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

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

今回の講義で扱う問題

前回と同じ**線形システム**

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

- 前回とは解き方が異なる ⇒ **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	2.5×10^{-11}秒	2.5×10^{-10}秒
10^2	2.5×10^{-9}秒	2.5×10^{-7}秒
10^3	2.5×10^{-7}秒	2.5×10^{-4}秒
10^4	2.5×10^{-5}秒	0.25秒
10^5	0.0025秒	250秒
10^6	0.25秒	2.5×10^5秒=約69.4時間
10^7	25秒	2.5×10^8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.

*直接解法も丸め誤差/桁落ち誤差が生じるがアルゴリズム(数式)上は誤差なし

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$A\boldsymbol{x} = \boldsymbol{b}$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($a_{11} \sim a_{nn}$⇒$a_{0,0} \sim a_{n-1,n-1}$)
- 右辺項をn列目として表記($b_1 \sim b_n$⇒$a_{0,n} \sim a_{n-1,n}$)

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

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

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

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**$x_i^{(0)}$を与えたとすると...

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

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

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

$x_i^{(0)}$から$x_i^{(1)}$を計算 ⇒ 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$\left| x_i^{(k)} - x_i^{(k+1)} \right| < \varepsilon$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

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

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);
}
```

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，
fabs((y[i]-x[i])/y[i]) > eps;

$x^{(k+1)}$を$x^{(k)}$として次の反復へ

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

$$\begin{pmatrix} 8 & 1 & 3 \\ 1 & 5 & 2 \\ 3 & 1 & 7 \end{pmatrix}\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 9 \\ -2 \end{pmatrix} \xrightarrow[\text{拡大行列表記}]{} \begin{pmatrix} 8 & 1 & 3 & 7 \\ 1 & 5 & 2 & 9 \\ 3 & 1 & 7 & -2 \end{pmatrix}$$

解) $(x_0, x_1, x_2) = (1, 2, -1)$

- 初期値として$\left(x_0^{(0)}, x_1^{(0)}, x_2^{(0)}\right) = (0,0,0)$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

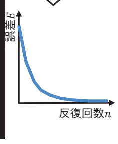
ヤコビ法の実行結果例1

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

解) $(x_0, x_1, x_2) = (1, 2, -1)$

```
 0 : x0 = 0,        x1 = 0,       x2 = 0
 1 : x0 = 0.875,    x1 = 1.8,     x2 = -0.285714
 2 : x0 = 0.757143, x1 = 1.73929, x2 = -0.917857
 3 : x0 = 1.00179,  x1 = 2.01571, x2 = -0.858673
 4 : x0 = 0.945038, x1 = 1.94311, x2 = -1.00301
 5 : x0 = 1.00824,  x1 = 2.0122,  x2 = -0.968318
 6 : x0 = 0.986595, x1 = 1.98568, x2 = -1.00527
 7 : x0 = 1.00377,  x1 = 2.00479, x2 = -0.992209
             ⋮
22 : x0 = 0.999999, x1 = 2,       x2 = -1
23 : x0 = 1,        x1 = 2,       x2 = -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

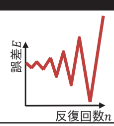
ヤコビ法の実行結果例2

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

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?

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

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

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

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

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

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

⇩ ベクトルと行列を用いた表記

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

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

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

十分な数(K回)反復して収束したとすると:

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

2つの式から以下が導出できる

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

ヤコビ法の収束条件

ヤコビ法で**収束するための条件**は?

$$\underbrace{\boldsymbol{x}^{(K)} - \boldsymbol{x}^{(k+1)}}_{\text{$k+1$ステップ目の誤差}} = H\Big(\underbrace{\boldsymbol{x}^{(K)} - \boldsymbol{x}^{(k)}}_{\text{kステップ目の誤差}}\Big)$$

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:

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

⇩

$$\frac{\sum_{j\neq i}|a_{i,j}|}{|a_{i,i}|} < 1 \qquad (i = 0,1,\dots,n-1)$$

* $\|H\|$は行列のノルム

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はある.
**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行目 : $|a_{i,i}| = 8,\quad \sum_{j\neq i}|a_{i,j}| = 4 \Rightarrow$ ○
2行目 : $|a_{i,i}| = 5,\quad \sum_{j\neq i}|a_{i,j}| = 3 \Rightarrow$ ○
3行目 : $|a_{i,i}| = 7,\quad \sum_{j\neq i}|a_{i,j}| = 4 \Rightarrow$ ○

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行目 : $|a_{i,i}| = 2,\quad \sum_{j\neq i}|a_{i,j}| = 4 \Rightarrow$ ×
2行目 : $|a_{i,i}| = 3,\quad \sum_{j\neq i}|a_{i,j}| = 3 \Rightarrow$ △
3行目 : $|a_{i,i}| = 3,\quad \sum_{j\neq i}|a_{i,j}| = 7 \Rightarrow$ ×

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!

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

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

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

上から順番に計算してるなら, 赤字で示した
$x_0^{(k)}, x_1^{(k)}, x_2^{(k)}$
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?

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

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

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

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

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

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

⇩ 行列表記

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

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

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

```
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
}
```

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,
`fabs(tmp-x[i])/tmp > eps;`

ガウス・ザイデル法

ガウス・ザイデル法の実行結果例

$$\begin{pmatrix} 8 & 1 & 3 \\ 1 & 5 & 2 \\ 3 & 1 & 7 \end{pmatrix}\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 9 \\ -2 \end{pmatrix} \xrightarrow{\text{拡大行列表記}} \begin{pmatrix} 8 & 1 & 3 & 7 \\ 1 & 5 & 2 & 9 \\ 3 & 1 & 7 & -2 \end{pmatrix}$$

解) $(x_0, x_1, x_2) = (1, 2, -1)$

- 初期値として$\left(x_0^{(0)}, x_1^{(0)}, x_2^{(0)}\right) = (0, 0, 0)$を設定
- 収束判定には絶対誤差を使用
  $\left|x_i^{(k+1)} - x_i^{(k)}\right| \le \varepsilon$
- $\varepsilon = 1 \times 10^{-6}$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

ガウス・ザイデル法の実行結果例

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

解) $(x_0, x_1, x_2) = (1, 2, -1)$

```
0 : x0 = 0.875,    x1 = 1.625,   x2 = -0.892857
1 : x0 = 1.0067,   x1 = 1.9558,  x2 = -0.996556
2 : x0 = 1.00423,  x1 = 1.99778, x2 = -1.0015
3 : x0 = 1.00084,  x1 = 2.00043, x2 = -1.00042
4 : x0 = 1.0001,   x1 = 2.00015, x2 = -1.00007
5 : x0 = 1.00001,  x1 = 2.00003, x2 = -1.00001
6 : x0 = 0.999999, x1 = 2,       x2 = -1
7 : x0 = 1,        x1 = 2,       x2 = -1
8 : x0 = 1,        x1 = 2,       x2 = -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)

⇒ ヤコビ法の半分以下の反復で収束

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**ヤ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**

*変数をブロック分けして処理するブロックガウスザイデル法など

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

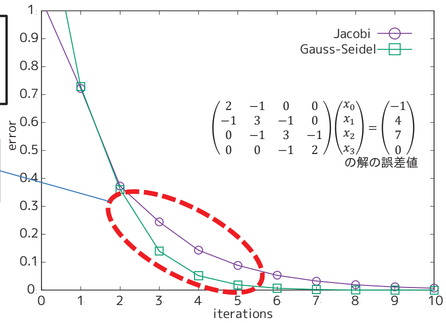
SOR法

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

⇩

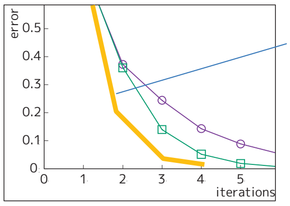
収束時の**傾き**がより
大きいことに注目



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$x^{(k+1)} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=0\sim n-1, j\neq i} a_{i,j}x_j^{(k)}\right)$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ω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なく, 問題毎に試行錯誤することが多い.

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**)

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 4 \\ 7 \\ 0 \end{pmatrix}$$

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
```

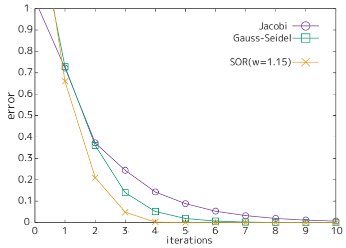
反復10回目で許容誤差(1×10^{-6})内の解に収束

SOR法

ヤコビ法, ガウス・ザイデル法との反復回数の比較

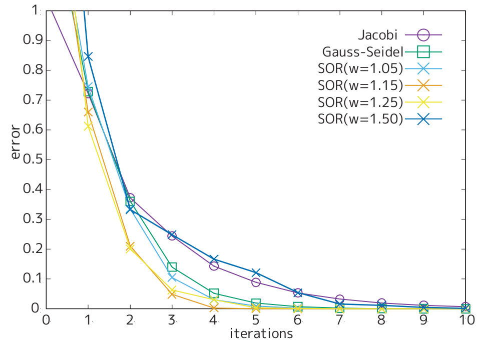
ヤコビ:30回, ガウス・ザイデル:17回, SOR法($\omega = 1.15$):10回

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?



ω	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ω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

$A\boldsymbol{x} = \boldsymbol{b}$ を変形すると $\boldsymbol{b} - A\boldsymbol{x} = 0$

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, $\boldsymbol{b} - A\boldsymbol{x}^{(k)} \neq 0$

残差 $\boldsymbol{r}^{(k)} = \boldsymbol{b} - A\boldsymbol{x}^{(k)}$ (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

$A\boldsymbol{x} = \boldsymbol{b}$ の両辺に$I\boldsymbol{x}$を足す

$A\boldsymbol{x} + I\boldsymbol{x} = \boldsymbol{b} + I\boldsymbol{x}$ ⇨ $\underline{\boldsymbol{x} = \boldsymbol{b} + (I - A)\boldsymbol{x}}$

これを反復式とすると

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{b} + (I - A)\boldsymbol{x}^{(k)} = \left(\boldsymbol{b} - A\boldsymbol{x}^{(k)}\right) + \boldsymbol{x}^{(k)}$$

$$\boxed{\boldsymbol{x}^{(k+1)} = \underbrace{\boldsymbol{r}^{(k)}}_{\text{残差ベクトル}} + \boldsymbol{x}^{(k)}}$$

代入

$k+1$ステップでの残差ベクトルは:

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{b} - A\boldsymbol{x}^{(k+1)} = \boldsymbol{b} - A\left(\boldsymbol{r}^{(k)} + \boldsymbol{x}^{(k)}\right)$$

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

共役勾配法の前に

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

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

$$\boxed{\boldsymbol{x}^{(k+1)} = \boldsymbol{r}^{(k)} + \boldsymbol{r}^{(k-1)} + \cdots + \boldsymbol{r}^{(0)} + \boldsymbol{x}^{(0)}}$$

上の式を下の式に代入してやると,

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

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

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

クリロフ部分空間

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

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

⇩

クリロフ部分空間

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

$\boldsymbol{x}^{(0)}$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**

*前半のヤコビ法, ガウス･ザイデル法は定常反復法

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

*CG法の元になったArnoldi法とかLanczos法もクリロフ部分空間法の一種

共役勾配法の計算手順

1. 初期近似解$\boldsymbol{x}^{(0)}$を設定
2. 初期近似解に対する**残差**$\boldsymbol{r}^{(0)}$を計算, **修正方向ベクトル**$\boldsymbol{p}^{(0)}$を初期化 : $\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{x}^{(0)}, \quad \boldsymbol{p}^{(0)} = \boldsymbol{r}^{(0)}$
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.* $\boldsymbol{y}^{(k)} = A\boldsymbol{p}^{(k)}$を計算
   - *b.* 修正係数 $\alpha^{(k)} = \frac{\boldsymbol{r}^{(k)}\cdot\boldsymbol{r}^{(k)}}{\boldsymbol{p}^{(k)}\cdot\boldsymbol{y}^{(k)}}$ を計算
   - *c.* $k+1$ステップの**近似値** $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{p}^{(k)}$を計算
   - *d.* $k+1$ステップの**残差**$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha^{(k)}\boldsymbol{y}^{(k)}$ を計算
   - *e.* $\|\boldsymbol{r}^{(k+1)}\| < \varepsilon$なら収束したとして反復終了
   - *f.* 方向ベクトル$\boldsymbol{p}$の修正係数$\beta^{(k)} = \frac{\boldsymbol{r}^{(k+1)}\cdot\boldsymbol{r}^{(k+1)}}{\boldsymbol{r}^{(k)}\cdot\boldsymbol{r}^{(k)}}$を計算
   - *g.* $k+1$ステップの**方向ベクトル**$\boldsymbol{p}^{(k+1)} = \boldsymbol{r}^{(k+1)} + \beta^{(k)}\boldsymbol{p}^{(k)}$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$\boldsymbol{b}$を格納
解$\boldsymbol{x}^{(k)}$は配列x[]に入れて返す.

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

// 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];
}

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

1. 初期値$x^{(0)}$の設定 (すべて0で初期化)

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

後の計算のために$\boldsymbol{r}^{(0)}\cdot\boldsymbol{r}^{(0)}$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. $\boldsymbol{y}^{(k)} = A\boldsymbol{p}^{(k)}$

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

c. 次ステップの近似値 $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{p}^{(k)}$

d. 次ステップの残差 $\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha^{(k)}\boldsymbol{y}^{(k)}$

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

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

f. 方向ベクトル$\boldsymbol{p}$の修正係数 $\beta^{(k)} = (\boldsymbol{r}^{(k+1)}\cdot\boldsymbol{r}^{(k+1)})/(\boldsymbol{r}^{(k)}\cdot\boldsymbol{r}^{(k)})$

g. k+1ステップの方向ベクトル $\boldsymbol{p}^{(k+1)} = \boldsymbol{r}^{(k+1)} + \beta^{(k)}\boldsymbol{p}^{(k)}$

共役勾配法の実行結果例

共役勾配法(CG法)の実行結果例

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 4 \\ 7 \\ 0 \end{pmatrix}$$

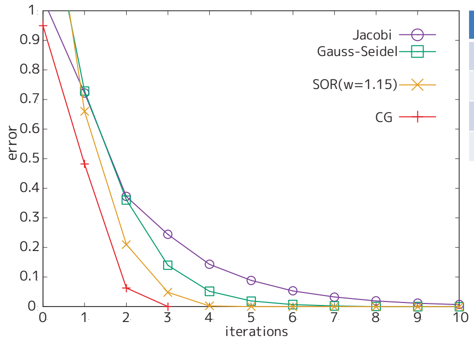
解) $(1, 3, 4, 2)$

```
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(1×10^{-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
SOR	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$A\boldsymbol{x} = \boldsymbol{b}$を書き換え

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

(元の式と解は変わらないことに注意)

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

(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

前処理付き共役勾配法

条件数とは?

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

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

$$\frac{\|\Delta x\|}{\|x\|} \leq \mathrm{cond}(A)\frac{\|\Delta b\|}{\|b\|}$$

⇨ **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*

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

(導出過程は
こちらを参照)

$$\Downarrow$$

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

*Dは対角行列なので$D^{\frac{1}{2}}$は単に対角要素に平方根をつければよい

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)

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

$$\Downarrow$$

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

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

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

$$\Downarrow$$ 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$\boldsymbol{x}^{(0)}$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**$\boldsymbol{r}^{(0)}$を計算, **修正方向ベクトル**$\boldsymbol{p}^{(0)}$を初期化 : $\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{x}^{(0)}$, $\boldsymbol{p}^{(0)} = (LDL^T)^{-1}\boldsymbol{r}^{(0)}$
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. $\boldsymbol{y}^{(k)} = A\boldsymbol{p}^{(k)}$を計算
   - b. 修正係数 $\alpha^{(k)} = \frac{\boldsymbol{r}^{(k)}\cdot\left(LDL^T\right)^{-1}\boldsymbol{r}^{(k)}}{\boldsymbol{p}^{(k)}\cdot\boldsymbol{y}^{(k)}}$ を計算
   - c. $k+1$ステップの**近似値** $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{p}^{(k)}$を計算
   - d. $k+1$ステップの**残差**$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha^{(k)}\boldsymbol{y}^{(k)}$ を計算
   - e. $\left\|\boldsymbol{r}^{(k+1)}\right\| < \varepsilon$なら収束したとして反復終了
   - f. $(LDL^T)^{-1}$を残差に掛ける $\tilde{\boldsymbol{r}}^{(k+1)} = (LDL^T)^{-1}\boldsymbol{r}^{(k+1)}$
   - g. 方向ベクトル$\boldsymbol{p}$の修正係数$\beta^{(k)} = \frac{\boldsymbol{r}^{(k+1)}\cdot\tilde{\boldsymbol{r}}^{(k+1)}}{\boldsymbol{r}^{(k)}\cdot\tilde{\boldsymbol{r}}^{(k)}}$を計算
   - h. $k+1$ステップの**方向ベクトル**$\boldsymbol{p}^{(k+1)} = \tilde{\boldsymbol{r}}^{(k+1)} + \beta^{(k)}\boldsymbol{p}^{(k)}$を計算

ICCG法の計算手順

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

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

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

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 4 \\ 7 \\ 0 \end{pmatrix}$$

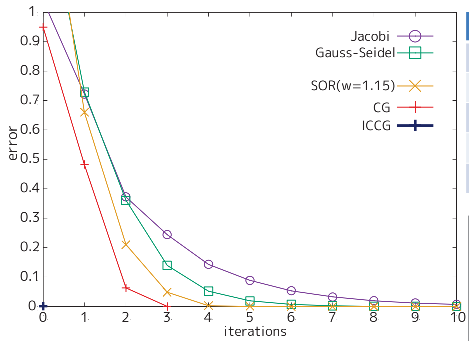
解) $(1,3,4,2)$

```
0 : x0 = 1, x1 = 3, x2 = 4, x3 = 2
```

反復1回目で許容誤差(1×10^{-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
SOR	10
CG	4
$ICCG$	**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：$A\boldsymbol{x} = \boldsymbol{b}$
- 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

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

最大値ノルム

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

p −ノルム

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

⇒ ベクトルのノルムの性質：正定値性: $\|\boldsymbol{x}\| \geq 0$,
斉次性(せいじせい): $\|\alpha\boldsymbol{x}\| = |\alpha|\|\boldsymbol{x}\|$,
劣加法性:$\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$

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

$$\|A\|_p = \max_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\|A\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}$$

: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)

$$\|A\|_1 = \max_{0 \leq j < n} \sum_{i=0}^{n-1} |a_{i,j}|$$

:列ごとに絶対値の和を
とった数値で最大のもの

$$\|A\|_\infty = \max_{0 \leq i < n} \sum_{j=0}^{n-1} |a_{i,j}|$$

: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.

*他にもフロベニウスノルム, 最大ノルム, トレースノルムなどがある

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

$\frac{\sum_{j \neq i} |a_{i,j}|}{|a_{i,i}|} < 1 \ \ (i = 0 \sim n - 1)$ 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？

$A\boldsymbol{x} = \boldsymbol{b}$の解を求める漸化式(ぜんかしき)が

$$\boldsymbol{x}^{k+1} = C\boldsymbol{x}^k + \boldsymbol{d}$$

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？

$A\boldsymbol{x} = \boldsymbol{b}$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の他にチェビシェフ準反復法なども

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