

# 情報数学C

## *Mathematics for Informatics C*

第3回線形連立方程式の反復解法  
(ヤコビ法, ガウス・ザイデル法, SOR法, 共役勾配法)

情報メディア創成学類  
藤澤誠

# 今回の講義内容

- **今日の問題**
- ヤコビ法とガウス・ザイデル法
- SOR法
- 共役勾配法
- 前処理付き共役勾配法

# 今回の講義で扱う問題

$$Ax = b$$

# 今回の講義で扱う問題

前回と同じ線形システム

$$Ax = b$$

- 前回とは解き方が異なる  $\Rightarrow$  反復法
- 前回の解き方の復習 (直接解法)
  - ガウスの消去法
  - 三角分解

線形システムを直接解いているので  
打ち切り誤差はない(丸め誤差はありうる)

# 直接解法の計算時間

## 直接解法の計算時間は？

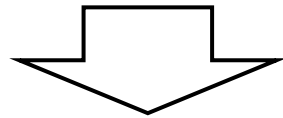
$n$	$O(n^2)$ の理論的な計算時間	$O(n^3)$ の理論的な計算時間
10	$2.5 \times 10^{-11}$ 秒	$2.5 \times 10^{-10}$ 秒
$10^2$	$2.5 \times 10^{-9}$ 秒	$2.5 \times 10^{-7}$ 秒
$10^3$	$2.5 \times 10^{-7}$ 秒	$2.5 \times 10^{-4}$ 秒
$10^4$	$2.5 \times 10^{-5}$ 秒	0.25秒
$10^5$	0.0025秒	250秒
$10^6$	0.25秒	$2.5 \times 10^5$ 秒=約69.4時間
$10^7$	25秒	$2.5 \times 10^8$ 秒=約7.94年

ガウスの消去法が $O(n^3)$ ,  
三角分解で $O(n^2)$

# 反復解法とは？

Aが変わらなければ三角分解でいいけど、  
そのような問題ばかりではない。

⇒ 前計算なしで $O(n^2)$ ぐらいにできないか？



反復解法

今回勉強するガウス・ザイデル法で $O(kn^2)$   
( $k$ は1以上の整数)

# 反復解法とは？

## 反復解法の特徴

- 反復処理で初期値から徐々に解に近づけていく方法
- 直接解法と異なり, **厳密な解ではない\***  
(打ち切り誤差が発生するため)
- 計算コストは $O(kn^2)$ 以下  
⇒ 反復回数 $k \ll n$ なら直接解法より  
**高速**に解ける.

# 今回の講義内容

- 今日の問題
- ヤコビ法とガウス・ザイデル法
- SOR法
- 共役勾配法
- 前処理付き共役勾配法



# 表記法の復習

## 線形システム $A\mathbf{x} = \mathbf{b}$ の表記法の復習

C言語のコードとの対応をとるために

- インデックス0スタート ( $a_{11} \sim a_{nn} \Rightarrow a_{0,0} \sim a_{n-1,n-1}$ )
- 右辺項を  $n$  列目として表記 ( $b_1 \sim b_n \Rightarrow a_{0,n} \sim a_{n-1,n}$ )

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} a_{0,n} \\ a_{1,n} \\ a_{2,n} \\ \vdots \\ a_{n-1,n} \end{pmatrix}$$

右辺までまとめて  $n \times (n + 1)$  の行列にしたものを**拡大行列**という

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & a_{n-1,n} \end{pmatrix}$$

# ヤコビ法

線形システムの反復解法で最も基本的なものがヤコビ法

- 3元連立方程式で考えてみよう

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 = b_0 \\ a_{10}x_0 + a_{11}x_1 + a_{12}x_2 = b_1 \\ a_{20}x_0 + a_{21}x_1 + a_{22}x_2 = b_2 \end{cases}$$

1つ目の式を $x_0$ について解くと:

$$x_0 = \frac{b_0 - a_{01}x_1 - a_{02}x_2}{a_{00}}$$

$x_1, x_2$ が分かっていないのだから当然解けないんだけど...

# ヤコビ法

$x_1, x_2$  についての式も求めて,  $x_i$  に何らかの初期値  $x_i^{(0)}$  を与えたとする...

$$x_0^{(1)} = \frac{b_0 - a_{01}x_1^{(0)} - a_{02}x_2^{(0)}}{a_{00}}$$

$$x_1^{(1)} = \frac{b_1 - a_{10}x_0^{(0)} - a_{12}x_2^{(0)}}{a_{11}}$$

$$x_2^{(1)} = \frac{b_2 - a_{20}x_0^{(0)} - a_{21}x_1^{(0)}}{a_{22}}$$

$x_i^{(0)}$  から  $x_i^{(1)}$  を計算  $\Rightarrow$  もしこの計算で  $x_i^{(1)}$  が  $x_i^{(0)}$  より解に近づいているならば, この計算を繰り返せばいつか解に収束する

# ヤコビ法

$k$  番目の繰り返しにおける値を  $x_i^{(k)}$  とすると:

$$x_0^{(k+1)} = \frac{b_0 - a_{01}x_1^{(k)} - a_{02}x_2^{(k)}}{a_{00}}$$

$$x_1^{(k+1)} = \frac{b_1 - a_{10}x_0^{(k)} - a_{12}x_2^{(k)}}{a_{11}}$$

$$x_2^{(k+1)} = \frac{b_2 - a_{20}x_0^{(k)} - a_{21}x_1^{(k)}}{a_{22}}$$

この計算を  $|x_i^{(k)} - x_i^{(k+1)}| < \varepsilon$  となるまで反復

ヤコビ法

収束するまでの反復数を  $K$  とすると  
計算量は  $O(Kn^2)$

# ヤコビ法

$n \times n$ の線形システムについてのヤコビ法

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j=0 \sim n-1, j \neq i} a_{i,j} x_j^{(k)} \right) \quad (i = 0, 1, \dots, n-1)$$

計算手順

1. 初期値 $x_i^{(0)}$ を設定( $i = 0 \sim n-1$ )
2. 以下の処理を反復( $k = 0, 1, \dots$ )
  - a. 上の式により, $i = 0 \sim n-1$ について $x_i^{(k+1)}$ を計算\*
  - b.  $\left| x_i^{(k)} - x_i^{(k+1)} \right| < \varepsilon$ となったら反復終了

\* $a_{i,i}$ で割るという計算が入っているが、ガウスの消去法と異なり、計算途中で $a_{i,j}$ は変わらないのであらかじめ対角成分を調べておけばよい(収束条件にも関わる)

# ヤコビ法

## ヤコビ法のコード例

配列yに $x^{(k+1)}$ , 配列xに $x^{(k)}$ を格納

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j=0 \sim n-1, j \neq i} a_{i,j} x_j^{(k)} \right)$$

```
int k;
for(k = 0; k < max_iter; ++k){
    for(int i = 0; i < n; ++i){
        y[i] = A[i][n];
        for(int j = 0; j < n; ++j){
            y[i] -= (j != i ? A[i][j]*x[j] : 0.0);
        }
        y[i] /= A[i][i];
    }
}
```

// 収束判定

```
int l;
```

```
for(l = 0; l < n; ++l) if(fabs(y[l]-x[l]) > eps) break; // 絶対誤差
if(l == n) break; // すべての解が許容誤差以下なら反復終了
```

```
swap(x, y);
```

```
}
```

$x^{(k+1)}$ を $x^{(k)}$ として次の反復へ

$b_i$ は拡大行列での  
 $a_{i,n}$ に相当

$j \neq i$ だったら  
加算していく

相対誤差を使う場合は,  
 $\text{fabs}((y[i]-x[i])/y[i]) > \text{eps};$

このコードでは収束しなくてもユーザが設定した最大反復回数max\_iterで打ち切るようにしている(無限ループを防ぐため)

# ヤコビ法

## ヤコビ法の実行結果例1

$$\begin{pmatrix} 8 & 1 & 3 \\ 1 & 5 & 2 \\ 3 & 1 & 7 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 9 \\ -2 \end{pmatrix} \Rightarrow \begin{pmatrix} 8 & 1 & 3 & 7 \\ 1 & 5 & 2 & 9 \\ 3 & 1 & 7 & -2 \end{pmatrix}$$

拡大行列表記

解)  $(x_0, x_1, x_2) = (1, 2, -1)$

- 初期値として  $(x_0^{(0)}, x_1^{(0)}, x_2^{(0)}) = (0, 0, 0)$  を設定
- 収束判定には絶対誤差を使用

$$\left| x_i^{(k+1)} - x_i^{(k)} \right| \leq \varepsilon$$

- $\varepsilon = 1 \times 10^{-6}$  を設定

# ヤコビ法

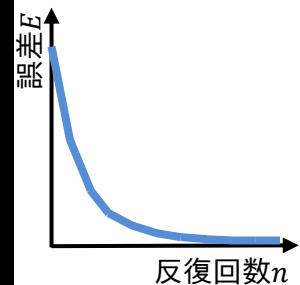
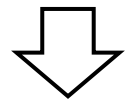
$$\begin{pmatrix} 8 & 1 & 3 & 7 \\ 1 & 5 & 2 & 9 \\ 3 & 1 & 7 & -2 \end{pmatrix}$$

## ヤコビ法の実行結果例1

解)  $(x_0, x_1, x_2) = (1, 2, -1)$

0	:	$x_0 = 0,$	$x_1 = 0,$	$x_2 = 0$
1	:	$x_0 = 0.875,$	$x_1 = 1.8,$	$x_2 = -0.285714$
2	:	$x_0 = 0.757143,$	$x_1 = 1.73929,$	$x_2 = -0.917857$
3	:	$x_0 = 1.00179,$	$x_1 = 2.01571,$	$x_2 = -0.858673$
4	:	$x_0 = 0.945038,$	$x_1 = 1.94311,$	$x_2 = -1.00301$
5	:	$x_0 = 1.00824,$	$x_1 = 2.0122,$	$x_2 = -0.968318$
6	:	$x_0 = 0.986595,$	$x_1 = 1.98568,$	$x_2 = -1.00527$
7	:	$x_0 = 1.00377,$	$x_1 = 2.00479,$	$x_2 = -0.992209$
	:	$\vdots$		
22	:	$x_0 = 0.999999,$	$x_1 = 2,$	$x_2 = -1$
23	:	$x_0 = 1,$	$x_1 = 2,$	$x_2 = -1$

最初の方は  
変化が大きい  
けど徐々に変  
化が小さく



反復24回目で許容誤差内の解に収束

⇒ 必ず収束するのか？



# ヤコビ法

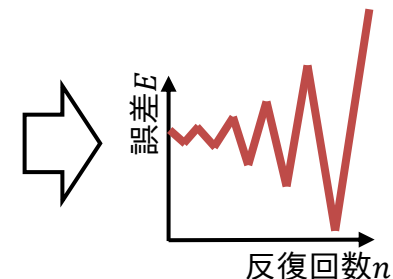
## ヤコビ法の実行結果例2

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 4 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 9 \\ 1 \\ 4 \end{pmatrix}$$

解)  $(x_0, x_1, x_2) = (1, -2, 3)$

```
0 : x0 = 0,          x1 = 0, x2 = 0
1 : x0 = 4.5,         x1 = 0.333333, x2 = 1.33333
2 : x0 = 2.33333,     x1 = -2.05556, x2 = -3.61111
3 : x0 = 10.9444,     x1 = 1.96296, x2 = 1.74074
4 : x0 = 0.907407,    x1 = -4.47531, x2 = -12.2284
  :
98 : x0 = -1.33e+24,  x1 = -7.49e+23, x2 = -1.32e+24
99 : x0 = 2.35e+24,   x1 = 1.32e+24,  x2 = 2.33e+24
```

反復で解の値がどんどん大きくなって  
いき収束しない⇒解の**発散**



# ヤコビ法の収束条件

ヤコビ法で収束するための条件は？

$$x_i^{(k+1)} = \frac{1}{\underline{a_{i,i}}} \left( b_i - \sum_{\underline{j=0 \sim n-1, j \neq i}} a_{i,j} x_j^{(k)} \right) \quad (i = 0, 1, \dots, n-1)$$

$D^{-1}$                        $L + U$

行列  $A = \{a_{i,j}\}$  を  $D + L + U$  と分解してみる

$$D = \begin{pmatrix} a_{0,0} & 0 & \cdots & 0 \\ 0 & a_{1,1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n-1,n-1} \end{pmatrix} \quad \Rightarrow \quad a_{i,i} \quad (i = j)$$

$$L = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{1,0} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & 0 \end{pmatrix} \quad U = \begin{pmatrix} 0 & a_{0,1} & \cdots & a_{0,n-1} \\ 0 & 0 & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \Rightarrow a_{i,j} \quad (i \neq j)$$

# ヤコビ法の収束条件

ヤコビ法の更新式を行列表記してみる

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j=0 \sim n-1, j \neq i} a_{i,j} x_j^{(k)} \right) \quad (i = 0, 1, \dots, n-1)$$

⇩ ベクトルと行列を用いた表記

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

⇩  $H = -D^{-1}(L + U)$ ,  $\mathbf{z} = D^{-1}\mathbf{b}$ と置く

$$\mathbf{x}^{(k+1)} = H\mathbf{x}^{(k)} + \mathbf{z}$$

十分な数( $K$ 回)反復して収束したとすると:

$$\mathbf{x}^{(K)} = H\mathbf{x}^{(K)} + \mathbf{z}$$

2つの式から以下が導出できる

$$\boxed{\mathbf{x}^{(K)} - \mathbf{x}^{(k+1)} = H(\mathbf{x}^{(K)} - \mathbf{x}^{(k)})}$$

# ヤコビ法の収束条件

ヤコビ法で収束するための条件は？

$$\underline{x^{(K)} - x^{(k+1)}} = H(\underline{x^{(K)} - x^{(k)}})$$

$k + 1$ ステップ目  
の誤差

$k$ ステップ目  
の誤差

反復する毎に誤差が小さくなれば収束するのだから  
条件は:

$$\|H\| < 1 \quad \text{ここで } H = -D^{-1}(L + U)$$



$$\frac{\sum_{j \neq i} |a_{i,j}|}{|a_{i,i}|} < 1$$

$$(i = 0, 1, \dots, n - 1)$$

\*  $\|H\|$ は行列のノルム 20

# ヤコビ法の収束条件

$$\frac{\sum_{j \neq i} |a_{i,j}|}{|a_{i,i}|} < 1$$

$i$ 行目の非対角成分の  
絶対値の和

$i$ 行目の対角成分の  
絶対値

ヤコビ法で収束するための十分条件\*

係数行列 $A$ の各行において対角要素の絶対値  
が非対角要素の絶対値の和より大きい\*\*

$$|a_{i,i}| > \sum_{j \neq i} |a_{i,j}|$$

対角優位

\*必要十分条件ではない. 対角優位でなくても収束することもある.

\*\*対角優位な行列の最大固有値の絶対値(スペクトル半径)は1より小さくなるのでこれを条件とする場合もある.

# ヤコビ法の収束条件

ヤコビ法の2つの例を比べてみよう

$$\begin{pmatrix} 8 & 1 & 3 \\ 1 & 5 & 2 \\ 3 & 1 & 7 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 9 \\ -2 \end{pmatrix}$$

$$1\text{行目} : |a_{i,i}| = 8, \quad \sum_{j \neq i} |a_{i,j}| = 4 \Rightarrow \bigcirc$$

$$2\text{行目} : |a_{i,i}| = 5, \quad \sum_{j \neq i} |a_{i,j}| = 3 \Rightarrow \bigcirc$$

$$3\text{行目} : |a_{i,i}| = 7, \quad \sum_{j \neq i} |a_{i,j}| = 4 \Rightarrow \bigcirc$$

} 収束する

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 4 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 9 \\ 1 \\ 4 \end{pmatrix}$$

$$1\text{行目} : |a_{i,i}| = 2, \quad \sum_{j \neq i} |a_{i,j}| = 4 \Rightarrow \times$$

$$2\text{行目} : |a_{i,i}| = 3, \quad \sum_{j \neq i} |a_{i,j}| = 3 \Rightarrow \triangle$$

$$3\text{行目} : |a_{i,i}| = 3, \quad \sum_{j \neq i} |a_{i,j}| = 7 \Rightarrow \times$$

} 収束しない

対角優位でなかったらどうするのか？

⇒ 並び替えで対角優位化(MC64オーダリング\*など)

\*Duffら, "On algorithms for permuting large entries to the diagonal of a sparse matrix", SIAM J. Matrix Anal. Appl. 22(4), pp.973-996, 2001.

# ガウス・ザイデル法

ヤコビ法の収束性を向上させるにはどうすれば良いか？

三元連立方程式の場合の式をもう一度見てみよう！

$$x_0^{(k+1)} = \frac{b_0 - a_{01}x_1^{(k)} - a_{02}x_2^{(k)}}{a_{00}}$$

$$x_1^{(k+1)} = \frac{b_1 - a_{10}\textcolor{red}{x}_0^{(k)} - a_{12}x_2^{(k)}}{a_{11}}$$

$$x_2^{(k+1)} = \frac{b_2 - a_{20}\textcolor{red}{x}_0^{(k)} - a_{21}\textcolor{red}{x}_1^{(k)}}{a_{22}}$$

上から順番に計算してるなら、赤字で示した

$\textcolor{red}{x}_0^{(k)}, \textcolor{red}{x}_1^{(k)}, \textcolor{red}{x}_2^{(k)}$

はすでに新しい値( $x^{(k+1)}$ )が計算済みでは？



# ガウス・ザイデル法

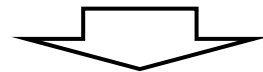
計算済みの $(k + 1)$ ステップでの値があるのならそれを使えば収束は早くなるのでは？

$$x_0^{(k+1)} = \frac{b_0 - a_{01}x_1^{(k)} - a_{02}x_2^{(k)}}{a_{00}}$$

$$x_1^{(k+1)} = \frac{b_1 - \textcolor{blue}{a}_{10}x_0^{(k+1)} - a_{12}x_2^{(k)}}{a_{11}}$$

$$x_2^{(k+1)} = \frac{b_2 - \textcolor{blue}{a}_{20}x_0^{(k+1)} - \textcolor{blue}{a}_{21}x_1^{(k+1)}}{a_{22}}$$

ガウス・  
ザイデル法



$x^{(k+1)}$ に掛かっている係数(青字)はすべて $i > j$   
( $i = 0$ から順番に計算しているので)  
⇒ 左下非対角成分



# ガウス・ザイデル法

$n \times n$ の線形システムについてのガウス・ザイデル法

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j=0 \sim i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1 \sim n-1} a_{i,j} x_j^{(k)} \right)$$

↓ 行列表記

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - L\mathbf{x}^{(k+1)} - U\mathbf{x}^{(k)}) \quad (i = 0, 1, \dots, n-1)$$

計算手順(ヤコビ法と同じ)

1. 初期値 $x_i^{(0)}$ を設定( $i = 0 \sim n-1$ )
2. 以下の処理を反復( $k$ 回目の反復)
  - a. 上の式により, $i = 0 \sim n-1$ について $x_i^{(k+1)}$ を計算
  - b.  $|x_i^{(k)} - x_i^{(k+1)}| < \varepsilon$ となったら反復終了

# ガウス・ザイデル法

## ガウス・ザイデル法のコード例

```
int k; // 計算反復回数
for(k = 0; k < max_iter; ++k){
    int l = 0;
    for(int i = 0; i < n; ++i){
        double tmp = x[i];
        x[i] = A[i][n];
        for(int j = 0; j < n; ++j){
            x[i] -= (j != i ? A[i][j]*x[j] : 0.0);
        }
        x[i] /= A[i][i];

        if(fabs(tmp-x[i]) > eps) l++;
    }

    // 収束判定
    if(l == 0) break; // すべての解が許容誤差以下なら反復終了
}
```

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j=0 \sim i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1 \sim n-1} a_{i,j} x_j^{(k)} \right)$$

$b_i$ は拡大行列  
での $a_{i,n}$ に相当

$j \neq i$ だったら  
加算していく

$x^{(k+1)}$ と $x^{(k)}$ を同じ配列に  
格納することで計算済み  
の値を使うことができる  
(ヤコビ法と違うところ)

相対誤差を使う場合は,  
 $\text{fabs}(tmp-x[i])/tmp > \text{eps};$

# ガウス・ザイデル法

## ガウス・ザイデル法の実行結果例

$$\begin{pmatrix} 8 & 1 & 3 \\ 1 & 5 & 2 \\ 3 & 1 & 7 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 9 \\ -2 \end{pmatrix} \Rightarrow \begin{pmatrix} 8 & 1 & 3 & 7 \\ 1 & 5 & 2 & 9 \\ 3 & 1 & 7 & -2 \end{pmatrix}$$

拡大行列表記

解)  $(x_0, x_1, x_2) = (1, 2, -1)$

- 初期値として  $(x_0^{(0)}, x_1^{(0)}, x_2^{(0)}) = (0, 0, 0)$  を設定
- 収束判定には絶対誤差を使用
$$\left| x_i^{(k+1)} - x_i^{(k)} \right| \leq \varepsilon$$
- $\varepsilon = 1 \times 10^{-6}$  を設定
- 対角優位 (収束条件はヤコビ法と同じ)

# ガウス・ザイデル法

$$\begin{pmatrix} 8 & 1 & 3 & 7 \\ 1 & 5 & 2 & 9 \\ 3 & 1 & 7 & -2 \end{pmatrix}$$

## ガウス・ザイデル法の実行結果例

解)  $(x_0, x_1, x_2) = (1, 2, -1)$

0	:	$x_0 = 0.875,$	$x_1 = 1.625,$	$x_2 = -0.892857$
1	:	$x_0 = 1.0067,$	$x_1 = 1.9558,$	$x_2 = -0.996556$
2	:	$x_0 = 1.00423,$	$x_1 = 1.99778,$	$x_2 = -1.0015$
3	:	$x_0 = 1.00084,$	$x_1 = 2.00043,$	$x_2 = -1.00042$
4	:	$x_0 = 1.0001,$	$x_1 = 2.00015,$	$x_2 = -1.00007$
5	:	$x_0 = 1.00001,$	$x_1 = 2.00003,$	$x_2 = -1.00001$
6	:	$x_0 = 0.999999,$	$x_1 = 2,$	$x_2 = -1$
7	:	$x_0 = 1,$	$x_1 = 2,$	$x_2 = -1$
8	:	$x_0 = 1,$	$x_1 = 2,$	$x_2 = -1$

反復9回目で許容誤差内の解に収束(ヤコビ法だと24回)

⇒ ヤコビ法の半分以上の反復で収束

# ガウス・ザイデル法

収束が早いのなら**ガウス・ザイデル法**だけでいいのでは？

⇒ 最近はや**コビ法**が使われることも多い

ガウス・ザイデル法の欠点

式の中に $i - 1$ 行目の計算結果 $x_{i-1}^{(k+1)}$ が含まれるので、  
 $i = 0$ から $n - 1$ で**逐次的に**計算しなければならない

**並列計算**しにくい(できなくはない\*けど…)



ヤコビ法は $i = 0$ から $n - 1$ まですべて独立して計算可能  
⇒ **並列化が容易**

# 今回の講義内容

- 今日の問題
- ヤコビ法とガウス・ザイデル法
- **SOR法**
- 共役勾配法
- 前処理付き共役勾配法

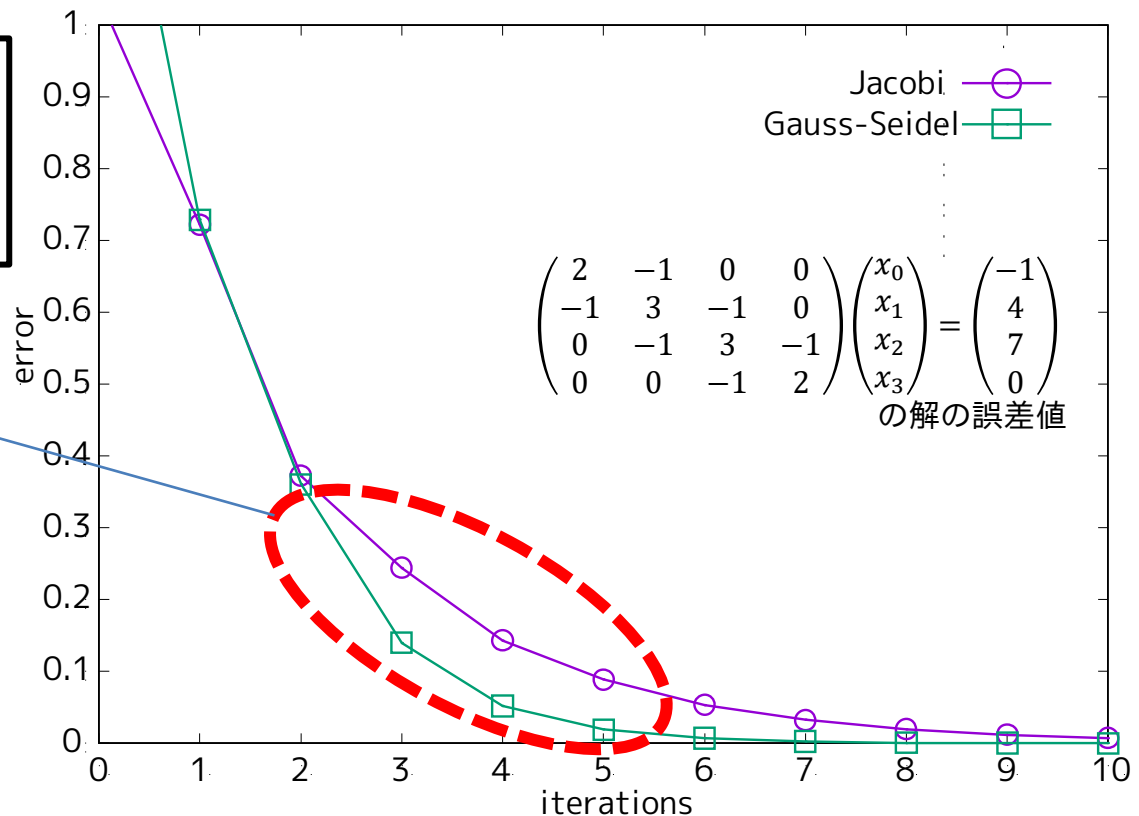
# SOR法

計算の並列性を保ったまま反復解法の**収束性**や**安定性**を高めるには？

ヤコビ法と比べて、  
ガウス・ザイデル法は  
より**早く収束**している



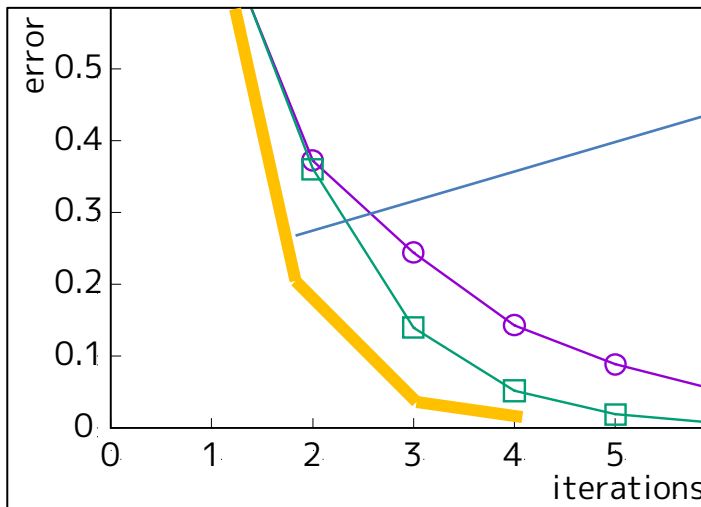
収束時の**傾き**がより  
大きいことに注目



反復回数(横軸)に対する誤差(縦軸)をプロットしたグラフ

# SOR法

傾きを大きくするにはどうすれば良いのか？



こんな風に**傾きが大きくなれば**  
**収束は早い**(反復数が少なくなる)?

誤差グラフ上の傾き:

$$x_i^{(k+1)} - x_i^{(k)}$$

$k$ ステップ目の解と $k + 1$ ステップ  
での解の差が傾き


例えば, ヤコビ法の式で得られた解 $x^{(k+1)} = \frac{1}{a_{i,i}}(b_i - \sum_{j=0 \sim n-1, j \neq i} a_{i,j} x_j^{(k)})$ を  
そのまま使わずに傾きを大きくするようにしたら...



# SOR法

## 逐次加速緩和法

(Successive Over-Relaxation : SOR法)

$$x_i^{(k+1)} = x_i^{(k)} + \omega \underbrace{(x_i^{(k+1)} - x_i^{(k)})}_{\text{誤差グラフの傾き}}$$


傾きを大きく(加速)するための係数

桁落ちを防ぎたい場合は:

$$x_i^{(k+1)} = (1 - \omega)x_i^{(k)} + \omega x_i^{(k+1)}$$

# SOR法

## 加速係数 $\omega$ の決め方

- 基本的には $1 < \omega < 2$ の範囲で  
**問題ごとに設定**する必要がある\*  
(安定性のために $0 < \omega < 1$ を使うことはある)
- 正定値対称行列を係数とする $Ax = b$ については,  
 $0 < \omega < 2$ で**必ず収束**することが証明されている(Ostrowskiの定理)
- $\omega = 1$ でガウス・ザイデル法と同じ.  
 $\omega < 1$ でガウス・ザイデル法で収束しなかった  
問題でも収束する可能性あり(あくまで可能性だけど)

\*問題によっては自動で決定できるものもあるが、一般的にこれという決定法があるわけではなく、問題毎に試行錯誤することが多い. 34

# SOR法

## SOR法のコード例

```
int k; // 計算反復回数
for(k = 0; k < max_iter; ++k){
    int l = 0;
    for(int i = 0; i < n; ++i){
        double tmp = x[i];
        x[i] = A[i][n];
        for(int j = 0; j < n; ++j){
            x[i] -= (j != i ? A[i][j]*x[j] : 0.0);
        }
        x[i] /= A[i][i];

        // 加速緩和係数wを使って次の解を計算
        x[i] = (1.0-w)*tmp+w*x[i];
    }

    // 収束判定
    . . .
}
```

ここはガウス・  
ザイデル法と全  
く同じ

ここがSOR法の計算  
$$x_i^{(k+1)} = (1 - \omega)x_i^{(k)} + \omega x_i^{(k+1)}$$

# SOR法

SOR法の実行結果例(ヤコビ法, ガウス・ザイデルと問題が違うので注意)

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 4 \\ 7 \\ 0 \end{pmatrix} \quad \text{解) } (1, 3, 4, 2)$$

加速係数は  $\omega = 1.15$  と設定 (0.05刻みで試行した結果)

```
0 : x0 = -0.575,      x1 = 1.3129167, x2 = 3.1866181, x3 = 1.8323054
1 : x0 = 0.26617708,  x1 = 2.6599673, x2 = 3.9273785, x3 = 1.9833968
2 : x0 = 0.91455464,  x1 = 2.9904126, x2 = 4.0008535, x3 = 2.0029812
3 : x0 = 1.0073041,   x1 = 3.0045652, x2 = 4.0027648, x3 = 2.0011426
4 : x0 = 1.0015294,   x1 = 3.0009613, x2 = 4.0003918, x3 = 2.0000539
5 : x0 = 1.0003233,   x1 = 3.0001299, x2 = 4.0000117, x3 = 1.9999986
6 : x0 = 1.0000262,   x1 = 2.999995,  x2 = 3.9999958, x3 = 1.9999978
7 : x0 = 0.99999322,  x1 = 2.9999965, x2 = 3.9999985, x3 = 1.9999994
8 : x0 = 0.99999903,  x1 = 2.9999996, x2 = 3.9999998, x3 = 2
9 : x0 = 0.99999989,  x1 = 3,          x2 = 4,          x3 = 2
```

反復10回目で許容誤差( $1 \times 10^{-6}$ )内の解に収束

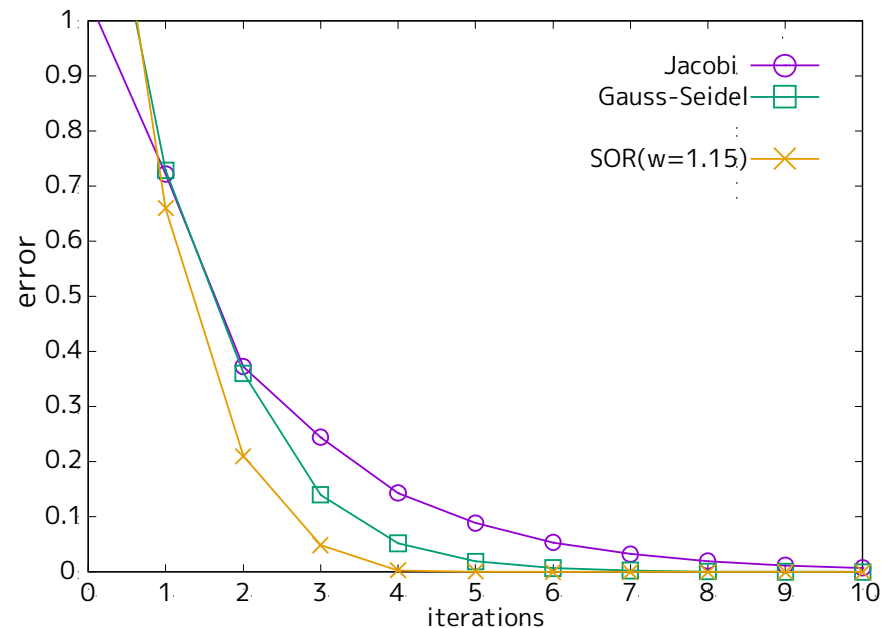
# SOR法

## ヤコビ法, ガウス・ザイデル法との反復回数の比較

ヤコビ:30回, ガウス・ザイデル:17回, SOR法( $\omega = 1.15$ ):10回

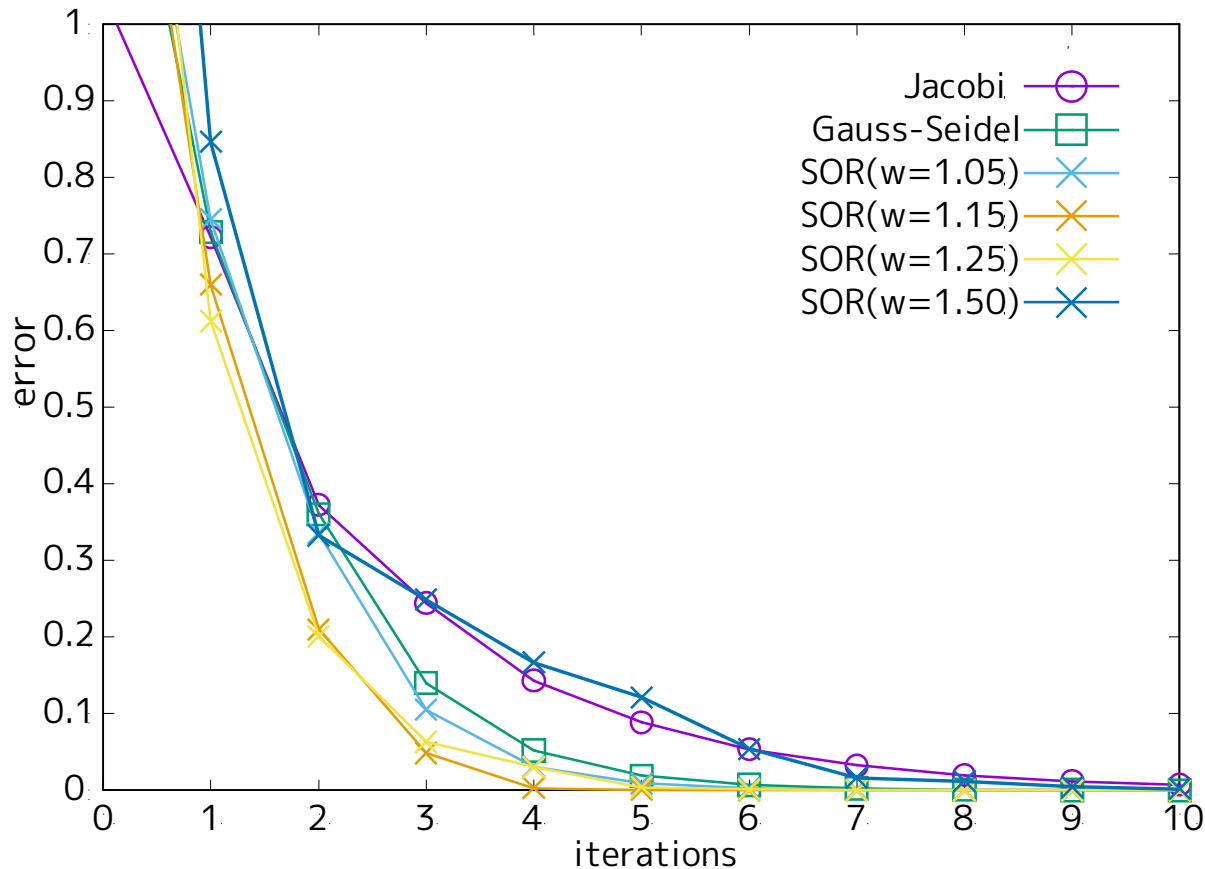
1回の反復にかかる計算コストはほぼ同じなので,

- ヤコビ法に対して**3倍の高速化**
- ガウス・ザイデル法に対して**1.7倍の高速化**



# SOR法

加速係数を変えた場合は？



$\omega$	反復回数
1.05	15
1.15	10
1.25	13
1.5	24

係数行列 $A$ によっては  
最適な $\omega$ を計算可能  
だけど、多くの場合は  
**問題毎に値を選択**  
する必要あり

# 今回の講義内容

- 今日の問題
- ヤコビ法とガウス・ザイデル法
- SOR法
- **共役勾配法**
- 前処理付き共役勾配法

# 共役勾配法の前に

## 残差について

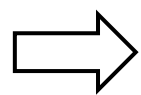
$Ax = b$  を変形すると  $b - Ax = 0$

$x$  が真値ならこの式となるが、 $k$  ステップ目の近似解の段階では、 $b - Ax^{(k)} \neq 0$



**残差**  $r^{(k)} = b - Ax^{(k)}$

(ここでの残差はベクトル  
ということに注意)



反復処理で**残差を如何にして  
ゼロベクトルに近づけるか**が重要  
⇒ 残差についてもう少し考えてみよう!



# 共役勾配法の前に

$Ax = b$  の両辺に  $Ix$  を足す

$$Ax + Ix = b + Ix \quad \Leftrightarrow \quad \underline{x = b + (I - A)x}$$

これを反復式とすると

$$x^{(k+1)} = b + (I - A)x^{(k)} = (b - Ax^{(k)}) + x^{(k)}$$

$$x^{(k+1)} = \underline{r^{(k)}} + x^{(k)}$$

残差ベクトル

$k + 1$  ステップでの残差ベクトルは:

$$r^{(k+1)} = b - Ax^{(k+1)} \leftarrow b - A(r^{(k)} + x^{(k)})$$

$r^{(k)} = b - Ax^{(k)}$  も使って整理すると:  $\underline{r^{(k+1)} = (I - A)r^{(k)}}$

代入

# 共役勾配法の前に

前のページの赤枠の式を初期値 $\mathbf{x}^{(0)}$ ,  $\mathbf{r}^{(0)}$ との関係式にしてみると:

$$\mathbf{r}^{(k)} = (\mathbf{I} - \mathbf{A})^k \mathbf{r}^{(0)}$$

$$\mathbf{x}^{(k+1)} = \mathbf{r}^{(k)} + \mathbf{r}^{(k-1)} + \dots + \mathbf{r}^{(0)} + \mathbf{x}^{(0)}$$

上の式を下の式に代入してやると,

$\mathbf{r}^{(k+1)}$ ,  $\mathbf{x}^{(k+1)}$  は

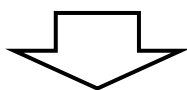
$\mathbf{r}^{(0)}, \mathbf{A}\mathbf{r}^{(0)}, \dots, \mathbf{A}^k \mathbf{r}^{(0)}$  の 線形結合 で表される

$c_0 \mathbf{r}^{(0)} + c_1 \mathbf{A}\mathbf{r}^{(0)} + c_2 \mathbf{A}^2 \mathbf{r}^{(0)} + \dots + c_k \mathbf{A}^k \mathbf{r}^{(0)}$  という形の式

# クリロフ部分空間

ベクトル  $\mathbf{r}^{(0)}, A\mathbf{r}^{(0)}, \dots, A^k\mathbf{r}^{(0)}$  の線形結合で表される部分空間

$$\mathbf{x}^{(k+1)} \in \text{span}\{\mathbf{r}^{(0)}, A\mathbf{r}^{(0)}, \dots, A^k\mathbf{r}^{(0)}\}$$



## クリロフ部分空間

$$\mathcal{K}_n(A, \mathbf{v}) = \text{span}\{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \dots, A^{n-1}\mathbf{v}\}$$

$\mathbf{x}^{(0)}$  からスタートして, クリロフ部分空間  $\mathcal{K}_n$  の中を探索することで解を得る非定常反復法\*

⇒ クリロフ部分空間法

# クリロフ部分空間法

## クリロフ部分空間法の種類

- **共役勾配法**(きょうやくこうばいほう, Conjugate Gradient method : **CG法**)
- 双共役勾配法(BiCG法)
- 安定化双共役勾配法(BiCGSTAB法)
- 共役残差法(CR法)
- 一般化共役残差法(GCR法)
- 一般化最小残差法(GMRES法)

などなど

# 共役勾配法

## 共役勾配法(CG法)

- 正定値対称**疎**行列を対象とした非定常反復解法
- [Arnoldi法](#)を対称行列に限定した[Direct版のLanczos法](#)\*を[Projection法](#)で残差ベクトルの[直交・共役関係](#)を用いて線形システムの解法にしたもの (リンク先をたどれば[アルゴリズムの導出](#)過程が分かるが長いので興味と意欲のある場合だけ見てください)
- 収束が非常に早い(ほとんどの場合SOR法より早い)
- 前処理で更に収束を早めることができる

# 共役勾配法の計算手順

1. 初期近似解 $\mathbf{x}^{(0)}$ を設定
2. 初期近似解に対する残差 $\mathbf{r}^{(0)}$ を計算, 修正方向ベクトル $\mathbf{p}^{(0)}$ を初期化:  $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$ ,  $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$
3. 以下のaからgの処理を収束するまで反復( $k = 0, 1, 2, \dots$ )
  - a.  $\mathbf{y}^{(k)} = A\mathbf{p}^{(k)}$ を計算
  - b. 修正係数  $\alpha^{(k)} = \frac{\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)}}{\mathbf{p}^{(k)} \cdot \mathbf{y}^{(k)}}$  を計算
  - c.  $k + 1$ ステップの近似値  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$ を計算
  - d.  $k + 1$ ステップの残差 $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\mathbf{y}^{(k)}$ を計算
  - e.  $\|\mathbf{r}^{(k+1)}\| < \varepsilon$ なら収束したとして反復終了
  - f. 方向ベクトル $\mathbf{p}$ の修正係数 $\beta^{(k)} = \frac{\mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)}}{\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)}}$ を計算
  - g.  $k + 1$ ステップの方向ベクトル $\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k)}\mathbf{p}^{(k)}$ を計算

# 共役勾配法のコード例

二次元配列A[][]に係数行列Aを格納, 配列b[]に右辺項ベクトル**b**を格納  
解 $x^{(k)}$ は配列x[]に入れて返す.

```
vector<double> r(n), p(n), y(n);  
x.assign(n, 0.0);
```

1. 初期値 $x^{(0)}$ の設定  
(すべて0で初期化)

```
// 第0近似解に対する残差の計算  
for(int i = 0; i < n; ++i){  
    double ax = 0.0;  
    for(int j = 0; j < n; ++j){  
        ax += A[i][j]*x[j];  
    }  
    r[i] = b[i]-ax; p[i] = r[i];  
}
```

2. 初期残差と方向ベクトルの計算  
 $r^{(0)} = b - Ax^{(0)}, \quad p^{(0)} = r^{(0)}$

```
double rr0 = dot(r, r, n), rr1;  
double alpha, beta;  
int k;  
for(k = 0; k < max_iter; ++k){  
    :  
}
```

後の計算のために $r^{(0)} \cdot r^{(0)}$ を計算しておく

3. 反復処理  
(for分の中身は次のページ)

# 共役勾配法のコード例

## 3の反復処理 a~g のコード例

```
for(k = 0; k < max_iter; ++k){  
    // y = AP の計算  
    for(int i = 0; i < n; ++i)  
        y[i] = dot(A[i], p, n);  
  
    // alpha = r*r/(P*AP)の計算  
    alpha = rr0/dot(p, y, n);  
  
    // 解x、残差rの更新  
    for(int i = 0; i < n; ++i){  
        x[i] += alpha*p[i]; r[i] -= alpha*y[i];  
    }  
  
    // (r*r)_(k+1)の計算  
    rr1 = dot(r, r, n);  
  
    // 収束判定 (||r|| <= eps)  
    e = sqrt(rr1);  
    if(e < eps) break;  
  
    // βの計算とPの更新  
    beta = rr1/rr0;  
    for(int i = 0; i < n; ++i) p[i] = r[i] + beta*p[i];  
  
    rr0 = rr1; // (r*r)_(k+1)を次のステップのために確保しておく
```

$$a. \mathbf{y}^{(k)} = A\mathbf{p}^{(k)}$$

$$b. \text{修正係数 } \alpha^{(k)} = \frac{\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)}}{\mathbf{p}^{(k)} \cdot \mathbf{y}^{(k)}}$$

c. 次ステップの近似値

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{p}^{(k)}$$

d. 次ステップの残差

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)} \mathbf{y}^{(k)}$$

後のために  $\mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)}$  を計算しておく

$$e. \text{収束判定 } \|\mathbf{r}^{(k+1)}\| < \varepsilon$$

f. 方向ベクトル  $\mathbf{p}$  の修正係数

$$\beta^{(k)} = (\mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)}) / (\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)})$$

g. k+1ステップの方向ベクトル

$$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k)} \mathbf{p}^{(k)}$$



# 共役勾配法の実行結果例

## 共役勾配法(CG法)の実行結果例

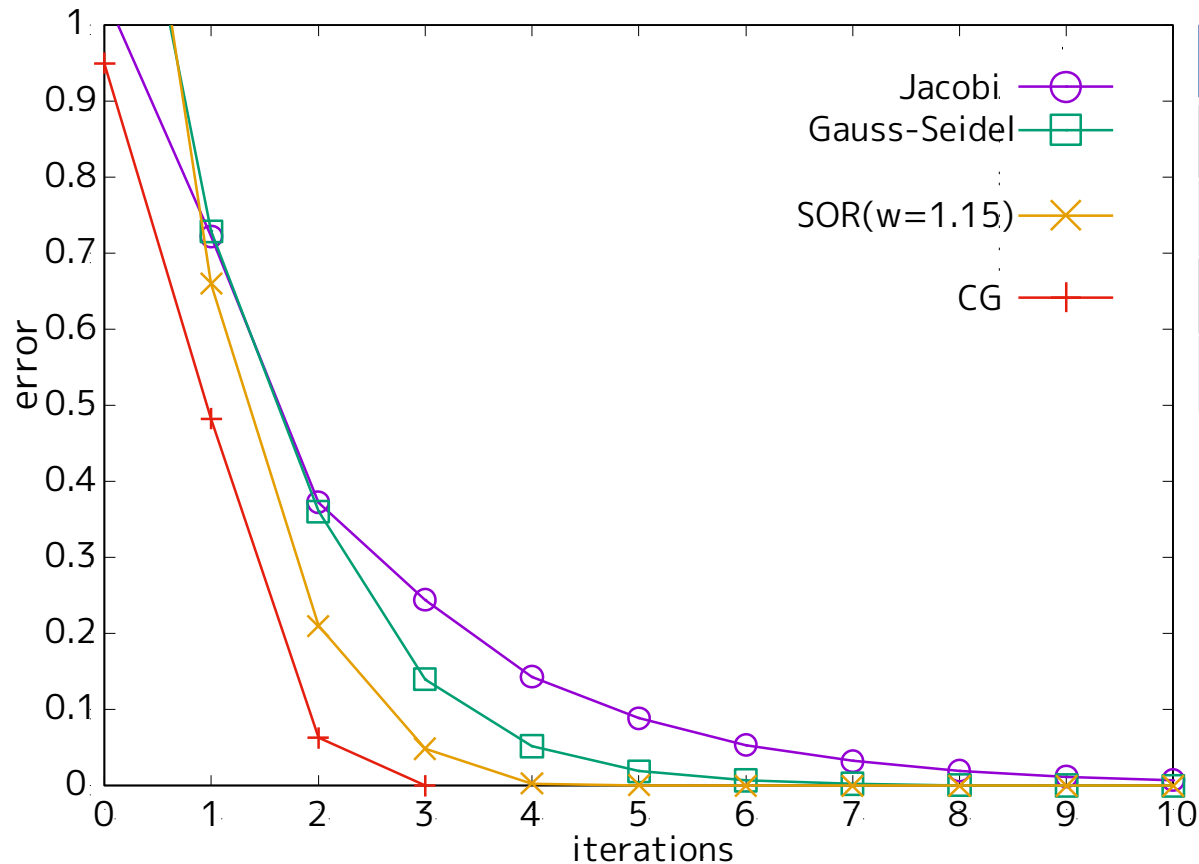
$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 4 \\ 7 \\ 0 \end{pmatrix} \quad \text{解) } (1, 3, 4, 2)$$

```
0 : x0 = -0.44295302, x1 = 1.7718121, x2 = 3.1006711, x3 = 0
1 : x0 = 0.41958577, x1 = 3.1012624, x2 = 3.7012405, x3 = 1.8587355
2 : x0 = 0.88691069, x1 = 2.9264458, x2 = 4.0201937, x3 = 2.082929
3 : x0 = 1, x1 = 3, x2 = 4, x3 = 2
```

反復4回目で許容誤差( $1 \times 10^{-6}$ )内の解に収束

# 共役勾配法の実行結果例

## 反復回数の比較



手法	反復回数
ヤコビ法	30
ガウス・ザイデル	17
<i>SOR</i>	10
共役勾配法	4

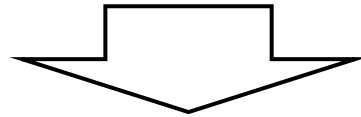
# 今回の講義内容

- 今日の問題
- ヤコビ法とガウス・ザイデル法
- SOR法
- 共役勾配法
- **前処理付き共役勾配法**

# 前処理付き共役勾配法

共役勾配法の収束をさらに早められないか？

⇒ 三角分解のように前処理で何かできるか？



## 前処理付き共役勾配法

- ・修正コレスキー分解付き共役勾配法(MCCG法)
- ・不完全コレスキー分解付き共役勾配法(ICCG法)  
など

# 前処理付き共役勾配法

係数行列 $A$ によって収束が変わる？

⇒ **条件数** で収束の早さを測れる

係数行列 $A$ を**条件数が小さくなる**ようにしてやればよい

ある正則行列\*  $M_1, M_2$  を使って  $Ax = b$  を書き換え

$$(M_1 A M_2)(M_2^{-1} x) = M_1 b$$

(元の式と解は  
変わらないことに注意)



$$(M_1 A M_2) y = M_1 b$$
$$M_2^{-1} x = y$$

(三角分解の時と同じ  
ようにすれば解ける)

\*正則行列は行列式が0でない, つまり逆行列がある行列のこと 53

# 前処理付き共役勾配法

## 条件数とは？

$$\text{条件数} : \text{cond}(A) = \|A^{-1}\| \|A\|$$

- その定義より,  $\text{cond}(A) \geq 1$
- 行列のノルムとして  $p = 2$  の  $p$ -ノルムを用いた場合は,  $\text{cond}(A)$  は  $A$  の **最大, 最小固有値の比の平方根**
- 線形システム  $Ax = b$  における条件数は右辺項ベクトル  $b$  が変化したときに **解  $x$  がどれだけ変化するのか** を表す  
⇒ **条件数大きいと  $b$  の小さな誤差も  $x$  の大きな誤差として現れてしまう**

$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\Delta b\|}{\|b\|} \quad \Rightarrow \quad \text{条件数を小さくしたい}$$

# 前処理付き共役勾配法

## 不完全コレスキー分解付き共役勾配法(ICCG法)

$A = LDL^T$ と分解  $\Rightarrow U = D^{\frac{1}{2}}L^T$ と置き換える\*

$$U^{-T}AU^{-1}U\mathbf{x} = U^{-T}\mathbf{b}$$

(導出過程は  
[こちらを参照](#))



$$(U^{-T}AU^{-1})\mathbf{y} = U^{-T}\mathbf{b}$$

$$U\mathbf{x} = \mathbf{y}$$

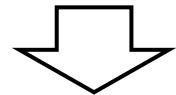
# 前処理付き共役勾配法

$U^{-T}AU^{-1}$ の条件数は？( $U^{-T}AU^{-1}$ を変形してみる)

$$U = D^{1/2}L^T \text{ から } U^T = LD^{1/2}, U^{-1} = L^{-T}(D^{1/2})^{-1}$$

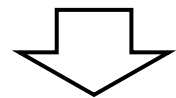


$$U^{-T}AU^{-1} = (D^{1/2})^{-1} \underline{L^{-1}AL^{-T}} (D^{1/2})^{-1}$$



$$A = LDL^T \text{ から } L^{-1}AL^{-T} = D$$

$$U^{-T}AU^{-1} = (D^{1/2})^{-1} \underline{D} (D^{1/2})^{-1}$$



対角行列の逆行列は対角成分の逆数をとったもの

$$U^{-T}AU^{-1} = I$$

単位行列の条件数は1



# ICCG法の計算手順

1. 初期近似解 $\mathbf{x}^{(0)}$ を設定
2. 不完全コレスキー分解で $L, D$ を計算
3. 初期近似解に対する残差 $\mathbf{r}^{(0)}$ を計算, 修正方向ベクトル $\mathbf{p}^{(0)}$ を初期化:  $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$ ,  $\mathbf{p}^{(0)} = (\mathbf{LDL}^T)^{-1}\mathbf{r}^{(0)}$
4. 以下のaからgの処理を収束するまで反復( $k = 0, 1, 2, \dots$ )
  - a.  $\mathbf{y}^{(k)} = A\mathbf{p}^{(k)}$ を計算
  - b. 修正係数  $\alpha^{(k)} = \frac{\mathbf{r}^{(k)} \cdot (\mathbf{LDL}^T)^{-1}\mathbf{r}^{(k)}}{\mathbf{p}^{(k)} \cdot \mathbf{y}^{(k)}}$  を計算
  - c.  $k + 1$ ステップの近似値  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$ を計算
  - d.  $k + 1$ ステップの残差 $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\mathbf{y}^{(k)}$ を計算
  - e.  $\|\mathbf{r}^{(k+1)}\| < \varepsilon$ なら収束したとして反復終了
  - f.  $(\mathbf{LDL}^T)^{-1}$ を残差に掛ける  $\tilde{\mathbf{r}}^{(k+1)} = (\mathbf{LDL}^T)^{-1}\mathbf{r}^{(k+1)}$
  - g. 方向ベクトル $\mathbf{p}$ の修正係数 $\beta^{(k)} = \frac{\mathbf{r}^{(k+1)} \cdot \tilde{\mathbf{r}}^{(k+1)}}{\mathbf{r}^{(k)} \cdot \tilde{\mathbf{r}}^{(k)}}$ を計算
  - h.  $k + 1$ ステップの方向ベクトル $\mathbf{p}^{(k+1)} = \tilde{\mathbf{r}}^{(k+1)} + \beta^{(k)}\mathbf{p}^{(k)}$ を計算

# ICCG法の計算手順

前ページのアルゴリズムで $\mathbf{p}^{(0)} = (\mathbf{LDL}^T)^{-1}\mathbf{r}^{(0)}$ と $\tilde{\mathbf{r}}^{(k+1)} = (\mathbf{LDL}^T)^{-1}\mathbf{r}^{(k+1)}$ は $(\mathbf{LDL}^T)^{-1}$ を直接計算するのではなく,

$$\mathbf{L}\mathbf{y} = \mathbf{r}^{(k+1)}$$

$$(\mathbf{DL}^T)\tilde{\mathbf{r}}^{(k+1)} = \mathbf{y}$$

として, 前進代入と後退代入で解く

ICCGのコード例はCG法のコード例と計算順はほぼ同じなので省略

# ICCG法の実行結果例

## ICCG法の実行結果例

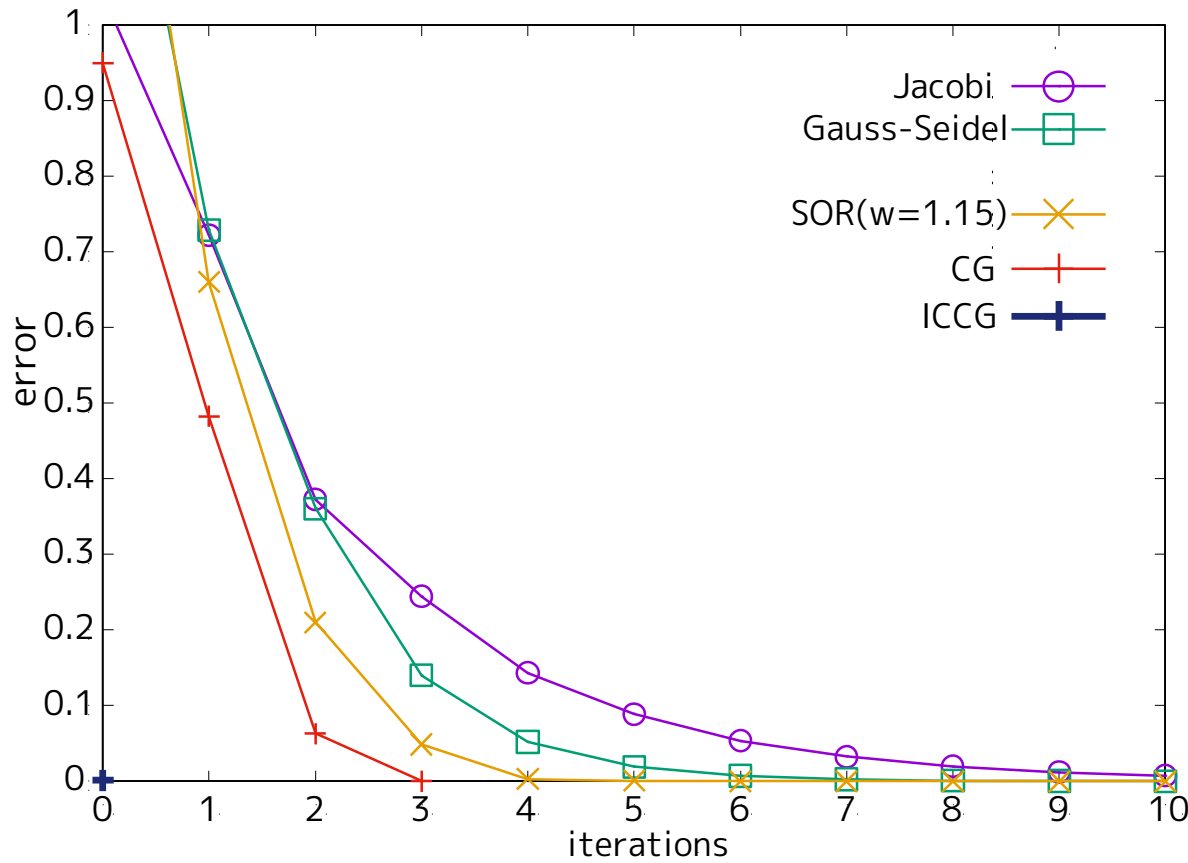
$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 4 \\ 7 \\ 0 \end{pmatrix} \quad \text{解) } (1, 3, 4, 2)$$

$\theta : x_0 = 1, x_1 = 3, x_2 = 4, x_3 = 2$

反復1回目で許容誤差( $1 \times 10^{-6}$ )内の解に収束

# 共役勾配法の実行結果例

## 反復回数の比較



手法	反復回数
ヤコビ法	30
ガウス・ザイデル	17
<i>SOR</i>	10
<i>CG</i>	4
<i>ICCG</i>	1

ICCGでは係数行列が**単位行列**になるので**1回で収束**する。ただし、実際にはIC分解に近似が含まれるため、特に $n$ が大きいと誤差により**1回では収束しなくなる**

# 今回のまとめ

- 今日の問題 :  $Ax = b$
- ヤコビ法とガウス・ザイデル法
  - 定常反復法による解法
- SOR法
  - 加速係数を用いて収束を早める方法
- 共役勾配法
  - クリロフ部分空間法による解法
- 前処理付き共役勾配法
  - 条件数とICCG法

# Appendix

(以降のページは補足資料です)

# 行列のノルム

ベクトルのノルム：空間におけるベクトルの長さ

ユークリッドノルム

$$\|\mathbf{x}\|_2 = \sqrt{|x_1|^2 + \cdots + |x_n|^2}$$

最大値ノルム

$$\|\mathbf{x}\|_\infty = \max(|x_1|, \dots, |x_n|)$$

$p$ -ノルム

$$\|\mathbf{x}\|_p = (|x_1|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

⇒ ベクトルのノルムの性質：正定値性:  $\|\mathbf{x}\| \geq 0$ ,

斉次性(せいじせい):  $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ ,

劣加法性:  $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$

行列のノルム：ベクトルのノルムを行列に対し一般化したもの

⇒ つまり, 正定値性:  $\|A\| \geq 0$ , 斉次性:  $\|\alpha A\| = |\alpha|\|A\|$ ,

劣加法性:  $\|A + B\| \leq \|A\| + \|B\|$  を満たすもの(ベクトルノルムと同じく複数の定義がある)

# 行列のノルム

$p$ -ノルム

$$\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \quad : \text{ベクトルに行列を掛けたときのそのベクトルの拡大率}$$

( $p = 1$ と $p = \infty$ の場合)

$$\|A\|_1 = \max_{0 \leq j < n} \sum_{i=0}^{n-1} |a_{i,j}| \quad : \text{列ごとに絶対値の和をとった数値で最大のもの}$$

$$\|A\|_\infty = \max_{0 \leq i < n} \sum_{j=0}^{n-1} |a_{i,j}| \quad : \text{行ごとに絶対値の和をとった数値で最大のもの}$$

ちなみに、 $p = 2$  (ユークリッドノルム)で正方行列だとスペクトルノルムと呼ばれ、 $A^*A$ の最大固有値の平方根になる( $A^*$ は $A$ の随伴行列(転置+複素共役))

$\rho(A)$ を行列 $A$ のスペクトル半径(最大固有値の絶対値)とすると、行列のノルム(誘導ノルム)に対して

$$\|A\| \geq \rho(A)$$


が成り立つ(証明は省略). そのため、 $\|A\| < 1$ の条件は、最大固有値の絶対値が1より小さい( $\rho(A) < 1$ )と考えることもできる.



# 行列のノルム

行列  $M$  の  $p$  ノルム

$$\|H\|_{\infty} = \|D^{-1}(L + U)\|_{\infty} = \max_{0 \leq i < n} \sum_{j=0, j \neq i}^{n-1} \left| \frac{a_{i,j}}{a_{i,i}} \right| = \max_{0 \leq i < n} \frac{\sum_{j \neq i} |a_{i,j}|}{|a_{i,i}|}$$

  
 $a_{i,i}$  は  $j$  を含まないので  
外に出せる

$\|H\|_{\infty} < 1 \Rightarrow 0 \leq i < n$  で  $\frac{\sum_{j \neq i} |a_{i,j}|}{|a_{i,i}|}$  の最大が 1 を超えないということなので  
 $\frac{\sum_{j \neq i} |a_{i,j}|}{|a_{i,i}|} < 1 \quad (i = 0 \sim n-1)$  が収束条件になる

# 定常反復法と非定常反復法

定常反復法とは？

$Ax = b$ の解を求める漸化式<sub>(ぜんかしき)</sub>が

$$x^{k+1} = Cx^k + d$$

という線形\*の形になっている方法(線形反復法ともいう)

⇒講義で紹介した以外にもADI法, マルチグリッド法なども

非定常反復法とは？

$Ax = b$ の解を求める漸化式が非線形・非定常な反復解法

⇒一般的に定常反復法より収束は速い.

クリロフ部分空間法他にチェビシェフ準反復法なども

$$*\text{ヤコビ法で } x_i^{(k+1)} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j \neq i} a_{i,j} x_j^{(k)} \right)$$