情報数学C

Mathematics for Informatics C

第2回 線形連立方程式の直接解法
(ガウスの消去法,ピボット選択付きガウス消去法,
LU分解,コレスキー分解)

情報メディア創成学類
藤澤誠

授業の進め方

1. その回の講義で対象となる数式(解きたい数式)を提示して説明
2. 解くためのアルゴリズムの説明
3. 実際のコード例を使った説明

コード例はgithubに置いてある

https://github.com/fujis/numerical

授業用資料はWebページに
https://fujis.github.io/numerical/

今回の講義内容

- **今日の問題**
- 数学での解き方とガウス消去法
- ピボット選択
- LU分解とコレスキー分解

今回の講義で扱う問題

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

今回の講義で扱う問題

線形連立方程式(線形システム)

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

⇩ 2元だと…

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix}$$

⇩ 展開すると…

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases}$$

中学数学で見慣れた**連立方程式**

この授業ではベクトルは太字($\boldsymbol{x}, \boldsymbol{b}$など),行列は大文字($A$など)で表す

今回の講義で扱う問題

線形システムは**どんなところで使われる?**

物理	流体力学のナビエ･ストークス方程式や電磁気学のマクスウェル方程式などを解くために
工学	制御工学でシステムの状態を表す状態方程式,出力方程式が線形システムになることが多い $y = Cx + Du$ (x:状態変数,u:入力,y:出力)
情報	データ解析における最小自乗法で将来予測,画像処理でのオプティカルフローの計算,CGでの曲線･曲面フィッティングなど

基本的に多変数の問題の場合,最終的に線形システムを解くことになることが多い

変数表現

n元線形連立方程式の要素の表し方

nが大きくなるとx, y, zでは変数が**足りなくなる**

⇒ $x_1, x_2, x_3, \ldots$ のように**添え字**を使って表現

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & & a_{3n} \\ & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix}$$

- a_{ij}は行列Aのi行目j列目の要素
- $a_{1,4}$のように添え字の間にカンマを入れる場合もあり
- 上記の式で求めたい**未知変数は** x_i

今回の講義内容

- 今日の問題
- **数学での解き方とガウス消去法**
- ピボット選択
- LU分解とコレスキー分解

数学での解き方は?

n元線形連立方程式の数学での解き方

$A\boldsymbol{x} = \boldsymbol{b}$で求めたいのは$\boldsymbol{x}$ (xはベクトルということに注意)

$$\boldsymbol{x} = A^{-1}\boldsymbol{b}$$

逆行列の公式

$$A^{-1} = \frac{1}{|A|}\mathrm{adj}(A)$$

- $|A|$は行列式 ⇒ 逆行列の存在条件$|A| \neq 0$
- $\mathrm{adj}(A)$は余因子行列

$$\mathrm{adj}(A)|_{ij} = (-1)^{i+j}\Delta_{ji}$$

行列式は$\det(A)$と書くことも

Δ_{ji}は行列Aからj行i列を除いた$n-1 \times n-1$の行列の**行列式**

数学での解き方は?

n元線形連立方程式の数学での解き方

$\boldsymbol{x} = A^{-1}\boldsymbol{b}$に前のページの式を代入すると

$$\boldsymbol{x} = \frac{1}{|A|}\begin{pmatrix} \Delta_{11} & -\Delta_{21} & \cdots & (-1)^{n+1}\Delta_{n1} \\ -\Delta_{12} & \Delta_{22} & & (-1)^{n+2}\Delta_{n2} \\ \vdots & & \ddots & \vdots \\ (-1)^{1+n}\Delta_{1n} & (-1)^{2+n}\Delta_{2n} & \cdots & \Delta_{nn} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

なんだかんだと整理すると$\boldsymbol{x}$の各要素x_iは

$$x_i = \frac{1}{|A|}\begin{vmatrix} a_{11} & \cdots & a_{1,i-1} & b_1 & a_{1,i+1} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2,i-1} & b_2 & a_{2,i+1} & \cdots & a_{2n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & \cdots & a_{n,i-1} & b_n & a_{n,i+1} & \cdots & a_{nn} \end{vmatrix}$$

クラメルの公式

行列Aのi列目をベクトル$\boldsymbol{b}$で置き換えた行列の行列式

数学での解き方は?

クラメルの公式で解けるならば**数値計算はいらない**のでは?

⇒ この方法は**計算時間的に現実的ではない**

行列式計算にライプニッツの公式を使ったときの
計算回数 : $(n-1) \times n!$回の乗算と$n!$回の加算

最新のCPU(1.5Tflops程度*)で計算したとして...

n	演算回数	理論的な計算時間
10	3,628,800	約0.0000024秒
20	約2.43×10^{18}	約1.62×10^{6}秒=約19日
30	約2.65×10^{32}	約1.76×10^{20}秒=約5.6兆年

実際の計算には使えない!

*flopsは1秒あたりの浮動小数点演算可能回数, サーバ用のXeon Phi で最高4Tflopsぐらい. GPUならもう一桁上だが実際の問題では$n = 10^6$とかもよくある.

計算量のオーダについて

計算量のだいたいの目安を示すために

$$O(n^2)$$

といった表し方を使う.

- 読み方は**オーダ** (上の例なら"nの2乗のオーダ")
- あくまで絶対的な計算量ではなく**要素数に対する計算量の増減**を表すもの
  例) 計算量 n^2と$3n^2$, $n(n-1)$はどれも $O(n^2)$
- 挿入ソートで$O(n^2)$, クイックソートで$O(n \log n)$, $O(n!)$は巡回セールスマン問題を総当たりで解く場合
  (前ページの計算量は$O(n \cdot n!)$) ⇒ その他計算量については付録参照

数値計算手法のための行列の性質

効率的な計算方法を知るための**行列の性質**の復習

- ある行/ある列を**入れ替えられる**

$$\begin{cases} x+2y=3 \\ 4x+5y=6 \end{cases} \quad \text{式の順番変更} = \text{行の入れ替え} \quad \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

  - 連立方程式で式や変数の**順番を変えても解は変わらない**
  - 行列Aの要素だけでなく右辺項bの要素も入れ替えなければならないことに注意(逆に右辺ベクトルと左辺の列は入れ替えられない)
  - 行の入れ替えなら(x,y)の順番は変わらないけど，列の入れ替えでは(x,y)の**順番が変わる**ので注意

数値計算手法のための行列の性質

効率的な計算方法を知るための**行列の性質**の復習

- 任意の**行だけスカラー倍**できる

$$\begin{cases} x+2y=3 \\ 4x+5y=6 \end{cases} \quad \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

解は同じ

$$\begin{cases} 3\times(x+2y)=3\times 3 \\ 4x+5y=6 \end{cases} \quad \begin{pmatrix} 3 & 6 \\ 4 & 5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 9 \\ 6 \end{pmatrix}$$

方程式の**両辺に同じ定数を掛けても解は変わらない.**
こちらは行だけで列は×

数値計算手法のための行列の性質

効率的な計算方法を知るための**行列の性質**の復習

- スカラー倍した行を**別の行と足し引き**できる

$$\begin{cases} 4\times(x+2y)=3\times 4 \\ 4x+5y=6 \end{cases} \quad \begin{pmatrix} 4 & 8 \\ 4 & 5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 12 \\ 6 \end{pmatrix}$$

上の式を4倍して下の式を引く

$$\begin{cases} 4x+8y=12 \\ 3y=6 \end{cases} \quad \begin{pmatrix} 4 & 8 \\ 0 & 3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 12 \\ 6 \end{pmatrix}$$

加減法と呼ばれる解法で使われる演算
行列としては**非対角成分が**0になっていることが重要

加減法を用いた解き方

加減法で連立方程式を解いてみよう!
数学で**逆行列を使わない**でも連立方程式を解いていたことを思いだそう ⇒ 加減法, 代入法

加減法の手順(2元連立方程式の場合)

1. 片方の式に実数を掛けて係数を合わせて加減算する

例) $\begin{cases} 2x+y=2 \\ x+3y=6 \end{cases}$ ⇨ $\begin{array}{rr} & 2x+y=2 \\ - & 2x+6y=12 \\ \hline & -5y=-10 \\ & y=2 \end{array}$

下の式に**2を掛けて**
上の式から**引く**

2. 結果を式に代入して, もう一つの解を得る

$x+3\times 2=6$ ⇨ $x=0$

ガウスの消去法

加減法を行列で表現した$A\boldsymbol{x}=\boldsymbol{b}$に適用してみよう
(ただし, **ある決まった手順**でできるようにする)

手順1 :片方の式に**実数を掛けて**係数を合わせて**加減算**

$$\begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \end{pmatrix}$$

1行目を**2で割って**, 2行目から**引く**

$$\begin{pmatrix} 2 & 1 \\ 0 & 2.5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$$

⇒ 1行目にa_{21}/a_{11}を掛けて, 2行目から引く
(a_{21}(左下非対角成分)が0になるようにする)

ガウスの消去法

手順2 : 結果を式に**代入**して, もう一つの解を得る

$$\begin{pmatrix} 2 & 1 \\ 0 & 2.5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$$

2行目を**2.5で割る**

$$\begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$:この時点で$x_2=2$

1行目に代入

$x_2=2$を代入すると$2x_1+1\times 2=2$ ⇨ $x_1=\frac{2-1\times 2}{2}=0$

⇒ 2行目をa_{22}で割って, $x_1=\frac{b_1-a_{12}x_2}{a_{11}}$を計算

ガウスの消去法

$n \times n$に拡張してみよう

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($b_1 \sim b_n \Rightarrow a_{0,n} \sim a_{n-1,n}$)

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

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

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

ガウスの消去法

手順1：片方の式に実数を掛けて係数を合わせて加減算する

$\Rightarrow$ 1行目にa_{10}/a_{00}を掛けて，2行目から引く

(a_{10}(**左下非対角成分**)が0になるようにする)

$a_{1,j} - a_{0,j}\dfrac{a_{1,0}}{a_{0,0}}$

$a_{2,j} - a_{0,j}\dfrac{a_{2,0}}{a_{0,0}}$

$\vdots$

$a_{n-1,j} - a_{0,j}\dfrac{a_{n-1,0}}{a_{0,0}}$

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

$(j = 0 \sim n)$

$(i = 1 \sim n-1)$とすると

$$a_{i,j} - a_{0,j}\frac{a_{i,0}}{a_{0,0}}$$

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

残りこの部分も0にしたい

ガウスの消去法

手順1：片方の式に実数を掛けて係数を合わせて加減算する

$a_{2,j} - a_{1,j}\dfrac{a_{2,1}}{a_{1,1}}$

$\vdots$

$a_{n-1,j} - a_{1,j}\dfrac{a_{n-1,1}}{a_{1,1}}$

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

$(j = \mathbf{1} \sim n)$

$(i = \mathbf{2} \sim n-1)$とすると

$$a_{i,j} - a_{\mathbf{1},j}\frac{a_{i,\mathbf{1}}}{a_{\mathbf{1},\mathbf{1}}}$$

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

ここの値を2,3,…と増やして**繰り返し処理**していくと

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & & a_{0,n-1} \\ 0 & a'_{1,1} & a'_{1,2} & \cdots & a'_{1,n-1} \\ 0 & 0 & a''_{2,2} & & a''_{2,n-1} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a^{(n-2)}_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} a_{0,n} \\ a'_{1,n} \\ a''_{2,n} \\ \vdots \\ a^{(n-2)}_{n-1,n} \end{pmatrix}$$

ガウスの消去法

手順1：片方の式に実数を掛けて係数を合わせて加減算する

[1列目]

$$a_{i,j} - a_{\mathbf{0},j}\frac{a_{i,\mathbf{0}}}{a_{\mathbf{0},\mathbf{0}}}$$

$(i = \mathbf{1} \sim n-1)$

$(j = \mathbf{0} \sim n)$

[2列目]

$$a_{i,j} - a_{\mathbf{1},j}\frac{a_{i,\mathbf{1}}}{a_{\mathbf{1},\mathbf{1}}}$$

$(i = \mathbf{2} \sim n-1)$

$(j = \mathbf{1} \sim n)$

…

[$n-1$列目]

$$a_{i,j} - a_{\boldsymbol{n-2},j}\frac{a_{i,\boldsymbol{n-2}}}{a_{\boldsymbol{n-2},\boldsymbol{n-2}}}$$

$(i = \boldsymbol{n-1} \sim n-1)$

$(j = \boldsymbol{n-2} \sim n)$

赤字のところをkとして一つの式にまとめると：

$$a_{i,j} - a_{k,j}\frac{a_{i,k}}{a_{k,k}}$$

$(k = 0 \sim n-2)$

$(i = \boldsymbol{k+1} \sim n-1)$

$(j = \boldsymbol{k} \sim n)$

前進消去

実際の計算では左下部分の計算は必要ない(0になるということは分かっている)ので， $j = k+1 \sim n$ でよい．

ガウスの消去法

手順1:前進消去のコード例

行列のサイズを$n \times n$として，2次元配列A[n][n+1]に
拡大行列の要素が格納されているとする．

```
// 前進消去(forward elimination)
for(int k = 0; k < n-1; ++k){
    for(int i = k+1; i < n; ++i){
        for(int j = k+1; j < n+1; ++j){
            A[i][j] = A[i][j]-A[k][j]*(A[i][k]/A[k][k]);
        }
    }
}
```

$$a'_{i,j} = a_{i,j} - a_{k,j}\frac{a_{i,k}}{a_{k,k}}$$

$j = k$から反復処理すると左下部分が0になるかを確認できるが， $j = k$のときにA[i][k]=A[i][j]として値が更新されてしまい， 誤った解となるので， jのfor文の前にその値をそれぞれ変数に退避して使った方が良い(サンプルプログラム参照)

ガウスの消去法

手順1:前進消去の実行結果例

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 4 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 9 \\ 1 \\ 4 \end{pmatrix} \xRightarrow{\text{拡大行列表記}} \begin{pmatrix} 2 & 1 & 3 & 9 \\ 1 & 3 & 2 & 1 \\ 3 & 4 & 3 & 4 \end{pmatrix}$$

解) $(x_0, x_1, x_2) = (1, -2, 3)$

```
k = 0
2 1    3    9
0 2.5  0.5 -3.5
0 2.5 -1.5 -9.5

k = 1
2 1    3    9
0 2.5  0.5 -3.5
0 0   -2   -6
```

左下部分がすべて
0になっている

ガウスの消去法

手順2 :結果を式に**代入**して, もう一つの解を得る

⇒ 2行目をa_{11}で割って, $x_0 = \frac{b_0 - a_{01}x_1}{a_{00}}$を計算

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & & a_{0,n-1} \\ 0 & a'_{1,1} & a'_{1,2} & \cdots & a'_{1,n-1} \\ 0 & 0 & a''_{2,2} & & a''_{2,n-1} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a^{(n-2)}_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} a_{0,n} \\ a'_{1,n} \\ a''_{2,n} \\ \vdots \\ a^{(n-2)}_{n-1,n} \end{pmatrix} \Uparrow$$

下から上へ計算
($n-1$行から0行へ)

$[n-1行目]$ $a_{n-1,n-1}x_{n-1} = a_{n-1,n}$ $\Rightarrow$ $x_{n-1} = \frac{a_{n-1,n}}{a_{n-1,n-1}}$

$[n-2行目]$ $a_{n-2,n-2}x_{n-2} + a_{n-2,n-1}\underbrace{x_{n-1}}_{既知} = a_{n-2,n}$

注) 下の式ではダッシュは省略している($a'_{i,j} \Rightarrow a_{i,j}$と記述)

$\Rightarrow$ $x_{n-2} = \frac{a_{n-2,n} - a_{n-2,n-1}x_{n-1}}{a_{n-2,n-2}}$

ガウスの消去法

手順2 :結果を式に**代入**して, もう一つの解を得る

$[n-1行目]$ $x_{n-1} = \frac{a_{n-1,n}}{a_{n-1,n-1}}$

$[n-2行目]$ $x_{n-2} = \frac{a_{n-2,n} - a_{n-2,n-1}x_{n-1}}{a_{n-2,n-2}}$

$[n-3行目]$ $x_{n-3} = \frac{a_{n-3,n} - a_{n-3,n-2}x_{n-2} - a_{n-3,n-1}x_{n-1}}{a_{n-3,n-3}}$

$\vdots$

$[0行目]$ $x_0 = \frac{a_{0,n} - a_{0,1}x_1 - a_{0,2}x_2 - \cdots - a_{0,n-1}x_{n-1}}{a_{0,0}}$

$\Downarrow$ i行目(赤字), j列目(青字)として一つの式にまとめると:

$[i行目]$ $x_i = \frac{a_{i,n} - \sum_{j=i+1}^{n-1} a_{i,j}x_j}{a_{i,i}}$ $\quad (i = n-1 \sim 0)$

ガウスの消去法

手順2 :結果を式に**代入**して, もう一つの解を得る

$$x_{n-1} = \frac{a_{n-1,n}}{a_{n-1,n-1}}$$

$$x_i = \frac{a_{i,n} - \sum_{j=i+1}^{n-1} a_{i,j}x_j}{a_{i,i}} \qquad (i = n-2 \sim 0)$$

後退代入

前進消去→後退代入の手順で線形システムを解く方法

⇒ **ガウスの消去法**

ガウスの消去法

手順2:後退代入のコード例

実際のプログラムでは解x_iを$a_{i,n}$に格納していくとすると

$$a_{i,n} = \frac{a_{i,n} - \sum_{j=i+1}^{n-1} a_{i,j}a_{j,n}}{a_{i,i}} \qquad (i = n-2 \sim 0)$$

```
// 後退代入(back substitution)
A[n-1][n] = A[n-1][n]/A[n-1][n-1];
for(int i = n-2; i >= 0; --i){
    double ax = 0.0;
    for(int j = i+1; j < n; ++j){
        ax += A[i][j]*A[j][n]; // Σ部分の計算
    }
    A[i][n] = (A[i][n]-ax)/A[i][i];
}
```


ガウスの消去法

手順2:後退代入の実行結果例

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 4 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 9 \\ 1 \\ 4 \end{pmatrix} \Rightarrow \begin{pmatrix} 2 & 1 & 3 & 9 \\ 0 & 2.5 & 0.5 & -3.5 \\ 0 & 0 & -2 & -6 \end{pmatrix}$$

前進消去後の拡大行列表記

解) $(x_0, x_1, x_2) = (1, -2, 3)$

```
i = 2
2  1    3    9
0  2.5  0.5 -3.5
0  0    1    3

i = 1
2  1  3    9
0  1  0   -2
0  0  1    3
```

```
i = 0
1  0  0    1
0  1  0   -2
0  0  1    3
```

⇧ 解が$a_{i,n}$に格納されている

ガウスの消去法

ガウスの消去法の**演算回数**

前進消去

$(k = 0 \sim n-2)$
×
$(i = k+1 \sim n-1)$
×
$(j = k+1 \sim n)$
の3重ループ
⇩
$\frac{1}{3}O(n^3)$

後退代入

$(i = n-2 \sim 0)$
×
$(j = i+1 \sim n-1)$
の2重ループ
⇩
$\frac{1}{2}O(n^2)$

ほぼ正確な解が得られるが,
計算時間的にはあまり良くない
⇒ 前計算可能ならLU分解
⇒ 近似値でいいなら反復解法(次週)

今回の講義内容

- 今日の問題
- 数学での解き方とガウス消去法
- **ピボット選択**
- LU分解とコレスキー分解

ガウスの消去法の問題

ガウスの消去法にはコンピュータで計算する上で
致命的な問題がある!

前進消去の式をもう一度見てみよう!

$$a_{i,j} - a_{k,j}\frac{a_{i,k}}{\boxed{a_{k,k}}} \quad \begin{array}{l}(k = 0 \sim n-2)\\(i = k+1 \sim n-1)\\(j = k \sim n)\end{array}$$

もし, $a_{k,k} = 0$だったら…

- 対角成分が0だと**「ゼロで割る」(ゼロ割)**が発生!
- ゼロでなくても非対角成分に比べてとても小さいと引き算で**桁落ち**が生じる可能性も...

ピボット選択

行列の性質を上手く使ってこの問題を解決しよう!

- ある行/ある列を**入れ替えられる**

$$\begin{cases} 2y = 3 \\ 4x + 5y = 6 \end{cases}$$

式の順番変更 = 行の入れ替え

$$\begin{pmatrix} 0 & 2 \\ 4 & 5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

このままだと$k = 0$で : $a'_{i,j} = a_{i,j} - a_{0,j}\frac{a_{i,0}}{0}$

ゼロ割!

1行目と2行目を入れ替えてから処理すれば,

$$\begin{pmatrix} 4 & 5 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \end{pmatrix} \qquad a'_{i,j} = a_{i,j} - a_{0,j}\frac{a_{i,0}}{4}$$

問題なし

ピボット選択

行列の性質を上手く使ってこの問題を解決しよう!

対角成分が0もしくはその絶対値が非常に小さい場合,
その行を**絶対値が大きい別の行や列と入れ替える**

⇒ ピボット選択/ピボッティング

- **部分的ピボッティング** : 行の入れ替えのみ行う
⇒未知ベクトル$(x_0, x_1, \dots, x_{n-1})$の入れ替えなし
- **完全ピボッティング** : 行と列の両方で入れ替えを行う
⇒未知ベクトル$(x_0, x_1, \dots, x_{n-1})$の入れ替えあり

ピボット選択

ピボット選択の手順(部分的ピボッティング)

前進消去のk回目の反復において

1. $i = k + 1$からnまでで$|a_{i,k}|$が最大の行pを探索
2. もし, $p \neq k$ならば, p行目とk行目を入れ替え

前進消去の過程で$a_{k,k} = 0$となることもあるので,
「前計算でピボット交換した後に前進消去」ではなく,
前進消去の**毎ステップ**で$k + 1 \sim n$行を調べる必要がある

ピボット選択

ピボット選択付きの前進消去のコード例

```
void Pivoting(vector< vector<double> > &A, int n, int k){
    // k行目以降でk列目の絶対値が最も大きい要素を持つ行を検索
    int p = k; // 絶対値が最大の行
    double am = fabs(A[k][k]);// 最大値
    for(int i = k+1; i < n; ++i){
        if(fabs(A[i][k]) > am){ p = i; am = fabs(A[i][k]); }
    }
    if(k != p) swap(A[k], A[p]); // k != pならば行を交換(ピボット選択)
}

// 前進消去(forward elimination)
for(int k = 0; k < n-1; ++k){
    Pivoting(A, n, k); // ピボット選択を毎kステップで行う
    for(int i = k+1; i < n; ++i){
        for(int j = k+1; j < n+1; ++j){
            A[i][j] = A[i][j]-A[k][j]*(A[i][k]/A[k][k]);
        }
    }
}
```

ピボット選択

ピボット選択しなかった場合の結果例

$$\begin{pmatrix}2 & 4 & 2\\1 & 2 & 3\\4 & 6 & 2\end{pmatrix}\begin{pmatrix}x_0\\x_1\\x_2\end{pmatrix}=\begin{pmatrix}6\\9\\8\end{pmatrix}\ \xRightarrow[\text{拡大行列表記}]{}\ \begin{pmatrix}2 & 4 & 2 & 6\\1 & 2 & 3 & 9\\4 & 6 & 2 & 8\end{pmatrix}$$

解) $(x_0, x_1, x_2) = (2, -1, 3)$

```
k = 0
2  4  2   6
0  0  2   6
0 -2 -2  -4

k = 1
2  4  2   6
0  0  2   6
0 -nan(ind) inf inf

x0 = -nan(ind), x1 = -nan(ind), x2 = -nan(ind)
```

前進消去1回目でa_{11}が0に

nanはnot-a-number
infはinfinityの略

ピボット選択

ピボット選択した場合の結果例

$$\begin{pmatrix}2 & 4 & 2\\1 & 2 & 3\\4 & 6 & 2\end{pmatrix}\begin{pmatrix}x_0\\x_1\\x_2\end{pmatrix}=\begin{pmatrix}6\\9\\8\end{pmatrix}\ \xRightarrow[\text{拡大行列表記}]{}\ \begin{pmatrix}2 & 4 & 2 & 6\\1 & 2 & 3 & 9\\4 & 6 & 2 & 8\end{pmatrix}$$

解) $(x_0, x_1, x_2) = (2, -1, 3)$

```
k = 0
4  6    2    8
0  0.5  2.5  7
0  1    1    2

k = 1
4  6  2   8
0  1  1   2
0  0  2   6

x0 = 2, x1 = -1, x2 = 3
```

1行目と2行目入れ替え後
前進消去

2行目と3行目入れ替え後
前進消去

ガウス・ジョルダン法

ガウスの消去法を使った**逆行列計算**

ガウスの消去法は線形システムを解いている
(解$\boldsymbol{x}$を求めるのが目的)
⇒ 逆行列そのものを求める計算に使えないか？

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

ベクトル$\boldsymbol{x}, \boldsymbol{b}$を行列$X$と単位行列$I$に置き換え

$$AX = I$$
$$A^{-1}AX = A^{-1}I$$
$$IX = A^{-1}$$

Xの係数行列を**単位行列**にすると
右辺が逆行列になる

ガウス・ジョルダン法

拡大行列を使って考えてみる

$$AX = I \ \xRightarrow[\text{拡大行列表記}]{}\ \left(\begin{array}{cccc|cccc} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} & 1 & 0 & \cdots & 0\\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} & 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} & 0 & 0 & \cdots & 1 \end{array}\right)$$

$$IX = A^{-1} \ \xRightarrow[\text{拡大行列表記}]{}\ \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & a'_{0,0} & a'_{0,1} & \cdots & a'_{0,n-1}\\ 0 & 1 & \cdots & 0 & a'_{1,0} & a'_{1,1} & \cdots & a'_{1,n-1}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1 & a'_{n-1,0} & a'_{n-1,1} & \cdots & a'_{n-1,n-1} \end{array}\right)$$

ガウスの消去法を拡張して逆行列を求める：
ガウス・ジョルダン法

ガウス・ジョルダン法

ガウス・ジョルダン法の手順

ガウスの消去法の拡張：

- 基本的には$n \times 2n$**行列**によるガウスの消去法
- i, jの範囲が$k+1$ ~になっていたが, これを**すべての要素に拡張**する
  $(i = 0 \sim n-1, j = 0 \sim 2n-1)$
- **対角要素を1**にするための処理を追加
- **前進消去のみ**

ガウス・ジョルダン法

ガウス・ジョルダン法の手順

$$\left(\begin{array}{cccc|cccc} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} & 1 & 0 & \cdots & 0\\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} & 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} & 0 & 0 & \cdots & 1 \end{array}\right)$$

$$a'_{i,j} = a_{i,j} - a_{k,j}\frac{a_{i,k}}{a_{k,k}} \quad (i \neq k) \quad \begin{array}{l}(k = 0 \sim n-1)\\(i = 0 \sim n-1)\\(j = 0 \sim 2n-1)\end{array}$$

$$a'_{i,j} = \frac{a_{i,j}}{a_{i,i}} \quad (i = k)$$

赤字：ガウスの消去法と違う部分

ガウス・ジョルダン法

ガウス・ジョルダン法の手順

$\boxed{k=0}$

$$\left(\begin{array}{cccc|cc} 1 & a'_{0,1} & \cdots & a'_{0,n-1} & 1/a_{0,0} & \cdots \\ 0 & a'_{1,1} & \cdots & a'_{1,n-1} & -a_{k,n}a_{1,0}/a_{0,0} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & a'_{n-1,1} & \cdots & a'_{n-1,n-1} & & \end{array}\right)$$

$\boxed{k=1}$

$$\left(\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & \\ 0 & 1 & \cdots & a''_{1,n-1} & \text{省略} \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \cdots & a''_{n-1,n-1} & \end{array}\right)$$

i,jが0 ~なので$k=1$の段階で**1行目も含んだ部分が単位行列**になっていることに注意

$\boxed{k=n-1}$

$$\left(\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & \\ 0 & 1 & \cdots & 0 & \text{省略} \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \cdots & 1 & \end{array}\right)$$

対角要素を1にするために最後の行$(n-1)$まで処理する

ガウス・ジョルダン法

ガウス・ジョルダン法のコード例

拡大行列$A(n\times 2n)$は初期化済みとする(左半分が単位行列)

```
// ガウス・ジョルダン法(Gauss-Jordan method)で逆行列計算
for(int k = 0; k < n; ++k){
    // 対角要素を1にするために, k行目のすべての要素をa_kkで割る
    for(int j = 0; j < 2*n; ++j){
        A[k][j] /= A[k][k];
    }

    // k列目の非対角要素を0にする
    for(int i = 0; i < n; ++i){
        if(i == k) continue;
        for(int j = 0; j < 2*n; ++j){
            A[i][j] -= A[k][j]*A[i][k];
        }
    }
}
```

$$a'_{i,j} = \frac{a_{i,j}}{a_{i,i}} \qquad (i = k)$$

$$a'_{i,j} = a_{i,j} - a_{k,j}\frac{a_{i,k}}{a_{k,k}} \qquad (i \neq k)$$

ガウス・ジョルダン法

ガウス・ジョルダン法の実行例

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 4 & 3 \end{pmatrix} \xRightarrow{\text{拡大行列表記}} \begin{pmatrix} 2 & 1 & 3 & 1 & 0 & 0 \\ 1 & 3 & 2 & 0 & 1 & 0 \\ 3 & 4 & 3 & 0 & 0 & 1 \end{pmatrix}$$

```
k = 0
1 0.5  1.5   0.5 0 0
0 2.5  0.5  -0.5 1 0
0 2.5 -1.5  -1.5 0 1

k = 1
1 0  1.4      0.6 -0.2 0
0 1  0.2     -0.2  0.4 0
0 0 -2       -1   -1   1
```

```
k = 2
1 0 0   -0.1 -0.9  0.7
0 1 0   -0.3  0.3  0.1
0 0 1    0.5  0.5 -0.5
```

解) $A^{-1} = \begin{pmatrix} -0.1 & -0.9 & 0.7 \\ -0.3 & 0.3 & 0.1 \\ 0.5 & 0.5 & -0.5 \end{pmatrix}$

解が合っているかは $A^{-1}A = I$となるかで確認できる

今回の講義内容

- 今日の問題
- 数学での解き方とガウス消去法
- ピボット選択
- **LU分解とコレスキー分解**

三角分解

ガウスの消去法は$A\boldsymbol{x}=\boldsymbol{b}$で, Aと$\boldsymbol{b}$**どちらの要素が一つでも変化したら1から再計算**になる

実際の問題に当てはめた場合, Aが**その系(システム)**を表し, $\boldsymbol{b}$が**外部からの影響**を表す

⇒ システムは変わらないのに外部からの影響が変わるたびに再計算するのは**非効率**

Aが変わらず, $\boldsymbol{b}$だけが変わった時に一部の再計算だけで済ませられないか？

三角分解

LU分解

LU分解

三角分解でも最も基本的なもの

LU分解の考え方

$A\boldsymbol{x}=\boldsymbol{b}$の行列$A$が$L$と$U$の**二つの行列に分解**できるとする

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

下三角行列

右上の非対角成分がすべて0の行列 (Lower triangle matrix)

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

上三角行列

対角成分が1で左下の非対角成分がすべて0の行列 (Upper triangle matrix)

LU分解

LU分解の考え方

$$A = LU \quad \Rightarrow \quad LU\boldsymbol{x} = \boldsymbol{b}$$

線形システムを
L, Uで表すと…

新たな変数 $\boldsymbol{y} = U\boldsymbol{x}$
を導入すると…

$$L\boldsymbol{y} = \boldsymbol{b}$$
$$U\boldsymbol{x} = \boldsymbol{y}$$

元の線形システム$A\boldsymbol{x} = \boldsymbol{b}$を**二つの線形システムに分解**できた！

これの何が良いのか？

LU分解

それぞれの**計算回数**を考えてみよう！

$\boxed{L\boldsymbol{y} = \boldsymbol{b}}$

$$\begin{pmatrix} l_{0,0} & 0 & 0 & & 0 \\ l_{1,0} & l_{1,1} & 0 & \cdots & 0 \\ l_{2,0} & l_{2,1} & l_{2,2} & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ l_{n-1,0} & l_{n-1,1} & l_{n-1,2} & \cdots & l_{n-1,n-1} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

[1行目]

$l_{0,0}y_0 = b_0 \Rightarrow y_0 = b_0/l_{0,0}$で簡単に求まる

[2行目]

$l_{1,0}y_0 + l_{1,1}y_1 = b_1 \Rightarrow y_1 = (b_1 - l_{1,0}y_0)/l_{1,1}$

y_0がもう分かっているのでこれも計算は簡単

LU分解

それぞれの**計算回数**を考えてみよう！

$\boxed{L\boldsymbol{y} = \boldsymbol{b}}$

$$\begin{pmatrix} l_{0,0} & 0 & 0 & & 0 \\ l_{1,0} & l_{1,1} & 0 & \cdots & 0 \\ l_{2,0} & l_{2,1} & l_{2,2} & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ l_{n-1,0} & l_{n-1,1} & l_{n-1,2} & \cdots & l_{n-1,n-1} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

$[i+1$行目$(y_i)]$

$$\sum_{j=0}^{i-1} l_{i,j}y_j + l_{i,i}y_i = b_i \quad \Rightarrow \quad y_i = \left(b_i - \sum_{j=0}^{i-1} l_{i,j}y_j\right)/l_{i,i}$$

$i = 0$から$n-1$まで順番に計算すれば$\boldsymbol{y}$を求めることができる．
ガウスの消去法における**「後退代入」と同じ手順**(順番が逆)

前進代入 **計算時間:** $\dfrac{1}{2}O(n^2)$

LU分解

それぞれの**計算回数**を考えてみよう！

$\boxed{U\boldsymbol{x} = \boldsymbol{y}}$

$$\begin{pmatrix} 1 & u_{0,1} & u_{0,2} & & u_{0,n-1} \\ 0 & 1 & u_{1,2} & \cdots & u_{1,n-1} \\ 0 & 0 & 1 & & u_{2,n-1} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$[n$行目$(x_{n-1})]$

$1 \cdot x_{n-1} = y_{n-1} \Rightarrow x_{n-1} = y_{n-1}$

$[n-1$行目$(x_{n-2})]$

$1 \cdot x_{n-2} + u_{n-2,n-1}x_{n-1} = y_{n-1}$

$\Rightarrow x_{n-2} = y_{n-1} - u_{n-2,n-1}x_{n-1}$

LU分解

それぞれの**計算回数**を考えてみよう！

$\boxed{U\boldsymbol{x} = \boldsymbol{y}}$

$$\begin{pmatrix} 1 & u_{0,1} & u_{0,2} & & u_{0,n-1} \\ 0 & 1 & u_{1,2} & \cdots & u_{1,n-1} \\ 0 & 0 & 1 & & u_{2,n-1} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$[i+1$行目$(x_i)]$

$$x_i + \sum_{j=i+1}^{n-1} u_{i,j}x_j = y_i \quad \Rightarrow \quad x_i = y_i - \sum_{j=i+1}^{n-1} u_{i,j}x_j$$

こちらもガウスの消去法における**「後退代入」と同じ手順**

後退代入 **計算時間:** $\dfrac{1}{2}O(n^2)$

LU分解

ガウスの消去法の演算回数

$\boxed{\text{前進消去}}$ $\dfrac{1}{3}O(n^3)$ $\boxed{\text{後退代入}}$ $\dfrac{1}{2}O(n^2)$

LU分解を用いた場合の演算回数

$\boxed{\text{前進代入}}$ $\dfrac{1}{2}O(n^2)$ $\boxed{\text{後退代入}}$ $\dfrac{1}{2}O(n^2)$

全体の計算時間が$O(n^3)$から$O(n^2)$まで**高速化**

(Aが変わらないのなら分解にかかる
時間は前計算として無視できる)

LU分解

$O(n^3)$と$O(n^2)$はどれくらい違う?

1000GflopsのCPUで計算したとして...

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	1×10^{-10}秒	1×10^{-9}秒
10^2	1×10^{-8}秒	1×10^{-6}秒
10^3	1×10^{-6}秒	1×10^{-3}秒
10^4	1×10^{-4}秒	1秒
10^5	0.01秒	1000秒
10^6	1秒	10^6秒=約278時間
10^7	100秒	10^9秒=約31.7年

今の計算機で実際の問題を解く場合はこのあたりの解像度/スケールの問題を扱うことが多い(もちろん分野によって異なるが)

LU分解

LU分解の方法

$A=LU$と下三角行列L, 上三角行列Uの性質を使えば簡単に求められる

$$A=\begin{pmatrix} \boxed{l_{0,0}} & 0 & 0 & & 0 \\ l_{1,0} & l_{1,1} & 0 & \cdots & 0 \\ l_{2,0} & l_{2,1} & l_{2,2} & & 0 \\ & \vdots & & \ddots & \vdots \\ l_{n-1,0} & l_{n-1,1} & l_{n-1,2} & \cdots & l_{n-1,n-1} \end{pmatrix}\begin{pmatrix} 1 & u_{0,1} & u_{0,2} & & u_{0,n-1} \\ 0 & 1 & u_{1,2} & \cdots & u_{1,n-1} \\ 0 & 0 & 1 & & u_{2,n-1} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

⇩ 行列Aの各要素$a_{i,j}$毎の式に展開

[1行目($a_{0,j}$)]

$a_{0,0}=l_{0,0}$ ⇨ $l_{0,0}$が決定

$a_{0,1}=l_{0,0}u_{0,1}$ ⇨ $l_{0,0}$は既知なので$u_{0,1}=a_{0,1}/l_{0,0}$

$a_{0,2}=l_{0,0}u_{0,2}$ ⇨ $l_{0,0}$は既知なので$u_{0,2}=a_{0,2}/l_{0,0}$

⋮

LU分解

LU分解の方法

$$A=\begin{pmatrix} l_{0,0} & 0 & 0 & & 0 \\ \boxed{l_{1,0} \quad l_{1,1}} & & 0 & \cdots & 0 \\ l_{2,0} & l_{2,1} & l_{2,2} & & 0 \\ & \vdots & & \ddots & \vdots \\ l_{n-1,0} & l_{n-1,1} & l_{n-1,2} & \cdots & l_{n-1,n-1} \end{pmatrix}\begin{pmatrix} 1 & u_{0,1} & u_{0,2} & & u_{0,n-1} \\ 0 & 1 & \boxed{u_{1,2} \quad \cdots \quad u_{1,n-1}} & & \\ 0 & 0 & 1 & & u_{2,n-1} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

(赤と青で示したのは計算済みで既知の要素,
枠で示したのはこのステップでの計算対象)

[2行目($a_{1,j}$)]

$a_{1,0}=l_{1,0}$ ⇨ $l_{1,0}$が決定

$a_{1,1}=l_{1,0}u_{0,1}+l_{1,1}$ ⇨ $u_{0,1},l_{1,0}$は既知なので$l_{1,1}=a_{1,1}-l_{1,0}u_{0,1}$

$a_{1,2}=l_{1,0}u_{0,2}+l_{1,1}u_{1,2}$ ⇨ $u_{0,2},l_{1,0},l_{1,1}$は既知なので$u_{1,2}=\frac{a_{1,2}-l_{1,0}u_{0,2}}{l_{1,1}}$

$a_{1,3}=l_{1,0}u_{0,3}+l_{1,1}u_{1,3}$ ⇨ $u_{0,3},l_{1,0},l_{1,1}$は既知なので$u_{1,3}=\frac{a_{1,3}-l_{1,0}u_{0,3}}{l_{1,1}}$

⋮

LU分解

LU分解の方法

$$A=\begin{pmatrix} l_{0,0} & 0 & 0 & & 0 \\ l_{1,0} & l_{1,1} & 0 & \cdots & 0 \\ \boxed{l_{2,0} \quad l_{2,1} \quad l_{2,2}} & & & & 0 \\ & \vdots & & \ddots & \vdots \\ l_{n-1,0} & l_{n-1,1} & l_{n-1,2} & \cdots & l_{n-1,n-1} \end{pmatrix}\begin{pmatrix} 1 & u_{0,1} & u_{0,2} & & u_{0,n-1} \\ 0 & 1 & u_{1,2} & \cdots & u_{1,n-1} \\ 0 & 0 & 1 & \boxed{\quad\quad u_{2,n-1}} & \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

(赤と青で示したのは計算済みで既知の要素,
枠で示したのはこのステップでの計算対象)

[3行目($a_{2,j}$)]

$a_{2,0}=l_{2,0}$ ⇨ $l_{2,0}=a_{2,0}$

$a_{2,1}=l_{2,0}u_{0,1}+l_{2,1}$ ⇨ $l_{2,1}=a_{2,1}-l_{2,0}u_{0,1}$

$a_{2,2}=l_{2,0}u_{0,2}+l_{2,1}u_{1,2}+l_{2,2}$ ⇨ $l_{2,2}=a_{2,2}-l_{2,0}u_{0,2}-l_{2,1}u_{1,2}$

$a_{2,3}=l_{2,0}u_{0,3}+l_{2,1}u_{1,3}+l_{2,2}u_{2,3}$ ⇨ $u_{2,3}=\frac{a_{2,3}-l_{2,0}u_{0,3}-l_{2,1}u_{1,3}}{l_{2,2}}$

$a_{2,4}=l_{2,0}u_{0,4}+l_{2,1}u_{1,4}+l_{2,2}u_{2,4}$ ⇨ $u_{2,4}=\frac{a_{2,4}-l_{2,0}u_{0,4}-l_{2,1}u_{1,4}}{l_{2,2}}$

⋮

このように1行ずつ展開式を評価していけば, L,Uは計算できる!

LU分解

LU分解の計算手順

1. 処理行をiとして, 以下を$i=0\sim n-1$で繰り返し($i=0$が1行目)

   a. $j=0\sim i$で$l_{i,j}$を計算 $(i\geq j)$

   $$l_{i,j}=a_{i,j}-\sum_{k=0}^{j-1}l_{i,k}u_{k,j}$$

   b. $j=i+1\sim n-1$で$u_{i,j}$を計算 $(i<j)$

   $$u_{i,j}=\frac{a_{i,j}-\sum_{k=0}^{j-1}l_{i,k}u_{k,j}}{l_{i,i}}$$

LU分解

LU分解のコード例

```
for(int i = 0; i < n; ++i){
    // l_ijの計算(i >= j)
    for(int j = 0; j <= i; ++j){
        double lu = A[i][j];
        for(int k = 0; k < j; ++k){
            lu -= L[i][k]*U[k][j];
        }
        L[i][j] = lu;
    }

    // u_ijの計算(i < j)
    for(int j = i+1; j < n; ++j){
        double lu = A[i][j];
        for(int k = 0; k < i; ++k){
            lu -= L[i][k]*U[k][j];
        }
        U[i][j] = lu/L[i][i];
    }
}
```

$$l_{i,j}=a_{i,j}-\sum_{k=0}^{j-1}l_{i,k}u_{k,j}$$

$$u_{i,j}=\frac{a_{i,j}-\sum_{k=0}^{j-1}l_{i,k}u_{k,j}}{l_{i,i}}$$

LU分解

LU行列の格納方法

$$L=\begin{pmatrix} l_{0,0} & 0 & 0 & & 0 \\ l_{1,0} & l_{1,1} & 0 & \cdots & 0 \\ l_{2,0} & l_{2,1} & l_{2,2} & & 0 \\ & \vdots & & \ddots & \vdots \\ l_{n-1,0} & l_{n-1,1} & l_{n-1,2} & \cdots & l_{n-1,n-1} \end{pmatrix},\quad U=\begin{pmatrix} 1 & u_{0,1} & u_{0,2} & \cdots & u_{0,n-1} \\ 0 & 1 & u_{1,2} & & u_{1,n-1} \\ 0 & 0 & 1 & & u_{2,n-1} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

0や1の部分まで格納するのは無駄では？

⇩ 一つの行列(一つの2次元配列)にまとめて格納

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

LU分解

LU分解のコード例(1つの行列にまとめたバージョン)

```
for(int i = 0; i < n; ++i){
    // l_ijの計算(i >= j)
    for(int j = 0; j <= i; ++j){
        double lu = A[i][j];
        for(int k = 0; k < j; ++k){
            lu -= A[i][k]*A[k][j];
        }
        A[i][j] = lu;
    }

    // u_ijの計算(i < j)
    for(int j = i+1; j < n; ++j){
        double lu = A[i][j];
        for(int k = 0; k < i; ++k){
            lu -= A[i][k]*A[k][j];
        }
        A[i][j] = lu/A[i][i];
    }
}
```

$$l_{i,j}=a_{i,j}-\sum_{k=0}^{j-1} l_{i,k}u_{k,j}$$

$a_{i,j}$は$l_{i,j}$の計算にしか使われないのでA[i][j]を上書きしても大丈夫

$$u_{i,j}=\frac{a_{i,j}-\sum_{k=0}^{j-1} l_{i,k}u_{k,j}}{l_{i,i}}$$

コレスキー分解

LU分解でAが正定値対称行列*ならば,

$$A=LL^T$$

と分解できる ⇒ **コレスキー分解**

コレスキー分解の式

$$l_{j,j}=\sqrt{a_{j,j}-\sum_{k=0}^{j-1} l_{j,k}^2}\qquad l_{i,j}=\frac{a_{i,j}-\sum_{k=0}^{j-1} l_{i,k}l_{j,k}}{l_{j,j}}\quad (i>j)$$

*正定値対称行列：対称でその固有値がすべて正の行列

コレスキー分解式の導出

下三角行列Lを以下とする.

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

$A=LL^T$の各要素は以下となる.

$$a_{i,j}=\sum_{k=0}^{j} l_{i,k}l_{j,k}$$

Lは左下しか要素がないので, kが$0\sim n-1$でなく$0\sim j$までになっていることに注意

コレスキー分解式の導出

$i=j$の場合と$i>j$の場合で式を分ける

$\boxed{i=j}$

$$a_{j,j}=\sum_{k=0}^{j-1} l_{j,k}^2+l_{j,j}^2$$

⇩

$$l_{j,j}=\sqrt{a_{j,j}-\sum_{k=0}^{j-1} l_{j,k}^2}$$

$\boxed{i>j}$

$$a_{i,j}=\sum_{k=0}^{j-1} l_{i,k}l_{j,k}+l_{i,j}l_{j,j}$$

⇩

$$l_{i,j}=\frac{a_{i,j}-\sum_{k=0}^{j-1} l_{i,k}l_{j,k}}{l_{j,j}}$$

どちらも右辺に未知数を含んでいないか？
⇒ $j=0$から順番に解いていけば問題なし

コレスキー分解の手順

$j=0$から順番に解いていく

$$j=0:\ l_{0,0}=\sqrt{a_{0,0}},\qquad l_{i,0}=\frac{a_{i,0}}{l_{0,0}}\quad (i>j)$$

$$j=1:\ l_{1,1}=\sqrt{a_{1,1}-l_{1,0}^2},\qquad l_{i,1}=\frac{a_{i,1}-l_{i,0}l_{1,0}}{l_{1,1}}\quad (i>j)$$

$$j=2:\ l_{2,2}=\sqrt{a_{2,2}-l_{2,0}^2-l_{2,1}^2},\qquad l_{i,2}=\frac{a_{i,2}-l_{i,0}l_{2,0}-l_{i,1}l_{2,1}}{l_{2,2}}\quad (i>j)$$

$$\vdots\qquad\qquad\vdots$$

コレスキー分解

コレスキー分解のコード例

```
for(int j = 0; j < n; ++j){
    // i == jについて解く
    double ll = A[j][j];
    for(int k = 0; k < j; ++k){
        ll -= L[j][k]*L[j][k];
    }
    L[j][j] = sqrt(ll);

    // i > jについて解く
    for(int i = j+1; i < n; ++i){
        ll = A[i][j];
        for(int k = 0; k < j; ++k){
            ll -= L[i][k]*L[j][k];
        }
        L[i][j] = ll/L[j][j];
    }
}
```

$$l_{j,j} = \sqrt{a_{j,j} - \sum_{k=0}^{j-1} l_{j,k}^2}$$

$$l_{i,j} = \frac{a_{i,j} - \sum_{k=0}^{j-1} l_{i,k} l_{j,k}}{l_{j,j}}$$

コレスキー分解

線形システムを解く手順はLU分解と同じ

$L\boldsymbol{y} = \boldsymbol{b}$　: 前進代入

$L^T\boldsymbol{x} = \boldsymbol{y}$　: 後退代入

LU分解と比べて

- Lを求めるだけなので**計算時間は約半分**
- 対角成分の計算に**平方根**が含まれてしまう
(平方根の中が負になると×, さらに計算量も増える)

⇒ 修正コレスキー分解, 不完全コレスキー分解
($A = LDL^T$と分解)

修正/不完全コレスキー分解の詳細についてはこちらのページ参照

今回の講義のまとめ

- 今日の問題 : $A\boldsymbol{x} = \boldsymbol{b}$
- 数学での解き方とガウス消去法
  - クラメルの公式と計算時間のオーダ
  - 前進消去と後退代入
- ピボット選択
  - ゼロ割を防ぐ方法
  - ガウス･ジョルダン法による逆行列計算
- LU分解とコレスキー分解
  - 三角分解による前計算を使った高速化

Appendix

(以降のページは補足資料です)

ライプニッツの公式による行列式計算

$n \times n$の行列式$\det(A)$はライプニッツの公式から

$$\det(A) = \sum_{\sigma \in S_n} \left(\mathrm{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}\right)$$

式の意味 : 置換*の成分の積に符号を掛けて足し合わせたもの
⇒ $n!$通りの総和 で n個の成分の積が$n-1$回
⇒ $(n-1) \times n!$回の乗算

*置換は簡単に言うと要素を並び替えたもの(厳密な定義ではない)
つまり, $n!$通りの並び替えパターンがあるということ

ランダウ記法と計算量

$O(n)$などの記法はランダウ記法やO-記法などと呼ばれる. 計算機科学の分野ではアルゴリズムの計算時間評価のために使われる.

[代表的な計算オーダとその名称]

$O(1)$:定数時間
$< O(\log(n))$:対数(二分探索など)
$< O(n)$:線形関数
$< O(n\log(n))$:準線形/線形対数時間(ヒープソート,FFTなど)
$< O(n^2)$:二乗時間(挿入ソート,DFTなど)
$< O(n^c)$:多項式時間
---- 越えられない壁 ----
$< O(2^n)$:指数時間(最も速い巡回セールスマン問題の厳密解法など)
$< O(n!)$:階乗関数(巡回セールスマン問題の可能解を全て列挙して調べるなど)
$< O(2^{c^n})$:二重指数時間

(定数時間とかの名称はWikipedia準拠)

$\sqrt{x}$の計算について

$\sqrt{x}$に限らず, $\sin x$や$\cos x$なども**テイラー展開**することで, **四則演算だけで計算**できるようになる.

関数$f(x)$の値aまわりのテイラー展開は:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

> $f^{(n)}$はn回目の反復ということではなく, ここではn階微分を表す
> $f^{(n)}(a) = \frac{d^n f(x)}{dx^n}|_{x=a}$

$\sqrt{x} = x^{\frac{1}{2}}$の1階微分は$\frac{1}{2\sqrt{x}}$なので$a = 0$まわりでテイラー展開するとゼロ割が起こってしまう. そのため, $a = 1$まわりで展開してみる.

$$\sqrt{x} = 1 + \frac{1}{2}(x-1) - \frac{1}{8}(x-1)^2 + \frac{1}{16}(x-1)^3 - \frac{5}{128}(x-1)^4 + \cdots$$

このようにして平方根も反復計算で求められる.
ちなみに一般化二項定理($(1+x)^\alpha = 1 + \alpha x + \frac{\alpha(\alpha-1)}{2!}x^2 + \cdots = \sum_{n=0}^{\infty}\binom{\alpha}{n}x^n$)を使っても同様に,

$$\sqrt{1+x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3 - \frac{5}{128}x^4 + \cdots$$

> $\binom{\alpha}{n}$は二項係数で$n = 0$で1, $n \geq 1$で$\frac{\alpha(\alpha-1)\dots(\alpha-n+1)}{n!}$

他にもニュートン法を用いる方法もあるがこれについては第4回目の講義で扱う予定

x^rのC言語での計算について

x^rのC言語(math.h)による実装(pow関数)では

$$x^r = \exp(\log_e x^r) = \exp(r \log_e x)$$

として実数の累乗を計算する.

$\log x$や$\exp(x)$はテイラー展開(マクローニン展開)により以下のように多項式展開して計算される.

$$\log x = \left(\frac{x-1}{x}\right) + \frac{1}{2}\left(\frac{x-1}{x}\right)^2 + \cdots + \frac{1}{n}\left(\frac{x-1}{x}\right)^n + \cdots$$

$$\exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots$$

C言語の倍精度(double)の桁数(小数点以下15桁)を得るためには$n = 18$程度は必要!

上記の関数や精度は実際のものと違う可能性があるので注意(環境によって違うこともある)