



数値計算法 第9回

連立一次方程式 (3)

濱本 雄治¹

情報工学部 情報通信工学科

2025 年 6 月 16 日

¹hamamoto@c.oka-pu.ac.jp

帯行列



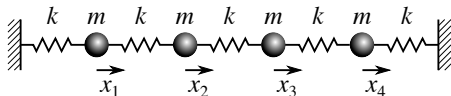
▶ 定義

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots \\ 0 & a_{32} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \cdots & 0 & a_{n,n-1} & a_{nn} \end{pmatrix} \quad (\text{三重対角行列})$$

▶ 特徴

- ▷ 要素の大部分がゼロ (**疎行列**) で、非ゼロ成分が対角線付近に存在
- ▷ 帯部分の要素だけ記憶すればよいのでメモリ効率がよい
- ▷ 計算を工夫することで計算量 $\sim \mathcal{O}(N \times \text{帯の幅})$

三重対角行列の例: バネで繋がれた質点系



▶ 運動方程式

$$\begin{cases} m\ddot{x}_1 = -2kx_1 + kx_2 \\ m\ddot{x}_2 = kx_1 - 2kx_2 + kx_3 \\ m\ddot{x}_3 = kx_2 - 2kx_3 + kx_4 \\ m\ddot{x}_4 = kx_3 - 2kx_4 \end{cases}$$

▶ 4つの質点と同じ振動数で $x_i = X_i \sin \omega t$ のように振動するとき

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix} = \frac{m\omega^2}{k} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix}$$

係数行列が帯行列の連立一次方程式



▶ 三重対角行列の場合

$$\begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots \\ 0 & a_{32} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \cdots & 0 & a_{n,n-1} & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{pmatrix}$$

▶ Gauss の消去法で解く場合の問題

- ▷ ピボット選択は帯構造を破壊 → 使用メモリと計算量の増加
- ▷ ピボット選択をしないとゼロに近いピボットで除算
→ 丸め誤差の累積と計算の不安定化

復習：非線形方程式の反復法



原理

1. 非線形方程式 $f(x) = 0$ を $x = g(x)$ の形に変形
2. 初期値 x_0 から漸化式

$$x_{k+1} = g(x_k)$$

で数列 $\{x_k\}$ を生成

3. 数列 $\{x_k\}$ が α に収束すれば α は非線形方程式の解

▶ 例：原始反復法

$$x_{k+1} = x_k - \underbrace{hf(x_k)}_{g(x_k)}$$

▶ 例：Newton 法

$$x_{k+1} = x_k - \underbrace{\frac{f(x_k)}{f'(x_k)}}_{g(x_k)}$$

連立一次方程式の反復法



原理

1. 連立一次方程式 $Ax = b$ を $x = Bx + c$ の形に変形
2. 初期値 x_0 から漸化式

$$x_{k+1} = Bx_k + c$$

でベクトル列 $\{x_k\}$ を生成

3. ベクトル列 $\{x_k\}$ が α に収束すれば α は連立一次方程式の解

Jacobi 法



- ▶ 係数 A を対角行列 D 、下三角行列 L 、上三角行列 U に分解

$$A = D + L + U$$

$$D = \begin{pmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & d_{nn} \end{pmatrix}, \quad L = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ \ell_{21} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \ell_{n1} & \cdots & \ell_{nn} & 0 \end{pmatrix}, \quad U = \begin{pmatrix} 0 & u_{12} & \cdots & u_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & u_{n-1,n} \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

- ▶ 漸化式の導出

$$(D + L + U)x = b, \quad x = D^{-1}[-(L + U)x + b],$$

$$\therefore x_{k+1} = \underbrace{-D^{-1}(L + U)}_B x_k + \underbrace{D^{-1}b}_c$$

- ▷ D^{-1} は D の対角要素を逆数に変更した対角行列であることに注意

Jacobi 法の例



▶ 2 変数の連立一次方程式

$$\begin{cases} 2x - y = 1 \\ -x + 2y = 1 \end{cases}, \quad \begin{cases} x = \frac{y+1}{2} \\ y = \frac{x+1}{2} \end{cases}$$

▶ 並列代入

$$(x, y) \leftarrow \left(\frac{y+1}{2}, \frac{x+1}{2} \right), \quad \therefore \begin{cases} x_{k+1} = \frac{y_k+1}{2} \\ y_{k+1} = \frac{x_k+1}{2} \end{cases}$$

(1) 古い x, y で右辺をすべて計算

(2) 左辺の変数にまとめて代入

Jacobi 法は並列計算に適している

Jacobi 法の計算例



▶ jacobi.c

```
#include <stdio.h>

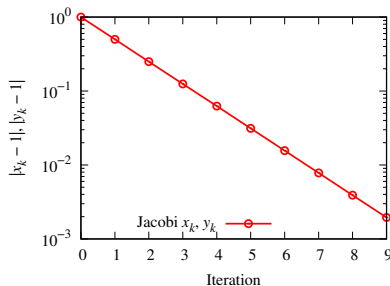
int main(){
    double x = 0;
    double y = 0;

    for(int i=0; i!=10; ++i){
        printf("%d %e %e\n", i, x, y);

        double tmp = x;
        x = (y + 1.0) / 2;
        y = (tmp + 1.0) / 2;
    }
}
```

▶ 実行結果

```
$ gcc jacobi.c; ./a.out
...
7 9.921875e-01 9.921875e-01
8 9.960938e-01 9.960938e-01
9 9.980469e-01 9.980469e-01
```



Gauss-Seidel 法



▶ 漸化式の導出

$$\begin{aligned}(\boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U})\boldsymbol{x} &= \boldsymbol{b}, & \boldsymbol{x} &= (\boldsymbol{D} + \boldsymbol{L})^{-1}[-\boldsymbol{U}\boldsymbol{x} + \boldsymbol{b}], \\ \therefore \boldsymbol{x}_{k+1} &= \underbrace{-(\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{U}}_{\boldsymbol{B}} \boldsymbol{x}_k + \underbrace{(\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{b}}_{\boldsymbol{c}}\end{aligned}$$

▶ Jacobi 法と比較すると

$$\triangleright \boldsymbol{D} \rightarrow \boldsymbol{D} + \boldsymbol{L}$$

$$\triangleright \boldsymbol{L} + \boldsymbol{U} \rightarrow \boldsymbol{U}$$

▶ Jacobi 法に類似した形で表すと

$$\boldsymbol{x}_{k+1} = -\boldsymbol{D}^{-1}(\boldsymbol{L}\boldsymbol{x}_{k+1} + \boldsymbol{U}\boldsymbol{x}_k) + \boldsymbol{D}^{-1}\boldsymbol{b}$$

Gauss-Seidel 法の例



- ▶ 2 変数の連立一次方程式 (ここまでは Jacobi 法と同じ)

$$\begin{cases} 2x - y = 1 \\ -x + 2y = 1 \end{cases}, \quad \begin{cases} x = \frac{y+1}{2} \\ y = \frac{x+1}{2} \end{cases}$$

- ▶ 逐次代入

$$x \leftarrow \frac{y+1}{2}; \quad y \leftarrow \frac{x+1}{2}, \quad \therefore \begin{cases} x_{k+1} = \frac{y_k+1}{2} \\ y_{k+1} = \frac{x_{k+1}+1}{2} = \frac{y_k+3}{2} \end{cases}$$

- (1) 古い y で $(y+1)/2$ を計算して x に代入
- (2) 新しい x で $(x+1)/2$ を計算して y に代入

y が早く更新されるため、一般に Jacobi 法より収束が早い

Gauss-Seidel 法の計算例



▶ gauss_seidel.c

```
#include <stdio.h>

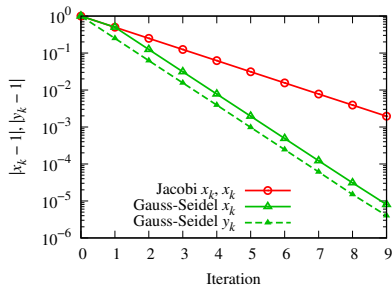
int main(){
    double x = 0;
    double y = 0;

    for(int i=0; i!=10; ++i){
        printf("%d %f %f\n", i, x, y);

        x = (y + 1.0) / 2;
        y = (x + 1.0) / 2;
    }
}
```

▶ 実行結果

```
$ gcc gauss_seidel.c; ./a.out
...
7 9.998779e-01 9.999390e-01
8 9.999695e-01 9.999847e-01
9 9.999924e-01 9.999962e-01
```



反復法の収束性



▶ y_k の収束性

$$y_{k+1} = \frac{y_k + 3}{2}, \quad \therefore y_{k+1} - 1 = \frac{1}{4}(y_k - 1) = \cdots = \left(\frac{1}{4}\right)^{k+1} (y_0 - 1)$$

$y \rightarrow 1$ に収束

▶ 順序を入れ替えた連立一次方程式:

$$\begin{cases} -x + 2y = 1 \\ 2x - y = 1 \end{cases}, \quad \therefore \begin{cases} x_{k+1} = 2y_k - 1 \\ y_{k+1} = 2x_{k+1} - 1 = 4y_k - 3 \end{cases}$$

$$\therefore y_{k+1} - 1 = 4(y_k - 1) = \cdots = 4^{k+1}(y_0 - 1)$$

$y \rightarrow \infty$ に発散



▶ 対角優位行列

$$|a_{ii}| > \sum_{j(\neq i)} |a_{ij}| \quad (i = 1, 2, \dots, n)$$

- ▶ 係数行列 A が対角優位行列なら Jacobi と Gauss-Seidel 法は任意の初期値に対して収束

▶ 上記の例

- ▷ 順序の入れ替え前

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \quad \rightarrow \quad \text{対角優位}$$

- ▷ 順序の入れ替え後

$$A = \begin{pmatrix} -1 & 2 \\ 2 & -1 \end{pmatrix} \quad \rightarrow \quad \text{対角優位でない}$$

Successive Over-Relaxation (SOR) 法



▶ Gauss-Seidel 法の漸化式

$$\mathbf{x}_{k+1}^{\text{GS}} = \mathbf{D}^{-1}[-(\mathbf{L}\mathbf{x}_{k+1} + \mathbf{U}\mathbf{x}_k) + \mathbf{b}]$$

▶ 修正量 $\mathbf{x}_{k+1}^{\text{GS}} - \mathbf{x}_k$ に加速パラメータ ω を掛けて収束を加速

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + \omega(\mathbf{x}_{k+1}^{\text{GS}} - \mathbf{x}_k) \\ &= (1 - \omega)\mathbf{x}_k + \omega\mathbf{D}^{-1}[-(\mathbf{L}\mathbf{x}_{k+1} + \mathbf{U}\mathbf{x}_k) + \mathbf{b}],\end{aligned}$$

$$(\mathbf{D} + \omega\mathbf{L})\mathbf{x}_{k+1} = (1 - \omega)\mathbf{D}\mathbf{x}_k - \omega\mathbf{U}\mathbf{x}_k + \omega\mathbf{b},$$

$$\therefore \mathbf{x}_{k+1} = (\mathbf{D} + \omega\mathbf{L})^{-1}[-\{(\omega - 1)\mathbf{D}\mathbf{x}_k + \omega\mathbf{U}\}\mathbf{x}_k + \omega\mathbf{b}]$$

▶ $\omega = 1$ のとき Gauss-Seidel 法を再現

▶ $1 < \omega < 2$ 、典型的には $\omega \sim 1.9$

SOR 法の例



▶ Gauss-Seidel 法の漸化式

$$\begin{cases} x_{k+1}^{\text{GS}} = \frac{y_k + 1}{2} \\ y_{k+1}^{\text{GS}} = \frac{x_{k+1} + 1}{2} \end{cases}$$

▶ SOR 法の漸化式

$$\begin{cases} x_{k+1} = x_k + \omega(x_{k+1}^{\text{GS}} - x_k) = x_k + \omega\left(\frac{y_k + 1}{2} - x_k\right) \\ y_{k+1} = y_k + \omega(y_{k+1}^{\text{GS}} - y_k) = y_k + \omega\left(\frac{x_{k+1} + 1}{2} - y_k\right) \end{cases}$$

SOR法の計算例



▶ gauss_seidel.c

```
#include <stdio.h>

int main(){
    double x = 0;
    double y = 0;
    double w = 1.1;

    for(int i=0; i!=10; ++i){
        printf("%d %.10e %.10e\n",
               i, x, y);

        x = (1-w) * x + w * (y+1.0)/2;
        y = (1-w) * y + w * (x+1.0)/2;
    }
}
```

▶ 実行結果

```
$ gcc gauss_seidel.c; ./a.out
...
7 9.99999956491e-01 9.9999985059e-01
8 9.99999996133e-01 9.9999999367e-01
9 1.00000000004e+00 1.00000000008e+00
```

