数値解析学

ガウス・ザイデル法

提出日：　平成 28 年 10 月 26 日
未来ロボティクス学科　2 年
1526084 中島 崇晴

1. 目的

以下の 3 つの条件を満たすプログラムをガウス・ザイデル法を使用して作成する。
・任意のプログラミング言語（推奨は C 言語）を用いる。
・入力する行列のサイズを可変にする。2×2 から 10×10 程度までを扱える。
あらかじめ指定する事はできる。
・収束・発散の判定部分をつける。

2. 理論

数値線形代数におけるガウス・ザイデル法とはn元の連立一次方程式Ax = bを反復方で解く手法の一つである。

ガウス・ザイデル法の式はベクトル x の名成分ごとに次のような式で書くことができ、数値解析ではこの式が用意られる。

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_j - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \tag{1}$$

ガウス・ザイデル法は、係数行列が正定値対称ならば収束する。
また、係数行列の名行列で非対角要素の絶対値の和が対角要素の絶対値よりも小さい場合

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}| \tag{2}$$

すなわち対角優位な行列ならば収束する。

3. 内容および方法

言語は C 言語を使用した。

まずは、入力部の作成は２次元配列を使い行った。
式(1)を使い計算しているところが以下の図１ある。

```
while(err/absx > EPS){
    err  = 0.0;
    absx = 0.0;

    for(n=1; n<=N; n++){
        sum=0.0;
        for(m=1; m<=N; m++){
            if(n != m){
                sum+=a[n][m]*x[m];
            }
        }
        new_x  = 1.0/a[n][n]*(b[n]-sum);
        err   += fabs(new_x-x[n]);
        absx  += fabs(new_x);
        x[n]   = new_x;
    }
```

図１　式(1)の計算部分

最後に収束判定を行うため式(2)を使用し、係数行列の名行列で非対角要素の絶対値の和が対角要素の絶対値よりも大きい場合を発散と判定をした。（図２）
また、誤差の許容範囲を満たした場合は収束するとした。

```
for(n=1; n<=N; n++){
    sum=0.0;
    for(m=1; m<=N; m++){
        if(n!=m){
            sum += fabs(a[n][m]);
        }
    }
    if(fabs(a[n][n]) < sum){
        printf("発散しました¥n");
        return false;
    }
}
```

図２　式(2)を使った発散判定

念のため、反復計算が 100 回より多くなる場合や x の値が inf または nan になった場合も発散したと判別できるようにした。

```
if(i > 100 || isnan(x[n]) != 0 || isinf(x[n]) != 0){
    printf("inf&nan¥n");
    printf("発散しました¥n");
    return false;
}
```

図３　発散判定

4. 結果

入力する行列のサイズを指定できるようにした。（図４）

理論的には、99×99 までの行列を計算できるように設定しているが、配列の大きさを変えることで更に大きな行列を計算できるようになっている。

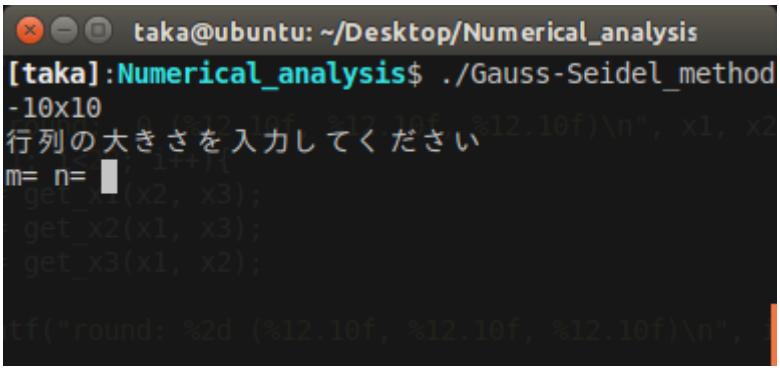

図４　行列のサイズを指定する様子

実際に 10×10 の行列 式(3)を代入したのが図５である。

$$\begin{bmatrix} 10 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 20 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 30 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 40 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 50 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 60 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 70 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 80 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 90 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 100 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 100 \\ 90 \\ 80 \\ 70 \\ 60 \\ 50 \\ 40 \\ 30 \\ 20 \\ 10 \end{bmatrix} \tag{3}$$

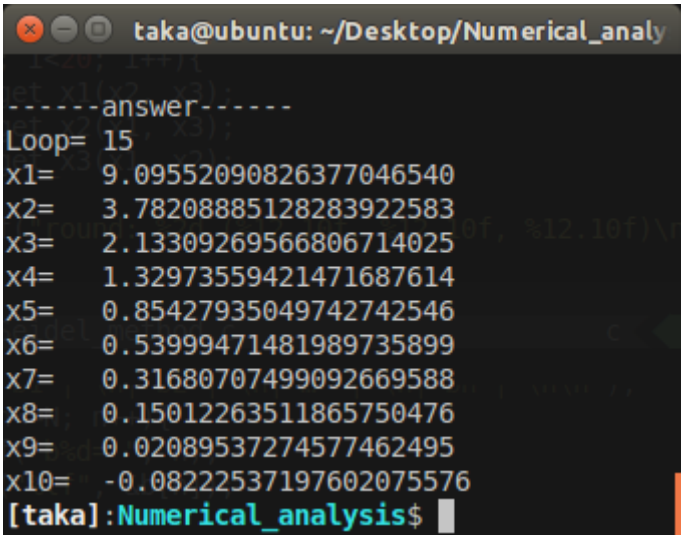

図５　10×10 の行列を入力して出力された結果

収束・発散の判定では、上記に書かれているように発散する式を入力すると発散するようにした。
実際に発散する式(4)を代入した様子が図 6 である。

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} \tag{4}$$

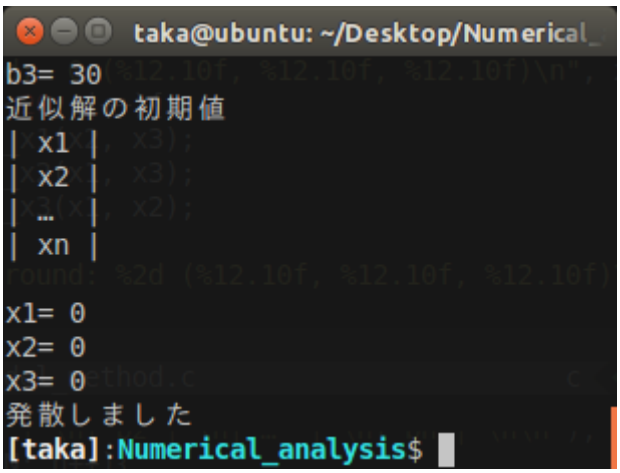

図６　発散判定

5. 考察

はじめ、ガウス・ザイデル法で作成する場合は、x の数分式を書くのでプログラムの行数が大きくなると思われたのですが、式(1)を使うことで数行に済ますことができた。また、今回は C 言語でプログラムを書いたが、C++に変えて行列演算ライブラリ Eigen やクラスを使うなどしてプログラムを整理または機能の追加をするべきだと思われる。

6. 付録　プログラム

Gauss-Seidel_method.c

```c
#include <stdio.h>

#include <math.h>
#define EPS (1e-20)        //計算誤差の許容値

int main(void){
    double a[100][100], b[100], x[100];
    double err=1, absx=1, sum, new_x;
    int i, n, m, N;

    //入力部
    printf("行列の大きさを入力してください¥nm= n= ");
    scanf("%d", &N);

    printf("係数行列を入力してください¥n");
    printf("| a11 a12 … a1m | ¥n| a21 a22 … a2m | ¥n"
          "| …             | ¥n| an1 an2 … anm | ¥n¥n");
    for(n=1; n<=N; n++){
        for(m=1; m<=N; m++){
            printf("a%d%d= ", n, m);
            scanf("%lf", &a[n][m]);
        }
    }

    printf("同次項を入力してください¥n");
    printf("| b1 | ¥n| b2 | ¥n| …  | ¥n| bn | ¥n¥n");
    for(n=1; n<=N; n++){
        printf("b%d= ", n);
```

```
    scanf("%lf", &b[n]);
}

printf("近似解の初期値¥n");
printf("| x1 | ¥n| x2 | ¥n| …  | ¥n| xn | ¥n¥n");
for(n=1; n<=N; n++){
    printf("x%d= ", n);
    scanf("%lf", &x[n]);
}

//収束判定
for(n=1; n<=N; n++){
    sum=0.0;
    for(m=1; m<=N; m++){
        if(n!=m){
            sum += fabs(a[n][m]);
        }
    }
    if(fabs(a[n][n]) < sum){
        printf("発散しました¥n");
        return false;
    }
}

//反復計算のループ
i = 0;
while(err/absx > EPS){
    err  = 0.0;
    absx = 0.0;

    for(n=1; n<=N; n++){
        sum=0.0;
        for(m=1; m<=N; m++){
            if(n != m){
                sum+=a[n][m]*x[m];
            }
```

```
            }

            new_x  = 1.0/a[n][n]*(b[n]-sum);
            err   += fabs(new_x-x[n]);
            absx  += fabs(new_x);
            x[n]   = new_x;
        }

        //収束判定
        if(i > 100 || isnan(x[n]) != 0 || isinf(x[n]) != 0){
            printf("inf&nan¥n");
            printf("発散しました¥n");
            return false;
        }
        i++;
    }

    printf("¥n------answer------¥n");
    printf("Loop= %d¥n", i);

    for(n=1; n<=N; n++){
        printf("x%d=%25.20f¥n", n, x[n]);
    }

    return 0;
}
```