第 1 章

連立 1 次方程式の解法

有限要素法に限らず差分法や境界要素法など微分方程式の近似解法では最終的には (大次元の) 連立 1 次方程式を解く．したがって，連立 1 次方程式の数値解法が重要な意味をもつことになる．この場合，どの近似解法を用いるかによって結果として得られる連立 1 次方程式の形が異なる．いいかえれば，連立 1 次方程式を行列形式

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

で書いたとき，行列 A の形が近似解法の種類によって変わる．

1.1 解法の種類

たとえば差分法 (中心差分) を用いて 2 次元のポアソン方程式を解いた場合，大きさは領域内の格子点数程度の行列で，非ゼロ要素が対角線およびそれに平行な合計 5 本の線上に並ぶ行列になる．有限要素法の場合には，節点数程度の大きさの行列で，差分法ほど規則的ではないが非ゼロ要素が対角線を中心として疎らに分布する．この場合，分布の仕方は節点番号の付け方に依存し，効率的な付け方をすれば非ゼロ要素が対角線からあまり離れない (すなわちバンド幅がせまい) ようにできる．いずれにせよ差分法や有限要素法では大次元の疎行列になる．たとえば格子数や節点が 100 であれば 100×100 程度の行列になる．一方，境界要素法では領域境界上の要素数程度の行列 (したがって差分法や有限要素法に比ればは圧倒的に小さい行列) となるが，全体が詰まった密な行列になる．

これらのことを反映して，それぞれの方法で用いられる連立 1 次方程式の解法も異なる．まず差分法では，行列の規則性がよいため，SOR 法などの反復法を用いても収束が比較的速い．そのため反復法がよく使われる．一方，境界要素法ではガウスの消去法など直接法 (消去法) が一般的である．有限要素法は中間的な性格をもっているが，行列の対称性を利用したり，バンド幅が小さくできるような場合には，演算量や記憶容量が節約できるため，確実に解が求まる消去法がよく用いられる．そこで本書では代表的な消去法について説明を行う．

1.2　ガウスの消去法

ガウスの消去法はすべての消去法の基礎となるため，まずはじめに説明する．

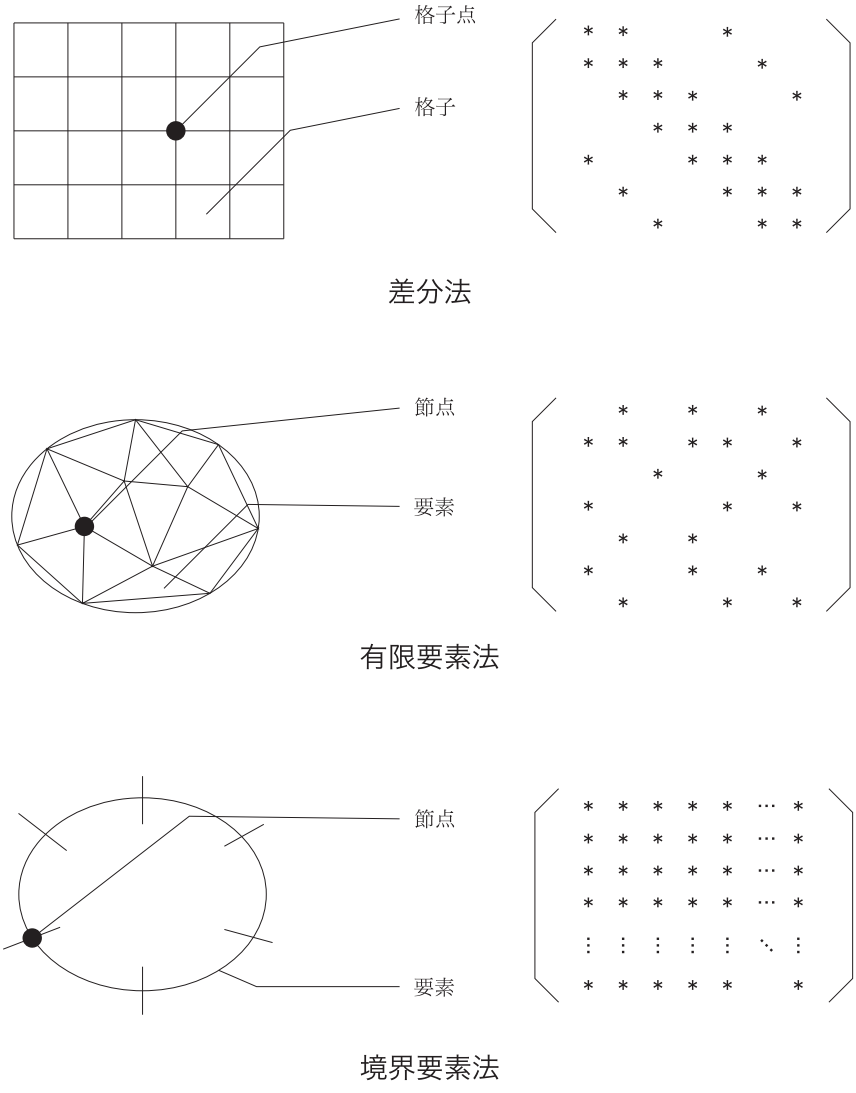

図 1.1　差分法・有限要素法・境界要素法に現われる行列（*は非ゼロ要素）

次の形の連立 1 次方程式

$$\begin{array}{ccccccccccc}
a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n-1}^{(1)}x_{n-1} & + & a_{1n}^{(1)}x_n & = & b_1^{(1)} \\
 & & a_{22}^{(2)}x_2 & + & \cdots & + & a_{2n-1}^{(2)}x_{n-1} & + & a_{2n}^{(2)}x_n & = & b_2^{(2)} \\
 & & & & \ddots & & & & & & \\
 & & & & & & a_{n-1n-1}^{(n-1)}x_{n-1} & + & a_{n-1n}^{(n-1)}x_n & = & b_{n-1}^{(n-1)} \\
 & & & & & & & & a_{nn}^{(n)}x_n & = & b_n^{(n)}
\end{array} \tag{1.1}$$

は以下に示すように簡単に解くことができる．すなわち，いちばん下の式から x_n を求め，次にこれを下から 2 番目の式に代入して x_{n-1} を求める．さらに x_n ，x_{n-1} を下から 3 番目の式に代入して x_{n-2} を求める．以下同様にすれば x_n，x_{n-1}，$x_{n-2}, \cdots, x_1$ の順に解がもとまる．アルゴリズム的に書けば次のようになる．

$j = n,\ n-1,$ ……，1 に対し次式を計算する：

$$x_j = \frac{1}{a_{jj}^{(j)}} \left(b_j^{(j)} - \sum_{k=j+1}^{n} a_{jk}^{(j)} x_k \right) \tag{A}$$

（ただし $k > n$ ならば総和は計算しない）

さて，一般の連立 1 次方程式

$$\begin{array}{c}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\
\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n
\end{array} \tag{1.2}$$

を解くには，式 (1.2) を式 (1.1) の形に書き換えればよい．それにはまず式 (1.2) の第 1 式を用いて第 2 式から第 n 式までの x_1 を消去する．ここで第 1 式を取り除けば，

$$\begin{array}{c}
a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\
\vdots \\
a_{n2}^{(2)}x_2 + a_{n3}^{(2)}x_3 + \cdots + a_{nn}^{(2)}x_n = b_n^{(2)}
\end{array} \tag{1.3}$$

という，$x_2 \sim x_n$ に関する連立 $n-1$ 元方程式になる．係数は式 (1.2) から変化するため，式 (1.3) では上添字をつけて区別している．なお，式 (1.2) では上添字 (1) が省略されていると解釈する．同様に式 (1.3) の第 1 式を用いて，式 (1.3) の第 2 式以下の式の x_2 を

消去する．その結果得られた方程式からいちばん上の式を取り除けば $n-2$ 元連立 1 次方程式

$$\begin{aligned} a_{33}^{(3)}x_3 + a_{34}^{(3)}x_4 + \cdots + a_{3n}^{(3)}x_n &= b_3^{(3)} \\ &\vdots \\ a_{n3}^{(3)}x_3 + a_{n4}^{(3)}x_4 + \cdots + a_{nn}^{(3)}x_n &= b_n^{(3)} \end{aligned} \tag{1.4}$$

を得る．以下，上の手順を繰り返し，最終的に

$$a_{nn}^{(n)}x_n = b_n^{(n)} \tag{1.5}$$

という方程式になるまで続ける．このとき各ステップで取り除いた方程式を上から順に並べれば式 (1.1) の形の方程式となる．これらを模式的に図示したのが図 1.2 である．式 (1.3) などを見てもわかるように，それぞれの消去の段階で係数や方程式の右辺は順に変化する．ただし，消去の方法は同じであるため，最終的に得られる係数は以下のアルゴリズムにより求めることができる．

$l = 1$，2，$\cdots$，$n-1$ の順に

各 l に対して $j = l+1$，$\cdots$，n の順に

$$\begin{aligned} m_{jl} &= a_{jl}^{(l)} / a_{ll}^{(l)} \\ a_{jk}^{(l+1)} &= a_{jk}^{(l)} - m_{jl}a_{lk}^{(l)} \qquad (k = l+1, \cdots, n) \\ b_j^{(l+1)} &= b_j^{(l)} - m_{jl}b_l^{(l)} \end{aligned} \tag{B}$$

の計算を行う．

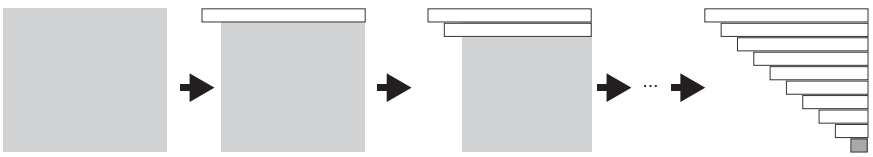

図 1.2　ガウスの消去法

アルゴリズム (B) を前進消去，(A) を後退代入とよび，(B) と (A) を組み合わせて連立 1 次方程式を解く方法をガウスの消去法とよんでいる．ガウスの消去法で注意すべき点は，(B) において $a_{ll}^{(l)}$ で割り算をおこなっている点で，もし $a_{ll}^{(l)}$ が 0 になれば計算できなくなる．$a_{ll}^{(l)}$ のことをピボットとよんでいる．またピボットが 0 でなくても絶対値の

小さい数であれば桁落ちがおきる可能性があり，以後の計算に大きな誤差が生じる恐れがある．

このことを防ぐもっとも手軽な方法は，消去の段階で得られる方程式

$$\begin{aligned} a_{ll}^{(l)}x_l + a_{ll+1}^{(l)}x_{l+1} + \cdots + a_{ln}^{(l)}x_n &= b_l^{(l)} \\ &\vdots \\ a_{il}^{(l)}x_l + a_{il+1}^{(l)}x_{l+1} + \cdots + a_{in}^{(l)}x_n &= b_i^{(l)} \\ &\vdots \\ a_{nl}^{(l)}x_l + a_{nl+1}^{(l)}x_{l+1} + \cdots + a_{nn}^{(l)}x_n &= b_n^{(l)} \end{aligned} \tag{1.6}$$

から次の消去に移る前に，方程式の入れ換えを行う．すなわち，式 (1.6) において各式の第 1 項の係数の絶対値が最大になる方程式がもとの方程式の i 番目である場合，第 1 式と第 i 式を入れ換えた上で消去を行う．この手続きのことをピボット選択とよぶ．あるいは第 1 式のなかで絶対値最大の係数をもつ変数を x_j とすれば，方程式全体で x_l を含む項と x_j を含む項を入れ換えてもよい．

〈部分ピボット選択付きの前進消去のアルゴリズム〉

1. n, A, b を入力する．
2. $l = 1, 2, \cdots, n-1$ に対して次の演算を行う．
   - 2.1　各 l について，$i = l+1, \cdots n$ に対して a_{il} の絶対値が最大になる i を見つける．
   - 2.2　$m = l, l+1, \cdots, n$ に対して a_{im} と a_{lm} を入れ換える．
   - 2.3　各 l について，$j = l+1, \cdots, n$ に対して次の演算を行う．
     - 2.3.1　$m_{jl} = a_{jl}/a_{ll}$
     - 2.3.2　$k = l+1, \cdots, n$ に対して次の演算を行う．

$$a_{jk} = a_{jk} - m_{jl}a_{lk}$$

     - 2.3.3　$b_j = b_j - m_{jl}b_l$

1.3 コレスキー法

行列を用いて連立 1 次方程式 (1.2) を

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1.7}$$

の形に書いたとき，A が正定値対称行列である場合を考えよう．ここで対称行列とは，もとの正方行列 A の行と列を入れ換えた転置行列 A^T ともとの行列 A が等しい行列のことであり，正定値とは，行列 A が 0 でない場合，任意のベクトル $\boldsymbol{x}$ に対して $\boldsymbol{x}$ と $A\boldsymbol{x}$ の内積を計算したとき

$$(\boldsymbol{x}, A\boldsymbol{x}) > 0$$

が成り立つことをいう．なお A が正定値対称でなくても A^T を A の転置行列としたとき

$$A^T A\boldsymbol{x} = A^T\boldsymbol{b}$$

とすれば $A^T A$ は正定値対称になる．

正定値対称行列 A を下三角行列

$$L = \begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & 0 \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix}$$

を用いて

$$A = L^T L \tag{1.8}$$

となるように分解する方法をコレスキー法とよんでいる．L の要素 l_{ij} は A の要素を a_{ij} として，式 (1.8) の積を直接計算することにより求めることができる．すなわち，要素 a_{ij} は m を i と j の小さい方の数として

$$a_{ij} = \sum_{k=1}^{m} l_{ik} l_{jk}$$

となる．したがって，$j < i$ のとき，$i = 2 \sim n$ に対して

$$\sum_{k=1}^{j} l_{ik} l_{jk} = a_{ij} \qquad (j = 1, 2, \cdots, i-1)$$

であるから

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) \qquad (j = 1, 2, \cdots, i-1) \tag{1.9}$$

となり，また $i = j$ のときは，$i = l \sim n$ に対して

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \tag{1.10}$$

となる．

このようにして L が求まれば，次のようにして連立 1 次方程式が解ける．すなわち，

$$L\boldsymbol{x} = \boldsymbol{y} \tag{1.11}$$

と書いたとき，式 (1.7) は

$$L^T \boldsymbol{y} = \boldsymbol{b} \tag{1.12}$$

となる．式 (1.12) は式 (1.1) と同じ上三角型であるためガウスの消去法における後退代入のアルゴリズムで解ける．また式 (1.11) は下三角型であるため，あとの式 (1.18) に示すように上から順に解を求めることができる．

〈3×3 の対称行列のコレスキー分解〉

$$\begin{aligned}
\begin{bmatrix} a & * & * \\ d & b & * \\ e & f & c \end{bmatrix} &= \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{23} \\ 0 & 0 & l_{33} \end{bmatrix} \\
&= \begin{bmatrix} l_{11}^2 & * & * \\ l_{11}l_{21} & l_{21}^2 + l_{22}^2 & * \\ l_{11}l_{31} & l_{21}l_{31} + l_{22}l_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{bmatrix} \\
&\quad (* \text{ 対称})
\end{aligned}$$

両辺の係数を比較すれば

$$l_{11} = \sqrt{a}, \quad l_{21} = d/l_{11} = d/\sqrt{a}, \quad l_{31} = e/l_{11} = e/\sqrt{a}$$

$$l_{22} = \sqrt{b - l_{21}^2} = \sqrt{b - d^2/a}, \quad l_{32} = (f - l_{21}l_{31})/l_{22} = \left(f - \frac{de}{a} \right) \Big/ \sqrt{b - \frac{d^2}{a}}$$

$$l_{33} = \sqrt{c - l_{31}^2 - l_{32}^2} = \sqrt{c - \frac{e^2}{a} - \left(f - \frac{de}{a} \right)^2 \Big/ \left(b - \frac{d^2}{a} \right)}$$

1.4　変形コレスキー法

コレスキー法のアルゴリズム (1.9)，(1.10) を見れば，コレスキー法では平方根の計算が必要であることがわかる．平方根の計算は乗除算に比べて時間がかかる上に精度的にも不利である．そこでコレスキー法を変形して平方根の計算を不要にした方法が以下に述べる変形コレスキー法である．ただしこの方法もコレスキー法と同じく対称行列に適用が限られる．(なお前節では下三角行列を用いて定式化したが，同じことなので本節では上三角行列を用いることにする．)

変形コレスキー法では式 (1.7) の行列 A を

$$A = U^T D U \tag{1.13}$$

と分解する．ここで

$$D = \begin{bmatrix} d_{11} & & & 0 \\ & d_{22} & & \\ & & \ddots & \\ 0 & & & d_{nn} \end{bmatrix} \quad U = \begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ & 1 & u_{23} & \cdots & u_{2n} \\ & & 1 & \cdots & u_{3n} \\ & 0 & & \ddots & \vdots \\ & & & & 1 \end{bmatrix} \tag{1.14}$$

である．コレスキー法と同様に式 (1.13) の右辺の積を計算して係数を比較することにより，D と U の各係数が次のアルゴリズムから求まることがわかる．

$$\begin{aligned} &d_{11} = a_{11} \\ &u_{1j} = a_{1j}/d_{11} \quad (j = 2, 3, \cdots, n) \\ &d_{ii} = a_{ii} - \sum_{m=1}^{i-1} u_{mi}^2 d_{mm} \quad (i = 2, 3, \cdots, n) \\ &u_{ij} = \left(a_{ij} - \sum_{m=1}^{i-1} u_{mj} d_{mm} u_{mi} \right) / d_{ii} \quad \begin{pmatrix} i = 2, 3, \cdots, n-1 \\ j = i+1, i+2, \cdots, n \end{pmatrix} \end{aligned} \tag{1.15}$$

このようにして D と U を求めたあと，

$$DU\boldsymbol{x} = \boldsymbol{z} \tag{1.16}$$

とおけば，式 (1.7) は

$$U^T \boldsymbol{z} = b \tag{1.17}$$

と書ける．式 (1.17) は前述のとおり式 (1.1) と逆の下三角型であり，簡単に $\boldsymbol{z}$ を求めることができる．具体的には前進消去

$$\begin{aligned} z_1 &= b_1 \\ z_i &= b_i - \sum_{m=1}^{i-1} u_{mi} z_m \quad (i = 2, 3, \cdots, n) \end{aligned} \tag{1.18}$$

を行えばよい．式 (1.16) は

$$U\boldsymbol{x} = D^{-1}\boldsymbol{z}$$

と書けば上三角型になり，右辺も直ちに計算できるため簡単に解ける (1.1 節 (A) 参照)．すなわち，右辺は D^{-1} が単に

$$\begin{bmatrix} \frac{1}{d_{11}} & & & 0 \\ & \frac{1}{d_{22}} & & \\ & & \ddots & \\ 0 & & & \frac{1}{d_{nn}} \end{bmatrix} \tag{1.19}$$

であることから，ベクトル $\boldsymbol{z}$ の各成分に上から順に $1/d_{11}, 1/d_{22}, \cdots$ をかければよい．これらの手続きをまとめれば，式 (1.16) は

$$\begin{aligned} x_n &= \frac{z_n}{d_{nn}} \\ x_i &= \frac{z_i}{d_{ii}} - \sum_{m=i+1}^{n} u_{im} x_m \quad (i = n-1, n-2, \cdots, 2, 1) \end{aligned} \tag{1.20}$$

により解くことができる．

1.5　帯行列を利用した変形コレスキー法

有限要素法で通常現われる剛性行列（剛性マトリックス）とよばれる行列は零要素を多く含んだ対称行列である．各要素の最大および最小となる節点番号の差ができるだけ小さくなるように節点番号を付けるならば，この剛性行列は対角線付近に非零要素が集中したいわゆる疎行列あるいは帯行列を形成する．一般に剛性行列は自由度が n のとき $n \times n$ の対称正方行列で，図 1.3 に示すようにバンド幅 m を持つ．

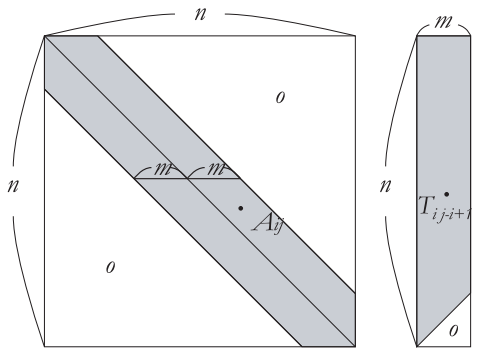

図 1.3　剛性行列と変形コレスキー法

したがって，対称な帯行列を記憶する場合，対称性を利用して右半分だけを記憶し，帯部分の外側は全部 0 であるから非零要素のみを記憶するならば，記憶場所をさらに節約することができる．

大きさが $n \times m$ の新しい行列を T とする．この場合もとの行列の要素間には次のような関係がある（$i \leq j$ とする）．

$$A_{ij} = T_{ij-i+1} \tag{1.21}$$

行列 A，すなわち行列 T の $U^T DU$ 分解は，式 (1.15) に式 (1.21) の関係を用いることによって得られるが，T は帯行列であるため常に帯の中にある要素のみを計算の対象とする．

式 (1.15) から，

$$
\begin{aligned}
&\frac{T_{1j}}{T_{11}} \to T_{1j} \quad (j = 2, 3, \cdots, m) \\
&T_{i1} - \sum_{l=i-m+1}^{i-1} T_{li-l+1} T_{li-l+1} T_{l1} \to T_{i1} \quad (i = 2, 3, \cdots, n) \\
&\left(T_{ij-i+1} - \sum_{l=j-m+1}^{i-1} T_{li-l+1} T_{lj-l+1} T_{l1} \right) / T_{i1} \to T_{ij-i+1} \\
&\qquad\qquad \left(\begin{array}{l} i = 2, 3, \cdots, n-1 \\ j = i+1, i+2, \cdots i+m-1 \end{array} \right)
\end{aligned} \tag{1.22}
$$

となる．式 (1.22) では，図 1.3 の $1 \sim m$ 行目において $l = j - m + 1$ の関係が成立しないため，次に示す 2 つのパラメータを導入する．

$$
\begin{aligned}
&i_k(j) : \text{行列の第 } j \text{ 列にはじめて非零要素が現れる行番号} \\
&j_k(i) : \text{行列の第 } i \text{ 行について非零要素の最後の列番号}
\end{aligned} \tag{1.23}
$$

式 (1.23) によって表されるパラメータをあらかじめ求めておくと，前述の行列 T による変形コレスキー分解および連立 1 次方程式の解は次のようになる．

$$
\begin{aligned}
&\frac{T_{1j}}{T_{11}} \to T_{1j} \quad (j = 2, 3, \cdots, j_k(i)) \\
&T_{i1} - \sum_{l=i_k(i)}^{i-1} T_{li-l+1} T_{li-l+1} T_{l1} \to T_{i1} \quad (i = 2, 3, \cdots, n) \\
&\left(T_{ij-i+1} - \sum_{l=i_k(j)}^{i-1} T_{li-l+1} T_{lj-l-1} T_{l1} \right) / T_{i1} \to T_{ij-i+1} \\
&\qquad\qquad \left(\begin{array}{l} i = 2, 3, \cdots, n-1 \\ j = i+1, i+2, \cdots j_k(i) \end{array} \right)
\end{aligned} \tag{1.24}
$$

また，式 (1.18) と，式 (1.20) は次のように表すことができる．

$$
\begin{aligned}
&y_1 \to z_1 \\
&y_i - \sum_{l=i_k(i)}^{i-1} T_{li-l+1} y_l \to z_i \quad (i = 2, 3, \cdots, n)
\end{aligned} \tag{1.25}
$$

$$
\begin{aligned}
&x_n \to z_n / T_{n1} \\
&z_i / T_{i1} - \sum_{l=i+1}^{j_k(i)} T_{il-i+1} x_l \to x_i \quad (i = n-1, n-2, \cdots, 2, 1)
\end{aligned} \tag{1.26}
$$

このようにして長方形行列 T のみを用いて，行列の分解から未知量 $\boldsymbol{x}$ の計算までの処理ができる．

エクセルプログラム

プログラム 1.1　ガウスの消去法

表 1.1　ガウスの消去法プログラムリスト

```
Option Explicit

Const MaxGensuu = 10 ’元数…2 元 1 次方程式であれば 2 となり _
                4 元 1 次方程式であれば 4 になる
Const StringTable = "XYZUVWABCDEFGHIJKLMNOPQRST" ’添え字の出る順番


Private Sub CMD_Calculation_Click()
    Dim A(MaxGensuu, MaxGensuu) As Double
    Dim B(MaxGensuu) As Double
    Dim X(MaxGensuu) As Double
    Dim R(MaxGensuu) As Double
    Dim I As Integer
    Dim J As Integer
    Dim N As Integer
    Dim PVT As Double

    N = Range("_N")

    For I = 1 To N
        For J = 1 To N
            A(I, J) = Range("_AI").Offset(I - 1, (J - 1) * 2)
        Next J
        B(I) = Range("_AI").Offset(I - 1, N * 2)
    Next I

    Call Calculation(N, A, B, X, R)

’   ---------------クリア -----------
    For I = 1 To MaxGensuu
        Range("_S").Offset(I - 1, 0) = ""
        Range("_XI").Offset(I - 1, 0) = ""
        Range("_RI").Offset(I - 1, 0) = ""
    Next I

’   ---------------S ---------------
    For I = 1 To N
        Range("_S").Offset(I - 1, 0) = Mid(StringTable, I, 1)
    Next I
’   ---------------解---------------
    For I = 1 To N
```

```
        Range("_XI").Offset(I - 1, 0) = X(I)
    Next I
'   ---------------誤差---------------
    For I = 1 To N
        Range("_RI").Offset(I - 1, 0) = R(I)
    Next I
End Sub

Sub Calculation(N As Integer, _
                A() As Double, _
                B() As Double, _
                ByRef X() As Double, _
                ByRef R() As Double)
    Dim PVT As Double
    Dim DIAG As Double
    Dim L As Integer
    Dim J As Integer
    Dim K As Integer
    Dim AM As Double
    Dim S As Double
    Dim I As Integer
    Dim AO(10, 10) As Double
    Dim BO(10) As Double

    PVT = 0.00005
    For I = 1 To N
        BO(I) = B(I)
    Next I
    For J = 1 To N
        For I = 1 To N
            AO(I, J) = A(I, J)
        Next I
    Next J
'*****ガウスの消去法
'
'*****前進消去
    For L = 1 To N - 1 '(************
        If (Abs(A(L, L)) < PVT) Then
            MsgBox "ピボットが" & L & "番目の消去で小さくなり過ぎました"
            Exit Sub
            End
        End If
        DIAG = 1# / A(L, L)
        For J = L + 1 To N
            AM = A(J, L) * DIAG
            For K = L + 1 To N
                A(J, K) = A(J, K) - AM * A(L, K)
            Next K
            B(J) = B(J) - AM * B(L)
        Next J
        For K = L + 1 To N
            A(L, K) = A(L, K) * DIAG
        Next K
```

```
        B(L) = B(L) * DIAG
    Next L
'*****後退代入
    X(N) = B(N) / A(N, N)
    For J = N - 1 To 1 Step -1
        S = 0#
        For K = J + 1 To N
          S = S + A(J, K) * X(K)
        Next K
        X(J) = B(J) - S
    Next J
''*****誤差計算
    For I = 1 To N
        R(I) = 0#
        For J = 1 To N
          R(I) = R(I) + AO(I, J) * X(J)
        Next J
        R(I) = R(I) - BO(I)
    Next I

End Sub

Private Sub CMD_SetForm_Click()
    Dim N As Long
    Dim I As Long
    Dim J As Long
    Dim SOEJI As String

    N = Range("_N")
    For I = 1 To MaxGensuu
        For J = 1 To MaxGensuu
            Range("_SOEJI").Offset(I - 1, (J - 1) * 2) = ""
        Next J
    Next I

    For I = 1 To N
        For J = 1 To N
            If J <> N Then
                SOEJI = Mid(StringTable, J, 1) & " +"
            Else
                SOEJI = Mid(StringTable, J, 1) & " ="
            End If
            Range("_SOEJI").Offset(I - 1, (J - 1) * 2) = SOEJI
        Next J
    Next I
End Sub
```

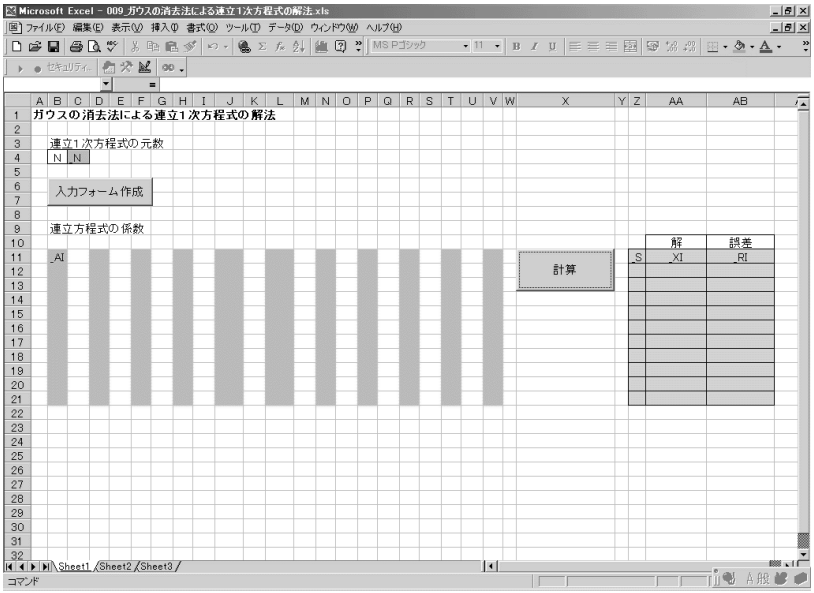

図 1.4　プログラム 1.1　ガウスの消去法のセルの名前

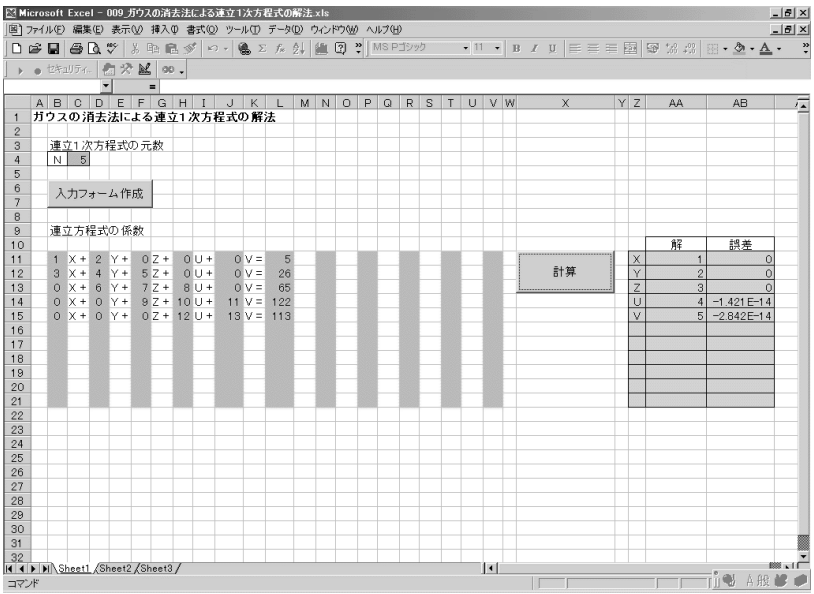

図 1.5　プログラム 1.1　ガウスの消去法の計算結果