数値計算-前期中間レポート

HI4 45 番 山口惺司

2024 年 5 月 29 日

課題 1.1

1.1.a 問題:

次の連立 1 次方程式をガウス・ザイデル法で解け.(解析解は $x = [0.643, -0.071]^T$)

$$\begin{cases} 3x_1 - x_2 = 2 \\ 2x_1 + 4x_2 = 1 \end{cases} \tag{1}$$

1.1.b アルゴリズム:

ガウス・ザイデル法のアルゴリズムを図 1 に示す.

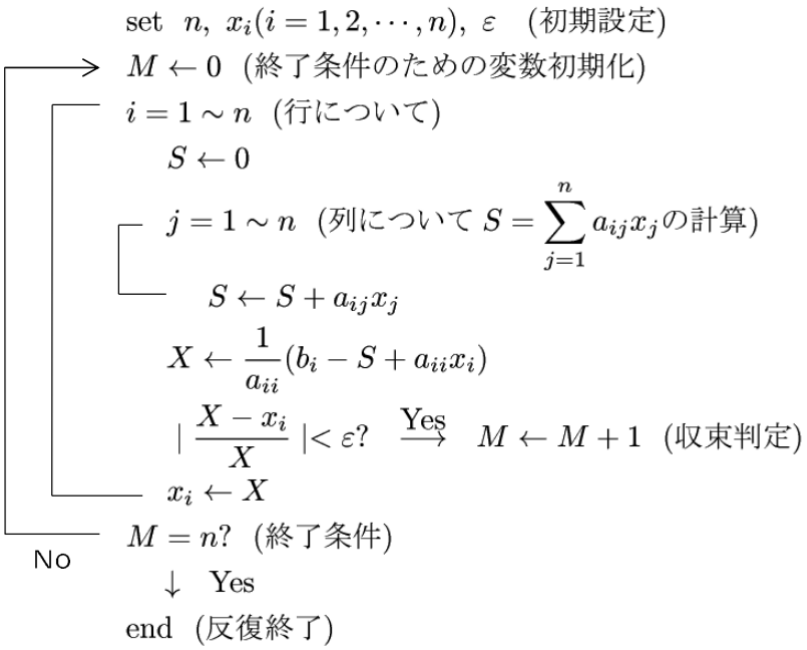

図 1　ガウス・ザイデル法のアルゴリズム

1.1.c プログラムリスト:

使用したプログラムを以下に示す.

ソースコード 1　演習課題 1.1 Python プログラム

```
#Gauss-Seidel
import numpy as np
import math
import matplotlib.pyplot as plt
import japanize_matplotlib

#ガウス・ザイデル法を用いた反復計算関数
def GaussSeidel(a, b, ans):
    n = len(a)
    x = [1] * n
    e = 1.0E-5
    M = 0
    x1 = [1]
    x2 = [1]
    count = 0
    print("初期値:",x)

    while(M != n): #終了条件
        M = 0
        for i in range(n): #行について
            S = 0;
            for j in range(n): #列について
                S += a[i][j] * x[j]
            X = (b[i] - S + a[i][i] * x[i]) / a[i][i]
            if X != 0:
                if abs((X - x[i]) / X) < e:
                    M += 1 #収束判定
            x[i] = X
        x1.append(x[0])
        x2.append(x[1])
        count += 1
        print(count,": ", x, sep = '')

    print("反復回数:", count)
    print("解析解の推定値と実測値の誤差率 [x1, x2](%): [", end = '')
    for i in range(n):
        print(error(x[i], ans[i]), end = '')
        if i == n - 1:
            print("]", sep = '')
        else:
            print(", ", end = '')
```

```
42      assX = assignment(a, x)
43      print("算出した解析解を連立方程式に代入:", assX)
44      print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2](%): [", end = '')
45      for i in range(n):
46          print(error(assX[i], b[i]), end = '')
47          if i == n - 1:
48              print("]", sep = '')
49          else:
50              print(", ", end = '')
51
52      graph(x1, x2, ans)
53
54  #誤差率を求める関数
55  def error(est, act):
56      return round(abs(100 * (est - act) / act), 3)
57
58  #連立方程式に解析解を代入する関数
59  def assignment(a, x):
60      n = len(a)
61      ans = [0] * n
62      for i in range(n):
63          for j in range(n):
64              ans[i] += a[i][j] * x[j]
65      return(ans)
66
67  #グラフの作図
68  def graph(x1, x2, ans):
69      x = np.linspace(-5, 5, 100)
70      y = -(2 - 3 * x)
71      plt.plot(x, y,label="3x$_{1}$ - x$_{2}$ = 2")
72      y = (1 - 2 * x) / 4
73      plt.plot(x, y, label="2x$_{1}$ + 4x$_{2}$ = 1")
74
75      plt.scatter(x1[1:], x2[1:])
76      plt.scatter(x1[0], x2[0], c = "magenta", label = "初期値")
77      plt.scatter(ans[0], ans[1], c = "cyan", label = "与えられた解析解")
78
79      plt.title("演習課題 1-1")
80      plt.xlabel("x$_{1}$")
81      plt.ylabel("x$_{2}$")
82      plt.legend()
83      plt.grid()
```

```
84      plt.xlim(0, 1.5)
85      plt.ylim(-1, 1.5)
86      plt.show()
87
88  #main 関数
89  def main():
90      a = [[3, -1], [2, 4]]
91      b = [2, 1]
92      ans = [0.643, -0.071]
93      GaussSeidel(a, b, ans)
94
95  if __name__ == "__main__":
96      main()
```

1.1.d 実行結果:

プログラムの実行結果を以下に示す.
また, 出力したグラフを図 2 に示す.
初期値: [1, 1]
1: [1.0, -0.25]
2: [0.5833333333333334, -0.041666666666666685]
3: [0.6527777777777778, -0.0763888888888889]
4: [0.6412037037037037, -0.07060185185185186]
5: [0.6431327160493827, -0.07156635802469136]
6: [0.6428112139917695, -0.07140560699588477]
7: [0.6428647976680384, -0.0714323988340192]
8: [0.6428558670553269, -0.07142793352766347]
9: [0.6428573554907788, -0.07142867774538941]
10: [0.6428571074182036, -0.07142855370910178]
反復回数: 10
解析解の推定値と実測値の誤差率 [x1, x2](%): [0.022, 0.604]
算出した解析解を連立方程式に代入: [1.9999998759637125, 1.0]
問題の右辺と代入して求めた右辺の誤差率 [b1, b2](%): [0.0, 0.0]
出力したグラフを図 2 に示す.

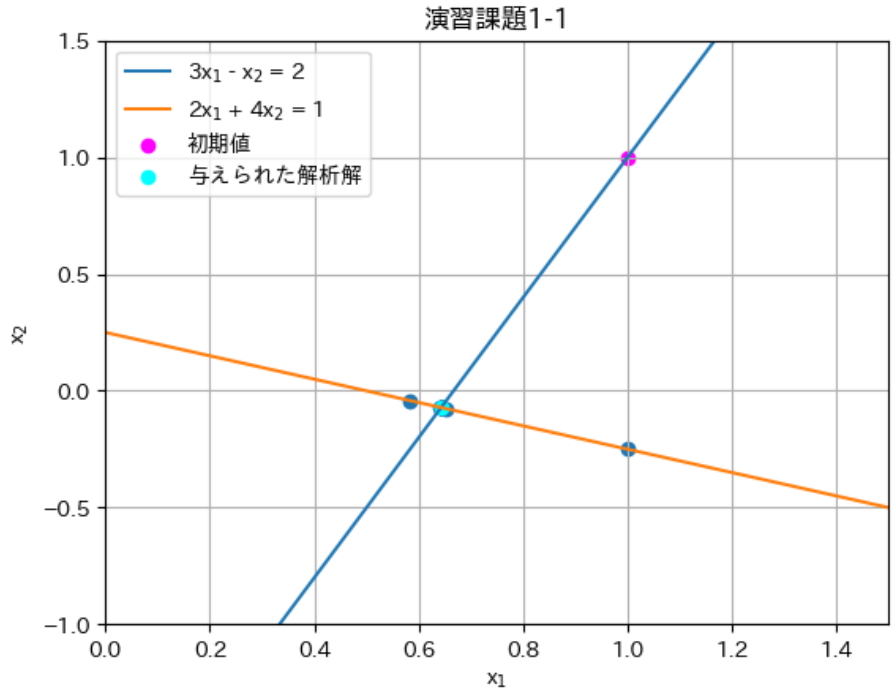

図 2　演習課題 1-1 グラフ

1.1.e 考察:

実行結果から ϵ の値を 10^{-5} に設定した時の反復回数は 10 回であった.
また, 解析解の推定値は $x_1 = 0.643, x_2 = -0.071$ であり, 反復するにつれて推定値に近づいていることがわかる.
算出した解析解を (1) 式の左辺に代入し計算したところ, 答えが 1.999..., 1.0... となり,(1) 式の右辺とほとんど一致するため, アルゴリズムを正しくプログラムに書くことができているとわかる.
グラフからも解析解が初期位置から, 直線の交点に近づいていることがわかる.

課題 1.2

1.2.a 問題:

次の連立 1 次方程式をガウス・ザイデル法と SOR 法で解け, ただし, このままで収束しない場合は行を入れ替えるピボット選択が必要になる.(解析解は $x = [0.667, 1.444]^T$)

$$\begin{cases} 4x_1 + 6_2 = 6 \\ 5x_1 + 3x_2 = 1 \end{cases} \tag{2}$$

1.2.b アルゴリズム:

ガウス・ザイデル法のアルゴリズムは課題 1.1 の図 1 を参照.
SOR 法のアルゴリズムを図 3 に示す.

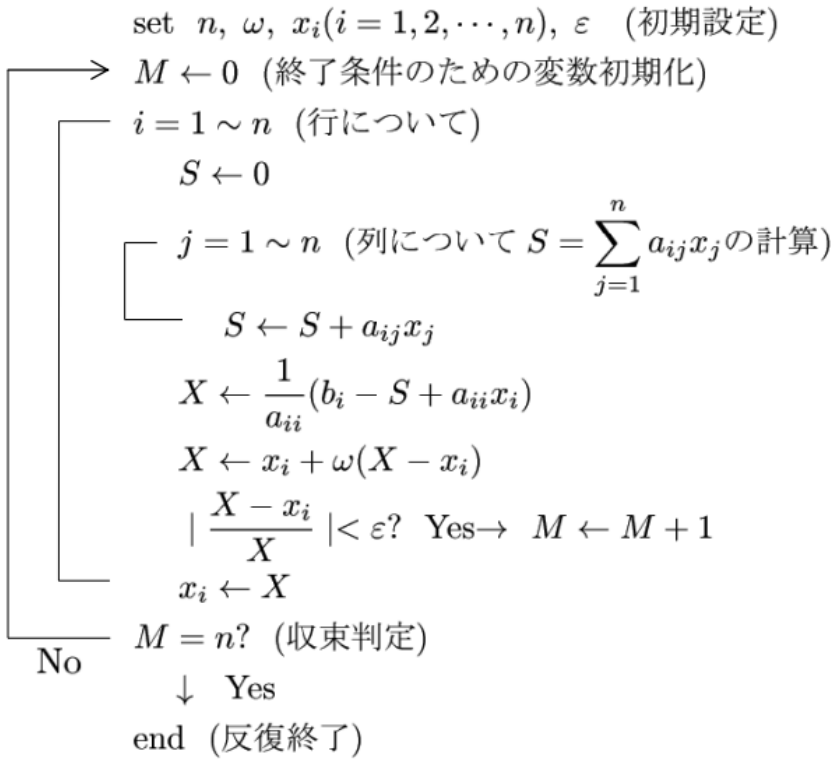

図 3　SOR 法のアルゴリズム

1.2.c プログラムリスト:

使用したガウス・ザイデル法・SOR 法プログラムを以下に示す.

ソースコード 2　演習課題 1.2-GaussSeidel Python プログラム

```
1 #Gauss-Seidel
2 import numpy as np
```

```
3 import math
4 import matplotlib.pyplot as plt
5 import japanize_matplotlib
6
7 #ガウス・ザイデル法を用いた反復計算関数
8 def gauss_seidel(a, b, ans):
9     n = len(a)
10     x = [1] * n
11     e = 1.0E-4
12     M = 0
13     x1 = [1]
14     x2 = [1]
15     count = 0
16     a1 = a.copy()
17     b1 = b.copy()
18     print("初期値:", x)
19
20     pivot_selection(a, b)
21
22     while(M != n): #終了条件
23         M = 0
24         for i in range(n): #行について
25             S = 0;
26             for j in range(n): #列について
27                 S += a[i][j] * x[j]
28             X = (b[i] - S + a[i][i] * x[i]) / a[i][i]
29             if X != 0:
30                 if abs((X - x[i]) / X) < e:
31                     M += 1 #収束判定
32             x[i] = X
33         x1.append(x[0])
34         x2.append(x[1])
35         count += 1
36         print(count,": ", x, sep = '')
37
38     print("反復回数:", count)
39     print("解析解の推定値と実測値の誤差率 [x1, x2](%): [", end = '')
40     for i in range(n):
41         print(error(x[i], ans[i]), end = '')
42         if i == n - 1:
43             print("]", sep = '')
44         else:
```

```
45              print(", ", end = '')
46      assX = assignment(a1, x)
47      print("算出した解析解を連立方程式に代入:", assX)
48      print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2](%): [", end = '')
49      for i in range(n):
50          print(error(assX[i], b1[i]), end = '')
51          if i == n - 1:
52              print("]", sep = '')
53          else:
54              print(", ", end = '')
55
56      graph(x1, x2, ans)
57
58  #ピボット選択
59  def pivot_selection(a, b):
60      n = len(a)
61      for i in range(n):
62          for j in range(n):
63              if a[j][i] > a[i][i] and i != j:
64                  tmp = a[i]
65                  a[i] = a[j]
66                  a[j] = tmp
67                  tmp = b[i]
68                  b[i] = b[j]
69                  b[j] = tmp
70
71  #誤差率を求める関数
72  def error(est, act):
73      return round(abs(100 * (est - act) / act), 3)
74
75  #連立方程式に解析解を代入する関数
76  def assignment(a, x):
77      n = len(a)
78      ans = [0] * n
79      for i in range(n):
80          for j in range(n):
81              ans[i] += a[i][j] * x[j]
82      return(ans)
83
84  #グラフの作図
85  def graph(x1, x2, ans):
86      x = np.linspace(-5, 5, 100)
```

```
87      y = (6 - 4 * x)/6
88      plt.plot(x, y,label="4x$_{1}$ + 6x$_{2}$ = 6")
89      y = (1 - 5 * x)/3
90      plt.plot(x, y, label="5x$_{1}$ + 3x$_{2}$ = 1")
91
92      plt.scatter(x1[1:], x2[1:])
93      plt.scatter(x1[0], x2[0], c = "magenta", label = "初期値")
94      plt.scatter(ans[0], ans[1], c = "cyan", label = "解析解")
95
96      plt.title("演習課題 1-2(ガウス・ザイデル法)")
97      plt.xlabel("x$_{1}$")
98      plt.ylabel("x$_{2}$")
99      plt.legend()
100     plt.grid()
101     plt.xlim(-2, 2)
102     plt.ylim(-1, 3)
103     plt.show()
104
105 #main 関数
106 def main():
107     a = [[4, 6], [5, 3]]
108     b = [6, 1]
109     ans = [-0.667, 1.444]
110     gauss_seidel(a, b, ans)
111
112 if __name__ == "__main__":
113     main()
```

ソースコード 3　演習課題 1.2-SOR Python プログラム

```
1  #SOR
2  import numpy as np
3  import math
4  import matplotlib.pyplot as plt
5  import japanize_matplotlib
6
7  #SOR 法を用いた反復計算関数
8  def SOR(a, b, ans, o):
9      n = len(a)
10     x = [1] * n
11     e = 1.0E-4
```

```
12      M = 0
13      x1 = [1]
14      x2 = [1]
15      count = 0
16      a1 = a.copy()
17      b1 = b.copy()
18      print("初期値:", x)
19
20      pivot_selection(a, b)
21
22      while(M != n): #終了条件
23          M = 0
24          for i in range(n): #行について
25              S = 0;
26              for j in range(n): #列について
27                  S += a[i][j] * x[j]
28              X = (b[i] - S + a[i][i] * x[i]) / a[i][i]
29              X = x[i] + o * (X - x[i])
30              if X != 0:
31                  if abs((X - x[i]) / X) < e:
32                      M += 1 #収束判定
33              x[i] = X
34          x1.append(x[0])
35          x2.append(x[1])
36          count += 1
37          print(count,": ", x, sep = '')
38
39      print("反復回数:", count)
40      print("解析解の推定値と実測値の誤差率 [x1, x2](%): [", end = '')
41      for i in range(n):
42          print(error(x[i], ans[i]), end = '')
43          if i == n - 1:
44              print("]", sep = '')
45          else:
46              print(", ", end = '')
47      assX = assignment(a1, x)
48      print("算出した解析解を連立方程式に代入:", assX)
49      print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2](%): [", end = '')
50      for i in range(n):
51          print(error(assX[i], b1[i]), end = '')
52          if i == n - 1:
53              print("]", sep = '')
```

```
54          else:
55              print(", ", end = '')
56
57      graph(x1, x2, ans)
58
59  #誤差率を求める関数
60  def error(est, act):
61      return round(abs(100 * (est-act) / act), 3)
62
63  #ピボット選択
64  def pivot_selection(a, b):
65      n = len(a)
66      for i in range(n):
67          for j in range(n):
68              if a[j][i] > a[i][i] and i != j:
69                  tmp = a[i]
70                  a[i] = a[j]
71                  a[j] = tmp
72                  tmp = b[i]
73                  b[i] = b[j]
74                  b[j] = tmp
75
76  #連立方程式に解析解を代入する関数
77  def assignment(a, x):
78      n = len(a)
79      ans = [0] * n
80      for i in range(n):
81          for j in range(n):
82              ans[i] += a[i][j] * x[j]
83      return(ans)
84
85  #グラフの作図
86  def graph(x1, x2, ans):
87      x = np.linspace(-5, 5, 100)
88      y = (6 - 4 * x)/6
89      plt.plot(x, y,label="4x$_{1}$ + 6x$_{2}$ = 6")
90      y = (1 - 5 * x)/3
91      plt.plot(x, y, label="5x$_{1}$ + 3x$_{2}$ = 1")
92
93      plt.scatter(x1[1:], x2[1:])
94      plt.scatter(x1[0], x2[0], c = "magenta", label = "初期値")
95      plt.scatter(ans[0], ans[1], c = "cyan", label = "解析解")
```

```
96
97      plt.title("演習課題 1-2(SOR 法)")
98      plt.xlabel("x$_{1}$")
99      plt.ylabel("x$_{2}$")
100     plt.legend()
101     plt.grid()
102     plt.xlim(-2, 2)
103     plt.ylim(-1, 3)
104     plt.show()
105
106 #main 関数
107 def main():
108     a = [[4, 6], [5, 3]]
109     b = [6, 1]
110     ans = [-0.667, 1.444]
111     SOR(a, b, ans, 1.11)
112
113 if __name__ == "__main__":
114     main()
```

1.2.d 実行結果:

ガウス・ザイデル法・SOR 法を使用したプログラムの実行結果を以下に示す.
また, 出力したグラフを図 4,5 に示す.

—ガウス・ザイデル法—
初期値: [1, 1]
1: [-0.4, 1.2666666666666666]
2: [-0.5599999999999999, 1.3733333333333333]
3: [-0.624, 1.4160000000000001]
4: [-0.6496000000000001, 1.4330666666666667]
5: [-0.65984, 1.4398933333333332]
6: [-0.663936, 1.4426240000000001]
7: [-0.6655744, 1.4437162666666667]
8: [-0.66622976, 1.4441531733333335]
9: [-0.6664919040000001, 1.4443279359999999]
10: [-0.6665967615999999, 1.4443978410666667]

11: [-0.6666387046400001, 1.4444258030933332]
反復回数: 11
解析解の推定値と実測値の誤差率 [x1, x2](%): [0.054, 0.029]
算出した解析解を連立方程式に代入: [6.0, 1.0000838860799997]
問題の右辺と代入して求めた右辺の誤差率 [b1, b2](%): [0.0, 0.008]

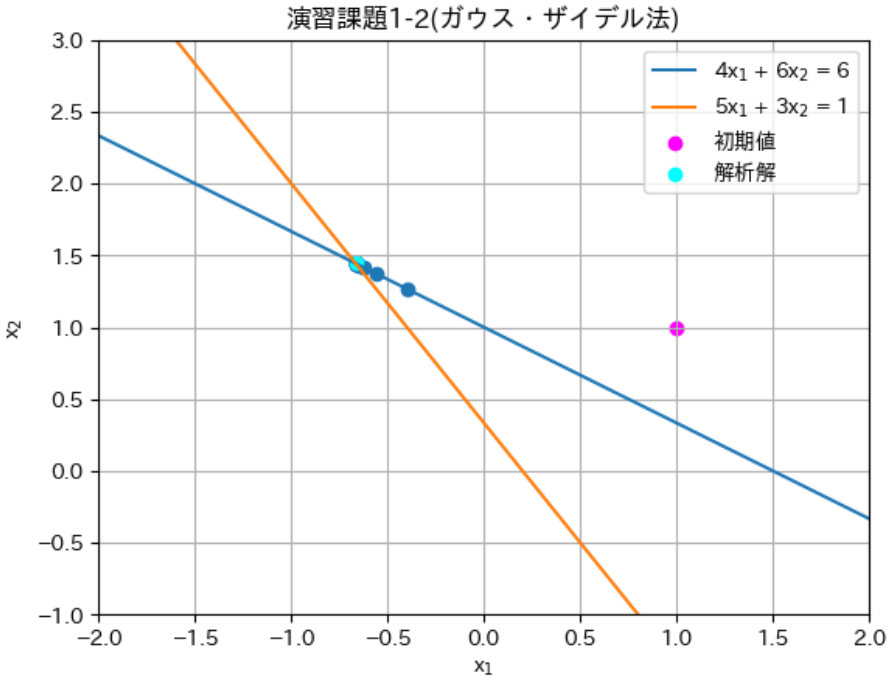

図 4　演習課題 1-2-GaussSeidel グラフ

——SOR 法——
初期値: [1, 1]
1: [-0.554, 1.4099600000000003]
2: [-0.6560933600000004, 1.4404134864]
3: [-0.6651451123423999, 1.443761899629376]
4: [-0.6663794627955004, 1.444306993509439]
5: [-0.6666067167697812, 1.4444152011235998]
6: [-0.6666537851036415, 1.4444381288530987]
反復回数: 6
解析解の推定値と実測値の誤差率 [x1, x2](%): [0.052, 0.03]
算出した解析解を連立方程式に代入: [6.000013632704027, 1.000045461041089]

問題の右辺と代入して求めた右辺の誤差率 [b1, b2](%): [0.0, 0.005]

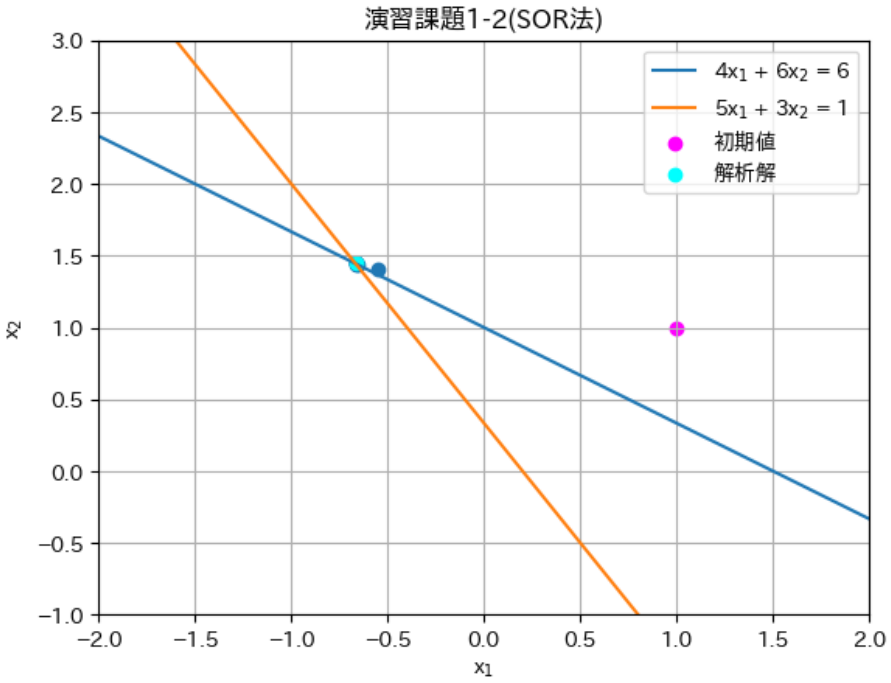

図 5　演習課題 1-2-SOR グラフ

1.2.e 考察:

$\epsilon = 10^{-5}$ のとき, ガウス・ザイデル法を使用した時は反復回数が 10 回,SOR 法を使用した時は反復回数が 6 回となった. この結果により,SOR 法はガウス・ザイデル法より速く処理を終わらせることができることがわかった.

グラフを見ても, ガウス・ザイデル法より SOR 法のほうが速く正しい解析解に近づいていることがわかる. また,SOR 法の ω を 0.01〜1.99 まで 0.01 刻みで値を増やしていったら, ω=1.11, 1.12, 1.16 において反復回数が 6 回で最も少なかったため, 今回の ω の値は 1.11 としている.

算出した解析解を (2) 式の左辺に代入し計算したところ, 答えが 6.000..., 1.000... となり,(2) 式の右辺とほとんど一致するため, アルゴリズムを正しくプログラムに書くことができているとわかる.

課題 1.3

1.3.a 問題:

次の連立 1 次方程式をガウス・ザイデル法と SOR 法で解け.(解析解は $x = [-1, -1, 4]^T$)

$$\begin{cases} 2x_1 + x_2 + x_3 = 1 \\ 2x_1 + x_3 = 2 \\ x_1 + x_2 + x_3 = 2 \end{cases} \tag{3}$$

1.3.b アルゴリズム:

ガウス・ザイデル法のアルゴリズムは課題 1.1 の図 1 を参照.
SOR 法のアルゴリズムは課題 1.2 の図 3 を参照.

1.3.c プログラムリスト:

使用したガウス・ザイデル法・SOR 法プログラムを以下に示す.

ソースコード 4　演習課題 1.3-GaussSeidel Python プログラム

```
1  #Gauss-Seidel
2  import numpy as np
3  import math
4  import matplotlib.pyplot as plt
5
6  #ガウス・ザイデル法を用いた反復計算関数
7  def GaussSeidel(a, b, ans):
8      n = len(a)
9      x = [1] * n
10     e = 1.0E-4
11     M = 0
12     x1 = [1]
13     x2 = [1]
14     x3 = [1]
15     count = 0
16     print("初期値:", x)
17
18     while(M != n): #終了条件
```

```
19          M = 0
20          for i in range(n): #行について
21              S = 0;
22              for j in range(n): #列について
23                  S += a[i][j] * x[j]
24              X = (b[i] - S + a[i][i] * x[i]) / a[i][i]
25              if X != 0:
26                  if abs((X - x[i]) / X) < e:
27                      M += 1 #収束判定
28              x[i] = X
29          x1.append(x[0])
30          x2.append(x[1])
31          x3.append(x[2])
32          count += 1
33          print(count,": ", x, sep = '')
34
35      print("反復回数:", count)
36      print("解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [", end = '')
37      for i in range(n):
38          print(error(x[i], ans[i]), end = '')
39          if i == n - 1:
40              print("]", sep = '')
41          else:
42              print(", ", end = '')
43      assX = assignment(a, x)
44      print("算出した解析解を連立方程式に代入:", assX)
45      print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [", end = '')
46      for i in range(n):
47          print(error(assX[i], b[i]), end = '')
48          if i == n - 1:
49              print("]", sep = '')
50          else:
51              print(", ", end = '')
52
53  #誤差率を求める関数
54  def error(est, act):
55      return round(abs(100 * (est-act) / act), 3)
56
57  #連立方程式に解析解を代入する関数
58  def assignment(a, x):
59      n = len(a)
60      ans = [0] * n
```

```
    for i in range(n):
        for j in range(n):
            ans[i] += a[i][j] * x[j]
    return(ans)

#main 関数
def main():
    a = [[2, 1, 1], [0, 2, 1], [1, 1, 1]]
    b = [1, 2, 2]
    ans = [-1, -1, 4]
    GaussSeidel(a, b, ans)

if __name__ == "__main__":
    main()
```

ソースコード 5　演習課題 1.3-SOR Python プログラム

```
#SOR
import numpy as np
import math
import matplotlib.pyplot as plt

#SOR 法を用いた反復計算関数
def SOR(a, b, ans, o):
    n = len(a)
    x = [1] * n
    e = 1.0E-4
    M = 0
    x1 = [1]
    x2 = [1]
    x3 = [1]
    count = 0
    print("初期値:", x)

    while(M != n): #終了条件
        M = 0
        for i in range(n): #行について
            S = 0;
            for j in range(n): #列について
                S += a[i][j] * x[j]
            X = (b[i] - S + a[i][i] * x[i]) / a[i][i]
```

```
25              X = x[i] + o * (X - x[i])
26              if X != 0:
27                  if abs((X - x[i]) / X) < e:
28                      M += 1 #収束判定
29              x[i] = X
30          x1.append(x[0])
31          x2.append(x[1])
32          x3.append(x[2])
33          count += 1
34          print(count,": ", x, sep = '')
35
36      print("反復回数:", count)
37      print("解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [", end = '')
38      for i in range(n):
39          print(error(x[i], ans[i]), end = '')
40          if i == n-1:
41              print("]", sep = '')
42          else:
43              print(", ", end = '')
44      assX = assignment(a, x)
45      print("算出した解析解を連立方程式に代入:", assX)
46      print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [", end = '')
47      for i in range(n):
48          print(error(assX[i], b[i]), end = '')
49          if i == n-1:
50              print("]", sep = '')
51          else:
52              print(", ", end = '')
53
54  #誤差率を求める関数
55  def error(est, act):
56      return round(abs(100 * (est-act) / act), 3)
57
58  #連立方程式に解析解を代入する関数
59  def assignment(a, x):
60      n = len(a)
61      ans = [0] * n
62      for i in range(n):
63          for j in range(n):
64              ans[i] += a[i][j] * x[j]
65      return(ans)
66
```

```
#main 関数
def main():
    a = [[2, 1, 1], [0, 2, 1], [1, 1, 1]]
    b = [1, 2, 2]
    ans = [-1, -1, 4]
    SOR(a, b, ans, 1.04)

if __name__ == "__main__":
    main()
```

1.3.d 実行結果:

ガウス・ザイデル法・SOR 法を使用したプログラムの実行結果を以下に示す.

ガウス・ザイデル法：
初期値: [1, 1, 1]
1: [-0.5, 0.5, 2.0]
2: [-0.75, 0.0, 2.75]
3: [-0.875, -0.375, 3.25]
4: [-0.9375, -0.625, 3.5625]
5: [-0.96875, -0.78125, 3.75]
6: [-0.984375, -0.875, 3.859375]
7: [-0.9921875, -0.9296875, 3.921875]
8: [-0.99609375, -0.9609375, 3.95703125]
9: [-0.998046875, -0.978515625, 3.9765625]
10: [-0.9990234375, -0.98828125, 3.9873046875]
11: [-0.99951171875, -0.99365234375, 3.9931640625]
12: [-0.999755859375, -0.99658203125, 3.996337890625]
13: [-0.9998779296875, -0.9981689453125, 3.998046875]
14: [-0.99993896484375, -0.9990234375, 3.99896240234375]
15: [-0.999969482421875, -0.999481201171875, 3.99945068359375]
16: [-0.9999847412109375, -0.999725341796875, 3.9997100830078125]
17: [-0.9999923706054688, -0.9998550415039062, 3.999847412109375]
18: [-0.9999961853027344, -0.9999237060546875, 3.999919891357422]
反復回数: 18
解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [0.0, 0.008, 0.002]

算出した解析解を連立方程式に代入: [1.0000038146972656, 2.000072479248047, 2.0]
問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [0.0, 0.004, 0.0]

SOR 法：
初期値: [1, 1, 1]
1: [-0.56, 0.48, 2.1232]
2: [-0.8112640000000001, -0.08326400000000012, 2.9253811200000004]
3: [-0.9254503424000002, -0.4778676224000003, 3.422435438592]
4: [-0.97415725072384, -0.7205517231718401, 3.705599915307827]
5: [-0.9932587698817598, -0.8580898870331966, 3.8571786065792417]
6: [-0.9997957833686731, -0.9314092799398779, 3.9341661215777237]
7: [-1.0014417263169326, -0.9685100120228212, 3.971383163010235]
8: [-1.001436369460778, -0.9863788442844094, 3.988472495774585]
9: [-1.0010312439964604, -0.9945505440314079, 3.9958661597179996]
10: [-1.0006428703971693, -0.9980683812921034, 3.9988250553681235]
11: [-1.000367755703644, -0.99946629353974, 3.9998744089983944]
12: [-1.0001975098103546, -0.9999560409375755, 4.0001647164179115]
13: [-1.0001006108573605, -1.000087410899811, 4.000188953970742]
14: [-1.0000487779625897, -1.0000947596287935, 4.000141720936209]
反復回数: 14
解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [0.005, 0.009, 0.004]
算出した解析解を連立方程式に代入:
[0.9999494053822358, 1.9999522016786218, 1.9999981833448253]
問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [0.005, 0.002, 0.0]

1.3.e 考察:

$\epsilon = 10^{-4}$ のとき, ガウス・ザイデル法を使用した時は反復回数が 18 回,SOR 法を使用した時は反復回数が 14 回となった. この結果により,SOR 法はガウス・ザイデル法より速く処理を終わらせることができることがわかった.
また,SOR 法の ω を 0.01〜1.68 まで 0.01 刻みで値を増やしていったら, ω=1.04, 1.05, 1.11, 1.12, 1.13 において反復回数が 14 回で最も少なかったため, 今回の ω の値は 1.04 としている.
ω を 1.68 より大きくすると, 反復回数が大幅に増え, 処理が終わらなくなった.
算出した解析解を (3) 式の左辺に代入し計算したところ, 答えが 0.999..., 1.999..., 1.999... となり,(3) 式の右辺とほとんど一致するため, アルゴリズムを正しくプログラムに書くことができているとわかる.

課題 1.4

1.4.a 問題:

次の連立 1 次方程式をガウス・ジョルダン法で解け.(解析解は $x = [-1, -1, 4]^T$)

$$\begin{cases} 2x_1 + x_2 + x_3 = 1 \\ 2x_1 + x_3 = 2 \\ x_1 + x_2 + x_3 = 2 \end{cases} \tag{4}$$

1.4.b アルゴリズム:

ガウス・ジョルダン法のアルゴリズムを図 6 に示す.

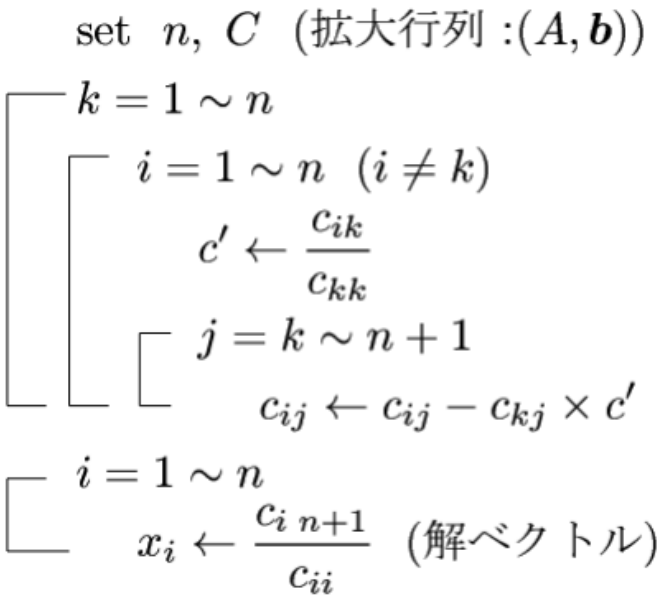

図 6　ガウス・ジョルダン法のアルゴリズム

1.4.c プログラムリスト:

使用したガウス・ジョルダン法プログラムを以下に示す.

ソースコード 6　演習課題 1.4-GaussJordan Python プログラム

```
#Gauss-Jordan
import numpy as np
import math
import matplotlib.pyplot as plt
import copy

#ガウス・ジョルダン法を用いた反復計算関数
def GaussJordan(a, b, ans):
    n = len(a)
    a1 = copy.deepcopy(a)
    C = a
    x = []
    for i in range(n):
        C[i].append(b[i]) #a,b の拡大係数行列
    x1 = [1]
    x2 = [1]
    x3 = [1]
    count = 0

    for k in range(n):
        for i in range(n):
            if i != k:
                Ctmp = C[i][k] / C[k][k]
                for j in range(k, n+1):
                    C[i][j] = C[i][j] - C[k][j] * Ctmp

    for i in range(n):
        x.append(C[i][n]/C[i][i])

    print("解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [", end = '')
    for i in range(n):
        print(error(x[i], ans[i]), end = '')
        if i == n-1:
            print("]", sep = '')
        else:
            print(", ", end = '')
    assX = assignment(a1, x)
```

```
    print("算出した解析解を連立方程式に代入:", assX)
    print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [", end = '')
    for i in range(n):
        print(error(assX[i], b[i]), end = '')
        if i == n-1:
            print("]", sep = '')
        else:
            print(", ", end = '')

#誤差率を求める関数
def error(est, act):
    return round(abs(100 * (est-act) / act), 3)

#連立方程式に解析解を代入する関数
def assignment(a, x):
    n = len(a)
    ans = [0] * n
    for i in range(n):
        for j in range(n):
            ans[i] += a[i][j] * x[j]
    return(ans)

#main 関数
def main():
    a = [[2, 1, 1], [0, 2, 1], [1, 1, 1]]
    b = [1, 2, 2]
    ans = [-1, -1, 4]
    GaussJordan(a, b, ans)

if __name__ == "__main__":
    main()
```

1.4.d 実行結果:

ガウス・ジョルダン法を使用したプログラムの実行結果を以下に示す.

ガウス・ジョルダン法：
解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [0.0, 0.0, 0.0]
算出した解析解を連立方程式に代入: [1.0, 2.0, 2.0]

問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [0.0, 0.0, 0.0]

1.4.e 考察:

ガウス・ジョルダン法はガウス・ザイデルや SOR 法と違い, 近似値を求めるアルゴリズムではないので, 正確な解析解が算出される.
算出された解析解を (4) 式の左辺に代入したところ, 答えが 1.0, 2.0, 2.0 となり,(4) 式の右辺と一致するため, このプログラムは正しいと言える.
また, ガウス・ジョルダン法,SOR 法のプログラムと比べて, ガウス・ジョルダン法のプログラムはコードを短く書くことができる.

課題 1.5

1.5.a 問題:

次の連立 1 次方程式を正規化を行ったガウス・ジョルダン法で解け.(解析解は $x = [-1, -1, 4]^T$)

$$\begin{cases} 2x_1 + x_2 + x_3 = 1 \\ 2x_1 + x_3 = 2 \\ x_1 + x_2 + x_3 = 2 \end{cases} \tag{5}$$

1.5.b アルゴリズム:

正規化を行ったガウス・ジョルダン法のアルゴリズムを図 7 に示す.

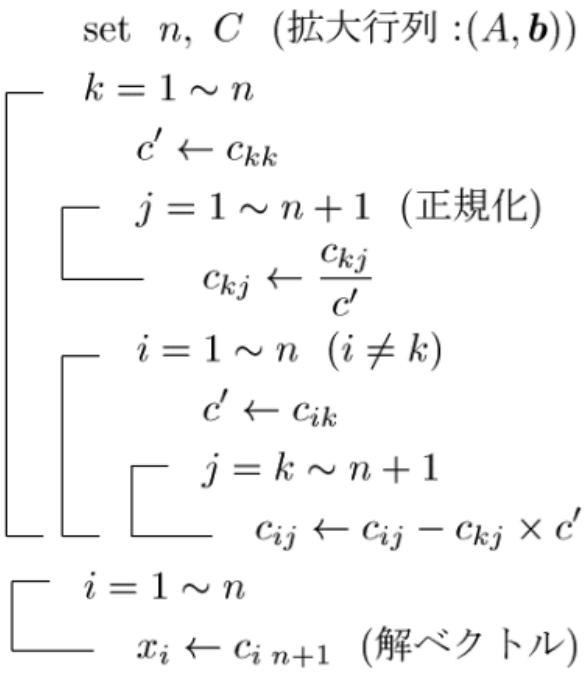

図 7　ガウス・ジョルダン法のアルゴリズム

1.5.c プログラムリスト:

使用したガウス・ジョルダン法プログラムを以下に示す.

ソースコード 7　演習課題 1.5-GaussJordan Python プログラム

```
1  #Gauss-Jordan
2  import numpy as np
3  import math
4  import copy
5
6  #ガウス・ジョルダン法を用いた反復計算関数
7  def GaussJordan(a, b, ans):
8      n = len(a)
9      a1 = copy.deepcopy(a)
10     C = a
11     x = []
12     for i in range(n):
13         C[i].append(b[i]) #a,b の拡大係数行列
14
```

```
15      x1 = [1]
16      x2 = [1]
17      x3 = [1]
18      count = 0
19
20      for k in range(n):
21          Ctmp = C[k][k]
22          for j in range(n+1):
23              C[k][j] = C[k][j]/Ctmp #正規化
24          for i in range(n):
25              if i != k:
26                  Ctmp = C[i][k]
27                  for j in range(k, n+1):
28                      C[i][j] = C[i][j] - C[k][j] * Ctmp
29
30      for i in range(n):
31          x.append(C[i][n])
32
33      print("算出された解析解: ", x)
34      print("解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [", end = '')
35      for i in range(n):
36          print(error(x[i], ans[i]), end = '')
37          if i == n - 1:
38              print("]", sep = '')
39          else:
40              print(", ", end = '')
41      assX = assignment(a1, x)
42      print("算出した解析解を連立方程式に代入:", assX)
43      print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [", end = '')
44      for i in range(n):
45          print(error(assX[i], b[i]), end = '')
46          if i == n-1:
47              print("]", sep = '')
48          else:
49              print(", ", end = '')
50
51  #誤差率を求める関数
52  def error(est, act):
53      return round(abs(100 * (est - act) / act), 3)
54
55  #連立方程式に解析解を代入する関数
56  def assignment(a, x):
```

```
57      n = len(a)
58      ans = [0] * n
59      for i in range(n):
60          for j in range(n):
61              ans[i] += a[i][j] * x[j]
62      return(ans)
63
64  #main 関数
65  def main():
66      a = [[2, 1, 1],[0, 2, 1], [1, 1, 1]]
67      b = [1, 2, 2]
68      ans = [-1, -1, 4]
69      GaussJordan(a, b, ans)
70
71  if __name__ == "__main__":
72      main()
```

1.5.d 実行結果:

ガウス・ジョルダン法を使用したプログラムの実行結果を以下に示す.
算出された解析解: [-1.0, -1.0, 4.0]
解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [0.0, 0.0, 0.0]
算出した解析解を連立方程式に代入: [1.0, 2.0, 2.0]
問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [0.0, 0.0, 0.0]

1.5.e 考察:

算出された解析解を (5) 式の左辺に代入したところ, 答えが 1.0, 2.0, 2.0 となり,(5) 式の右辺と一致するため, このプログラムは正しいと言える. 課題 1.4 と違い,1.5 のプログラムは正規化を行っている.

課題 1.6

1.6.a 問題:

次の連立 1 次方程式をガウス・ジョルダン法で解け.(解析解は $x = [1, 2, 3, 4, 5]^T$)

$$\begin{cases} 2x_1 + 3x_2 + 4x_3 + 5x_4 + 6x_5 = 70 \\ 3x_1 + 4x_2 + 5x_3 + 6x_4 + 2x_5 = 60 \\ 4x_1 + 5x_2 + 6x_3 + 2x_4 + 3x_5 = 55 \\ 5x_1 + 6x_2 + 2x_3 + 3x_4 + 4x_5 = 55 \\ 6x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 = 60 \end{cases} \tag{6}$$

1.6.b アルゴリズム:

正規化を行ったガウス・ジョルダン法のアルゴリズムは図 7 に示した通り.

1.6.c プログラムリスト:

使用したガウス・ジョルダン法プログラムを以下に示す.

ソースコード 8　演習課題 1.6-GaussJordan Python プログラム

```
1  #Gauss-Jordan
2  import numpy as np
3  import math
4  import copy
5
6  #ガウス・ジョルダン法を用いた反復計算関数
7  def GaussJordan(a, b, ans):
8      n = len(a)
9      a1 = copy.deepcopy(a)
10     b1 = copy.deepcopy(b)
11     C = a
12     x = []
13     for i in range(n):
14         C[i].append(b[i]) #a,b の拡大行列
15
16     x1 = [1]
```

```
    x2 = [1]
    x3 = [1]
    count = 0

    pivot_selection(a, b)
    print(C)
    for k in range(n):
        Ctmp = C[k][k]
        for j in range(n+1):
            C[k][j] /= Ctmp #正規化
        for i in range(n):
            if i != k:
                Ctmp = C[i][k]
                for j in range(k, n+1):
                    C[i][j] = C[i][j] - C[k][j] * Ctmp

    for i in range(n):
        x.append(C[i][n])

    print("算出された解析解: ", x)
    print("解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [", end = '')
    for i in range(n):
        print(error(x[i], ans[i]), end = '')
        if i == n-1:
            print("]", sep = '')
        else:
            print(", ", end = '')
    assX = assignment(a1, x)
    print("算出した解析解を連立方程式に代入:", assX)
    print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [", end = '')
    for i in range(n):
        print(error(assX[i], b1[i]), end = '')
        if i == n-1:
            print("]", sep = '')
        else:
            print(", ", end = '')

#ピボット選択
def pivot_selection(a, b):
    n = len(a)
    for i in range(n):
        for j in range(n):
```

```
            if a[j][i] > a[i][i] and i != j:
                tmp = a[i]
                a[i] = a[j]
                a[j] = tmp
                tmp = b[i]
                b[i] = b[j]
                b[j] = tmp

 #誤差率を求める関数
def error(est, act):
    return round(abs(100 * (est - act) / act), 3)

#連立方程式に解析解を代入する関数
def assignment(a, x):
    n = len(a)
    ans = [0] * n
    for i in range(n):
        for j in range(n):
            ans[i] += a[i][j] * x[j]
    return(ans)

#main 関数
def main():
    a = [[2, 3, 4, 5, 6],[3, 4, 5, 6, 2], [4, 5, 6, 2, 3], [5, 6, 2, 3, 4],
        [6, 2, 3, 4, 5]]
    b = [70, 60, 55, 55, 60]
    ans = [1, 2, 3, 4, 5]
    GaussJordan(a, b, ans)

if __name__ == "__main__":
    main()
```

1.6.d 実行結果:

ガウス・ジョルダン法を使用したプログラムの実行結果を以下に示す.
算出された解析解: [1.0, 1.9999999999999998, 3.0000000000000004, 4.0, 4.999999999999999] 解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [0.0, 0.0, 0.0, 0.0, 0.0] 算出した解析解を連立方程式に代入: [70.0, 60.0, 55.0, 55.0, 60.0] 問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [0.0, 0.0, 0.0, 0.0, 0.0]

1.6.e 考察:

算出された解析解を (6) 式の左辺に代入したところ, 答えが 1.0, 2.0, 2.0 となり,(6) 式の右辺と一致するため, このプログラムは正しいと言える. 今回のプログラムではピボット選択をしている. 算出された解析解に少しだけ誤差が生じたが, これはおそらく計算の過程で割り切ることができない式が出てきたためだと考える.

課題 1.7

1.7.a 問題:

次の連立 1 次方程式を逆行列を使って解け.(解析解は $x = [-3, 4, 1]^T$)

$$\begin{pmatrix} 1 & 2 & 1 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix} \tag{7}$$

1.7.b アルゴリズム:

逆行列のアルゴリズムを図 8 に示す.

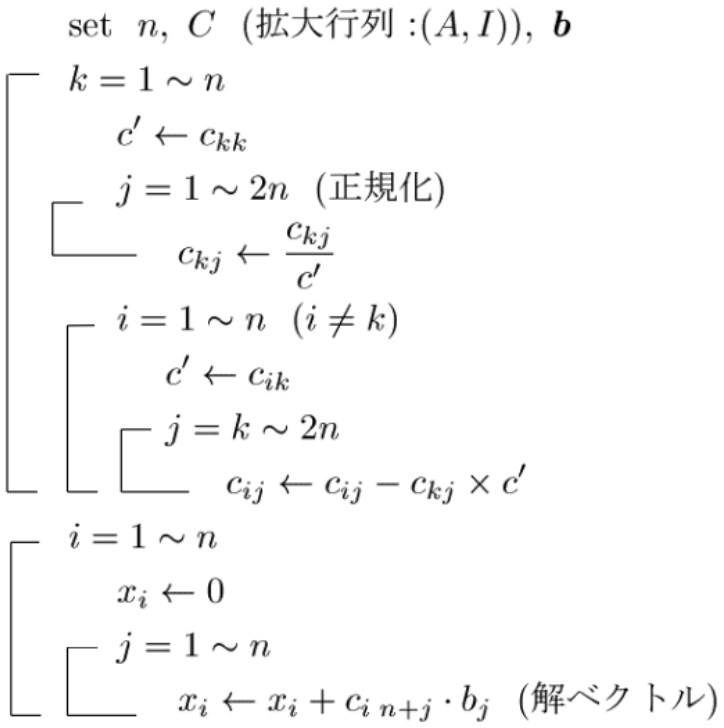

図 8　逆行列のアルゴリズム

1.7.c プログラムリスト:

使用したガウス・ジョルダン法プログラムを以下に示す.

ソースコード 9　演習課題 1.7-InverseMatrix Python プログラム

```
1  #Inverse Matrix
2  import numpy as np
3  import math
4  import copy
5
6  #逆行列による連立 1次方程式の解法
7  def InverseMatrix(a, b, ans):
8      n = len(a)
9      a1 = copy.deepcopy(a)
10     b1 = copy.deepcopy(b)
11     C = a
12     x = [0] * n
13
```

```
14      unit = np.identity(n) #単位行列の作成
15
16      for i in range(n): #拡大行列の作成
17          for j in range(n):
18              C[i].append(unit[i, j])
19      print("計算前:",C)
20      x1 = [1]
21      x2 = [1]
22      x3 = [1]
23      count = 0
24
25      for k in range(n):
26          Ctmp = C[k][k]
27          for j in range(2 * n):
28              C[k][j] /= Ctmp #正規化
29          for i in range(n):
30              if i != k:
31                  Ctmp = C[i][k]
32                  for j in range(k, 2 * n):
33                      C[i][j] = C[i][j] - C[k][j] * Ctmp
34
35      for i in range(n):
36          x[i] = 0
37          for j in range(n):
38              x[i] = x[i] + C[i][n+j] * b[j]
39      print("計算後:", C)
40      print("算出された解析解: ", x)
41      print("解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [", end = '')
42      for i in range(n):
43          print(error(x[i], ans[i]), end = '')
44          if i == n-1:
45              print("]", sep = '')
46          else:
47              print(", ", end = '')
48      assX = assignment(a1, x)
49      print("算出した解析解を連立方程式に代入:", assX)
50      print("問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [", end = '')
51      for i in range(n):
52          print(error(assX[i], b1[i]), end = '')
53          if i == n-1:
54              print("]", sep = '')
55          else:
```

```
56              print(", ", end = '')
57
58  #連立方程式に解析解を代入する関数
59  def assignment(a, x):
60      n = len(a)
61      ans = [0] * n
62      for i in range(n):
63          for j in range(n):
64              ans[i] += a[i][j] * x[j]
65      return(ans)
66
67  #main 関数
68  def main():
69      a = [[1, 2, 1],[4, 5, 6], [7, 8, 9]]
70      b = [4, 2, 2]
71      ans = [-3, 4, -1]
72      InverseMatrix(a, b, ans)
73
74  if __name__ == "__main__":
75      main()
```

1.7.d 実行結果:

計算前：[[1, 2, 1, 1.0, 0.0, 0.0], [4, 5, 6, 0.0, 1.0, 0.0], [7, 8, 9, 0.0, 0.0, 1.0]]
計算後：[[1.0, 0.0, 0.0, -0.5, -1.6666666666666665, 1.1666666666666665], [-0.0, 1.0, 0.0, 1.0, 0.3333333333333333, -0.3333333333333333], [-0.0, -0.0, 1.0, -0.5, 1.0, -0.5]]
算出された解析解: [-3.0, 4.0, -1.0]
解析解の推定値と実測値の誤差率 [x1, x2, x3](%): [0.0, 0.0, 0.0]
算出した解析解を連立方程式に代入: [4.0, 2.0, 2.0]
問題の右辺と代入して求めた右辺の誤差率 [b1, b2, b3](%): [0.0, 0.0, 0.0]

1.7.e 考察:

逆行列を使用した場合もガウス・ザイデル法や SOR 法と違い, 正確な値が算出される. 算出された解析解を (6) 式の左辺に代入したところ, 答えが 4.0, 2.0, 2.0 となり,(6) 式の右辺と一致するため, このプログラムは正しいと言える.

実行結果から計算前の拡大係数行列は (8) 式のようになる.

$$\left(\begin{array}{ccc|ccc} 1 & 2 & 1 & 1 & 0 & 0 \\ 4 & 5 & 6 & 0 & 1 & 0 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right) \tag{8}$$

実行結果から計算後の拡大係数行列は (9) 式のようになる.

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -0.5 & -1.667 & 1.167 \\ 0 & 1 & 0 & 1 & 0.333 & -0.333 \\ 0 & 0 & 1 & -0.5 & 1 & -0.5 \end{array}\right) \tag{9}$$

(8),(9) 式から正しく逆行列を求めることができている.

課題 1.8

1.8.a 問題:

$f(x) = e^x$ をマクローリン展開し, 近似次数の違いによるグラフを示せ.

1.8.b アルゴリズム:

マクローリン展開の一般形を式 (10) に示す.

$$f(x) = \sum_{k=0}^{\infty} f^{(k)}(0)\frac{x^k}{k!} \tag{10}$$

式 (8) のマクローリン展開を有限個の項 ($k = n$) で打ち切った時の近似式を式 (11) に示す.

$$f(x) \approx \sum_{k=0}^{n} f^{(k)}(x_0)\frac{(x - x_0)^k}{k!} \tag{11}$$

1.8.c プログラムリスト:

使用したマクローリン展開プログラムを以下に示す.

ソースコード 10　演習課題 1.8 Maclaurin expansion Python プログラム

```
1 #Approximating functions
2 import numpy as np
3 import math
```

```
4  import matplotlib.pyplot as plt
5  import japanize_matplotlib
6  import sympy as sym
7  from sympy.plotting import plot
8
9  #グラフの作成
10 def graph(x, y, n):
11     y1 = math.e ** x #e^x
12     plt.xlim(-5, 5)
13     plt.ylim(-5, 5)
14     plt.grid()
15     plt.plot(x, y1, label = "e$^{x}$")
16     for i in range(n):
17         plt.plot(x, y[i], "--", label = str(i) + "次近似")
18     plt.legend()
19     plt.show()
20
21 #main 関数
22 def main():
23     a = sym.symbols("a") #a を変数を表す文字として扱う
24     fa = math.e ** a #e^a
25     x = np.linspace(-5, 5, 100)
26     y = []
27     S = 0
28     n = 10
29
30     for i in range(n):
31         S = 0
32         for k in range(i+1): #マクローリン展開
33             S += ((x ** k) / math.factorial(k)) * sym.diff(fa, a, k).subs(a, 0)
34         y.append(S)
35
36     graph(x, y, n)
37
38 if __name__ == "__main__":
39     main()
```

1.8.d 実行結果:

マクローリン展開を使用したプログラムの実行結果を図 9 に示す.

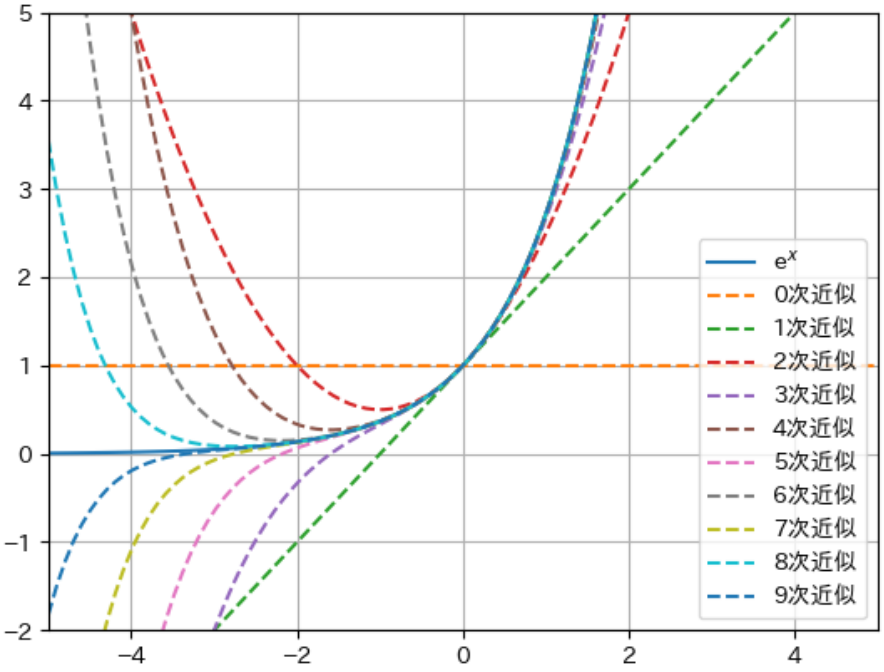

図 9　演習課題 1-8 Maclaurin expansion グラフ

1.8.e 考察:

今回は n=0〜9, つまり 9 次近似まで求めた. n が大きくなるにつれて関数 e^x に近づいて行っているのがわかる. また,$f(x) = e^x$ をマクローリン展開の式に当てはめると,

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots \tag{12}$$

となる.

応用課題 1.9

1.9.a 問題:

図 10 のような直流電流 $E_1 = 100[V], E_2 = 100[V]$, 抵抗 $R = 1[\Omega]$ から構成された回路網がある. 各節点の電圧 $V_1, V_2, \cdots, V_{12}$をガウス・ザイデル法と逆行列の 2 つの解法を使って解け.(解は $V =$

$[39.785, 48.387, 39.785, 10.753, 13.978, 10.753, -10.753, -13.978, -10.753, -39.785, -48.387, -39.785]^T)$

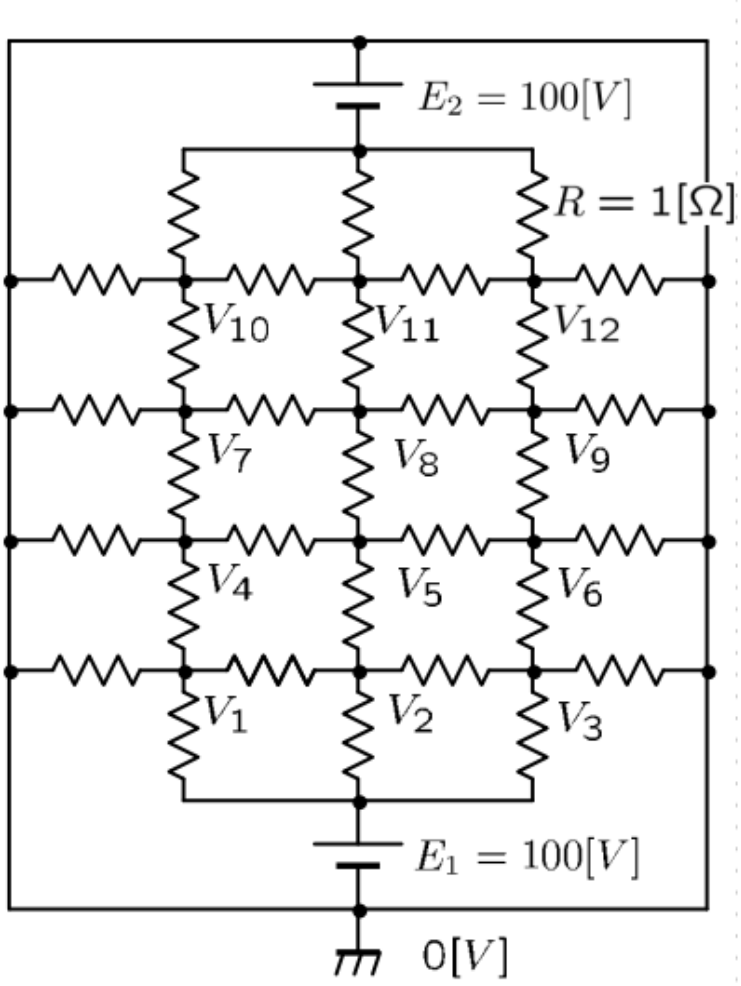

図 10　応用課題 1.9 の回路図

図 9 の回路図から式 (10) のような 12 元連立 1 次方程式が得られる.

$$\begin{cases} -4x_1 + x_2 + x_4 = -100 \\ x_1 - 4x_2 + x_3 + x_5 = -100 \\ x_2 - 4x_3 + x_6 = -100 \\ x_1 - 4x_4 + x_5 + x_7 = 0 \\ x_2 + x_4 - 4x_5 + x_6 + x_8 = 0 \\ x_3 + x_5 - 4x_6 + x_9 = 0 \\ x_4 + x_8 - 4x_7 + x_{10} = 0 \\ x_5 + x_7 - 4x_8 + x_9 + x_{11} = 0 \\ x_6 + x_8 - 4x_9 + x_{12} = 0 \\ x_7 - 4x_{10} + x_{11} = 100 \\ x_8 + x_{10} - 4x_{11} + x_{12} = 100 \\ x_9 + x_{11} - 4x_{12} = 100 \end{cases} \tag{13}$$

1.9.b アルゴリズム:

ガウス・ザイデル法, 逆行列のアルゴリズムはそれぞれ図 1,8 に示した通り.

1.9.c プログラムリスト:

使用したガウス・ザイデル法, 逆行列プログラムを以下に示す.

ソースコード 11　演習課題 1.9-Gauss-Seidel Python プログラム

```
1  #Gauss-Seidel
2  import numpy as np
3  import math
4  import matplotlib.pyplot as plt
5  import japanize_matplotlib
6
7  #ガウス・ザイデル法を用いた反復計算関数
8  def gauss_seidel(a, b, ans):
9      n = len(a)
10     x = [1] * n
11     e = 1.0E-4
12     M = 0
```

```
13      a1 = a.copy()
14      b1 = b.copy()
15
16      count = 0
17
18      while(M != n): #終了条件
19          M = 0
20          for i in range(n): #行について
21              S = 0;
22              for j in range(n): #列について
23                  S += a[i][j] * x[j]
24              X = (b[i] - S + a[i][i] * x[i]) / a[i][i]
25              if X != 0:
26                  if abs((X - x[i]) / X) < e:
27                      M += 1 #収束判定
28              x[i] = X
29          print(x)
30          count += 1
31
32      print("反復回数:", count)
33      for i in range(n):
34          print("V", str(i+1), ": ", x[i], sep = '')
35
36      print("解析解の推定値と実測値の誤差率 [V1〜V12](%): [", end = '')
37      for i in range(n):
38          print(error(x[i], ans[i]), end = '')
39          if i == n - 1:
40              print("]", sep = '')
41          else:
42              print(", ", end = '')
43      assX = assignment(a1, x)
44      print("算出した解析解を連立方程式に代入:", assX)
45      print("問題の右辺と代入して求めた右辺の誤差率 [V1〜V12](%): [", end = '')
46      for i in range(n):
47          print(error(assX[i], b1[i]), end = '')
48          if i == n - 1:
49              print("]", sep = '')
50          else:
51              print(", ", end = '')
52
53  #誤差率を求める関数
54  def error(est, act):
```

```
    return round(abs(100 * (est - act) / (act + 1)), 3)

#連立方程式に解析解を代入する関数
def assignment(a, x):
    n = len(a)
    ans = [0] * n
    for i in range(n):
        for j in range(n):
            ans[i] += a[i][j] * x[j]
    return(ans)

#ピボット選択
def pivot_selection(a, b):
    n = len(a)
    for i in range(n):
        for j in range(n):
            if a[j][i] > a[i][i] and i != j:
                tmp = a[i]
                a[i] = a[j]
                a[j] = tmp
                tmp = b[i]
                b[i] = b[j]
                b[j] = tmp

#main 関数
def main():
    a = [[-4, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
         [1, -4, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
         [0, 1, -4, 0, 0, 1, 0, 0, 0, 0, 0, 0],
         [1, 0, 0, -4, 1, 0, 1, 0, 0, 0, 0, 0],
         [0, 1, 0, 1, -4, 1, 0, 1, 0, 0, 0, 0],
         [0, 0, 1, 0, 1, -4, 0, 0, 1, 0, 0, 0],
         [0, 0, 0, 1, 0, 0, -4, 1, 0, 1, 0, 0],
         [0, 0, 0, 0, 1, 0, 1, -4, 1, 0, 1, 0],
         [0, 0, 0, 0, 0, 1, 0, 1, -4, 0, 0, 1],
         [0, 0, 0, 0, 0, 0, 1, 0, 0, -4, 1, 0],
         [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, -4, 1],
         [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, -4]]
    b = [-100, -100, -100, 0, 0, 0, 0, 0, 0, 100, 100, 100]
    ans = [39.785, 48.387, 39.785, 10.753, 13.978, 10.753, -10.753, -13.978,
        -10.753, -39.785, -48.387, -39.785]

```

```
96      gauss_seidel(a, b ,ans)
97
98  if __name__ == "__main__":
99      main()
```

ソースコード 12　演習課題 1.9-InverseMatrix Python プログラム

```
1   #Inverse Matrix
2   import numpy as np
3   import math
4   import copy
5
6   #逆行列による連立 1次方程式の解法
7   def InverseMatrix(a, b, ans):
8       n = len(a)
9       a1 = copy.deepcopy(a)
10      b1 = copy.deepcopy(b)
11      C = a
12      x = [0] * n
13
14      unit = np.identity(n) #単位行列の作成
15
16      for i in range(n):
17          for j in range(n):
18              C[i].append(unit[i, j])
19
20      count = 0
21
22      for k in range(n):
23          Ctmp = C[k][k]
24          for j in range(2 * n):
25              C[k][j] /= Ctmp #正規化
26          for i in range(n):
27              if i != k:
28                  Ctmp = C[i][k]
29                  for j in range(k, 2 * n):
30                      C[i][j] = C[i][j] - C[k][j] * Ctmp
31
32      for i in range(n):
33          x[i] = 0
34          for j in range(n):
35              x[i] = x[i] + C[i][n+j] * b[j]
36
```

```
37      for i in range(n):
38          print("V", str(i+1), ": ", x[i], sep = '')
39
40      print("解析解の推定値と実測値の誤差率 [V1〜V12](%): [", end = '')
41      for i in range(n):
42          print(error(x[i], ans[i]), end = '')
43          if i == n - 1:
44              print("]", sep = '')
45          else:
46              print(", ", end = '')
47      assX = assignment(a1, x)
48      print("算出した解析解を連立方程式に代入:", assX)
49      print("問題の右辺と代入して求めた右辺の誤差率 [V1〜V12](%): [", end = '')
50      for i in range(n):
51          print(error(assX[i], b1[i]), end = '')
52          if i == n - 1:
53              print("]", sep = '')
54          else:
55              print(", ", end = '')
56
57  #誤差率を求める関数
58  def error(est, act):
59      return round(abs(100 * (est - act) / (act + 1)), 3)
60
61  #連立方程式に解析解を代入する関数
62  def assignment(a, x):
63      n = len(a)
64      ans = [0] * n
65      for i in range(n):
66          for j in range(n):
67              ans[i] += a[i][j] * x[j]
68      return(ans)
69
70  #main 関数
71  def main():
72      a = [[-4, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
73           [1, -4, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
74           [0, 1, -4, 0, 0, 1, 0, 0, 0, 0, 0, 0],
75           [1, 0, 0, -4, 1, 0, 1, 0, 0, 0, 0, 0],
76           [0, 1, 0, 1, -4, 1, 0, 1, 0, 0, 0, 0],
77           [0, 0, 1, 0, 1, -4, 0, 0, 1, 0, 0, 0],
78           [0, 0, 0, 1, 0, 0, -4, 1, 0, 1, 0, 0],
```

```
79          [0, 0, 0, 0, 1, 0, 1, -4, 1, 0, 1, 0],
80          [0, 0, 0, 0, 0, 1, 0, 1, -4, 0, 0, 1],
81          [0, 0, 0, 0, 0, 0, 1, 0, 0, -4, 1, 0],
82          [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, -4, 1],
83          [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, -4],
84        ]
85    b = [-100, -100, -100, 0, 0, 0, 0, 0, 0, 100, 100, 100]
86    ans = [39.785, 48.387, 39.785, 10.753, 13.978, 10.753, -10.753, -13.978,
          -10.753, -39.785, -48.387, -39.785]
87    InverseMatrix(a, b, ans)
88
89 if __name__ == "__main__":
90    main()
```

1.9.d 実行結果:

ガウス・ザイデル法・逆行列を使用したプログラムの実行結果を以下に示す.
また, ガウス・ザイデル法の実行結果では解析解を小数点第四位で四捨五入している.

――ガウス・ザイデル法――

初期値: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

1: [25.5, 31.875, 33.219, 6.875, 10.188, 11.102, 2.219, 3.602, 3.926, -24.195, -29.898, -31.493]

2: [34.688, 44.524, 38.907, 11.774, 17.751, 15.146, -2.205, -2.606, -4.738, -33.026, -41.781, -36.63]

3: [39.075, 48.933, 41.02, 13.655, 18.782, 13.766, -5.494, -8.308, -7.793, -36.819, -45.439, -38.308]

4: [40.647, 50.112, 40.969, 13.484, 17.264, 12.61, -7.911, -10.97, -9.167, -38.337, -46.904, -39.018]

5: [40.899, 49.783, 40.598, 12.563, 15.997, 11.857, -9.186, -12.315, -9.869, -39.023, -47.589, -39.364]

6: [40.587, 49.295, 40.288, 11.85, 15.172, 11.398, -9.872, -13.039, -10.251, -39.365, -47.942, -39.548]

7: [40.286, 48.936, 40.084, 11.396, 14.673, 11.127, -10.252, -13.443, -10.466, -39.549, -48.135, -39.65]

8: [40.083, 48.71, 39.959, 11.126, 14.38, 10.968, -10.466, -13.672, -10.588, -39.65, -48.243, -39.708]

9: [39.959, 48.575, 39.886, 10.968, 14.21, 10.877, -10.588, -13.802, -10.658, -39.708, -48.304, -39.74]

10: [39.886, 48.495, 39.843, 10.877, 14.112, 10.824, -10.658, -13.877, -10.698, -39.74, -48.339, -39.759]

11: [39.843, 48.45, 39.819, 10.824, 14.055, 10.794, -10.698, -13.92, -10.721, -39.759, -48.359, -39.77]

12: [39.819, 48.423, 39.804, 10.794, 14.023, 10.776, -10.721, -13.945, -10.735, -39.77, -48.371, -39.776]

13: [39.804, 48.408, 39.796, 10.777, 14.004, 10.766, -10.735, -13.959, -10.742, -39.776, -48.378, -39.78]

14: [39.796, 48.399, 39.791, 10.766, 13.993, 10.761, -10.742, -13.967, -10.747, -39.78, -48.382, -39.782]

15: [39.791, 48.394, 39.789, 10.76, 13.987, 10.757, -10.747, -13.972, -10.749, -39.782, -48.384, -39.783]

16: [39.788, 48.391, 39.787, 10.757, 13.983, 10.755, -10.749, -13.975, -10.751, -39.783, -48.385, -39.784]

17: [39.787, 48.389, 39.786, 10.755, 13.981, 10.754, -10.751, -13.976, -10.752, -39.784, -48.386, -39.785]

18: [39.786, 48.388, 39.785, 10.754, 13.98, 10.753, -10.752, -13.978, -10.752, -39.785, -48.387, -39.785]
19: [39.785, 48.388, 39.785, 10.753, 13.979, 10.753, -10.752, -13.978, -10.752, -39.785, -48.387, -39.785]
反復回数: 19
V1: 39.785
V2: 48.388
V3: 39.785
V4: 10.753
V5: 13.979
V6: 10.753
V7: -10.752
V8: -13.978
V9: -10.752
V10: -39.785
V11: -48.387
V12: -39.785
解析解の推定値と実測値の誤差率 [V1〜V12]
(%): [0.0, 0.002, 0.0, 0.0, 0.007, 0.0, 0.01, 0.0, 0.01, 0.0, 0.0, 0.0]
算出した解析解を連立方程式に代入: [-99.99899999999998, -100.003, -99.99899999999998, -5.329070518200751e-15, 1.7763568394002505e-15, -5.329070518200751e-15, -0.001999999999995339, -7.105427357601002e-15, -0.001999999999995339, 100.00099999999998, 100.0, 100.00099999999998]
問題の右辺と代入して求めた右辺の誤差率 [V1〜V12](%): [0.001, 0.003, 0.001, 0.0, 0.0, 0.0, 0.2, 0.0, 0.2, 0.001, 0.0, 0.001]

——逆行列——
V1: 39.78494623655914
V2: 48.38709677419353
V3: 39.78494623655913
V4: 10.752688172043007
V5: 13.978494623655912
V6: 10.752688172043005
V7: -10.752688172043012
V8: -13.978494623655912
V9: -10.752688172043012
V10: -39.78494623655914
V11: -48.38709677419354
V12: -39.78494623655914
解析解の推定値と実測値の誤差率 [V1〜V12](%): [0.0, 0.0, 0.0, 0.003, 0.003, 0.003, 0.003, 0.004,

0.003, 0.0, 0.0, 0.0]
算出した解析解を連立方程式に代入: [-100.00000000000003, -99.99999999999994, -99.99999999999997, 1.7763568394002505e-14, -1.7763568394002505e-14, 1.0658141036401503e-14, 0.0, 0.0, -7.105427357601002e-15, 100.00000000000003, 99.99999999999997, 100.00000000000003]
問題の右辺と代入して求めた右辺の誤差率 [V1〜V12](%): [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

1.9.e 考察:

ガウス・ザイデル法では $\epsilon = 10^{-5}$ にしたところ, 反復回数は 19 回だった. 算出した解析解を式 (12) の左辺に代入し計算したら式 (12) の右辺とほとんど一致したためプログラムは正しいと言える.
逆行列を用いた計算では, 実行結果から解析解を式 (12) にの左辺に代入し計算したところ式 (12) の右辺とほとんど一致したため, プログラムは正しいといえる.
逆行列を用いた計算では, 本来誤差は生じないはずだが, わずかに発生した理由は, 計算する過程で割り切れない数字が出てきたためだと考えられる.

応用課題 1.10

1.10.a 問題:

$f(x) = \log_e(1+x), (x > -1)$ をマクローリン展開し, 近似次数の違いによるグラフを示せ.

1.10.b アルゴリズム:

マクローリン展開の一般形は課題 1-8 の式 (8) に示した通り.
式 (10) のマクローリン展開を有限個の項 ($k = n$) で打ち切った時の近似式は課題 1-8 の式 (11) に示した通り.

1.10.c プログラムリスト:

使用したマクローリン展開プログラムを以下に示す.

ソースコード 13　演習課題 1.10 Maclaurin expansion Python プログラム

```
1 #Approximating functions
2 import numpy as np
3 import math
```

```
import matplotlib.pyplot as plt
import japanize_matplotlib
import sympy as sym
from sympy.plotting import plot

def graph(x, y, n):
    y1 = np.log(x + 1)
    plt.xlim(-2, 3)
    plt.ylim(-3, 3)
    plt.grid()
    plt.plot(x, y1, label = "log$_{e}$(1+x)")
    for i in range(0, n):
        plt.plot(x, y[i], "--", label = str(i) + "次近似")
    plt.legend()
    plt.show()

#main 関数
def main():
    a = sym.symbols("a") #a を変数を表す文字として扱う
    x = np.linspace(-5, 5, 100)
    y = []
    fx = sym.log(a + 1)
    S = 0
    n = 6

    for i in range(n):
        S = 0
        for k in range(i+1):
            S += ((x ** k) / math.factorial(k)) * sym.diff(fx, a, k).subs(a, 0)
        y.append(S)

    graph(x, y, n)

if __name__ == "__main__":
    main()
```

1.10.d 実行結果:

マクローリン展開を使用したプログラムの実行結果を以下に示す.

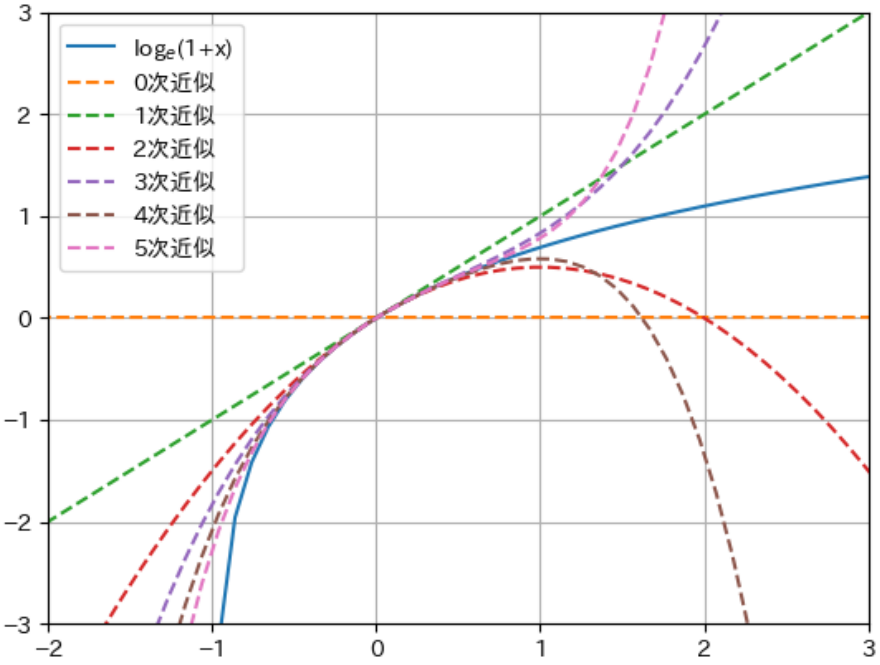

図 11　演習課題 1-10 グラフ

1.10.e 考察:

今回は n=0〜5, つまり 5 次近似まで求めた.
$\log(x+1)$ をマクローリン展開の式に当てはめると,

$$\log_e(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots + (-1)^{n-1}\frac{x^n}{n} \cdots \tag{14}$$

となる. 収束半径を計算してみると, ダランベールの公式を用いて,

$$r = \lim_{n\to\infty}\frac{a_n}{a_{n+1}} = \lim_{n\to\infty}\frac{\frac{1}{n}}{\frac{1}{(n+1)}} = \lim_{n\to\infty}\frac{n+1}{n} = 1 \tag{15}$$

となり, 収束範囲 r=1 ということがわかる.
$x = -1, 1$ での収束・発散を考えると, $x = -1$ のとき

$$log_e(x+1) = -1 - \frac{1}{2} - \frac{1}{3} - \frac{1}{4} - \cdots - \frac{1}{n} \cdots = -\infty \tag{16}$$

$x = 1$ のとき

$$\log_e(x+1) = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + (-1)^{n-1}\frac{1}{n} \cdot \cdot \tag{17}$$

となり, 収束することがわかる. 以上のことから,$\log_e(1+x)$ の収束半径は $-1 < x \leq 1$ であることがわかる. また, 出力したグラフを見ると,$-1 < x \leq 1$ の範囲しか収束していないことがわかる.

感想

ガウス・ザイデル法や SOR 法, ガウス・ジョルダン法, 逆行列を用いた計算, など連立 1 次方程式の解析解を求める計算をプログラムで書いたが, それぞれの良い点や悪い点が理解できた.
アルゴリズムを理解することは難しかったが, プログラムを書きながら原理を理解できてよかった.