[問題] Ⅳ-B

ニュートン法により $f(x) = 2 - e^x$ の根を求めるプログラムを初期値を $x = 1$ として作成し、根の真値を有効数字10進16桁まで示した $\alpha = 0.6931471805599453$ と比べて絶対誤差が 10^{-8} 以下となる最低の反復回数 n を求めよ。また n 回反復した時の根の近似値と絶対誤差も求めよ。近似値は有効数字10進11桁以降を切り捨てて求め、絶対誤差は有効数字10進4桁以降を切り捨ててよ。

作成したプログラムも提出すること。プログラミング言語は問わない。

[略解] IV-B

プログラムを実行することにより $n = 4$ であることがわかる。

また根の近似値は 0.6931471805

[数値解析 第5回]

連立一次方程式の数値解法

速く解くための下準備

連立一次方程式 (Linear systems)

- n 本の一次方程式が与えられた時に未知数 $x_1, x_2, \ldots, x_n$ の値を求める問題 (**n 元連立一次方程式**)

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

  - $n = 2$ の場合は平面上で 2 本の直線の交点を求める問題
  - $n = 3$ の場合は 3 次元空間中の 3 つの平面の交点を求める問題
  - 一般の n では n 次元空間中の $n - 1$ 個の $n - 1$ 次元面の交点を求める問題

連立一次方程式の行列表示

- 連立一次方程式は**係数行列** A を用いると $A\boldsymbol{x} = \boldsymbol{b}$ と書ける

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \ \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- A が**正則である** (逆行列を持つ) とき解は一意に定まり、A の逆行列を用いて $\boldsymbol{x} = A^{-1}\boldsymbol{b}$ と表せる
  - しかし A^{-1} を求める手法は遅いので使わない (定数倍だけど)

連立一次方程式の数値解法

- **直接法** (解を手堅く確実に求める)
  - 手法　ガウスの消去法、**LU 分解**
  - 用途　$\mathcal{O}(n^3)$ $(A \in \mathbb{R}^{n \times n})$ の計算量に耐えられる問題に用いられる

- **反復法** (解にだんだん近づけていく)
  - 手法　**ガウス-ザイデル法**、SOR 法、共役勾配法
  - 用途　記憶領域を節約できる (安い) ため大規模疎行列に用いられる

[問題] V-A

5×5 の行列 A および 5 次元の縦ベクトル $\boldsymbol{b}$ がそれぞれ以下のように与えられたとする。

$$A = \begin{bmatrix} 14 & 14 & -9 & 3 & -5 \\ 14 & 52 & -15 & 2 & -32 \\ -9 & -15 & 36 & -5 & 16 \\ 3 & 2 & -5 & 47 & 49 \\ -5 & -32 & 16 & 49 & 79 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} -15 \\ -100 \\ 106 \\ 329 \\ 463 \end{bmatrix}.$$

この時行列 A に対して LU 分解および前進代入・後退代入を行うプログラムを作成し連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の解 $\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5]^{\mathsf{T}}$ を有効数字 10 進 3 桁まで求め 4 桁を四捨五入して答えよ。

作成したプログラムも提出すること。プログラミング言語は問わない。

[略解] V-A

$$\boldsymbol{x} = \begin{bmatrix} 3.17 \times 10^{-14} \\ 1.00 \\ 2.00 \\ 3.00 \\ 4.00 \end{bmatrix}$$

1つ目の要素はほぼ 0

[手法] 速く解くための下準備

LU分解 (ピボット選択なし)

LU factorization (no pivoting)

Input: A

Output: L, A [A は実行後 U になる]

1: $L \leftarrow I$
2: **for** $k = 1, 2, \ldots, n-1$ **:**
3: **for** $i = k+1, k+2, \ldots, n$ **:**
4: $m_{ik} \leftarrow a_{ik}/a_{kk}$ [$a_{kk} \neq 0$ を仮定]
5: $a_{ik} \leftarrow 0$
6: **for** $j = k+1, k+2, \ldots, n$ **:**
7: $a_{ij} \leftarrow a_{ij} - m_{ik}a_{kj}$
8: $l_{ik} \leftarrow m_{ik}$

[手法解説]

LU分解 (LU factorization)

正則行列 A が与えられた時、A を下三角行列 L と上三角行列 U を用いて $A = LU$ のように変形する手法。

- **ガウスの消去法**に現れる行の基本変形の操作を行列 L に記録したものと同じ
- $A = LU$ と変形することで元の $A\boldsymbol{x} = \boldsymbol{b}$ という問題を $L\boldsymbol{y} = \boldsymbol{b}$, $U\boldsymbol{x} = \boldsymbol{y}$ という簡単な 2 つの問題に直す
- 乗算の回数は $A \in \mathbb{R}^{n \times n}$ とした時 $\frac{1}{3}n^3 + \mathcal{O}(n^2)$ となる

[手法] 下準備の後の仕上げ

前進代入 ($\ell\ell$. 1–6) • 後退代入 ($\ell\ell$. 7–12)

Input: L, U, $\boldsymbol{b}$
Output: $\boldsymbol{x}$

1: $y_1 \leftarrow b_1$
2: **for** $i = 2, \ldots, n$ **:**
3: $\quad sum \leftarrow 0$
4: $\quad$ **for** $j = 1, 2, \ldots, i-1$ **:**
5: $\quad\quad sum \leftarrow sum + l_{ij} y_j$
6: $\quad y_i \leftarrow b_i - sum$
7: $x_n \leftarrow y_n / u_{ii}$
8: **for** $i = n-1, \ldots, 1$ **:**
9: $\quad sum \leftarrow 0$
10: $\quad$ **for** $j = i+1, i+2, \ldots, n$ **:**
11: $\quad\quad sum \leftarrow sum + u_{ij} x_j$
12: $\quad x_i \leftarrow (y_i - sum) / u_{nn}$

[手法解説]

前進代入・後退代入 (forward substitution/backward substitution)

$L\boldsymbol{y} = \boldsymbol{b}$ から $\boldsymbol{y}$ を求めるのが**前進代入**、求めた $\boldsymbol{y}$ を使って

$U\boldsymbol{x} = \boldsymbol{y}$ から $\boldsymbol{x}$ を求めるのが**後退代入**

- 乗算の回数はどちらも $\frac{1}{2}n^2 + \mathcal{O}(n)$ $(A \in \mathbb{R}^{n \times n})$ となる
  - 前進代入と後退代入で合わせて $n^2 + \mathcal{O}(n)$
  - LU 分解の $\frac{1}{3}n^3 + \mathcal{O}(n^2)$ に比べれば小さい
  - 同じ A で異なる $\boldsymbol{b}$ が与えられた時はLU 分解済みの場合 $n^2 + \mathcal{O}(n)$ で解が求まる

[補足] ピボット選択の導入

- **動機**
  - LU 分解のアルゴリズムで示した $a_{kk} \neq 0$ の仮定を取り除き、さらに丸め誤差の影響をより小さくしたい
- **方法の概要**
  - k (一番外側のループのループ制御変数) の値が κ であるとき、κ 行目にある $a_{\kappa\kappa}, \ldots, a_{n\kappa}$ の成分から絶対値が最大のものを探し、その成分を含む行を p 行目とする
  - A の κ 行目と p 行目の成分をすべて入れ替える (ピボットとする行を κ 行から p 行に入れ替える)

[補足] 乗除算の回数で計算量を比べる

- **LU 分解**
  - $\displaystyle\sum_{k=1}^{n-1}\sum_{i=k+1}^{n}\sum_{j=k+1}^{n} 1 = \frac{1}{3}n^3 + \mathcal{O}(n^2)$
    - ★ 逆行列 A^{-1} を求める場合は $n^3 + \mathcal{O}(n^2)$ かかる
- **前進代入**
  - $\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{i-1} 1 = \frac{1}{2}n^2 + \mathcal{O}(n)$
- **後退代入**
  - $\displaystyle\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 = \frac{1}{2}n^2 + \mathcal{O}(n)$

[補足] 連立一次方程式における残差

- **残差** (residual; 近似値の旨さを測る指標) の連立一次方程式での定義

問題設定　$A\boldsymbol{x} = \boldsymbol{b}$ の近似解 $\tilde{\boldsymbol{x}}$ が得られた場合

定義　$\boldsymbol{r} \equiv \boldsymbol{b} - A\tilde{\boldsymbol{x}}$

考え方　$\boldsymbol{r}$ の長さ $||\boldsymbol{r}||_2 = \sqrt{r_1^2 + \cdots + r_n^2}$ が小さいほど良い近似であると考えられる

使い方 1　$||\boldsymbol{r}||_2$ を計算すれば近似解の相対的な旨さが測れる

使い方 2　**反復改良**に使える

[補足] 反復改良

- 近似解 $\tilde{\boldsymbol{x}}^{(0)}$ から計算できる残差を $\boldsymbol{r}^{(0)}$ $(= \boldsymbol{b} - A\tilde{\boldsymbol{x}}^{(0)})$ とおく
- $A\boldsymbol{y}^{(0)} = \boldsymbol{r}^{(0)}$ を解いて得られる $\boldsymbol{y}^{(0)}$ を用いて新たな近似解 $\tilde{\boldsymbol{x}}^{(1)} = \tilde{\boldsymbol{x}}^{(0)} + \boldsymbol{y}^{(0)}$ とおく
  - A のLU分解が完了している場合、$\frac{1}{2}n^2$ で $\boldsymbol{y}^{(0)}$ が求まる
- $\tilde{\boldsymbol{x}}^{(i+1)} = \tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{y}^{(i)}$ を繰り返す

[道具] 定評のあるライブラリを使おう

- 汎用性が高い数値計算手法の多くはライブラリが存在する
  - **プロ**の手によって実装されている
  - よく**検証**されている
  - 自分で実装するよりも**高速で正確**
  - 自分で実装する場合も既存のライブラリの結果と**整合性が取れているかを確認**すべし

[道具] 線形代数ライブラリ例

- **BLAS** (Basic Linear Algebra Subprograms)
  - 行列とベクトルに対する基本的な線形代数の操作を行うライブラリ
- **LAPACK** (Linear Algebra Package)
  - 連立一次方程式の数値解法、QR分解、特異値分解、固有値問題の数値解法
- FORTRANで実装されておりFORTRANやC/C++, Pythonなどで使用可能

[問題] V-B

5×5 の行列 A および 5 次元の縦ベクトル $\boldsymbol{b}$ がそれぞれ以下のように与えられたとする。

$$A = \begin{bmatrix} 14 & 14 & -9 & 3 & -5 \\ 14 & 52 & -15 & 2 & -32 \\ -9 & -15 & 36 & -5 & 16 \\ 3 & 2 & -5 & 47 & 49 \\ -5 & -32 & 16 & 49 & 79 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} -15 \\ -100 \\ 106 \\ 329 \\ 463 \end{bmatrix}.$$

この時行列 A に対してガウス-ザイデル法を実行するプログラムを作成し連立一次方程式

$A\boldsymbol{x} = \boldsymbol{b}$ の解 $\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5]^{\mathsf{T}}$ を有効数字 10 進 3 桁まで求め 4 桁を四捨五入して答えよ。

作成したプログラムも提出すること。プログラミング言語は問わない。

[手法] 疎行列を扱うための安い方法

ガウス-ザイデル法 (Gauss-Seidel iteration)

Input: A, $k_{\max}$, $\boldsymbol{x}$ (初期ベクトル)
Output: $\boldsymbol{x}$

1: **for** $k = 1, 2, \ldots, k_{\max}$ **:**
2: **for** $i = 1, \ldots, n$ **:**
3: $sum = 0$
4: **for** $j = 1, \ldots, i-1$ **:**
5: $sum \leftarrow sum + a_{ij}x_j$
6: **for** $j = i+1, \ldots, n$ **:**
7: $sum \leftarrow sum + a_{ij}x_j$
8: $x_i = (-sum + b_i)/a_{ii}$

おまけ

※ 配列のインデックスが 0 から始まるプログラミング言語用のアルゴリズム

[手法] 速く解くための下準備

LU 分解 (LU factorization)

※ 配列のインデックスが 0 から始まるプログラミング言語用

Input: A
Output: L, A [A は実行後 U になる]

```
1: L ← I
2: for k = 0, 1, 2, ..., n − 2 :
3:     for i = k + 1, k + 2, ..., n − 1 :
4:         m_ik ← a_ik / a_kk      [a_kk ≠ 0 を仮定]
5:         a_ik ← 0
6:         for j = k + 1, k + 2, ..., n − 1 :
7:             a_ij ← a_ij − m_ik a_kj
8:         l_ik ← m_ik
```

[手法] 下準備の後の仕上げ

前進代入 ($\ell\ell$. 1–6) • 後退代入 ($\ell\ell$. 7–12)

※ 配列のインデックスが 0 から始まるプログラミング言語用

Input: L, U, $\boldsymbol{b}$
Output: $\boldsymbol{x}$

1: $y_0 \leftarrow b_0$
2: **for** $i = 1, \ldots, n-1$ **:**
3: $\quad sum \leftarrow 0$
4: $\quad$ **for** $j = 0, 1, \ldots, i-1$ **:**
5: $\quad\quad sum \leftarrow sum + l_{ij} y_j$
6: $\quad y_i \leftarrow b_i - sum$
7: $x_{n-1} \leftarrow y_{n-1} / u_{n-1,n-1}$
8: **for** $i = n-2, \ldots, 0$ **:**
9: $\quad sum \leftarrow 0$
10: $\quad$ **for** $j = i+1, i+2, \ldots, n-1$ **:**
11: $\quad\quad sum \leftarrow sum + u_{ij} x_j$
12: $\quad x_i \leftarrow (y_i - sum) / u_{ii}$

[手法] 疎行列を扱うための安い方法

ガウス-ザイデル法 (Gauss-Seidel iteration)

※ 配列のインデックスが 0 から始まるプログラミング言語用

Input: A, $k_{\max}$, $\boldsymbol{x}$ (初期ベクトル)
Output: $\boldsymbol{x}$

```
1: for k = 1, 2, ..., k_max :
2:     for i = 0, ..., n − 1 :
3:         sum = 0
4:         for j = 0, 1, ..., i − 1 :
5:             sum ← sum + a_ij x_j
6:         for j = i + 1, i + 2, ..., n − 1 :
7:             sum ← sum + a_ij x_j
8:         x_i = (−sum + b_i)/a_ii
```