2001.8.1

数値シミュレーションの基礎(1)

青木 孝
神奈川大学 理学部 情報科学科

Introduction to Numerical Simulation(1)

Takashi Aoki
Department of Information Science, Kanagawa University

Abstract. I introduce to Numerical Simulation through thermal equation.
· Control Volume(CV) method
· Band symmetric Gauss
· ICCG(Incomplete Choleski Conjugate Gradient)

1 はじめに

村田は、数値シミュレーションを、次の三段階からなるとみよ、と言う(文献[1])。
(1) 現象のモデル化：偏微分方程式の問題に表現
(2) 上記問題を解く：離散化と数値解法
(3) 得られた結果を目的に照らして評価する：モデルの当否、解法の当否を含む
そして、これら一連のプロセスが完結して、はじめて数値シミュレーションも完結するという特徴を持つ。以上の事情を、簡単な、板の温度分布 $u = u(x, y)$ の定常解（時間依存がない）を例にとって解説する。

2 CV 法

板の温度分布 $u = u(x, y)$ を知りたい。元になるのは、熱方程式となる。

$$div(-\kappa \nabla u) = f \tag{2.1}$$

$\kappa(x, y)$ は熱伝導率、$f(x, y)$ は熱源分布を表わす（境界条件は、シミュレーション場の所で後述する）。

この方程式 (2.1) は、普通には、初期値·境界値問題を設定し、それを差分法によって空間差分化して常微分方程式を作り、台形法 (クランク-ニコルソン) によって解く。この時、式 (2.1) は、

$$-\kappa(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = f' \tag{2.2}$$

と書かれることが多い。この式 (2.2) は、たとえばκに場所依存性、温度依存性がある場合 ($\kappa = \kappa(x, y, u)$) には間違いである。κは温度勾配 ∇u と直接くっついていなければならない。

$$\frac{\partial}{\partial x}(-\kappa \frac{\partial u}{\partial x}) + \frac{\partial}{\partial y}(-\kappa \frac{\partial u}{\partial y}) = f' \tag{2.3}$$

この式(2.3)に基づいて離散化したモデルの解は、**式(2.2)の解とは違ったものになる。**

なぜ、そのような誤りをするかと言うと、熱方程式のモデル化の元になっている考え方：熱の保存則を忘れて、微分方程式がひとり歩きしてしまったことによる。実は、式(2.1)は、

「領域Qの表面Γからの流出量$+\int_\Gamma \boldsymbol{\Phi}\cdot\boldsymbol{n}ds$は、領域$Q$内での発生量$\int_\Omega f\ dv$と等しい。」

という物理量の保存則を表している。

$$\int_\Gamma \boldsymbol{\Phi}\cdot\boldsymbol{n}ds = \int_\Omega f\ dv \tag{2.4}$$

$\boldsymbol{\Phi}$は単位面積当たり、単位時間当たりの熱量の流量ベクトルを表わし、$\boldsymbol{n}$はΓ上に立てた外向き単位法線ベクトルを表わす。dsは面積要素、dvは体積要素。

式(2.4)を保存則の積分形式という。左辺を、ガウスの発散公式を使って、

$$\int_\Gamma \boldsymbol{\Phi}\cdot\boldsymbol{n}ds = \int_\Omega div\boldsymbol{\Phi}dv \tag{2.5}$$

と変形すると、式(2.4)は、

$$\int_\Omega div\boldsymbol{\Phi}dv = \int_\Omega f\ dv \tag{2.6}$$

となる。ここで、熱は温度勾配$\nabla u(x,y)$に比例して、高温部から低温部に流れる：

$$\Phi = -\kappa\nabla u \quad (\kappa\text{は熱伝導率})$$

とすれば、

$$\int_\Omega div(-\kappa\nabla u)\ dv = \int_\Omega f\ dv \tag{2.7}$$

となり、積分をはずせば、式(2.1)が出る。

村田は、前に述べた、保存則のようなモデルの元になるような考え方を、忘れてしまうような過ちをしないよう、

「積分形式保存則を満足するように差分化するCV(Control Volume)法」

を奨めている。シミュレーションでは、収束判定値にもよるが、0.001%以下で、保存則を満足させることはたやすい。

具体的には、式(2.4)に基づき$\Phi = -\kappa\nabla u$とした次式で、空間離散化を行なう。

$$\int_\Gamma(-\kappa\nabla u)\cdot\boldsymbol{n}\ ds = \int_\Omega f\ dv \tag{2.8}$$

次に、縦10cm(y方向) 横11cm(x方向)の長方形の例で、離散化の手順を説明する。今Fig.1のような10cm×11cm四方の板をメッシュに分け、各格子点での温度分布を考える。Fig.1では、メッシュを横(x)、縦(y)方向とも1cmきざみで均等にとっているが、実務上、大事な所（熱源の境界など）はメッシュを精しくして解くので、不均等あみ目となる。格子点には、Fig.1のように通し番号を付ける。**全格子点数が方程式の元数となり**、Fig.1では、$N = M(y\text{方向}) \times M1(x\text{ 方向}) = 10\times 10 = 100$点となる。$i$番目の格子点と、隣あう十字方向の4点はそれぞれ、$i+1$, $i-1$, $i+m$, $i-m$点となっている。この例では、境界条件は、Fig.1のDA, AB, AC境界を固定境界とし、簡単のために温度$u = 0[C]$と与える。上側のDC境界は、ノイマン境界($\kappa\nabla u = 0 : flux$が0)とする。

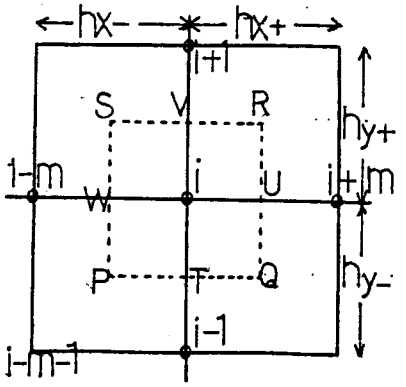

Fig.2. Mesh for CV method

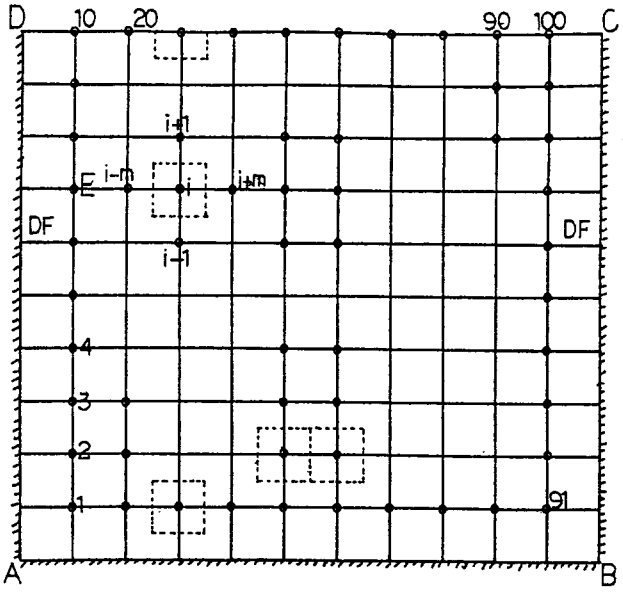

Fig.1. Simulation Field

このFig.1の各節(格子)点iのi点回りに、小領域PQRS(Control Volumeという)を作り、その周囲をΓに見立てて、式(2.8)のCV法による離散式を作る(Fig.2)。辺PQ, QR, RS, SPに立てた外向き単位法線ベクトルが、$(n_x, n_y)^T = (0,-1)^T,\ (1,0)^T,\ (0,1)^T,\ (-1,0)^T$であることに留意し、また、

$$\nabla u \cdot \boldsymbol{n} = (\frac{\partial u}{\partial x},\ \frac{\partial u}{\partial y})^T \cdot (n_x,\ n_y)^T$$

であることを使うと、左辺の線積分:

$$\int_\Gamma (-\kappa \nabla u)\cdot \boldsymbol{n}\, ds = \int_{PQ} + \int_{QR} + \int_{RS} + \int_{SP} \tag{2.9}$$

の各項は次のように書ける。

$$\int_{PQ} \doteq -d_P(\frac{u_i - u_{i-1}}{hy_-})(-1)\frac{hx_-}{2} - d_Q(\frac{u_i - u_{i-1}}{hy_-})(-1)\frac{hx_+}{2} \tag{2.10}$$

$$\int_{QR} \doteq -d_Q(\frac{u_{i+m} - u_i}{hx_+})(+1)\frac{hy_-}{2} - d_R(\frac{u_{i+m} - u_i}{hx_+})(+1)\frac{hy_+}{2} \tag{2.11}$$

$$\int_{RS} \doteq -d_R(\frac{u_{i+1} - u_i}{hy_+})(+1)\frac{hx_+}{2} - d_S(\frac{u_{i+1} - u_i}{hy_+})(+1)\frac{hx_-}{2} \tag{2.12}$$

$$\int_{SP} \doteq -d_S(\frac{u_i - u_{i-m}}{hx_-})(-1)\frac{hy_+}{2} - d_P(\frac{u_i - u_{i-m}}{hx_-})(-1)\frac{hy_-}{2} \tag{2.13}$$

ここで、T点での$\frac{\partial u}{\partial y}$は$\frac{u_i - u_{i-1}}{hy_-}$、W点での$\frac{\partial u}{\partial x}$は$\frac{u_i - u_{i-m}}{hx_-}$などと、各点での$u_i$を使って差分化している。また、伝導度$\kappa(x,y)$は、メッシュで区分けした面内は一定と考えて差分化する。たとえば、P点での$\kappa(x,y) \equiv d_P$は、4点$i,\ i-m,\ i-m-1,\ i-1$が囲む面内一定として、プログラムレベルでは$DF(i)$に格納するなど工夫する。配列DFの範囲は、$DF(N+M)$まで必要となる。

これら(2.10)〜(2.13)式を、$u_{i-m},\ u_{i-1},\ u_i,\ u_{i+1},\ u_{i+m}$項毎に整理すれば、第$i$番方程式:

$$a_{i,i-m}u_{i-m} + a_{i,i-1}u_{i-1} + a_{i,i}u_i + a_{i,i+1}u_{i+1} + a_{i,i+m}u_{i+m} = f_i \tag{2.14}$$

の係数(左辺)は、次式となる（式(2.8)の右辺=f_iは後述)。

$$a_{i,i} = \frac{1}{2}\left[d_P\left(\frac{hx_-}{hy_-}+\frac{hy_-}{hx_-}\right)+d_Q\left(\frac{hx_+}{hy_-}+\frac{hy_-}{hx_+}\right)+d_R\left(\frac{hx_+}{hy_+}+\frac{hy_+}{hx_+}\right)+d_S\left(\frac{hx_-}{hy_+}+\frac{hy_+}{hx_-}\right)\right]$$

$$a_{i,i-m} = -\frac{1}{2}\left[d_S\frac{hy_+}{hx_-}+d_P\frac{hy_-}{hx_-}\right]$$

$$a_{i,i-1} = -\frac{1}{2}\left[d_P\frac{hx_-}{hy_-}+d_Q\frac{hx_+}{hy_-}\right]$$

$$a_{i,i+1} = -\frac{1}{2}\left[d_R\frac{hx_+}{hy_+}+d_S\frac{hx_-}{hy_+}\right]$$

(2.15)
$$a_{i,i+m} = -\frac{1}{2}\left[d_Q\frac{hy_-}{hx_+}+d_R\frac{hy_+}{hx_+}\right]$$

式(2.14)の左辺は、式(2.10)～(2.13)を式(2.9)に代入したもので、式(2.14)は、行列方程式になっている($\mathbf{A}\ (a_{k,j})\mathbf{U}(u_k) = \mathbf{F}(f_k)$)。**Fig.1 の通し番号の節点** i $(=1 \rightarrow 100)$ は、

$i = m \times (j-1) + k \qquad (j = 1, m1,\ k = 1, m;\ m1$ は x **方向の点の数)**

などのように指定する。Fig.1 の場合は、はなはだ簡単に、$hx = hx_+ = hx_- = 1[cm]$, $hy = hy_+ = hy_- = 1[cm]$ である。今、$d_P = d_Q = d_R = d_S = 1[\frac{cal}{C \cdot sec \cdot cm}]$ とすれば、式(2.15)はそれぞれ、

(2.16) $a_{i,i} = 4.0 \quad a_{i,i-m} = -1.0 \quad a_{i,i-1} = -1.0 \quad a_{i,i+1} = -1.0 \quad a_{i,i+m} = -1.0$

となる。問題を簡単にするのは、$hx = hy = 1[cm]$ とすることより、むしろ $hx = hy$の均等あみ目とすることが主役である。その時、

$$a_{i,i} = \frac{1}{2}\left[2d_P + 2d_Q + 2d_R + 2d_S\right]$$

$$a_{i,i-m} = -\frac{1}{2}\left[d_S + d_P\right] \qquad a_{i,i-1} = -\frac{1}{2}\left[d_P + d_Q\right]$$

$$a_{i,i+1} = -\frac{1}{2}\left[d_R + d_S\right] \qquad a_{i,i+m} = -\frac{1}{2}\left[d_Q + d_R\right]$$

などとなる。実務上、避けて通れない不均等あみ目 ($hx_+ \neq hx_-$, $hy_+ \neq hy_-$) は、行列 $\mathbf{A}(a_{i,j})$ を解きづらくする一因となり、あみ目の取り方が、解いた結果に影響する。

村田の奨めるあみ目の取り方は、Fig.1 の $1[cm]$ を分割する分割数を MJとして、$1[cm]$ 毎に MJの値を変えてやる方法である。$1[cm]$ 毎に、$hx = \frac{1.0}{MJ}$が変わることになる。メッシュを倍に精しくしたければ、$MJ = 2$ とすればよい。プログラムレベルでは、節点 i 番(j列に当たる)の、たとえば hx_-, hx_+を、それぞれ $DHX(J)$, $DHX(J+1)$ に格納しておく(y も同様)。$1cm$ 毎の境界では、$hx_- \neq hx_+$となるわけである。Fig.1 で、MJを導入して均等あみ目 ($hx_- = hx_+$) とすると、

$M = MJ \cdot 10 \qquad M1 = MJ \cdot 11 - 1 \qquad N = M \cdot M1$

というあみ目になる。行列 $\mathbf{A}$ は、$n \times n$ 行列なので、メモリサイズは、$8 \times n^2$ Byte 必要。

これら係数も、上側のノイマン境界の節点 (CD 上) では、左辺の積分区間を Fig.2 の下半分だけにする必要があり、扱いが異なる。線積分は、

$$\int_{\Gamma\ CD}(-\kappa\nabla u)\cdot \boldsymbol{n}\ ds = \int_{WP}\ +\ \int_{PQ}\ +\ \int_{QU} \tag{2.17}$$

となり、CD 上の節点では、係数は (2.15) と同様の手順で、

$$a_{i,i} = \frac{1}{2}\left[d_P\left(\frac{hx_-}{hy_-}+\frac{hy_-}{hx_-}\right)+d_Q\left(\frac{hx_+}{hy_-}+\frac{hy_-}{hx_+}\right)\right]$$

$$a_{i,i-m} = -\frac{1}{2}\left[d_P\frac{hy_-}{hx_-}\right]$$

$$a_{i,i-1} = -\frac{1}{2}\left[d_P\frac{hx_-}{hy_-}+d_Q\frac{hx_+}{hy_-}\right]$$

$$a_{i,i+1} = 0.0$$

$$a_{i,i+m} = -\frac{1}{2}\left[d_Q\frac{hy_-}{hx_+}\right] \tag{2.18}$$

となる。$hx = hy = 1[cm]$、$d_P = d_Q = 1[\frac{cal}{C\cdot sec\cdot cm}]$ の下では、

$$a_{i,i} = 2.0 \quad a_{i,i-m} = -0.5 \quad a_{i,i-1} = -1.0 \quad a_{i,i+1} = 0.0 \quad a_{i,i+m} = -0.5 \tag{2.19}$$

となる。

ここでのシミュレーションでは、$\kappa = 1.0$ の一定では、面白くないので、Fig.1 の両側の伝導率κに段差をつける。

$i = 1 \rightarrow 10$ $\quad\quad$ $\kappa_i(x,y) = DF$(一定値)

$i = 101 \rightarrow 110$ $\quad\quad$ $\kappa_i(x,y) = DF$(一定値)

それ以外は、$\kappa_i(x,y) = 1.0$ とする。

u_iは節点 1〜100 までだが、κ_iは 1〜110 まであることに注意。この変数 DFを、$DF = 1.0[\frac{cal}{C\cdot sec\cdot cm}]$ にすれば、$\kappa = 1$ 一定の固定境界、$DF = 0.0$ にすれば、両サイドはノイマン境界 $(-\kappa\nabla u = 0)$ と同じになる。$i = 1 \rightarrow 10$ の各係数は、

$$a_{i,i} = \frac{1}{2}[DF\cdot 2\ +\ 2\ +\ 2\ +\ DF\cdot 2] = [2\ +\ 2DF]$$

$$a_{i,i+1} = -\frac{1}{2}[1\ +\ DF] \qquad a_{i,i+m} = 0.0 \tag{2.20}$$

となる。

また、節点 i に隣あう各 $i+1,\ i-1,\ i+m,\ i-m$ のいずれかに、固定境界がある場合には、違った扱いをする必要がある。例えば、左側に固定境界 $u = u_0(=0)$ を持つ E点 (Fig.1) の場合には、i 番方程式が、

$$a_{i,i-m}u_0 + a_{i,i-1}u_{i-1} + a_{i,i}u_i + a_{i,i+1}u_{i+1} + a_{i,i+m}u_{i+m} = f_i \tag{2.21}$$

となる。この時、未知数項でない$a_{i,i-m}u_0$は、右辺に移項してf_iの仲間に入れる。したがって、

(2.22) $$a_{i,i-1}u_{i-1} + a_{i,i}u_i + a_{i,i+1}u_{i+1} + a_{i,i+m}u_{i+m} = f_i - a_{i,i-m}u_0$$

を解くことになるので、係数は、

$a_{i,i-m} = 0.0$

$a_{i,i-1}$, $a_{i,i}$, $a_{i,i+1}$, $a_{i,i+m}$は、式(2.15)と同じと設定する。$u_0 = 0$ならば、なおさら簡単で、右辺に手を加える必要はなくなる。

以上より、式(2.9)の左辺は、次の行列の積で表せる(Fig.3)。今、この**A**が、$a_{i,j} = a_{j,i}$の対称性（隣あうControl Volume境界で、連続であることからも明らか）を利用すると、各iについて、$a_{i,i}$, $a_{i,i+1}$, $a_{i,i+m}$だけ作れば十分だということになる。この時、

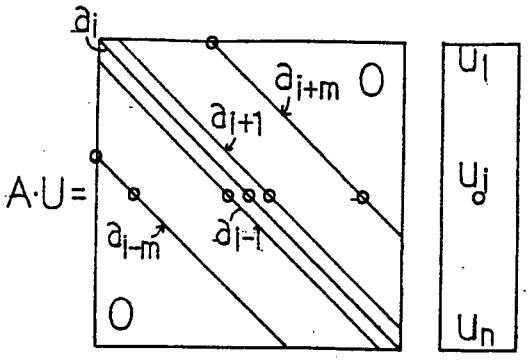

Fig.3. Band Matrix

(2.23) $$a_{i-m,i}u_{i-m} + a_{i-1,i}u_{i-1} + a_{i,i}u_i + a_{i,i+1}u_{i+1} + a_{i,i+m}u_{i+m} = f_i$$

を解くことになる。プログラムレベルでは、

$a_{i,i} = AA(I)$, $a_{i,i+1} = AB(I)$, $a_{i,i+m} = AC(I)$

と配列を用意し、それぞれの範囲を$AA(N)$, $AB(-M:N)$, $AC(-M:N)$ (−M:Nは−M～Nまでの事)とし、

$$AB(N) = 0.0, \quad A(0) = 0$$

(2.24) *do* $I = N - M + 1, N$ [$AC(I) = 0.0$] *do* $I = 1 - M, 0$ [$AC(I) = 0.0$]

と初期設定しておく工夫だけで、**A**·**U**が次式のように楽に計算できる。

do $I = 1, N$

[fi=AC(I-M)*U(I-M)+AB(I-1)*U(I-1)+AA(I)*U(I)+AB(I)*U(I+1)+AC(I)*U(I+M)]

(2.24a)

節点1について見ておく。節点1では、

(2.25) $$a_{1-m}u_0 + a_{1-1}u_0 + a_1u_1 + a_{1+1}u_{1+1} + a_{1+m}u_{1+m} = f_1$$

である。$a_{1-m} = 0$, $a_{1-1} = 0$としなければいけない。実際には、式(2.24a)から、

AC(1-M)*U(1-M)+AB(0)*U(0)+AA(1)*U(1)+AB(1)*U(2)+AC(1)*U(1+M)= fi

(2.25a)

となる。u_iの範囲は、$U(-M:N+M)$で、$U(-M)$～$U(0) = 0.0$、$U(N+1)$～$U(N+M) = 0.0$と始末する。$a_{1-m} = 0$, $a_{1-1} = 0$は、式(2.24)から満足している。

後回しにした、右辺の始末は、

$$f_i = \int_\Omega f \, dv$$

の面積分になる。Fig.2の面PQRSから、発生または吸収される熱量を4つの部分の合算で求める。伝導率κの場合のように、面(i, i-m, i-m-1, i-1)内は一定の熱源をもつとして離散

化するので、この面の熱源を、$f_P[\frac{cal}{sec\cdot cm^3}]$ とすれば、$\frac{1}{4}$面 (iWPT) に当たる f_iへの寄与は、$f_P\frac{hx_-}{2}\frac{hy_-}{2}[\frac{cal}{sec}]$（厚さが1[cm] あると思う）となる。

したがって、節点 i での f_iは、

$$f_i = f_P\frac{hx_-}{2}\frac{hy_-}{2} + f_Q\frac{hx_+}{2}\frac{hy_-}{2} + f_R\frac{hx_+}{2}\frac{hy_+}{2} + f_S\frac{hx_-}{2}\frac{hy_+}{2} \quad [\frac{cal}{sec}] \tag{2.26}$$

と求まる。$hx = hy = 1$ ならば、$f_i = \frac{1}{4}(f_p + f_Q + f_R + f_S)$。

当然、上側のノイマン境界上の節点 (CD 上) では、積分区間が下半分しかないので、

$$f_{i\ CD} = f_P\frac{hx_-}{2}\frac{hy_-}{2} + f_Q\frac{hx_+}{2}\frac{hy_-}{2} \quad [\frac{cal}{sec}] \tag{2.27}$$

となる。

式 (2.26) は、$hx = hy$均等あみ目で、$f_P = f_Q = f_R = f_S = fc$ ならば、

$$f_i = \frac{hx\ hy}{4}(f_p + f_Q + f_R + f_S) = fc \cdot hx\ hy$$

と簡単になる。

今回のシミュレーションでは、熱源を節点 i に対して、

i=42 to 44, i=52 to 54 では： $f_i = +0.2\ [\frac{cal}{sec}]$

i=46 to 48, i=56 to 58 では： $f_i = -0.2\ [\frac{cal}{sec}]$

と与える。プログラムレベルでは、面 (i, i-m, i-m-1, i-1) の熱源密度を FP(I) の配列に格納しておく。FP の範囲は、FP(N+M) まで必要で、伝導率κと同じ扱いをする。離散化時には、各節点 i での温度 u_iや、メッシュで区切った面での密度として扱う$\kappa(d_P)$, f_Pなどの物理量には、それぞれの扱いに特に注意を必要とする。

また、はじめに対称の熱源 f_iを与えるにはそれなりの意味がある。もし、離散化が間違っていれば、解は非対称になり、そのデバッグになる。

3 (対称) 帯ガウス (BSGLU)

かくして、式 (2.23) により、

$$\mathbf{A}(a_{i,j}) \cdot \mathbf{U}(u_i) = \mathbf{F}(f_i) \tag{3.1}$$

の n 行 n 列の連立一次方程式を解くという問題に帰着したわけである。

また、**A** は、各行 (i) につき、$\mid a_{i,i} \mid \geq \sum_{j\neq i} \mid a_{i,j} \mid$ になっており ((2.16) を見よ)、行対角優位になっている。**A** が、行対角優位の時には、ピボット (部分軸) 選択なしのガウス (消去法) で、上三角化でき、解けることが分かっている。部分軸選択があるなしで、計算時間が主部だけで倍違う。

数値シミュレーションとしての (3.1) の帯状の疎行列のソルバーとしては、基本的には対称帯ガウス（後述）を使い、それで実務的にメモリ、計算時間の点で負担が大きい場合には、この例のように対称正定値問題ならば、反復法である前処理つき共役勾配法 (ICCG, 後

述) を使うということになる。非対称ならば、双共役勾配法 (BCG) を改良したBCGSTABを使う (推奨)。

また、**AU=F** を解く時、**U=A^{-1}F** とはしない。逆行列 **A**$^{-1}$は理論的には重要だが、実際に計算するものではないことに注意すべきである。以上、これからのあらすじを述べた。順に説明する。

まず、ふつうのガウスから始める。ガウスの消去法の核 (主部) は、

「 k 番方程式 (軸方程式) に、$\alpha_{ik} = -a_{ik}/a_{kk}$をかけたものを、i 番方程式 ($i \geq k+1$) に加える： $a_{ij} = a_{ij} + \alpha_{ik} \cdot a_{kj}$; $j = k+1, n$ 」

ところである (Fig.4)。この操作を、どういう順番でやるかには任意性があって、2 つのやり方がある。

[1] 行ガウス（先に j を動かす；GLUR）

```
do i=k+1,n
    α = −a_ik/a_kk
    do j=k+1,n
        a_ij = a_ij + α·a_kj
```

[2] 列ガウス（先に i を動かす；GLU）

```
do j=k+1,n
    β = −a_kj/a_kk
    do i=k+1,n
        a_ij = a_ij + a_ik·β
```

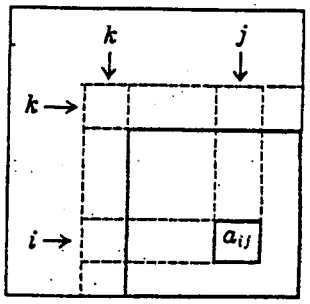

Fig.4 Gauss's elimination

FORTRAN の場合、a_{ij}は、主記憶に A(1,1),A(2,1),A(3,1)…A(1,2),A(2,2) のように、i の添字から順次隣の番地に格納される。したがって、FORTRAN では、最も内側のループ内で配列の左側の添字が動くようにし向ける列ガウスの方が良い。たとえ、配列 A(N×N×8 Byte) が主メモリに全て入る場合でも、列ガウスは、行ガウスより 3 倍程度速いとみるべきである。メモリに入りきらない大きさの配列の場合には、仮想メモリ方式計算機では、ページスワップなどのメモリの入れ替えが起こり、行ガウスは、列ガウスに比べて大きく不利になる。

AX=F を解く、軸選択あり (対角項に最大の数値を選ぶ) 列ガウスの全体プログラムを、念のため付録 1 にのせる。軸選択ありにした理由は、軸選択なしのガウス消去法が上三角行列の条件数をひどく大きくしうることがあるので、精度的に安心できないからである。軸選択すれば、精度的に安心である。また、対角優位の場合には、軸選択なしでも精度的に安心であることが分かっている。

付録 1 において、入力として、A(N,N) に係数を、F(N) に右辺をセットすると、F(N) に解が出力される。サブルーチン GLU は、**A** に対する前進消去で、サブルーチン GSLV は、**F** に対する前進消去と後退代入の部分である。(CLOCK0 と CLOCK は、CPU 時間計測のために作った。) **A**、**F** の設定は、今回のシミュレーションによって与えている。この行列 **A** の場合には、行対角優位なので軸選択が起こらないで計算できて、解を持つための必要十分条件：IR=0 を満たす。

プログラムレベルでは、列ガウスと行ガウスの違いは、次の部分の違いである。

[1] 行ガウス（GLUR）

```
            DO 130 J = K+1, N
              IF( IPK.NE.K )  THEN
                 W = A(IPK,J)
                 A(IPK,J) = A(K,J)
                 A(K,J) = W
              END IF
  130       CONTINUE
C
            DO 150 I = K+1, N
              T = A(I,K)
              DO 140 J = K+1, N
                A(I,J) = A(I,J)+T*A(K,J)
  140         CONTINUE
  150       CONTINUE
```

[2] 列ガウス（付録1: GLU）

```
            DO 130 J = K+1, N
              IF( IPK.NE.K )  THEN
                 W = A(IPK,J)
                 A(IPK,J) = A(K,J)
                 A(K,J) = W
              END IF
C
              T = A(K,J)
              DO 140 I = K+1, N
                A(I,J) = A(I,J)+A(I,K)*T
  140         CONTINUE
  130       CONTINUE
```

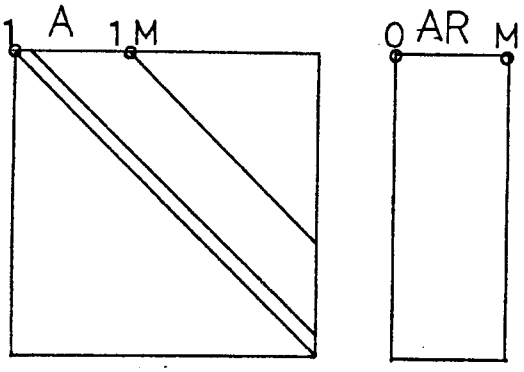

Fig.5 Dimension A(n,n), AR(0:m,n)

以上、列ガウスにしても、行列 A(N,N) をそのまま計算に使うので、メモリも計算時間も負担が大きすぎる。そこで、今の問題で扱う式 (2.23) のような帯状の疎配列の特徴をうまく利用して、特にメモリ使用量を減らす工夫をする。その基礎的な方法として (対称) 帯ガウスがある。

考え方としては、各 i 番方程式 (2.23) を、AR(0:M,N) の 2 次元配列を用意して、

$$AR(j-i,i) = a_{ij}$$

の規則で格納して (Fig.5)、この配列 AR() を使って列ガウスを解く。$AR(0,i) = a_{ii}$ となるように工夫している。プログラムレベルでは、式 (2.24a) の、AA(I), AB(I), AC(I) を、

do I=1,N

[AR(0,I)=AA(I) : AR(1,I)=AB(I) : AR(M,I)=AC(I)]

として埋め込むことになる。

このAR()を使えば、列ガウスの主部は、次のように書き換わる。

[3] 対称帯 列ガウス

do j=k+1, min(k+m,n)

$\beta = -ar_{j-k,k}/ar_{0,k}$

do i=j, min(k+m,n)

$ar_{i-j,j} = ar_{i-j,j} + \beta \cdot ar_{i-k,k}$

これは、列ガウスの $a_{kk} = ar_{0,k}$, $a_{ij} = ar_{j-i,i}$ などと、単に書き換えたにすぎない。この工夫により、所要メモリは列ガウスに比べ、n^2から$n \times m$のオーダーに減る。主部の積和回数も、n^2からm^2に減っている。今、n=100, m=10の場合に、A(100,100)は80kByteのメモリを要する。あみ目のきざみ幅を、hx=0.1[mm]にして、mを10倍精しくしただけで、Aは950M Byteのメモリを必要としてしまう。これでは、理論上はガウスで解けるが、実務上では解けない。

4 シミュレーション結果

全節までの方針をもりこみ、プログラム化すると次のようになる。

```
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*4 CPUT
      PARAMETER(MJ=1,M=MJ*10,M2=MJ*11-1,N=M*M2,EPS=1.0D-7)
      DIMENSION AR(0:M,N),AA(N),AB(-M:N),AC(-M:N),F(N)
C
      READ(5,*) DF
      WRITE(6,*) ' % DF= ',DF
      WRITE(6,*) ' % M= ',M,' M2= ',M2,' N= ',N
      FMJ= DFLOAT(MJ)
      DHX=1.0D0/FMJ
      DHY=1.0D0/FMJ
C
      DO 10 I=1,M-1
        AA(I)=2.0D0*(DF+1.0D0)
        AB(I)=-(1.0D0+DF)/2.0D0
        AC(I)=-1.0D0
   10 CONTINUE
      AA(M)=DF+1.0D0
      AB(M)=0.0D0
      AC(M)=-0.5D0
C
      DO 30 J=1,M2-2
        DO 20 K=1,M-1
          AA(M*J+K)=4.0D0
          AB(M*J+K)=-1.0D0
          AC(M*J+K)=-1.0D0
   20   CONTINUE
        AA(M*J+M)=2.0D0
        AB(M*J+M)=0.0D0
        AC(M*J+M)=-0.5D0
   30 CONTINUE
C
      DO 40 I=N-M+1, N-1
        AA(I)=2.0D0*(DF+1.0D0)
        AB(I)=-(1.0D0+DF)/2.0D0
        AC(I)=0.0D0
   40 CONTINUE
```

```
      AA(N)=DF+1.0D0
      AB(N)=0.0D0
      AC(N)=0.0D0
C
      DO 50 I=1,N
        AR(0,I)=AA(I)
        AR(1,I)=AB(I)
        AR(M,I)=AC(I)
   50 CONTINUE
C
*C    Uhen ***
      DO 60 J= 5*MJ-1, 6*MJ-1
        DO 70 I=0,2*MJ
          F(M*J +2*MJ+I)= 0.2D0*(DHX*DHY)
          F(M*J +6*MJ+I)=-0.2D0*(DHX*DHY)
   70   CONTINUE
   60 CONTINUE
C
      CALL CLOCK0
C
*C    AR*U=F wo Toku; Kotae --> F ***
      CALL BSGLU(N,M,AR,EPS,IR)
      CALL BSGSLV(N,M,AR,F)
C
      CALL CLOCK(CPUT)
      UMIN=F(1)
      JUMIN=1
      KUMIN=1
      UMAX=F(1)
      JUMAX=1
      KUMAX=1
      DO 130 J=1,M2
        DO 120 K=1,M
          I=M*(J-1)+K
          IF(F(I).LT.UMIN) THEN
            UMIN=F(I)
            JUMIN=J
            KUMIN=K
          ENDIF
          IF(F(I).GT.UMAX) THEN
            UMAX=F(I)
            JUMAX=J
            KUMAX=K
          ENDIF
  120   CONTINUE
  130 CONTINUE
C
      WRITE(6,*) ' % MJ= ',MJ
      WRITE(6,*) ' % CPUT= ',CPUT
      WRITE(6,*) ' % UMIN,(x,y)= ',UMIN,'(',JUMIN,',',KUMIN,')'
      WRITE(6,*) ' % UMAX,(x,y)= ',UMAX,'(',JUMAX,',',KUMAX,')'
C
      WRITE(6,*) ' z=[ '
C
CC    DO 80 K= M,MJ,-MJ
CC      WRITE(6,2000) (F(M*J+K),J=MJ-1,MJ*10-1,MJ),' ; '
CC 80 CONTINUE
*C    MATLAB You ***
      DO 80 K= MJ,M,MJ
        WRITE(6,2000) (F(M*J+K),J=MJ-1,MJ*10-1,MJ),' ; '
   80 CONTINUE
 2000 FORMAT(1H ,10F6.2,A3)
C
      WRITE(6,*) ' ]; '
      WRITE(6,*) ' contour(0:1:9, 0:1:9, z,10);'
```

```
C
      STOP
      END
CCC
      SUBROUTINE BSGLU(N,M,AR,EPS,IR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AR(0:M,N)
C
      IR=0
      DO 100 K=1,N
        IF(ABS(AR(0,K)).LE.EPS) THEN
         IR=IR+1
         RETURN
        ENDIF
C
        DO 120 I=K+1,MIN(K+M,N)
          T=-AR(I-K,K)/AR(0,K)
          DO 130 J=I,MIN(K+M,N)
130          AR(J-I,I)=AR(J-I,I)+T*AR(J-K,K)
120     CONTINUE
100   CONTINUE
      RETURN
      END
CCC
      SUBROUTINE BSGSLV(N,M,AR,B)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AR(0:M,N),B(N)
C
      DO 100 K=1,N
       BK=B(K)/AR(0,K)
       DO 110 I=K+1,MIN(K+M,N)
110       B(I)=B(I)-AR(I-K,K)*BK
100   CONTINUE
C
      DO 200 K=N,1,-1
        S=-B(K)
        DO 210 J=K+1,MIN(K+M,N)
210        S=S+AR(J-K,K)*B(J)
        B(K)=-S/AR(0,K)
200   CONTINUE
      RETURN
      END
```

ここで、サブルーチン BSGLU は、軸選択なしの対称帯ガウスで、BSGSLV は、右辺の計算である。それぞれ、列ガウスのサブルーチン GLU、GSLV に対応する。対称帯 列ガウス (BSGLU) の主部は、

```
        DO 120 I=K+1,MIN(K+M,N)
          T=-AR(I-K,K)/AR(0,K)
          DO 130 J=I,MIN(K+M,N)
130          AR(J-I,I)=AR(J-I,I)+T*AR(J-K,K)
120     CONTINUE
```

となる。(CLOCK0, CLOCK は付録1と同じ。)

また、メインプログラムの

WRITE(6,*) ' z=['

以下は、MATLAB により、u_i(F(I)) の温度分布を等高線表示するための準備となっている。これは、実行時にリダイレクションを利用し、MATLAB の M ファイル (この例では

elp.m) を作るためである。プログラムファイルを elp.f として、コンパイラを f77 とすると、DF=1.0, MJ=1 の下で、

%> f77 elp.f

%> elp > elp.m

1.0

とする。elp.m には、

```
% DF= 1.000000000000000
% M= 10  M2= 10  N= 100
% MJ= 1
% CPUT= 0.000000000e+00
% UMIN,(x,y)= -0.3525687318769837 (6 ,8 )
% UMAX,(x,y)= 0.2137456301207766 (6 ,3 )
z=[
 0.01  0.01  0.03  0.05  0.08  0.08  0.05  0.03  0.01  0.01 ;
 0.01  0.02  0.05  0.10  0.20  0.20  0.10  0.05  0.02  0.01 ;
 0.01  0.02  0.05  0.10  0.21  0.21  0.10  0.05  0.02  0.01 ;
 0.00  0.00  0.01  0.05  0.14  0.14  0.05  0.01  0.00  0.00 ;
-0.01 -0.03 -0.04 -0.05 -0.05 -0.05 -0.05 -0.04 -0.03 -0.01 ;
-0.03 -0.06 -0.10 -0.15 -0.25 -0.25 -0.15 -0.10 -0.06 -0.03 ;
-0.04 -0.09 -0.14 -0.22 -0.34 -0.34 -0.22 -0.14 -0.09 -0.04 ;
-0.05 -0.10 -0.16 -0.24 -0.35 -0.35 -0.24 -0.16 -0.10 -0.05 ;
-0.05 -0.11 -0.16 -0.22 -0.28 -0.28 -0.22 -0.16 -0.11 -0.05 ;
-0.05 -0.11 -0.16 -0.22 -0.26 -0.26 -0.22 -0.16 -0.11 -0.05 ;
];
contour(0:1:9, 0:1:9, z,10);
```

が入る。ここで、MATLAB 上での 2 次元配列 (Z) の 1 行目は、等高線の作図では下側になるので注意すること。実際に、各 u_iの温度を Fig.1 の格子点の並びで出力すると、左下が節点 1、右上が節点 100 で、

```
-0.05 -0.11 -0.16 -0.22 -0.26 -0.26 -0.22 -0.16 -0.11 -0.05
-0.05 -0.11 -0.16 -0.22 -0.28 -0.28 -0.22 -0.16 -0.11 -0.05
-0.05 -0.10 -0.16 -0.24 -0.35 -0.35 -0.24 -0.16 -0.10 -0.05
-0.04 -0.09 -0.14 -0.22 -0.34 -0.34 -0.22 -0.14 -0.09 -0.04
-0.03 -0.06 -0.10 -0.15 -0.25 -0.25 -0.15 -0.10 -0.06 -0.03
-0.01 -0.03 -0.04 -0.05 -0.05 -0.05 -0.05 -0.04 -0.03 -0.01
 0.00  0.00  0.01  0.05  0.14  0.14  0.05  0.01  0.00  0.00
 0.01  0.02  0.05  0.10  0.21  0.21  0.10  0.05  0.02  0.01
 0.01  0.02  0.05  0.10  0.20  0.20  0.10  0.05  0.02  0.01
 0.01  0.01  0.03  0.05  0.08  0.08  0.05  0.03  0.01  0.01
```

となっている。この elp.m を使い、62-101,103 ワークステーション上で作図する。

%WS > matlab　(と起動)

Command Window に MATLAB のプロンプト (>>) が出たら、elp と入力する。elp.m の M ファイルが読み込まれ、等高線が出力される。

%WS > elp

この等高線は、絵が出ている時に、

%WS > print -deps elp.eps

と入力すれば、eps ファイルに出力できる。実際に等高線を出力したものが、Fig.6 である。両サイドの熱伝導率が、DF=1.0$[\frac{cal}{C \cdot sec \cdot cm}]$ で、MJ=1 の時には、各節点中の、最小温度 (u_{min})、最大温度 (u_{max}) は、

$u_{min} = -0.3525[C]\ (x,y) = (6,8)$;　$u_{max} = +0.2137[C]\ (x,y) = (6,3)$

となる。いずれも、熱源付近になる。

これを、DF=0.0 とすると、左右の固定境界はノイマン境界と同等な条件となり、u_iは、

```
-0.49 -0.50 -0.52 -0.55 -0.58 -0.58 -0.55 -0.52 -0.50 -0.49
-0.48 -0.49 -0.52 -0.56 -0.60 -0.60 -0.56 -0.52 -0.49 -0.48
-0.45 -0.46 -0.50 -0.56 -0.66 -0.66 -0.56 -0.50 -0.46 -0.45
-0.41 -0.42 -0.45 -0.51 -0.62 -0.62 -0.51 -0.45 -0.42 -0.41
-0.34 -0.35 -0.37 -0.41 -0.50 -0.50 -0.41 -0.37 -0.35 -0.34
-0.26 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.27 -0.26
-0.19 -0.18 -0.16 -0.12 -0.04 -0.04 -0.12 -0.16 -0.18 -0.19
-0.12 -0.11 -0.08 -0.03  0.08  0.08 -0.03 -0.08 -0.11 -0.12
-0.07 -0.06 -0.04  0.02  0.11  0.11  0.02 -0.04 -0.06 -0.07
-0.03 -0.03 -0.01  0.01  0.04  0.04  0.01 -0.01 -0.03 -0.03
```

で、 $u_{min} = -0.6595[C]$ $(x,y) = (6,8)$; $u_{max} = +0.1133[C]$ $(x,y) = (6,2)$
と変わる。その時、等高線は Fig.7 となる。左右の境界では、ノイマン境界なので、等高線は境界に対して垂直に切れる。u_iの温度分布は、熱源に対して妥当な分布となっている。

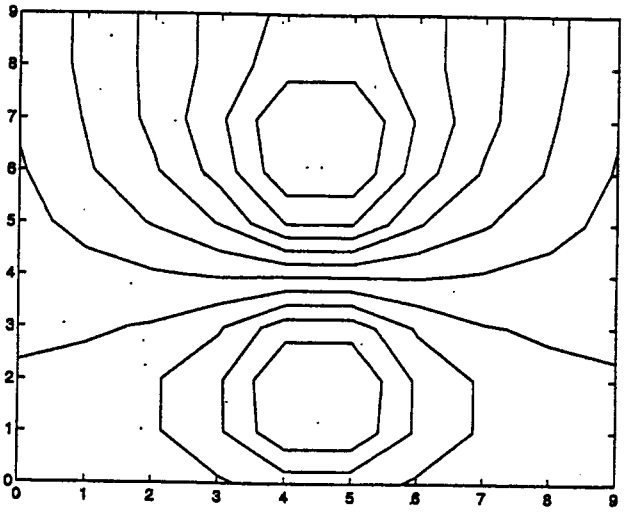

Fig.6 u_i distribution(DF=1.0,MJ=1)

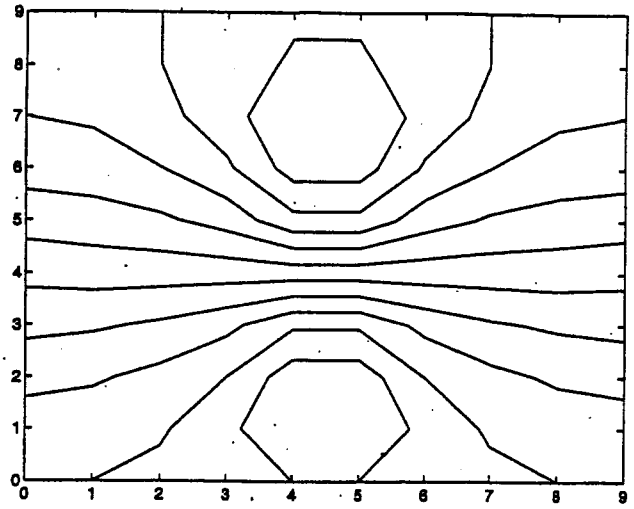

Fig.7 u_i distribution(DF=0.0,MJ=1)

最後に、DF=1.0 の場合で、u_{min}, u_{max}のあみ目 (MJ) 依存性を見ておく (Fig.8)。MJ 値によって、u_{min}, u_{max}の位置は、ほとんど変わらない。実測は、日立 FLORA 330 DK2(Pentium3) 933M Hz メモリ 256MByte で行なった。

通常、あみ目が精しくなるにつれて、値は小さくなる。Fig.8 によれば、MJ=20(hx=hy=1/20cm) の結果に対して、MJ=6 の解は u_{min}, u_{max}とも、まだ13%程度違っている。2%程度の精度が必要ならば、MJ=15 にしなければならない。

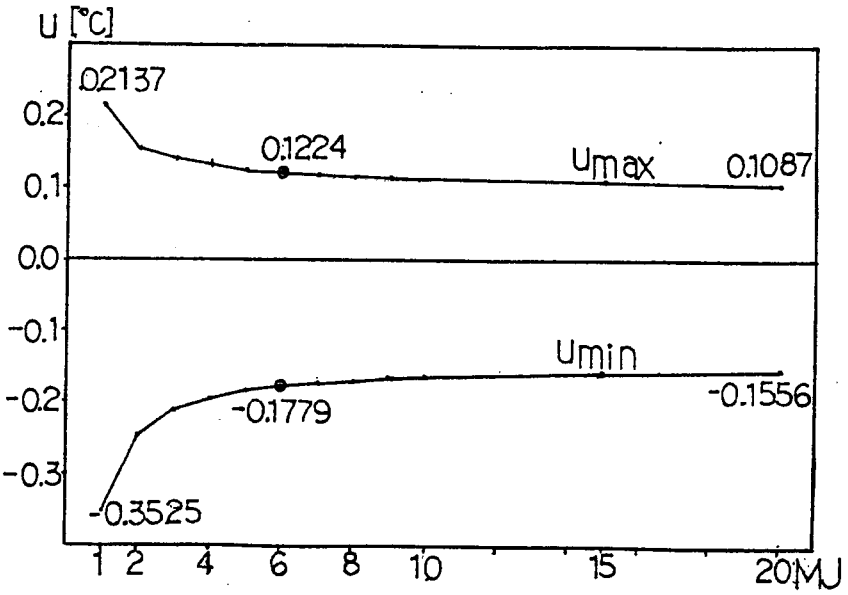

Fig.8 MJ dependence of u_{min}, u_{max}

Table 1 に、MJ を変えた時の、元数 n、配列 A, AR のサイズ (MB)、ソルバを対称帯列ガウス (BSGLU)、列ガウス (GLU)、行ガウス (GLUR) として解いた時の CPU 時間を示す。帯ガウスでは、MJ=6 まで 1 秒以内で解けるが、列ガウスは MJ=6 で 1018 秒もかかる。行ガウスでは、その 4.3 倍かかる。これでは、実務上、列ガウスは使えない。なお帯ガウスは、MJ=15 で 3 秒 (u_{min}=−0.1587[C]; u_{max}=+0.1105[C])、MJ=20 で 8 秒 (u_{min}=−0.1556[C]; u_{max}=+0.1087[C]) で計算し終る。

Table 1 Solver performance for MJ(mesh size)

MJ	u_{min}(C)	u_{max}(C)	N	A(MB)	AR(MB)	BSGLU(s)	GLU(s)	GLUR(s)
1	-0.3525	0.2137	100	0.08	0.008	0.	0.	0.
2	-0.2481	0.1620	420	1.4	0.067	0.	1.0	2.0
3	-0.2100	0.1407	960	7.3	0.23	0.	16	54
4	-0.1942	0.1325	1720	23	0.55	0.	87	164
5	-0.1837	0.1258	2700	58	1.0	0.	335	860
6	-0.1779	0.1224	3900	121	1.8	0.	1018	4451

5 ICCG(共役勾配法)

実務的には、メモリ、計算時間の負担を減らしたい一心で、直接法の帯ガウスの代わりに、反復解法である ICCG 法を使う。ただし、反復法は解の精度が問題になるので、帯ガウスは ICCG を組み込んだ際のデバッグにぜひとも必要である。

反復解法の原理を説明する (反復解法とは言うものの、始めはやはりガウスの消去法にたよる)。方程式:

(5.1) $$\mathbf{Ax} = \mathbf{b}$$

において、行列 $\mathbf{A}$ を、$\mathbf{A}$=$\mathbf{A}_0$−$\mathbf{R}$ と分離し、左辺の一部を右辺に移項して、

(5.2) $$\mathbf{A}_0\mathbf{x} = \mathbf{Rx} + \mathbf{b}$$

と書く。$\mathbf{A}_0$をガウスの消去法で LU 分解 ($\mathbf{A}_0$=$\mathbf{LU}$) することは容易とすると、反復式:

(5.3) $$\mathbf{A}_0\mathbf{x}^{(k)} = \mathbf{R}\mathbf{x}^{(k-1)} + \mathbf{b}$$

は、$\mathbf{A}_0$に対する LU 分解を一度行なっておけば、後は、各 k について右辺を計算し、その右辺に対して前進・後退代入を行なうだけで計算が進められる。この時、(5.2)(5.3) 式より、

(5.4) $$\mathbf{A}_0(\mathbf{x} - \mathbf{x}^{(k)}) = \mathbf{R}(\mathbf{x} - \mathbf{x}^{(k-1)})$$

すなわち、

(5.5) $$\begin{aligned} \mathbf{x} - \mathbf{x}^{(k)} &= \mathbf{A}_0^{-1}\mathbf{R}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\ &= (\mathbf{A}_0^{-1}\mathbf{R})^2(\mathbf{x} - \mathbf{x}^{(k-2)}) \quad \cdots \\ &= (\mathbf{A}_0^{-1}\mathbf{R})^k(\mathbf{x} - \mathbf{x}^{(0)}) \end{aligned}$$

この式を繰り返し、

この式の両辺のノルムをとると、

(5.6) $$\| \mathbf{x} - \mathbf{x}^{(k)} \| \leq \| \mathbf{A}_0^{-1}\mathbf{R} \|^k \cdot \| \mathbf{x} - \mathbf{x}^{(0)} \|$$

よって、$\| \mathbf{A}_0^{-1}\mathbf{R} \| \leq 1$ なら ($k \to \infty$ にて)、$\mathbf{x}^{(k)} \to \mathbf{x}$ が保証される。$\mathbf{A}=\mathbf{A}_0-\mathbf{R}$ と分離した時、$\mathbf{A}_0$ が $\mathbf{A}$ の主部を形成し、$\mathbf{A}_0$ と比べ $\mathbf{R}$ の部分がわずかであればある程、$\| \mathbf{A}_0^{-1}\mathbf{R} \|$ は、1 と比べ、より小さくなる。単純には、帯行列の 3 重対角部：$(a_{i,i-1},\ a_{i,i},\ a_{i,i+1})$ を $\mathbf{A}_0$ とし、外側の $(a_{i,i-m},\ a_{i,i+m})$ を $-\mathbf{R}$ に選ぶ方法があるが、それは、収束が遅く実用的でない。

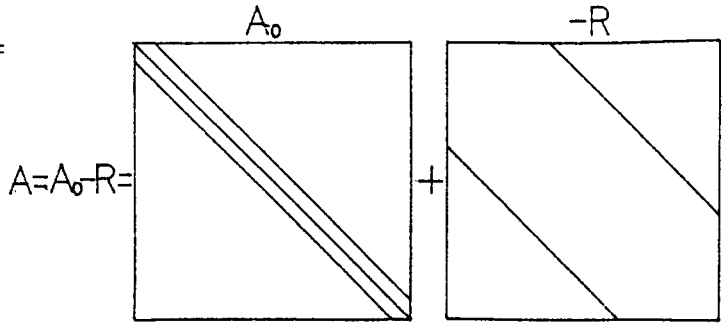

Fig.9 Simplest selection of $\mathbf{A}_0$, $\mathbf{R}$ (Bad)

もっとよい $\mathbf{A}_0$ と $\mathbf{R}$ を選ぶ方法がある。

$\mathbf{R}=\mathbf{A}_0-\mathbf{A}$ ($\mathbf{A}_0=\mathbf{L}\cdot\mathbf{U}$): 今は対称なので $\mathbf{U}$ だけ考えればよい。

$=\mathbf{L}\cdot\mathbf{U}-\mathbf{A}$ ただし、$\mathbf{L}$ の対角要素は 1

の i 行要素が、(i, i−m+2), (i, i+m−2) の所だけに生ずるような、$\mathbf{A}_0=\mathbf{L}\cdot\mathbf{U}$ の上三角行列 $\mathbf{U}$ を作る (Fig.10)。

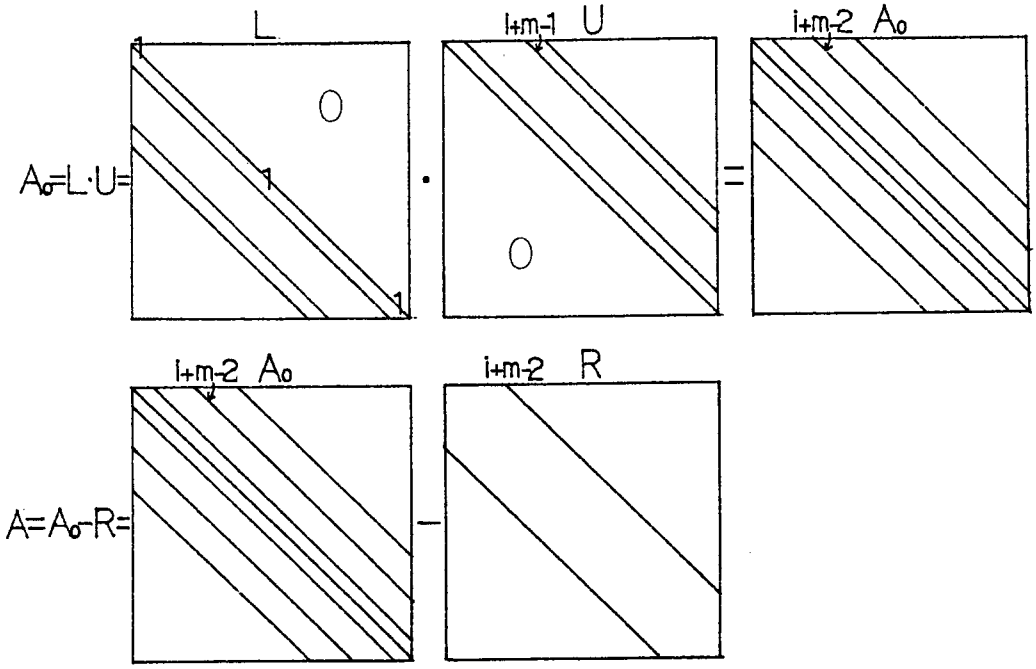

Fig.10 $\mathbf{A}_0$, $\mathbf{R}$ Set by Incomplete LU

この $\mathbf{L}\cdot\mathbf{U}$ は、元の行列 $\mathbf{A}$ にはならないことから不完全 LU 分解と呼ばれる。$\mathbf{U}$ の i 行要素は、

$u_{i,i}\ (=d_i^{-1})$, $u_{i,i+1}\ (=\tilde{b}_i)$, $u_{i,i+m-1}\ (=e_i)$, $u_{i,i+m}\ (=c_i$: 元のまま)

をもつ。(後述するが、この $u_{i,i+m-1}$ を使う不完全 LU 分解を前処理に使った ICCG 法を、特に ICCG(1,2) と名づける。さらに、内側にもう 1 本 $u_{i,i+m-2}$ を増やしたものは ICCG(1,3) と呼ぶ。)

元の行列 $\mathbf{A}$ の要素を、

$a_i=$ $\mathbf{A}$ の (i,i) 要素, $b_i=$ $\mathbf{A}$ の (i,i+1) 要素, $c_i=$ $\mathbf{A}$ の (i,i+m) 要素

とすると、

do i= 1,n

$u = a_i - c_{i-m}^2\, d_{i-m} - e_{i-m+1}^2\, d_{i-m+1}$

$d_i = 1/u$

$\tilde{b}_i = b_i - c_{i-m+1}\, e_{i-m+1}\, d_{i-m+1}$

$e_i = -c = i-1\ \tilde{b}_{i-1}\, d_{i-1}$ $\quad e_i = -c_{i-1}\,\tilde{b}_{i-1}\,d_{i-1}$

とすればよい。$u_{i,i+m}$は、元の **A** のまま:= c_i。e_iは、元の **A** では 0。この時、**R** の要素 (i,i-m+2) (i,i+m-2) は、元の行列 **A** が、

$a_i = 4, \quad a_{i+1} = -1, \quad a_{i+m} = -1$

という、ここで扱う標準行列の時、その値はかなり小さなものとなっている。プログラムは、次のようになる。

```
      SUBROUTINE ICDCMP( N,M1,AA,AB,AC,AE,UB,DI, U)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AA(N),AB(-M1:N),AC(-M1:N),DI(-M1:N),
     +          AE(-M1:N), UB(-M1:N)
      DO 100 I = 1, N
        DIV=AA(I)-UB(I-1)*UB(I-1)*DI(I-1)
     +           -AC(I-M1)*AC(I-M1)*DI(I-M1)
     +           -AE(I-M1+1)*AE(I-M1+1)*DI(I-M1+1)
     +      -( UB(I-1)*AE(I-1)*DI(I-1)
     +        +UB(I-M1+1)*AE(I-M1+1)*DI(I-M1+1))*U
        DI(I)=1.0D0/DIV
        UB(I)=AB(I)-AC(I-M1+1)*AE(I-M1+1)*DI(I-M1+1)
        AE(I)=-UB(I-1)*AC(I-1)*DI(I-1)
  100 CONTINUE
      RETURN
      END
```

各変数は、DIV=u, DI(I)=d_i, UB(I)=$\tilde{b}_i$, AE(I)=e_i と対応している。なお、プログラム中の変数DIV の右辺の最後にかかっている、パラメータ U は、「後(うしろ：人の名前) のパラメータ」と呼ばれており、不完全 LU 分解が安定に進行するための「まじない」である。U=0.98 がベストだが、U=0.95 を通常とする。後の ICCG でも出てくるが、式(5.5)：$\mathbf{x} - \mathbf{x}^{(k)} = \mathbf{A}_0^{-1}\mathbf{R}(\mathbf{x} - \mathbf{x}^{(k-1)})$ のような収束判定値までの反復回数 (k) が、U=0.95 では、U=0.98 の場合に比べて 2 割程度増える。

また、行列 **A** が M 行列ならば、不完全 LU 分解が可能であることが知られる。ここに、行列 **A**=$(a_{i,j})$ が M 行列であるとは、

$a_{i,i} > 0, \quad a_{i,j} \le 0 \ \ (i \ne j)$ かつ $\mathbf{A}^{-1}$の各要素が ≥ 0

のこととする。この M 行列である条件は、$\mathbf{A}^{-1}$を使うので実務的でない。そこで、ここでは実務的な Axelsson の提案を採用する (文献 [6])。

(1) $a_{i,i} > 0,\ i = 1, \cdots, n-1;\ a_{N,N} \ge 0$

(2) $i \ne j$ の $a_{i,j} \le 0$

(3) $i < N$なる各 i 行に必ず、$i \ne j, a_{i,j} < 0$ なる j がある

今、熱方程式の例では、不等あみ目でも、この M 行列の条件を満たす。

かくして、不完全 LU 分解できる $\mathbf{A}_0$= **A**+**R** が見つかった。この行列 $\mathbf{A}_0$と、式 (5.5) に従う、力ずくの反復法は、Stone 法と呼ばれる。

次に、(本題である)1970 年代に入って見直された、前処理つき共役勾配法 (PCG: Pre-

condition Conjugate Gradient) と、その前処理に不完全 **LU** 分解を利用した **ICCG** 法 **(Incomplete Choleski CG)** について説明する。ICCG の親に当たる CG 法は、1952 年に Hestenes と Stieful の天才が考えた方法で、先の式(5.5) による力ずくの反復法 (Stone 法など) よりも、解に至る勾配を考えて早く収束させるもので理論上速く計算できるはずだった。しかし、実際には、前処理しないと収束が遅く、時間がかかりすぎて実用的でなかった。それが、ICCG の出現により、飛躍的に解消された。目安として、小さい元数 (n) の問題では、Stone 法はメモリはくわないが、帯ガウス BSGLU より 5 倍遅く、ICCG よりも 10 倍程度遅い。元数 (n) が増えれば、差はもっとひらく。

まとめれば、　力ずくの反復法＋不完全 LU 分解＝ Stone 法

PCG 法　　＋前処理に不完全 LU 分解＝ ICCG 法　となる。

PCG 法から、ICCG 法に至る過程は、次のようになる。

(5.7) $$A\mathbf{x} = \mathbf{b}$$

に対して、行列 A に近い対称正定値行列 C を選び、

(5.8) $$C = U^T U \quad (\text{コレスキー分解})$$

とおく (U は上三角行列)、

(5.9) $$\mathbf{x} = U^{-1}\tilde{\mathbf{x}}$$

とおき、式(5.7) に、U^{-T} $(= (U^{-1})^T)$ をかけた式：$U^{-T}A\mathbf{x} = U^{-T}\mathbf{b}$ に代入して、$\tilde{\mathbf{x}}$についての方程式に変換すると、

(5.10) $$U^{-T}AU^{-1}\tilde{\mathbf{x}} = U^{-T}\mathbf{b}$$

となる。**この式 (5.10) に CG 法を適用する。それが PCG 法の原算法である** (文献 [9])。CG 法では、$\tilde{A} \equiv U^{-T}AU^{-1}$が対称正定値である必要がある。その点で、$\tilde{A}$は合格となっている。行列 C が A に近い時、$U^{-T}AU^{-1}$の固有値は、元の A の固有値よりも 1 の近くに密集する仕掛けになっている。そのことは、もし極端に C=A とすると、$U^{-T}AU^{-1} = I$(単位行列) となって全固有値が 1 になってしまうことから、容易に想像できる。**CG 法は、A または$\tilde{A}$の固有値分布によって収束の速さが決まる。**

Reid は、行列 C として、A の 3 重対角部 $(a_{i,i-1},\ a_{i,i},\ a_{i,i+1})$ を採用する着想で、CG 法に永い眠りから光をあてた。しかし、行列 A の 3 重対角部を LU 分解するのには、まだそれなりの負担 (メモリ、計算時間の点で) が存在した。

式(5.8) の行列 C に、不完全 LU 分解を使ったものが ICCG 法で、その事で格段に進歩した。

(5.11) $$C = LU\ = U^T U \quad (\text{不完全 } LU \text{分解})$$

C が対称行列の場合には、上三角行列 U だけ使う。この時、**ICCG の常用算法は、$A\mathbf{x} = \mathbf{b}$ を解くのに、次のようになる。**常用算法は、原算法を、計算時間と精度を考え、式(5.9) の関係に注目して、˜の付かない世界に変換したもの。

初期近似 $\mathbf{x}_0$ を選ぶ

$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$; $\mathbf{P}_0 = C^{-1}\mathbf{r}_0$; $k = 0$

while $\| \mathbf{r}_k \| > eps \cdot \| \mathbf{b} \|$ *do*

$\mathbf{q}_k = A\mathbf{P}_k$

$\alpha_k = (\mathbf{r}_k^T C^{-1}\mathbf{r}_k)/(\mathbf{P}_k^T \mathbf{q}_k)$

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{P}_k$

$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{q}_k$

$\beta_k = (\mathbf{r}_{k+1}^T C^{-1}\mathbf{r}_{k+1})/(\mathbf{r}_k^T C^{-1}\mathbf{r}_k)$

$\mathbf{P}_{k+1} = C^{-1}\mathbf{r}_{k+1} + \beta_k \mathbf{P}_k$

$k = k + 1$

ここで、$C = U^T U$, $C^{-1} = U^T U^{-1}$

常用算法の中の、$\mathbf{P}_0 = C^{-1}\mathbf{r}_0 = (U^T U)^{-1}\mathbf{r}_0 = U^{-T}U^{-1}\mathbf{r}_0$ において、$\mathbf{P}_0 = U^{-T}U^{-1}\mathbf{r}_0$の作り方について説明する。$U^{-T}U^{-1}$を作って、それを $\mathbf{r}_0$にかけるのではない！。そのようなことをしたら、C も帯、U も帯であっても、$U^{-T}U^{-1}$は密行列化する。必ず、

solve $U\mathbf{z} = \mathbf{r}_0$ (後退代入により、$\mathbf{z} = U^{-1}\mathbf{r}_0$ を作る)

solve $U^T\mathbf{P}_0 = \mathbf{z}$ (前進代入より、$\mathbf{P}_0 = U^{-T}\mathbf{z} = U^{-T}U^{-1}\mathbf{r}_0$ を作る)

ICCG(1,2) 用のサブルーチン:ICCG12 は、次のようになる。

```
      SUBROUTINE ICCG12(AA,AB,AC,AE,UB,DI,P,R,W,B0
     +           ,X,AP,EPSLON,ERR,M1,N,KCOUNT,BNRM)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AA(N),AB(-M1:N),AC(-M1:N),DI(-M1:N)
     +,   P(-M1:N+M1),W(-M1:N+M1),B0(N),X(-M1:N+M1)
     +,   AP(N),R(N),UB(-M1:N),AE(-M1:N)
      DO 10 I = 1, N
   10  R(I)=B0(I)-(AA(I)*X(I)+AB(I)*X(I+1)+AC(I)*X(I+M1)
     +           +AB(I-1)*X(I-1)+AC(I-M1)*X(I-M1))
CC    DO 20 I = 1, N
CC      R(I) = B0(I)-R(I)
CC 20 CONTINUE
***
      DO 30 I = 1, N
   30  W(I)=(R(I)-AC(I-M1)*W(I-M1)-UB(I-1)*W(I-1)
     +              -AE(I-M1+1)*W(I-M1+1))*DI(I)
      DO 40 I = N, 1, -1
   40  W(I)=W(I)-DI(I)*(UB(I)*W(I+1)+AC(I)*W(I+M1)
     +                           +AE(I)*W(I+M1-1))
***
      RR1 = 0.0D0
      DO 60 I = 1, N
        P(I) = W(I)
        RR1 = RR1+R(I)*W(I)
   60 CONTINUE
***           ITERATION
      DO 1000 K = 1, 200
       RAP = 0.0D0
       DO 100 I = 1, N
        AP(I)=AA(I)*P(I)+AB(I)*P(I+1)+AC(I)*P(I+M1)
     +         +AB(I-1)*P(I-1)+AC(I-M1)*P(I-M1)
  100   RAP = RAP+P(I)*AP(I)
CC     DO 110 I = 1, N
CC110   RAP = RAP+P(I)*AP(I)
       ALPHA = RR1/RAP
       ERR = 0.0D0
```

```
      DO 200 I = 1, N
       X(I) = X(I)+ALPHA*P(I)
       R(I) = R(I)-ALPHA*AP(I)
  200  ERR = ERR+R(I)*R(I)
      ERR = DSQRT(ERR)/BNRM
      IF( ERR.LT.EPSLON )   GO TO 2000
***
      DO 230 I = 1, N
  230  W(I)=(R(I)-AC(I-M1)*W(I-M1)-UB(I-1)*W(I-1)
     +                -AE(I-M1+1)*W(I-M1+1))*DI(I)
      DO 240 I = N, 1, -1
  240  W(I)=W(I)-DI(I)*(UB(I)*W(I+1)+AC(I)*W(I+M1)
     +                              +AE(I)*W(I+M1-1))
***
      RR2 = 0.0D0
      DO 300 I = 1, N
  300  RR2 = RR2+R(I)*W(I)
      BETA = RR2/RR1
      RR1 = RR2
      DO 400 I = 1, N
  400  P(I) = W(I)+BETA*P(I)
 1000 CONTINUE
 2000 KCOUNT = K
*
      RETURN
      END
```

AU=F を解く、帯ガウスにかわる ICCG 法による解法部分 (メインプログラム) は次のようになる。

```
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*4 CPUT
      PARAMETER(MJ=30,M=MJ*10,M2=MJ*11-1,N=M*M2,EPS=1.0D-7)
      DIMENSION AA(N),AB(-M:N),AC(-M:N),F(N)
     +,  B1(N),BW(-M:N+M)
     +,  DI(-M:N),P(-M:N+M),W(-M:N+M)
     +,  X(-M:N+M),AP(N),R(N),UB(-M:N),AE(-M:N)
C
      DATA EPSN/1.0D-5/,NIT/128/
           EPSICW=EPSN/16.0
      USHIRP=0.98D0
*C          0.98(Best) ***
      J1=M*(5*MJ-1) +3*MJ
      J2=M*(6*MJ-1) +3*MJ
      J3=M*(5*MJ-1) +8*MJ
      J4=M*(6*MJ-1) +8*MJ
C
      READ(5,*) DF
      WRITE(6,*) ' % DF= ',DF
      WRITE(6,*) ' % M= ',M,' M2= ',M2,' N= ',N
      FMJ= DFLOAT(MJ)
      DHX=1.0D0/FMJ
      DHY=1.0D0/FMJ
C
      DO 10 I=1,M-1
        AA(I)=2.0D0*(DF+1.0D0)
        AB(I)=-(1.0D0+DF)/2.0D0
        AC(I)=-1.0D0
   10 CONTINUE
      AA(M)=DF+1.0D0
      AB(M)=0.0D0
      AC(M)=-0.5D0
C
```

```
      DO 30 J=1,M2-2
        DO 20 K=1,M-1
          AA(M*J+K)=4.0D0
          AB(M*J+K)=-1.0D0
          AC(M*J+K)=-1.0D0
   20   CONTINUE
        AA(M*J+M)=2.0D0
        AB(M*J+M)=0.0D0
        AC(M*J+M)=-0.5D0
   30 CONTINUE
C
      DO 40 I=N-M+1, N-1
        AA(I)=2.0D0*(DF+1.0D0)
        AB(I)=-(1.0D0+DF)/2.0D0
        AC(I)=0.0D0
   40 CONTINUE
      AA(N)=DF+1.0D0
      AB(N)=0.0D0
      AC(N)=0.0D0
C
*C    Uhen ***
      DO 60 J= 5*MJ-1, 6*MJ-1
        DO 70 I=0,2*MJ
          F(M*J +2*MJ+I)= 0.2D0*(DHX*DHY)
          F(M*J +6*MJ+I)=-0.2D0*(DHX*DHY)
   70   CONTINUE
   60 CONTINUE
C
      CALL CLOCK0
C
*C    BW(Ui) no Syokiti ***
      DO 111 K=1,N
  111   BW(K)=0.0D0
C
*C    A*U=F wo Toku; Kotae --> F ***
      BNRM=0.0
      DO 110 I=1,N
  110   BNRM=BNRM+F(I)*F(I)
      BNRM=DSQRT(BNRM)
*
      CALL ICDCMP(N,M,AA,AB,AC,AE,UB,DI, USHIRP)
      DO 226 I=1,N
  226   X(I)=BW(I)
C
      KCT=0
      DO 230 IN=1,NIT
        CALL ICCG12(AA,AB,AC,AE,UB,DI, P,R,W,F,X
     +            ,AP,EPSICW,ERR,M,N,KCOUNT,BNRM )
C
        KCT=KCT+KCOUNT
        CNR=0.0D0
        DO 116 I=1,N
          B1(I)=F(I)-(AC(I-M)*X(I-M)+AB(I-1)*X(I-1)
     +          +AA(I)*X(I)+AB(I)*X(I+1)+AC(I)*X(I+M))
  116     CNR=CNR+B1(I)*B1(I)
        CNR=DSQRT(CNR)/BNRM
*
        WRITE(6,5800) '%',IN,X(J1),X(J2),X(J3),X(J4),CNR,KCOUNT
 5800   FORMAT(1H,X,I4,F10.5,F10.5,F10.5,F10.5,D14.6,I5)
        IF(CNR.LT.EPSN) GO TO 250
          EPSICW=EPSICW/2.0
C
  230 CONTINUE
  250 CONTINUE
```

```
C
      DO 220 I=1,N
*C      BW(I)= X(I)  ***
  220   F(I) = X(I)
C
      CALL CLOCK(CPUT)
      STOP
      END
```

ここで、反復法に必要なパラメータは、EPSN= 10^{-5}, EPSICW (eps のこと)=EPSN/16, NIT=128 とした。離散化部分は、帯ガウスと同じ。

初期近似の $\mathbf{x}_0(= BW(I))$ には、

$$\mathbf{x}_0 = 0.0$$

を与えているが、電導率κ に段差などがあり複雑な場合には、それらしい初期値を与えないと解が収束しない。その初期値のよしあしによって、収束時間も変わってくる。

また、ICCG 法は、ICCG12 の外側の IN(プログラム上) ループで、実際に解く **AU=F** の精度を評価しておく必要がある。

kct=0
$\mathbf{x}_0$を選ぶ
bnrm=$\| \mathbf{F} \|$ の計算
CALL ICDCMP：$C = \mathbf{A}_0$を不完全 LU 分解 ($\mathbf{A}_0 = \mathbf{A} + R$)
while $\| \mathbf{F} - A\mathbf{x} \| /$bnrm $>$ EPSN do
  CALL ICCG12：($U^{-T}AU^{-1}\tilde{\mathbf{x}} = U^{-T}\mathbf{b}$ 相当を解く) kcount 回の反復
  kct=kct+kcount

ICCG 内の判定値：EPSICW は EPSN より小さく、場合によるが EPSN/16 程度がよい。ここでの例では、1 回の ICCG で全体の評価値 EPSN より小さくなる。EPSN= 10^{-5}で、帯ガウスの結果と ICCG の結果は、6 桁まで合う。

ICCG は、1 次元配列 (n 個程度) を、AA, AB, AC も含め作業領域など 11 本必要とするので、m が 11 本当たり (MJ=1) から、帯ガウスと ICCG の計算 (CPU) 時間、メモリ量に差がでる。MJ=30 では、m=300, n=98700 となり、メモリ量、CPU 時間は、対称帯ガウス:237MByte(715 秒)、ICCG 法：9MByte(7 秒) の差となる。MJ=30, DF=1 の時出力結果は、次のようになる。

```
% DF= 1.000000000000000
% M= 300  M2= 329  N= 98700
%    1   0.08895   0.08895  -0.12250  -0.12250  0.463624d-06   84
% MJ= 30
% CPUT= 7.00000000
% UMIN,(x,y)= -0.1526176586286306 (165 ,218 )
% UMAX,(x,y)= 0.1067951164771567 (165 ,85 )
z=[
 0.00  0.01  0.01  0.02  0.03  0.03  0.02  0.01  0.01  0.00 ;
 0.00  0.01  0.02  0.04  0.07  0.07  0.04  0.02  0.01  0.00 ;
 0.00  0.01  0.02  0.04  0.09  0.09  0.04  0.02  0.01  0.00 ;
 0.00  0.00  0.01  0.02  0.05  0.05  0.02  0.01  0.00  0.00 ;
-0.01 -0.01 -0.01 -0.02 -0.02 -0.02 -0.02 -0.01 -0.01 -0.01 ;
-0.01 -0.02 -0.03 -0.05 -0.09 -0.09 -0.05 -0.03 -0.02 -0.01 ;
-0.01 -0.03 -0.05 -0.08 -0.13 -0.13 -0.08 -0.05 -0.03 -0.01 ;
-0.02 -0.04 -0.06 -0.08 -0.12 -0.12 -0.08 -0.06 -0.04 -0.02 ;
-0.02 -0.04 -0.06 -0.08 -0.10 -0.10 -0.08 -0.06 -0.04 -0.02 ;
-0.02 -0.04 -0.06 -0.08 -0.09 -0.09 -0.08 -0.06 -0.04 -0.02 ;
];
contour(0:1:9, 0:1:9, z,10);
```

また、モデルを今までの：$\mathbf{Ax}^{(k)}=\mathbf{b}$ から、$\mathbf{x}^{(k)}=\mathbf{x}^{(k-1)}+\delta\mathbf{x}$ として、

$\mathbf{A}\delta\mathbf{x}=\mathbf{b}-A\mathbf{x}^{(k-1)}$ に変えて解くモデル (ここではδ方式と呼ぶ) もある。通常δ方式の方が、収束までの反復回数 **kcount** が余計にかかり、収束時間も長くなる。δ方式では、今までのモデルに比べ kcount が 3 倍に増える場合もあり、CPU 時間で 2 割程度増える。この例の場合、$\mathbf{x}_0(=BW(I))=0.0$ であるので、δ方式も元方式も、反復としては全く同じ振る舞いをする。δ方式のプログラム (主部) は、元方式と 3 箇所の変更点で、次のようになる。

```
*C     BW(Ui) no Syokiti ***
       DO 111 K=1,N
  111    BW(K)=0.0D0
C
*C     A*U=F wo Toku; Kotae --> F ***
*C     Delta You ***
       DO 716 I=1,N
         F(I)=F(I)-(AC(I-M)*X(I-M)+AB(I-1)*X(I-1)
     +         +AA(I)*X(I)+AB(I)*X(I+1)+AC(I)*X(I+M))
  716  CONTINUE
CC
       BNRM=0.0
       DO 110 I=1,N
  110    BNRM=BNRM+F(I)*F(I)
       BNRM=DSQRT(BNRM)
*
       CALL ICDCMP(N,M,AA,AB,AC,AE,UB,DI, USHIRP)
*C     Delta You ni Henkou ***
       DO 226 I=1,N
  226    X(I)= 0.0D0
C
       KCT=0
       DO 230 IN=1,NIT
         CALL ICCG12(AA,AB,AC,AE,UB,DI, P,R,W,F,X
     +             ,AP,EPSICW,ERR,M,N,KCOUNT,BNRM )
C
         KCT=KCT+KCOUNT
         CNR=0.0D0
         DO 116 I=1,N
           B1(I)=F(I)-(AC(I-M)*X(I-M)+AB(I-1)*X(I-1)
     +           +AA(I)*X(I)+AB(I)*X(I+1)+AC(I)*X(I+M))
  116      CNR=CNR+B1(I)*B1(I)
         CNR=DSQRT(CNR)/BNRM
*
         WRITE(6,5800) IN,X(J1),X(J2),X(J3),X(J4),CNR,KCOUNT
 5800    FORMAT(1H,I4,F10.5,F10.5,F10.5,F10.5,D14.6,I5)
         IF(CNR.LT.EPSN) GO TO 250
           EPSICW=EPSICW/2.0
C
  230  CONTINUE
  250  CONTINUE
C
*C     Delta You ni Henkou ***
       DO 220 I=1,N
*C       BW(I)= BW(I)+ X(I) ***
  220    F(I) = BW(I)+ X(I)
```

6 おわりに

Fig.1 のシミュレーション場において、熱方程式を CV 法で離散化した、AU=F を、これまで述べた、列ガウス (GLU)、行ガウス (GLUR)、対称帯 (列) ガウス (BSGLU)、ICCG 法 (ICCG12) のソルバで解いた結果を Table 2 に示す。計算はすべて、日立 FLORA330 DX2 (Pentium3) 933MHz, メモリ 256MB で行なった。これらから、メモリと CPU 時間の制約から列 (行) ガウスは、すぐに実用的でないことが分かる。ICCG 法は、帯ガウスに比較して、M=250(MJ=25)、N=68500 くらいから威力を発揮する。CPU 比 17 倍、メモリ比 22 倍である。M がさらに大きくなると、この比はさらに大きくなる。MJ=30 と MJ=20 との u_{min}, u_{max}のあみ目誤差は、2%程度になっている。できれば、MJ=30 で温度分布 u_iを計算したい。となれば、ICCG にたよらざるを得ない。

Table 2 Solver(Gauss, ICCG) performance for MJ(mesh size)

MJ	N	A(MB)	AR(MB)	GLU(s)	GLUR(s)	BSGLU(s)	ICCG(s)[MB]	kcount
1	100	0.08	0.008	0.	0.	0.	0. [0.009]	8
5	2700	58	1	335	860	0.	0. [0.238]	20
10	10900	950	9	—	—	1	1 [0.96]	31
15	24600	4800	29	—	—	3	1 [2]	44
20	43800	15000	70	—	—	8	2 [3.8]	56
25	68500	38000	137	—	—	69	4 [6]	69
30	98700	78000	237	—	—	715	7 [8.6]	84

最後に、同じ拡散系の方程式をもつ、ポアソン方程式についてふれておく。

$$div[-\epsilon\nabla\psi] = e(p - n - Na + Nd) \tag{6.1}$$

は、ψについて熱方程式の u と同様に、

$$\mathbf{A}\psi = \mathbf{F} \tag{6.2}$$

の形に CV 法により離散化できる。しかし、右辺 $\mathbf{F}$ の $e(p-n-Na+Nd)$ の電荷 p や n に、$e^{\frac{e\psi}{kT}}$ (T:温度、k:ボルツマン定数) の形の非線形性を持つために、(6.1) 式をそのまま離散化しても収束しない。

そこで、Newton 反復の方法を取り入れ、(6.1) 式を

$$div[-\epsilon\nabla\psi^{(k)}] = \mathbf{F}_0 + \frac{\partial F}{\partial\psi}\delta\,; \quad \psi^{(k)} - \psi^{(k-1)} = \delta, \ \mathbf{F}_0 = e(p - n - Na + Nd) \tag{6.3}$$

として、

$$div[-\epsilon\nabla\psi^{(k)}] - \frac{\partial F}{\partial\psi}\psi^{(k)} = \mathbf{F}_0 - \frac{\partial F}{\partial\psi}\psi^{(k-1)}\,; \qquad \frac{\partial F}{\partial\psi} = e\cdot\frac{e}{kT}(p^{(k)} + n^{(k)}) \tag{6.4}$$

に基づいて離散化すれば、収束する。これは、天才 Gummel が最初に発見した。もちろん我々は、前式を CV 法にしたがって離散化する。

$$\int_\Gamma(-\epsilon\nabla\psi^{(k)})\cdot\mathbf{n}dS-\int_\Omega\frac{\partial F}{\partial\psi}\psi^{(k)}d\mathbf{V}=\int_\Omega(\mathbf{F}_0-\frac{\partial F}{\partial\psi}\psi_{(k-1)})d\mathbf{V} \tag{6.5}$$

この左辺第 2 項は、ψ_iの対角項に加わり行対角優位性を増し、解きやすい方向に働く。

参考文献

[1] 小国 力編著, 村田 健郎、三好 俊郎、ドンガラ J,J、長谷川 秀彦著、行列計算ソフトウェア (WS、スーパーコン、並列計算機), 丸善,pp.252-275,Nov.1991

[2] 村田 健郎,CV 法と、手作り数値シミュレーションシステムの奨め, 日本計算工学会, 計算工学 vol.3,No.1,pp.44-53,1998

[3] 村田 健郎,CV 法と、手作り数値シミュレーションシステムの奨め (第 2 回)-非線形純拡散問題と割線反復法-, 日本計算工学会, 計算工学 vol.3,No.3

[4] 村田 健郎,CV 法と、手作り数値シミュレーションシステムの奨め (第 3 回)-移流拡散系の離散化：特に指数法について-, 日本計算工学会, 計算工学 vol.3,No.4, pp.246-253,Dec.1998

[5] 村田 健郎,CV 法と、手作り数値シミュレーションシステムの奨め (第 4 回)-連立非線形移流拡散系：半導体デバイス解析の場合-, 日本計算工学会, 計算工学 vol.4,No.2,1999

[6] 村田 健郎,「BASIC 数学」連載：線形数値計算法とその応用, 現代数学社,1992

[7] 村田 健郎, 線形代数と線形計算法序節, サイエンス社,1986

[8] 小国 力著,MATLAB と利用の実際 [第 2 版], サイエンス社,2001

[9] 村田 健郎, 名取 亮, 唐木 幸比古著, 岩波書店,pp.56-137,1990

付録 1 (列ガウス:GLU) のプログラム

```
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*4 CPUT
      PARAMETER(MJ=2,M=MJ*10,M2=MJ*11-1,N=M*M2,EPS=1.0D-7)
      DIMENSION A(N,N),AA(N),AB(-M:N),AC(-M:N),F(N), IP(N)
C
      READ(5,*) DF
      WRITE(6,*) ' % DF= ',DF
      WRITE(6,*) ' % M= ',M,' M2= ',M2,' N= ',N
      FMJ= DFLOAT(MJ)
      DHX=1.0D0/FMJ
      DHY=1.0D0/FMJ
C
      DO 10 I=1,M-1
        AA(I)=2.0D0*(DF+1.0D0)
        AB(I)=-(1.0D0+DF)/2.0D0
        AC(I)=-1.0D0
   10 CONTINUE
      AA(M)=DF+1.0D0
      AB(M)=0.0D0
      AC(M)=-0.5D0
C
```

```
      DO 30 J=1,M2-2
        DO 20 K=1,M-1
          AA(M*J+K)=4.0D0
          AB(M*J+K)=-1.0D0
          AC(M*J+K)=-1.0D0
   20   CONTINUE
        AA(M*J+M)=2.0D0
        AB(M*J+M)=0.0D0
        AC(M*J+M)=-0.5D0
   30 CONTINUE
C
      DO 40 I=N-M+1, N-1
        AA(I)=2.0D0*(DF+1.0D0)
        AB(I)=-(1.0D0+DF)/2.0D0
        AC(I)=0.0D0
   40 CONTINUE
      AA(N)=DF+1.0D0
      AB(N)=0.0D0
      AC(N)=0.0D0
C
      DO 50 I=1,N
        A(I,I)=AA(I)
  50  CONTINUE
      DO 52 I=1,N-1
        A(I,I+1)=AB(I)
        A(I+1,I)=AB(I)
  52  CONTINUE
      DO 54 I=1,N-M
        A(I,I+M)=AC(I)
        A(I+M,I)=AC(I)
  54  CONTINUE
C
*C    Uhen ***
      DO 60 J= 5*MJ-1, 6*MJ-1
        DO 70 I=0,2*MJ
          F(M*J +2*MJ+I)= 0.2D0*(DHX*DHY)
          F(M*J +6*MJ+I)=-0.2D0*(DHX*DHY)
   70   CONTINUE
   60 CONTINUE
C
      CALL CLOCK0
C
*C    AR*U=F wo Toku; Kotae --> F ***
      CALL GLU (N,A, IP,EPS,IR)
      IF( IR.EQ.0 )  THEN
        CALL GSLV (N, A, F,IP)
      ENDIF
C
      CALL CLOCK(CPUT)
C
      STOP
      END
CCC
      SUBROUTINE GLU (N, A, IP, EPS, IR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(N,N), IP(N)
C
C forward elimination for A
      IR = 0
      DO 100 K = 1, N
C
C     pivoting
        AMAX = ABS(A(K,K))
        IPK = K
```

```
            DO 110 I = K+1, N
              AIK = ABS(A(I,K))
              IF( AIK.GT.AMAX )  THEN
                 IPK = I
                 AMAX = AIK
              END IF
  110       CONTINUE
            IP(K) = IPK
C
            IF( AMAX.GT.EPS )  THEN
               IF( IPK.NE.K )  THEN
                  W = A(IPK,K)
                  A(IPK,K) = A(K,K)
                  A(K,K) = W
               END IF
C
               DO 120 I = K+1, N
                 A(I,K) = -A(I,K)/A(K,K)
  120          CONTINUE
C
               DO 130 J = K+1, N
                 IF( IPK.NE.K )  THEN
                    W = A(IPK,J)
                    A(IPK,J) = A(K,J)
                    A(K,J) = W
                 END IF
C
                 T = A(K,J)
                 DO 140 I = K+1, N
                   A(I,J) = A(I,J)+A(I,K)*T
  140            CONTINUE
  130          CONTINUE
C
            ELSE
               IR = IR+1
               RETURN
            END IF
  100 CONTINUE
      RETURN
      END
C
      SUBROUTINE GSLV ( N, A, B, IP )
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(N,N), B(N), IP(N)
C
C forward elimination for B
      DO 100 K = 1, N
C
        IF( IP(K).NE.K )  THEN
           W = B(IP(K))
           B(IP(K)) = B(K)
           B(K) = W
        END IF
C
        T = B(K)
        DO 110 I = K+1, N
  110     B(I) = B(I)+A(I,K)*T
  100 CONTINUE
C
C backward substitution for B
      B(N) = B(N)/A(N,N)
      DO 200 K = N-1, 1, -1
        T = B(K+1)
        DO 210 I = 1, K
  210     B(I) = B(I)-A(I,K+1)*T
        B(K) = B(K)/A(K,K)
  200 CONTINUE
      RETURN
      END
```