シミュレーション工学　ヤコビ法，ガウス・ザイデル法

三浦夢生

2020.10.14 Wed

1　反復法

与えられた方程式に対して，ガウスの消去法や LU 分解といった有限回の手順を踏んで代数的に厳密に解く直接法に対して，ヤコビ法やガウス・ザイデル法などは反復法と呼ばれ，与えられた初期値からある収束する解を近似的に求める方法である．反復法は直接法に比べてメモリの使用量，計算量が少なく済む反面，終了条件の影響が大きい，収束しないことがあるといった短所もある．

2　ヤコビ法

与えられた連立一次方程式

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{1}$$

に対して解を

$${x_i}^{(k*1)} = \frac{1}{a_{ii}}(b_i - a_{i1}{x_1}^{(k)} - a_{i2}{x_2}^{(k)} - \cdots - a_{ii-1}{x_{i-1}}^{(k)} - a_{ii+1}{x_{i+1}}^{(k)} - \cdots - a_{in}{x_n}^{(k)}) \tag{2}$$

とし，初期値 ${x_i}^0$ と終了条件を与えて一つずつ解を計算する．終了条件はしばしば $|{x_i}^{(k+1)} - {x_i}^{(k)}| < \epsilon$ とされる．ここで ϵ は精度を決める定数である．

3　ガウス・ザイデル法

概ねヤコビ法と同様であるが，解の求め方に違いがある．ヤコビ法では解を一つずつ求めていたが，ガウス・ザイデル法では以下のようにする．右辺の変数を常に最新の値を用いて計算する．

$$\begin{aligned} {x_1}^{(k+1)} &= \frac{1}{a_{11}}(b_1 - a_{12}{x_2}^{(k)} - a_{13}{x_3}^{(k)} - a_{14}{x_4}^{(k)} - \cdots - a_{1n}{x_n}^{(k)}) \\ {x_2}^{(k+1)} &= \frac{1}{a_{22}}(b_2 - a_{21}{x_1}^{(k+1)} - a_{23}{x_3}^{(k)} - a_{24}{x_4}^{(k)} - \cdots - a_{2n}{x_n}^{(k)}) \\ {x_3}^{(k+1)} &= \frac{1}{a_{33}}(b_3 - a_{31}{x_1}^{(k+1)} - a_{32}{x_2}^{(k+1)} - a_{34}{x_4}^{(k)} - \cdots - a_{3n}{x_n}^{(k)}) \\ &\vdots \end{aligned} \tag{3}$$

$${x_n}^{(k+1)} = \frac{1}{a_{nn}}(b_n - a_{n1}{x_1}^{(k+1)} - a_{n2}{x_2}^{(k+1)} - a_{n3}{x_3}^{(k+1)} - \cdots - a_{nn-1}{x_{n-1}}^{(k+1)}) \tag{4}$$

4　手計算

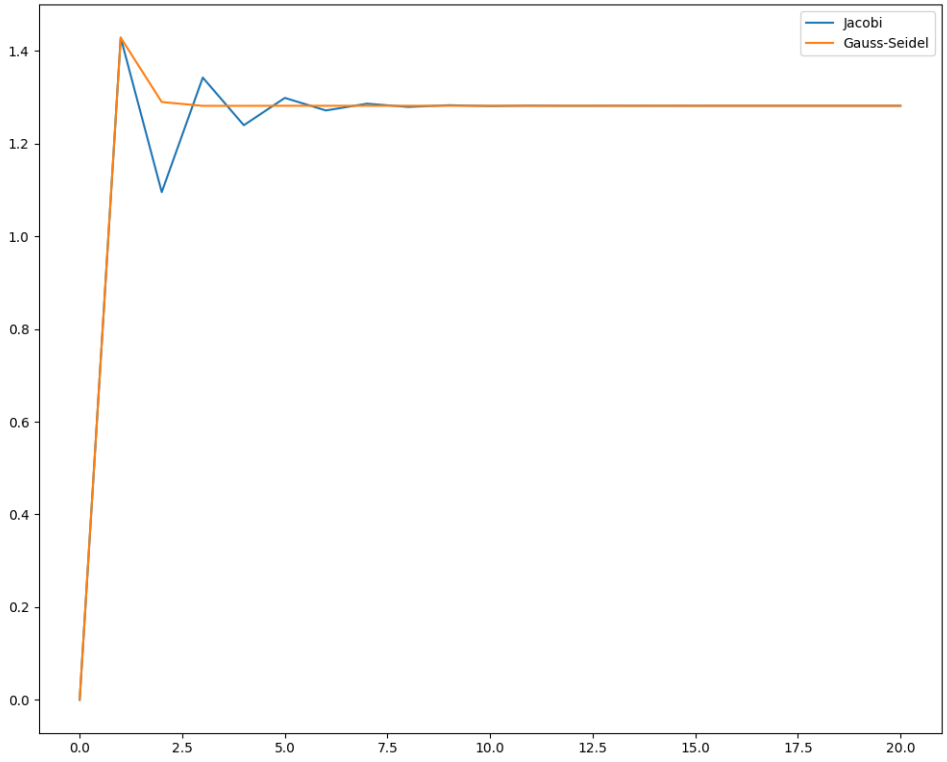
$k = 3$ まで手計算を行った画像を付録 6.5 に示す．

今回手計算の労力を加味し，小数第 5 位までで計算を行った．

アルゴリズムの理解の助けになり，実装する際にも役立った．

5　実行結果

実行回数をともに 20 回に設定し，x 成分の変化を表したグラフを以下に示す．



この場合どちらも試行回数 10 回目までには収束しており，またヤコビ法よりもガウス・ザイデル法のほうが収束が早いことがわかる．

グラフからもわかるが，試行回数が少ないと収束しないが多すぎても無駄な計算になってしまうことを確認した．

またヤコビ法，ガウス・ザイデル法で終了条件を満たすまで計算を続けた結果をそれぞれ以下に示す．ただし，終了条件は前後の差の絶対値が $\epsilon = 10^{-7}$ となるまでとした．

```
% python3 JacobiMethod.py
x=1.4285714285714286, y=1.0, z=0.6666666666666666 / count=1
x=1.0952380952380953, y=0.5714285714285714, z=0.015873015873015817 / count=2
x=1.342403628117914, y=0.8571428571428572, z=0.23280423280423276 / count=3
x=1.2396069538926682, y=0.7448979591836735, z=0.0826404635928445 / count=4
x=1.2985458733758055, y=0.8140589569160998, z=0.14289913496262704 / count=5
```

```
x=1.271448967594092, y=0.7840945902170392, z=0.10674793138889882 / count=6
x=1.2860585067150232, y=0.8010384047799014, z=0.1227575882400776 / count=7
x=1.2790637741057063, y=0.793208591070593, z=0.11386308580336103 / count=8
x=1.282723605331812, y=0.7974183710605263, z=0.11802740873075655 / count=9
x=1.2809324016397086, y=0.7953992710594898, z=0.11581085290608861 / count=10
x=1.2818541461611905, y=0.7964543799552533, z=0.11688193150467925 / count=11
x=1.2813973938621983, y=0.7959375074155964, z=0.11632539642353991 / count=12
x=1.2816302428196178, y=0.7962033021083977, z=0.1165991877809793 / count=13
x=1.2815140460470917, y=0.7960715242296805, z=0.11645884533728573 / count=14
x=1.2815729692993927, y=0.7961386772426313, z=0.11652859280186394 / count=15
x=1.2815434481648058, y=0.796105156536877, z=0.11649311440814675 / count=16
x=1.2815583735209757, y=0.7961221510763442, z=0.11651084822886196 / count=17
x=1.2815508789237045, y=0.7961136352240548, z=0.1165018666365574 / count=18
x=1.2815546616432614, y=0.7961179401458279, z=0.11650637072004738 / count=19
x=1.2815527597734397, y=0.7961157782745746, z=0.11650409514177706 / count=20
x=1.2815537187774102, y=0.7961168693501537, z=0.11650523840326629 / count=21
x=1.2815532362633306, y=0.7961163207515989, z=0.11650466159941313 / count=22
x=1.2815534794356533, y=0.7961165973673038, z=0.1165049516909491 / count=23
x=1.2815533570358284, y=0.7961164581864374, z=0.1165048054474202 / count=24
x=1.2815534187026747, y=0.7961165283277389, z=0.11650487904100339 / count=25
calc 25 times.
```

```
% python3 GaussSeidelMethod.py
x=1.4285714285714286, y=0.8214285714285714, z=0.07539682539682542 / count=1
x=1.2896825396825398, y=0.810515873015873, z=0.10989858906525571 / count=2
x=1.2813838498362309, y=0.7986150478710002, z=0.11570968407939307 / count=3
x=1.281423654852888, y=0.7964309116136166, z=0.11642888393926379 / count=4
x=1.2815301886439794, y=0.7961478949422787, z=0.11649954865391167 / count=5
x=1.281550429678557, y=0.7961188655449635, z=0.1165047271119995 / count=6
x=1.2815530971758626, y=0.7961165901860173, z=0.11650489278780254 / count=7
x=1.2815533748911967, y=0.7961164933431745, z=0.11650486335423141 / count=8
x=1.2815533971354804, y=0.7961165016002282, z=0.116504855658706 / count=9
calc 9 times.
```

このように，ヤコビ法よりもガウス・ザイデル法のほうが収束が早く，計算する回数が少なくて済むことがわかる．終了条件を 10^{-9} や 10^{-12} などに変えてみても，ガウス・ザイデル法のほうが早く収束することを確認した．

6 付録

今回行った手計算の画像及び作成したソースコードを以下に示す．

6.1 ヤコビ法　プログラム 1

```
import numpy as np
```

```
2
3 def Jacobi_method(x0, y0, z0, A, b):
4     EPSILON = 1.0e-7
5     x_t = x0
6     y_t = y0
7     z_t = z0
8     cnt = 0
9
10    x = 0
11    y = 0
12    z = 0
13
14    X = [x]
15    Y = [y]
16    Z = [z]
17
18    '''
19    for i in range(20):
20        x = (b[0] - (A[0][1] * y_t + A[0][2] * z_t)) / A[0][0]
21        y = (b[1] - (A[1][0] * x_t + A[1][2] * z_t)) / A[1][1]
22        z = (b[2] - (A[2][0] * x_t + A[2][1] * y_t)) / A[2][2]
23        X.append(x)
24        Y.append(y)
25        Z.append(z)
26        x_t = x
27        y_t = y
28        z_t = z
29    '''
30
31    while True:
32        cnt += 1
33        x = (b[0] - (A[0][1] * y_t + A[0][2] * z_t)) / A[0][0]
34        y = (b[1] - (A[1][0] * x_t + A[1][2] * z_t)) / A[1][1]
35        z = (b[2] - (A[2][0] * x_t + A[2][1] * y_t)) / A[2][2]
36        X.append(x)
37        Y.append(y)
38        Z.append(z)
39        print(f'x={x}, y={y}, z={z} / count={cnt}')
40        if np.abs(x - x_t) < EPSILON and np.abs(y - y_t) < EPSILON and np.abs(z - z_t)
              < EPSILON:
41            break
42        x_t = x
43        y_t = y
44        z_t = z
45    print(f'calc {cnt} times.')
46
```

```
47      return X, Y, Z, cnt
48
49  if __name__ == '__main__':
50      A = np.array([[7, 1, 2],
51                    [1, 8, 3],
52                    [2, 3, 9]], dtype=float)
53      b = np.array([10, 8, 6], dtype=float)
54
55      x0 = 0
56      y0 = 0
57      z0 = 0
58
59      X, Y, Z, cnt = Jacobi_method(x0, y0, z0, A, b)
```

6.2 ヤコビ法　プログラム 2

```
1   import numpy as np
2
3   def Jacobi_method(A, b, x_init):
4       EPSILON = 1.0e-7
5       x_t = x_init
6       size = len(x_t)
7       flag = 0
8       cnt = 0
9
10      x = np.zeros(size, dtype=float)
11
12      '''
13      for time in range(10):
14          for i in range(size):
15              s = 0
16              for j in range(size):
17                  if i != j:
18                      s += A[i][j] * x_t[j]
19              x[i] = (b[i] - s) / A[i][i]
20          for i in range(size):
21              x_t[i] = x[i]
22      '''
23
24      while True:
25          cnt += 1
26          #
27          # calculation
28          #
29          for i in range(size):
```

```
            s = 0
            for j in range(size):
                if i != j:
                    s += A[i][j] * x_t[j]
            x[i] = (b[i] - s) / A[i][i]
        #
        # Exit conditions
        #
        flag = 0
        for i in range(size):
            if np.abs(x[i] - x_t[i]) < EPSILON:
                flag += 1
        if flag == size:
            break
        #
        # update value
        #
        for i in range(size):
            x_t[i] = x[i]
    print(f'calc {cnt} times.')

    return x

if __name__ == '__main__':
    A = np.array([[7, 1, 2],
                  [1, 8, 3],
                  [2, 3, 9]], dtype=float)
    b = np.array([10, 8, 6], dtype=float)
    x_init = np.array([0, 0, 0], dtype=float)

    x = Jacobi_method(A, b, x_init)

    for i in range(len(x)):
        print(f'x[{i}] = {x[i]}')
```

6.3 ガウス・ザイデル法　プログラム 1

```
import numpy as np

def GaussSeidel_method(x0, y0, z0, A, b):
    EPSILON = 1.0e-7
    x_t = x0
    y_t = y0
    z_t = z0
    cnt = 0
```

```
9
10      x = 0
11      y = 0
12      z = 0
13
14      X = [x]
15      Y = [y]
16      Z = [z]
17
18      '''
19      for i in range(20):
20          x = (b[0] - (A[0][1] * y + A[0][2] * z)) / A[0][0]
21          y = (b[1] - (A[1][0] * x + A[1][2] * z)) / A[1][1]
22          z = (b[2] - (A[2][0] * x + A[2][1] * y)) / A[2][2]
23          X.append(x)
24          Y.append(y)
25          Z.append(z)
26          x_t = x
27          y_t = y
28          z_t = z
29      '''
30
31      while True:
32          cnt += 1
33          x = (b[0] - (A[0][1] * y + A[0][2] * z)) / A[0][0]
34          y = (b[1] - (A[1][0] * x + A[1][2] * z)) / A[1][1]
35          z = (b[2] - (A[2][0] * x + A[2][1] * y)) / A[2][2]
36          X.append(x)
37          Y.append(y)
38          Z.append(z)
39          print(f'x={x}, y={y}, z={z} / count={cnt}')
40          if np.abs(x - x_t) < EPSILON and np.abs(y - y_t) < EPSILON and np.abs(z - z_t)
                < EPSILON:
41              break
42          x_t = x
43          y_t = y
44          z_t = z
45      print(f'calc {cnt} times.')
46
47      return X, Y, Z, cnt
48
49  if __name__ == '__main__':
50      A = np.array([[7, 1, 2],
51                    [1, 8, 3],
52                    [2, 3, 9]], dtype=float)
53      b = np.array([10, 8, 6], dtype=float)
```

```
54
55      x0 = 0
56      y0 = 0
57      z0 = 0
58
59      X, Y, Z, cnt = GaussSeidel_method(x0, y0, z0, A, b)
```

6.4 ガウス・ザイデル法　プログラム 2

```
1  import numpy as np
2
3  def GaussSeidel_method(A, b, x_init):
4      EPSILON = 1.0e-7
5      x_t = x_init
6      size = len(x_t)
7      flag = 0
8      cnt = 0
9
10     x = np.zeros(size, dtype=float)
11
12     '''
13     for time in range(10):
14         for i in range(size):
15             s = 0
16             for j in range(size):
17                 if i != j:
18                     s += A[i][j] * x[j]
19             x[i] = (b[i] - s) / A[i][i]
20         for i in range(size):
21             x_t[i] = x[i]
22     '''
23
24     while True:
25         cnt += 1
26         #
27         # calculation
28         #
29         for i in range(size):
30             s = 0
31             for j in range(size):
32                 if i != j:
33                     s += A[i][j] * x[j]
34             x[i] = (b[i] - s) / A[i][i]
35         #
36         # Exit conditions
```

```
37          #
38          flag = 0
39          for i in range(size):
40              if np.abs(x[i] - x_t[i]) < EPSILON:
41                  flag += 1
42          if flag == size:
43              break
44          #
45          # update value
46          #
47          for i in range(size):
48              x_t[i] = x[i]
49      print(f'calc {cnt} times.')
50
51      return x
52
53  if __name__ == '__main__':
54      A = np.array([[7, 1, 2],
55                    [1, 8, 3],
56                    [2, 3, 9]], dtype=float)
57      b = np.array([10, 8, 6], dtype=float)
58      x_init = np.array([0, 0, 0], dtype=float)
59
60      x = GaussSeidel_method(A, b, x_init)
61
62      for i in range(len(x)):
63          print(f'x[{i}] = {x[i]}')
```

6.5 手計算

ヤコビ法　ちょっと手計算してみよう

$$\begin{cases} 7x + y + 2z = 10 & \cdots ① \\ x + 8y + 3z = 8 & \cdots ② \\ 2x + 3y + 9z = 6 & \cdots ③ \end{cases}$$

初期値 $x^{(0)} = 0$
$y^{(0)} = 0$
$z^{(0)} = 0$

①より
$$x = \frac{10 - y - 2z}{7}$$

②より
$$y = \frac{8 - x - 3z}{8}$$

③より
$$z = \frac{6 - 2x - 3y}{9}$$

$k = 1$

$$x^{(1)} = \frac{10 - 0 - 0}{7} \fallingdotseq 1.428857$$

$$y^{(1)} = \frac{8 - 0 - 0}{8} = 1$$

$$z^{(1)} = \frac{6 - 0 - 0}{9} \fallingdotseq 0.66667$$

$$\begin{cases} x^{(1)} = 1.428857 \\ y^{(1)} = 1 \\ z^{(1)} = 0.66667 \end{cases}$$

$k = 2$

$$x^{(2)} = \frac{10 - 1 - 2 \cdot 0.66667}{7} \fallingdotseq 1.09524$$

$$y^{(2)} = \frac{8 - 1.428857 - 3 \cdot 0.66667}{8} \fallingdotseq 0.57143$$

$$z^{(2)} = \frac{6 - 2 \cdot 1.428857 - 3 \cdot 1}{9} \fallingdotseq 0.01587$$

$$\begin{cases} x^{(2)} = 1.09524 \\ y^{(2)} = 0.57143 \\ z^{(2)} = 0.01587 \end{cases}$$

$k = 3$

$$x^{(3)} = \frac{10 - 0.57143 - 2 \cdot 0.01587}{7} \fallingdotseq 1.34240$$

$$y^{(3)} = \frac{8 - 1.09524 - 3 \cdot 0.57143}{8} \fallingdotseq 0.64881$$

$$z^{(3)} = \frac{6 - 2 \cdot 1.09524 - 3 \cdot 0.57143}{9} \fallingdotseq 0.23280$$

$$\begin{cases} x^{(3)} = 1.34240 \\ y^{(3)} = 0.64881 \\ z^{(3)} = 0.23280 \end{cases}$$

ガウス・ザイデル法　ちょっと手計算してみよう

$$\begin{cases} 7x + y + 2z = 10 & \cdots ① \\ x + 8y + 3z = 8 & \cdots ② \\ 2x + 3y + 9z = 6 & \cdots ③ \end{cases}$$

初期値 $x^{(0)} = 0$
$y^{(0)} = 0$
$z^{(0)} = 0$

①より
$$x = \frac{10 - y - 2z}{7}$$

②より
$$y = \frac{8 - x - 3z}{8}$$

③より
$$z = \frac{6 - 2x - 3y}{9}$$

$k = 1$

$$x^{(1)} = \frac{10 - y^{(0)} - 2z^{(0)}}{7} = \frac{10 - 0 - 0}{7} \fallingdotseq 1.42857$$

$$y^{(1)} = \frac{8 - x^{(1)} - 3z^{(0)}}{8} = \frac{8 - 1.428857 - 0}{8} = 0.82143$$

$$z^{(1)} = \frac{6 - 2x^{(1)} - 3y^{(1)}}{9} = \frac{6 - 2.857714 - 2.46429}{9} \fallingdotseq 0.07540$$

$$\begin{cases} x = 1.42857 \\ y = 0.82143 \\ z = 0.07540 \end{cases}$$

$k = 2$

$$x^{(2)} = \frac{10 - y^{(1)} - 2z^{(1)}}{7} = \frac{10 - 0.82143 - 0.15080}{7} \fallingdotseq 1.28968$$

$$y^{(2)} = \frac{8 - x^{(1)} - 3z^{(1)}}{8} = \frac{8 - 1.28968 - 0.22620}{8} \fallingdotseq 0.81052$$

$$z^{(2)} = \frac{6 - 2x^{(2)} - 3y^{(2)}}{9} = \frac{6 - 2.57936 - 2.43156}{9} \fallingdotseq 0.10990$$

$$\begin{cases} x = 1.28968 \\ y = 0.81052 \\ z = 0.10990 \end{cases}$$

$k = 3$

$$x^{(3)} = \frac{10 - y^{(2)} - 2z^{(2)}}{7} = \frac{10 - 0.81052 - 0.21980}{7} \fallingdotseq 1.28138$$

$$y^{(3)} = \frac{8 - x^{(2)} - 3z^{(2)}}{8} = \frac{8 - 1.28138 - 0.32970}{8} \fallingdotseq 0.79862$$

$$z^{(3)} = \frac{6 - 2x^{(2)} - 3y^{(2)}}{9} = \frac{6 - 2.56276 - 2.39586}{9} \fallingdotseq 0.11571$$

$$\begin{cases} x = 1.28138 \\ y = 0.79862 \\ z = 0.11571 \end{cases}$$