シミュレーション

4 年電子情報工学科
34 番 横前洸佑

提出日：2020/2/17（月）
提出期限：2020/2/17（月）10:00

1　課題 11

課題 11 では、ガウスの消去法を用いて連立方程式 (1) を解き、x_1, x_2, x_3, x_4 を求める。

$$\begin{aligned} x_1 + 2x_2 + x_3 + 5x_4 &= 20.5 \\ 8x_1 + x_2 + 3x_3 + x_4 &= 14.5 \\ x_1 + 7x_2 + x_3 + x_4 &= 18.5 \\ x_1 + x_2 + 6x_3 + x_4 &= 9.0 \end{aligned} \tag{1}$$

1.1　作成したプログラム

今回作成したプログラムをソースコード 1 に示す。

ソースコード 1　課題 11 のプログラム

```
#include<stdio.h>
#include<math.h>

void gauss(void);
void print_eq(void);

double result[4];
double eq[4][5] = {
    {1, 2, 1, 5, 20.5},
    {8, 1, 3, 1, 14.5},
    {1, 7, 1, 1, 18.5},
    {1, 1, 6, 1, 9.0}
};

int main(void)
{
    gauss();
    printf("\n結果\n");
    for(int i = 0; i < 4; i++)
        printf("x[%d] = %f\n", i + 1, result[i]);

    return (0);
}

void gauss(void)
{
    //前進削除
    for(int k = 0; k < 3; k++)
    {
        print_eq();
        printf("\n第%d列の掃き出し\n", k + 1);
        for(int i = k + 1; i < 4; i++)
        {
            double m = eq[i][k] / eq[k][k];
            for(int j = k; j < 5; j++)
            {
                eq[i][j] -= (eq[k][j] * m);
            }
        }
    }

    print_eq();

    //後退代入
    for(int i = 0; i < 4; i++)
    {
        result[i] = eq[i][4];
```

```
48      }
49
50      for(int i = 3; i >= 0; i--)
51      {
52          for(int j = i + 1; j <= 3; j++)
53          {
54              result[i] -= eq[i][j] * result[j];
55          }
56
57          result[i] /= eq[i][i];
58      }
59  }
60
61  //二次元配列の内容を表示
62  void print_eq(void)
63  {
64       for(int i = 0; i < 4; i++)
65      {
66          for(int j = 0; j < 5; j++)
67          {
68              printf("%f ", eq[i][j]);
69          }
70          printf("\n");
71      }
72  }
```

ガウスの消去法の計算部分を関数として用意し、main 関数内から呼び出している。

1.2 プログラムの実行結果

実行結果を以下に示す。

```
1.000000 2.000000 1.000000 5.000000 20.500000
8.000000 1.000000 3.000000 1.000000 14.500000
1.000000 7.000000 1.000000 1.000000 18.500000
1.000000 1.000000 6.000000 1.000000 9.000000

第 1 列の掃き出し
1.000000 2.000000 1.000000 5.000000 20.500000
0.000000 -15.000000 -5.000000 -39.000000 -149.500000
0.000000 5.000000 0.000000 -4.000000 -2.000000
0.000000 -1.000000 5.000000 -4.000000 -11.500000

第 2 列の掃き出し
1.000000 2.000000 1.000000 5.000000 20.500000
0.000000 -15.000000 -5.000000 -39.000000 -149.500000
0.000000 0.000000 -1.666667 -17.000000 -51.833333
0.000000 0.000000 5.333333 -1.400000 -1.533333
```

```
第 3 列の掃き出し
1.000000 2.000000 1.000000 5.000000 20.500000
0.000000 -15.000000 -5.000000 -39.000000 -149.500000
0.000000 0.000000 -1.666667 -17.000000 -51.833333
0.000000 0.000000 0.000000 -55.800000 -167.400000

結果
x[1] = 1.000000
x[2] = 2.000000
x[3] = 0.500000
x[4] = 3.000000
```

1.3 考察

元の方程式に計算によって得た値を代入する。

$$\begin{aligned} &1.0 + 2 \times 2.0 + 0.5 + 5 \times 3.0 = 20.5 \\ &8 \times 1.0 + 2.0 + 3 \times 0.5 + 3.0 = 14.5 \\ &1.0 + 7 \times 2.0 + 0.5 + 3.0 = 18.5 \\ &1.0 + 2.0 + 6 \times 0.5 + 3.0 = 9.0 \end{aligned} \tag{2}$$

式がすべて成り立つ。この結果よりプログラムは正しく動作していると言える。

2 課題 12

課題 12 では、ガウスの消去法を用いて連立方程式 (3) を解き、x_1, x_2, x_3, x_4 を求める。また、ピボット選択なしで、float 型で変数を宣 â¿Ť した場合と double 型で変数を宣 â¿Ť した場合について解いた時、ピボット選択ありで、float 型で変数を宣 â¿Ť した場合と double 型で変数を宣 â¿Ť した場合について解いた時の結果の違いを考察する。

$$\begin{aligned} &1.0x_1 + 0.96x_2 + 0.84x_3 + 0.64x_4 = 3.44 \\ &0.96x_1 + 0.9214x_2 + 0.4406x_3 + 0.2222x_4 = 2.5442 \\ &0.84x_1 + 0.4406x_2 + 1.0x_3 + 0.3444x_4 = 2.6250 \\ &0.64x_1 + 0.2222x_2 + 0.3444x_3 + 1.0x_4 = 2.2066 \end{aligned} \tag{3}$$

2.1 作成したプログラム

今回作成したプログラムをソースコード 2 に示す。

ソースコード 2　課題 12 のプログラム

```
#include<stdio.h>
#include<math.h>

void gauss(void);
void print_eq(void);
void pivoting(int);

double result[4];
double eq[4][5] = {
    {1.0, 0.96, 0.84, 0.64, 3.44},
    {0.96, 0.9214, 0.4406, 0.2222, 2.5442},
    {0.84, 0.4406, 1.0, 0.3444, 2.6250},
    {0.64, 0.2222, 0.3444, 1.0, 2.2066}
    };

int main(void)
{
    gauss();
    printf("\n結果\n");
    for(int i = 0; i < 4; i++)
        printf("x[%d] = %f\n", i + 1, result[i]);

    return (0);
}


void gauss(void)
{
    //前進削除
    for(int k = 0; k < 3; k++)
    {
        print_eq();
        printf("\n第%d列の掃き出し\n", k + 1);
        for(int i = k + 1; i < 4; i++)
        {
            pivoting(k);
            double m = eq[i][k] / eq[k][k];
            for(int j = k; j < 5; j++)
            {
                eq[i][j] -= (eq[k][j] * m);
            }
        }
    }

    print_eq();

    //後退代入
    for(int i = 0; i < 4; i++)
    {
        result[i] = eq[i][4];
    }

    for(int i = 3; i >= 0; i--)
    {
        for(int j = i + 1; j <= 3; j++)
        {
            result[i] -= eq[i][j] * result[j];
        }

        result[i] /= eq[i][i];
    }
}


//二次元配列の内容を表示
void print_eq(void)
{
     for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 5; j++)
        {
```

```
72              printf("%f ", eq[i][j]);
73          }
74          printf("\n");
75      }
76  }
77
78  //ピボット選択
79  void pivoting(int k)
80  {
81      if(eq[k][k] == 0.0)
82      {
83          printf("a_kkが0のためピボット選択を実行します。\n");
84          //最大値検索
85          int max_i;
86          for(int i = k; i <= (3 - k); i++)
87          {
88              double tmp = eq[i][k];
89              if(tmp < eq[i + 1][k])
90              {
91                  tmp = eq[i + 1][k];
92                  max_i = i + 1;
93              }
94          }
95
96          //列交換
97          for(int i = 0; i < 5; i++)
98          {
99              double tmp = eq[k][i];
100             eq[k][i] = eq[max_i][i];
101             eq[max_i][i] = tmp;
102         }
103     }
104 }
```

ガウスの消去法の計算部分を関数として用意し、main 関数内から呼び出している。また、gauss() 関数内 36 行目でピボット選択を行う。

2.2 プログラムの実行結果

ピボット選択なしで、float 型で実行した結果を以下に示す。

```
1.000000 0.960000 0.840000 0.640000 3.440000
0.960000 0.921400 0.440600 0.222200 2.544200
0.840000 0.440600 1.000000 0.344400 2.625000
0.640000 0.222200 0.344400 1.000000 2.206600

第 1 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 -0.365800 0.294400 -0.193200 -0.264600
0.000000 -0.392200 -0.193200 0.590400 0.005000
```

```
第 2 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.430725 717.235229 1386.666016
0.000000 0.000000 717.235229 769.796082 1487.031372

第 3 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.430725 717.235229 1386.666016
0.000000 0.000000 0.000000 1.342590 1.342651

結果
x[1] = 1.000012
x[2] = 1.000000
x[3] = 0.999951
x[4] = 1.000045
```

ピボット選択なしで、double 型で実行した結果を以下に示す。

```
1.000000 0.960000 0.840000 0.640000 3.440000
0.960000 0.921400 0.440600 0.222200 2.544200
0.840000 0.440600 1.000000 0.344400 2.625000
0.640000 0.222200 0.344400 1.000000 2.206600

第 1 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 -0.365800 0.294400 -0.193200 -0.264600
0.000000 -0.392200 -0.193200 0.590400 0.005000

第 2 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
```

```
0.000000 0.000000 669.342600 717.140600 1386.483200
0.000000 0.000000 717.140600 769.694600 1486.835200

第 3 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.342600 717.140600 1386.483200
0.000000 0.000000 0.000000 1.342727 1.342727

結果
x[1] = 1.000000
x[2] = 1.000000
x[3] = 1.000000
x[4] = 1.000000
```

ピボット選択ありで、float 型で実行した結果を以下に示す。

```
1.000000 0.960000 0.840000 0.640000 3.440000
0.960000 0.921400 0.440600 0.222200 2.544200
0.840000 0.440600 1.000000 0.344400 2.625000
0.640000 0.222200 0.344400 1.000000 2.206600

第 1 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 -0.365800 0.294400 -0.193200 -0.264600
0.000000 -0.392200 -0.193200 0.590400 0.005000

第 2 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.430725 717.235229 1386.666016
0.000000 0.000000 717.235229 769.796082 1487.031372
```

```
第 3 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.430725 717.235229 1386.666016
0.000000 0.000000 0.000000 1.342590 1.342651

結果
x[1] = 1.000012
x[2] = 1.000000
x[3] = 0.999951
x[4] = 1.000045
```

ピボット選択ありで、double 型で実行した結果を以下に示す。

```
1.000000 0.960000 0.840000 0.640000 3.440000
0.960000 0.921400 0.440600 0.222200 2.544200
0.840000 0.440600 1.000000 0.344400 2.625000
0.640000 0.222200 0.344400 1.000000 2.206600

第 1 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 -0.365800 0.294400 -0.193200 -0.264600
0.000000 -0.392200 -0.193200 0.590400 0.005000

第 2 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.342600 717.140600 1386.483200
0.000000 0.000000 717.140600 769.694600 1486.835200

第 3 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
```

```
0.000000 0.000000 669.342600 717.140600 1386.483200
0.000000 0.000000 0.000000 1.342727 1.342727

結果
x[1] = 1.000000
x[2] = 1.000000
x[3] = 1.000000
x[4] = 1.000000
```

2.3 考察

元の方程式に計算によって得た値を代入する。

$$\begin{aligned} 1.0 + 0.96 + 0.84 + 0.64 &= 3.44 \\ 0.96 + 0.9214 + 0.4406 + 0.2222 &= 2.5442 \\ 0.84 + 0.4406 + 1.0 + 0.3444 &= 2.6250 \\ 0.64 + 0.2222 + 0.3444 + 1.0 &= 2.2066 \end{aligned} \tag{4}$$

式がすべて成り立つ。この結果よりプログラムは正しく動作していると言える。

3 課題 13

表 1 に示す 7 組のデータに対して 2 次式で近似を行うプログラムを作成する。

表 1　データ

i	1	2	3	4	5	6	7
x_i	0.0	0.1	0.2	0.3	0.4	0.5	0.6
y_i	0.000	0.034	0.138	0.282	0.479	0.724	1.120

3.1 作成したプログラム

今回作成したプログラムをソースコード 3 に示す。

ソースコード 3　課題 13 のプログラム

```
1  #include<stdio.h>
2  #include<math.h>
3
4  void print_eq(void);
5  void pivoting(int);
6
```

```
7  #define DATA_NUM 7
8
9  double data[3][DATA_NUM] = {
10     {1, 2, 3, 4, 5, 6, 7},   //i
11     {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6},   //x_i
12     {0.000, 0.034, 0.138, 0.282, 0.479, 0.724, 1.120}    //y_i
13 };
14
15 double eq[3][4] = {0.0};
16
17 int main(void)
18 {
19     eq[0][0] = DATA_NUM;
20     for(int i = 0; i < DATA_NUM; i++)
21     {
22         eq[0][1] += data[1][i];
23         eq[0][2] += pow(data[1][i], 2.0);
24         eq[0][3] += data[2][i];
25         eq[1][0] += data[1][i];
26         eq[1][1] += pow(data[1][i], 2.0);
27         eq[1][2] += pow(data[1][i], 3.0);
28         eq[1][3] += data[1][i] * data[2][i];
29         eq[2][0] += pow(data[1][i], 2.0);
30         eq[2][1] += pow(data[1][i], 3.0);
31         eq[2][2] += pow(data[1][i], 4.0);
32         eq[2][3] += pow(data[1][i], 2.0) * data[2][i];
33     }
34
35     //前進削除
36     for(int k = 0; k < 2; k++)
37     {
38         print_eq();
39         printf("\n第%d列の掃き出し\n", k + 1);
40
41         for(int i = k + 1; i < 3; i++)
42         {
43             pivoting(k);
44             double m = eq[i][k] / eq[k][k];
45             for(int j = k; j < 4; j++)
46             {
47                 eq[i][j] -= (eq[k][j] * m);
48             }
49         }
50     }
51
52     print_eq();
53
54     //後退代入
55
56     double result[3] = {eq[0][3], eq[1][3], eq[2][3]};
57
58     for(int i = 2; i >= 0; i--)
59     {
60         for(int j = i + 1; j <= 2; j++)
61         {
62             result[i] -= eq[i][j] * result[j];
63         }
64
65         result[i] /= eq[i][i];
66     }
67
68     printf("\n結果\n");
69     for(int i = 0; i < 3; i++)
70         printf("result[%d] = %f\n", i, result[i]);
71
72     return (0);
73 }
74
75
76 //二次元配列の内容を表示
77 void print_eq(void)
78 {
79      for(int i = 0; i < 3; i++)
```

```
80      {
81          for(int j = 0; j < 4; j++)
82          {
83              printf("%f ", eq[i][j]);
84          }
85          printf("\n");
86      }
87  }
88
89  //ピボット選択
90  void pivoting(int k)
91  {
92      if(eq[k][k] == 0.0)
93      {
94          printf("a_kkが0のためピボット選択を実行します。\n");
95          //最大値検索
96          int max_i;
97          for(int i = k; i <= (2 - k); i++)
98          {
99              double tmp = eq[i][k];
100             if(tmp < eq[i + 1][k])
101             {
102                 tmp = eq[i + 1][k];
103                 max_i = i + 1;
104             }
105         }
106
107         //列交換
108         for(int i = 0; i < 4; i++)
109         {
110             double tmp = eq[k][i];
111             eq[k][i] = eq[max_i][i];
112             eq[max_i][i] = tmp;
113         }
114     }
115 }
```

3.2 プログラムの実行結果

プログラムの実行結果を以下に示す。

```
7.000000 2.100000 0.910000 2.777000
2.100000 0.910000 0.441000 1.341200
0.910000 0.441000 0.227500 0.692080

第 1 列の掃き出し
7.000000 2.100000 0.910000 2.777000
0.000000 0.280000 0.168000 0.508100
0.000000 0.168000 0.109200 0.331070

第 2 列の掃き出し
7.000000 2.100000 0.910000 2.777000
0.000000 0.280000 0.168000 0.508100
```

```
0.000000 0.000000 0.008400 0.026210

結果
result[0] = 0.008333
result[1] = -0.057500
result[2] = 3.120238
```

4 課題 14