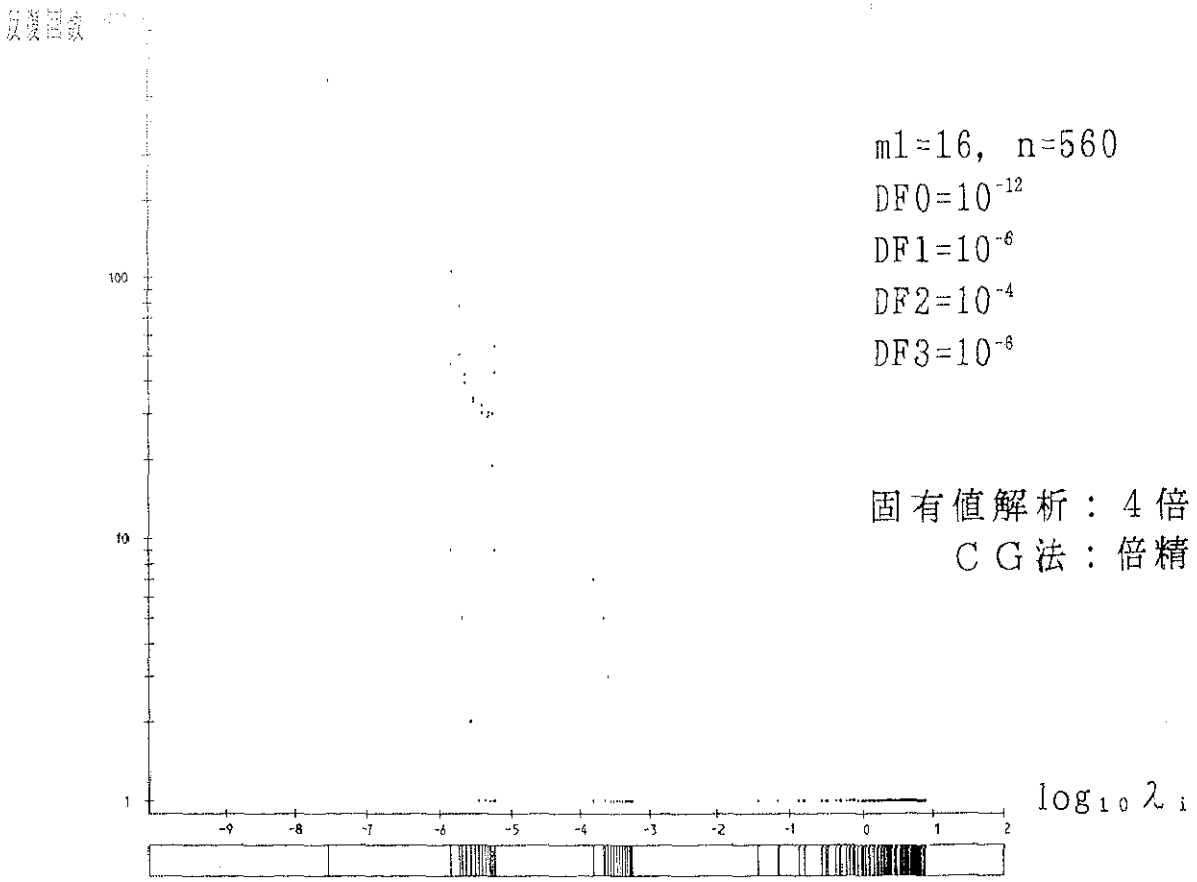


図４．４　固有値分布$\log_{10}\lambda_i$とv_iに対するＣＧ法の反復回数

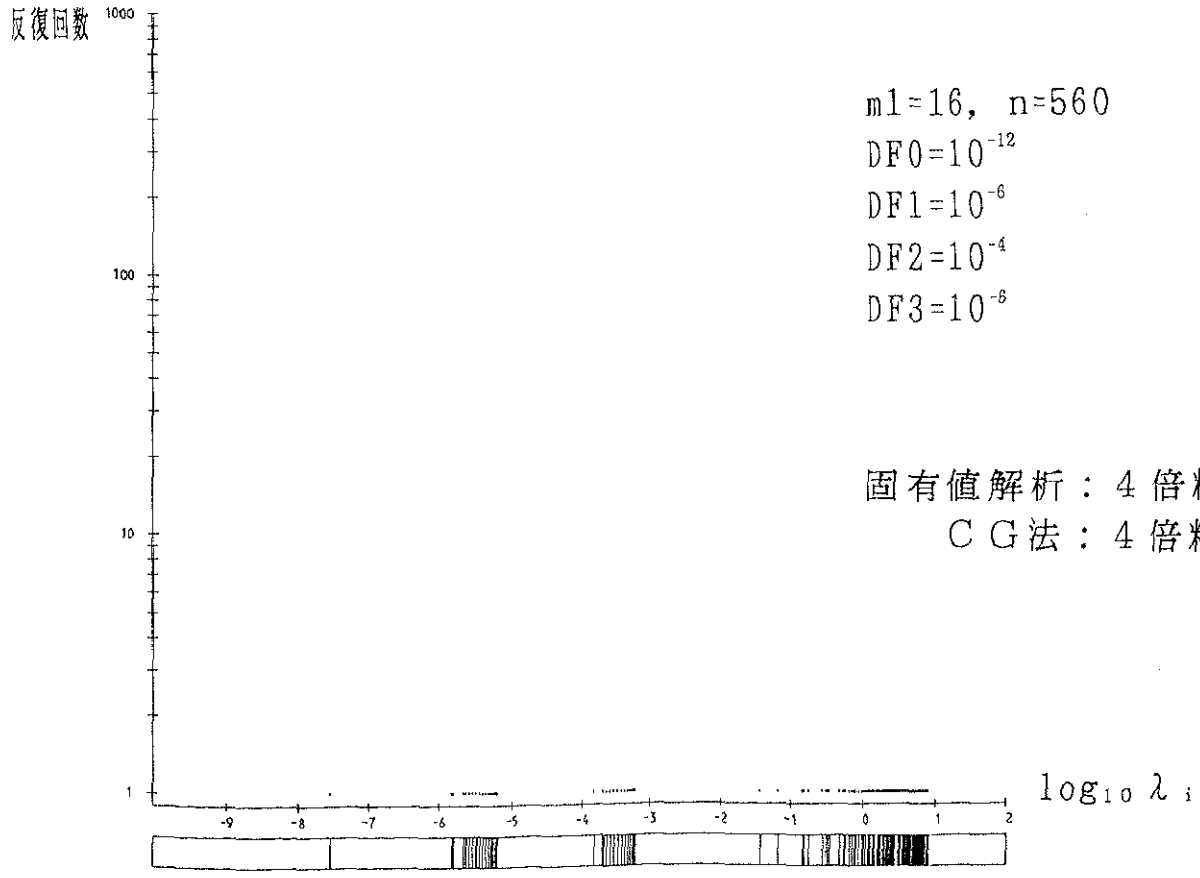


図４．５　固有値分布$\log_{10}\lambda_i$とv_iに対するＣＧ法の反復回数

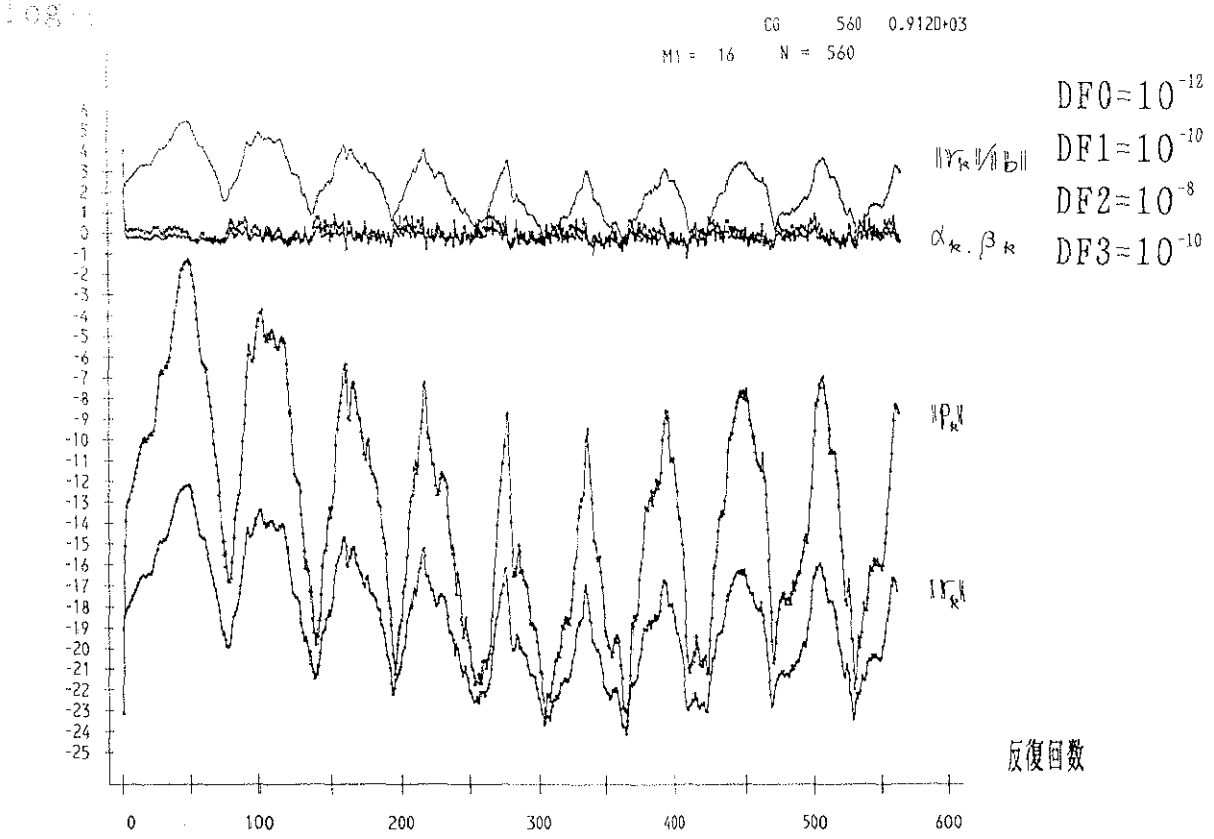


図４．６　倍精度計算によるＣＧ法の収束過程

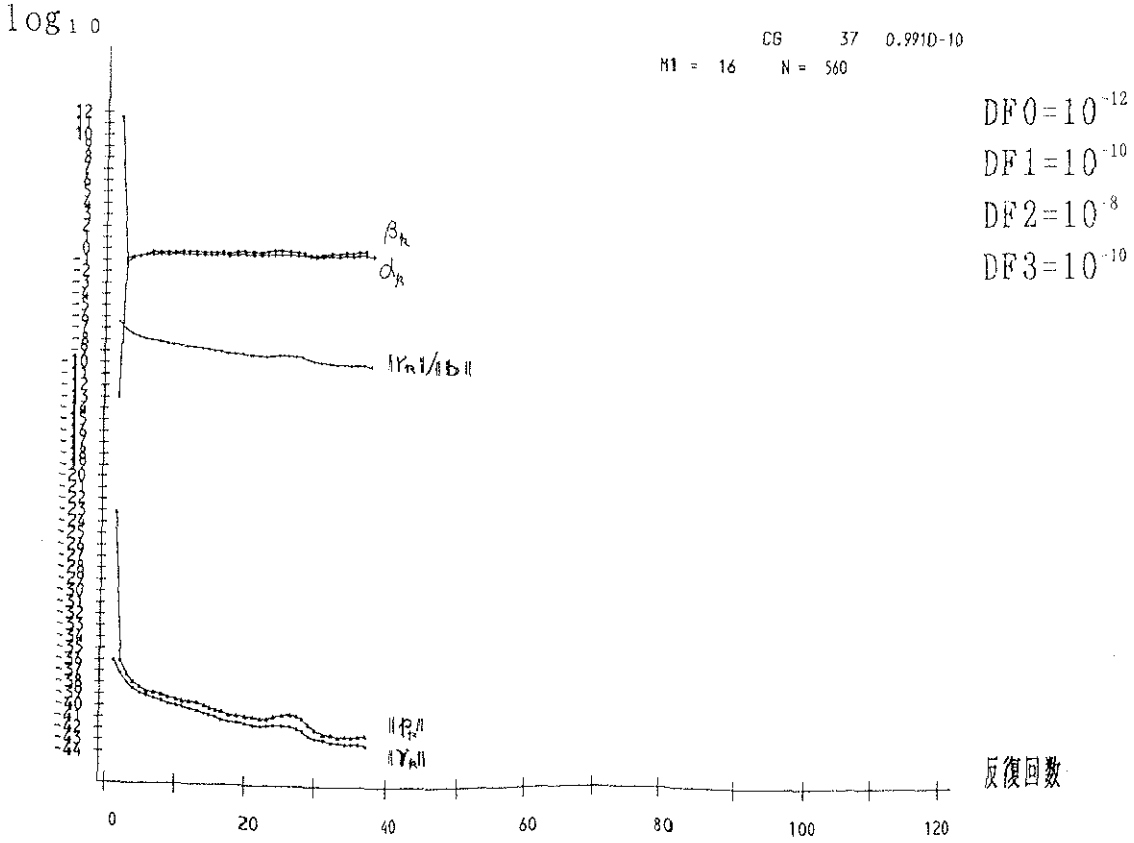


図４．７4倍精度計算によるＣＧ法の収束過程

シロンぎりぎりまでは使えないことと、倍精度計算で用いた固有値･固有ベクトルの精度は10^{-12}程度しかないこと、この場合の$\lambda_{max} \approx 10^{1}$であることより、実際は$\lambda_i \geqq 10^{-3}$程度の場合が保証されるにすぎない。これは図4.4からも裏付けられる。

もともとのCG法には誤差に対する修正が用意されていないので、いったん誤差が混入すると反復のふるまいはおかしくなる。このことは、小さい固有値に対応した固有ベクトルを与えると誤差のため収束しにくいともいいかえられる。

(b) Σv_i (i=k-1,k)、 (c) Σv_i (i=k-2,k)

図4.8、図4.9に(000)、図4.10、図4.11に(646)、図4.12、図4.13に(A8A)を示す。計算はすべて4倍精度で行った。

(000)のケースでは、隣接する固有ベクトルを2本与えれば2回，3本与えれば3回で理論どおり収束する。(646)のケースでは、隣接する2本の固有ベクトルを与えると2回または1回で収束している。3本にすると、3回または2回で収束しているが、10^{-5}以下の固有値に対応した固有ベクトルの部分では、10回以上の反復が必要なものもある。(A8A)ではその傾向がいちだんと顕著になり、理論どおり2回または3回以下で収束する部分と収束しない部分がはっきりと別れてしまう。

これらの結果で隣接する2本の固有ベクトルを与えたときに1回、3本与えたときに2回となるのは重複または密集のためである。実験的に重複または密集の効果を調べるため、隣接する2本の固有ベクトルを与えた場合、固有値の差と収束に要した反復回数をまとめたのが表4.2～表4.4である。縦軸にlogスケールで固有値の差、横軸に収束に要した反復回数が示してある。表4.2から明らかなように、(646)のケースでは固有値の差が10^{-10}以下になると反復回数が1回になる。表4.3の(A8A)ケースでも同様なことがわかるが、3回以上の反復を要するものが散らばって存在するためわかりにくい。そこで値が10^{-7}以下の固有値に対応した50組の固有ベクトルの組を除いてカウントしたものが表4.4である。特別に小さい10^{-7}以下の固有値を除いてカウントすることによって、ここでも差が10^{-10}以下になると反復回数が減少することがわかる。

3本の隣接する固有ベクトルの組について同様な方法を適用してみたが、このときも差が10^{-10}以下になると反復回数が1回または2回となる。このときは、10^{-5}以下の固有値に対応した固有ベクトルの組を除いてカウントした。10本の場合も同様な傾向が観察される。計算を4倍精度で行なっても、解を構成する固有ベクトルの本数が増えるにつれて、あるいは小さな値の固有値が現れるにつれて、解を構成する固有ベクトルの本数以上の反復が必要になる。

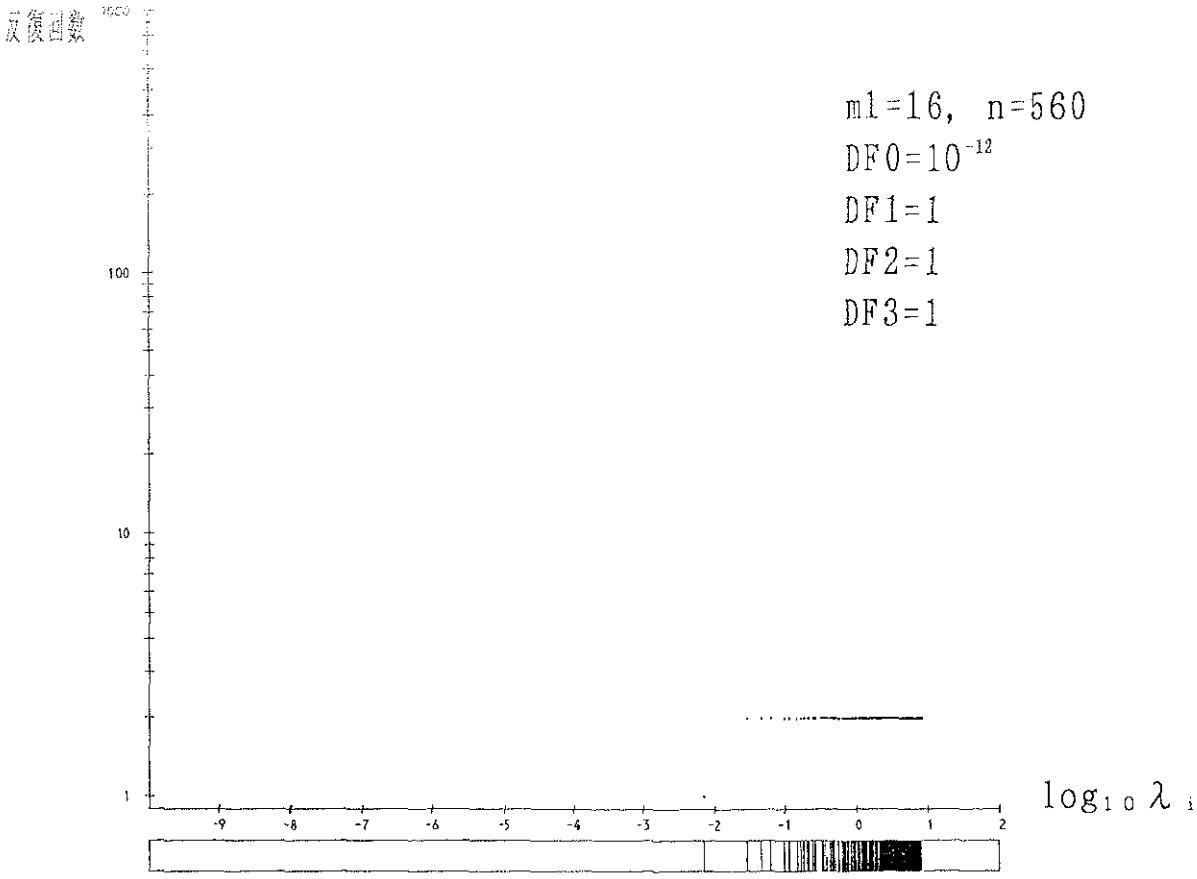


図４．８　固有値分布$\log_{10}\lambda_i$と$\sum_{i=k-1}^{k} v_i$に対するＣＧ法の反復回数

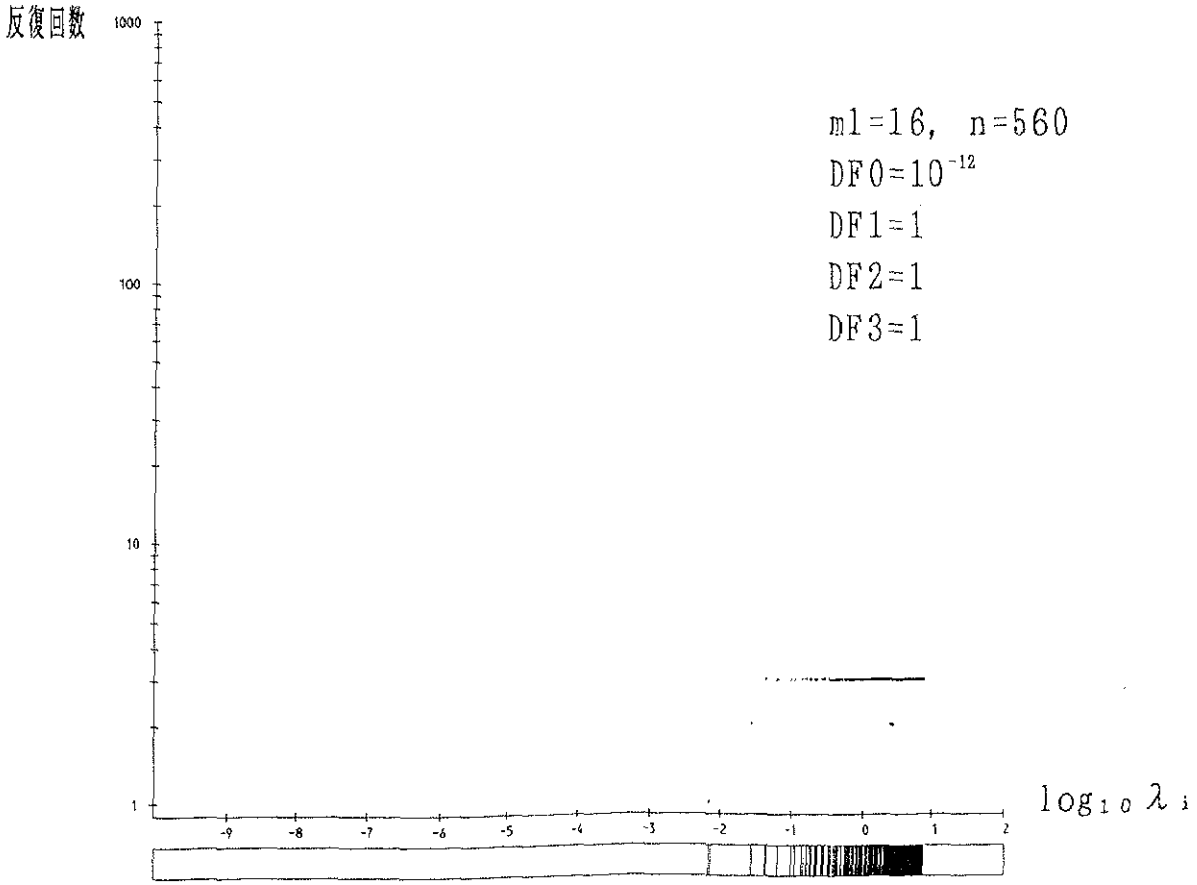


図４．９　固有値分布$\log_{10}\lambda_i$と$\sum_{i=k-2}^{k} v_i$に対するＣＧ法の反復回数

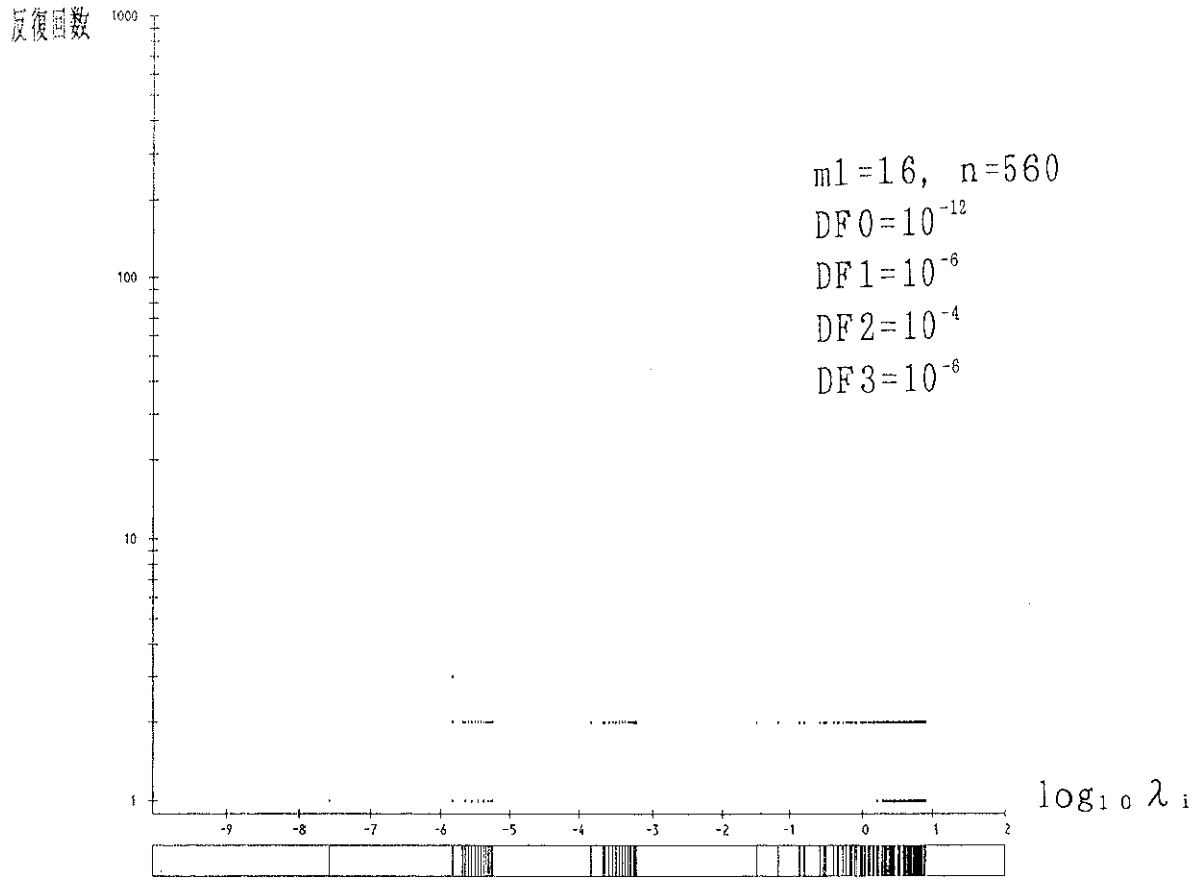


図４.10 固有値分布$\log_{10}\lambda_i$と$\sum_{i=k-1}^{k} v_i$に対するＣＧ法の反復回数

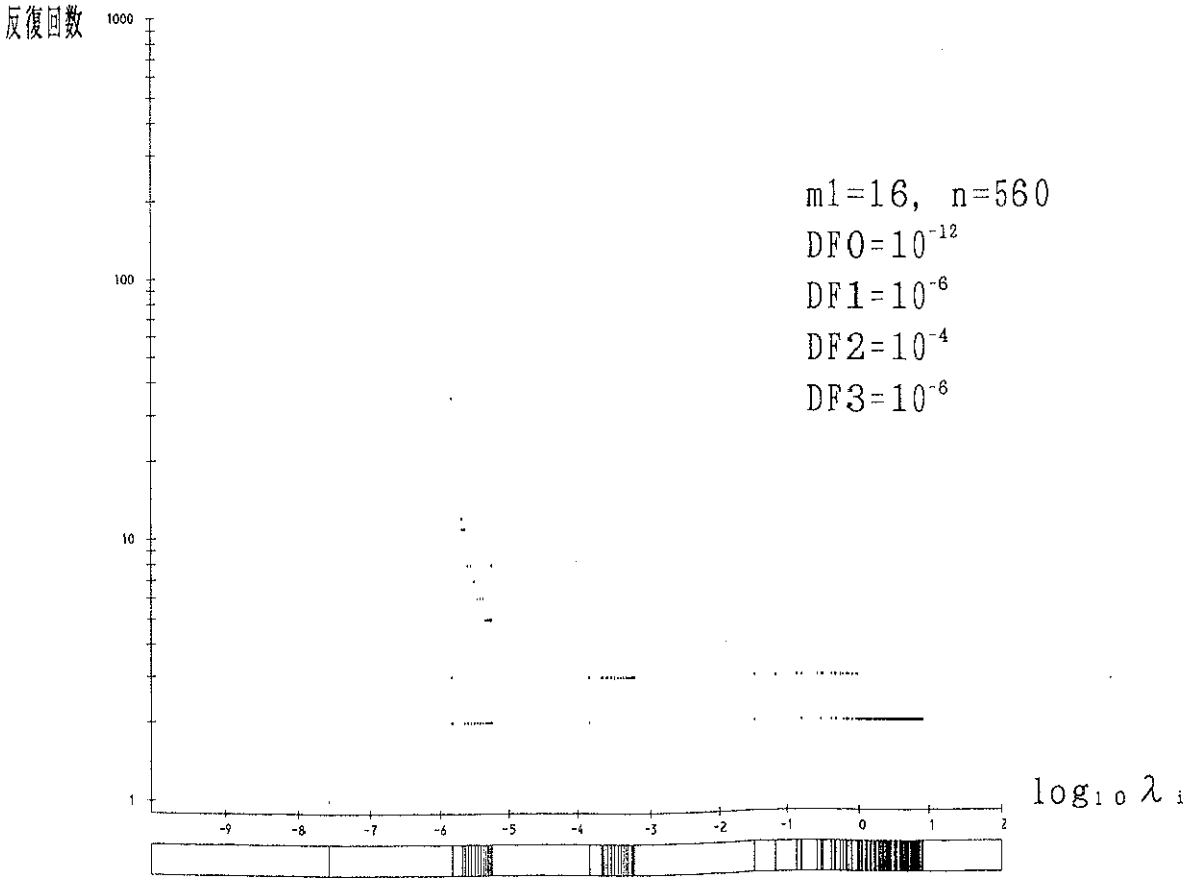


図４.11 固有値分布$\log_{10}\lambda_i$と$\sum_{i=k-2}^{k} v_i$に対するＣＧ法の反復回数

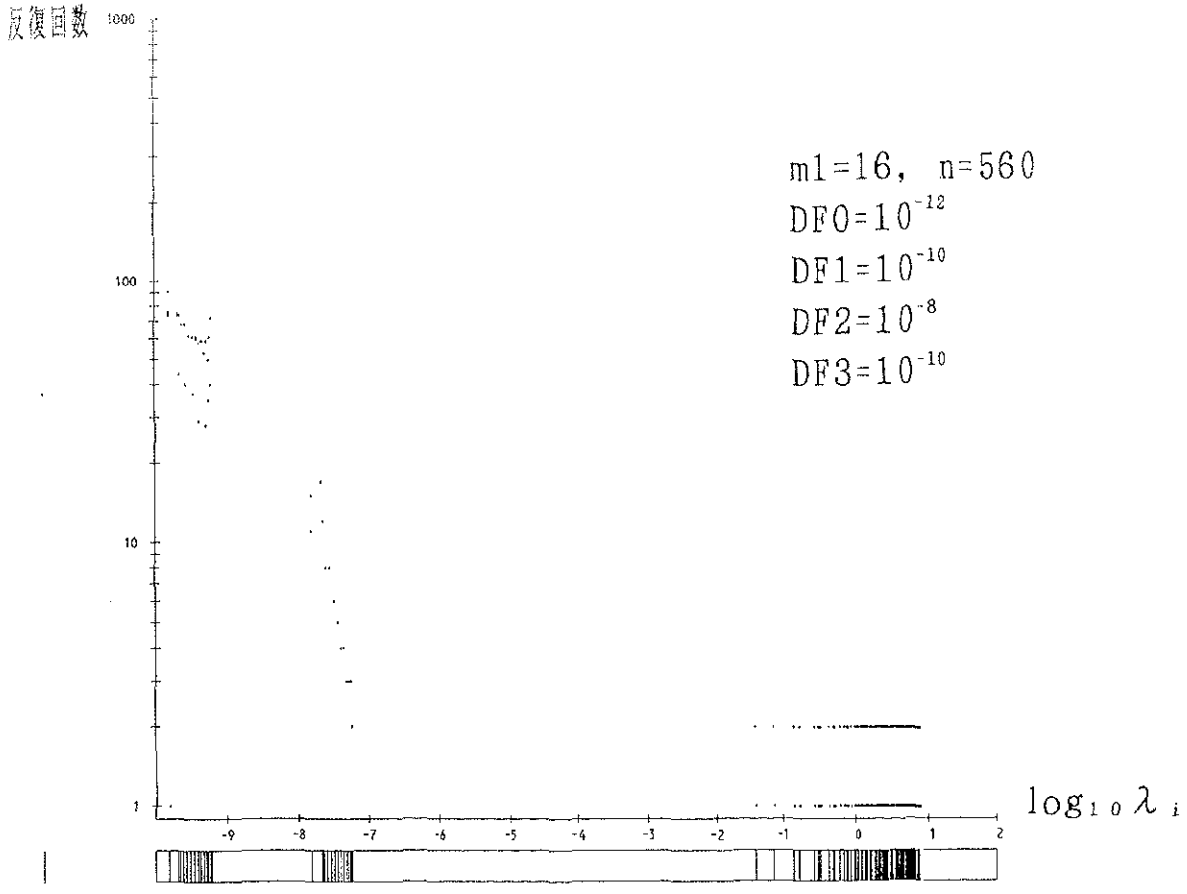


図４.12 固有値分布$\log_{10}\lambda_i$と$\sum_{i=k-1}^{k} v_i$に対するＣＧ法の反復回数

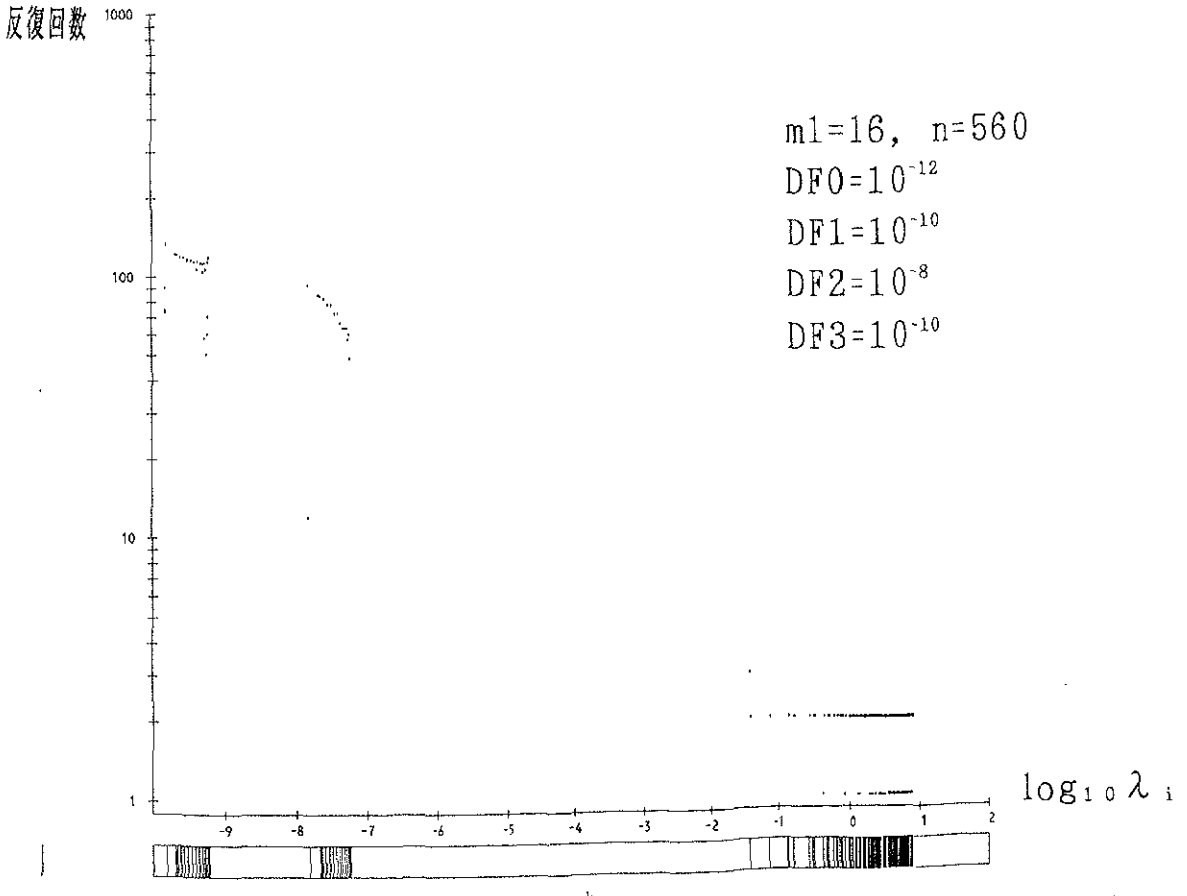


図４.13 固有値分布$\log_{10}\lambda_i$と$\sum_{i=k-2}^{k} v_i$に対するＣＧ法の反復回数

表４．２ 固有値の隣接度と反復回数の関係

$\sum_{i=k-1}^{k} v_i$ を与えて４倍精度で計算

	ml	n	DF0	DF1	DF2	DF3
反復回数 / $\log_{10}\lvert\lambda_{i-1}-\lambda_i\rvert$	16	560	0.10D-11	0.10D-05	0.10D-03	0.10D-05

	1	2	3	4	5	6	7	8	9	10	20	30	40	50	60	70	80	90	100	200	300	400	500	SUM
0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28
-1	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
-2	0	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38
-3	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
-4	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32
-5	0	35	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36
-6	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30
-7	1	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11
-8	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59
-9	51	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	76
-10	99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	99
-11	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20
-12	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
-13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-17	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
-18	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
-19	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
-20	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
-21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	181	378	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	560

表４．３ 固有値の隣接度と反復回数の関係

$\sum_{i=k-1}^{k} v_i$ を与えて４倍精度で計算

	ml	n	DF0	DF1	DF2	DF3
反復回数 / $\log_{10}\lvert\lambda_{i-1}-\lambda_i\rvert$	16	560	0.10D-11	0.10D-09	0.10D-07	0.10D-09

	1	2	3	4	5	6	7	8	9	10	20	30	40	50	60	70	80	90	100	200	300	400	500	SUM
0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28
-1	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
-2	0	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38
-3	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20
-4	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9
-5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-7	0	4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5
-8	0	11	4	2	1	1	0	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	23
-9	29	5	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	36
-10	17	0	0	0	0	0	0	0	0	0	0	0	0	5	6	2	0	0	0	0	0	0	0	30
-11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3	0	0	0	0	0	0	0	4
-12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	3
-13	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	0	0	0	0	0	0	0	0	5
-14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-15	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7
-16	56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56
-17	71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	71
-18	99	0	0	0	0	0	0	0	0	0	2	2	3	0	0	0	0	0	0	0	0	0	0	106
-19	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20
-20	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
-21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	302	211	4	2	1	1	0	2	0	4	2	3	3	7	10	7	0	1	0	0	0	0	0	560

表４．４ 固有値の隣接度と反復回数の関係

$\lambda_i > 10^{-7}$について$\sum_{i=k-1}^{k} v_i$を与えて４倍精度で計算

m1 = 16　　n = 560　　DF0 = 0.10D-11　　DF1 = 0.10D-09　　DF2 = 0.10D-07　　DF3 = 0.10D-09

$\log_{10}(\lambda_{i-1}-\lambda_i)$ \ 反復回数	1	2	3	4	5	6	7	8	9	10	20	30	40	50	60	70	80	90	100	200	300	400	500	SUM
0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28
-1	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
-2	0	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38
-3	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20
-4	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9
-5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-7	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
-8	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9
-9	29	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
-10	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17
-11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-15	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7
-16	56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56
-17	71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	71
-18	99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	99
-19	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20
-20	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
-21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	301	209	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	510

CG法を4倍精度で計算すると、隣接する固有値の差が10^{-10}以下の場合は重複または密集とみなされ、反復回数が減少する。しかし、解を構成する固有ベクトルの本数が増えるにつれて、誤差によって理論からかけ離れていく部分が増える。ここでも小さい固有値に対応した固有ベクトルは収束しにくい。

(d) Σv_i (i=1,k)

低次モード(小さな固有値に対応)の影響をみるため、小さい固有値に対応する固有ベクトル成分から加え合わせたものを用意し、4倍精度で計算した結果を図4.14から図4.16に示す。

図4.14の(000)のケースでも、小さい固有値に対応した固有ベクトル成分を加え合わせていくと急激に反復回数が増加する。10^{-1}をこえて10^{0}に近づくころから、反復回数の増加はゆるやかになり、最終的に560本の固有ベクトルを加え合わせた解はおよそ150回程度の反復で収束している。

図4.15の(646)のケースでは、増え方が(000)よりも急激で、約10本をこえるあたりで560回の反復打切りに達してしまう。図4.16の(A8A)のケースは、増え方が急激で反復打切りに達してしまうのは(646)と同じだが、10^{0}より大きな固有値に対応した固有ベクトルを加えるようになると再び560回以下で収束するようになる。

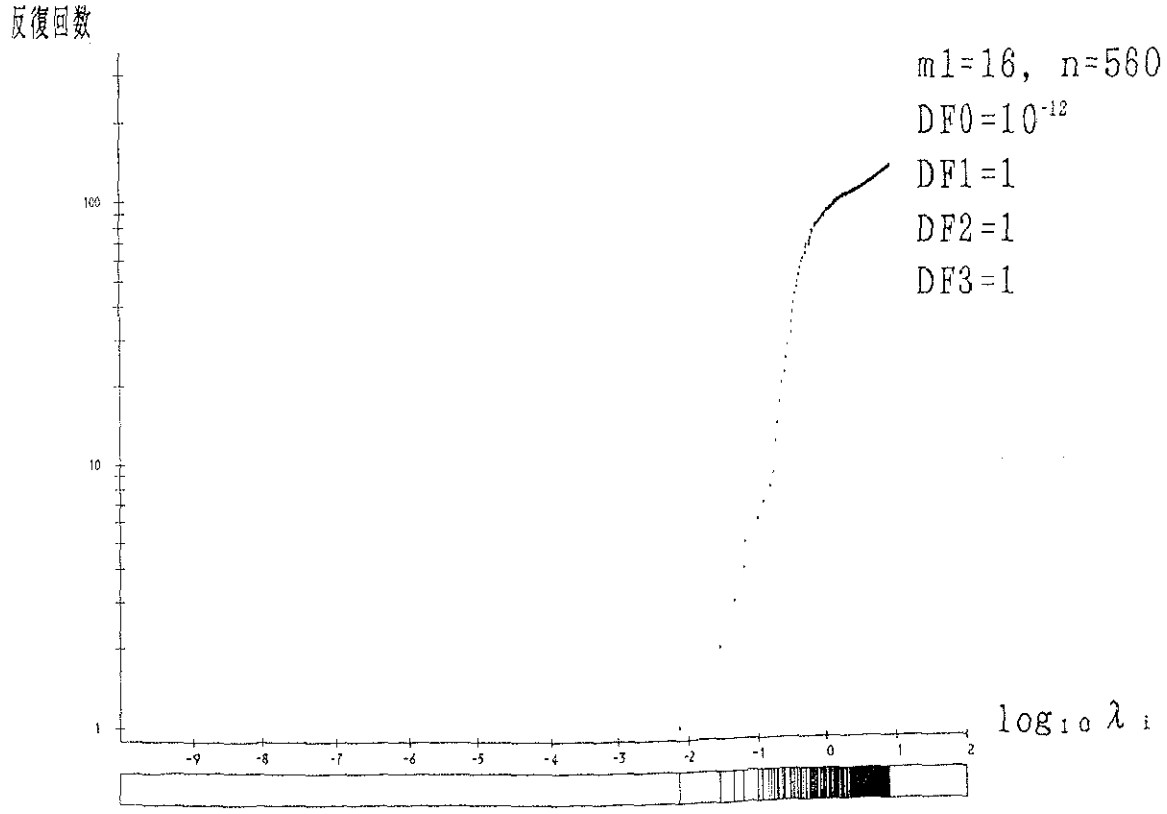


図4.14 固有値分布$\log_{10}\lambda_i$と$\sum_{i=1}^{k} v_i$に対するCG法の反復回数

これから、小さい固有値に対応した固有ベクトル成分が反復回数に対して支配的で、大きな固有値に対応した固有ベクトル成分はそれほど反復回数を増加させないことがわかる。(A8A)で大きな固有ベクトル成分を加え合わせることで反復回数が減少したのは、評価の対象にしている「残差」の意味が小さな固有値だけのときと変わったためである。

なぜなら、残差ノルムは、残差ベクトルに含まれる最大の固有値に対応した固有ベクトル成分によって決まる。行列Aを1回作用させると固有ベクトル成分v_iは$\lambda_i v_i$になることから、反復を1回行うと、残差ベクトル中の各固有ベクトル成分も対応する固有値λ_i倍される。もし固有値の大きさに10^{-7}倍の開きがあれば、固有ベクトルも10^{-7}倍され、残差ノルムに対する影響は10^{-7}倍になる。したがって小さい固有値に対応した固有ベクトル成分だけのときは固有値が小さいことによる収束の遅さが現れていた。しかし、解として固有値の値が大きく異なる2つの固有値群に対応した固有ベクトルを与えた場合は、残差ノルムの評価には大きい固有値に対応した固有ベクトル成分のみが反映し、小さい固有値に対応した固有ベクトル成分は無視されるようになる。しかも、小さな固有値に対応した固有ベクトル成分に対する収束の遅さとも関係なくなる。

倍精度で計算を行った場合も大体の傾向は同じである。(646)のケースについて、縦軸に反復を打ち切られたときの相対残差ノルムをとったグラフを示す。図4.17が4倍精度計算、図4.18が倍精度計算である。同じ560回で打ち切られたとしても、そのときの相対残差ノルムは大きく違う。4倍精度計算のほうが相対残差ノルムが小さいのは当然であるが、加え合わせる固有ベクトル成分に対応する固有値が大きくなるにつれて相対残差ノルムが小さくなる傾向を示しており、その傾向は倍精度計算のほうに顕著である。

倍精度計算の結果と4倍精度計算の結果の違いは演算精度の差と考えられる。そして、そのような精度の差が現れるのは、この場合は10^{-3}以下の固有値に対応した部分である。念のため、加え合わせたもの正規化して固有ベクトルの本数の差(解ベクトルのノルムの差)による違いを調べてみたが、560本程度ではまったく差がなかった。

(A8A)について、線形結合を$\Sigma\lambda_i v_i$としたものを図4.19、$\Sigma v_i/\lambda_i$としたものを図4.20に示す。図4.21に$\Sigma v_i/\lambda_i$の反復を打ち切ったときの相対残差ノルムを示す。これらは4倍精度で計算してある。

$\Sigma\lambda_i v_i$は、小さい固有値に対応した固有ベクトル成分の影響をより小さくするため、Σv_iよりも収束が速い。$\Sigma v_i/\lambda_i$は、小さい固有値に対応した固有ベクトル成分の影響をより大きくし、すべての固有ベクトル成分が右辺bに同じ重みで含まれるようになる。係数行列Aを作用させると小さくなってしまう固有ベクトル成分をより大きな重みで含むため、収束はしにくく、含まれる誤差も大きくなる。

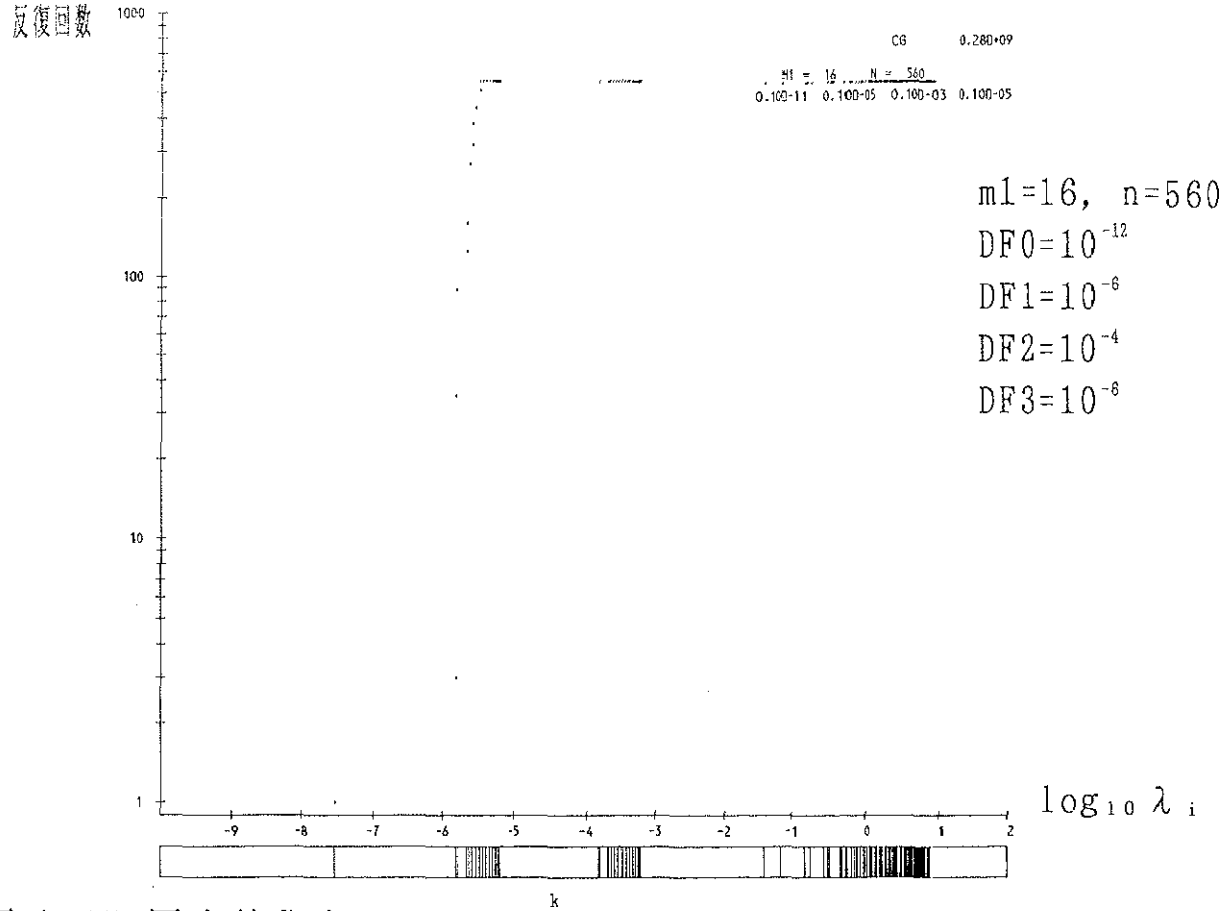


図４.15 固有値分布$\log_{10}\lambda_i$と$\sum_{i=1}^{k} v_i$に対するＣＧ法の反復回数

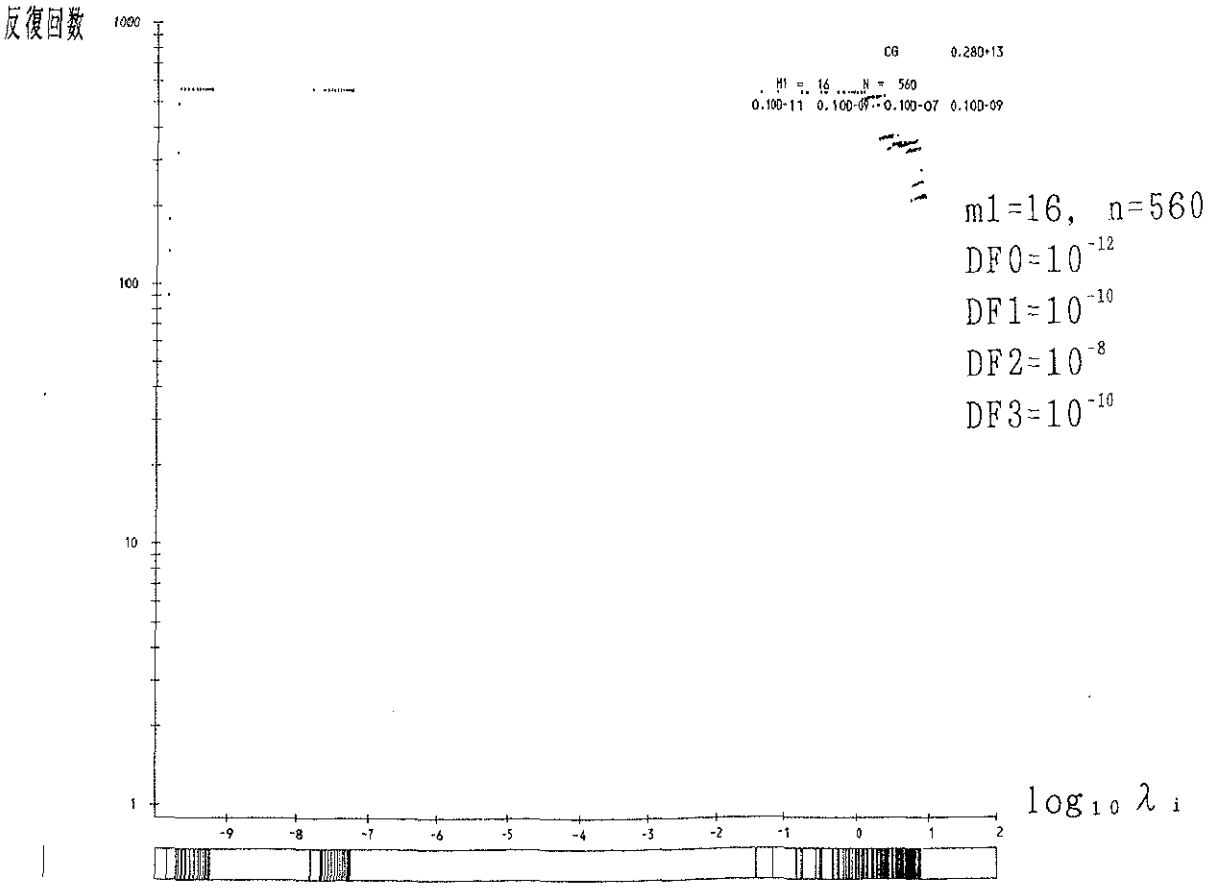


図４.16 固有値分布$\log_{10}\lambda_i$と$\sum_{i=1}^{k} v_i$に対するＣＧ法の反復回数

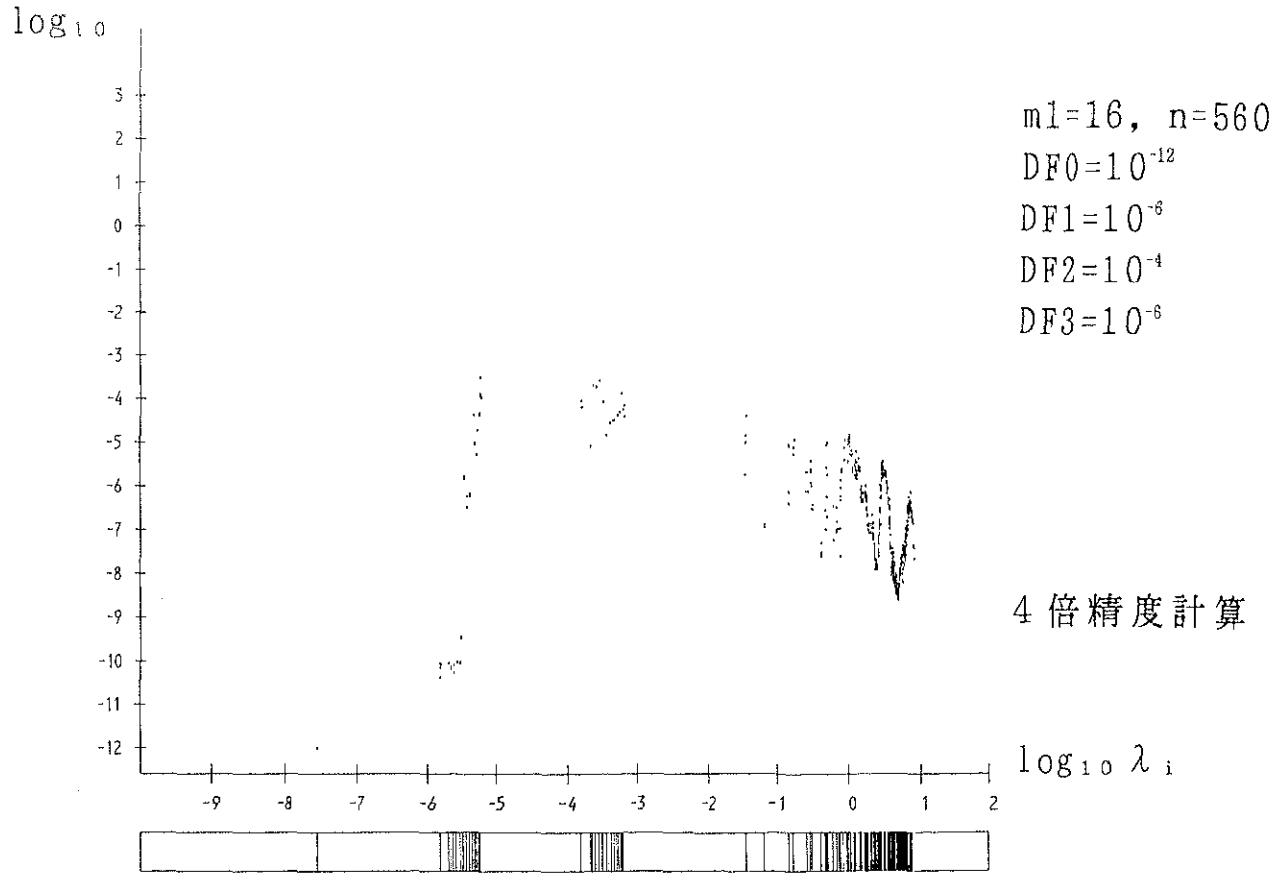


図４.17 固有値分布$\log_{10}\lambda_i$と$\sum_{i=1}^{k} v_i$に対するＣＧ法の収束

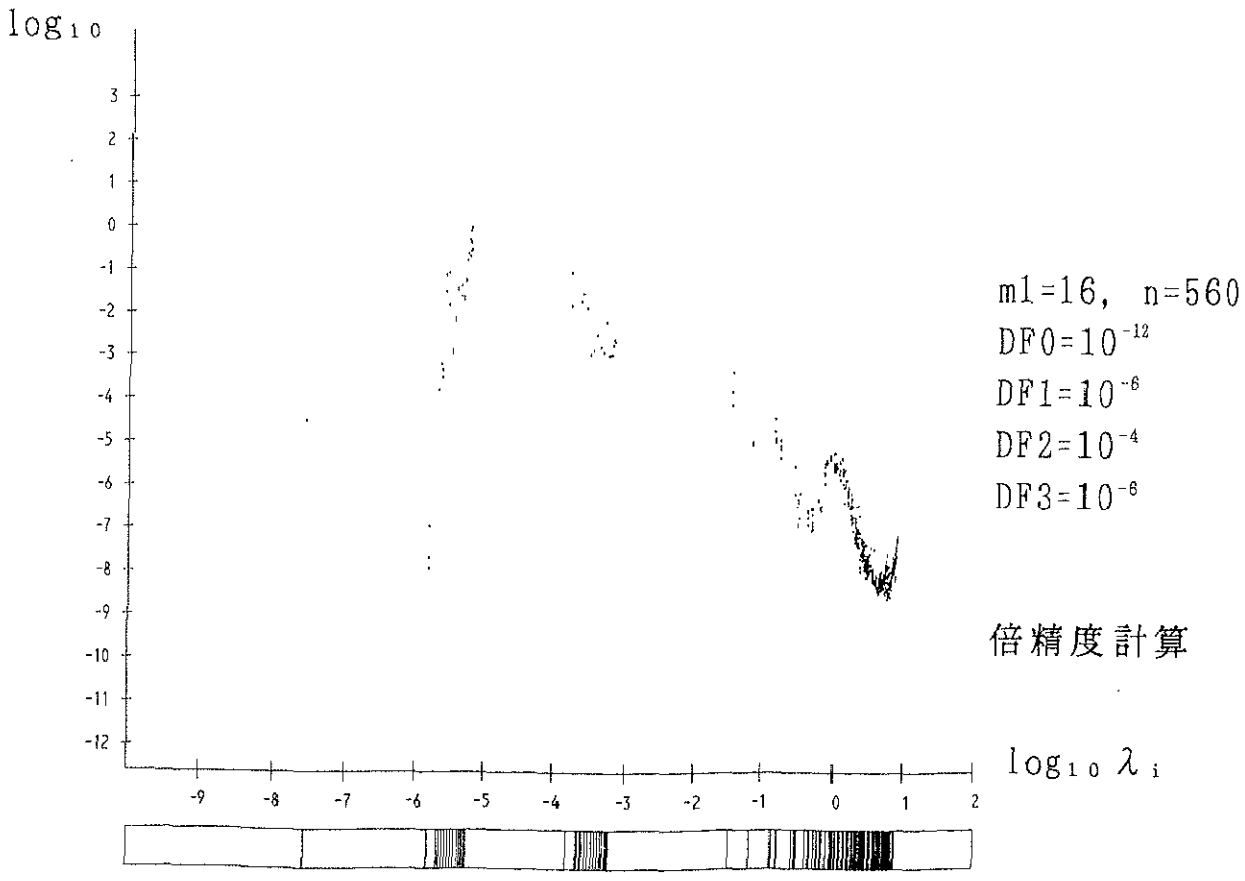


図４.18 固有値分布$\log_{10}\lambda_i$と$\sum_{i=1}^{k} v_i$に対するＣＧ法の収束

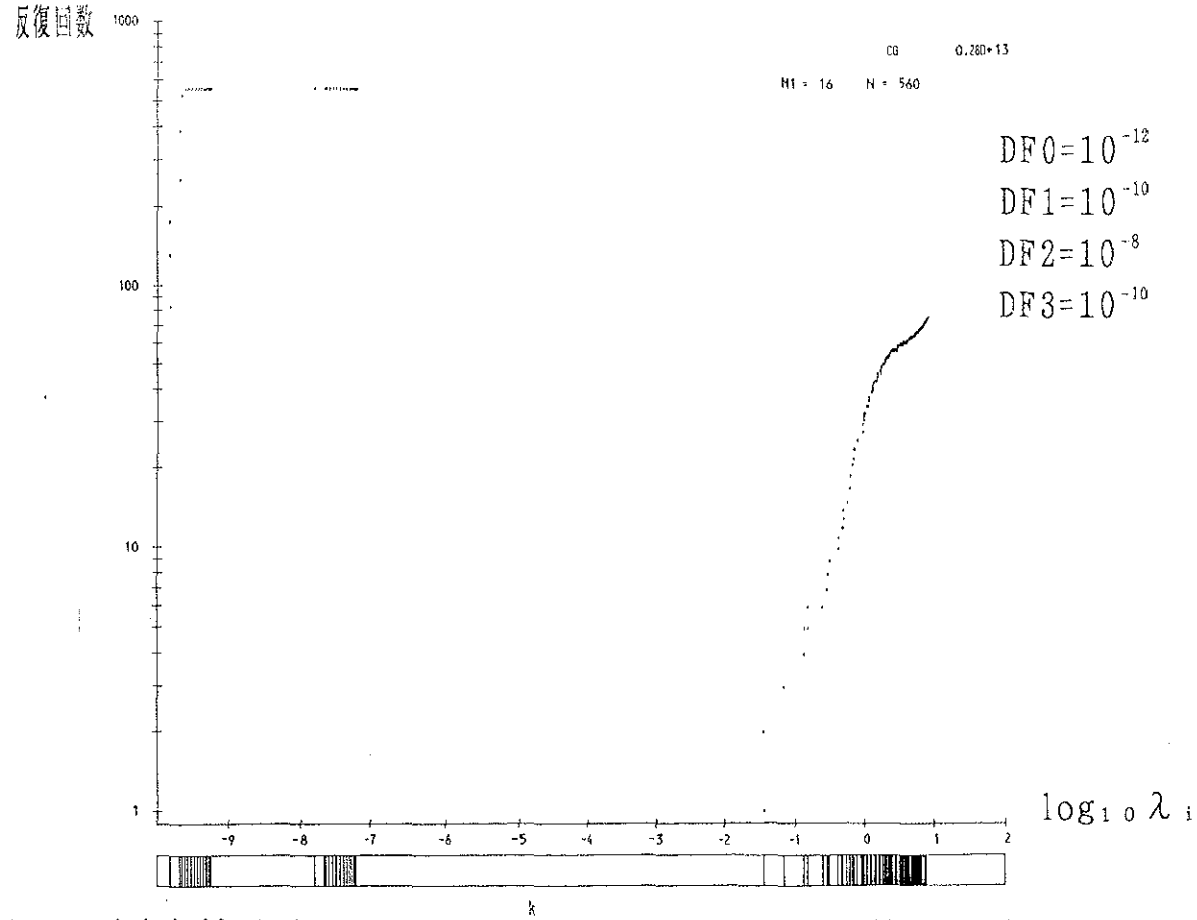


図４．19 固有値分布$\log_{10}\lambda_i$と$\sum_{i=1}^{k}\lambda_i v_i$に対するＣＧ法の反復回数

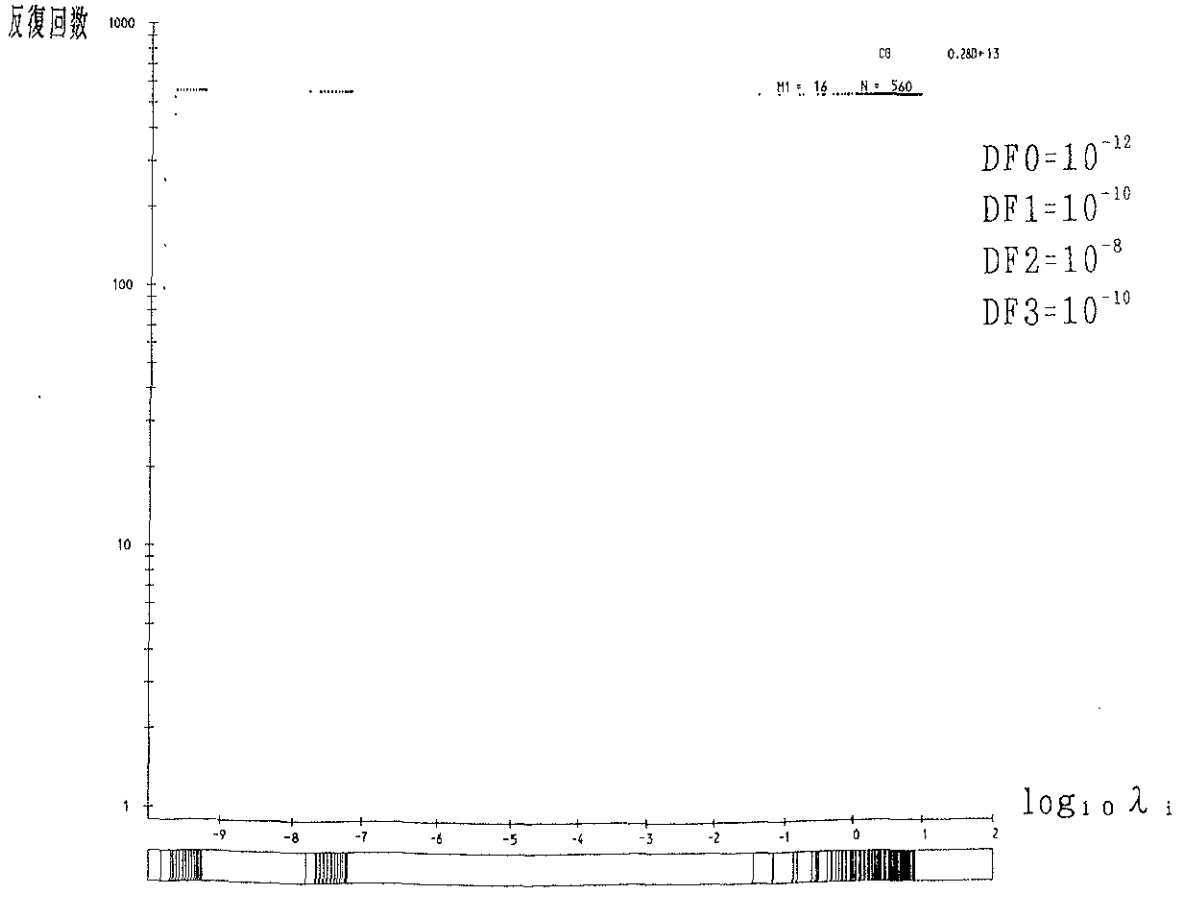


図４．20 固有値分布$\log_{10}\lambda_i$と$\sum_{i=1}^{k}(v_i/\lambda_i)$に対するＣＧ法の反復回数

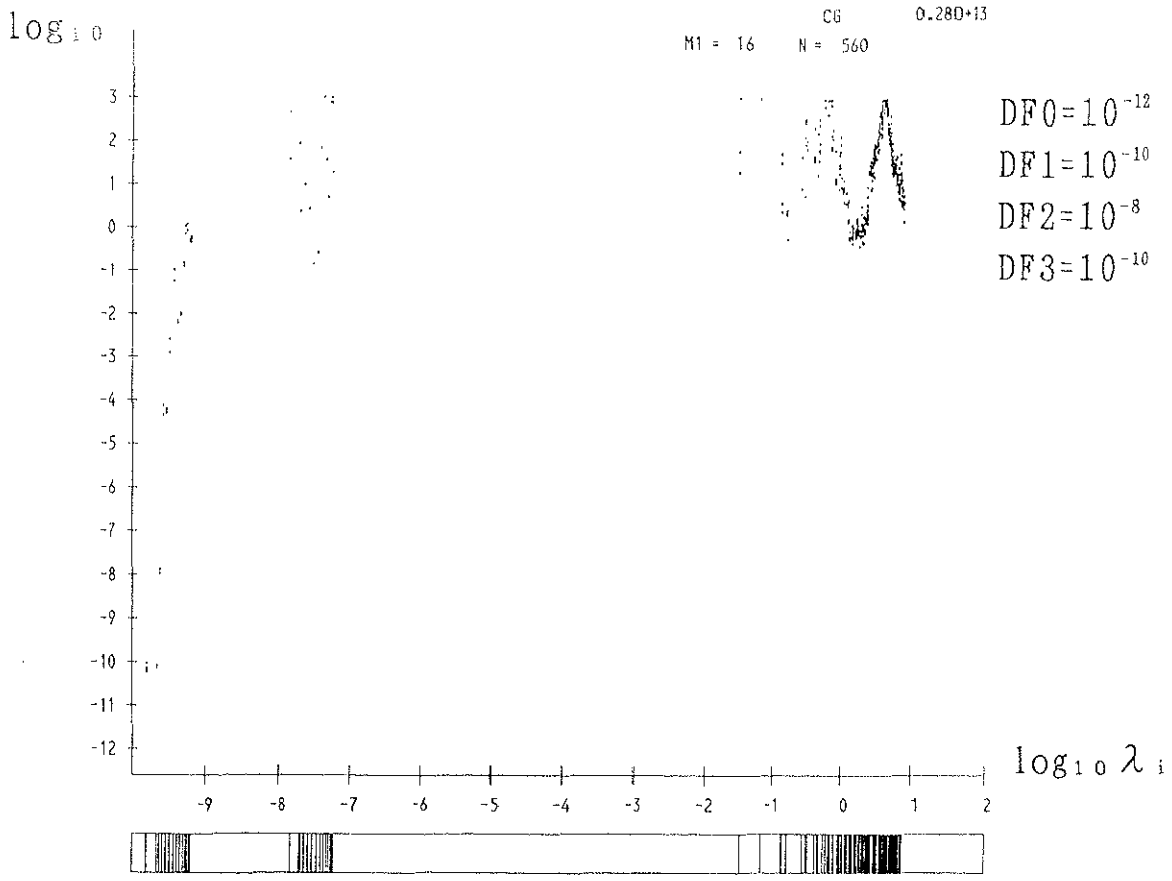


図４．21　４倍精度計算によるＣＧ法の収束

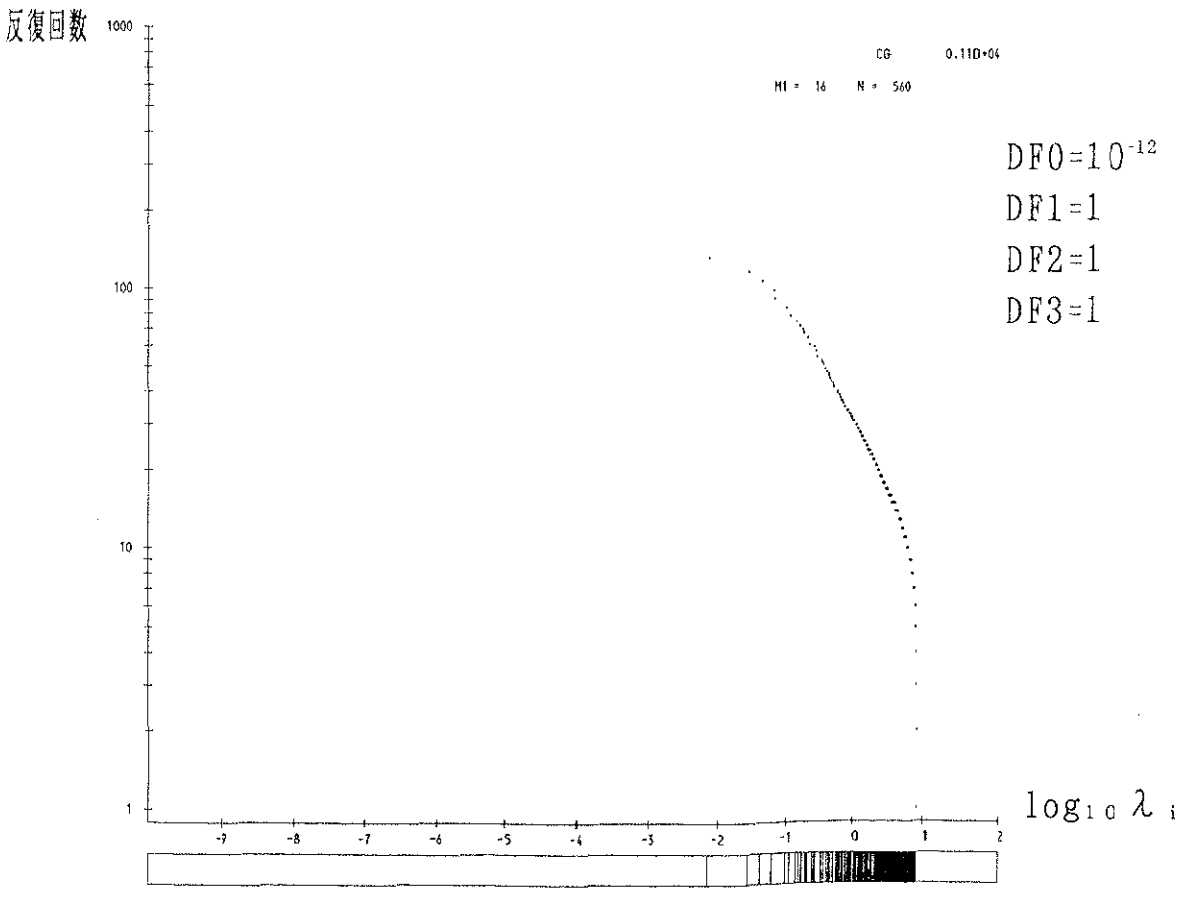


図４．22　固有値分布$\log_{10}\lambda_i$と$\sum_{i=k}^{n} v_i$に対するＣＧ法の反復回数

$\Sigma\lambda_i v_i$の場合、Σv_iとは傾向が大きく異なり、10^{-7}以下の固有値に対応した固有ベクトル成分とそれより大きい固有値に対応した固有ベクトル成分の2つのグループによりはっきりと分かれてしまう。10^{-7}以下の固有値に対応した固有ベクトル成分だけからなる部分は、Σv_iの場合と大差ない。これは解ベクトルのノルムこそ違うものの、線形結合の係数は大差ないためである。前述したように、解が複数の固有値群に属する固有ベクトル成分を含むようになると、解を構成した時点で小さな固有値のグループに属する固有ベクトル成分の影響が弱くなり、第2の固有値群の中だけで小さい固有値に対応する固有ベクトル成分から加え合わせたのと同じになるからである。

$\Sigma v_i/\lambda_i$は、Σv_i以上に収束しにくく、560回の反復で到達できる値もはるかに大きい。このような問題の解をCG法で求めるのは難しい。

(e) Σv_i (i=k,n)

高次モード(大きな固有値に対応)の影響をみるため、大きな固有値に対応する固有ベクトル成分から加え合わせたものを用意した。4倍精度で計算した結果を図4.22から図4.24に示す。

図4.22の(000)のケースのように、大きい固有値に対応した固有ベクトル成分から順に加え合わせていくときの反復回数はゆるやかに増加する。最終的に560本の固有ベクトルを加え合わせた解はおよそ150回程度の反復で収束している。これは低次モードから加え合わせたときと同じになる。

図4.23の(646)のケースでは、別のグループに属する固有値に対応した固有ベクトルを加え合わせるようになると、急激に反復回数が増加する。この例では大きく4つの固有値群に対応して反復回数が増加している。

図4.24の(A8A)のケースでは、別の固有値群に属する固有ベクトル成分を加え合わせても反復回数はほとんど増加しない。これは固有値のオーダーに約10^{10}の開きがあるためで、この場合CG法の反復過程では10^{-9}以下の固有値に対応した固有ベクトル成分が無視されている。しかも(646)よりも反復回数の増え方はゆるやかである。

これからも小さい固有値に対応した固有ベクトル成分が反復回数に対して支配的であることがわかる。しかし、最大固有値と最小固有値の開きが大きくなると、小さな固有値に対応した固有ベクトル成分は残差ベクトルに対して影響を持たなくなる。必然的に、解ベクトル中の小さい固有値に対応した固有ベクトル成分は誤差の影響を強く受け、反復過程で更新される残差ベクトルと解から計算し直した残差ベクトルの間に差がでるようになる。この場合も、正規化の効果はなかった。

(A8A)について、線形結合を$\Sigma\lambda_i v_i$としたものを図4.25、$\Sigma v_i/\lambda_i$としたものを図4.26に示す。

$\Sigma\lambda_i v_i$は、小さい固有値に対応した固有ベクトル成分の影響をより小さくするため、10^{-7}以下の固有値に対応した固有ベクトル成分がほとんど影響しなくなっている。逆に$\Sigma v_i/\lambda_i$は、小さい固有値に対応した固有ベクトル成分の影響をより大きくするため、10^{-9}以下の固有値に対応した固有ベクトル成分が含まれるようになると、いちだんと収束がしにくくなる。

これらの実験から明らかになったことは次のようにまとめられる。

(1) CG法を倍精度で実行する場合、最大固有値が10^{1}のときに10^{-3}以下の固有値に対応した固有ベクトル成分は収束しにくい。
(2) 4倍精度計算の場合、重複とみなせるのは固有値の差が10^{-10}以下で、重複の数だけ反復回数が減少する。
(3) 小さな固有値に対応した固有ベクトル成分を多く含むと反復回数、誤差とも大きくなる。
(4) 複数の固有ベクトル成分を与えると理論上は本数分だけの反復回数で収束するが、実際は誤差の影響によって小さな固有値に対応した固有ベクトル成分に対しては理論どおりにはいかない。
(5) 収束は固有値の分布に強く関係するが、最大固有値と最小固有値だけでは情報不足である。

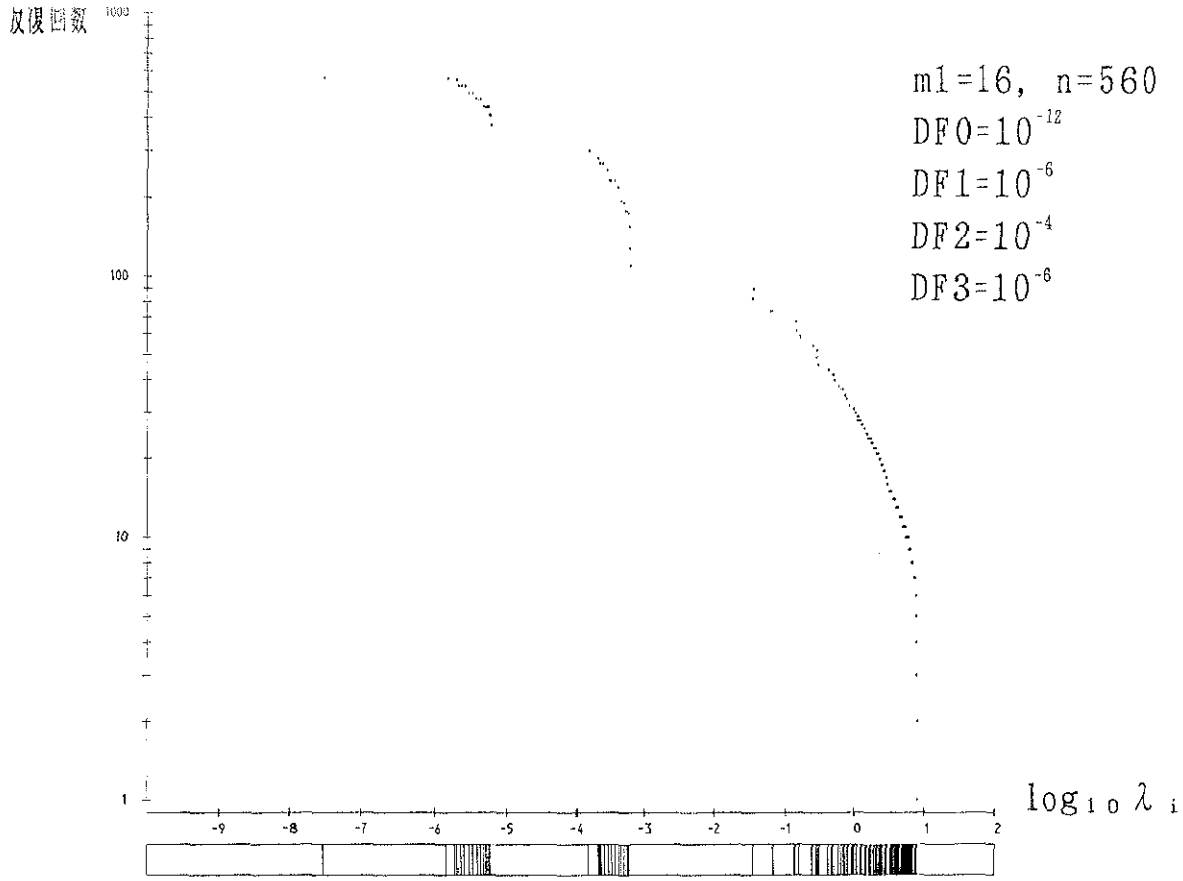


図４．23 固有値分布$\log_{10}\lambda_i$と$\sum_{i=k}^{n} v_i$に対するＣＧ法の反復回数

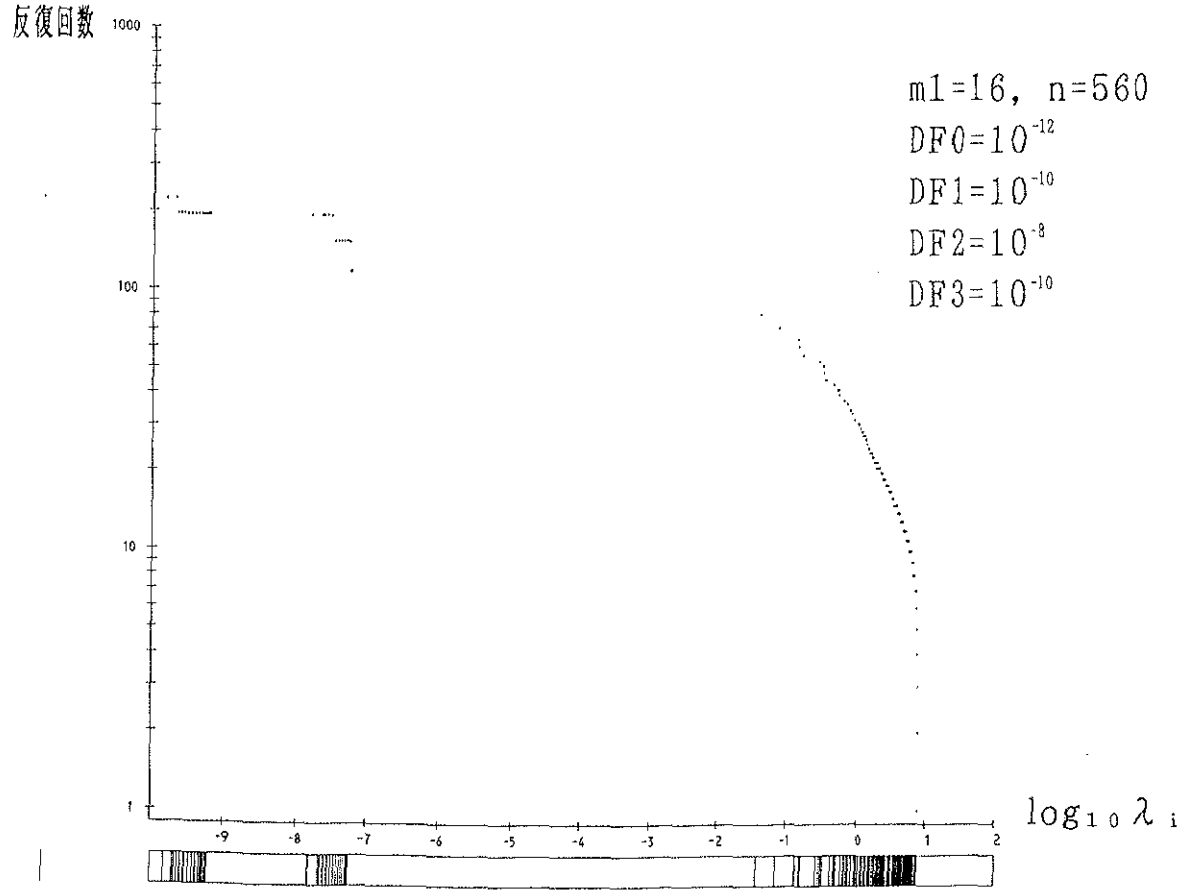


図４．24 固有値分布$\log_{10}\lambda_i$と$\sum_{i=k}^{n} v_i$に対するＣＧ法の反復回数

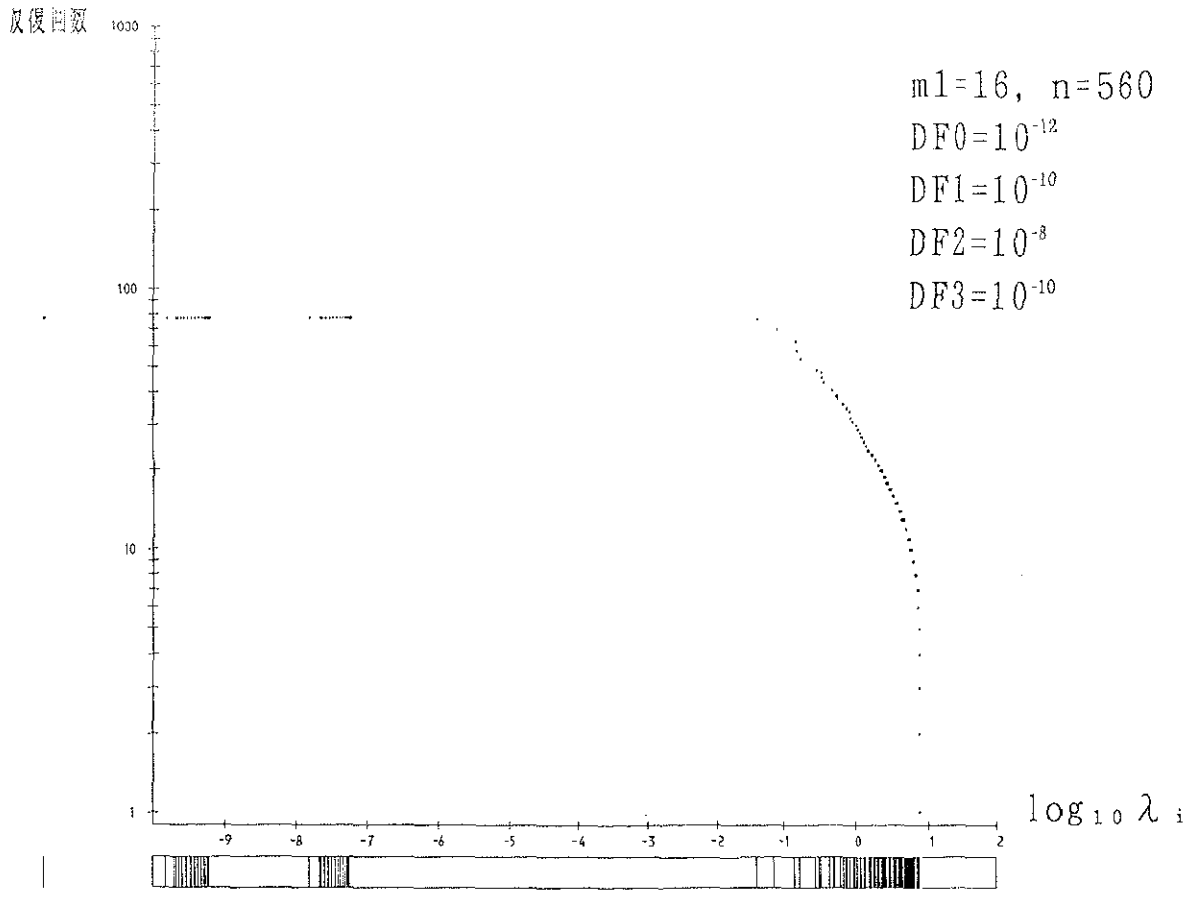


図４．25 固有値分布$\log_{10}\lambda_i$と$\sum_{i=k}^{n}\lambda_i v_i$に対するＣＧ法の反復回数

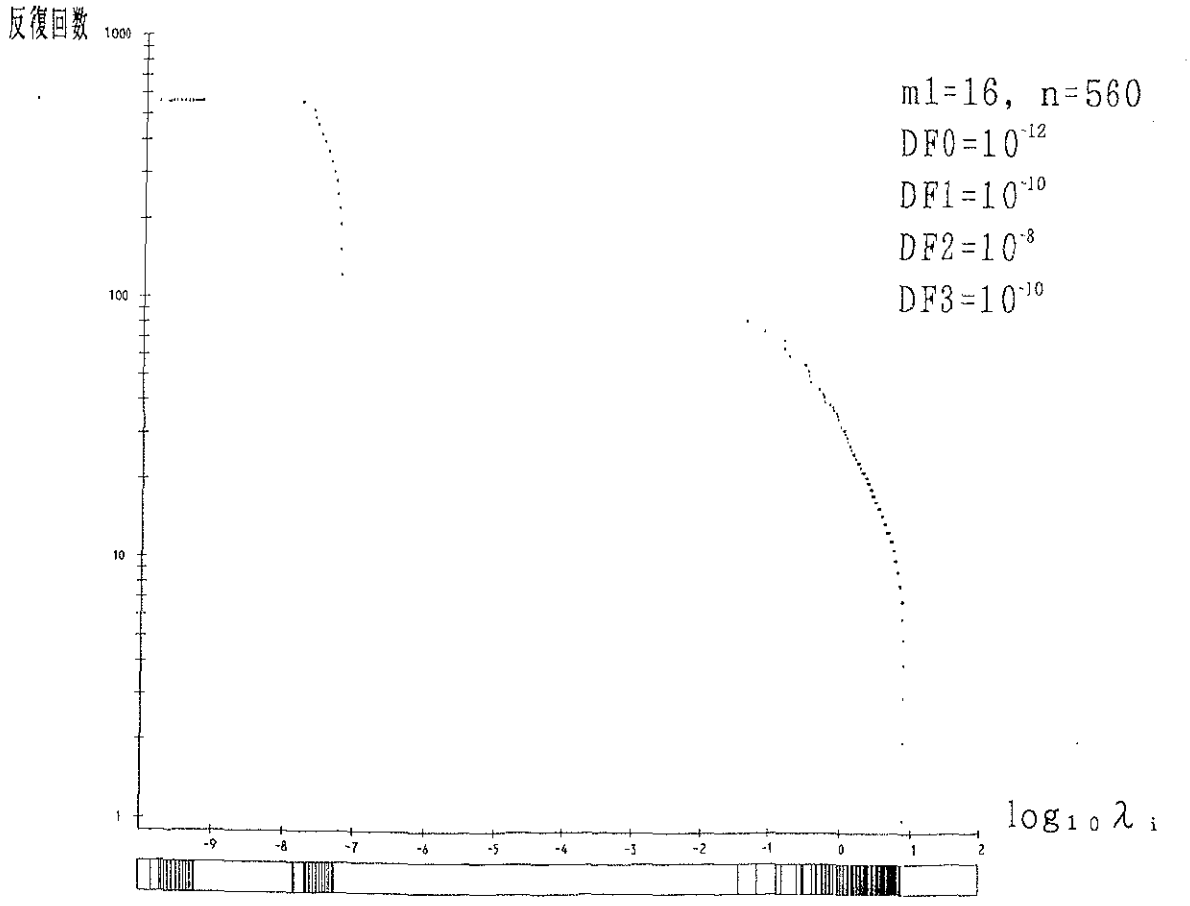


図４．26 固有値分布$\log_{10}\lambda_i$と$\sum_{i=k}^{n}(v_i/\lambda_i)$に対するＣＧ法の反復回数

5. CG法の収束の改善

CG法は対称で正定値な行列Aを係数にもつ連立一次方程式$Ax=b$に対する反復解法であり、収束特性は行列Aの固有値･固有ベクトルの分布と、方程式の右辺bまたは解xがその固有ベクトル成分をどのように含むかによって大きく異なってくる。本研究で解こうとしている問題はある物理的な場における拡散方程式を差分法で離散化したものであり、係数行列や右辺、解などはきわめて広いバラエティをもつ。このような問題に対して、数学的に何か適当な変換を施し、問題をより解きやすい形にしたうえで計算を行うことが多い。ここでは、代表的な方法としての前処理付き共役勾配法(Preconditioned Conjugate Gradient Method)をとりあげる。なかでもMeijerinkの不完全U^TDU分解を用いたICCG法(Incomplete Cholesky Conjugate Gradient Method)と、ICCG法にGustafsson流の変更を加えたMICCG法(Modified Incomplete Cholesky Conjugate Gradient Method)が安定で速く収束することが知られているので、これらについて詳しい解析を行う。

5.1 PCG法(Preconditioned Conjugate Gradient Method)

PCG法とは前処理付き共役勾配法(Preconditioned Conjugate Gradient Method)の略称で、前処理を施した方程式にCG法を適用する算法である[51]。PCG法ではAに近い対称正定値行列$M=\widetilde{U}^T\widetilde{U}$($\widetilde{U}$は上三角行列)を用いて、$Ax=b$の左から$\widetilde{U}^{-T}$を作用させ、変換$x=\widetilde{U}^{-1}y$を用いて

$$\widetilde{U}^{-T}A\widetilde{U}^{-1}y=\widetilde{U}^{-T}b,\quad x=\widetilde{U}^{-1}y \tag{5.1}$$

の形にしてからCG法を適用する。Aが対称正定値ならば$\widetilde{U}^{-T}A\widetilde{U}^{-1}$も対称正定値となり、CG法が適用できる。PCG法では$\widetilde{U}$をどのようにとるかによって収束が異なる。$\widetilde{U}$が単位行列ならばCG法になる。$\widetilde{U}^T\widetilde{U}=A$ならば$x=\widetilde{U}^{-1}\widetilde{U}^{-T}b=A^{-1}b$となり、CG法を適用するまでもなく直接解を求めたことになる。これから、$M=\widetilde{U}^T\widetilde{U}$を$A$に近くできれば効果的なことがわかる。

PCG法のアルゴリズムを示す[51],[25]。まず(5.1)式を

$$B=\widetilde{U}^{-T}A\widetilde{U}^{-1}$$
$$y=\widetilde{U}x$$
$$c=\widetilde{U}^{-T}b$$

とおいて、$By=c$に対してCG法を適用すると以下のアルゴリズムになる。

(I)

① y_0:初期解を用意; ② $\widetilde{r}_0=c-By_0$:初期残差ベクトル

③ $\widetilde{p}_0=\widetilde{r}_0$: 初期修正ベクトル; $k=0$

while $\|r_k\| > \varepsilon$ do

④ $\alpha_k=(\widetilde{r}_k,\widetilde{r}_k)/(\widetilde{p}_k,B\widetilde{p}_k)$

⑤ $y_{k+1}=y_k+\alpha_k\widetilde{p}_k$; ⑥ $\widetilde{r}_{k+1}=\widetilde{r}_k-\alpha_kB\widetilde{p}_k$

⑦ $\beta_k=(\tilde{r}_{k+1},\tilde{r}_{k+1})/(\tilde{r}_k,\tilde{r}_k)$

⑧ $\tilde{p}_{k+1}=\tilde{r}_{k+1}+\beta_k\tilde{p}_k$; $k=k+1$

実際の計算をA、x、bで行うため、①~⑧をB、y、cを用いた式からA、x、bを用いた式に直す。すなわち

$$y=\tilde{U}x$$
$$\tilde{r}=\tilde{U}^{-T}r$$
$$\tilde{p}=\tilde{U}p$$

を利用して①~⑧を変形する。

① $y_0=Ux_0$

② $\tilde{r}_0=c-By_0$
$=\tilde{U}^{-T}b-\tilde{U}^{-T}A\tilde{U}^{-1}\tilde{U}x_0=\tilde{U}^{-T}(b-Ax_0)$から
$\tilde{U}^T\tilde{r}_0=b-Ax_0=r_0$

③ $\tilde{p}_0=\tilde{r}_0$は$p_0=\tilde{U}^{-1}\tilde{U}^{-T}r_0=(\tilde{U}^T\tilde{U})^{-1}r_0$

④ $(\tilde{r}_k,\tilde{r}_k)=(\tilde{U}^{-T}r_k,\tilde{U}^{-T}r_k)=(r_k,\tilde{U}^{-1}\tilde{U}^{-T}r_k)=(r_k,(\tilde{U}^T\tilde{U})^{-1}r_k)$
$(\tilde{p}_k,B\tilde{p}_k)=(\tilde{U}p_k,\tilde{U}^{-T}A\tilde{U}^{-1}\tilde{U}p_k)=(p_k,Ap_k)$

⑤ $y_{k+1}=y_k+\alpha_k\tilde{p}_k$に$\tilde{U}^{-1}$を作用させた
$\tilde{U}^{-1}y_{k+1}=\tilde{U}^{-1}y_k+\alpha_k\tilde{U}^{-1}\tilde{U}p_k$から$x_{k+1}=x_k+\alpha_k p_k$

⑥ $\tilde{r}_{k+1}=\tilde{r}_k-\alpha_k B\tilde{p}_k$に$\tilde{U}^T$を作用させた
$\tilde{U}^T\tilde{r}_{k+1}=\tilde{U}^T\tilde{r}_k-\alpha_k\tilde{U}^T\tilde{U}^{-T}A\tilde{U}^{-1}\tilde{U}p_k$から$r_{k+1}=r_k-\alpha_k Ap_k$

⑦ ④と同様に$(\tilde{r}_{k+1},\tilde{r}_{k+1})=(r_{k+1},(\tilde{U}^T\tilde{U})^{-1}r_{k+1})$

⑧ $\tilde{p}_{k+1}=\tilde{r}_{k+1}+\beta_k\tilde{p}_k$に$\tilde{U}^{-1}$を作用させた
$\tilde{U}^{-1}\tilde{U}p_{k+1}=\tilde{U}^{-1}\tilde{U}^{-T}r_{k+1}+\beta_k\tilde{U}^{-1}\tilde{U}p_k$から$p_{k+1}=(\tilde{U}^T\tilde{U})^{-1}r_{k+1}+\beta_k p_k$

これから(I)のアルゴリズムはA、x、bを用いると次のように書ける。

(II)

x_0:初期解を用意; $r_0=b-Ax_0$:初期残差ベクトル
$p_0=(\tilde{U}^T\tilde{U})^{-1}r_0$:初期修正ベクトル; $k=0$
while $\|r_k\|>\varepsilon$ do
$\alpha_k=(r_k,(\tilde{U}^T\tilde{U})^{-1}r_k)/(p_k,Ap_k)$
$x_{k+1}=x_k+\alpha_k p_k$; $r_{k+1}=r_k-\alpha_k Ap_k$
$\beta_k=(r_{k+1},(\tilde{U}^T\tilde{U})^{-1}r_{k+1})/(r_k,(\tilde{U}^T\tilde{U})^{-1}r_k)$
$p_{k+1}=(\tilde{U}^T\tilde{U})^{-1}r_{k+1}+\beta_k p_k$; $k=k+1$

PCG法では理論的にMが$\tilde{U}^T\tilde{U}$と分解できることが大事で、実際には$M^{-1}=(\tilde{U}^T\tilde{U})^{-1}$が容易に作用させられればよい。逆行列の計算は計算量、精度の点で問題が多いので数値計算では用いないのが常識である。そこで実際のプログラムでは

$(\widetilde{U}^T\widetilde{U})^{-1}$を作らずに済ます。$(\widetilde{U}^T\widetilde{U})^{-1}r$を作るには$z=(\widetilde{U}^T\widetilde{U})^{-1}r$とおいて

$$\widetilde{U}^{-1}(\widetilde{U}^T)^{-1}r=z$$

を解けばよい。ここで$\widetilde{U}$は上三角行列である。まず$w=(\widetilde{U}^T)^{-1}r$とおいて

$$\widetilde{U}^T w=r$$

から前進代入によってwを求める。次に$z=\widetilde{U}^{-1}w$とおいて

$$\widetilde{U}z=w$$

から後退代入によってzを求める。

PCG法の効果は前処理$M=\widetilde{U}^T\widetilde{U}$の選び方によって異なるので、具体的に$\widetilde{U}$として何を選ぶかが問題となる。以下では、Meijerinkの不完全U^TDU分解を用いたICCG法と[41],[42],[51]、ICCG法にGustafsson流の変更を加えたMICCG法[18],[51],[71]について示す。ICCG法はMeijerinkの不完全コレスキー分解を前処理として用いた共役勾配法(Incomplete Cholesky Conjugate Gradient Method)の略称で、1977年にMeijerinkとvan der Vorstによって発表され、1981年には同一の著者によってユーザ向けのGuidelinesが発表されている[42]。

一般に対称正定値行列Aを

$$A=U^TDU+R \qquad (D\text{は対角行列、}U\text{は上三角行列}) \qquad (5.2)$$

に分解する方法を不完全コレスキー分解といい[7]、対角行列Dの要素をd_i、上三角行列Uの要素を$u_{i,j}$で表す(以後、不完全コレスキー分解を不完全U^TDU分解ともいう)。

なお、PCG法、特にICCG法では理論上前処理行列が$\widetilde{U}^T\widetilde{U}$と分解できることが必要なだけで、実用上は$U^TDU$と分解されていてもなんら問題はない。$U^TDU$とするか$\widetilde{U}^T\widetilde{U}$とするかは、前処理の作りやすさ、プログラムの速さに関係する。必要ならば、U^TDUを$U^T\sqrt{D}\sqrt{D}U$とすれば$(\sqrt{D}U)^T(\sqrt{D}U)$と分解できるし、$(U^TDU)^{-1}=U^{-1}D^{-1}U^{-T}$を作用させるのも容易である。しかもMeijerinkの不完全U^TDU分解の場合はAがM-行列なら$\widetilde{d}_i^{-1}>0$が保証されている[41]。M-行列の定義は

$$A=sI-B、\ s>0、\ B\geqq 0、\ s\geqq S(B)$$

で[76]、十分条件は

$$a_{ii}>0、\ a_{ij}\leqq 0、\quad A^{-1}>0$$

である[46]。

Meijerinkが用いた不完全U^TDU分解では、

$$G\subset\{(i,j);a_{i,j}\neq 0\}$$

となる格子集合Gを指定して$(i,j)\subset G$のときだけ$u_{i,j}$を作る。格子集合Gを大きくとるとU^TDUはよりAに近づくが、メモリ容量や計算時間が多く必要となる。

Meijerinkの不完全$U^T DU$分解ではDを

$$u^T_{i,i}\tilde{d}_i u_{i,i} = u^T_{i,i} = u_{i,i} \tag{5.3}$$

とする。そのため、$\tilde{d}_i = u_{i,i}^{-1}$、$u_{i,i}\tilde{d}_i = 1$、$u_{i,i}^2\tilde{d}_i = u_{i,i}$になる。

このUとDは、$A = U^T DU$を要素で表した関係式

$$a_{i,j} = \Sigma u^T_{i,k}\tilde{d}_k u_{k,j} \tag{5.4}$$

から、以下のアルゴリズムによって求められる[51],[25]。

```
do i = 1, n
   u_{i,i} = a_{i,i} - Σ u_{k,i}^2 d̃_k ;   d̃_i = u_{i,i}^{-1}
   do j = i+1, n
      if (i,j) ∈ G  then
         u_{i,j} = a_{i,j} - Σ u_{k,i} d̃_k u_{k,j}
```

対象としている問題は2次元の矩形領域における拡散方程式を差分法で離散化して得られた、対称で正定値の規則的疎行列なので、文献[42]のGuidelinesに沿って4種類の前処理(2次元5点差分法用の不完全$U^T DU$分解)が適用できる。これら4種類の前処理をMeijerinkの呼び方にならってICCG(1,1)、ICCG(1,2)、ICCG(1,3)、ICCG(2,4)と呼ぶ。なお、一般にICCGという名称はCG法の適用までを含めるが、ここでは反復過程と収束特性解析との対応をわかりやすくするため、前処理だけの場合にもICCGと呼ぶ。

また、2次元の場を差分法で離散化したときの係数行列Aは、非対角帯半幅m1、元数nとすると、第i番方程式は

$$a_{i,i-m1}x_{i-m1} + a_{i,i-1}x_{i-1} + a_{i,i}x_i + a_{i,i+1}x_{i+1} + a_{i,i+m1}x_{i+m1} = f_i$$

のように、非ゼロ要素は5本の対角部分で構成される。以下では図5.1で示したように、方程式番号iを固定したとき、この上三角行列の対角要素をa_i、対角の右どなりの要素をb_i、対角からm1離れた要素をc_iで表している。

(1) ICCG(1,1)

Meijerinkの不完全$U^T DU$分解で、一番簡単なのは

$$G = \{(i,j); a_{i,j} \neq 0\}$$

である。このときにはAの非対角要素をそのままUの非対角要素に使い、対角要素は$\tilde{d}_i u_{i,i} = 1$となるように決めればよい。Dの要素を$\tilde{d}_i$、Uの非ゼロ要素を対角から順に$\tilde{d}_i^{-1}$, $\tilde{b}_i$, $\tilde{c}_i$とする。要素間の関係式として表すと、

$$a_{i,j} = \Sigma u^T_{i,k}\tilde{d}_k u_{k,j}$$

となるが、$u^T_{i,k}$と$u_{k,j}$が非ゼロになるのはkがi,i-1,i-m1のときだけなので

$$a_{i,j} = u^T_{i,i}\tilde{d}_i u_{i,j} + u^T_{i,i-1}\tilde{d}_{i-1} u_{i-1,j} + u^T_{i,i-m1}\tilde{d}_{i-m1}u_{i-m1,j}$$

となる。これから$u_{i,i+1} = a_{i,i+1}$、$u_{i,i+m1} = a_{i,i+m1}$が導かれる。Meijerinkによる条

件$\tilde{d}_i u_{i,i} = 1$を用いて対角要素を求めると

$$u_{i,i} = \tilde{d}_i^{-1} = a_{i,i} - u^T_{i,i-1} \tilde{d}_{i-1} u_{i-1,i} - u^T_{i,i-m1} \tilde{d}_{i-m1} u_{i-m1,i}$$

となる。Aの要素を用いて書き直すと

$$\tilde{d}_i^{-1} = u_{i,i} = a_{i,i} - a_{i,i-1} \tilde{d}_{i-1} a_{i-1,i} - a_{i,i-m1} \tilde{d}_{i-m1} a_{i-m1,i}$$
$$= a_i - b_{i-1}^2 \tilde{d}_{i-1} - c_{i-m1}^2 \tilde{d}_{i-m1}.$$

ICCG(1,1)用の不完全$U^T DU$分解アルゴリズムは以下のようになる。

do i = 1, n
$\tilde{d}_i^{-1} = a_i - b_{i-1}^2 \tilde{d}_{i-1} - c_{i-m1}^2 \tilde{d}_{i-m1}$
$\tilde{b}_i = b_i$
$\tilde{c}_i = c_i$

ICCG(1,1)法を実行するだけならこれで充分だが、収束特性の解析用に$U^T DU$を求めておく。

$$(DU)_{i,j} = \tilde{d}_i u_{i,j}$$
$$(U^T DU)_{i,j} = \Sigma u^T_{i,k} (DU)_{k,j} \qquad (5.5)$$

kのとりうる値はi,i-1,i-m1の3通りだから(5.5)式は

$$(U^T DU)_{i,j} = u^T_{i,i} \tilde{d}_i u_{i,j} + u^T_{i,i-1} \tilde{d}_{i-1} u_{i-1,j} + u^T_{i,i-m1} \tilde{d}_{i-m1} u_{i-m1,j} \qquad (5.6)$$

となる。(5.6)式から、各iについてjのとりうる値は①i、②i+1、③i+m1、④i+m1-1の4通りである。Aのi+m1-1列はゼロであったが、不完全$U^T DU$分解の結果から$U^T DU$を作ると非ゼロ要素になる。このような非ゼロ要素をfill-in と呼ぶ。

①j=iのとき

$$(U^T DU)_{i,i} = u^T_{i,i} \tilde{d}_i u_{i,i} + u^T_{i,i-1} \tilde{d}_{i-1} u_{i-1,i} + u^T_{i,i-m1} \tilde{d}_{i-m1} u_{i-m1,i}$$

$u_{ii} = \tilde{d}_i^{-1}$から

$$= u_{i,i} + u^T_{i,i-1} \tilde{d}_{i-1} u_{i-1,i} + u^T_{i,i-m1} \tilde{d}_{i-m1} u_{i-m1,i}$$

a、b、c、$\tilde{d}$を用いて書き直すと

$$= \tilde{d}_i^{-1} + \tilde{b}_{i-1}^2 \tilde{d}_{i-1} + \tilde{c}_{i-m1}^2 \tilde{d}_{i-m1}$$
$$= a_i \text{(行列Aの対角部分)}.$$

②j=i+1のとき

$$(U^T DU)_{i,i+1} = u^T_{i,i} \tilde{d}_i u_{i,i+1} + u^T_{i,i-1} \tilde{d}_{i-1} u_{i-1,i+1} + u^T_{i,i-m1} \tilde{d}_{i-m1} u_{i-m1,i+1}$$

$u_{i-1,i+1} = 0$、$u_{i-m1,i+1} = 0$なので

$$= \tilde{b}_i = b_i.$$

③j=i+m1のとき

$$(U^T DU)_{i,i+m1} = u^T_{i,i} \tilde{d}_i u_{i,i+m1} + u^T_{i,i-1} \tilde{d}_{i-1} u_{i-1,i+m1} + u^T_{i,i-m1} \tilde{d}_{i-m1} u_{i-m1,i+m1}$$

$$= u_{i,i+m1} + 0 + 0$$
$$= \tilde{c}_i = c_i .$$

④j=i+m1-1のとき

$$(U^T DU)_{i,i+m1-1} = u^T_{i,i}\tilde{d}_i u_{i,i+m1-1} + u^T_{i,i-1}\tilde{d}_{i-1} u_{i-1,i+m1-1} + u^T_{i,i-m1}\tilde{d}_{i-m1} u_{i-m1,i+m1-1}$$

$u_{i,i+m1-1}=0$、$u_{i-m1,i+m1-1}=0$なので

$$= \tilde{b}_{i-1}\,\tilde{d}_{i-1}\,\tilde{c}_{i-1} .$$

このfill-in を$\tilde{e}_i$ とする。Uと$U^T DU$の関係を図5.2に示す。

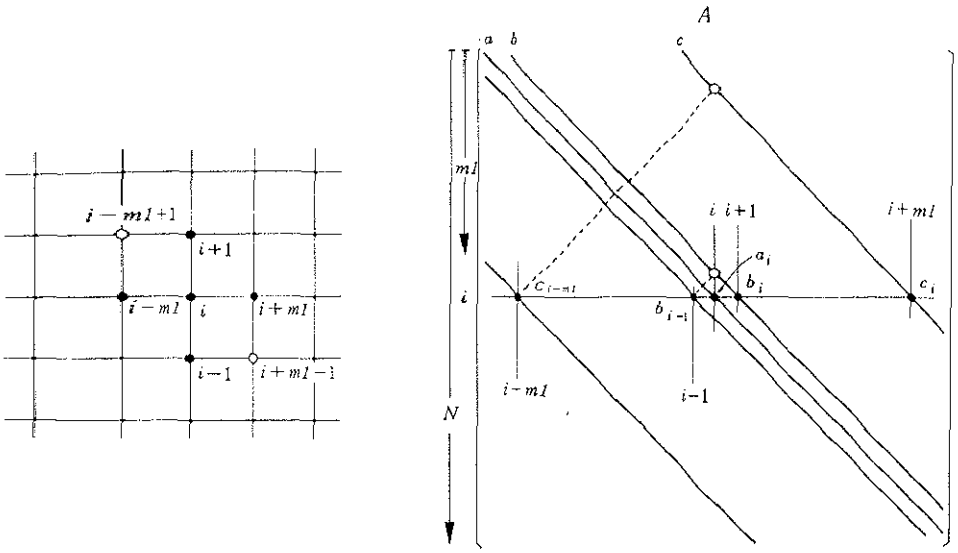


図５．１　２次元の場と離散化行列

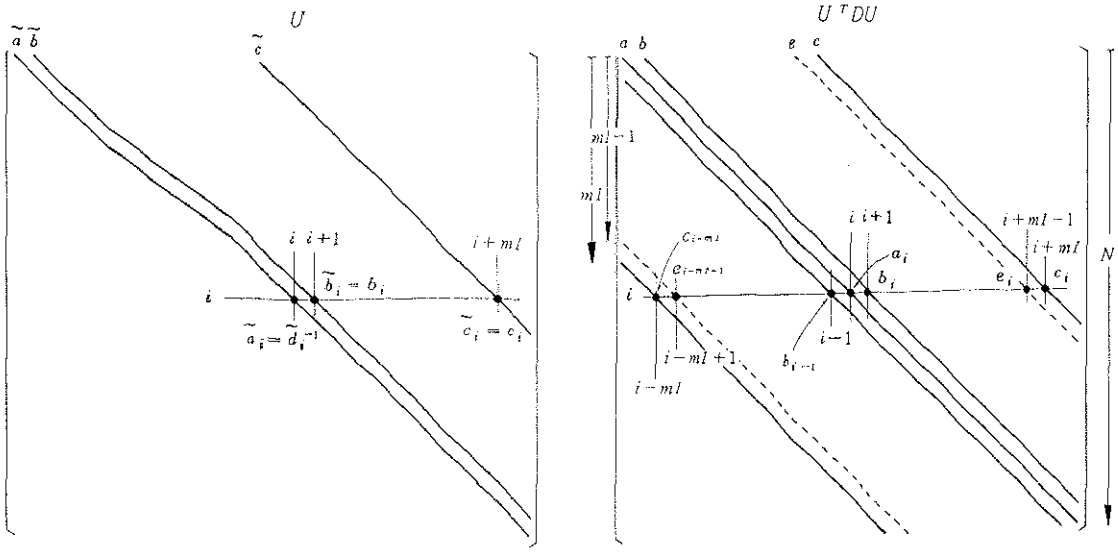


図５．２　ＩＣＣＧ(１，１)のＵとＵ^{T}ＤＵ

(2) ICCG(1,2)

ICCG(1,1)のUとDから作った$U^T DU$でfill-in となる部分までを不完全$U^T DU$分解の対象にするのがICCG(1,2)である。格子集合は

$$G=\{(i,j);a_{i,j}\neq 0, j=i+m1-1\}$$

となる。ICCG(1,1)と同様にUの非ゼロ要素を①$\widetilde{d_i}^{-1}$, ②$\widetilde{b_i}$, ③$\widetilde{c_i}$, ④$\widetilde{e_i}$とすると、以下のようになる。

不完全$U^T DU$分解を行うと、対角要素は

$$\text{①}\quad u_{i,i}=\widetilde{d_i}^{-1}=a_{i,j}-u^T_{i,i-1}\widetilde{d}_{i-1}u_{i-1,i}-u^T_{i,i-m1}\widetilde{d}_{i-m1}u_{i-m1,i}-u^T_{i,i-m1+1}\widetilde{d}_{i-m1+1}u_{i-m1+1,i}$$
$$=a_i-\widetilde{b}_{i-1}{}^2\widetilde{d}_{i-1}-\widetilde{c}_{i-m1}{}^2\widetilde{d}_{i-m1}-\widetilde{e}_{i-m1+1}{}^2\widetilde{d}_{i-m1+1}$$

となる。

非対角要素は

$$a_{i,j}=u^T_{i,i}\widetilde{d_i}u_{i,j}+u^T_{i,i-1}\widetilde{d}_{i-1}u_{i-1,j}+u^T_{i,i-m1}\widetilde{d}_{i-m1}u_{i-m1,j}+u^T_{i,i-m1+1}\widetilde{d}_{i-m1+1}u_{i-m1+1,j}$$

から、$u^T_{i,i}\widetilde{d_i}=1$を代入して

$$u_{i,j}=a_{i,j}-u^T_{i,i-1}\widetilde{d}_{i-1}u_{i-1,j}-u^T_{i,i-m1}\widetilde{d}_{i-m1}u_{i-m1,j}-u^T_{i,i-m1+1}\widetilde{d}_{i-m1+1}u_{i-m1+1,j}$$

として求められる。

$$\text{②}\quad \widetilde{b_i}=u_{i,i+1}=a_{i,i+1}$$
$$-u^T_{i,i-1}\widetilde{d}_{i-1}u_{i-1,i+1}-u^T_{i,i-m1}\widetilde{d}_{i-m1}u_{i-m1,i+1}-u^T_{i,i-m1+1}\widetilde{d}_{i-m1+1}u_{i-m1+1,i+1}$$

$u_{i-1,i+1}=0$、$u_{i-m1,i+1}=0$なので

$$=b_i-\widetilde{e}_{i-m1+1}\widetilde{d}_{i-m1+1}\widetilde{c}_{i-m1+1}.$$

$$\text{③}\quad \widetilde{c_i}=u_{i,i+m1}=a_{i,i+m1}$$
$$-u^T_{i,i-1}\widetilde{d}_{i-1}u_{i-1,i+m1}-u^T_{i,i-m1}\widetilde{d}_{i-m1}u_{i-m1,i+m1}-u^T_{i,i-m1+1}\widetilde{d}_{i-m1+1}u_{i-m1+1,i+m1}$$

$u_{i-1,i+m1}=0$、$u_{i-m1,i+m1}=0$、$u_{i-m1+1,i+m1}=0$なので

$$=a_{i,i+m1}=c_i.$$

$$\text{④}\quad \widetilde{e_i}=u_{i,i+m1-1}=a_{i,i+m1-1}-u^T_{i,i-1}\widetilde{d}_{i-1}u_{i-1,i+m1-1}$$
$$-u^T_{i,i-m1}\widetilde{d}_{i-m1}u_{i-m1,i+m1-1}-u^T_{i,i-m1+1}\widetilde{d}_{i-m1+1}u_{i-m1+1,i+m1-1}$$
$$=0-\widetilde{b}_{i-1}\widetilde{d}_{i-1}\widetilde{c}_{i-1}-0-0.$$

これから、ICCG(1,2)用の不完全$U^T DU$分解アルゴリズムは以下のようになる。

```
do i = 1,  n
  d̃_i^{-1} = a_i - b̃_{i-1}^2 d̃_{i-1} - c̃_{i-m1}^2 d̃_{i-m1} - ẽ_{i-m1+1}^2 d̃_{i-m1+1}
  b̃_i = b_i - c̃_{i-m1+1} d̃_{i-m1+1} ẽ_{i-m1+1}
  ẽ_i = - b̃_{i-1} d̃_{i-1} c̃_{i-1}
  c̃_i = c_i
```

ICCG(1,2)の場合も、収束特性の解析用に$U^T DU$を求めておく。

$$(U^T DU)_{i,j} = \Sigma u^T_{i,k} (DU)_{k,j} \tag{5.7}$$

kのとりうる値はi,i-1,i-m1, i-m1+1の4通りだから(5.7)式は

$$(U^T DU)_{i,j} = u^T_{i,i} \tilde{d}_i u_{i,j} + u^T_{i,i-1} \tilde{d}_{i-1} u_{i-1,j} + u^T_{i,i-m1} \tilde{d}_{i-m1} u_{i-m1,j} + u^T_{i,i-m1+1} \tilde{d}_{i-m1+1} u_{i-m1+1,j} \tag{5.8}$$

となる。(5.8)式から、各iについてjのとりうる値は①i、②i+1、③i+m1、④i+m1-1, ⑤i+m1-2の5通りである。ここではi+m1-2列がfill-inとなる。

結果だけを書くと

$$(U^T DU)_{i,i} = a_i ; \quad (U^T DU)_{i,i+1} = b_i$$
$$(U^T DU)_{i,i+m1-1} = 0; \quad (U^T DU)_{i,i+m1} = c_i$$
$$(U^T DU)_{i,i+m1-2} = \tilde{b}_{i-1} \tilde{d}_{i-1} \tilde{e}_{i-1} .$$

i+m1-2列のfill-inを$\tilde{f}_i$として、Uと$U^T DU$の関係を図5.3に示す。

(3) ICCG(1,3)

ICCG(1,3)では、ICCG(1,1)とICCG(1,2)でfill-inとなった部分$\tilde{e}_i$、$\tilde{f}_i$までを不完全$U^T DU$分解の対象にする。格子集合は

$$G = \{(i,j); a_{i,j} \neq 0 , j = i+m1-1, i+m1-2\}$$

となる。これまでと同様に不完全$U^T DU$分解を行うと、アルゴリズムは

do i = 1, n

$$\tilde{d}_i^{-1} = a_i - \tilde{b}_{i-1}^2 \tilde{d}_{i-1} - \tilde{c}_{i-m1}^2 \tilde{d}_{i-m1} - \tilde{e}_{i-m1+1}^2 \tilde{d}_{i-m1+1} - \tilde{f}_{i-m1+2}^2 \tilde{d}_{i-m1+2}$$
$$\tilde{b}_i = b_i - \tilde{e}_{i-m1+1} \tilde{d}_{i-m1+1} \tilde{c}_{i-m1+1} - \tilde{f}_{i-m1+2} \tilde{d}_{i-m1+2} \tilde{e}_{i-m1+2}$$
$$\tilde{e}_i = -\tilde{b}_{i-1} \tilde{d}_{i-1} \tilde{c}_{i-1}$$
$$\tilde{f}_i = -\tilde{b}_{i-1} \tilde{d}_{i-1} \tilde{e}_{i-1}$$
$$\tilde{c}_i = c_i$$

となる。

ICCG(1,3)の場合の$U^T DU$は

$$(U^T DU)_{i,i} = a_i ; \quad (U^T DU)_{i,i+1} = b_i$$
$$(U^T DU)_{i,i+m1-2} = 0; \quad (U^T DU)_{i,i+m1-1} = 0$$
$$(U^T DU)_{i,i+m1} = c_i ; \quad (U^T DU)_{i,i+2} = \tilde{f}_{i-m1+2} \tilde{d}_{i-m1+2} \tilde{c}_{i-m1+2}$$
$$(U^T DU)_{i,i+m1-3} = \tilde{b}_{i-1} \tilde{d}_{i-1} \tilde{f}_{i-1}$$

となる。i+2列のfill-inを$\tilde{h}_i$、i+m1-3列のfill-inを$\tilde{g}_i$として、Uと$U^T DU$の関係を図5.4に示す。

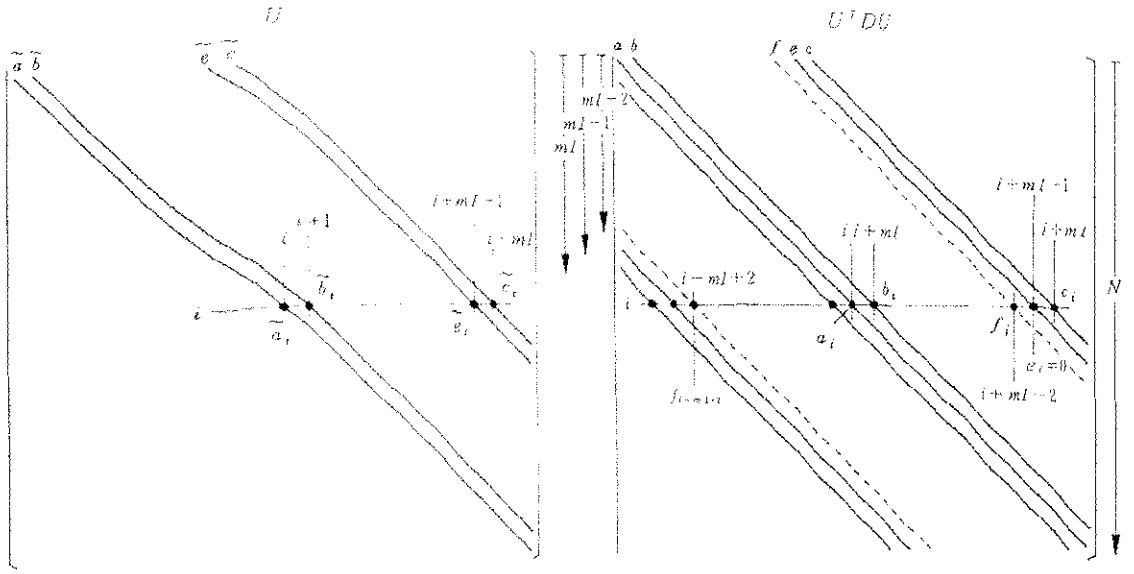


図 5.3 ICCG(1,2)のUとU^{T}DU

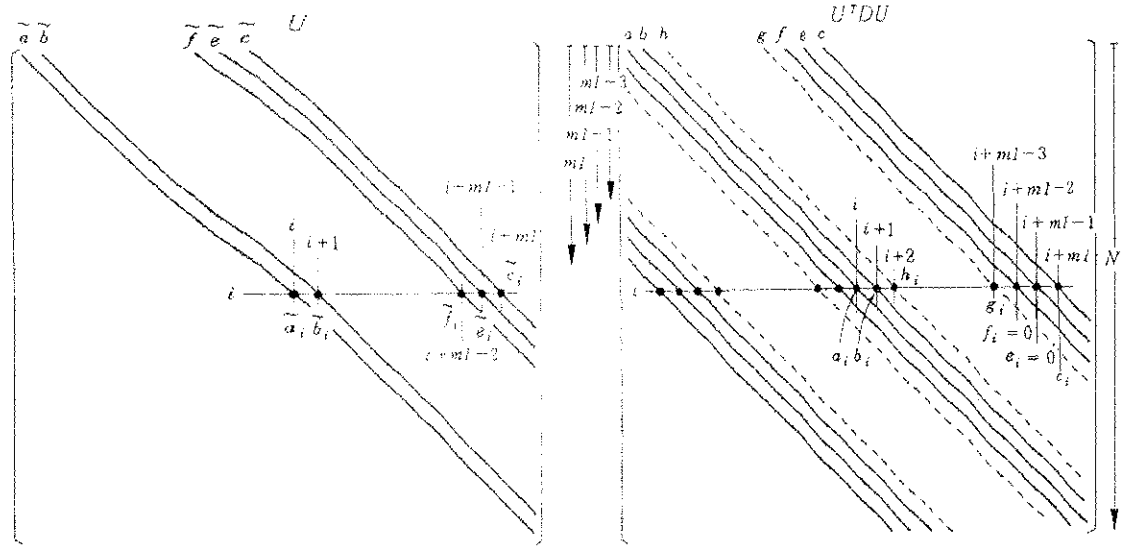


図 5.4 ICCG(1,3)のUとU^{T}DU

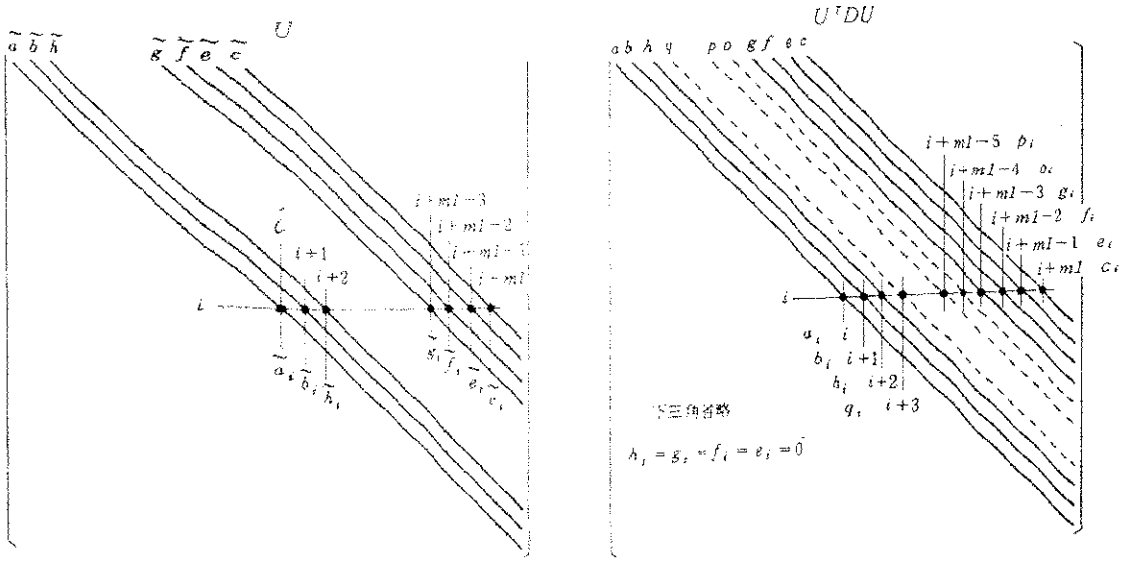


図 5.5 ICCG(2,4)のUとU^{T}DU

(4) ICCG(2,4)

ICCG(2,4)では、ICCG(1,3)までに現れたすべての部分を不完全$U^T DU$分解の対象にする。格子集合は

$$G=\{(i,j);a_{i,j}\neq 0, j=i+m1-1,i+m1-2,i+m1-3,i+2\}$$

となる。不完全$U^T DU$分解のアルゴリズムは

do i = 1, n

$$\widetilde{d}_i^{-1}=a_i-\widetilde{b}_{i-1}^2\widetilde{d}_{i-1}-\widetilde{c}_{i-m1}^2\widetilde{d}_{i-m1}-\widetilde{e}_{i-m1+1}^2\widetilde{d}_{i-m1+1}-\widetilde{f}_{i-m1+2}^2\widetilde{d}_{i-m1+2}$$
$$-\widetilde{h}_{i-2}^2\widetilde{d}_{i-2}-\widetilde{f}_{i-m1+2}^2\widetilde{d}_{j-m1+2}-\widetilde{g}_{i-m1+3}^2\widetilde{d}_{i-m1+3}$$

$$\widetilde{b}_i=b_i-\widetilde{b}_{i-1}\widetilde{d}_{i-1}\widetilde{h}_{i-1}-\widetilde{e}_{i-m1+1}\widetilde{d}_{i-m1+1}\widetilde{c}_{i-m1+1}-\widetilde{f}_{i-m1+2}\widetilde{d}_{i-m1+2}\widetilde{e}_{i-m1+2}$$
$$-\widetilde{g}_{i-m1+3}\widetilde{d}_{i-m1+3}\widetilde{h}_{i-m1+3}$$

$$\widetilde{h}_i=-\widetilde{f}_{i-m1+2}\widetilde{d}_{i-m1+2}\widetilde{c}_{i-m1+2}-\widetilde{g}_{i-m1+3}\widetilde{d}_{i-m1+3}\widetilde{e}_{i-m1+3}$$

$$\widetilde{g}_i=-\widetilde{b}_{i-1}\widetilde{d}_{i-1}\widetilde{f}_{i-1}-\widetilde{h}_{i-2}\widetilde{d}_{i-2}\widetilde{e}_{i-2}$$

$$\widetilde{f}_i=-\widetilde{b}_{i-1}\widetilde{d}_{i-1}\widetilde{e}_{i-1}-\widetilde{h}_{i-2}\widetilde{d}_{i-2}\widetilde{c}_{i-2}$$

$$\widetilde{e}_i=-\widetilde{b}_{i-1}\widetilde{d}_{i-1}\widetilde{c}_{i-1}$$

$$\widetilde{c}_i=c_i$$

となる。

ICCG(2,4)の場合の$U^T DU$は

$$(U^T DU)_{i,i}=a_i;\quad (U^T DU)_{i,i+1}=b_i$$
$$(U^T DU)_{i,i+2}=0;\quad (U^T DU)_{i,i+m1-3}=0$$
$$(U^T DU)_{i,i+m1-2}=0;\quad (U^T DU)_{i,i+m1-1}=0$$
$$(U^T DU)_{i,i+m1}=c_i;\quad (U^T DU)_{i,i+3}=\widetilde{g}_{i-m1+3}\widetilde{d}_{i-m1+3}\widetilde{c}_{i-m1+3}$$
$$(U^T DU)_{i,i+m1-5}=\widetilde{h}_{i-2}\widetilde{d}_{i-2}\widetilde{g}_{i-2}+\widetilde{g}_{i-m1+3}\widetilde{d}_{i-m1+3}\widetilde{c}_{i-m1+3}$$
$$(U^T DU)_{i,i+m1-4}=\widetilde{b}_{i-1}\widetilde{d}_{i-1}\widetilde{g}_{i-1}+\widetilde{h}_{i-2}\widetilde{d}_{i-2}\widetilde{f}_{i-2}$$

となる。i+3 列のfill-in をq_i、i+m1-5列のfill-in を$\widetilde{p}_i$、i+m1-4列のfill-in を$\widetilde{o}_i$として、Uと$U^T DU$の関係を図5.5に示す。

(5) MICCG法

MICCG法は修正コレスキー共役勾配法(Modified Incomplete Cholesky Conjugate Gradient Method)の略称で、1978年にGustafssonによって発表された[18]。Gustafssonの改良は、点iの近傍の点jではxjとxiに差がないと仮定して、Meijerinkの不完全$U^T DU$分解から得られた$U^T DU$のfill-in部分のウエイトを対角要素に負わせ、不完全$U^T DU$分解時の不完全さをなるべく少なくしようということである[51]。

例としてICCG(1,1)をとりあげる。Meijerinkの不完全$U^T DU$分解から得られた$U^T DU$でfill-inとなるのは

i-m1＋1列とi＋m1-1列

であり、未知数x_{i-m1+1}とx_{i+m1-1}の係数に対応する。点i-m1＋1と点i＋m1-1の位置を図5.1に示す。

そこで、x_{i-m1+1}、x_{i+m1-1}とx_iには差がないとして、fill-inの値の符号を変えてx_iの係数に付加するように$\tilde{d}_i^{-1}$を変更する。これによって不完全$U^T DU$分解の不完全性が少しは改善される。ところが、fill-inに対応した量をそのまま対角要素から抜き去ると$\tilde{d}_i^{-1}$が負やゼロになって$U^T DU$が対称正定値行列かつM-行列という条件を満足しなくなる恐れがある。そこで対称正定値性、M-行列性を崩さないように対角要素に補正する量を加減する必要がある。したがってGustafssonのアイデアは

$$\tilde{d}_i^{-1}=(1+\varepsilon)a_i-\tilde{b}_{i-1}^2\tilde{d}_{i-1}-\tilde{c}_{i-m1}^2\tilde{d}_{i-m1}-\{\tilde{b}_{i-1}\tilde{d}_{i-1}\tilde{c}_{i-1}+\tilde{b}_{i-m1}\tilde{d}_{i-m1}\tilde{c}_{i-m1}\}$$

と書ける[18]。Gustafssonの変更には色々な変種があるが[51],[56],[68],]70]、ここでは文献[51],[68]にあるような、対角項a_iに$(1+\varepsilon)$をかける代わりに、fill-inに対応した量に修正パラメータuをかけてから減ずる算法

$$\tilde{d}_i^{-1}=a_i-\tilde{b}_{i-1}^2\tilde{d}_{i-1}-\tilde{c}_{i-m1}^2\tilde{d}_{i-m1}-\{\tilde{b}_{i-1}\tilde{d}_{i-1}\tilde{c}_{i-1}+\tilde{b}_{i-m1}\tilde{d}_{i-m1}\tilde{c}_{i-m1}\}u.$$

を採用する。これはICCG(1,1)に対する修正なので、MICCG(1,1)とよぶ。fill-inに対応した量を修正するので、Gustafsson流ということになる。Gustafsson流の変更は不完全$U^T DU$分解の対角行列Dを少し変えるだけで、反復の過程はMeijerinkの不完全$U^T DU$分解による前処理と同じであり、必要なメモリ容量も元のICCG法と同じである。これで収束が速くなるのならば申し分ない。他のICCG法に対する修正は次のようになる。

①MICCG(1,2)では

$$\tilde{d}_i^{-1}=a_i-\tilde{b}_{i-1}^2\tilde{d}_{i-1}-\tilde{c}_{i-m1}^2\tilde{d}_{i-m1}-\tilde{e}_{i-m1+1}^2\tilde{d}_{i-m1+1}$$
$$-\{\tilde{b}_{i-1}\tilde{d}_{i-1}\tilde{e}_{i-1}+\tilde{b}_{i-m1+1}\tilde{d}_{i-m1+1}\tilde{e}_{i-m1+1}\}u.$$

②MICCG(1,3)では

$$\tilde{d}_i^{-1}=a_i-\tilde{b}_{i-1}^2\tilde{d}_{i-1}-\tilde{c}_{i-m1}^2\tilde{d}_{i-m1}-\tilde{e}_{i-m1+1}^2\tilde{d}_{i-m1+1}-\tilde{f}_{i-m1+2}^2\tilde{d}_{i-m1+2}$$
$$-\{\tilde{c}_{i-m1+2}\tilde{d}_{i-m1+2}\tilde{f}_{i-m1+2}+\tilde{c}_{i-m1}\tilde{d}_{i-m1}\tilde{f}_{i-m1}$$
$$+\tilde{b}_{i-1}\tilde{d}_{i-1}\tilde{f}_{i-1}+\tilde{b}_{i-m1+2}\tilde{d}_{i-m1+2}\tilde{f}_{i-m1+2}\}u.$$

③MICCG(2,4)では

$$\widetilde{d}_i^{-1} = a_i - \widetilde{b}_{i-1}{}^2\widetilde{d}_{i-1} - \widetilde{c}_{i-m1}{}^2\widetilde{d}_{i-m1} - \widetilde{e}_{i-m1+1}{}^2\widetilde{d}_{i-m1+1} - \widetilde{f}_{i-m1+2}{}^2\widetilde{d}_{i-m1+2}$$
$$- \widetilde{h}_{i-2}{}^2\widetilde{d}_{i-2} - \widetilde{f}_{i-m1+2}{}^2\widetilde{d}_{i-m1+2} - \widetilde{g}_{i-m1+3}{}^2\widetilde{d}_{i-m1+3}$$
$$- \{2\widetilde{g}_{i-m1+3}\widetilde{d}_{i-m1+3}\widetilde{c}_{i-m1+3} + 2\widetilde{g}_{i-m1}\widetilde{d}_{i-m1}\widetilde{c}_{i-m1} + \widetilde{b}_{i-1}\widetilde{d}_{i-1}\widetilde{g}_{i-1} + \widetilde{h}_{i-2}\widetilde{d}_{i-2}\widetilde{f}_{i-2}$$
$$+ \widetilde{h}_{i-2}\widetilde{d}_{i-2}\widetilde{g}_{i-2} + \widetilde{b}_{i-m1+3}\widetilde{d}_{i-m1+3}\widetilde{g}_{i-m1+3} + \widetilde{f}_{i-m1+2}\widetilde{d}_{i-m1+2}\widetilde{h}_{i-m1+2}$$
$$+ \widetilde{g}_{i-m1+3}\widetilde{d}_{i-m1+3}\widetilde{h}_{i-m1+3}\}u.$$

Gustafsson 流の変更におけるuの値は0.8 ~0.975 ならば性能に大差がないとされているが[51]、一般的にはu＝0.95程度が用いられている[71]。

5.2 PCG法の収束特性解析

PCG法は反復解法であり、前処理によっても収束特性が異なる。しかし前処理と収束特性の関係は一部しか明らかになっていない[25]。4章では$Ax=b$に対するCG法の収束特性を明らかにしたが、PCG法は$\widetilde{U}^{-T}A\widetilde{U}^{-1}y=\widetilde{U}^{-T}b, y=\widetilde{U}x$に対するCG法ということから、4章の結果と対応づけて、主としてICGG法とMICCG法の収束特性の検討を行う。

(1) $\widetilde{U}^{-T}A\widetilde{U}^{-1}$の固有値分布

まずはじめに、前処理が解こうとしている問題とどのような関係にあるかを調べておきたい。ここで注目するのは

- ·場に対する依存性
- ·問題の大きさに対する依存性
- ·右辺bまたは解xに対する依存性
- ·前処理の効果
- ·MICCG法におけるuの最適値

である。場の物理定数やメッシュの大きさ、解や右辺を変えると、CG法で解こうとしている方程式$Ax=b$の性質が変化する。このとき、どのような前処理が有効なのか、それぞれの前処理の効果や特徴はどうなっているかなどを実験から明らかしたい。

そのためには、テストデータを変えて問題を繰り返し解くだけではなく、PCG法の反復作用素$\widetilde{U}^{-T}A\widetilde{U}^{-1}$に対する解析が必要である。PCG法の収束特性を正攻法で調べるためには、まずPCG法の反復作用素$\widetilde{U}^{-T}A\widetilde{U}^{-1}$の固有値分布を調べる必要があろう。必要なのは固有値だけなので、固有ベクトルは計算しないでも、固有値が精度よく求められればよい。実際に$\widetilde{U}^{-T}A\widetilde{U}^{-1}$を作ると$\widetilde{U}^{-T}A\widetilde{U}^{-1}$は密行列になるため、PCG法が対象にするような大規模な問題では解析が不可能になる恐れがある。

そこで、標準固有値問題

$$\widetilde{U}^{T}A\widetilde{U}^{-1}\widetilde{v}=\mu\widetilde{v}$$

を、$w=\widetilde{U}^{-1}\widetilde{v}$とおいて左から$\widetilde{U}^{T}$を作用させ

$$Aw=\mu\widetilde{U}^{T}\widetilde{U}w=\mu Mw,\ M=\widetilde{U}^{T}\widetilde{U}$$

という一般固有値問題に帰着して解析を行う[15]。ここでA、$M=\widetilde{U}^{T}\widetilde{U}$は対称正定値疎行列である。実際は$M$として不完全$U^{T}DU$分解から作られた$U^{T}DU$を用いる。一般固有値問題に対しては二分法･同時逆反復法のプログラムEIGV43[52]を利用する。ところが、標準固有値問題を一般固有値問題に拡張すると新たな問題点が生じる。

Parlett[61]によれば、行列A、Mが共にゼロ固有値をもつとき固有ベクトルが不定となることがあり、行列A、Mが共にゼロに近い固有値をもち、対応する固有空間が近いとA、Mの摂動に対して固有値のはなはだしい変動をひきおこすことがある。その結果、∞の固有値や、複素固有値をもつこともありうる。これらはMを対称正定値行列に限定すると理論的には消滅するが、行列が半正定値に近くなれば数値計算的には問題となる。

第2点としてA、Mが近いことから密集固有値が多くなるという問題が生じる。特にPCG法ではAと$M=\widetilde{U}^{T}\widetilde{U}$が近いほど収束が良くなり、対応する反復作用素$\widetilde{U}^{T}A\widetilde{U}^{-1}$には重複、密集固有値が多くなって、一般固有値問題としてはより難しい問題となる。多くの密集固有値と互いに直交する固有ベクトルを精度よく求めるには、行列の摂動に対して頑健なプログラムを用いる必要がある。

(2) 固有ベクトルによる解の構成

PCG法の場合も、解が固有ベクトルのどのような一次結合として表されるかという観点からの解析が必要である。そのためには、一般固有値問題$Aw_i=\mu_iMw_i$, $M=U^{T}DU$の全固有値μ_i、全固有ベクトルw_iが必要である。

対称正定値行列Aを係数とする連立一次方程式$Ax=b$の解xがAの固有ベクトルv_iを用いて

$$x=\Sigma c_iv_i$$

のように展開されるとする。ただし、固有ベクトルv_iは$Av_i=\lambda_iv_i$ $(\lambda_1\leqq\lambda_2\leqq\ldots\leqq\lambda_n)$を満足する。

この関係は、PCG法において前処理された方程式

$$(\widetilde{U}^{T}A\widetilde{U}^{-1})y=\widetilde{U}^{T}b,\ x=\widetilde{U}^{-1}y,\ M=\widetilde{U}^{T}\widetilde{U}$$

では

$$y=\Sigma c_i\widetilde{v}_i$$

となる。ただし、$\widetilde{v}_i$は標準固有値問題$(\widetilde{U}^{T}A\widetilde{U}^{-1})\widetilde{v}_i=\mu_i\widetilde{v}_i$ $(\mu_1\leqq\mu_2\leqq\ldots\leqq\mu_n)$から求められる。

この式に左側から$\tilde{U}^T$を作用させ、$\tilde{v}_i=\tilde{U}w_i$とすると

$$\tilde{U}^T(\tilde{U}^{-T}A\tilde{U}^{-1})\tilde{U}w_i=\mu i\tilde{U}^T\tilde{U}w_i$$

となって、μiとw_iは

$$Aw_i=\mu_iMw_i,\quad M=\tilde{U}^T\tilde{U}\quad(\mu_1\leqq\mu_2\leqq\ldots\leqq\mu_n)$$

の形の一般固有値問題を満足する。実際のMは不完全U^TDU分解から作られるU^TDUを使用する。したがって、PCG法の解ベクトルは

$$\begin{aligned}y&=\Sigma\tilde{c}_i\tilde{v}_i\\&=\Sigma\tilde{c}_i\tilde{U}w_i,\end{aligned}$$

$y=\tilde{U}x$より

$$\begin{aligned}x&=\tilde{U}^{-1}(\Sigma\tilde{c}_i\tilde{U}w_i)\\&=\Sigma\tilde{c}_iw_i\end{aligned}$$

と書ける。

これから、CG法に対して連立一次方程式$Ax=b$の解xを行列Aの固有値λ_i、固有ベクトルv_iによって展開したことは、PCG法に対しては解xを一般固有値問題$Aw_i=\mu_i(U^TDU)w_i$の固有値μ_i、固有ベクトルw_iによって展開することに対応する。また、固有値と固有ベクトルが変わることによって展開の係数c_iと$\tilde{c}_i$も変わる。

共役勾配法(CG法)では、$Av_i=\lambda_iv_i$となる1本のv_iを解とするような問題は理論上1回の反復で収束するし、実際、精度さえ確保すれば1回の反復で収束した。倍精度計算では最大固有値が約10^1である問題を解こうとすると、$\lambda_i\leqq10^{-3}$となる固有値に対応した固有ベクトル成分は収束しにくかった。

同様に前処理付き共役勾配法(PCG法)では、$(\tilde{U}^{-T}A\tilde{U}^{-1})\tilde{v}_i=\mu_i\tilde{v}_i$となる1本の$\tilde{v}_i=\tilde{U}w_i$を解$y=\tilde{U}w_i$とする問題、すなわち$Aw_i=\mu_iMw_i$, $M=U^TDU$となるw_iを解xとする問題は理論上1回の反復で収束するはずである。

CG法での解析と同様に、解を固有ベクトルの組合わせによって構成するため、次のような5とおりの解を用意する。固有ベクトルの組はM–正規かつM–直交で、固有値は小さい順に並べられている($\mu_1\leqq\mu_2\leqq\ldots\leqq\mu_n$)とする。

(a) w_k
(b) Σw_i (i=k-1,k)
(c) Σw_i (i=k-2,k)
(d) Σw_i (i=1,k)
(e) Σw_i (i=k,n)

(a)は理論との比較ならびに固有値の違いによる変化をみるため、(b)と(c)は重複ならび密集の効果をみるための組合せである。(d)は低次モード(小さな固有値に対応)の影響をみるための組合せで、(e)は高次モード(大きな固有値に対応)の影響をみるための組合せである。

(3) 残差多項式$R_k(A)$

残差多項式$R_k(A)$を用いた解析[43],[64]のためには、すべての固有値･固有ベクトルを精度よく求める必要がある。固有値･固有ベクトルを使用せずに残差多項式を求める方法も考えられるが[43]、計算の手間からみて比較的小規模な行列(100元程度?)で適用が困難になる。さて、CG法のアルゴリズムで$p_0=r_0$とおいて反復を始めると

$$\begin{aligned} r_1 &= r_0 - \alpha_0 A p_0 \\ &= (I - \alpha_0 A) r_0 \\ p_1 &= r_1 + \beta_0 p_0 \\ &= (I - \alpha_0 A) r_0 + \beta_0 r_0 \\ &= ((1+\beta_0) I + \alpha_0 A) r_0 . \end{aligned}$$

$R_0(A)=I$、$R_1(A)=I-\alpha_0 A$、$P_1(A)=(1+\beta_0)I+\alpha_0 A$とすると、

$$\begin{aligned} r_{k+1} &= r_k - \alpha_k A p_k \\ &= (R_k(A) - \alpha_k A P_k(A)) r_0 \\ &= R_{k+1}(A) r_0 . \end{aligned}$$

同様に

$$\begin{aligned} p_{k+1} &= r_k + \beta_k p_k \\ &= (R_k(A) + \beta_k P_k(A)) r_0 \\ &= P_{k+1}(A) r_0 \end{aligned}$$

と書ける。したがって、CG法では第k回の反復で

$$(A^{-1} r_k , r_k) = (A^{-1} R_k(A) r_0 , R_k(A) r_0)$$

を最小化することになる。

これから、CG法の残差ベクトルr_kは初期残差ベクトルr_0とAの多項式$R_k(A)$から構成されていることがわかる。横軸に固有値、縦軸にそのときの残差多項式の値をプロットすれば、反復回数kに関してk次の多項式が得られる。固有ベクトルv_iが正規直交であることを利用して、残差ベクトルr_kと固有ベクトルv_iと内積をとれば、k次の多項式$R_k(A)$が未知でも各点λ_iにおける多項式の値が求められる。厳密には、固有ベクトルの向きには正負があるので、λ_iにおける多項式の値の絶対値$|R_k(\lambda_i)|$が求められる。

PCG法の場合は、反復作用素$\tilde{U}^{-T} A \tilde{U}^{-1}$に対して同様の議論を行えばよい。一般固有値問題$Aw_i=\mu_i M w_i$, $M=U^T D U$($\mu_1 \leqq \mu_2 \leqq \ldots \leqq \mu_n$)から得られる固有値$\mu_i$、固有ベクトル$w_i$を用いれば、PCG法の場合も残差ベクトル$r_k$と固有ベクトル$w_i$から、$\mu_i$における多項式の値の絶対値$|R_k(\mu_i)|$が求められる。それには、CG法では内積をとるだけでよかったものが、PCG法では$R_k(\mu_i)=(r_k , U^T D U w_i)$、固有ベクトル$w_i$はM-正規、M-直交でなければならない[19],[46]。

いずれの場合も、初期残差ベクトル$r_0=\Sigma \tilde{c}_i \tilde{v}_i$(または$\Sigma \tilde{c}_i w_i$)を意図的に構成し、反復過程で各成分がどのように変化するかをみることができる。

5.3 前処理と収束の関係(数値実験)

テスト問題は、4章と同じ図4.1の場における拡散方程式を差分法で離散化して得られた行列を用いた。分割をm1としたとき非対角帯半幅m1、元数n＝m1(2m1＋3)の対称正定値疎行列になる。□部の拡散係数k(x,y)を1,▨部の拡散係数k(x,y)をDF0、DF1、DF2、DF3とする。拡散係数を1、10^{-3}、10^{-6}のように変化させ、拡散係数の値に段差をもたせる。

収束判定は、反復の間に更新されるr_kを用いた相対残差ノルム$\|r_k\|/\|b\|$で行った。反復が終了した後、収束とみなされた解x_kから改めて相対残差ノルム$\|Ax_k-b\|/\|b\|$を計算しなおす。収束判定値$\varepsilon=0.22\times10^{-10}$とし、すべて倍精度で計算を行った。

CG法、4種類のICCG法とu＝0.95とした4種類のMICCG法で連立一次方程式$Ax=b$を解いて収束回数を調べる。右辺ベクトルbは、$(1,1,\ldots,1)^T$、$(-1,1,\ldots,(-1)^n)^T$と(0,1)の一様乱数の3通りの解xを与えて、$b=Ax$から作成した。反復の初期ベクトル$x=o$とした。

m1＝16、n＝560の結果を表5.1に示す。拡散係数DF(＝DF1＝DF2＝DF3)が1から10^{-6}と小さくなるにつれて収束に要する反復回数が増え、前処理を複雑にすると収束に要する反復回数が減少する。解ベクトルxが$(1,1,\ldots,1)^T$のときと一様乱数のときに収束が悪く、特にCG法は一様乱数のときに収束が悪い。原因は、一様乱数の場合はすべての固有ベクトル成分が含まれるためと考えられる。いずれの場合にも前処理は有効であり、場に対する依存性を和らげる効果もあることがわかる。

DF(＝DF1＝DF2＝DF3)＝10^{-6}で$x=(1,1,\ldots,1)^T$のときには、解から改めて計算した相対残差ノルムが反復中の相対残差ノルムより2桁大きかった。収束判定をきつくすればすむ問題でもあるが、PCG法の場合も反復過程では誤差に注意が必要なことを意味している。

DF(＝DF1＝DF2＝DF3)＝10^{-3}　でm1　を24(n＝1224)，32(2144)，64(8384)，128(33152)と変えたときの収束に要する反復回数を表5.2に示す。収束特性に大きな変化はないが、nが大きくなると収束に要する反復回数が増える。m1が大きくなると、MICCG(1,2)がICCG(1,3)よりも少ない反復回数で収束するようになる。これはMICCG法が近傍の値が等しいとした補正なので、同一の場を細かいメッシュで近似したことが原因といえよう。解xが$(-1,1,\ldots,(-1)^n)^T$のときにMICCG法の効果が小さいのも同様なことが原因といえよう。このようにMICCG法はnが大きくなるとより有効となり、結果的に前処理を複雑にしたICCG法よりも早く収束するようにもなる。

表５．１　ＣＧ法とＰＣＧ法の反復回数

$m1=16$, $N=560$, $u=0.95$, $\varepsilon=0.22\times10^{-10}$

DF	x	CG	ICCG(1,1)	MICCG(1,1)	ICCG(1,2)	MICCG(1,2)	ICCG(1,3)	MICCG(1,3)	ICCG(2,4)	MICCG(2,4)
1	$(1,1,\cdots,1)^T$	77	41	29	27	22	21	19	16	98
	$(-1,1,\cdots,(-1)^n)^T$	63	35	27	24	19	20	17	14	96
	一　様　乱　数	137	41	29	26	22	21	19	16	98
10^{-1}	$(1,1,\cdots,1)^T$	109	47	35	31	27	26	23	20	202
	$(-1,1,\cdots,(-1)^n)^T$	85	42	32	27	23	23	21	17	172
	一　様　乱　数	174	46	34	30	26	25	22	19	171
10^{-3}	$(1,1,\cdots,1)^T$	302	54	44	36	35	30	30	23	170
	$(-1,1,\cdots,(-1)^n)^T$	167	35	42	27	30	27	27	21	176
	一　様　乱　数	439	54	43	35	32	29	28	22	169
10^{-6}	$(1,1,\cdots,1)^T$*	579	83	70	55	55	46	48	36	149
	$(-1,1,\cdots,(-1)^n)^T$	266	26	25	27	29	22	26	18	108
	一　様　乱　数	677	63	53	41	42	35	36	27	114

DF$=10^{-6}$, $x=(1,\cdots,1)^T$ は $\|Ax_k-b\|/\|b\|\cong10^{-8}$

表５．２　ＣＧ法とＰＣＧ法の反復回数

DF $=10^{-3}$, $\omega=0.95$

	x	CG	ICCG(1,1)	MICCG(1,1)	ICCG(1,2)	MICCG(1,2)	ICCG(1,3)	MICCG(1,3)	ICCG(2,4)	MICCG(2,4)
$m1=24$ $N=1224$	$(1,1,\cdots,1)^T$	443	77	55	50	40	41	35	32	30*
	$(-1,1,\cdots,(-1)^n)^T$	263	49	52	36	35	37	32	22	27*
	一様乱数	591	76	53	49	38	40	33	31	28*
$m1=32$ $N=2144$	$(1,1,\cdots,1)^T$	547	100	64	65	48	53	39	40	33*
	$(-1,1,\cdots,(-1)^n)^T$	293	60	47	46	33	46	36	28	30*
	一様乱数	704	97	61	63	44	52	37	38	31*
$m1=64$ $N=8384$	$(1,1,\cdots,1)^T$	800	187	106	122	72	99	60	73	50*
	$(-1,1,\cdots,(-1)^n)^T$	439	106	71	78	62	79	53	49	44*
	一様乱数	946	180	101	115	67	94	57	68	47*
$m1=128$ $N=33152$	$(1,1,\cdots,1)^T$	1071	353	190	229	125	186	100	138	82*
	$(-1,1,\cdots,(-1)^n)^T$	606	127	118	138	76	113	80	81	58*
	一様乱数	1380	330	176	204	114	169	93	124	76*

* $\omega=0.65$

表5.1においてu＝0.95としたMICCG(2,4)は拡散係数や解xによらず収束に要する反復回数が多い。MICCG(2,4)においてuを変化させたときの$\widetilde{d}_i^{-1}$の最小値と収束に要する反復回数の関係から、$\widetilde{d}_i^{-1}$に負のものがあると急激に反復回数が増加し、逆にuの値が最適値から外れていても$\widetilde{d}_i^{-1}$が正である限り反復回数は大きく増加しないことが知られている[25]。

一般に使われているMICCG(1,1)ではu＝0.95でよくても、不完全U^TDU分解を複雑にするとuの調整が必要になる。文献[68],[56],[71]ではuは0.8～0.975でよいとしているが、それは3次元問題で$G=\{(i,j);\ a_{i,j}\neq 0\}$とした場合のことであり、MICCG(2,4)の場合を含めて一般的には通用しない。MICCG(2,4)での結果からすると、単純に$\widetilde{d}_i^{-1}$が正になるようにuを決めればよかった。しかし、事前にパラメータが適切かどうかを調べる方法はわかっていないので、$\widetilde{d}_i^{-1}$が正になるかどうかは実際に前処理行列を作ってから確かめる必要がある。

結果として、MICCG法の前処理を安定にするには、与えられたuを用いて$\widetilde{d}_i^{-1}$を計算し、すべての$\widetilde{d}_i^{-1}$がε以上になった時点で反復を開始するようにすればよい。$\min(\widetilde{d}_i^{-1})$が判定値ε以下になった場合はuを0.05程度小さくして$\widetilde{d}_i^{-1}$を計算し直す。この手続きを続けてu≦0になった場合は、u＝0としてICCG法にする。ICCG法の場合はAが対称正定値ならば$\widetilde{d}_i^{-1}>0$が保証されている[41]。

このような方法でuを決めると、確実にMICCG法がICCG法よりも高速になる。実際、u＝0.65とすればMICCG(2,4)もICCG(2,4)より速く収束する。このようにすると前処理にかかる演算量が増えるが、それは反復に要する演算量と比べればわずかで、前処理が安定になって反復回数が減少する効果のほうが大きい。

A、4種類のICCG法とu＝0.95とした4種類のMICCG法の前処理に対応したU^TDUの最大固有値、最小固有値、条件数を表5.3に示す。Aの条件数はDF＝10^{-3}のとき10^3のオーダー、DF＝10^{-6}のとき10^6のオーダーとなり、条件数が大きくなるとPCG法の反復回数が増加するのは確かである。しかし、U^TDUの条件数だけを比較すれば、MICCG法のほうがICCG法よりもいつでも若干大きい。このように、条件数からはMICCG法の収束がよいことを説明できない。

ICCG法とMICCG法は、次元n、拡散係数、右辺などによって収束に要する反復回数が変化するが、かなり安定で、前処理を施した行列$\widetilde{U}^TA\widetilde{U}^{-1}$が正定値でなくても、すなわち$\widetilde{d}_i^{-1}<0$でも収束することがあった。いずれの場合も前処理は有効で、場に対する依存性を和らげる効果もあった。

表５．３ 条件数，最大固有値λmax，最小固有値λmin

$m1=16$, $N=560$, $u=0.95$

DF	A	$M=U^TDU$							
		ICCG(1,1)	MICCG(1,1)	ICCG(1,2)	MICCG(1,2)	ICCG(1,3)	MICCG(1,3)	ICCG(2,4)	MICCG(2,4)
1	1.15×10^{3} $\frac{7.951}{0.71\times10^{-2}}$	17.2 $\frac{8.524}{0.494}$	186.0 $\frac{7.977}{0.42\times10^{-1}}$	41.8 $\frac{7.745}{0.185}$	365.7 $\frac{7.367}{0.20\times10^{-1}}$	63.4 $\frac{8.085}{0.127}$	483.3 $\frac{7.941}{0.16\times10^{-1}}$	129.1 $\frac{7.908}{0.61\times10^{-1}}$	1.72×10^{3} * $\frac{7.805}{0.45\times10^{-2}}$
10^{-1}	3.55×10^{3} $\frac{7.918}{0.22\times10^{-2}}$	60.9 $\frac{8.484}{0.139}$	230.5 $\frac{7.932}{0.34\times10^{-1}}$	67.0 $\frac{7.714}{0.115}$	485.5 $\frac{7.273}{0.14\times10^{-1}}$	83.4 $\frac{8.053}{0.96\times10^{-2}}$	711.0 $\frac{7.907}{0.11\times10^{-1}}$	156.8 $\frac{7.874}{0.50\times10^{-1}}$	1.49×10^{3} ** $\frac{7.775}{0.51\times10^{-2}}$
10^{-3}	2.83×10^{5} $\frac{7.918}{0.27\times10^{-4}}$	5.65×10^{3} $\frac{8.484}{0.14\times10^{-2}}$	9.69×10^{3} $\frac{7.932}{0.81\times10^{-3}}$	5.15×10^{3} $\frac{7.713}{0.14\times10^{-2}}$	7.20×10^{3} $\frac{7.261}{0.10\times10^{-2}}$	5.38×10^{3} $\frac{8.054}{0.14\times10^{-2}}$	6.97×10^{3} $\frac{7.906}{0.11\times10^{-2}}$	8.02×10^{3} $\frac{7.873}{0.98\times10^{-3}}$	5.16×10^{4} *** $\frac{7.797}{0.15\times10^{-3}}$
10^{-6}	2.83×10^{8} $\frac{7.918}{0.27\times10^{-7}}$	5.65×10^{6} $\frac{8.484}{0.14\times10^{-5}}$	9.67×10^{6} $\frac{7.932}{0.82\times10^{-6}}$	5.14×10^{6} $\frac{7.713}{0.14\times10^{-5}}$	7.18×10^{6} $\frac{7.261}{0.10\times10^{-5}}$	5.37×10^{6} $\frac{8.054}{0.14\times10^{-5}}$	6.95×10^{6} $\frac{7.906}{0.11\times10^{-5}}$		条 件 数 $\frac{\lambda_{max}}{\lambda_{min}}$

* $u=0.6$, ** $u=0.55$, *** $u=0.45$

(1) $\tilde{U}^{-T}A\tilde{U}^{-1}$の固有値分布

大規模問題におけるメモリネックを回避して、PCG法の反復作用素$\tilde{U}^{-T}A\tilde{U}^{-1}$の固有値分布を調べるため、

$$Aw=\mu U^T DUw$$

という一般固有値問題を解析した。A、$U^T DU$は対称正定値疎行列である。

m1＝16、n＝560、DF0 ＝ 10^{-12}, DF1 ＝ 10^{-6}, DF2 ＝ 10^{-4}, DF3 ＝ 10^{-6} としたとき、ICCG(1,1)においてu＝0 からu＝1.0まで変えたときに固有値分布がどのように変化するかを図5.6に、最小固有値μ_1、最大固有値μ_n、条件数Condを表5.4 に示す。ICCG(1,1) はMICCG(1,1) においてu＝0とおいた特別な場合になる。ICCG(1,1)は1の周辺に固有値を密集させるが、いくつかの小さい固有値が残ることがわかる。MICCG(1,1)のパラメータuは、ICCG(1,1)で密集させた固有値に対して最小固有値μ_1 をより1に近づけ、最大固有値μ_n を大きくして分布を全体的に右へずらすことがわかる。そのため、uを1.0に近づけると条件数が小さくなる。前処理を詳しくしたICCG(1,2)などよりもMICCG(1,1)が高速になるのは、このように分布を右へずらす効果によると考えられる。

ICCG(1,1) とu＝0.98 のときのMICCG(1,1)の固有値分布を図5.7 に示す。ICCG(1,1)ではμ_{243} ＝ 0.99127からμ_{320} ＝ 1.00181までが1に近接し、μ_{273}からμ_{288}までは完全に1になっている。MICCG(1,1) にこのような密集はない。MICCG(1,1)のときμ_1 ＝ 0.184511×10^{-5}に対する固有ベクトルw_1では37回、μ_2 ＝ 0.370624×10^{-3}に対する固有ベクトルw_2では14回、w_{189}とw_{205}では2回で収束し、その他の固有ベクトルでは1回で収束している。最大固有値は0.591024×10^{1}で、条件数は0.32×10^{7}になる。

u＝1.0になると、最小固有値μ_1 が1.0000、最大固有値μ_n が0.61028×10^{1} と固有値分布が大幅に変わるが、それでも対角要素$\tilde{d}_i^{-1}$は正のままである。同様の傾向はMICCG(1,2)、MICCG(1,3)の場合にも起こる。これらの場合には、Gustafsson の仮定とはかけ離れていることが原因であろう。

いずれの前処理でも多くの固有値が1の近傍に近寄る傾向はあるが、それでも10^{-3}以下の固有値は残ったままであり、固有値分布だけでは収束特性を説明できない。ましてや条件数はこの用途には無力である。条件数で説明できるのは、ICCG法ならばICCG(1,1),ICCG(1,2),ICCG(1,3)の収束特性の関係をみるとか、ひとつのICCG法、たとえばICCG(1,2)に限定して対象とする問題の相違による収束特性の違いを説明するといったことに限られるだろう。

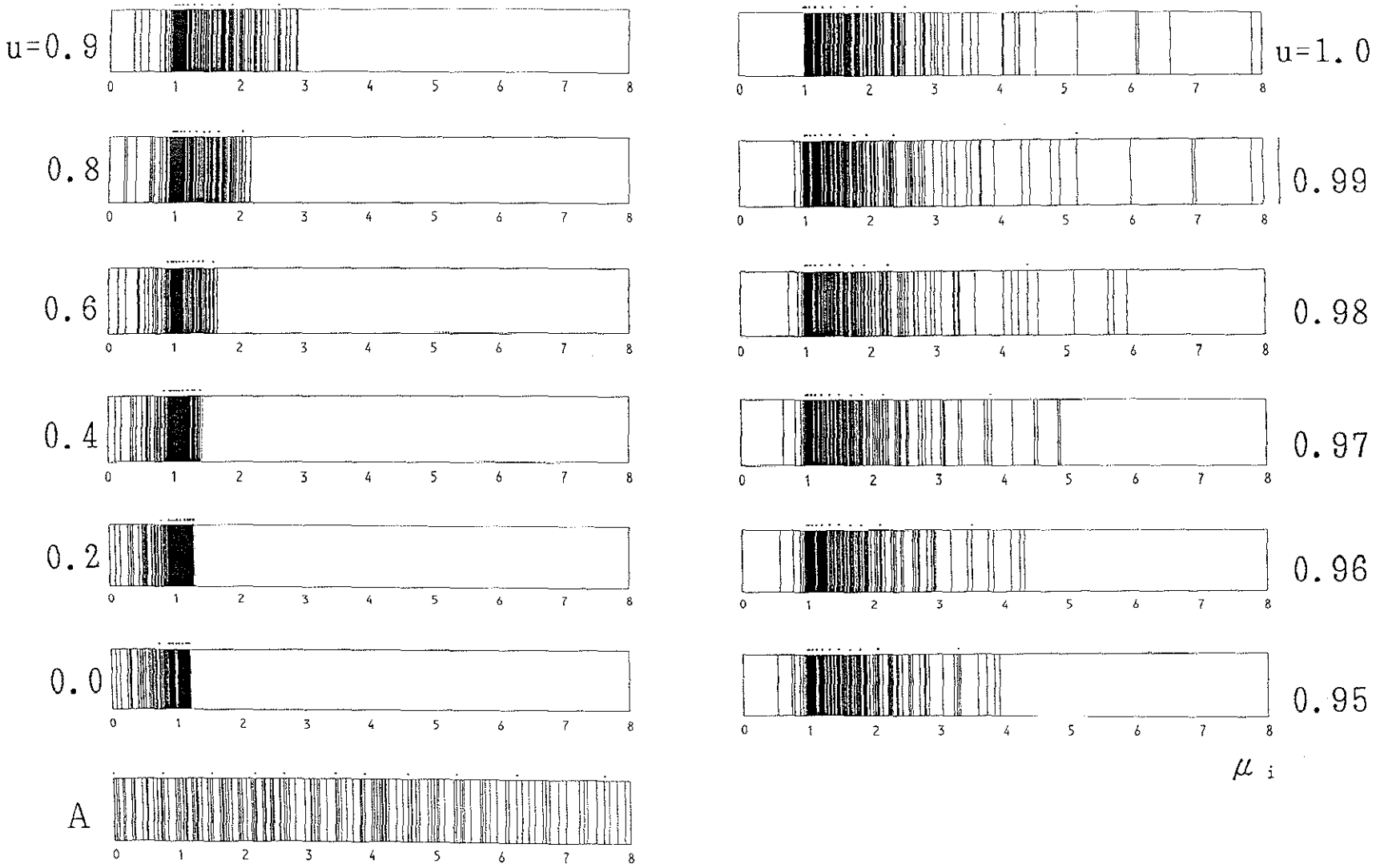


図５．６ MICCG(1,1)法の固有値分布

m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

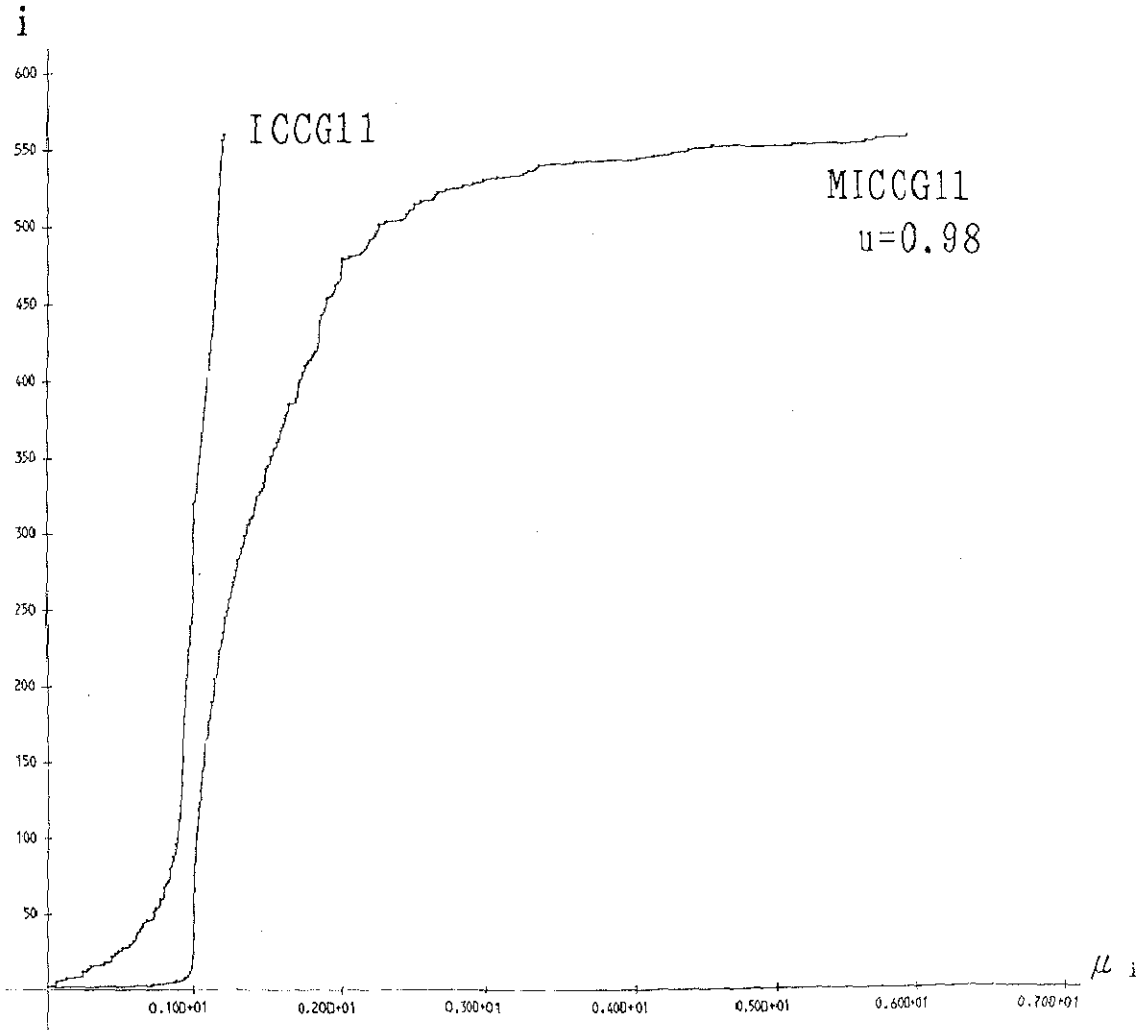


図５．７ ICCG(1,1)とu=0.98のMICCG(1,1)の固有値分布
m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

表5.4 MICCG(1,1)法の最小固有値、最大固有値、条件数

u	最小固有値 μ_1	最大固有値 μ_n	条件数
0	$0.568352070075405\times10^{-7}$	$0.121869482077242\times10^{1}$	0.21×10^{8}
0.2	$0.679443780511567\times10^{-7}$	$0.130834517262019\times10^{1}$	0.19×10^{8}
0.4	$0.859047039300119\times10^{-7}$	$0.144022277815298\times10^{1}$	0.17×10^{8}
0.6	$0.120526565804780\times10^{-6}$	$0.166288505344901\times10^{1}$	0.14×10^{8}
0.8	$0.219405453436998\times10^{-6}$	$0.216756791623452\times10^{1}$	0.99×10^{7}
0.9	$0.408406013586569\times10^{-6}$	$0.288841327693622\times10^{1}$	0.71×10^{7}
0.95	$0.774396817861467\times10^{-6}$	$0.391110939230800\times10^{1}$	0.51×10^{7}
0.96	$0.954700101187370\times10^{-6}$	$0.431149478151598\times10^{1}$	0.45×10^{7}
0.97	$0.125307290772385\times10^{-5}$	$0.488631930577816\times10^{1}$	0.39×10^{7}
0.98	$0.184511843266185\times10^{-5}$	$0.591024665036752\times10^{1}$	0.32×10^{7}
0.99	$0.360548851977467\times10^{-5}$	$0.823440503888139\times10^{1}$	0.23×10^{7}

表5.5 Gustafsson流の補正と最小固有値、最大固有値の変化

μ_n：最大固有値 μ_1：最小固有値	MICCG(1,1)	MICCG(1,2)	MICCG(1,3)
u=0	0.12186×10^{1} 0.56835×10^{-7}	0.12067×10^{1} 0.15616×10^{-6}	0.11492×10^{1} 0.23230×10^{-6}
u=0.95	0.39111×10^{1} 0.77439×10^{-6}	0.36542×10^{1} 0.20686×10^{-5}	0.36030×10^{1} 0.29950×10^{-5}
u=1.0	0.61028×10^{1} 0.10000×10^{1}	0.10796×10^{2} 0.10000×10^{1}	0.10870×10^{2} 0.10000×10^{1}

(2) 固有ベクトルによる解の構成

以下、m1＝16、n＝560、DF0 = 10^{-12}, DF1 = 10^{-6}, DF2 = 10^{-4}, DF3 = 10^{-6}の場合のみについて示す。固有ベクトル1本だけ与えた場合(a)で、ICCG(1,1)（実際はu＝0としたMICCG(1,1)を実行）、u＝0.95とu＝1.0の場合のMICCG(1,1)の結果を図5.8に示す。ICCG(1,1)とu＝0.95のMICCG(1,1)では、固有値分布が異なるだけで、収束の傾向はいずれも同じで、CG法の場合とも変わるところはない。ただし、u＝1.0にすると固有値分布が大幅に変わる。ICCG(1,1)の場合、μ_1 = 0.568352×10^{-7}に対する固有ベクトルw_1では46回、μ_2 = 0.114181×10^{-4}に対する固有ベクトルw_2では17回で収束し、その他の固有ベクトルでは1回で収束した。このときの最大固有値μ_nは0.121869×10^{1}で、条件数は0.21×10^{8}になる。前処理なしのCG法と比べると、条件数は1桁、$\mu_i \leqq 10^{-3}$の固有値の数と対応する固有ベクトルの反復回数も大幅に減少している。

ICCG(1,1)、ICCG(1,2)、ICCG(1,3)の結果を図5.9に示す。図5.8のICCG(1,1)やu＝0.95としたMICCG(1,1)と同様、固有値分布は変わるが10^{-3}以下の固有値2つはそのまま残り、前処理によって固有値の値が大きくなると収束に要する反復回数がいくらか減少する。その他の固有ベクトル成分はすべて1回の反復で収束し、倍精度計算でも充分に解けるようになった。

隣接する固有ベクトル2本(b)または3本(c)を与えた場合を図5.10と図5.11に示す。CG法と大きな違いはないが、隣接度が10^{-10}以下の固有値が減ったため、解を構成している固有ベクトルの本数よりも少ない反復回数で収束するようなケースが少なくなっている。これは、文献に書かれているPCG法の有効性の説明とは違っている[51],[54]。

低次の固有ベクトルから順に加え合わせて解を構成した場合(d)を図5.12～図5.15に示す。CG法と異なり、ICCG法では、固有ベクトルの本数を追加したとき反復回数の増え方がゆるやかになる。u＝0.95、u＝1.0とおいたMICCG法ではさらに増え方がゆるやかになる。ICCG法、u＝0.95、u＝1.0のMICCG法の場合にすべての固有ベクトルを加えた解Σw_iを与えると、MICCG(1,1)ではそれぞれ80、70、60回程度の反復で収束する。同様にMICCG(1,2)ではそれぞれ50、50、40回程度、MICCG(1,3)ではそれぞれ40、40、30回程度と、560回でも収束しなかったCG法に比べれば大幅に少ない回数で収束するようになっている。

高次の固有ベクトルから順に加え合わせて解を構成した場合(e)を図5.16、図5.17に示す。ICCG(1,1)、ICCG(1,2)、ICCG(1,3)とも増え方、収束に要する反復回数ともほぼ同じである。u＜1.0のMICCG法でもこの傾向は同じである。ところが、u＝1.0としたMICCG法では、増え方はそれほど急激ではないが、高次モードだけのときに収束に要する反復回数が他のuのときよりも多く必要になる。

これらを総合して考えると、uを大きくすることによって最小固有値μ_1がゆっくりと1に近づいてゆく。uを1にすると最小固有値μ_1は1となって、固有値分布は

完全に変わる。MICCG法の収束に要する反復回数は、解に低次の固有ベクトル成分から加え合わせるときにはuを大きくすると減少する。ところが、高次の固有ベクトル成分から加え合わせる場合は、固有値の分布が右にずれているだけで、収束に要する反復回数に変化はない。u=1の場合は特別で、固有値分布が大きく変化するとともに、高次の固有ベクトル成分だけで解を構成した場合に収束に要する反復回数が大きく増加する。

したがって、問題に応じてuを決めるなら、uの取り方は用途によって異なってくる。一般的な低次と高次のモードを共に豊富に含んだ問題を相手にするならば、uは1をこえない程度で大きくするのがよさそうである。この点で、従来からいわれている $0.8 \leqq u \leqq 0.975$ というのは妥当ともいえる[68],[56],[71]。ただし、MICCG(2,4)などで異なることは前に述べたとおりである。uの値と不完全コレスキー分解の詳しさによって、$\tilde{U}^{-T}A\tilde{U}^{-1}$の性質がどのように変化するかは今後調べてみる必要がある。

ICCG法、MICCG法でより詳しい近似を達成する不完全コレスキー分解を選べば、収束に必要な反復回数は確実に減少するが、1回の反復に要する演算量がより多く必要になる。しかも演算量とCPU時間は使用するコンピュータによって異なるため、解こうとしている問題、特に固有値μの分布や、方程式の元数によって最適な前処理は異なってくる。したがって、ここでMICCG(1,1)、MICCG(1,2)、MICCG(1,3)のどれが良いかをここで決めることは不可能である。結果的に問題と計算環境に合わせた選択が必要である。とはいえ、ここで行った実験によれば、収束に必要な反復回数のうえではMICCG(1,2)が無難であり、一般にMICCG(1,1)はMICCG(1,2)よりもおよそ30~40%は余計にかかる。MICCG(1,3)は2~5%短くなることが普通だが、時としてMICCG(1,2)より余計にかかることがある。

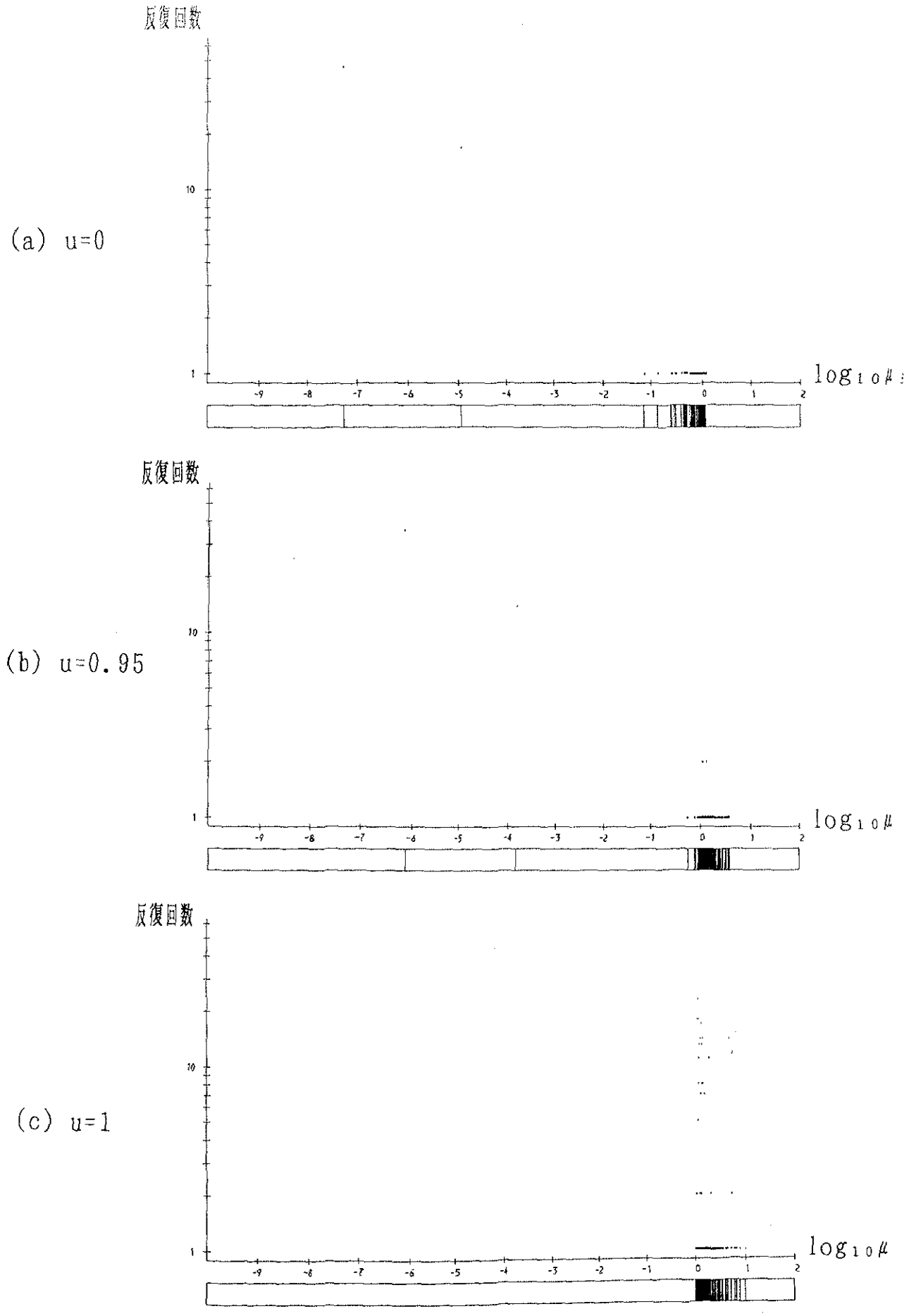


図５．８ 固有値分布$\log_{10}\mu_i$とw_iに対する反復回数 MICCG(1,1)
m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

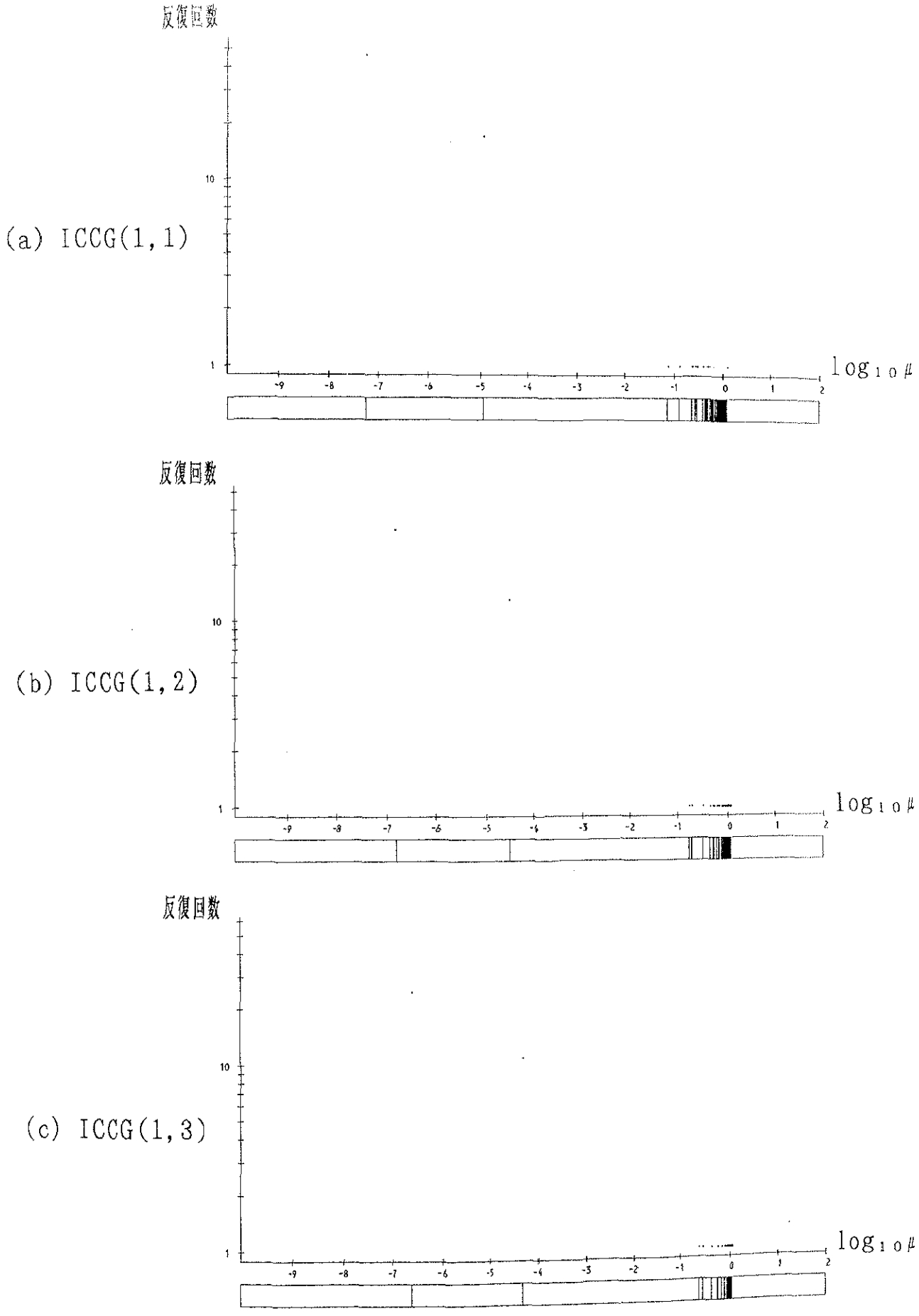


図５．９ 固有値分布$\log_{10}\mu_i$とw_iに対する反復回数
m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

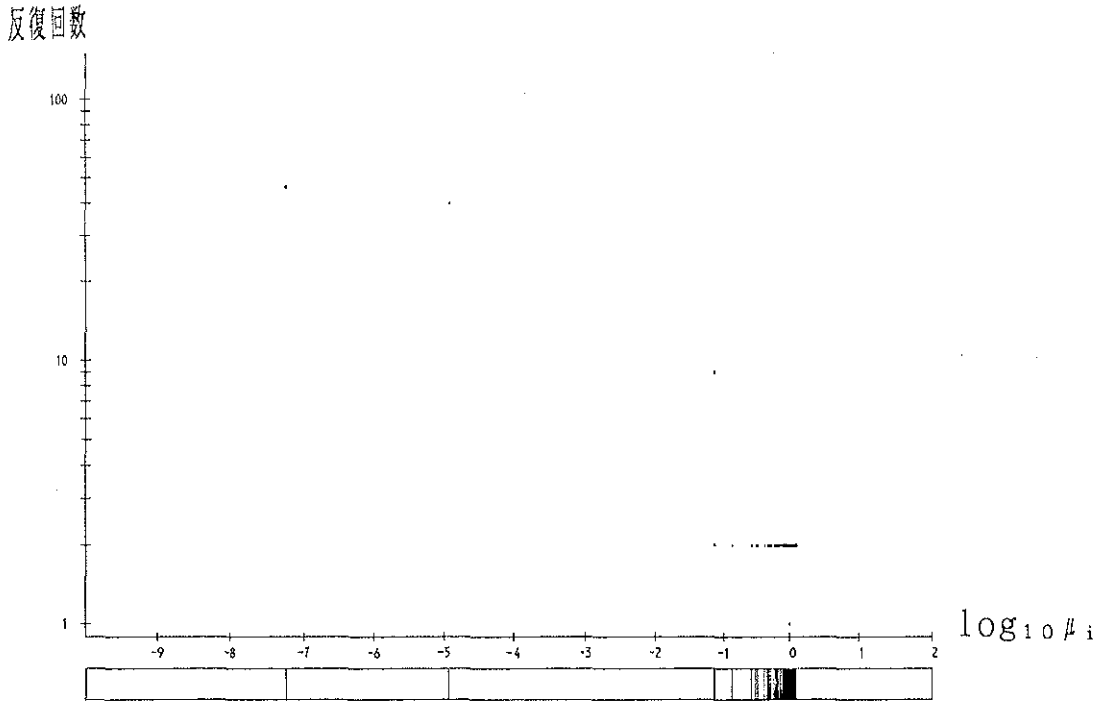


図５．10 固有値分布$\log_{10}\mu_i$と$\sum_{i=k-1}^{k} w_i$に対する反復回数 ICCG(1,1)
m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

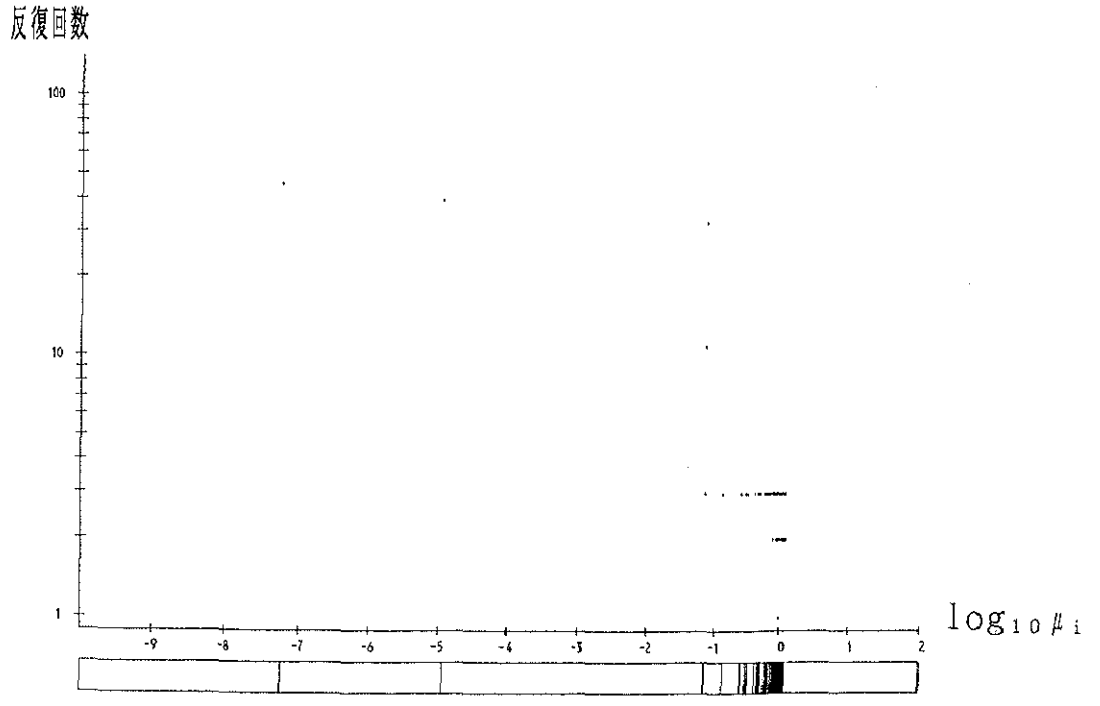


図５．11 固有値分布$\log_{10}\mu_i$と$\sum_{i=k-2}^{k} w_i$に対する反復回数 ICCG(1,1)
m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

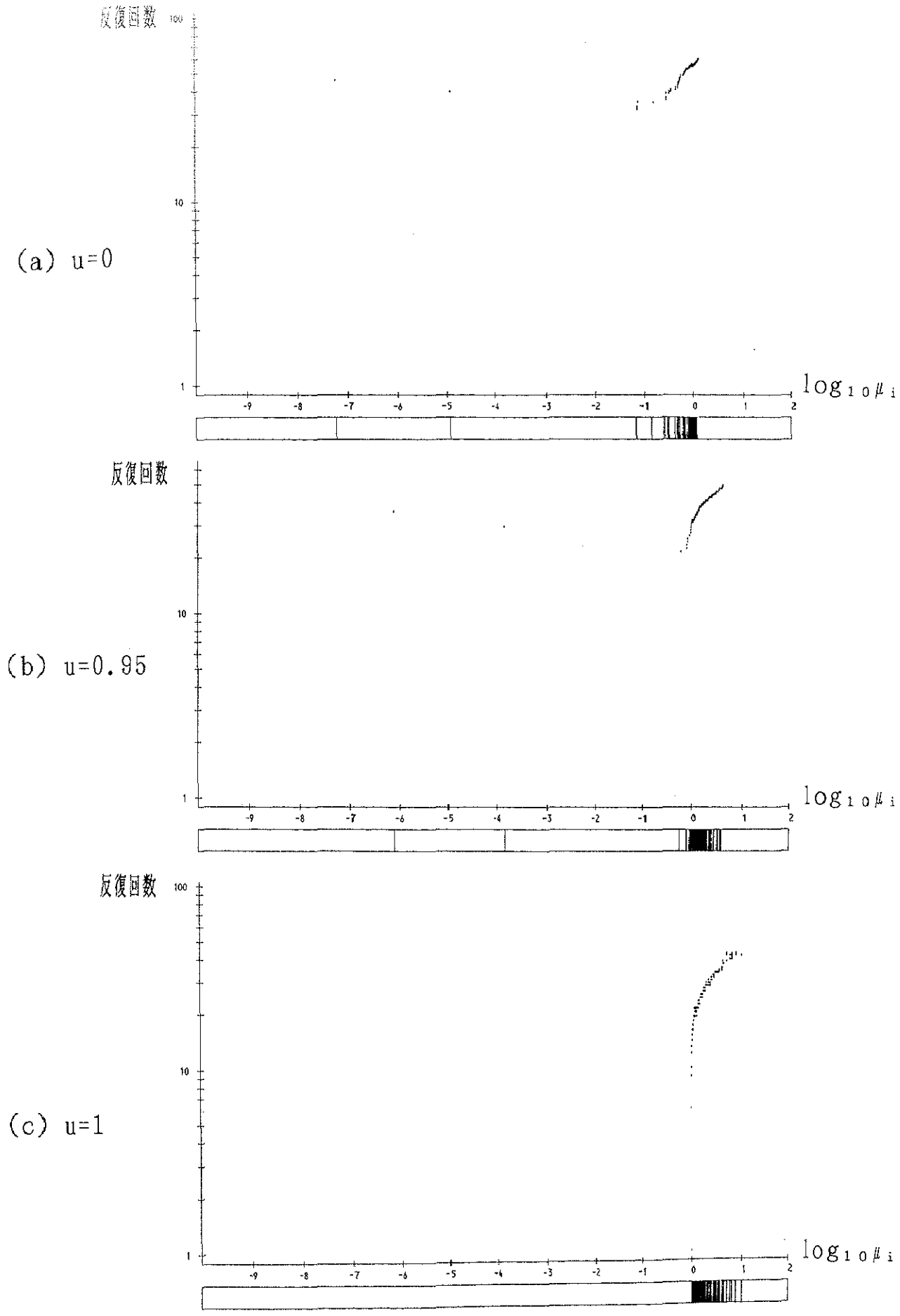


図５．12 固有値分布$\log_{10}\mu_i$と$\sum_{i=1}^{k} w_i$に対する反復回数 MICCG(1,1)

m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

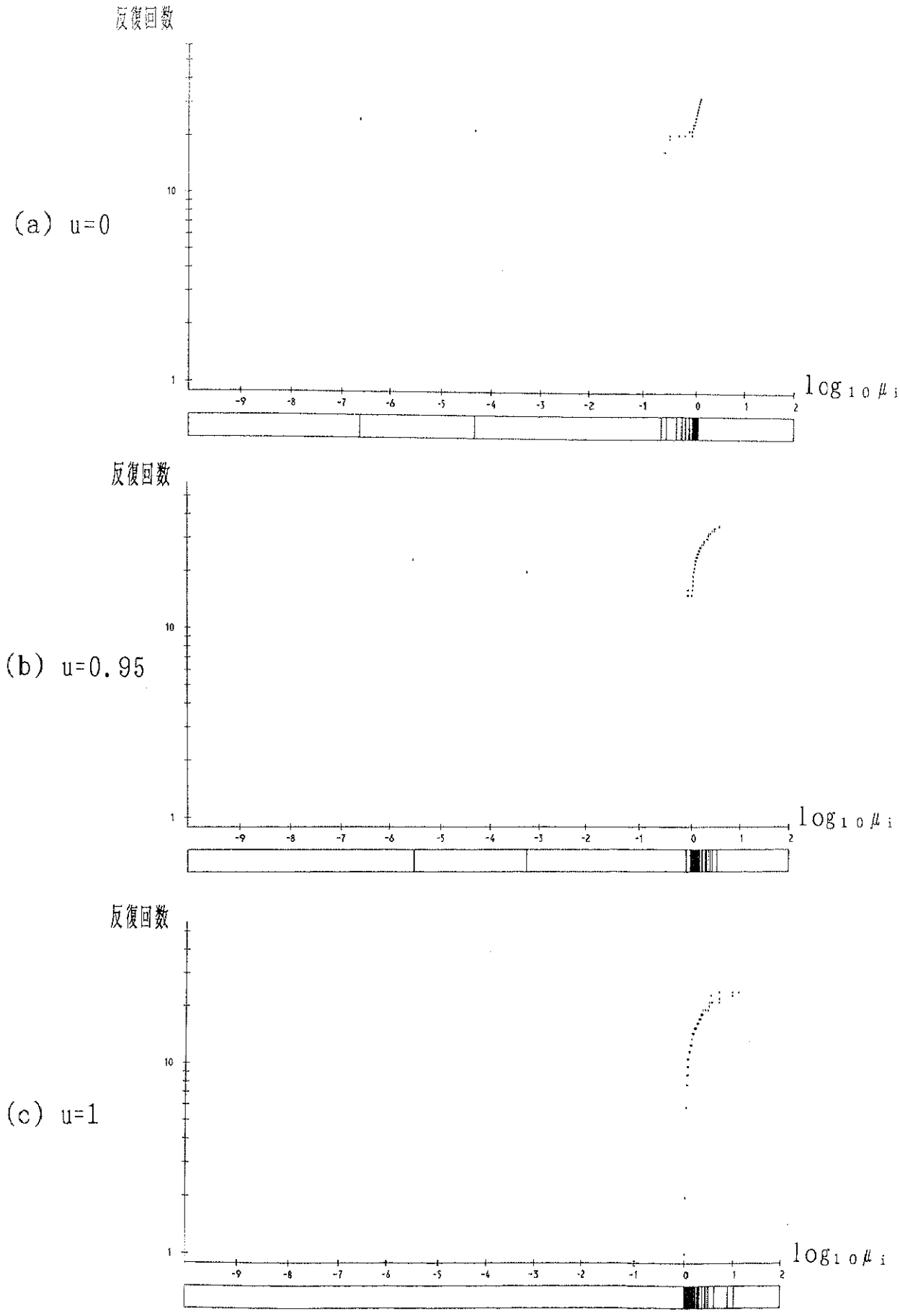


図５．13 固有値分布$\log_{10}\mu_i$と$\sum_{i=1}^{k} w_i$に対する反復回数 MICCG(1,3)
m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

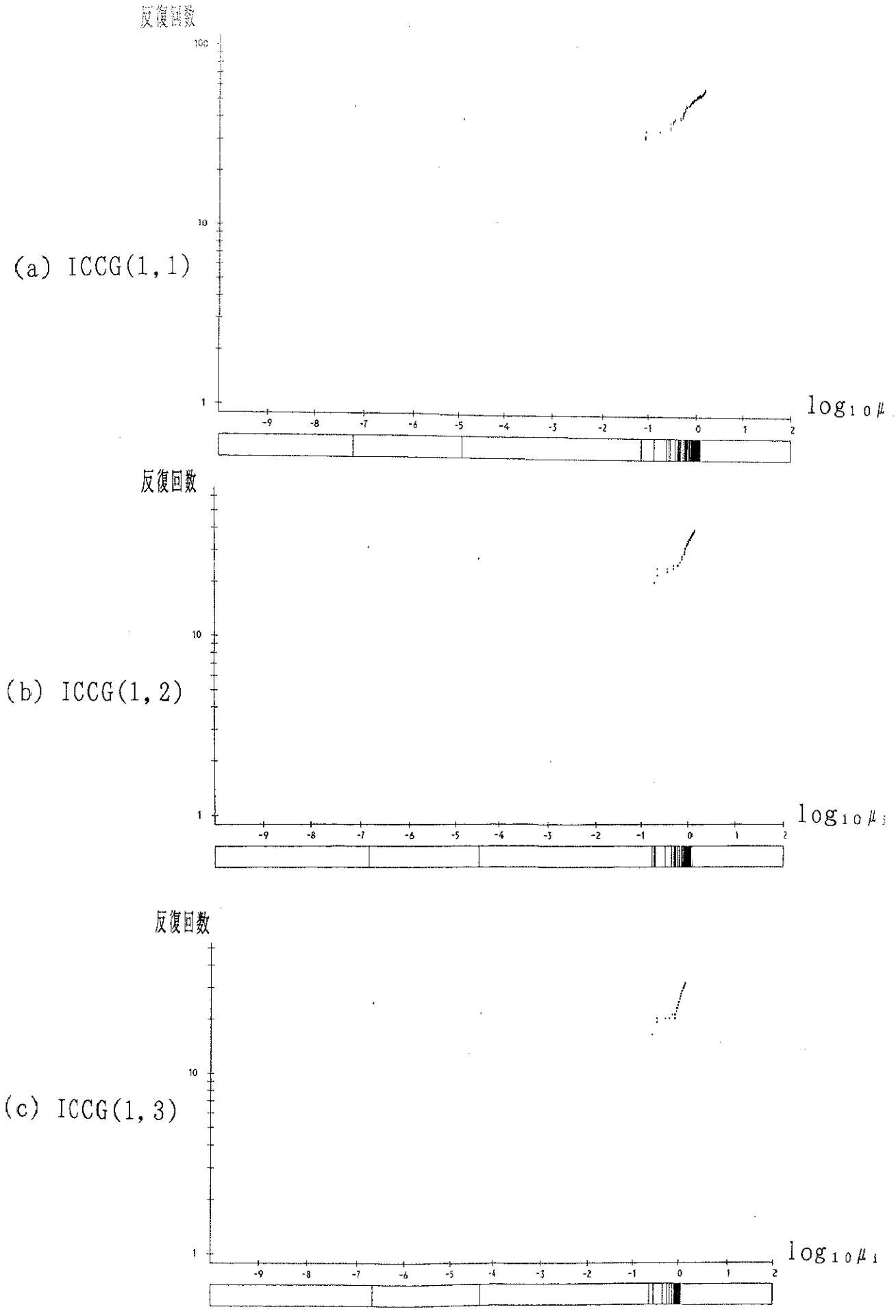


図５．14 固有値分布$\log_{10}\mu_i$と$\sum_{i=1}^{k} w_i$に対する反復回数

m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

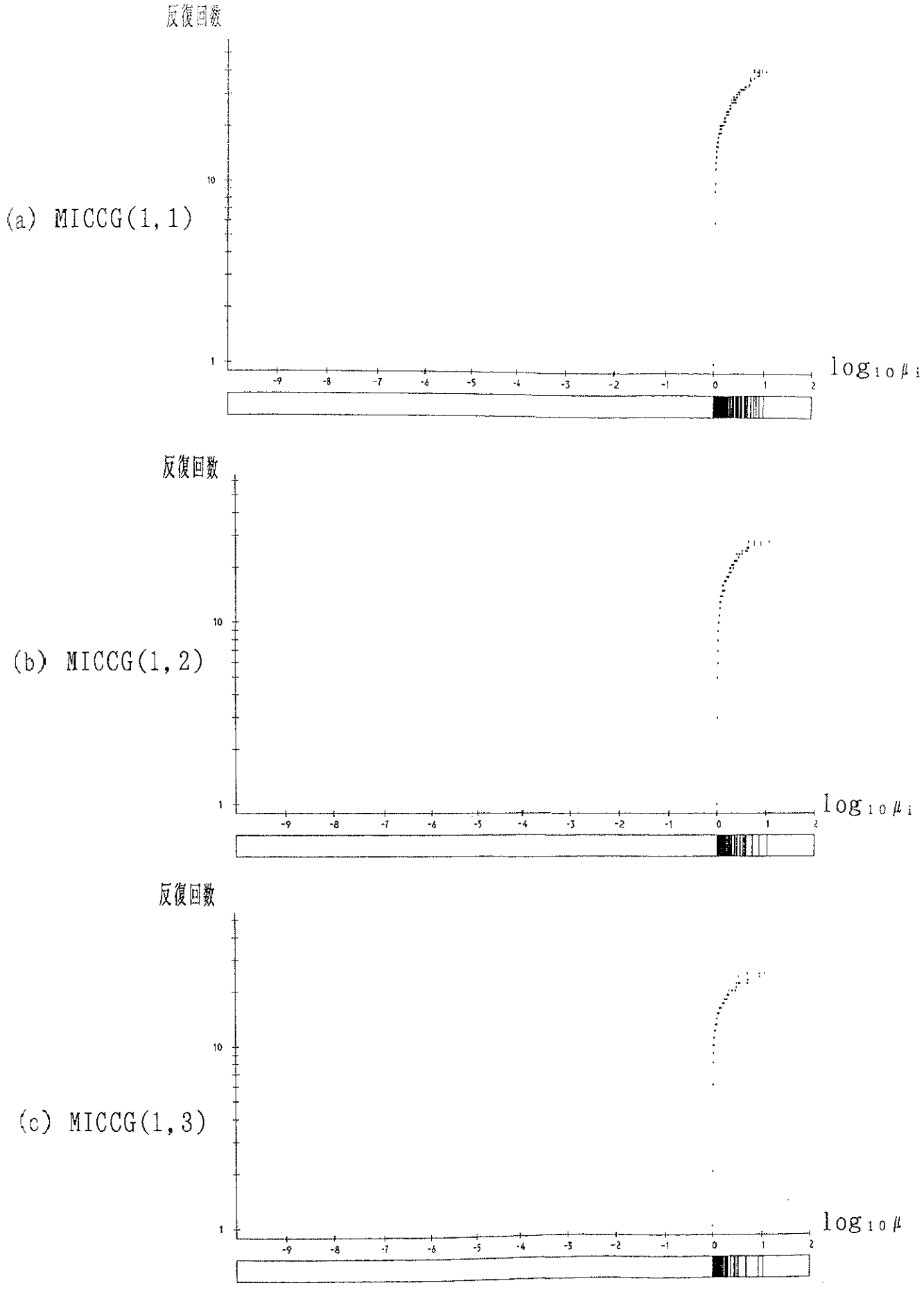


図５．15 固有値分布$\log_{10}\mu_i$と$\sum_{i=1}^{k} w_i$に対する反復回数 ；u=1.0

m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

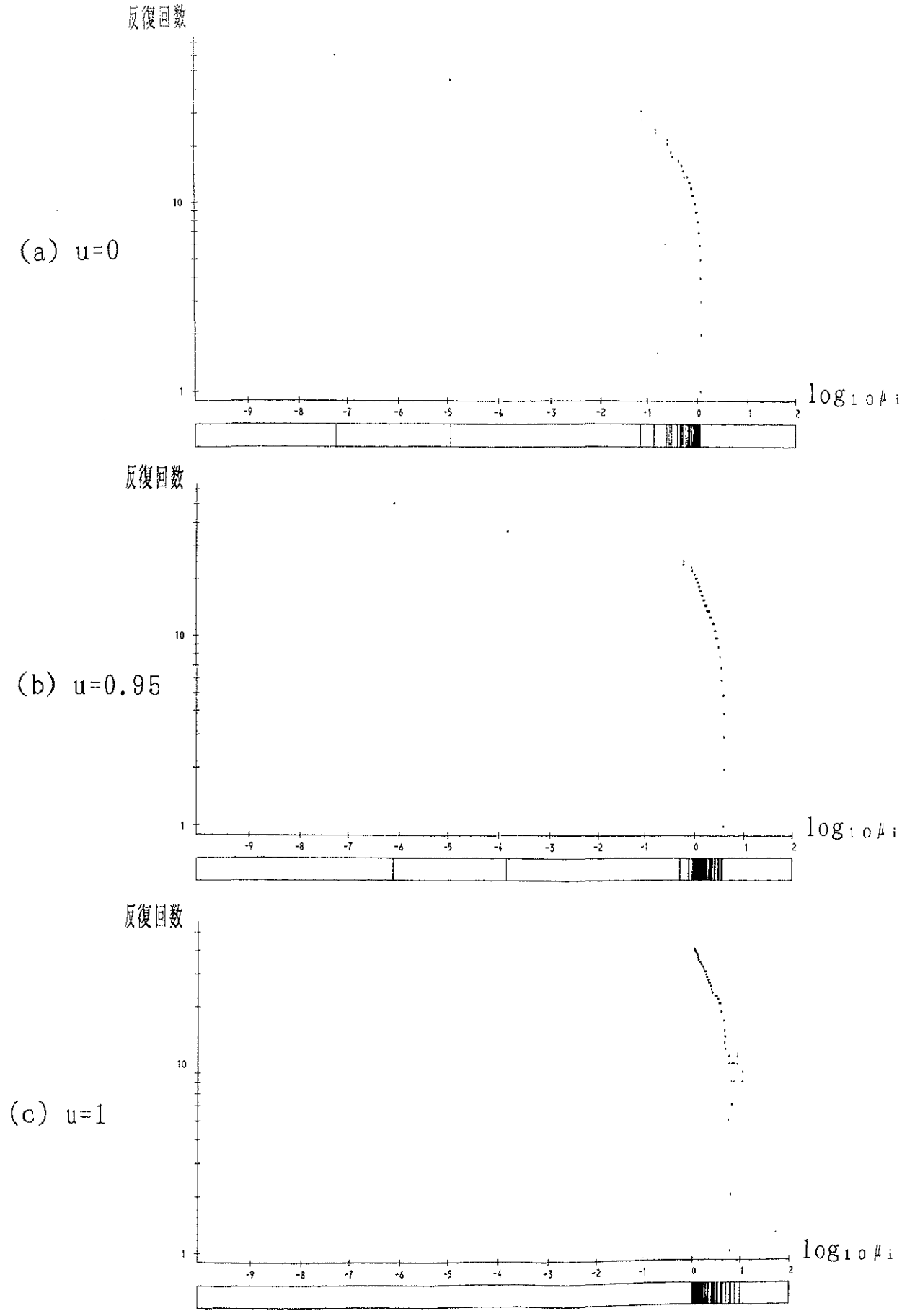


図５．16 固有値分布$\log_{10}\mu_i$と$\sum_{i=k}^{n} w_i$に対する反復回数 MICCG(1,1)
m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

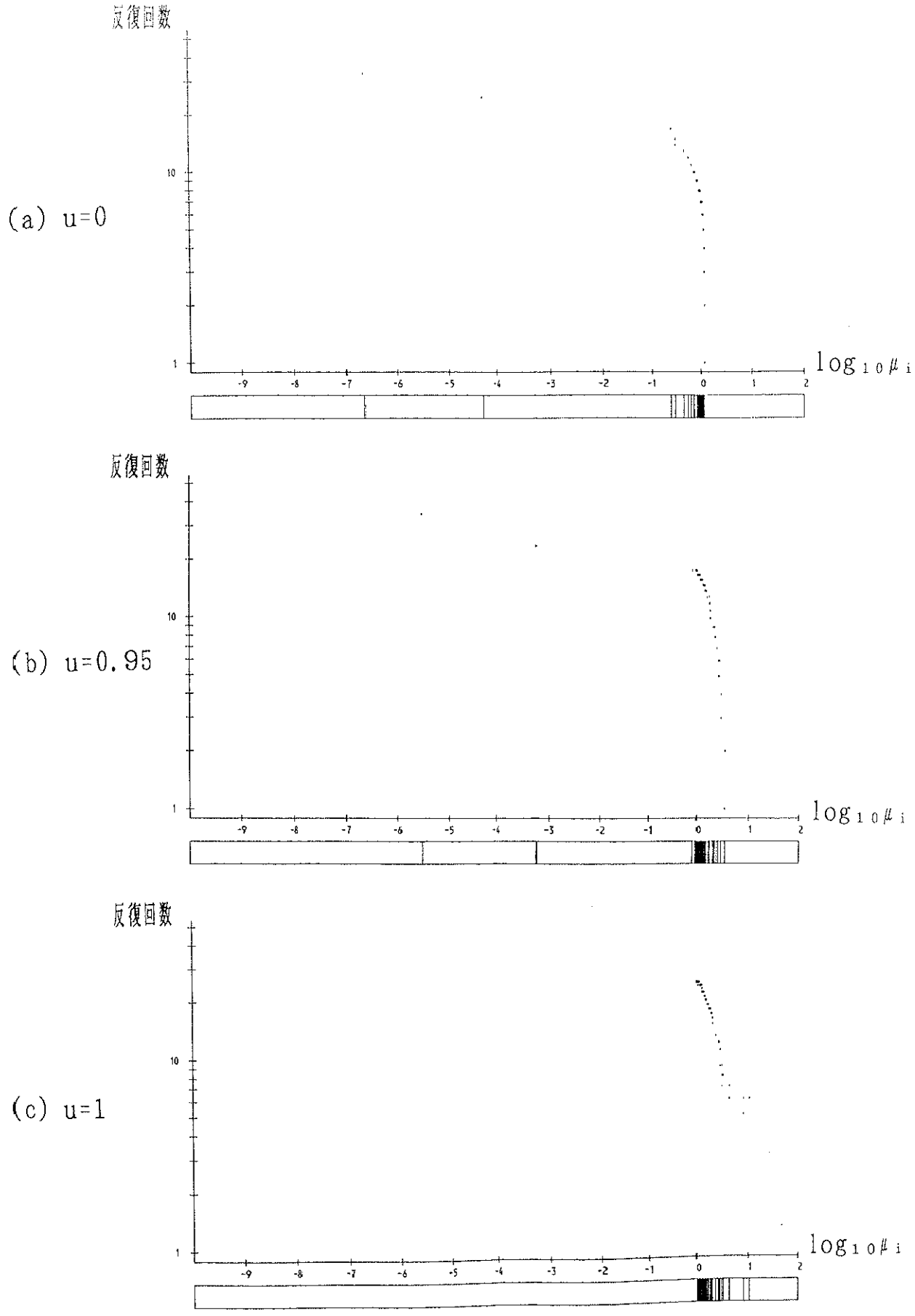


図5．17 固有値分布$\log_{10}\mu_i$と$\sum_{i=k}^{n} w_i$に対する反復回数 MICCG(1,3)

m1=16, n=560, DF0=10^{-12}, DF1=10^{-6}, DF2=10^{-4}, DF3=10^{-6}

5.4 望ましい前処理

大規模な疎行列に対しては、メモリ容量の節約のために反復解法を使う。ところが反復解法は直接解法よりも速く実用上十分な解が得られることもあるが、最悪の場合には収束しない可能性があり、安定で収束のよい実用的な反復解法のプログラムが必要とされている。

連立一次方程式$Ax=b$の解xを係数行列Aの固有値λ_i·固有ベクトルv_iによって展開することは、PCG法のために前処理$\widetilde{U}^{-T}A\widetilde{U}^{-1}$を施したときには、解$x$を一般固有値問題$Aw_i=\mu_i(U^TDU)w_i$の固有値$\mu_i$·固有ベクトル$w_i$によって展開することに対応する。すなわち、CG法では$Av_i=\lambda_i v_i$となる1本のv_iを解xとする問題、すなわち$Av_i=\lambda_i v_i$となるv_iを解xとする問題は理論上1回のCG反復で収束し、PCG法では$(\widetilde{U}^{-T}A\widetilde{U}^{-1})\widetilde{v_i}=\mu_i\widetilde{v_i}$となる1本の$\widetilde{v_i}$を解$y$,$y=\widetilde{U}x$とする問題、すなわち$Aw_i=\mu_iMw_i$, $M=U^TDU$となる1本のw_iを解xとする問題は理論上1回のCG反復で収束するはずである。

$Ax=b$をCG法を用いて解くときは解xが係数行列Aの固有ベクトル成分Σc_iv_iをどのように含むかによって、またPCG法を用いるときは$\Sigma\widetilde{c_i}w_i$ $(Aw_i=\mu_iU^TDUw_i)$をどのように含むかによって反復回数が大きく変化する。4章のCG法の実験結果によれば、10^{-3}以下の固有値に対応した固有ベクトル成分に対してCG法を用いるときには多くの反復回数を必要とすること、隣接する固有値の差が10^{-10}以下の場合は重複または密集とみなせて収束に要する反復回数が少なくてすむことが明らかになっている。したがって、収束特性の解析には

·固有値の分布

·解は固有ベクトルのどのような一次結合として表されるか

の両方を考えなければならない。

具体的な方法を示すのは難しいが、このような知識を活用すれば

(1) 小さな固有値に対応した固有ベクトル成分を含まないようにする

(2) あるいは収束の遅い固有ベクトル成分の収束を加速する

という条件を満足するような前処理によって、固有値·固有ベクトルの分布を変えてやればよいはずである。このとき、前処理に対する評価方法として、解または右辺を固有ベクトルによって展開する方法が利用できる。ただし、PCG法が対象とする連立一次方程式の大きさと、固有値解析が現実的に可能な行列の大きさには大きな開きがある。そしてこのような研究のためには4倍精度計算が倍精度計算の数倍以内でできるハードウエアと4倍精度演算に対応した固有値解析プログラムの整備が必要になる。

5.5 PCG法を用いた条件数の概算

連立一次方程式の誤差解析や反復解法の収束特性についての情報を得ようとするときには行列の条件数が必要となる。条件数だけでは収束特性の解析には大して役にたたないことは前に述べたが、それでも手軽な情報としての価値は否定できない。条件数の推定には色々な方法が提案されており、直接解法を用いて連立一次方程式を解くプログラムではLU分解と同時に条件数の推定が行われている[12],[34][33],[35],[55],[77]。しかし反復解法を使うようになると、このような方法は適用できない。

そこで、2次元の場における拡散方程式を差分法で離散化して得られた大規模な対称正定値疎行列の条件数を、ICCG法またはMICCG法で解くのに付随して簡単なプログラムと少ないCPU時間で概算する方法について述べる。

条件数は

$$\mathrm{Cond}(A) = \|A\|_2 \cdot \|A^{-1}\|_2$$

で定義されるが、Aが対称行列ならば

$$= |\lambda_{max}| / |\lambda_{min}|$$

となる。初歩的な固有値解析の知識を使えば、絶対値が最大と最小の固有値を求めるには、それぞれべき乗法と逆反復法を使えばよく[46]、目的とするのは条件数なので精度は2桁程度で充分である。必要な演算は、行列Aにベクトルx_kをかける操作、ベクトルの内積、スカラー倍などであり、逆反復法ではAを係数とする方程式を解くことが必要なので、ICCG法またはMICCG法を用いた逆反復法を行えばよい。しかも方程式を解くのと同時に条件数を概算する場合、新たに必要となるのはx_kとx_k'を格納する配列だけである。

(1) 最大固有値の概算

最大固有値λ_{max}を求めるにはAに対するべき乗法を使う。一般的な収束判定では固有値と固有ベクトルの両方の収束を調べるために残差ノルム$\|Ax_k - \lambda_k x_k\|$を用いるが、対称正定値行列の固有値はすべて正なので固有値の相対誤差$|\lambda_k - \lambda_{k-1}| / \lambda_k$でみてもよい。ここでは最大固有値の大まかな値だけが必要なので、最大固有値の付近に密集した固有値群がある場合にはそれらの平均値が求まってもよい。べき乗法は収束が遅いため、途中で反復を打ち切ると最大固有値に近い固有値のある種の平均値を概算していることになるが、その場合でも最大固有値の値とオーダーが変わることはないであろう。

(2) 最小固有値の概算

最小固有値λ_{min}を求めるにはAに対する逆反復法を使う。逆反復法ではMeijerinkの不完全$U^T DU$分解を前処理としたPCG法を用いて連立一次方程式を解く。このアルゴリズムは逆反復法の反復とPCG法の反復の2重の反復になるので、以後、前者を外部反復、後者を内部反復と呼ぶ。

最小固有値 λ_{min} は正の小さな値であり、条件数は最小固有値によって大きく変化する。そのため最小固有値はなるべく正確に求める必要があるが、λ_{min} は小さな値なので相対誤差 $|\lambda_k - \lambda_{k-1}| / \lambda_k$ では収束判定がしにくい。そこで外部反復の収束判定にはμ_k ($=\lambda_{min}^{-1}$)の相対誤差 $|\mu_k - \mu_{k-1}| / \mu_k$ を使う。相対誤差を用いるのは最大固有値の場合と同じ理由による。

内部反復で問題になるのは、収束判定値と初期ベクトルである。PCG法の収束判定値は、外部反復で用いる固有値の相対誤差 $rerr_k$ を用いて

$$\max(\min(10^{-2}\cdot rerr_k, 10^{-3}), 10^{-8})$$

とする。これは内部反復の収束判定値を、外部反復の相対誤差よりも2桁小さめとして最大は10^{-3}、最小は10^{-8}にするという意味である。外部反復の誤差が大きいときは内部反復の判定を甘くしてもよいことが経験的に知られている。PCG法は残差ノルムで収束判定を行うため、速く収束させるためには初期ベクトルの方向と大きさの両方を解に近づけることが重要であり、初期ベクトルは外部反復が収束したときの解 $\mu_k x_k$ にできるだけ近いものを選ぶ。そこでPCG法の初期ベクトルには

$$\mu_{k-1} x_k$$

を用いる。

(3) 数値実験

べき乗法と逆反復法の初期ベクトルには一様乱数ベクトルを用いる。理由は乱数ベクトルにはすべての固有ベクトル成分が含まれていると考えられるからである。べき乗法と逆反復法の収束判定値は$\varepsilon = 10^{-3}$とした。この値は普通に必要とされる値よりも1桁詳しい値である。べき乗法の最大反復回数は50回、逆反復法の最大反復回数(外部反復)は20回、外部反復1回に対するPCG法の反復回数(内部反復)の上限は

$$\max(KLIMIT/10, 30)$$

とした。KLIMITはPCG法で方程式の解を求める際の反復回数の上限値である。最悪の場合、内部反復の合計はPCG法で解を求めるときの反復回数の上限値の約2倍か、あるいは600回程度になる。

DFを1.0と10^{-6}にしたときの反復回数とHITAC M-660K上でのCPU時間を表5.6に示す。*は内部反復の総数である。

これらの問題では、べき乗法は反復25回程度、逆反復法は外部反復3～4回程度で収束した。逆反復法における内部反復の総数は最大でも400回程度であり、$x=(1,1,\cdots,1)^T$ となるような問題を解く際のCPU時間の約2倍に収まっている。これらの反復回数は初期ベクトル x_0' に依存し、乱数の値によっても変化するが、傾向は同じである。

$m1=16$、$n=560$、$DF=1.0$の問題で、べき乗法と逆反復法がどのように収束していくかを示したのが表5.7と表5.8である。べき乗法の収束が遅いことがわかる。べき乗法、逆反復法ともに、相対誤差が10^{-10}で残差ノルムは10^{-4}といったよ

表5.6 M-660Kでの 反復回数とCPU時間 (sec)

(* total counts for PCG iteration)

$m1$ (n)	iterations (CPU time)	power method λ_{max}	inverse iterations λ_{min}^{-1}	**solve** $Ax=b$ $\varepsilon=0.22\times10^{-10}$
64 (8384)	DF=1.0	24 (0.674)	4:257* (16.05)	134 (8.331)
	DF=10^{-6}	25 (0.706)	3:293* (18.40)	273 (16.92)
128 (33152)	DF=1.0	24 (2.822)	4:388* (101.1)	251 (65.08)
	DF=10^{-6}	24 (2.820)	2:198* (51.88)	503 (130.1)
256 (131840)	DF=1.0	24 (11.37)	3:297* (316.5)	474 (496.7)
	DF=10^{-6}	24 (11.33)	2:198* (209.1)	954 (989.0)

うに、相対誤差と残差ノルムによる収束判定の間には3~7桁の開きがある。ところが実際の固有値の近似値をみると、相対誤差による判定でも十分実用的なことがわかる。

表5.7 べき乗法の収束

(m1=16, n=560, DF=1.0)

relative error $\lvert\lambda_k-\lambda_{k-1}\rvert/\lambda_k$	iterations k	λ_{max} $0.795144501051437\times10^{+1}$	residual $\lVert Ax_k-\lambda_k x_k\rVert$
10^{-2}	10	$0.741994485110952\times10^{+1}$	0.44×10^{0}
10^{-4}	79	$0.785114105042031\times10^{+1}$	0.55×10^{-1}
10^{-6}	993	$0.794997043881958\times10^{+1}$	0.56×10^{-2}
10^{-8}	1819	$0.795143120337347\times10^{+1}$	0.56×10^{-3}
10^{-10}	2622	$0.795144487231575\times10^{+1}$	0.56×10^{-4}

表5.8 逆反復法の収束

(m1 = 16, n = 560, DF = 1.0, * total counts for PCG iteration)

relative error $\lvert\mu_k - \mu_{k-1}\rvert / \mu_k$	iterations k	$\mu_k = \lambda_{min}^{-1}$ $0.140233608465129\times10^{+3}$	residual $\lVert x_k - \mu_k A x_k \rVert$
10^{-2}	3:55*	$0.140228713428252\times10^{+3}$	0.47×10^{0}
10^{-4}	4:75*	$0.140233566025292\times10^{+3}$	0.57×10^{-1}
10^{-6}	5:92*	$0.140233593647460\times10^{+3}$	0.81×10^{-2}
10^{-8}		$0.140233608437951\times10^{+3}$	
10^{-10}	7:131*	$0.140233608437951\times10^{+3}$	0.35×10^{-3}

対照のため、$m1=16$、n = 560のときに、スツルム二分法･同時逆反復法を用いた厳密な固有値解析を行うと[50],[51]、DF = 1.0のときは

$$\lambda_{max} = 0.795144501051437\times10^{+1}$$

$$\lambda_{min} = 0.713095819857379\times10^{-2}$$

$$Cond(A) = 0.11505982633648\times10^{+4}$$

となる。この方法による条件数の概算は、相対誤差が10^{-4}のとき、λ_{max} は1桁、λ_{min} は6桁、条件数は2桁の精度で求まっている。

$m1=16$、N = 560、DF = 1.0の問題で内部反復の収束判定値を10^{-8}に固定したときには、PCG法の総反復回数は137回になる。内部反復の収束判定値を外部反復の収束に対応させて$10^{-2}\cdot rerr_k$ にすると 101回、$\max(\min(10^{-2}\cdot rerr_k, 10^{-3}), 10^{-8})$ にすると76回になる。したがって内部反復の収束判定値をうまく選んでむだな反復を行わないようにすると、PCG法の総反復回数を約60%に減少させられる。また、Aに対するべき乗法の収束が多少悪くても条件数に与える影響は小さい。

概算の程度は必要に応じて変えられる。通常CPU時間が気になるような大規模問題において、条件数を概算するのに必要なCPU時間はPCG法で方程式を解く場合のおよそ3倍以内である。得られた数値解の「質」についての手掛かりを得るためにこの程度の費用を払うことは許されることであろう。

逆反復法において別の反復解法を使って方程式を解いてもよい。その場合は内部反復の回数と与えられた方程式を解くための反復回数が変わるが、外部反復の回数は変わらない。条件数の概算に必要なメモリは、PCG法を用いて方程式を解くのに必要なメモリの他には x_k と x_k' を格納する配列だけである。このような条件数の概算方法は、一般の疎行列に対する反復解法の場合にも拡張できる。

なお、この方法で求めたλ_k、μ_k ($=\lambda_{min}^{-1}$) は値が大きくなりながら収束するので、条件数

$$Cond(A)=\lambda_{max}/\lambda_{min}$$
$$=\lambda_k \cdot \mu_k$$

は実際よりも小さめの値が概算されるが、実用上はこれで差し支えないであろう。PCG法とあわせてこのような方法を用いれば、大規模な対称正定値疎行列を係数とする連立一次方程式の誤差解析や反復解法の収束特性の解析に条件数が手軽に利用できる。

図4.1に示した3つのケースの条件数をこの方法で概算した結果を表5.9に示す。

表5.9 条件数の概算結果　ICCG(1,2)

m1=16, n=560	条件数	条件数の概算値	$Ax=b$をとく (s;秒)	条件数の概算 (s;秒)
(000)	0.11550×10^{4}	0.109×10^{4}	0.122s	0.252s (2.06倍)
(646)	0.28300×10^{9}	0.280×10^{9}	0.243s	0.323s (1.30倍)
(A8A)	0.28263×10^{13}	0.278×10^{13}	0.553s	0.447s (0.80倍)

6. PCG法の高速化

6.1 高速化の際の問題点

PCG法の主な利用分野である大規模問題ではメモリ容量とCPU時間が多量に必要となるため、PCG法のプログラムはベクトルパイプライン方式のスーパーコンピュータにおいて高速に実行されることが重要である。PCG法の代表的な前処理であるICCG法やMICCG法に現れる演算のうち、内積、ベクトルの和、行列とベクトルの積など、多くの演算はスーパーコンピュータに適合し高速に実行されるが、不完全$U^T DU$分解とその結果を用いた前処理

$$p' = (U^T DU)^{-1} r$$

は一次巡回演算となるためベクトルパイプライン方式のスーパーコンピュータでは高速化されないため[68],[45]、なんらかの対策が必要になる。

この場合、(1)なんらかの方法で不完全$U^T DU$分解とその結果を用いた前処理を高速化するか、(2)スーパーコンピュータ専用の前処理を導入し[11],[62],[70]、収束を多少犠牲にしても高速化を達成するかのいずれかである。しかし、(2)の方法では、コンピュータごとに前処理が異なり、収束特性の解析結果、解や誤差の比較が不可能になる。汎用的なプログラムにするなら、必然的に(1)の方法をとらざるをえない。

不完全$U^T DU$分解とその結果を用いた前処理に対する高速化の方法として、超平面法とよばれる方法が提案･利用されている。超平面法では、あらかじめ同時に実行可能な点のリスト(リストベクトル)を作成しておき、リストを参照しながら点のグループごとにベクトルパイプラインで同時に実行する[39],[45],[68]。そのため、1重のループが2重のループになることと間接的なメモリ参照になることで、スーパーコンピュータでは高速になるが汎用コンピュータでは遅くなる。汎用コンピュータにおいてループのオーバーヘッドは許容できても、メモリの間接参照は致命的である。しかも、スーパーコンピュータで高速に実行されるとはいえ、スカラー実行と比較した場合のことであり、実メモリ方式のスーパーコンピュータであってもメモリの間接参照は好ましくない。スーパーコンピュータと汎用コンピュータの双方で高速に実行するには、なんらかの対策が必要である。

6.2 超平面法とその問題点

まずは、超平面法のアルゴリズムをあらい直してみる。不完全$U^T DU$分解を用いた前処理

$$r' = (U^T DU)^{-1} r$$

を節点iに作用させるときの前進代入と後退代入で参照される点を図6.1に黒丸で

示す。図6.1に黒丸で示された点を点iと同一のループ内で参照･更新すると、ベクトルパイプライン方式のスーパーコンピュータでは解の正当性が保証されない。そこで、これらの点を同時に参照･更新しないようなグループをうまく作ってやればよい[27]。

それには、係数行列のもとになった2次元矩形領域を図6.2のような座標として表し、前処理を作用させるときに参照される点を含まない超平面(2次元空間では直線)の方程式を求めればよい。0≦x≦m2-1、0≦y≦m1-1、n=m1･m2であり、節点番号は(0,0)からy軸方向につけるものとする。したがって、節点番号iと座標x、yの関係は

$$i=1+x\cdot(m1-1)+y \qquad (6.1)$$

となる。i、x、yは非負の整数値しかとらない。以下の議論は、x軸方向に番号づけした場合でも同様にできる。

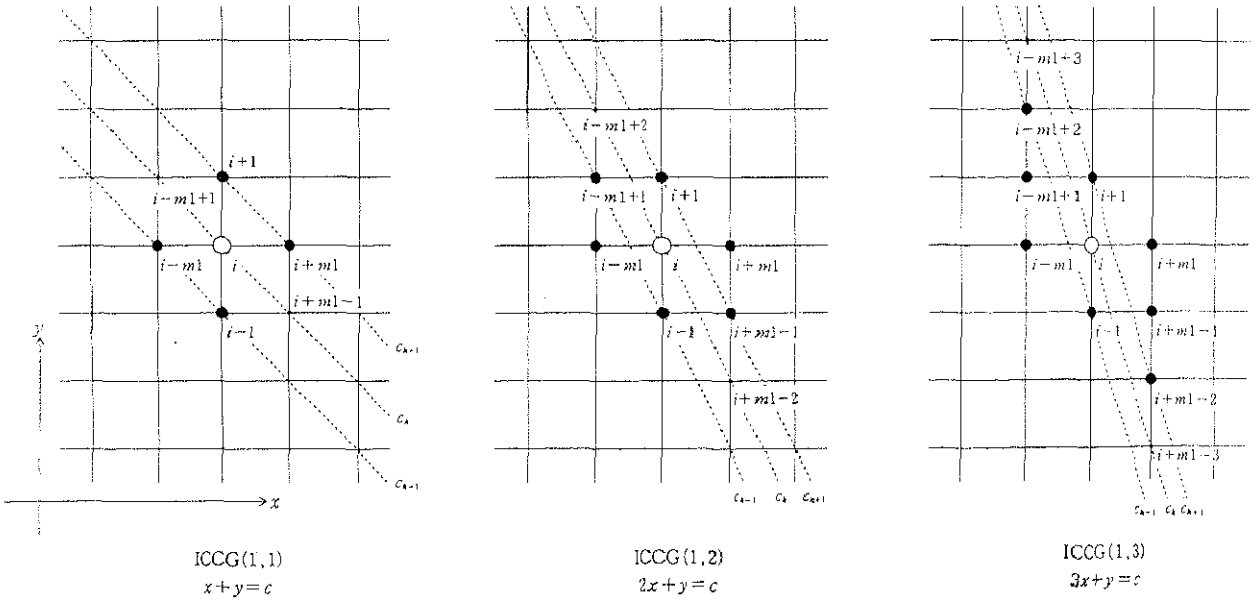


図６．１　$(U^T D U)^{-1} r$ で参照・更新される点　　○更新, ●参照

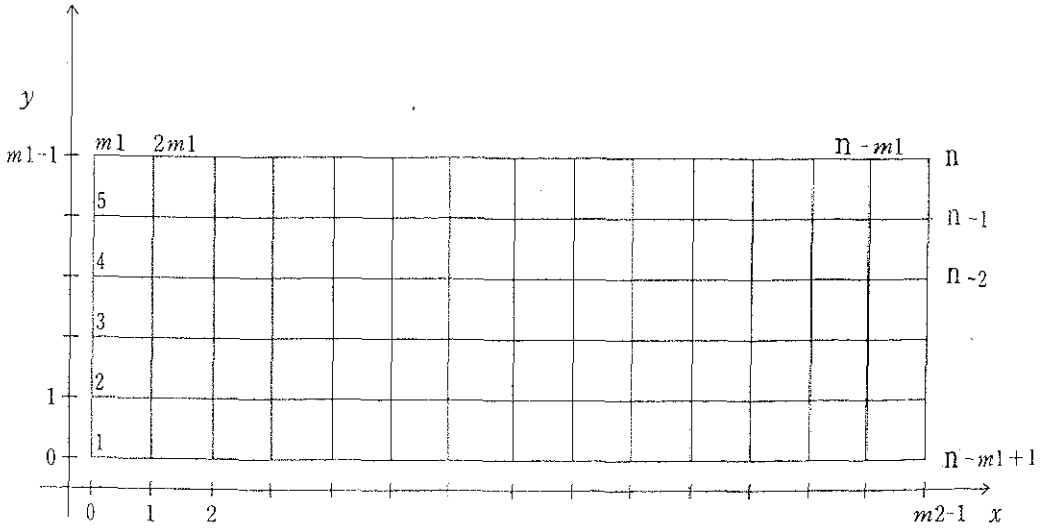


図６．２　超平面法を適用する座標系　(m1=6, n=90, m2=n/m1)

ICCG(1,1)ならびにMICCG(1,1)の前処理で節点iと同時に実行できる点にはi-m1±1、i+m1±1、…などがあり、直線としてはy+x=cとy−x=cが考えられる。リストベクトルによる順序づけに関しては、MICCG法とICCG法は全く同じなので、以下ICCG法に関してのみ述べる。また、MICCG法は反復回数が変わるだけで反復1回あたりのCPU時間についてもICCG法とほとんど同じである。前進消去は節点1から行う必要があることと、後退代入は節点nから行う必要があることから

$$y+x=c_k\ (\text{一定}) \tag{6.2}$$

が求める超平面となる。図6.1は、前進消去では節点iの計算が終われば上側の点i+1と右側の点i+m1の計算が同時にできることも示している。x、yが決められた範囲を動くと

$$0 \leqq x+y \leqq m2+m1-2$$

となるので、この範囲内の値c_kについて節点をグループ分けすればよい。

同様にICCG(1,2)の場合は

$$2x+y=c_k\ (\text{一定}),\ 0 \leqq 2x+y \leqq 2m2+m1-3, \tag{6.3}$$

ICCG(1,3)では

$$3x+y=c_k\ (\text{一定}),\ 0 \leqq 3x+y \leqq 3m2+m1-4 \tag{6.4}$$

となる。このようなリストベクトルを使うと平衡時のループ長は(6.2)がm1、(6.3)がm1/2、(6.4)がm1/3となる。前処理が複雑になると平衡時のループ長が短くなるが、平衡状態は長くなる。

実際のプログラムでは、x、yの値と超平面の距離c_kを

```
do i = 1, n
   x = (i-1)/m1
   y = mod(i-1,m1)
   c_i = f(x,y)
```

で決定し、これらをc_iの値でグループ分けしてリストベクトルを作ればよい。リストベクトルの作成はベクトル実行が難しいが、不完全U^TDU分解と同様、最初の1回だけなので全体の実行時間に対する影響はわずかである。m1=16の問題にリストベクトルを用いたとき、ループ長が前処理によってどのように変化するかを図6.3に示す。

文献[27]によれば、スーパーコンピュータS-820/80では、リストベクトルの作成とリストベクトルを用いた不完全U^TDU分解のCPU時間は、リストベクトルを使わない不完全U^TDU分解のCPU時間よりも小さく、ときには半分程度になる。汎用コンピュータM-660Kでリストベクトルを作って使うと、リストベクトルを使わないときの1.3倍から1.5倍のCPU時間が必要になる。

表6.1にある問題に対する反復過程のCPU時間を示す。スーパーコンピュータS-820/80でリストベクトルを用いると、m1= 256の問題ではICCG(1,1)が15倍、

ICCG(1,2)が14倍、ICCG(1,3)が12倍に高速化される。したがって、スーパーコンピュータで高速化を図るためにはリストベクトルの使用が必須である。ところが汎用コンピュータではリストベクトルを使うと10%~30%遅くなる。

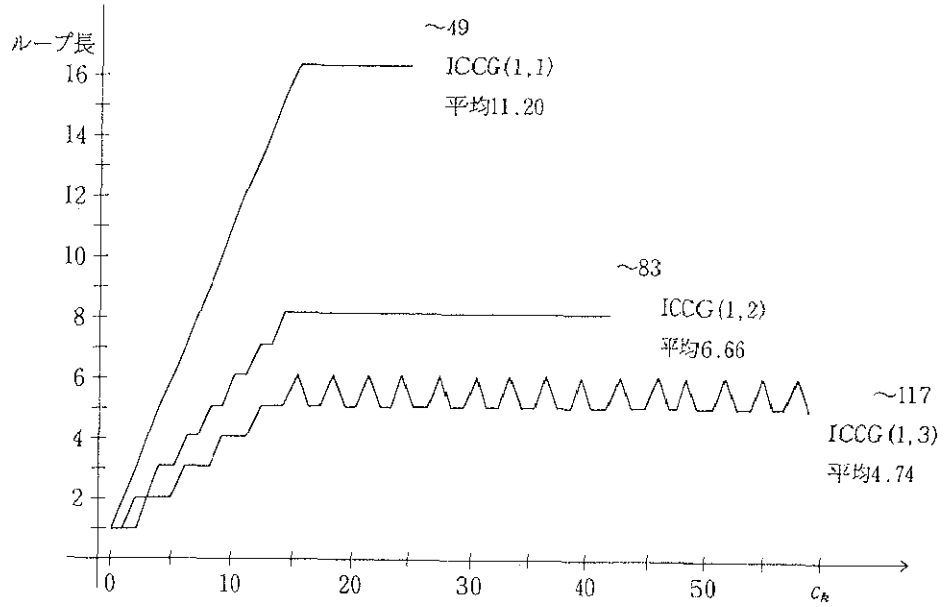


図６．３ 超平面の距離 c_k とループ長

m1=16, m2=35, n=560 中央までを表示

表６．１ ＩＣＣＧ法の反復過程のＣＰＵタイム（s）

東京大学大型計算機センターにて 1990.10.13.
図書館情報大学にて 1990.10.13.～17.

$m1$ (n)	(　　)内は反復１回あたり	反復回数	S-820/80		M-660K	
			リストベクトル使用		リストベクトル使用	
64 (8384)	ICCG(1,1)	134	0.106	0.839	9.226	8.294
	ICCG(1,2)	89	0.110	0.674	6.867	6.311
	ICCG(1,3)	72	0.139	0.650	6.042	5.487
128 (33152)	ICCG(1,1)	251	0.542	6.537	72.749	66.734
	ICCG(1,2)	169	0.515	5.222	52.623	48.390
	ICCG(1,3)	136	0.620	4.993	47.008	42.985
256 (131840)	ICCG(1,1)	474	3.176 (0.006)	49.761 (0.104)	630.441 (1.330)	493.411 (1.040)
	ICCG(1,2)	318	2.780 (0.008)	39.546 (0.124)	404.186 (1.271)	359.596 (1.130)
	ICCG(1,3)	259	3.152 (0.012)	38.193 (0.147)	360.848 (1.393)	318.184 (1.228)

全体のCPU時間は反復1回あたりのCPU時間に収束までの反復回数をかけたものになる。表6.1のカッコ内に反復1回あたりのCPU時間を示す。スーパーコンピュータでリストベクトルを使ったICCG(1,1)を1とすると、ICCG(1,2)が1.3倍、ICCG(1,3)が2.0倍になる。汎用コンピュータでリストベクトルを使わないと、ICCG(1,2)が1.1倍、ICCG(1,3)が1.2倍となる。汎用コンピュータでの値は大体の演算量を反映している。スーパーコンピュータにおけるICCG(1,3)のように、1回の反復に対する高速化の程度が小さくても反復回数が大幅に減少するなら、全体のCPU時間は少なくなる。CPU時間は前処理による反復回数の減少と、前処理の高速化の程度によって決まる。したがって、一概にどの前処理がよいと断言することはできない。

6.3 問題向きの汎用的なプログラム

以下では、1回の反復をスーパーコンピュータでどのように高速化するかについて述べる。スーパーコンピュータではリストベクトルを使うことで10倍以上の高速化が図れたが、汎用コンピュータでリストベクトルを使うとオーバーヘッドが大きくなる。そこで、スーパーコンピュータと汎用コンピュータの双方で高速に実行するための対策を考える。

(6.1)、(6.2)、(6.3)から、それぞれの超平面の方程式は、

ICCG(1,1): $y = c_k - x$、
ICCG(1,2): $y = c_k - 2x$、
ICCG(1,3): $y = c_k - 3x$

とも書ける。これらの方程式でc_kの値を固定してxの値を1つずつ変えると、yの値は一定の値だけ増減する。これを節点番号またはメモリ参照として考えれば、リストベクトルとはいっても、内容的には等間隔参照になっていることを示している。したがって、ベクトルパイプライン方式のスーパーコンピュータにおいて同時実行が可能な点のリストの先頭と末尾がわかれば、ストライドを指定した2重のループを構成すればよく、リストベクトルを用いてメモリの間接参照をする必要はない。ここでは、リストベクトルはそのような点のリストの先頭と末尾の情報だけを保持していればよい。メモリの間接参照さえ避ければ、汎用コンピュータでも許容できる程度のオーバーヘッドで済むはずである[13]。

各ICCG法に対する超平面の方程式はすでに求められているから、ベクトルパイプライン方式のスーパーコンピュータにおいて同時実行が可能な点のリストの先頭と末尾をリストベクトルとして保持することと、ほぼ等価な情報を毎回生成するようなプログラムを作ることができる。

まず、xが1増えればICCG(1,1)のyは1だけ減少することから、ICCG(1,1)における同時に実行できる節点番号の差、すなわちループのストライドはm1-1とな

る。同様な理由で、ループのストライドはICCG(1,2)でm1−2、ICCG(1,3)でm1-3となる。

次に原点からの距離がlの超平面に属する節点のうちで、最小の節点番号と最大の節点番号を求める。ICCG(1,1)の場合、図6.2のように考えれば、最小の節点番号は(0,l)または(x,m1−1)のいずれかにあるのは明らかである。m=m1−1とすれば、後者は

$$\begin{aligned} x &= l-y \\ &= l-(m1-1) \end{aligned}$$

より、

$$\begin{aligned} i &= 1+(l-m)*m1+m1-1 \\ &= (l-m+1)*m1 \end{aligned} \tag{6.5}$$

となる。したがって、最小の節点番号i_{min}は$\max((l-m+1)*m1, l+1)$である。原点からの距離がlの超平面は(x,0)を通るが、そのときの節点番号がnを越えないようにすれば、最大の節点番号が求まる。実際は、プログラミング言語FORTRAN77などで1以外のストライドを指定したDOループでは、厳密な終点が必要ではなく最大の節点番号があり得る範囲の最大値を指定すれば充分である。したがって、最大の節点番号の上限i_{max}は$\min(l*m1+1, n)$でよい。

ICCG(1,2)の場合、最小の節点番号は(x,m1−1)、(2,m1−2)または(0,l)のいずれかにある。超平面の方程式の傾きが2になったため、このような場合分けが必要になる。m=m1−1として、プログラムふうに記述すれば

```
if l < m then
    i = l+1
else if mod(l-m,2) = 0 then
    i = m1*((l-m)/2)+1)
else
    i = m1*((l-m+1)/2)+1)-1
end if
```

となる。最大の節点番号の上限i_{max}は$\min(m1*(\mathrm{int}(l/2)+\mathrm{mod}(l,2)+1, n)$である。

ICCG(1,3)の場合、最小の節点番号は(x,m1−1)、(x,m1−2)、(x,m1-3)または(0,l)のいずれかにある。超平面の方程式の傾きが3になったため、さらに場合分けが必要になる。m=m1−1として、プログラムふうに記述すれば

```
if l < m then
    i = l+1
else if mod(l-m,3) = 0 then
    i = m1*int((l-m)/3)+m1
else
    i = m1*int((l-m)/3)+2*m1+mod(l-m,3)-3
```

end if

となる。最大の節点番号の上限i_{max}はmin(m1*(int(l/3)+mod(l,3)+1, n)である。

これらをプログラムとして陽に作り込めば、毎回のループで若干のオーバーヘッドはかかるが、リストベクトル用の領域が不要になるためプログラムは単純になる。m1=256、n=131840の問題について、スーパーコンピュータS-3800と汎用コンピュータM-660Kにおける全体のCPU時間と各コンピュータ上での加速率を表6.2に示す。この場合、全体の処理に対するリストベクトルの作成、U^tDU分解などは無視しうる値である。

表6.2　ICCG法の高速化　(s:秒,カッコ内は倍率)

(a) ICCG(1,1), 474回	S-3800	M-660K
原形	22.633(1.00)	494.76(1.00)
リストベクトル 間接参照	1.705(13.27)	656.62(1/1.32)
リストベクトル 等間隔参照	1.232(18.37)	606.06(1/1.22)
改良	1.219(18.56)	600.99(1/1.21)

(b) ICCG(1,2), 318回	S-3800	M-660K
原形	19.162(1.00)	360.66(1.00)
リストベクトル 間接参照	1.658(11.55)	444.27(1/1.23)
リストベクトル 等間隔参照	1.221(15.69)	399.44(1/1.10)
改良	1.196(16.02)	403.16(1/1.11)

実メモリ方式のスーパーコンピュータS-3800上でメモリ参照を間接参照から等間隔参照にすると、ICCG(1,1)とICCG(1,2)では40%弱、ICCG(1,3)では約30%の高速化が達成される。仮想メモリ方式の汎用コンピュータM-660K上で間接的なメモリ参照によるオーバーヘッドはICCG(1,1)が30%、ICCG(1,2)が20%、ICCG(1,3)が15%である。これを等間隔参照に変更することで、ICCG(1,1)が20%、ICCG(1,2)が10%、ICCG(1,3)が5%といずれもオーバーヘッドが減少する。ICCG(1,1)では、それでもオーバーヘッドが大きい。等間隔参照相互では、

(c) ICCG(1,3), 259回	S-3800	M-660K
原形	18.426(1.00)	315.40(1.00)
リストベクトル 間接参照	1.755(10.49)	367.21(1/1.16)
リストベクトル 等間隔参照	1.323(13.92)	330.25(1/1.04)
改良	1.334(13.81)	434.12(1/1.05)

S-3800、M-660Kともリストベクトルを作る方法とプログラムに組み込む方法に差がなかった。したがって、リストベクトルを作るメリットはなく、プログラムを単純にするためには積極的にリストベクトルの使用をやめるべきである。

このように、リストベクトルを使わずに超平面法が実現できた。その結果、ベクトルパイプライン方式のスーパーコンピュータS-3800では、13倍から18倍の高速化が達成できる。これはリストベクトルを用いた間接的なメモリ参照に比べても、30%から40%の高速化であり、効果は著しい。一方、汎用コンピュータでは、一般的な超平面法の実現方法に比べると10%程度オーバーヘッドが減少して、超平面法のオーバーヘッドを5%から20%程度にできる。特に、ICCG(1,2)とICCG(1,3)ではオーバーヘッドが10%以下なので充分実用にたえる。

2次元矩形領域を差分法で離散化して得られた規則的対称疎行列に対しては、リストベクトルを使わずに超平面法が実現できた。これは問題の特殊性を利用したためで、一般の対称疎行列すべてに適用できるわけではない。しかし、2次元矩形領域を差分法で離散化して得られた行列だけでも、充分に利用価値はある。最終的にできあがったプログラムは、汎用コンピュータでは超平面法によるオーバーヘッドがあるものの、一般的なリストベクトルを使った超平面法に比べると、スーパーコンピュータと汎用コンピュータの双方で高速に実行できる。そのため、スーパーコンピュータの高速性をICCG法とMICCG法の収束を一切変化させることなく活用できる。

7. 結論

熱や電磁気といった拡散方程式で記述される物理現象を数値シミュレーションによって解こうとするとき、離散化して得られた大規模な対称疎行列をコンピュータを用いて高速に解く必要がある。本研究では、2次元の矩形領域における拡散方程式を差分法を用いて離散化し、得られた対称正定値行列、かつM-行列である規則的疎行列を係数にもつ連立一次方程式の数値解法を各種コンピュータ上で高速に実行するうえでの問題点を検証し、高精度、安定、かつ汎用的で、高速なプログラムを作成した。対象を2次元矩形領域に限定しているが、離散化を行う場の物理定数を大きく変化させることによって、得られた行列の固有値分布や条件数は広いバラエティをもち、ここでの結果は一般的な問題に広く適用できる。

このような問題に対する数値解法は、帯構造をもった連立一次方程式に対する直接解法と、規則的疎行列に対する反復解法に大別される。帯行列に対する直接解法としては、帯行列に対するガウスの消去法、Martin-Wilkinsonの特殊ガウス、対称帯ガウスをとりあげた。規則的疎行列に対する反復解法としては、共役勾配法(Conjugate Gradient Method)と、Meijerink流の不完全コレスキー分解を用いた前処理を施してからCG法を適用するICCG法(Incomplete Cholesky Conjugate Gradient Method)、ICCG法の前処理にGustafssonが提唱したような変更を施すMICCG法(Modified Incomplete Cholesky Conjugate Gradient Method)をとりあげた。これらを実用的なコンピュータプログラムとして実現するには、精度、安定性、必要なメモリ容量、高速性などに検討と配慮が必要である。

直接解法である帯行列に対するガウスの消去法、Martin-Wilkinsonの特殊ガウス、対称帯ガウスなどは、昔からアルゴリズムが知られており、小規模問題用のプログラムであれば容易に作成できる。ところが、限られた計算環境でより大規模な問題を解きたいとか、要素並列方式のベクトルパイプラインをもったスーパーコンピュータを使ってより高速に実行したいといった場合には、コンピュータに合わせた見直しが必要となる。

特に、3重ループの一番外側のループに対するアンローリングである2段同時が、演算量に対してロード･ストアなどのデータの移動が少ないために高速化が図れた。大規模な問題では、メモリ参照を効果的に行なうMartin-Wilkinsonの特殊ガウスが高速であり、同一の係数行列をもつ複数の方程式を解くとCPU時間に大きな差が生じる。正定値性と対称性を利用してピボッティングをせず、消去範囲も1/4で済ます対称帯ガウスは、スーパーコンピュータでの加速率は低いものの、演算量とメモリ参照量が少ないため、帯ガウス、Martin-Wilkinsonの特殊ガウスに比べると早く解が得られた。しかも拡散方程式を差分法で離散化して得られた対称正定値行列に対しては、ピボッティングの有無によって精度が変化しなかった。これらのプログラムの適用範囲を実用的な大きさの問題に限定すると、

多数のベクトルレジスタをもつスーパーコンピュータではレジスタ上にデータが常駐化できれば、一層の高速化を図ることができる。それには2重以上のループで不変なデータを認識する必要があり、コンパイラに対する指示文を利用してもうまくいかないことがある。このような場合にはアセンブラを使えば対応可能だが、そうすると汎用性を失うので、高速化はFORTRANで記述できる範囲にとどめた。その結果、これらのプログラムはほとんどどこでも実行可能で、それなりの高速性が達成できる。

規則的疎行列に対する反復解法は共役勾配法(Conjugate Gradient Method)を基本反復式とした。CG法は理論上は係数行列の元数以下の反復で収束することが保証されているが、コンピュータで実行すると誤差のために収束しないこともある。精度と安定性を調べるために、解こうとしている連立一次方程式の右辺または解を係数行列の固有ベクトルで展開する方法を提案した。この方法は、(1)4倍精度演算などの高精度演算が適用できるため、倍精度演算ではわからなかった現象がみえてくること、(2)右辺または解に対する依存性が調べられるため実用的であること、(3)前処理付きの共役勾配法(Preconditioned Conjugate Gradient Method)法の前処理に対する評価にも利用できるという特徴がある。理論的考察と数値実験によって、(1)CG法は低次モードに対する収束が悪いこと(倍精度演算のとき、固有値が10^{-3}以下の部分)、(2)密集固有値によって反復が減少するのは、固有値の差が10^{-10}程度以下であること、(3)Meijerink流の不完全コレスキー分解を用いた前処理、ICCG法の前処理に対するGustafssonが提唱したような変更はCG法の前処理として有効であることなどが、従来いわれていたよりも一段と深く明らかになった。

また、ICCG法やMICCG法の場合の条件数の推定方法を提案した。この方法を用いて、ICCG法やMICCG法で方程式を解くのと同時に条件数を推定するのなら、付加的に必要なのはわづかの作業領域(元数の約2倍)と、方程式を解くのに必要なCPU時間のおよそ3倍以内である。

ICCG法やMICCG法の前処理である不完全コレスキー分解は、数学的には安定で収束に必要な反復回数を大幅に減少させるが、ベクトルパイプライン方式のスーパーコンピュータでは高速に実行できないことが問題になっている。特定のスーパーコンピュータに高速な前処理も提案されているが、精度、安定性、汎用性を考えると採用しがたい。そして不完全コレスキー分解の高速化手法として超平面法が提案･利用されているが、間接的なメモリ参照になるためにベクトルパイプライン方式のスーパーコンピュータでもオーバーヘッドが大きく、汎用的とは言いがたかった。そこで、不完全コレスキー分解を作用させる部分と超平面法を検討し、ここで対象としているような2次元の矩形領域を差分法で離散化して得られた行列の場合には、間接的なメモリ参照をせずに高速化できることを明らかにした。このようにして作られたプログラムは、スーパーコンピュータで

は今までの超平面法よりも著しく速く、汎用コンピュータやワークステーションなどでは今までの超平面法よりもオーバーヘッドが少ないという特徴をもつ。しかも数学的な収束特性、精度、安定性に影響を与えず、小さいオーバーヘッドで、スーパーコンピュータに対して大幅な高速化ができるため、汎用的といえよう。

2次元の矩形領域における拡散方程式を差分法で離散化して得られた連立一次方程式の数値解法の検討を行なった。帯行列に対するガウスの消去法、Martin-Wilkinsonの特殊ガウス、対称帯ガウス、ICCG法、MICCG法など、問題の大きさと目的に合わせた高速なプログラムが作成できた。これらは、精度がよく、安定で、しかも汎用的に利用できる。また、CG法の収束特性の解析に用いた、固有ベクトルによって解または右辺を展開する方法は強力な道具建てであり、前処理の優劣の比較、PCG法などの性能比較にも活用できる。

付録に帯行列に対するガウスの消去法、Martin-Wilkinsonの特殊ガウス、対称帯ガウス、ICCG法、MICCG法などのFORTRANソースプログラムを載せる。

謝辞

本研究をすすめるにあたり、神奈川大学教授村田健郎先生(元図書館情報大学教授)には随所でご指導して頂きました。また、図表などの作成は日立製作所ソフトウェア開発本部勤務の妻里美に、休日返上で協力してもらいました。深く感謝致します。

最後に、大規模行列計算というメモリ容量やCPU時間を大量に消費する分野で、このような実験を主体とした研究を可能にしてくれた、図書館情報大学総合情報処理センター(旧図書館情報システム開発センター)の設備とスタッフの協力にも厚く御礼申しあげます。

参考文献

[1] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J. J., Croz, J. Du, Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov , S., Sorensen, D., *LAPACK Users' Guide*, SIAM, Philadelphia, Pennsylvania, 1992, 235p.

[2] ASHCRAFT, C. CLEVELAND, GRIMES, ROGER G., *ON VECTORIZING INCOMPLETE FACTORIZATION AND SSOR PRECONDITIONERS*, SIAM J. SCI. STAT. COMPUT., Vol. 9, No. 1, pp. 122-151(1988)

[3] Axelsson, Owe. and Lindskog, Gunhild, *On the Eigenvalue Distribution of a Class of Preconditioning Methods*, Numer. Math., Vol. 48, p. 479-498 (1986)

[4] Axelsson, Owe. and Lindskog, Gunhild, *On the Rate of Convergence of the Preconditioned Conjugate Gradient Method*, Numer. Math., Vol. 48, p. 499-523(1986)

[5] CHAN, TONY F. and ELMAN C., *FOURIER ANALYSIS OF ITERATIVE METHODS FOR ELLIPTIC PROBLEMS*, SIAM Review, Vol. 31, No. 1, pp.20-49(1989)

[6] CHAN, TONY F., *FOURIER ANALYSIS OF RELAXED INCOMPLETE FACTORIZATION PRECONDITIONERS*, SIAM J. SCI. STAT. COMPUT., Vol. 12, No. 3, pp. 668-680(1991)

[7] CONCUS, P., GOLUB, G. H. and MEURANT, G., *BLOCK PRECONDITIONING FOR THE CONJUGATE GRADIENT METHOD*, SIAM J. SCI. STAT. COMPUT., Vol. 6, No. 1, pp. 220-252(1985)

[8] Dongarra, J. J., Moler, C. B., Bunch J. R. and Stewart, G. W., *LINPACK Users' Guide*, SIAM, Philadelphia, Pennsylvania, 1979.

[9] DONGARRA, J. J., GUSTAVSON, F. G. and KARP, A., *IMPLEMENTING LINEAR ALGEBRA ALGORITHMS FOR DENSE MATRICES ON A VECTOR PIPELINE MACHINE*, SIAM Review, Vol. 26, No. 1, pp.91-112(1984)

[10] Dongarra, J. J., Croz, Du, Duff, I. and Hammarling, S., *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Softw., Vol. 16, No. 1, p. 1-17(1990)

[11] Dongarra, J. J., Duff, Iain S., Sorensen, Danny C. and van der Vorst, Henk A., *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, Pennsylvania, 1991, 256p.

[12] Forsythe, G. E, Malcolm, M. A. and Moler, C. E., *Computer Methods for Mathematical Computations,* Prentice-Hall, Englewood Cliffs, New Jersey, 1977(森正武訳, 計算機のための数値計算法, 科学技術出版社, 東京, 1978, 285p)

[13] 藤野清次, 長谷川秀彦. ベクトル計算機におけるFill-in付き(M)ICCG法の性能評価. 日本応用数理学会論文誌. Vol. 2, No. 2, p. 105–118(1992)

[14] GOLUB, GENE H. and O'LEARY, DIANNE P., *SOME HISTORY OF CONJUGATE GRADIENT AND LANCZOS ALGORITHMS* : 1948-1976, SIAM Review, Vol. 31, No. 1, pp. 50-102(1989)

[15] Greenbaum, A., *Behavior of Slightly Perturbed Lanczos and Conjugate-Gradient Recurrences.* LINEAR ALGEBRA AND ITS APPLICATIONS, 113, pp. 7-63(1989)

[16] Greenbaum, Anne, Li, Congming and Chao, Han Zheng, *Comparison of Linear Systems Solvers Applied to Diffusion-Type Finite Element Equations,* Numer. Math., Vol. 56, p. 529-546(1989)

[17] GREENBAUM, A. and STRAKOS, Z., *PREDICTING THE BEHAVIOR OF FINITE PRECISION LANCZOS AND CONJUGATE GRADIENT COMPUTATIONS.* SIAM J. MATRIX ANAL. APPL., Vol. 13, No. 1, pp. 121-137(1992)

[18] Gustafsson, Ivar, *A class of first order factorization method,* BIT, Vol. 18, p. 142-156(1978)

[19] 長谷川秀彦, 北上純一, 村田健郎. 残差多項式によるPCG法の収束特性の解析. Iterative Methods for Nonsymmetric Systems and Their Applications (非対称行列系の反復解法とその応用) 名取亮, 野寺隆編. 神奈川, 慶応義塾大学, 1987, p.19–28(Advances in Numerical Methods for Large Sparse Sets of Linear Equations, No.3)

[20] 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)

[21] 長谷川秀彦. 帯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. 45-59(1987).

[22] 長谷川秀彦. 帯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. 6, No. 2, p. 89-111(1987).

[23] 長谷川秀彦. 帯行列を係数とする連立一次方程式の解法(III). 図書館情報大学研究報告. Vol. 7, No. 1, p. 61-73(1988).

[24] 長谷川秀彦. 帯行列に対する直接解法の高速化. 情報処理学会論文誌. Vol. 30, No. 4, p. 402–410(1989)

[25] 長谷川秀彦, 春日里美. 対称正定値行列Aと不完全コレスキー分解 $U^T U$ の固有値分布. 図書館情報大学研究報告. Vol. 8, No. 1, p. 111-131(1989)

[26] 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, p.49-63(1990)

[27] 長谷川秀彦. 対称正定値疎行列に対するPCG法のプログラム(Ⅰ). 図書館情報大学研究報告. Vol. 9, No. 2, p. 65-82(1990)

[28] 長谷川秀彦, 川端裕一. 連立一次方程式の高速解法について. 数値解析研究会資料 No. 99. 東京, 情報処理学会, 1990, 90–NA–35, p. 1–10 (情報処理学会研究報告, Vol. 90, No. 99)

[29] 長谷川秀彦. 対称正定値疎行列の条件数概算法. 日本応用数理学会論文誌. Vol. 1, No. 2, p. 169–176(1991)

[30] 長谷川秀彦. 大規模行列計算ソフトウェアについて. 東京大学大型計算 機センターニュース. Vol. 24, No. 6, p. 121–148(1992)

[31] 長谷川秀彦. 共役勾配法の右辺依存性. ParallelProcessing for Scientific Computing (科学計算における並列処理) 名取亮, 野寺隆編. 神奈川, 慶応義塾大学, 1993, p. 25–31 (Advances in Numerical Methods for Large Sparse Sets of Linear Equations, No. 9)

[32] Hestenes, Magunus R. and Stiefel, Eduard, *Methods of Cojugate Gradients for Solving Linear Systems*, J. Res. Nat. Bur. Standards, Vol. 49, No. 6, p. 409-436(1952)

[33] Higham, N., *Efficient Algorithms for Computing the Condition Number of a Tridiagonal Matrix*, SIAM J. SCI. STAT. COMPUT., Vol. 7, No. 1, pp. 150-165(1986)

[34] HIGHAM, NICHOLAS J., *FORTRAN Codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation*, ACM Trans. Math. Softw., Vol. 14, No. 4, p. 381-396(1988)

[35] Higham, Nicholas J., *Estimating the matrix p-norm*, Numer. Math., Vol. 62, p. 539-555(1992)

[36] KERSHAW, DAVID S., *The Incomplete Cholesky - Conjugate Gradient Method for the iterative Solution of Systems of Linear Equations*, Journal of Computational Physics, Vol. 26, p.43-65(1978)

[37] KIGHTLEY, J. R., THOMPSON, C. P., *ON THE PERFORMANCE OF SOME RAPID ELLIPTIC SOLVERS ON A VECTOR PROCESSOR*. SIAM J. SCI. STAT. COMPUT. Vol. 8, No. 5, pp. 701-715(1987)

[38] 香田健二. 新スーパーコンS-3800/480の使い方-VOS3編-. 東京大学大型計算機センターニュース. Vol. 25, No. 1, p. 57-76(1993)

[39] Lamport Leslie, *The Parallel Execution of DO Loops*, Communications of the ACM, Vol. 17, No. 2, p. 83-93(1974)

[40] Manteuffel, T. A., *An Incomplete Factorization Technique for Positive Definite Linear Systems*. MATHEMATICS OF COMPUTATION. Vol. 34, No. 150, p.473-497(1980)

[41] Meijerink, J. A. and van der Vorst, H. A., *An Iterative Solution Method for Linear Systems of which the CoefficientMatrix is aSymmetric M-matrix*, MATHEMATICS OF COMPUTATION, Vol. 31, No. 137, p. 148-162(1977)

[42] Meijerink, J. A. and van der Vorst, H. A., *Guidelines for the Usage of Incomplete Decompositions in Solving Sets of Linear Equationsas They Occur in Practical Problems*, Journal of Computational Physics, Vol. 44, p. 134-155(1981)

[43] 森正武. ICCG法と残差多項式. 京都, 京都大学数理解析研究所, 1986, 数理解析研究所講究録585, p. 113-129

[44] 村田健郎. 科学技術計算と高速算法. Computer Today. No.2, p.51–61(1984)

[45] 村田健郎, 小国力, 唐木幸比古. スーパーコンピュータ. 東京, 丸善, 1985, 304p.

[46] 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.

[47] 村田健郎. 前処理つき共役勾配法·共役残差法. 情報処理. Vol. 27, No. 5, p.498–507(1986)

[48] 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)

[49] 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, p19-36.

[50] 村田健郎, 春日里美. $Ax=\lambda Mx$型問題に対する二分法·同時逆反復法. 図書館情報大学研究報告. Vol. 7, No. 2, p. 99-131(1988)

[51] 村田健郎, 名取亮, 唐木幸比古. 大型数値シミュレーション. 東京, 岩波書店, 1990, 293p.

[52] 村田健郎, 三好俊郎, 小国力, Dongarra, J. J., 長谷川秀彦. 行列計算ソフトウェア. 東京, 丸善, 1991, 391p. (プログラム·フロッピーディスク付き)

[53] 長島重夫, 田中義一. スーパーコンピュータ. 東京, オーム社, 1992, 204p.

[54] 名取亮. 線形計算. 東京, 朝倉書店, 1993, 141p.

[55] Natori, M. and Tsukamoto, A., *A Fast method for Estimating theCondition Number of a Matrix*, J. Information Processing, Vol. 6, No. 3, p. 138-140 (1983)

[56] 日本物理学会編. スーパーコンピュータ. 東京, 培風館, 1985, 278p.

[57] 二宮市三. スーパーコンピュータと数学ライブラリ. 情報処理. Vol. 27, No. 11, p. 1235-1241(1986)

[58] 野寺隆. 大規模疎行列に対するPCG法. 神奈川, 慶応義塾大学, 1983, 92p.(SEMINAR ON MATHEMATICAL SCIENCES, No. 7, 1983)

[59] Notay, Yvan, *On the convergence rate of the conjugate gradients inpresence of rounding errors*. Numer. Math., Vol. 65, p. 301-317(1993)

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

[61] Parlett, B. N., *The Symmetric Eigenvalue Problem.*, Prentice-Hall, Englewood Cliffs, New Jersey, 1980, 348p.

[62] SAAD, YOUCEF, *PRACTICAL USE OF POLYNOMIAL PRECONDITIONINGS FOR THE CONJUGATE GRADIENT*, SIAM J. SCI. STAT. COMPUT., Vol. 6, No. 4, pp. 865-881(1985)

[63] van der Sluis, A. and van der Vorst, H. A., *The Rate of Convergence of Conjugate Gradients*. Numer. Math., Vol. 48, p. 543-560(1986)

[64] Stiefel, E. L., *Kernel Polynomials in linear algebra and their numerical applications in further contributions to the solution of simultaneous Linear Equations and the Determination of eigenvalues*, NBS Applied Mathematics, Ser. 49, p. 1-22(1958)

[65] Strakos, Z., *On the Real Convergence Rate of the Conjugate Gradient Method.* LINEAR ALGEBRA AND ITS APPLICATIONS. 154-156, pp. 535-549(1991)

[66] 田中義一, 岩澤京子. ベクトル計算機のためのコンパイル技術. 情報処理. Vol. 31, No. 6, p. 736-743(1990)

[67] 戸川隼人. 新しい応用の数学17 共役勾配法. 東京, 教育出版, 1977, 157p.

[68] 後保範. ベクトル計算機向きICCG法. 京都, 京都大学数理解析研究所, 1984, 数理解析研究所講究録514, p. 110-134

[69] Varga, R. S., 渋谷政昭ほか訳. 計算機による大型行列の反復解法. 東京, サイエンス社, 1972, 290p.

[70] VAN DER VORST, HENK A., *A VECTORIZABEL VARIANT OF SOME ICCG METHODS*, SIAM J. SCI. STAT. COMPUT., Vol. 3, No. 3, pp. 350-356(1982)

[71] VAN DER VORST, HENK A., *High Performance Preconditionning*, SIAM J. SCI. STAT. COMPUT., Vol. 10, No. 6, pp. 1174-1185(1989)

[72] Wilkinson, J. H., Reinsch, C. R., Linear Algebra:*Contribution I/4* by R. S. Martin and J. H. Wilkinson. Chief editor: F. L. Bauer. Berlin, Springer-Verlag, 1971, p. 50-56. (Handbook for Automatic Computation, v. 2)

[73] Wilkinson, J. H., Reinsch, C. R., Linear Algebra:*Contribution I/6* by R. S. Martin and J. H. Wilkinson. Chief editor: F. L. Bauer. Berlin, Springer-Verlag, 1971, p. 70-92. (Handbook for Automatic Computation, v. 2)

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

[75] Wilkinson, J. H., Reinsch, C. R., Linear Algebra:*Contribution II/1* by H. Rutishauser. Chief editor: F. L. Bauer. Berlin, Springer-Verlag, 1971, p. 202-211. (Handbook for Automatic Computation, v. 2)

[76] Windisch, Gunther, *M-matrices in Numerical Analysis,* Leipzig, Teubner Verlagsgesellschaft, 1989, 140p.

[77] Zlatev, Z., Wasniewski, J. and Schaumburg, K., *Condition Number Estimators in a Sparse Matrix Software,* SIAM J. SCI. STAT. COMPUT., Vol. 7, No. 4, pp. 1175-1189(1986)

（１）帯ガウス（帯行列に対するガウスの消去法）

```
      *
    1      SUBROUTINE BGLU1( A, N, ML, MU, EPS, WK, IP, IER )
      *
      *       BGLU1
      *             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
      *
      *             SOLVES SIMULTANEOUS LINEAR EQUATIONS
      *             BY GAUSSIAN ELIMINATION METHOD FOR GENERAL BAND MATRIX.
      *
      *       INPUT - -
      *           A(-ML:MU+ML,N)
      *                     R *8  : 2-DIM. ARRAY CONTAINING REAL BAND MATRIX.
      *           N         I *4  : ORDER OF MATRIX.
      *           ML        I *4  : LOWER BAND WIDTH.
      *           MU        I *4  : UPPER BAND WIDTH.
      *           EPS       R *8  : PARAMETER TO CHECK SINGULARITY OF THE
      *                             MATRIX. ( STANDARD VALUE 3.52D-15 )
      *       OUTPUT - -
      *           A(-ML:MU+ML,N)
      *                           : RESULT OF GAUSSIAN ELIMINATION.
      *           IP(N)     I *4  : PIVOT NUMBER.
      *           IER       I *4  : = 0,  FOR NORMAL EXECUTION.
      *                             = 1,  FOR SINGULAR MATRIX.
      *                             = 3,  FOR INVALID ARGUEMENT.
      *       WORKING  -
      *           WK(N)     R *8  : 1-DIM. ARRAY.
      *
    2      IMPLICIT REAL*8 (A-H,O-Z)
    3      DIMENSION A(-ML:MU+ML,*), IP(*), WK(*)
      *           LEFT HAND SIDE
    4      IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
    5      IF( ( N.LE.0 ).OR.( ML.LE.0 ).OR.( MU.LE.0 ).OR.
     +     ( ML.GE.N ).OR.( MU.GE.N ) )  THEN
    6        IER = 3
    7        WRITE(*,*) ' (SUBR. BGLU1)  INVALID ARGUMENT.  ML, MU, N =',
     +             ML, MU, N
    8        RETURN
    9      END IF
      *
   10      IER = 0
   11      DO 100 K = 1, N
      *           FIND MAXIMUM ELEMENT IN THE K-TH COLUMN.
   12        AMAX = ABS(A(0,K))
   13        IPK = K
   14        DO 110 I = K+1, MIN(K+ML,N)
   15          AIK = ABS(A(K-I,I))
   16          IF( AIK.GT.AMAX )  THEN
   17             IPK = I
   18             AMAX = AIK
   19          END IF
   20  110   CONTINUE
   21        IP(K) = IPK
      *
   22        IF( AMAX.GT.EPS )  THEN
   23           IF( IPK.NE.K )  THEN
   24             DO 120 J = K, MIN(K+MU+ML,N)
   25               W = A(J-IPK,IPK)
   26               A(J-IPK,IPK) = A(J-K,K)
   27               A(J-K,K) = W
   28  120          CONTINUE
   29           END IF
```

```
   *
   C*         DO 125 J = K+1, MIN(K+MU+ML,N)
   C*125        WK(J) = A(J-K,K)
   *             COMPUTE ALFA AND PERFORM GAUSSIAN ELIMINATION.
 30            DO 130 I = K+1, MIN(K+ML,N)
 31              A(K-I,I) = -A(K-I,I)/A(0,K)
   C*            T = A(K-I,I)
   *VOPTION LOOP(512)
 32              DO 140 J = K+1, MIN(K+MU+ML,N)
 33   140          A(J-I,I) = A(J-I,I)+A(K-I,I)*A(J-K,K)
   C*140          A(J-I,I) = A(J-I,I)+T*WK(J)
 34   130      CONTINUE
   *              MATRIX IS SINGULAR.
 35         ELSE
 36            IER = 1
 37            IP(K) = K
 38            DO 150 I = K+1, MIN(K+ML,N)
 39   150        A(K-I,I) = 0.0D0
 40            WRITE(*,*)  '  (SUBR. BGLU1)  MATRIX IS SINGULAR AT K =', K
 41            RETURN
 42         END IF
 43   100 CONTINUE
 44       RETURN
 45       END
```

（２）帯ガウス　右辺(帯行列に対するガウスの消去法)

```
   *
  1      SUBROUTINE BGSLV1( A, N, ML, MU, B, IP )
   *
   *       BGSLV1
   *              COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
   *
   *              SOLVES SIMULTANEOUS LINEAR EQUATIONS
   *              BY GAUSSIAN ELIMINATION METHOD FOR GENERAL BAND MATRIX.
   *
   *       INPUT - -
   *            A(-ML:MU+ML,N)
   *                    R *8  : RESULT OF GAUSSIAN ELIMINATION.
   *            N       I *4  : ORDER OF MATRIX.
   *            ML      I *4  : LOWER BAND WIDTH.
   *            MU      I *4  : UPPER BAND WIDTH.
   *            B(N)    R *8  : 1-DIM. ARRAY CONTAINING THE RIGHT HAND
   *                            SIDE VECTOR.
   *            IP(N)   I *4  : PIVOT NUMBER.
   *       OUTPUT - -
   *            B(N)          : SOLUTION.
   *
  2      IMPLICIT REAL*8 (A-H,O-Z)
  3      DIMENSION A(-ML:MU+ML,*), B(*), IP(*)
   *             FORWARD ELIMINATION PROCESS
  4      DO 100 K = 1, N
  5        IPK = IP(K)
  6        IF( IPK.NE.K )  THEN
  7           W =B(IPK)
  8           B(IPK) = B(K)
  9           B(K) = W
 10        END IF
   *             GAUSSIAN ELIMINATION
   C*       T = B(K)
 11         DO 110 I = K+1, MIN(K+ML,N)
 12   110     B(I) = B(I)+A(K-I,I)*B(K)
   C*110     B(I) = B(I)+A(K-I,I)*T
 13   100 CONTINUE
   *             BACKWARD SUBSTITUTION PROCESS
 14       DO 200 K = N, 1, -1
 15         S = -B(K)
 16         DO 210 J = K+1, MIN(K+MU+ML,N)
 17   210     S = S+A(J-K,K)*B(J)
 18         B(K) = -S/A(0,K)
 19   200 CONTINUE
 20       RETURN
 21       END
```

（３）帯ガウス　２段同時

```
      *
    1       SUBROUTINE BGLU2( A, N, ML, MU, EPS, WK1, WK2, IP, IER )
      *
      *       BGLU2
      *             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
      *
      *             SOLVES SIMULTANEOUS LINEAR EQUATIONS
      *             BY GAUSSIAN ELIMINATION METHOD FOR GENERAL BAND MATRIX.
      *
      *       INPUT - -
      *            A(-ML-1:MU+ML+1,N+1)
      *                    R *8  : 2-DIM. ARRAY CONTAINING REAL BAND MATRIX.
      *            N       I *4  : ORDER OF MATRIX.
      *            ML      I *4  : LOWER BAND WIDTH.
      *            MU      I *4  : UPPER BAND WIDTH.
      *            EPS     R *8  : PARAMETER TO CHECK SINGULARITY OF THE
      *                            MATRIX. ( STANDARD VALUE 3.52D-15 )
      *       OUTPUT - -
      *            A(-ML-1:MU+ML+1,N+1)
      *                          : RESULT OF GAUSSIAN ELIMINATION.
      *            IP(N+1)  I *4 : PIVOT NUMBER.
      *            IER      I *4 : = 0,  FOR NORMAL EXECUTION.
      *                            = 1,  FOR SINGULAR MATRIX.
      *                            = 3,  FOR INVALID ARGUEMENT.
      *       WORKING  -
      *            WK1(N), WK2(N)
      *                    R *8  : 1-DIM. ARRAY.
      *
    2       IMPLICIT REAL*8 (A-H,O-Z)
    3       DIMENSION A(-ML-1:MU+ML+1,*), IP(*), WK1(*), WK2(*)
      *            LEFT HAND SIDE
    4       IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
    5       IF( ( N.LE.0 ).OR.( ML.LE.0 ).OR.( MU.LE.0 ).OR.
           +    ( ML.GE.N ).OR.( MU.GE.N ) )  THEN
    6          IER = 3
    7          WRITE(*,*) '  (SUBR. BGLU2)  INVALID ARGUMENT.  ML, MU, N =',
           +             ML, MU, N
    8          RETURN
    9       END IF
      *
   10       IER = 0
   11       DO 10 I = 1, N
   12         A(-ML-1,I) = 0.0D0
   13    10   A(MU+ML+1,I) = 0.0D0
   14       IF( MOD(N,2).NE.0 )  THEN
   15          DO 20 J = -ML-1, MU+ML+1
   16    20      A(J,N+1) = 0.0D0
   17          A(0,N+1) = 1.0D0
   18       END IF
      *
   19       DO 100 K = 1, N, 2
      *
   20         K1 = K+1
   21         IMAX = MIN(K+ML,N)
   22         JMAX = MIN(K+MU+ML,N)
   23         IMAX1 = MIN(K1+ML,N)
   24         JMAX1 = MIN(K1+MU+ML,N)
      *             FIND MAXIMUM ELEMENT IN THE K-TH COLUMN.
   25         AMAX = ABS(A(0,K))
   26         IPK = K
   27         DO 110 I = K+1, IMAX
   28           AIK = ABS(A(K-I,I))
   29           IF( AIK.GT.AMAX )  THEN
   30              IPK = I
   31              AMAX = AIK
   32           END IF
   33    110    CONTINUE
```

```
      IP(K) = IPK
*           EXCHANGE FOR K-TH ELIMINATION.
      IF( AMAX.GT.EPS ) THEN
        IF( IPK.NE.K ) THEN
          DO 120 J = K, JMAX
            W = A(J-IPK,IPK)
            A(J-IPK,IPK) = A(J-K,K)
            A(J-K,K) = W
  120     CONTINUE
        END IF
*           COMPUTE ALFA
        T = A(0,K)
        DO 130 I = K+1, IMAX1
  130     A(K-I,I) = -A(K-I,I)/T
        T = A(1,K)
        DO 140 I = K+1, IMAX
  140     A(K1-I,I) = A(K1-I,I)+A(K-I,I)*T
*           MATRIX IS SINGULAR.
      ELSE
        IER = 1
        IP(K) = K
        DO 150 I = K+1, IMAX1
  150     A(K-I,I) = 0.0D0
        WRITE(*,*)  '  (SUBR. BGLU2)  MATRIX IS SINGULAR AT K =', K
        RETURN
      END IF
*           FIND MAXIMUM ELEMENT IN THE K+1'TH COLUMN.
      AMAX1 = ABS(A(0,K1))
      IPK1 = K1
      DO 200 I = K1+1, IMAX1
        AIK = ABS(A(K1-I,I))
        IF( AIK.GT.AMAX1 ) THEN
          IPK1 = I
          AMAX1 = AIK
        END IF
  200 CONTINUE
      IP(K1) = IPK1
*           EXCHANGE FOR K+1'TH ELIMINATION.
      IF( AMAX1.GT.EPS ) THEN
        IF( IPK1.NE.K1 ) THEN
          DO 210 J = K, JMAX1
            W = A(J-IPK1,IPK1)
            A(J-IPK1,IPK1) = A(J-K1,K1)
            A(J-K1,K1) = W
  210     CONTINUE
        END IF
*           COMPUTE ALFA
        T = A(0,K1)
        DO 220 I = K1+1, IMAX1
  220     A(K1-I,I) = -A(K1-I,I)/T
*           MATRIX IS SINGULAR.
      ELSE
        IER = 1
        IP(K1) = K1
        DO 230 I = K1+1, IMAX1
  230     A(K1-I,I) = 0.0D0
        WRITE(*,*)  '  (SUBR. BGLU2)  MATRIX IS SINGULAR AT K =', K1
        RETURN
      END IF
*
      T = A(-1,K1)
      DO 240 J = K1+1, JMAX1
        WK1(J) = A(J-K,K)
        A(J-K1,K1) = A(J-K1,K1)+T*WK1(J)
  240   WK2(J) = A(J-K1,K1)
*           GAUSSIAN ELIMINATION FOR K-TH AND K+1'TH PROCESS.
      DO 300 I = K1+1, IMAX1
        T = A(K-I,I)
        T1 = A(K1-I,I)
*VOPTION LOOP(512)
        DO 310 J = K1+1, JMAX1
          A(J-I,I) = A(J-I,I)+T*WK1(J)+T1*WK2(J)
  310   CONTINUE
  300 CONTINUE
  100 CONTINUE
      RETURN
      END
```

（４）帯ガウス　２段２行同時

```
*
      SUBROUTINE BGLU4( A, N, ML, MU, EPS, WK1, WK2, IP, IER )
*
*       BGLU4
*             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
*
*             SOLVES SIMULTANEOUS LINEAR EQUATIONS
*             BY GAUSSIAN ELIMINATION METHOD FOR GENERAL BAND MATRIX.
*
*       INPUT - -
*           A(-ML-1:MU+ML+1,N+1)
*                   R *8  : 2-DIM. ARRAY CONTAINING REAL BAND MATRIX.
*           N       I *4  : ORDER OF MATRIX.
*           ML      I *4  : LOWER BAND WIDTH.
*           MU      I *4  : UPPER BAND WIDTH.
*           EPS     R *8  : PARAMETER TO CHECK SINGULARITY OF THE
*                           MATRIX. ( STANDARD VALUE 3.52D-15 )
*       OUTPUT - -
*           A(-ML-1:MU+ML+1,N+1)
*                         : RESULT OF GAUSSIAN ELIMINATION.
*           IP(N+1)  I *4 : PIVOT NUMBER.
*           IER      I *4 : = 0,  FOR NORMAL EXECUTION.
*                           = 1,  FOR SINGULAR MATRIX.
*                           = 3,  FOR INVALID ARGUEMENT.
*       WORKING  -
*           WK1(N), WK2(N)
*                   R *8  : 1-DIM. ARRAY.
*
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(-ML-1:MU+ML+1,*), IP(*), WK1(*), WK2(*)
*           LEFT HAND SIDE
      IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
      IF( ( N.LE.0 ).OR.( ML.LE.0 ).OR.( MU.LE.0 ).OR.
     +    ( ML.GE.N ).OR.( MU.GE.N ) )  THEN
         IER = 3
         WRITE(*,*) '  (SUBR. BGLU4)  INVALID ARGUMENT.  ML, MU, N =',
     +             ML, MU, N
         RETURN
      END IF
*
      IER = 0
      DO 10 I = 1, N
        A(-ML-1,I) = 0.0D0
   10   A(MU+ML+1,I) = 0.0D0
      IF( MOD(N,2).NE.0 )  THEN
         DO 20 J = -ML-1, MU+ML+1
   20      A(J,N+1) = 0.0D0
         A(0,N+1) = 1.0D0
      END IF
*
      DO 100 K = 1, N, 2
*
        K1 = K+1
        IMAX = MIN(K+ML,N)
        JMAX = MIN(K+MU+ML,N)
        IMAX1 = MIN(K1+ML,N)
        JMAX1 = MIN(K1+MU+ML,N)
*             FIND MAXIMUM ELEMENT IN THE K-TH COLUMN.
        AMAX = ABS(A(0,K))
        IPK = K
        DO 110 I = K+1, IMAX

          AIK = ABS(A(K-I,I))
          IF( AIK.GT.AMAX )  THEN
             IPK = I
             AMAX = AIK
          END IF
  110   CONTINUE
        IP(K) = IPK
```

```
*            EXCHANGE FOR K-TH ELIMINATION.
      IF( AMAX.GT.EPS )  THEN
         IF( IPK.NE.K )  THEN
            DO 120 J = K, JMAX
              W = A(J-IPK,IPK)
              A(J-IPK,IPK) = A(J-K,K)
              A(J-K,K) = W
120         CONTINUE
         END IF
*           COMPUTE ALFA
         T = A(0,K)
         DO 130 I = K+1, IMAX1
           A(K-I,I) = -A(K-I,I)/T
130      T = A(1,K)
         DO 140 I = K+1, IMAX
           A(K1-I,I) = A(K1-I,I)+A(K-I,I)*T
*           MATRIX IS SINGULAR.
      ELSE
         IER = 1
         IP(K) = K
         DO 150 I = K+1, IMAX1
           A(K-I,I) = 0.0D0
150      WRITE(*,*)  '  (SUBR. BGLU4)  MATRIX IS SINGULAR AT K =', K
         RETURN
      END IF
*           FIND MAXIMUM ELEMENT IN THE K+1'TH COLUMN.
      AMAX1 = ABS(A(0,K1))
      IPK1 = K1
      DO 200 I = K1+1, IMAX1
        AIK = ABS(A(K1-I,I))
        IF( AIK.GT.AMAX1 )  THEN
           IPK1 = I
           AMAX1 = AIK
        END IF
200   CONTINUE
      IP(K1) = IPK1
*           EXCHANGE FOR K+1'TH ELIMINATION.
      IF( AMAX1.GT.EPS )  THEN
         IF( IPK1.NE.K1 )  THEN
            DO 210 J = K, JMAX1
              W = A(J-IPK1,IPK1)
              A(J-IPK1,IPK1) = A(J-K1,K1)
              A(J-K1,K1) = W
210         CONTINUE
         END IF
*           COMPUTE ALFA
         T = A(0,K1)
         DO 220 I = K1+1, IMAX1
           A(K1-I,I) = -A(K1-I,I)/T
220      MATRIX IS SINGULAR.
*
      ELSE
         IER = 1
         IP(K1) = K1
         DO 230 I = K1+1, IMAX1
           A(K1-I,I) = 0.0D0
230      WRITE(*,*)  '  (SUBR. BGLU4)  MATRIX IS SINGULAR AT K =', K1
      END IF
*
      T = A(-1,K1)
      DO 240 J = K1+1, JMAX1
        WK1(J) = A(J-K,K)
        A(J-K1,K1) = A(J-K1,K1)+T*WK1(J)
240     WK2(J) = A(J-K1,K1)
*           GAUSSIAN ELIMINATION 2 COL. FOR K-TH AND K+1'TH PROCESS.
      DO 300 I = K1+1, IMAX1, 2
        T = A(K-I,I)
        T1 = A(K1-I,I)
        U = A(K-I-1,I+1)
        U1 = A(K-I,I+1)
*VOPTION LOOP(512)
        DO 310 J = K1+1, JMAX1
          A(J-I,I)     = A(J-I,I)    +T*WK1(J)+T1*WK2(J)
          A(J-I-1,I+1) = A(J-I-1,I+1)+U*WK1(J)+U1*WK2(J)
310     CONTINUE
300   CONTINUE
100 CONTINUE
      RETURN
      END
```

（５）帯ガウス 右辺　２段同時と２段２行同時用

```
*
1       SUBROUTINE BGSLV4( A, N, ML, MU, B, IP )
*
*         BGSLV4
*               COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
*
*               SOLVES SIMULTANEOUS LINEAR EQUATIONS
*               BY GAUSSIAN ELIMINATION METHOD FOR GENERAL BAND MATRIX.
*
*         INPUT - -
*              A(-ML-1:MU+ML+1,N+1)
*                      R *8  : RESULT OF GAUSSIAN ELIMINATION.
*              N       I *4  : ORDER OF MATRIX.
*              ML      I *4  : LOWER BADN WIDTH.
*              MU      I *4  : UPPER BAND WIDTH.
*              B(N+1)  R *8  : 1-DIM. ARRAY CONTAINING THE RIGHT HAND
*                              SIDE VECTOR.
*              IP(N+1) I *4  : PIVOT NUMBER.
*         OUTPUT - -
*              B(N+1)        : SOLUTION.
*
2       IMPLICIT REAL*8 (A-H,O-Z)
3       DIMENSION A(-ML-1:MU+ML+1,*), B(*), IP(*)
*               FORWARD ELIMINATION PROCESS
4       DO 100 K = 1, N, 2
*               EXCHANGE FOR K-TH ELIMINATION.
5         IPK = IP(K)
6         IF( IPK.NE.K )  THEN
7           W = B(IPK)
8           B(IPK) = B(K)
9           B(K) = W
10        END IF
*               EXCHANGE FOR K+1'TH ELIMINATION.
11        K1 = K+1
12        IPK1 = IP(K1)
13        IF( IPK1.NE.K1 )  THEN
14          W = B(IPK1)
15          B(IPK1) = B(K1)
16          B(K1) = W
17        END IF
*
18        B(K1) = B(K1)+A(-1,K1)*B(K)
*               GAUSSIAN ELIMINATION FOR K-TH AND K+1'TH PROCESS.
19        T = B(K)
20        T1 = B(K1)
21        DO 110 I = K1+1, MIN(K1+ML,N)
22  110     B(I) = B(I)+A(K-I,I)*T+A(K1-I,I)*T1
23  100 CONTINUE
*               BACKWARD SUBSTITUTION PROCESS
24      IF( MOD(N,2).EQ.0 )  THEN
25        NEND = N
26      ELSE
27        NEND = N+1
28        B(N+1) = 0.0D0
29      END IF
30      DO 200 K = NEND, 1, -2
31        S1 = -B(K)
32        S0 = -B(K-1)
33        DO 210 J = K+1, MIN(K+MU+ML,N)
34          S1 = S1+A(J-K,K)*B(J)
35  210     S0 = S0+A(J-K+1,K-1)*B(J)
36        B(K) = -S1/A(0,K)
37        B(K-1) = (-S0-A(1,K-1)*B(K))/A(0,K-1)
38  200 CONTINUE
39      RETURN
40      END
```

(6) Martin-Wilkinsonの特殊ガウス

```
*
1       SUBROUTINE BHLU1( A, AC, N, ML, MU, EPS, WK, IP, IER )
*
*       BHLU1
*             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
*
*             SOLVES SIMULTANEOUS LINEAR BAND EQUATIONS
*             BY WILIKINSON'S GAUSSIAN ELIMINATION METHOD.
*
*       INPUT - -
*            A(0:ML+MU,N)
*                    R *8  : 2-DIM. ARRAY CONTAINING REAL BAND MATRIX.
*            N       I *4  : ORDER OF MATRIX.
*            ML      I *4  : LOWER BAND WIDTH.
*            MU      I *4  : UPPER BAND WIDTH.
*            EPS     R *8  : PARAMETER TO CHECK SINGULARITY OF THE
*                            MATRIX. ( STANDARD VALUE 3.52D-15 )
*       OUTPUT - -
*            A(0:ML+MU,N)   : UPPER TRIANGULAR MATRIX.
*            AC(ML,N) R *8  : 2-DIM. ARRAY CONTAINING ALFA.
*            IP(N)    I *4  : PIVOT NUMBER.
*            IER      I *4  : = 0,  FOR NORMAL EXECUTION.
*                             = 1,  FOR SINGULAR MATRIX.
*                             = 3,  FOR INVALID ARGUEMENT.
*       WORKING  -
*            WK(N)    R *8  : 1-DIM. ARRAY.
*
2       IMPLICIT REAL*8 (A-H,O-Z)
3       DIMENSION A(0:MU+ML,*), AC(ML,*), IP(*), WK(*)
*            LEFT HAND SIDE
4       IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
5       IF( ( N.LE.0 ).OR.( ML.LE.0 ).OR.( MU.LE.0 ).OR.
     +    ( ML.GE.N ).OR.( MU.GE.N ) )  THEN
6          IER = 3
7          WRITE(*,*) '  (SUBR. BHLU1)  INVALID ARGUMENT.  ML, MU, N =',
     +             ML, MU, N
8          RETURN
9       END IF
*
10      IER = 0
11      DO 100 K = 1, N
12        JMAX = MIN(K+MU+ML,N)
*              FIND MAXIMUM ELEMENT IN THE K-TH COLUMN.
13        AMAX = ABS(A(0,K))
14        IPK = K
15        DO 110 I = K+1, MIN(K+ML,N)
16          AIK = ABS(A(0,I))
17          IF( AIK.GT.AMAX )  THEN
18             IPK = I
19             AMAX = AIK
20          END IF
21  110   CONTINUE
22        IP(K) = IPK
*
23        IF( AMAX.GT.EPS )  THEN
24           IF( IPK.NE.K )  THEN
25              DO 120 J = K, MIN(K+MU+ML,N)
26                W = A(J-K,IPK)
27                A(J-K,IPK) = A(J-K,K)
28                A(J-K,K) = W
29  120           CONTINUE
30           END IF
*
C*           DO 125 J = K+1, MIN(K+MU+ML,N)
C*125         WK(J) = A(J-K,K)
```

```
*            COMPUTE ALFA AND PERFORM GAUSSIAN ELIMINATION.
31          DO 130 I = K+1, MIN(K+ML,N)
32            AC(I-K,K) = -A(0,I)/A(0,K)
C*            T = -A(0,I)/A(0,K)
*VOPTION LOOP(512)
33            DO 140 J = K+1, MIN(K+MU+ML,N)
34  140         A(J-K-1,I) = A(J-K,I)+AC(I-K,K)*A(J-K,K)
C*140           A(J-K-1,I) = A(J-K,I)+T*WK(J)
35            A(JMAX-K,I) = 0.0D0
36  130       CONTINUE
*             MATRIX IS SINGULAR.
37        ELSE
38          IER = 1
39          A(0,K) = EPS
40          DO 220 I = K+1, MIN(K+ML,N)
41            AC(I-K,K) = 0.0D0
*VOPTION LOOP(512)
42            DO 230 J = K+1, MIN(K+MU+ML,N)
43  230         A(J-K-1,I) = A(J-K,I)
44            A(JMAX-K,I) = 0.0D0
45  220       CONTINUE
46            WRITE(*,*)  '  (SUBR. BHLU1)  MATRIX IS SINGULAR AT K =', K
47            RETURN
48        END IF
49  100 CONTINUE
50      RETURN
51      END
```

（７）Martin-Wilkinsonの特殊ガウス 右辺

```
*
1       SUBROUTINE BHSLV1( A, AC, N, ML, MU, B, IP )
*
*         BHSLV1
*               COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
*
*               SOLVES SIMULTANEOUS LINEAR BAND EQUATIONS
*               BY WILKINSON'S GAUSSIAN ELIMINATION METHOD.
*
*         INPUT - -
*             A(0:ML+MU,N)
*                       R *8  : 2-DIM. ARRAY FOR UPPER TRIANGULAR MATRIX.
*             AC(ML,N) R *8  : 2-DIM. ARRAY CONTAINING ALFA.
*             N         I *4  : ORDER OF MATRIX.
*             ML        I *4  : LOWER BAND WIDTH.
*             MU        I *4  : UPPER BAND WIDTH.
*             B(N)      R *8  : 1-DIM. ARRAY CONTAINING THE RIGHT HAND
*                               SIDE VECTOR.
*             IP(N)     I *4  : PIVOT NUMBER.
*         OUTPUT - -
*             B(N)            : SOLUTION.
*
2       IMPLICIT REAL*8 (A-H,O-Z)
3       DIMENSION A(0:MU+ML,*), AC(ML,*), B(*), IP(*)
*             FORWARD ELIMINATION PROCESS
4       DO 100 K = 1, N
5         IPK = IP(K)
6         IF( IPK.NE.K )  THEN
7            W =B(IPK)
8            B(IPK) = B(K)
9            B(K) = W
10        END IF
*             GAUSSIAN ELIMINATION
C*      T = B(K)
11        DO 110 I = K+1, MIN(K+ML,N)
12  110     B(I) = B(I)+AC(I-K,K)*B(K)
C*110     B(I) = B(I)+AC(I-K,K)*T
13  100 CONTINUE
```

```
*            BACKWARD SUBSTITUTION PROCESS
      DO 200 K = N, 1, -1
        S = -B(K)
        JMAX = MIN(K+MU+ML,N)
        DO 210 J = K+1, JMAX
  210     S = S+A(J-K,K)*B(J)
        B(K) = -S/A(0,K)
  200 CONTINUE
      RETURN
      END
```

（８）Martin-Wilkinsonの特殊ガウス　２段同時

```
*
      SUBROUTINE BHLU2( A, AC, N, ML, MU, EPS, WK1, WK2, IP, IER )
*
*       BHLU2
*             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
*
*             SOLVES SIMULTANEOUS LINEAR BAND EQUATIONS
*             BY WILIKINSON'S GAUSSIAN ELIMINATION METHOD.
*
*       INPUT - -
*            A(0:ML+MU+1,N+1)
*                    R *8  : 2-DIM. ARRAY CONTAINING REAL BAND MATRIX.
*            N       I *4  : ORDER OF MATRIX.
*            ML      I *4  : LOWER BAND WIDTH.
*            MU      I *4  : UPPER BAND WIDTH.
*            EPS     R *8  : PARAMETER TO CHECK SINGULARITY OF THE
*                            MATRIX. ( STANDARD VALUE 3.52D-15 )
*       OUTPUT - -
*            A(0:ML+MU+1,N+1)
*                          : UPPER TRIANGULAR MATRIX.
*            AC(ML+1,N+1)
*                    R *8  : 2-DIM. ARRAY CONTAINING ALFA.
*            IP(N+1)  I *4  : PIVOT NUMBER.
*            IER      I *4  : = 0,  FOR NORMAL EXECUTION.
*                             = 1,  FOR SINGULAR MATRIX.
*                             = 3,  FOR INVALID ARGUEMENT.
*       WORKING  -
*            WK1(N), WK2(N)
*                    R *8  : 1-DIM. ARRAY.
*
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(0:MU+ML+1,*), AC(ML+1,*), IP(*), WK1(*), WK2(*)
*           LEFT HAND SIDE
      IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
      IF( ( N.LE.0 ).OR.( ML.LE.0 ).OR.( MU.LE.0 ).OR.
     +    ( ML.GE.N ).OR.( MU.GE.N ) )  THEN
         IER = 3
         WRITE(*,*) ' (SUBR. BHLU2)  INVALID ARGUMENT.  ML, MU, N =',
     +              ML, MU, N
         RETURN
      END IF
*
      IER = 0
      DO 10 I = 1, N
   10   A(MU+ML+1,I) = 0.0D0
      IF( MOD(N,2).NE.0 )  THEN
         DO 20 J = 0, MU+ML+1
   20      A(J,N+1) = 0.0D0
         A(0,N+1) = 1.0D0
      END IF
*
      DO 100 K = 1, N, 2
*
        K1 = K+1
        IMAX = MIN(K+ML,N)
        JMAX = MIN(K+MU+ML,N)
        IMAX1 = MIN(K1+ML,N)
        JMAX1 = MIN(K1+MU+ML,N)
*           FIND MAXIMUM ELEMENT IN THE K-TH COLUMN.
        AMAX = ABS(A(0,K))
        IPK = K
```

```
      DO 110 I = K+1, IMAX
        AIK = ABS(A(0,I))
        IF( AIK.GT.AMAX )  THEN
           IPK = I
           AMAX = AIK
        END IF
110   CONTINUE
      IP(K) = IPK
*           EXCHANGE FOR K-TH ELIMINATION.
      IF( AMAX.GT.EPS )  THEN
         IF( IPK.NE.K )  THEN
            DO 120 J = K, JMAX
              W = A(J-K,IPK)
              A(J-K,IPK) = A(J-K,K)
              A(J-K,K) = W
120         CONTINUE
         END IF
*           COMPUTE ALFA
         T = A(0,K)
         T1 = A(1,K)
         DO 130 I = K+1, IMAX
           AC(I-K,K) = -A(0,I)/T
130        A(0,I) = A(1,I)+AC(I-K,K)*T1
         AC(IMAX+1-K,K) = 0.0D0
*           MATRIX IS SINGULAR.
      ELSE
         IER = 1
         IP(K) = K
         DO 150 I = K+1, IMAX
           A(0,I) = A(1,I)
150        AC(I-K,K) = 0.0D0
         AC(IMAX+1-K,K) = 0.0D0
         WRITE(*,*)  '  (SUBR. BHLU2)  MATRIX IS SINGULAR AT K =', K
         RETURN
      END IF
*           FIND MAXIMUM ELEMENT IN THE K+1'TH COLUMN.
      AMAX1 = ABS(A(0,K1))
      IPK1 = K1
      DO 200 I = K1+1, IMAX1
        AIK = ABS(A(0,I))
        IF( AIK.GT.AMAX1 )  THEN
           IPK1 = I
           AMAX1 = AIK
        END IF
200   CONTINUE
      IP(K1) = IPK1
*           EXCHANGE FOR K+1'TH ELIMINATION.
      IF( AMAX1.GT.EPS )  THEN
         IF( IPK1.NE.K1 )  THEN
            DO 210 J = K, JMAX1
              W = A(J-K,IPK1)
              A(J-K,IPK1) = A(J-K,K1)
              A(J-K,K1) = W
210         CONTINUE
            W = AC(IPK1-K,K)
            AC(IPK1-K,K) = AC(1,K)
            AC(1,K) = W
         END IF
*           COMPUTE ALFA
         T = A(0,K1)
         DO 220 I = K1+1, IMAX1
220        AC(I-K1,K1) = -A(0,I)/T
      ELSE
*           MATRIX IS SINGULAR.
         IER = 1
         IP(K1) = K1
         DO 230 I = K1+1, IMAX1
230        AC(I-K1,K1) = 0.0D0
         IF( ( MOD(N,2).NE.0 ).AND.( K1.EQ.(N+1) ) )  THEN
            IER = 0
         ELSE
            WRITE(*,*)  '  (SUBR. BHLU2)  MATRIX IS SINGULAR AT K =',
     +                  K1
            RETURN
         END IF
      END IF
```

```
*            GAUSSIAN ELIMINATION FOR K-TH AND K+1'TH PROCESS.
94         IF( IPK1.LE.(K+ML) )  THEN
*
95           A(JMAX+1-K,K) = 0.0D0
96           T = AC(1,K)
97           DO 240 J = K1+1, JMAX1
98             WK1(J) = A(J-K,K)
99             A(J-K1,K1) = A(J-K,K1)+T*WK1(J)
100  240       WK2(J) = A(J-K1,K1)
101          A(JMAX1-K,K1) = 0.0D0
*
102          DO 300 I = K1+1, IMAX
103            T = AC(I-K,K)
104            T1 = AC(I-K1,K1)
*VOPTION LOOP(512)
105            DO 310 J = K1+1, JMAX1
106  310         A(J-K-2,I) = A(J-K,I)+T*WK1(J)+T1*WK2(J)
107            A(JMAX1-K,I) = 0.0D0
108            A(JMAX1-K-1,I) = 0.0D0
109  300       CONTINUE
110          T = AC(IMAX-K,K1)
111          DO 320 J = K1+1, JMAX1
112  320       A(J-K-2,IMAX+1) = A(J-K1,IMAX+1)+T*WK2(J)
113          A(JMAX1-K-1,IMAX+1) = 0.0D0
*
114        ELSE
115          A(JMAX+1-K,K) = 0.0D0
116          DO 350 J = K1+1, JMAX1
117            WK1(J) = A(J-K,K)
118  350       WK2(J) = A(J-K1,K1)
119          DO 400 I = K1+1, IMAX1
120            T = AC(I-K,K)
121            T1 = AC(I-K1,K1)
*VOPTION LOOP(512)
122            DO 410 J = K1+1, JMAX1
123  410         A(J-K-2,I) = A(J-K,I)+T*WK1(J)+T1*WK2(J)
124            A(JMAX1-K,I) = 0.0D0
125            A(JMAX1-K-1,I) = 0.0D0
126  400       CONTINUE
127        END IF
128  100 CONTINUE
129      RETURN
130      END
```

（９）Martin-Wilkinsonの特殊ガウス　２段２行同時

```
*
1      SUBROUTINE BHLU4( A, AC, N, ML, MU, EPS, WK1, WK2, IP, IER )
*
*        BHLU4
*              COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
*
*              SOLVES SIMULTANEOUS LINEAR BAND EQUATIONS
*              BY WILIKINSON'S GAUSSIAN ELIMINATION METHOD.
*
*        INPUT - -
*             A(0:ML+MU+1,N+1)
*                      R *8  : 2-DIM. ARRAY CONTAINING REAL BAND MATRIX.
*             N        I *4  : ORDER OF MATRIX.
*             ML       I *4  : LOWER BAND WIDTH.
*             MU       I *4  : UPPER BAND WIDTH.
*             EPS      R *8  : PARAMETER TO CHECK SINGULARITY OF THE
*                              MATRIX. ( STANDARD VALUE 3.52D-15 )
*        OUTPUT - -
*             A(0:ML+MU+1,N+1)
*                            : UPPER TRIANGULAR MATRIX.
*             AC(ML+1,N+1)
*                      R *8  : 2-DIM. ARRAY CONTAINING ALFA.
*             IP(N+1)  I *4  : PIVOT NUMBER.
*             IER      I *4  : = 0,  FOR NORMAL EXECUTION.
*                              = 1,  FOR SINGULAR MATRIX.
*                              = 3,  FOR INVALID ARGUEMENT.
```

```
*       WORKING  -
*            WK1(N), WK2(N)
*                    R *8  : 1-DIM. ARRAY.
*
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(0:MU+ML+1,*), AC(ML+1,*), IP(*), WK1(*), WK2(*)
*             LEFT HAND SIDE
      IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
      IF( ( N.LE.0 ).OR.( ML.LE.0 ).OR.( MU.LE.0 ).OR.
     +    ( ML.GE.N ).OR.( MU.GE.N ) )  THEN
         IER = 3
         WRITE(*,*) '  (SUBR. BHLU4)  INVALID ARGUMENT.  ML, MU, N =',
     +              ML, MU, N
         RETURN
      END IF
*
      IER = 0
      DO 10 I = 1, N
   10   A(MU+ML+1,I) = 0.0D0
      IF( MOD(N,2).NE.0 )  THEN
         DO 20 J = 0, MU+ML+1
   20      A(J,N+1) = 0.0D0
         A(0,N+1) = 1.0D0
      END IF
*
      DO 100 K = 1, N, 2
*
        K1 = K+1
        IMAX = MIN(K+ML,N)
        JMAX = MIN(K+MU+ML,N)
        IMAX1 = MIN(K1+ML,N)
        JMAX1 = MIN(K1+MU+ML,N)
*            FIND MAXIMUM ELEMENT IN THE K-TH COLUMN.
        AMAX = ABS(A(0,K))
        IPK = K

        DO 110 I = K+1, IMAX
          AIK = ABS(A(0,I))
          IF( AIK.GT.AMAX )  THEN
             IPK = I
             AMAX = AIK
          END IF
  110   CONTINUE
        IP(K) = IPK
*            EXCHANGE FOR K-TH ELIMINATION.
        IF( AMAX.GT.EPS )  THEN
           IF( IPK.NE.K )  THEN
              DO 120 J = K, JMAX
                W = A(J-K,IPK)
                A(J-K,IPK) = A(J-K,K)
  120           A(J-K,K) = W
           END IF
*             COMPUTE ALFA
           T = A(0,K)
           T1 = A(1,K)
           DO 130 I = K+1, IMAX
             AC(I-K,K) = -A(0,I)/T
  130        A(0,I) = A(1,I)+AC(I-K,K)*T1
           AC(IMAX+1-K,K) = 0.0D0
*             MATRIX IS SINGULAR.
        ELSE
           A(0,K) = EPS
           IER = 1
           IP(K) = K
           DO 150 I = K+1, IMAX
             A(0,I) = A(1,I)
  150        AC(I-K,K) = 0.0D0
           AC(IMAX+1-K,K) = 0.0D0
           WRITE(*,*)  '  (SUBR. BHLU4)  MATRIX IS SINGULAR AT K =', K
           RETURN
        END IF
```

```
*            FIND MAXIMUM ELEMENT IN THE K+1'TH COLUMN.
      AMAX1 = ABS(A(0,K1))
      IPK1 = K1
      DO 200 I = K1+1, IMAX1
        AIK = ABS(A(0,I))
        IF( AIK.GT.AMAX1 )  THEN
          IPK1 = I
          AMAX1 = AIK
        END IF
  200 CONTINUE
      IP(K1) = IPK1
*            EXCHANGE FOR K+1'TH ELIMINATION.
      IF( AMAX1.GT.EPS )  THEN
        IF( IPK1.NE.K1 )  THEN
          DO 210 J = K, JMAX1
            W = A(J-K,IPK1)
            A(J-K,IPK1) = A(J-K,K1)
  210       A(J-K,K1) = W
          W = AC(IPK1-K,K)
          AC(IPK1-K,K) = AC(1,K)
          AC(1,K) = W
        END IF
*          COMPUTE ALFA
        T = A(0,K1)
        DO 220 I = K1+1, IMAX1
  220     AC(I-K1,K1) = -A(0,I)/T
*          MATRIX IS SINGULAR.
      ELSE
        IER = 1
        A(0,K1) = EPS
        IP(K1) = K1
        DO 230 I = K1+1, IMAX1
  230     AC(I-K1,K1) = 0.0D0
        IF( ( MOD(N,2).NE.0 ).AND.( K1.EQ.(N+1) ) )  THEN
          IER = 0
        ELSE
          WRITE(*,*)  '  (SUBR. BHLU4)  MATRIX IS SINGULAR AT K =',
     +                K1
          RETURN
        END IF
      END IF
*            GAUSSIAN ELIMINATION 2 COL. FOR K-TH AND K+1'TH PROCESS.
      IF( IPK1.LE.(K+ML) )  THEN
*
        A(JMAX+1-K,K) = 0.0D0
        T = AC(1,K)
        DO 240 J = K1+1, JMAX1
          WK1(J) = A(J-K,K)
          A(J-K1,K1) = A(J-K,K1)+T*WK1(J)
  240     WK2(J) = A(J-K1,K1)
        A(JMAX1-K,K1) = 0.0D0
*
        LOOPI = IMAX-K1
        IF( MOD(LOOPI,2).EQ.0.OR.LOOPI.LT.0 )  THEN
          DO 301 I = K1+1, IMAX, 2
            T = AC(I-K,K)
            T1 = AC(I-K1,K1)
            U = AC(I+1-K,K)
            U1 = AC(I-K,K1)
*VOPTION LOOP(512)
            DO 311 J = K1+1, JMAX1
              A(J-K-2,I) = A(J-K,I)+T*WK1(J)+T1*WK2(J)
  311         A(J-K-2,I+1) = A(J-K,I+1)+U*WK1(J)+U1*WK2(J)
            A(JMAX1-K,I) = 0.0D0
            A(JMAX1-K-1,I) = 0.0D0
            A(JMAX1-K,I+1) = 0.0D0
            A(JMAX1-K-1,I+1) = 0.0D0
  301       CONTINUE
        ELSE
          DO 300 I = K1+1, IMAX-1, 2
            T = AC(I-K,K)
            T1 = AC(I-K1,K1)
            U = AC(I+1-K,K)
            U1 = AC(I-K,K1)
```

```
*VOPTION LOOP(512)
123                 DO 310 J = K1+1, JMAX1
124                   A(J-K-2,I) = A(J-K,I)+T*WK1(J)+T1*WK2(J)
125   310             A(J-K-2,I+1) = A(J-K,I+1)+U*WK1(J)+U1*WK2(J)
126                 A(JMAX1-K,I) = 0.0D0
127                 A(JMAX1-K-1,I) = 0.0D0
128                 A(JMAX1-K,I+1) = 0.0D0
129                 A(JMAX1-K-1,I+1) = 0.0D0
130   300           CONTINUE
131                 T = AC(IMAX-K,K)
132                 T1 = AC(IMAX-K1,K1)
133                 DO 309 J = K1+1, JMAX1
134   309             A(J-K-2,IMAX) = A(J-K,IMAX)+T*WK1(J)+T1*WK2(J)

135                 A(JMAX1-K,IMAX) = 0.0D0
136                 A(JMAX1-K-1,IMAX) = 0.0D0
137               END IF
*
138               T = AC(IMAX-K,K1)
139               DO 320 J = K1+1, JMAX1
140   320           A(J-K1-1,IMAX+1) = A(J-K1,IMAX+1)+T*WK2(J)
141               A(JMAX1-K-1,IMAX+1) = 0.0D0
142             ELSE
143               A(JMAX+1-K,K) = 0.0D0
144               DO 350 J = K1+1, JMAX1
145                 WK1(J) = A(J-K,K)
146   350           WK2(J) = A(J-K1,K1)
147               LOOPI = IMAX1-K1
148               IF( MOD(LOOPI,2).EQ.0.OR.LOOPI.LT.0 )  THEN
149                 DO 400 I = K1+1, IMAX1, 2
150                   T = AC(I-K,K)
151                   T1 = AC(I-K1,K1)
152                   U = AC(I+1-K,K)
153                   U1 = AC(I-K,K1)
*VOPTION LOOP(512)
154                   DO 410 J = K1+1, JMAX1
155                     A(J-K-2,I) = A(J-K,I)+T*WK1(J)+T1*WK2(J)
156   410               A(J-K-2,I+1) = A(J-K,I+1)+U*WK1(J)+U1*WK2(J)
157                   A(JMAX1-K,I) = 0.0D0
158                   A(JMAX1-K-1,I) = 0.0D0
159                   A(JMAX1-K,I+1) = 0.0D0
160                   A(JMAX1-K-1,I+1) = 0.0D0
161   400             CONTINUE
162               ELSE
163                 DO 401 I = K1+1, IMAX1-1, 2
164                   T = AC(I-K,K)
165                   T1 = AC(I-K1,K1)
166                   U = AC(I+1-K,K)
167                   U1 = AC(I-K,K1)
*VOPTION LOOP(512)
168                   DO 411 J = K1+1, JMAX1
169                     A(J-K-2,I) = A(J-K,I)+T*WK1(J)+T1*WK2(J)
170   411               A(J-K-2,I+1) = A(J-K,I+1)+U*WK1(J)+U1*WK2(J)
171                   A(JMAX1-K,I) = 0.0D0
172                   A(JMAX1-K-1,I) = 0.0D0
173                   A(JMAX1-K,I+1) = 0.0D0
174                   A(JMAX1-K-1,I+1) = 0.0D0
175   401             CONTINUE
176                 T = AC(IMAX1-K,K)
177                 T1 = AC(IMAX1-K1,K1)
178                 DO 412 J = K1+1, JMAX1
179   412             A(J-K-2,IMAX1) = A(J-K,IMAX1)+T*WK1(J)+T1*WK2(J)
180                 A(JMAX1-K,IMAX1) = 0.0D0
181                 A(JMAX1-K-1,IMAX1) = 0.0D0
182               END IF
183             END IF
184   100 CONTINUE
185       RETURN
186       END
```

(10) Martin-Wilkinsonの特殊ガウス 右辺

2段同時と2段2行同時用

```
      *
    1       SUBROUTINE BHSLV4( A, AC, N, ML, MU, B, IP )
      *
      *       BHSLV4
      *             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
      *
      *             SOLVES SIMULTANEOUS LINEAR BAND EQUATIONS
      *             BY WILKINSON'S GAUSSIAN ELIMINATION METHOD.
      *
      *       INPUT - -
      *           A(0:ML+MU+1,N+1)
      *                   R *8  : 2-DIM. ARRAY FOR UPPER TRIANGULAR MATRIX.
      *           AC(ML+1,N+1)
      *                   R *8  : 2-DIM. ARRAY CONTAINING ALFA.
      *           N       I *4  : ORDER OF MATRIX.
      *           ML      I *4  : LOWER BAND WIDTH.
      *           MU      I *4  : UPPER BAND WIDTH.
      *           B(N+1)  R *8  : 1-DIM. ARRAY CONTAINING THE RIGHT HAND
      *                           SIDE VECTOR.
      *           IP(N+1) I *4  : PIVOT NUMBER.
      *       OUTPUT - -
      *           B(N+1)        : SOLUTION.
      *
    2       IMPLICIT REAL*8 (A-H,O-Z)
    3       DIMENSION A(0:MU+ML+1,*), AC(ML+1,*), B(*), IP(*)
      *           FORWARD ELIMINATION PROCESS
    4       DO 100 K = 1, N, 2
      *           EXCHANGE FOR K-TH ELIMINATION.
    5         IPK = IP(K)
    6         IF( IPK.NE.K )  THEN
    7           W = B(IPK)
    8           B(IPK) = B(K)
    9           B(K) = W
   10         END IF
      *           EXCHANGE FOR K+1'TH ELIMINATION.
   11         K1 = K+1
   12         IPK1 = IP(K1)
   13         IF( IPK1.NE.K1 )  THEN
   14           W = B(IPK1)
   15           B(IPK1) = B(K1)
   16           B(K1) = W
   17         END IF
      *
   18         B(K1) = B(K1)+AC(1,K)*B(K)
      *           GAUSSIAN ELIMINATION FOR K-TH AND K+1'TH PROCESS.
   19         T = B(K)
   20         T1 = B(K1)
   21         DO 110 I = K1+1, MIN(K1+ML,N)
   22   110     B(I) = B(I)+AC(I-K,K)*T+AC(I-K1,K1)*T1
   23   100 CONTINUE
      *           BACKWARD SUBSTITUTION PROCESS
   24       IF( MOD(N,2).EQ.0 )  THEN
   25         NEND = N
   26       ELSE
   27         NEND = N+1
   28         B(N+1) = 0.0D0
   29         A(0,N+1) = 1.0D0
   30       END IF
   31       DO 200 K = NEND, 1, -2
   32         JMAX = MIN(K+MU+ML,N)
   33         S1 = -B(K)
   34         S0 = -B(K-1)
   35         DO 210 J = K+1, JMAX
   36           S1 = S1+A(J-K,K)*B(J)
   37   210     S0 = S0+A(J-K+1,K-1)*B(J)
   38         B(K) = -S1/A(0,K)
   39         B(K-1) = (-S0-A(1,K-1)*B(K))/A(0,K-1)
   40   200 CONTINUE
   41       RETURN
   42       END
```

(11) 対称帯ガウス

```
      *
    1       SUBROUTINE BSLU1( A, N, MU, EPS, WK, IER )
      *
      *       BSLU1
      *             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
      *
      *             SOLVES SIMULTANEOUS LINEAR SYMMETRIC BAND EQUATIONS
      *             BY SYMMETRIC GAUSSIAN ELIMINATION METHOD.
      *
      *       INPUT - -
      *            A(0:MU,N)
      *                     R *8  : 2-DIM. ARRAY FOR UPPER TRIANGULAR MATRIX.
      *            N        I *4  : ORDER OF MATRIX.
      *            MU       I *4  : UPPER BAND WIDTH.
      *            EPS      R *8  : PARAMETER TO CHECK SINGULARITY OF THE
      *                             MATRIX. ( STANDARD VALUE 3.52D-15 )
      *       OUTPUT - -
      *            A(0:MU,N)      : RESULT OF GAUSSIAN ELIMINATION.
      *            IER      I *4  : = 0,  FOR NORMAL EXECUTION.
      *                             = 1,  FOR SINGULAR MATRIX.
      *                             = 3,  FOR INVALID ARGUEMENT.
      *       WORKING  -
      *            WK(N)    R *8  : 1-DIM. ARRAY.
      *
    2       IMPLICIT REAL*8 (A-H,O-Z)
    3       DIMENSION A(0:MU,*), WK(*)
      *            LEFT HAND SIDE
    4       IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
    5       IF( ( N.LE.0 ).OR.( MU.LE.0 ).OR.( MU.GE.N ) )  THEN
    6         IER = 3
    7         WRITE(*,*) '  (SUBR. BSLU1)  INVALID ARGUMENT.  MU, N =',
     +                MU, N
    8         RETURN
    9       END IF
      *
   10       IER = 0
   11       DO 100 K = 1, N
      *
   12         IF( ABS(A(0,K)).LE.EPS )  THEN
   13           IER = 1
   14           WRITE(*,*)  '  (SUBR. BSLU1)  MATRIX IS SINGULAR AT K =', K
   15           RETURN
   16         END IF
      *
      C*       DO 110 J = K+1, MIN(K+MU,N)
      C*110      WK(J) = A(J-K,K)
      *             GAUSSIAN ELIMINATION.
   17         DIVA = -1.0D0/A(0,K)
   18         DO 120 I = K+1, MIN(K+MU,N)
      C*         T = -A(I-K,K)/A(0,K)
   19           T = A(I-K,K)*DIVA
      *VOPTION LOOP(512)
   20           DO 130 J = I, MIN(K+MU,N)
   21    130      A(J-I,I) = A(J-I,I)+T*A(J-K,K)
      C*130      A(J-I,I) = A(J-I,I)+T*WK(J)
   22    120    CONTINUE
   23    100  CONTINUE
   24         RETURN
   25       END
```

(12) 対称帯ガウス 右辺

```
      *
     1      SUBROUTINE BSSLV1( A, N, MU, B )
      *
      *       BSSLV1
      *             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
      *
      *             SOLVES SIMULTANEOUS LINEAR SYMMETRIC BAND EQUATIONS
      *             BY SYMMETRIC GAUSSIAN ELIMINATION METHOD.
      *
      *       INPUT - -
      *           A(0:MU,N)
      *                   R *8  : RESULT OF GAUSSIAN ELIMINATION.
      *           N       I *4  : ORDER OF MATRIX.
      *           MU      I *4  : UPPER BAND WIDTH.
      *           B(N)    R *8  : 1-DIM. ARRAY CONTAINING THE RIGHT HAND
      *                           SIDE VECTOR.
      *       OUTPUT - -
      *           B(N)          : SOLUTION.
      *
     2      IMPLICIT REAL*8 (A-H,O-Z)
     3      DIMENSION A(0:MU,*), B(*)
      *            FORWARD ELIMINATION PROCESS
     4      DO 100 K = 1, N
     5        BK = -B(K)/A(0,K)
     6        DO 110 I = K+1, MIN(K+MU,N)
     7  110     B(I) = B(I)+A(I-K,K)*BK
     8  100 CONTINUE
      *            BACKWARD SUBSTITUTION PROCESS
     9      DO 200 K = N, 1, -1
    10        S = -B(K)
    11        DO 210 J = K+1, MIN(K+MU,N)
    12  210     S = S+A(J-K,K)*B(J)
    13        B(K) = -S/A(0,K)
    14  200 CONTINUE
    15      RETURN
    16      END
```

(13) 対称帯ガウス　2段同時

```
      *
     1      SUBROUTINE BSLU2( A, N, MU, EPS, WK1, WK2, IER )
      *
      *       BSLU2
      *             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
      *
      *             SOLVES SIMULTANEOUS LINEAR SYMMETRIC BAND EQUATIONS
      *             BY SYMMETRIC GAUSSIAN ELIMINATION METHOD.
      *
      *       INPUT - -
      *           A(0:MU+2,N+1)
      *                   R *8  : 2-DIM. ARRAY FOR UPPER TRIANGULAR MATRIX.
      *           N       I *4  : ORDER OF MATRIX.
      *           MU      I *4  : UPPER BAND WIDTH.
      *           EPS     R *8  : PARAMETER TO CHECK SINGULARITY OF THE
      *                           MATRIX. ( STANDARD VALUE 3.52D-15 )
      *       OUTPUT - -
      *           A(0:MU+2,N+1)  : RESULT OF GAUSSIAN ELIMINATION.
      *           IER     I *4  : = 0,  FOR NORMAL EXECUTION.
      *                           = 1,  FOR SINGULAR MATRIX.
      *                           = 3,  FOR INVALID ARGUEMENT.
      *       WORKING  -
      *           WK1(N), WK2(N)
      *                   R *8  : 1-DIM. ARRAY.
      *
     2      IMPLICIT REAL*8 (A-H,O-Z)
     3      DIMENSION A(0:MU+2,*), WK1(*), WK2(*)
```

```
*            LEFT HAND SIDE
4       IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
5       IF( ( N.LE.0 ).OR.( MU.LE.0 ).OR.( MU.GE.N ) )  THEN
6          IER = 3
7          WRITE(*,*) '  (SUBR. BSLU2)  INVALID ARGUMENT.  MU, N =',
     +               MU, N
8          RETURN
9       END IF
*
10      DO 10 I = 1, N
11        A(MU+1,I) = 0.0D0
12   10   A(MU+2,I) = 0.0D0
13      IF( MOD(N,2).NE.0 )  THEN
14         A(0,N+1) = 1.0D0
15         DO 20 J = 1, MU+2
16   20      A(J,N+1) = 0.0D0
17      END IF
*
18      IER = 0
19      DO 100 K = 1, N, 2
20        K1 = K+1
*
21        IF( ABS(A(0,K)).LE.EPS )  THEN
22           IER = 1
23           WRITE(*,*)  '  (SUBR. BSLU2)  MATRIX IS SINGULAR AT K =', K
24           RETURN
25        END IF
*
26        T = -A(1,K)/A(0,K)
27        DO 110 J = K+1, MIN(K1+MU,N)
28          WK1(J) = A(J-K,K)
29          A(J-K1,K1) = A(J-K1,K1)+T*WK1(J)
30          WK2(J) = A(J-K1,K1)
31   110   CONTINUE
*
32        IF( ABS(A(0,K1)).LE.EPS )  THEN
33           IER = 1
34           WRITE(*,*)  '  (SUBR. BSLU2)  MATRIX IS SINGULAR AT K =', K1
35           RETURN
36        END IF
*              GAUSSIAN ELIMINATION FOR K-TH AND K+1'TH PROCESS.
37        DIVA = -1.0D0/A(0,K)
38        DIVA1 = -1.0D0/A(0,K1)
39        DO 130 I = K1+1, MIN(K1+MU,N)
C*          T = -A(I-K,K)/A(0,K)
C*          T1 = -A(I-K1,K1)/A(0,K1)
40          T = A(I-K,K)*DIVA
41          T1 = A(I-K1,K1)*DIVA1
*VOPTION LOOP(512)
42          DO 140 J = I, MIN(K1+MU,N)
43   140      A(J-I,I) = A(J-I,I)+T*WK1(J)+T1*WK2(J)
44   130   CONTINUE
45   100 CONTINUE
46      RETURN
47      END
```

(14) 対称帯ガウス　2段2行同時

```
*
      SUBROUTINE BSLU4( A, N, MU, EPS, WK1, WK2, IER )
*
*        BSLU4
*              COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
*
*              SOLVES SIMULTANEOUS LINEAR SYMMETRIC BAND EQUATIONS
*              BY SYMMETRIC GAUSSIAN ELIMINATION METHOD.
*
*        INPUT - -
*            A(0:MU+2,N+1)
*                     R *8  : 2-DIM. ARRAY FOR UPPER TRIANGULAR MATRIX.
*            N        I *4  : ORDER OF MATRIX.
*            MU       I *4  : UPPER BAND WIDTH.
*            EPS      R *8  : PARAMETER TO CHECK SINGULARITY OF THE
*                             MATRIX. ( STANDARD VALUE 3.52D-15 )
*        OUTPUT - -
*            A(0:MU+2,N+1)  : RESULT OF GAUSSIAN ELIMINATION.
*            IER      I *4  : = 0,  FOR NORMAL EXECUTION.
*                             = 1,  FOR SINGULAR MATRIX.
*                             = 3,  FOR INVALID ARGUEMENT.
*        WORKING  -
*            WK1(N), WK2(N)
*                     R *8  : 1-DIM. ARRAY.
*
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(0:MU+2,*), WK1(*), WK2(*)
*           LEFT HAND SIDE
      IF( EPS.LT.0.0D0 )  EPS = 3.52D-15
      IF( ( N.LE.0 ).OR.( MU.LE.0 ).OR.( MU.GE.N ) )  THEN
         IER = 3
         WRITE(*,*) '  (SUBR. BSLU4)  INVALID ARGUMENT.  MU, N =',
     +              MU, N
         RETURN
      END IF
*
      IER = 0
      DO 10 I = 1, N
        A(MU-1,I) = 0.0D0
   10   A(MU-2,I) = 0.0D0
      IF( MOD(N,2).NE.0 )  THEN
          A(0,N+1) = 1.0D0
          DO 20 J = 1, MU+2
   20       A(J,N+1) = 0.0D0
      END IF
*
      DO 100 K = 1, N, 2
        K1 = K+1
*
        IF( ABS(A(0,K)).LE.EPS )  THEN
           IER = 1
           WRITE(*,*)  '  (SUBR. BSLU4)  MATRIX IS SINGULAR AT K =', K
           RETURN
        END IF
*
        T = -A(1,K)/A(0,K)
        DO 110 J = K+1, MIN(K1+MU,N)
          WK1(J) = A(J-K,K)
          A(J-K1,K1) = A(J-K1,K1)+T*WK1(J)
          WK2(J) = A(J-K1,K1)
  110   CONTINUE
*
        IF( ABS(A(0,K1)).LE.EPS )  THEN
           IER = 1
           WRITE(*,*)  '  (SUBR. BSLU4)  MATRIX IS SINGULAR AT K =', K1
           RETURN
        END IF
```

```
*            GAUSSIAN ELIMINATION 2 COL. FOR K-TH AND K+1'TH PROCESS.
      DIVA = -1.0D0/A(0,K)
      DIVA1 = -1.0D0/A(0,K1)
      DO 120 I = K1+1, MIN(K1+MU,N), 2
C*      T = -A(I-K,K)/A(0,K)
C*      T1 = -A(I-K1,K1)/A(0,K1)
C*      U = -A(I+1-K,K)/A(0,K)
C*      U1 = -A(I-K,K1)/A(0,K1)
        T = A(I-K,K)*DIVA
        T1 = A(I-K1,K1)*DIVA1
        U = A(I+1-K,K)*DIVA
        U1 = A(I-K,K1)*DIVA1
        A(0,I) = A(0,I)+T*WK1(I)+T1*WK2(I)
*VOPTION LOOP(512)
        DO 130 J = I+1, MIN(K1+MU,N)
          A(J-I,I)     = A(J-I,I)    +T*WK1(J)+T1*WK2(J)
          A(J-I-1,I+1) = A(J-I-1,I+1)+U*WK1(J)+U1*WK2(J)
  130   CONTINUE
  120 CONTINUE
  100 CONTINUE
      RETURN
      END
```

(15) 対称帯ガウス 右辺 2段同時と2段2行同時用

```
*
      SUBROUTINE BSSLV4( A, N, MU, B )
*
*       BSSLV4
*             COPYRIGHT : H.HASEGAWA, OCT.  4 1991 V.1
*
*             SOLVES SIMULTANEOUS LINEAR SYMMETRIC BAND EQUATIONS
*             BY SYMMETRIC GAUSSIAN ELIMINATION METHOD.
*
*       INPUT - -
*            A(0:MU+2,N+1)
*                    R *8  : RESULT OF GAUSSIAN ELIMINATION.
*            N       I *4  : ORDER OF MATRIX.
*            MU      I *4  : UPPER BAND WIDTH.
*            B(N+1)  R *8  : 1-DIM. ARRAY CONTAINING THE RIGHT HAND
*                            SIDE VECTOR.
*       OUTPUT - -
*            B(N+1)        : SOLUTION.
*
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(0:MU+2,*), B(*)
*             FORWARD ELIMINATION PROCESS
      DO 100 K = 1, N, 2
        K1 = K+1
        T = -A(1,K)/A(0,K)
        B(K1) = B(K1)+T*B(K)
        BK = -B(K)/A(0,K)
        BK1 = -B(K1)/A(0,K1)
        DO 110 I = K1+1, MIN(K1+MU,N)
  110     B(I) = B(I)+A(I-K,K)*BK+A(I-K1,K1)*BK1
  100 CONTINUE
*             BACKWARD SUBSTITUTION PROCESS
      IF( MOD(N,2).EQ.0 )  THEN
        NEND = N
      ELSE
        NEND = N+1
        B(N+1) = 0.0D0
      END IF
      DO 200 K = NEND, 1, -2
        S1 = -B(K)
        S0 = -B(K-1)
        DO 210 J = K+1, MIN(K+MU,N)
          S1 = S1+A(J-K,K)*B(J)
  210     S0 = S0+A(J-K+1,K-1)*B(J)
        B(K) = -S1/A(0,K)
        B(K-1) = (-S0-A(1,K-1)*B(K))/A(0,K-1)
  200 CONTINUE
      RETURN
      END
```

(16) 差分法による離散化

```
*
*            MATRIX ELEMENTS FROM DIFFUSION OPERATOR DIV(-DF*GRAD( ));
*            DF = DIFFUSION COEFFICIENTS ( BY F.D.M. )
*
      SUBROUTINE INPUT( N, M1, AA, AB, AC, WK, DF0, DF1, DF2, DF3, A )
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION AA(N), AB(-M1:N), AC(-M1:N), WK(0:N+M1), A(-M1:0,N)
*             SET DIFFUSION COEFFICIENT
      DO 100 J = 0, N+M1
  100   WK(J) = 1.0D0
      DO 110 J = 1, N
  110   AA(J) = 0.0
      DO 120 J = -M1, N
        AB(J) = 0.0
  120   AC(J) = 0.0
      DO 130 I = 1, M1*2
  130   WK(I) = DF1
      DO 140 I = M1*(M1+1)+1, M1*(M1+3)
  140   WK(I) = DF2
      DO 150 I = M1*(2*M1+2)+1, M1*(2*M1+3)+M1
  150   WK(I) = DF3
      WK(0) = DF0
      DO 160 I = M1, N+M1, M1
  160   WK(I) = DF0
*             MAKE A MATRIX
      DO 200 I = 1, N
        AA(I) = WK(I-1)+WK(I+M1-1)+WK(I+M1)+WK(I)
        AB(I) = -(WK(I+M1)+WK(I))*0.5D0
  200   AC(I) = -(WK(I+M1-1)+WK(I+M1))*0.5D0
      DO 210 I = N-M1+1, N
  210   AC(I) = 0.0
*
        DO 300 J = 1, N
          DO 310 I = -M1, 0
  310       A(I,J) = 0.0
          A(0,J) = AA(J)
          A(-1,J) = AB(J-1)
          A(-M1,J) = AC(J-M1)
  300   CONTINUE
      RETURN
      END
```

(17) (M)ＩＣＣＧ（１，１）法

```
*
*        ＩＣＣＧ法      （５点差分）
*
      SUBROUTINE PCG21D( AA, AB, AC, DINV, P, R, W, BO, X, AP,
     +             EPSLON, ERR, M1, N, KCOUNT, KLIMIT, BNORM, UP,
     +             X1, X2, IOPT, COND )
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AA(N), AB(-M1:N), AC(-M1:N), DINV(-M1:N),
     +         P(1-M1:N+M1), W(1-M1:N+M1), BO(N), X(1-M1:N+M1),
     +         AP(N), R(N), X1(1-M1:N+M1), X2(N)
      REAL *8 LAM1, LAM2
      DATA KKE1, KKE2, EPSEIG, EPSD / 50, 20, 1.0D-3, 0.22D-12 /
*
*        初期設定
*
    1 UP = MAX( 0.0D0, UP )
      CALL JCCG11( N, M1, AA, AB, AC, DINV, UP, DMIN )
      IF( ( UP.GT.0.0 ).AND.( DMIN.LE.EPSD ) )  THEN
        UP = UP-0.05
        GO TO 1
      END IF
*
```

```
12      IF( ABS(IOPT).GT.2 )  THEN
   *
   *        べき乗法
13        LAM1 = 0.0D0
14        CALL RANDOM( N, X2 )
15        DO 100 K0 = 1, KKE1
16          S = 0.0
17          DO 110 I = 1, N
18            X1(I) = X2(I)
19  110         S = S + X1(I)*X1(I)
20          DO 120 I = 1, N
21  120         X1(I) = X1(I)/SQRT(S)
22          CALL MAKEAX( N, M1, AA, AB, AC, X1, X2 )
23          S = 0.0
24          DO 130 I = 1, N
25  130         S = S + X1(I)*X2(I)
26          RERR = ABS(S-LAM1)/S
27          LAM1 = S
28          IF( RERR.LT.EPSEI6 )  GO TO 1001
29  100   CONTINUE
   *
   *        逆反復法
30  1001  CALL RANDOM( N, X1(1) )
31        LAM2 = LAM1
32        KLIMI1 = KLIMIT/10
33        RERR = 1.0
34        DO 200 K0 = 1, KKE2
35          S = 0.0
36          DO 210 I = 1, N
37            X2(I) = X1(I)
38  210         S = S + X2(I)*X2(I)
39          DO 220 I = 1, N
40            X2(I) = X2(I)/SQRT(S)
41  220         X1(I) = X2(I)*LAM2
42          EPSL1 = MAX( MIN( RERR*1.0D-2, 1.0D-3 ), 1.0D-8 )
43          CALL PCG11( AA, AB, AC, DINV, P, R, W, X2, X1, AP,
     +                EPSL1, ERR, M1, N, KCOUNT, KLIMI1, BNORM, IERR )
44          S = 0.0
45          DO 230 I = 1, N
46  230       S = S+X1(I)*X2(I)
47          RERR = ABS(S-LAM2)/S
48          LAM2 = S
49          IF( RERR.LT.EPSEI6 )  GO TO 2001
50  200   CONTINUE
51  2001  COND = LAM1*LAM2
52      END IF
   *
   *        ＩＣＣＧ（1,1）で方程式を解く
53      CALL PCG11( AA, AB, AC, DINV, P, R, W, B0, X, AP,
     +          EPSLON, ERR, M1, N, KCOUNT, KLIMIT, BNORM, IERR )
54      RETURN
55      END
```

（18）（M）ＩＣＣＧ（１，１）を用いて方程式を解く

```
   *
   *        ＩＣＣＧ(1,1)法      （５点差分）
   *
1       SUBROUTINE PCG11( AA, AB, AC, DINV, P, R, W, B, X, AP,
     +             EPSLON, ERR, M1, N, KCOUNT, KLIMIT, BNORM, IERR )
2       IMPLICIT REAL*8 (A-H,O-Z)
3       DIMENSION AA(N), AB(-M1:N), AC(-M1:N), DINV(-M1:N),
     +          P(1-M1:N+M1), W(1-M1:N+M1), B(N), X(1-M1:N+M1),
     +          AP(N), R(N)
   *
4       DO 100 I = 1-M1, 0
5         P(I) = 0.0D0
6   100   W(I) = 0.0D0
7       DO 110 I = N+1, N+M1
8         P(I) = 0.0D0
9   110   W(I) = 0.0D0
```

```
*
*          初期残差ベクトルとBノルムの計算
10       BNORM = 0.0D0
11       CALL MAKEAX( N, M1, AA, AB, AC, X, R )
12       DO 120 I = 1, N
13         R(I) = B(I)-R(I)
14   120   BNORM = BNORM+B(I)*B(I)
*
15       CALL LDU11N( N, M1, W, R, AB, AC, DINV )
*
16       RR1 = 0.0D0
17       DO 130 I = 1, N
18         P(I) = W(I)
19         RR1 = RR1+R(I)*W(I)
20   130 CONTINUE
*
*          反復計算
*
21       DO 1000 K = 1, KLIMIT
*
22         CALL MAKEAX( N, M1, AA, AB, AC, P, AP )
23         RAP = 0.0D0
24         DO 200 I = 1, N
25   200     RAP = RAP+P(I)*AP(I)
*
26         IF( ( ABS(RAP).EQ.0.0 ).OR.( ABS(RR1).EQ.0.0D0 ) ) THEN
27           IERR = 1
28           KCOUNT = K
29           RETURN
30         END IF
*
*          解ベクトルと残差ベクトルの更新、ノルムの計算
31         ALPHA = RR1/RAP
32         ERR = 0.0D0
33         DO 210 I = 1, N
34           X(I) = X(I)+ALPHA*P(I)
35           R(I) = R(I)-ALPHA*AP(I)
36           ERR = ERR+R(I)*R(I)
37   210     CONTINUE
38         ERR = SQRT(ERR/BNORM)
*
*          収束判定
39         IF( ERR.LT.EPSLON )  THEN
40           IERR = 0
41           KCOUNT = K
42           RETURN
43         END IF
*
44         CALL LDU11N( N, M1, W, R, AB, AC, DINV )
*
*          修正ベクトルの更新
45         RR2 = 0.0D0
46         DO 220 I = 1, N
47   220     RR2 = RR2+R(I)*W(I)
*
48         BETA = RR2/RR1
49         RR1 = RR2
50         DO 230 I = 1, N
51   230     P(I) = W(I)+BETA*P(I)
52  1000 CONTINUE
*
53       IERR = 2
54       KCOUNT = KLIMIT
55       RETURN
56       END
```

(19) 不完全U^TDU（コレスキー）分解（1，1）

```
*
*         不完全コレスキー分解(1,1)
*
      SUBROUTINE JCCG11( N, M1, A, B, C, DINV, UP, DMIN )
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(N), B(-M1:N), C(-M1:N), DINV(-M1:N)
*
      DMIN = 4.0D0
      DO 10 I = -M1, 0
   10   DINV(I) = 0.0D0
*
      M = M1-1
      DO 100 L = 0, M1+N/M1-2
*VOPTION INDEP(DINV)
        DO 100 I = MAX((L-M+1)*M1,L+1), MIN(L*M1+1,N), M1-1
          WK = A(I)-B(I-1)*B(I-1)*DINV(I-1)
     +             -C(I-M1)*C(I-M1)*DINV(I-M1)
     +             -( B(I-1)*C(I-1)*DINV(I-1)
     +               +B(I-M1)*C(I-M1)*DINV(I-M1) )*UP
          DINV(I) = 1.0D0/WK
          DMIN = MIN( DMIN, WK )
  110   CONTINUE
  100 CONTINUE
      RETURN
      END
```

(20) $(U^TDU)^{-1}$ をベクトルに作用させる（1，1）

```
*
*         (LDU)－1を作用させる
*
      SUBROUTINE LDU11N( N, M1, W, R, B, C, DINV )
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION W(1-M1:N+M1), R(N), DINV(-M1:N), B(-M1:N), C(-M1:N)
*         前進代入
      M = M1-1
      DO 100 L = 0, M1+N/M1-2
*VOPTION INDEP(W)
        DO 100 I = MAX((L-M+1)*M1,L+1), MIN(L*M1+1,N), M1-1
          W(I) = ( R(I)-C(I-M1)*W(I-M1)-B(I-1)*W(I-1) )*DINV(I)
  110   CONTINUE
  100 CONTINUE
*
      DO 200 I = 1, N
  200   W(I) = W(I)/DINV(I)
*         後退代入
      DO 300 L = M1+N/M1-2, 0, -1
*VOPTION INDEP(W)
        DO 300 I = MAX((L-M+1)*M1,L+1), MIN(L*M1+1,N), M1-1
          W(I) = ( W(I)-B(I)*W(I+1)-C(I)*W(I+M1) )*DINV(I)
  310   CONTINUE
  300 CONTINUE
      RETURN
      END
```

(21) (M)ＩＣＣＧ（１，２）法

```
*
*        ＩＣＣＧ法      （５点差分）
*
      SUBROUTINE PCG220( AA, AB, AC, DINV, P, R, W, B, X, AP,
     +             EPSLON, ERR, M1, N, KCOUNT, KLIMIT, BNORM, UP,
     +             BW, AE, X1, X2, IOPT, COND )
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AA(N),AB(-M1:N),AC(-M1:N),DINV(-M1:N),P(1-M1:N+M1),
     +          W(1-M1:N+M1),B(N),X(1-M1:N+M1),AP(N),R(N),
     +          BW(-M1:N), AE(-M1:N), X1(1-M1:N+M1), X2(N)
      REAL *8 LAM1, LAM2
      DATA KKE1, KKE2, EPSEIG, EPSD / 50, 20, 1.0D-3, 0.22D-12 /
*
*        初期設定
*
    1 UP = MAX( 0.0D0, UP )
      CALL JCCG12( N, M1, AA, AB, BW, AC, AE, DINV, UP, DMIN )
      IF( ( UP.GT.0.0 ).AND.( DMIN.LE.EPSD ) )  THEN
        UP = UP-0.05
        GO TO 1
      END IF
*
      IF( ABS(IOPT).GT.2 )  THEN
*
*        べき乗法
        LAM1 = 0.0D0
        CALL RANDOM( N, X2 )
        DO 100 K0 = 1, KKE1
          S = 0.0
          DO 110 I = 1, N
            X1(I) = X2(I)
  110       S = S + X1(I)*X1(I)
          DO 120 I = 1, N
  120       X1(I) = X1(I)/SQRT(S)
          CALL MAKEAX( N, M1, AA, AB, AC, X1, X2 )
          S = 0.0
          DO 130 I = 1, N
  130       S = S + X1(I)*X2(I)
          RERR = ABS(S-LAM1)/S
          LAM1 = S
          IF( RERR.LT.EPSEIG )  GO TO 1001
  100   CONTINUE
*
*        逆反復法
 1001   CALL RANDOM( N, X1(1) )
        LAM2 = LAM1
        KLIMI1 = KLIMIT/10
        RERR = 1.0
        DO 200 K0 = 1, KKE2
          S = 0.0
          DO 210 I = 1, N
            X2(I) = X1(I)
  210       S = S + X2(I)*X2(I)
          DO 220 I = 1, N
            X2(I) = X2(I)/SQRT(S)
  220       X1(I) = X2(I)*LAM2
          EPSL1 = MAX( MIN( RERR*1.0D-2, 1.0D-3 ), 1.0D-8 )
          CALL PCG12( AA, AB, AC, DINV, P, R, W, X2, X1, AP,
     +                EPSL1, ERR, M1, N, KCOUNT, KLIMI1, BNROM,
     +                BW, AE, IERR )
          S = 0.0
          DO 230 I = 1, N
  230       S = S+X1(I)*X2(I)
          RERR = ABS(S-LAM2)/S
          LAM2 = S
          IF( RERR.LT.EPSEIG )  GO TO 2001
  200   CONTINUE
 2001   COND = LAM1*LAM2
      END IF
*
```

```
*         ＩＣＣＧ（1,2）で方程式を解く
53      CALL PCG12( AA, AB, AC, DINV, P, R, W,
     +            B, X, AP, EPSLON, ERR, M1, N,
     +            KCOUNT, KLIMIT, BNORM, BW, AE, IERR )
54      RETURN
55      END
```

（22）（Ｍ）ＩＣＣＧ（１，２）を用いて方程式を解く

```
*
*         ＩＣＣＧ(1,2)法      （５点差分）
*
 1      SUBROUTINE PCG12( AA, AB, AC, DINV, P, R, W, B, X, AP,
     +              EPSLON, ERR, M1, N, KCOUNT, KLIMIT, BNORM,
     +              BW, AE, IERR )
 2      IMPLICIT REAL*8 (A-H,O-Z)
 3      DIMENSION AA(N),AB(-M1:N),AC(-M1:N),DINV(-M1:N),P(1-M1:N+M1),
     +          W(1-M1:N+M1),B(N),X(1-M1:N+M1),AP(N),R(N),
     +          BW(-M1:N),AE(-M1:N)
*
 4      DO 100 I = 1-M1, 0
 5        P(I) = 0.0D0
 6  100   W(I) = 0.0D0
 7      DO 110 I = N+1, N+M1
 8        P(I) = 0.0D0
 9  110   W(I) = 0.0D0
*
*         初期残差ベクトルとＢノルムの計算
10      BNORM = 0.0D0
11      CALL MAKEAX( N, M1, AA, AB, AC, X, R )
12      DO 120 I = 1, N
13        R(I) = B(I)-R(I)
14  120   BNORM = BNORM+B(I)*B(I)
*
15      CALL LDU12N( N, M1, W, R, BW, AC, AE, DINV )
*
16      RR1 = 0.0D0
17      DO 130 I = 1, N
18        P(I) = W(I)
19        RR1 = RR1+R(I)*W(I)
20  130 CONTINUE
*
*         反復計算
*
21      DO 1000 K = 1, KLIMIT
*
22        CALL MAKEAX( N, M1, AA, AB, AC, P, AP )
23        RAP = 0.0D0
24        DO 200 I = 1, N
25  200     RAP = RAP+P(I)*AP(I)
*
26        IF( ( ABS(RAP).EQ.0.0 ).OR.( ABS(RR1).EQ.0.0D0 ) )  THEN
27          IERR = 1
28          KCOUNT = K
29          RETURN
30        END IF
*
*         解ベクトルと残差ベクトルの更新、ノルムの計算
31        ALPHA = RR1/RAP
32        ERR = 0.0D0
33        DO 210 I = 1, N
34          X(I) = X(I)+ALPHA*P(I)
35          R(I) = R(I)-ALPHA*AP(I)
36          ERR = ERR+R(I)*R(I)
37  210   CONTINUE
38        ERR = SQRT(ERR/BNORM)
*
*         収束判定
39        IF( ERR.LT.EPSLON )  THEN
40          IERR = 0
```

```
41            KCOUNT = K
42            RETURN
43          END IF
  *
44          CALL LDU12N( N, M1, W, R, BW, AC, AE, DINV )
  *
  *           修正ベクトルの更新
45          RR2 = 0.0D0
46          DO 220 I = 1, N
47   220      RR2 = RR2+R(I)*W(I)
  *
48          BETA = RR2/RR1
49          RR1 = RR2
50          DO 230 I = 1, N
51   230      P(I) = W(I)+BETA*P(I)
52  1000 CONTINUE
  *
53        IERR = 2
54        KCOUNT = KLIMIT
55        RETURN
56        END
```

(23) 不完全$U^{T}DU$（コレスキー）分解（1，2）

```
  *
  *           不完全コレスキー分解(1,2)
  *
1       SUBROUTINE JCCG12( N, M1, A, B, BW, C, E, DINV, UP, DMIN )
2       IMPLICIT REAL*8 (A-H,O-Z)
3       DIMENSION A(N), B(-M1:N), C(-M1:N), E(-M1:N), DINV(-M1:N),
      +           BW(-M1:N)
  *
4       DMIN = 4.0D0
5       DO 10 I = -M1, 0
6         BW(I) = 0.0D0
7         E(I) = 0.0D0
8    10   DINV(I) = 0.0D0
  *
9       M = M1-1
10      DO 100 L = 0, M1+2*N/M1-3
11        IF( L.LT.M )  THEN
12           IA = L+1
13        ELSE IF( MOD(L-M,2).EQ.0 )  THEN
14           IA = M1*((L-M)/2+1)
15        ELSE
16           IA = M1*((L-M+1)/2+1)-1
17        END IF
  *VOPTION INDEP(DINV,BW,E)
18        DO 110 I = IA, MIN( M1*INT(L/2)+MOD(L,2)+1, N ), M1-2
19          WK = A(I)-BW(I-1)*BW(I-1)*DINV(I-1)
      +             -C(I-M1)*C(I-M1)*DINV(I-M1)
      +             -E(I-M1+1)*E(I-M1+1)*DINV(I-M1+1)
      +             -( BW(I-1)*E(I-1)*DINV(I-1)
      +               +BW(I-M1+1)*E(I-M1+1)*DINV(I-M1+1) )*UP
20          DINV(I) = 1.0D0/WK
21          BW(I) = B(I)-C(I-M1+1)*E(I-M1+1)*DINV(I-M1+1)
22          E(I) = -BW(I-1)*C(I-1)*DINV(I-1)
23          DMIN = MIN( DMIN, WK )
24   110  CONTINUE
25   100 CONTINUE
26      RETURN
27      END
```

(24) $(U^T D U)^{-1}$ をベクトルに作用させる（1，2）

```
      *
      *        （ＬＤＵ）－１を作用させる
      *
     1      SUBROUTINE LDU12N( N, M1, W, R, B, C, E, DINV )
     2      IMPLICIT REAL*8 (A-H,O-Z)
     3      DIMENSION W(1-M1:N+M1), R(N), DINV(-M1:N), B(-M1:N),
           +          C(-M1:N), E(-M1:N)
      *        前進代入
     4      M = M1-1
     5      DO 100 L = 0, M1+2*N/M1-3
     6        IF( L.LT.M )  THEN
     7           IA = L+1
     8        ELSE IF( MOD(L-M,2).EQ.0 )  THEN
     9           IA = M1*((L-M)/2+1)
    10        ELSE
    11           IA = M1*((L-M+1)/2+1)-1
    12        END IF
    *VOPTION INDEP(W)
    13        DO 110 I = IA, MIN( M1*INT(L/2)+MOD(L,2)+1, N ), M1-2
    14          W(I) = ( R(I)-C(I-M1)*W(I-M1)-B(I-1)*W(I-1)
           +                 -E(I-M1+1)*W(I-M1+1) )*DINV(I)
    15   110   CONTINUE
    16   100 CONTINUE
      *
    17      DO 200 I = 1, N
    18   200   W(I) = W(I)/DINV(I)
      *        後退代入
    19      DO 300 L = M1+2*N/M1-3, 0, -1
    20        IF( L.LT.M )  THEN
    21           IA = L+1
    22        ELSE IF( MOD(L-M,2).EQ.0 )  THEN
    23           IA = M1*((L-M)/2+1)
    24        ELSE
    25           IA = M1*((L-M+1)/2+1)-1
    26        END IF
    *VOPTION INDEP(W)
    27        DO 310 I = IA, MIN( M1*INT(L/2)+MOD(L,2)+1, N ), M1-2
    28          W(I) = ( W(I)-B(I)*W(I+1)-C(I)*W(I+M1)
           +                 -E(I)*W(I+M1-1) )*DINV(I)
    29   310   CONTINUE
    30   300 CONTINUE
    31      RETURN
    32      END
```

(25) (M)ＩＣＣＧ（1，3）法

```
      *
      *        ＩＣＣＧ法      （５点差分）
      *
     1      SUBROUTINE PCG23D( AA, AB, AC, DINV, P, R, W, BO, X, AP,
           +             EPSLON, ERR, M1, N, KCOUNT, KLIMIT, BNORM, UP,
           +             BW, AE, AF, X1, X2, IOPT, COND )
     2      IMPLICIT REAL*8 (A-H,O-Z)
     3      DIMENSION AA(N),AB(-M1:N),AC(-M1:N),DINV(-M1:N),P(1-M1:N+M1),
           +          W(1-M1:N+M1),BO(N),X(1-M1:N+M1),AP(N),R(N),
           +          BW(-M1:N),AE(-M1:N),AF(-M1:N), X1(1-M1:N+M1), X2(N)
     4      REAL *8 LAM1, LAM2
     5      DATA KKE1, KKE2, EPSEIG, EPSD / 50, 20, 1.0D-3, 0.22D-12 /
      *
      *        初期設定
      *
     6    1 UP = MAX( 0.0D0, UP )
     7      CALL JCCG13( N, M1, AA, AB, BW, AC, AE, AF, DINV, UP, DMIN )
     8      IF( ( UP.G.0.0 ).AND.( DMIN.LE.EPSD ) )  THEN
     9         UP = UP-0.05
    10         GO TO 1
    11      END IF
      *
```

```
12       IF( ABS(IOPT).GT.2 )  THEN
  *
  *         べき乗法
13         LAM1 = 0.000
14         CALL RANDOM( N, X2 )
15         DO 100 KO = 1, KKE1
16           S = 0.0
17           DO 110 I = 1, N
18             X1(I) = X2(I)
19   110       S = S + X1(I)*X1(I)
20           DO 120 I = 1, N
21   120       X1(I) = X1(I)/SQRT(S)
22           CALL MAKEAX( N, M1, AA, AB, AC, X1, X2 )
23           S = 0.0
24           DO 130 I = 1, N
25   130       S = S + X1(I)*X2(I)
26           RERR = ABS(S-LAM1)/S
27           LAM1 = S
28           IF( RERR.LT.EPSEIG )  GO TO 1001
29   100   CONTINUE
  *
  *         逆反復法
30  1001   CALL RANDOM( N, X1(1) )
31         LAM2 = LAM1
32         KLIMI1 = KLIMIT/10
33         RERR = 1.0
34         DO 200 KO = 1, KKE2
35           S = 0.0
36           DO 210 I = 1, N
37             X2(I) = X1(I)
38   210       S = S + X2(I)*X2(I)
39           DO 220 I = 1, N
40             X2(I) = X2(I)/SQRT(S)
41   220       X1(I) = X2(I)*LAM2
42           EPSL1 = MAX( MIN( RERR*1.0D-2, 1.0D-3 ), 1.0D-8 )
43           CALL PCG13( AA, AB, AC, DINV, P, R, W, X2, X1, AP,
     +                 EPSL1, ERR, M1, N, KCOUNT, KLIMI1, BNORM,
     +                 BW, AE, AF, IERR )
44           S = 0.0
45           DO 230 I = 1, N
46   230       S = S+X1(I)*X2(I)
47           RERR = ABS(S-LAM2)/S
48           LAM2 = S
49           IF( RERR.LT.EPSEIG )  GO TO 2001
50   200   CONTINUE
51  2001   COND = LAM1*LAM2
52       END IF
  *
  *         ＩＣＣＧ（1,3）で方程式を解く
53       CALL PCG13( AA, AB, AC, DINV, P, R, W, B0, X, AP,
     +            EPSLON, ERR, M1, N, KCOUNT, KLIMIT,
     +            BNORM, BW, AE, AF, IERR )
54       RETURN
55       END
```

(26) (M)ＩＣＣＧ（１，３）を用いて方程式を解く

```
      *
      *          ＩＣＣＧ(1,3)法       （５点差分）
      *
      SUBROUTINE PCG13( AA, AB, AC, DINV, P, R, W, B, X, AP,
     +                  EPSLON, ERR, M1, N, KCOUNT, KLIMIT, BNORM,
     +                  BW, AE, AF, IERR )
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AA(N),AB(-M1:N),AC(-M1:N),DINV(-M1:N),P(1-M1:N+M1),
     +          W(1-M1:N+M1),B(N),X(1-M1:N+M1),AP(N),R(N),
     +          BW(-M1:N),AE(-M1:N),AF(-M1:N)
      *
      DO 100 I = 1-M1, 0
        P(I) = 0.0D0
  100   W(I) = 0.0D0
      DO 110 I = N+1, N+M1
        P(I) = 0.0D0
  110   W(I) = 0.0D0
      *
      *          初期残差ベクトルとＢノルムの計算
      BNORM = 0.0D0
      CALL MAKEAX( N, M1, AA, AB, AC, X, R )
      DO 120 I = 1, N
        R(I) = B(I)-R(I)
  120   BNORM = BNORM+B(I)*B(I)
      *
      CALL LDU13N( N, M1, W, R, BW, AC, AE, AF, DINV )
      *
      RR1 = 0.0D0
      DO 130 I = 1, N
        P(I) = W(I)
        RR1 = RR1+R(I)*W(I)
  130 CONTINUE
      *
      *          反復計算
      *
      DO 1000 K = 1, KLIMIT
      *
        CALL MAKEAX( N, M1, AA, AB, AC, P, AP )
        RAP = 0.0D0
        DO 200 I = 1, N
  200     RAP = RAP+P(I)*AP(I)
      *
        IF( ( ABS(RAP).EQ.0.0 ).OR.( ABS(RR1).EQ.0.0 ) )  THEN
          IERR = 1
          KCOUNT = K
          RETURN
        END IF
      *
      *          解ベクトルと残差ベクトルの更新、ノルムの計算
        ALPHA = RR1/RAP
        ERR = 0.0D0
        DO 210 I = 1, N
          X(I) = X(I)+ALPHA*P(I)
          R(I) = R(I)-ALPHA*AP(I)
          ERR = ERR+R(I)*R(I)
  210   CONTINUE
        ERR = SQRT(ERR/BNORM)
      *
      *          収束判定
        IF( ERR.LT.EPSLON )  THEN
          IERR = 0
```

```
41            KCOUNT = K
42            RETURN
43          END IF
   *
44          CALL LDU13N( N, M1, W, R, BW, AC, AE, AF, DINV )
   *
   *           修正ベクトルの更新
45          RR2 = 0.0D0
46          DO 220 I = 1, N
47    220     RR2 = RR2+R(I)*W(I)
   *
48          BETA = RR2/RR1
49          RR1 = RR2
50          DO 230 I = 1, N
51    230     P(I) = W(I)+BETA*P(I)
52   1000 CONTINUE
   *
53        IERR = 2
54        KCOUNT = KLIMIT
55        RETURN
56        END
```

(27) 不完全U^T D U (コレスキー) 分解 (1, 3)

```
   *
   *           不完全コレスキー分解(1,3)
   *
 1        SUBROUTINE JCCG13( N, M1, A, B, BW, C, E, F, DINV, UP, DMIN )
 2        IMPLICIT REAL*8 (A-H,O-Z)
 3        DIMENSION A(N), B(-M1:N), C(-M1:N), E(-M1:N), DINV(-M1:N),
        +          BW(-M1:N), F(-M1:N)
   *
 4        DMIN = 4.0D0
 5        DO 10 I = -M1, 0
 6          BW(I) = 0.0D0
 7          E(I) = 0.0D0
 8          F(I) = 0.0D0
 9     10   DINV(I) = 0.0D0
   *
10        M = M1-1
11        DO 100 L = 0, M1+3*N/M1-4
12          IF( L.LT.M )  THEN
13            IA = L+1
14          ELSE IF( MOD(L-M,3).EQ.0 )  THEN
15            IA = M1*INT((L-M)/3)+M1
16          ELSE
17            IA = M1*INT((L-M)/3)+2*M1+MOD(L-M,3)-3
18          END IF
   *VOPTION INDEP(DINV,BW,E,F)
19          DO 110 I = IA, MIN(M1*INT(L/3)+MOD(L,3)+1,N), M1-3
20            WK = A(I)-BW(I-1)*BW(I-1)*DINV(I-1)
        +               -C(I-M1)*C(I-M1)*DINV(I-M1)
        +               -E(I-M1+1)*E(I-M1+1)*DINV(I-M1+1)
        +               -F(I-M1+2)*F(I-M1+2)*DINV(I-M1+2)
        +               -( F(I-M1+2)*C(I-M1+2)*DINV(I-M1+2)
        +                 +F(I-M1)*C(I-M1)*DINV(I-M1) )*UP
        +               -( BW(I-1)*F(I-1)*DINV(I-1)
        +                 +BW(I-M1+2)*F(I-M1+2)*DINV(I-M1+2) )*UP
21            DINV(I) = 1.0D0/WK
22            BW(I) = B(I)-C(I-M1+1)*E(I-M1+1)*DINV(I-M1+1)
        +                 -E(I-M1+2)*F(I-M1+2)*DINV(I-M1+2)
23            E(I) = -BW(I-1)*C(I-1)*DINV(I-1)
24            F(I) = -BW(I-1)*E(I-1)*DINV(I-1)
25            DMIN = MIN( DMIN, WK )
26    110   CONTINUE
27    100 CONTINUE
28        RETURN
29        END
```

(28) $(U^T D U)^{-1}$ をベクトルに作用させる (1, 3)

```
*
*        (LDU)－1を作用させる
*
      SUBROUTINE LDU13N( N, M1, W, R, B, C, E, F, DINV )
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION W(1-M1:N+M1), R(N), DINV(-M1:N), B(-M1:N),
     +          C(-M1:N), E(-M1:N), F(-M1:N)
*        前進代入
      M = M1-1
      DO 100 L = 0, M1+3*N/M1-4
        IF( L.LT.M )  THEN
          IA = L+1
        ELSE IF( MOD(L-M,3).EQ.0 )  THEN
          IA = M1*INT((L-M)/3)+M1
        ELSE
          IA = M1*INT((L-M)/3)+2*M1+MOD(L-M,3)-3
        END IF
*VOPTION INDEP(W)
        DO 110 I = IA, MIN(M1*INT(L/3)+MOD(L,3)+1,N), M1-3
          W(I) = ( R(I)-C(I-M1)*W(I-M1)-B(I-1)*W(I-1)
     +                -E(I-M1+1)*W(I-M1+1)
     +                -F(I-M1+2)*W(I-M1+2) )*DINV(I)
  110   CONTINUE
  100 CONTINUE
*
      DO 200 I = 1, N
  200   W(I) = W(I)/DINV(I)
*        後退代入
      DO 300 L = M1+3*N/M1-4, 0, -1
        IF( L.LT.M )  THEN
          IA = L+1
        ELSE IF( MOD(L-M,3).EQ.0 )  THEN
          IA = M1*INT((L-M)/3)+M1
        ELSE
          IA = M1*INT((L-M)/3)+2*M1+MOD(L-M,3)-3
        END IF
*VOPTION INDEP(W)
        DO 310 I = IA, MIN(M1*INT(L/3)+MOD(L,3)+1,N), M1-3
          W(I) = ( W(I)-B(I)*W(I+1)-C(I)*W(I+M1)
     +                -E(I)*W(I+M1-1)
     +                -F(I)*W(I+M1-2) )*DINV(I)
  310   CONTINUE
  300 CONTINUE
      RETURN
      END
```

(29) 対称疎行列をベクトルにかける

```
*
*                  行列Aにベクトルxを掛ける
*
      SUBROUTINE MAKEAX( N, M1, A, B, C, X, AX )
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(N), B(-M1:N), C(-M1:N), X(1-M1:N+M1), AX(N)
*
      DO 100 I = 1, N
        AX(I) = A(I)*X(I)+B(I)*X(I+1)+C(I)*X(I+M1)
     +         +B(I-1)*X(I-1)+C(I-M1)*X(I-M1)
  100 CONTINUE
      RETURN
      END
```

(30) 差分法による離散化

```
*
*        ひもを作る
*
      SUBROUTINE INPUT(N,M1,A,B,C,WK,DF0,DF1,DF2,DF3)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION A(N), B(-M1:N), C(-M1:N), WK(0:N+M1)
*
      DO 100 J = 0, N+M1
  100   WK(J) = 1.0D0
      DO 105 J = 1, N
  105   A(J) = 0.0
      DO 110 J = -M1, N
        B(J) = 0.0
  110   C(J) = 0.0
      DO 120 I = 1, M1*2
  120   WK(I) = DF1
      DO 130 I = M1*(M1+1)+1, M1*(M1+3)
  130   WK(I) = DF2
      DO 135 I = M1*(2*M1+2)+1, M1*(2*M1+3)+M1
  135   WK(I) = DF3
      WK(0) = DF0
      DO 140 I = M1, N+M1, M1
  140   WK(I) = DF0
*
      DO 160 I = 1, N
        A(I) = WK(I-1)+WK(I+M1-1)+WK(I+M1)+WK(I)
        B(I) = -(WK(I+M1)+WK(I))*0.5D0
  160   C(I) = -(WK(I+M1-1)+WK(I+M1))*0.5D0
      DO 165 I = N-M1+1, N
  165   C(I) = 0.0
      RETURN
      END
```

(31) 乱数生成

```
*
      SUBROUTINE RANDOM( N, Y )
      REAL *8 Y(N)
      DATA IBP31M,BNM31 /2147483647,Z39200000/
      DATA IVURN,IFP11  /584287,48828125/
*
      DO 100 I = 1, N
      IVURN = IVURN*IFP11
      IF( IVURN.LT.0 ) IVURN = (IVURN+IBP31M)+1
      X = FLOAT(IVURN)*BNM31
      Y(I) = X
  100 CONTINUE
      RETURN
      END
```