数値計算　前期中間レポート
連立 1 次方程式の解法・関数近似

人間情報システム工学科 4 年 36 号
松山　京介

2021 年 5 月 30 日

課題

課題 1　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 $\boldsymbol{x} = [0.643, -0.071]^T$)

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

アルゴリズム

次の n 元の連立 1 次方程式：

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

の計算機を用いた反復法による解法を考える．この連立方程式を変形して，

$$\begin{cases} x_1^{(k)} &= \frac{1}{a_{11}}\{b_1 - (a_{12}x_2^{(k-1)} + a_{13}x_3^{(k-1)} + \cdots + a_{1n}x_n^{(k-1)})\} \\ x_2^{(k)} &= \frac{1}{a_{22}}\{b_2 - (a_{21}x_1^{(k)} + a_{23}x_3^{(k-1)} + \cdots + a_{2n}x_n^{(k-1)})\} \\ & \vdots \\ x_n^{(k)} &= \frac{1}{a_{nn}}\{b_n - (a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + \cdots + a_{nn-1}x_{n-1}^{(k)})\} \end{cases} \tag{2}$$

とする．ただし，x_i の右肩の添え字 (k) は k 回繰り返し計算後の x_i の近似値を表す．ガウス・ザイデル法は初期値 $\boldsymbol{x}^{(0)} = [x_2^{(0)}, x_3^{(0)}, \cdots, x_n^{(0)}]^T$ を与えて，2 式から k 回後の近似値 $x^{(k)} = [x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \cdots, x_n^{(k)}]^T$ を解析解に収束させようとするアルゴリズムである．(ここで右肩上の T はベクトルの転置を意味する)

ガウス・ザイデル法の収束するための十分条件は，

$$\sum_{j=1, i\neq j}^{n} |\frac{a_{ij}}{a_{ii}}| < 1 \tag{3}$$

または，

$$\sum_{j=1, i\neq j}^{n} |a_{ij}| < |a_{ii}| \tag{4}$$

ただし，$i = 1, 2, \cdots, n$ である．この式が成り立つならば初期値 $\boldsymbol{x}^{(0)}$ に関わらず解析解に収束することが知られている．しかし必ずしもこの条件が満足されなくても近似解が得られることもある．

行列表現すると，

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{pmatrix},$$

$$A_U = \begin{pmatrix} 0 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \cdots & \frac{a_{1n}}{a_{11}} \\ 0 & 0 & \frac{a_{23}}{a_{22}} & \cdots & \frac{a_{2n}}{a_{22}} \\ \vdots & & & \ddots & \vdots \\ 0 & & \cdots & 0 & \frac{a_{n-1n}}{a_{n-1n-1}} \\ 0 & & \cdots & & 0 \end{pmatrix}, A_L = \begin{pmatrix} 0 & 0 & \cdots & & 0 \\ \frac{a_{21}}{a_{22}} & 0 & \cdots & & 0 \\ \frac{a_{31}}{a_{33}} & \frac{a_{32}}{a_{33}} & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \frac{a_{n1}}{a_{nn}} & \cdots & \cdots & \frac{a_{nn-1}}{a_{nn}} & 0 \end{pmatrix}$$

より $(k-1)$ 回目と k 回目の解ベクトル x の関係は次のようになる．

$$\boldsymbol{x}^{(k)} = \boldsymbol{b} - A_U \boldsymbol{x}^{(k-1)} - A_L \boldsymbol{x}^{(k)} \tag{5}$$

よって，ガウス・ザイデル法のアルゴリズムは図 1 のようになる．

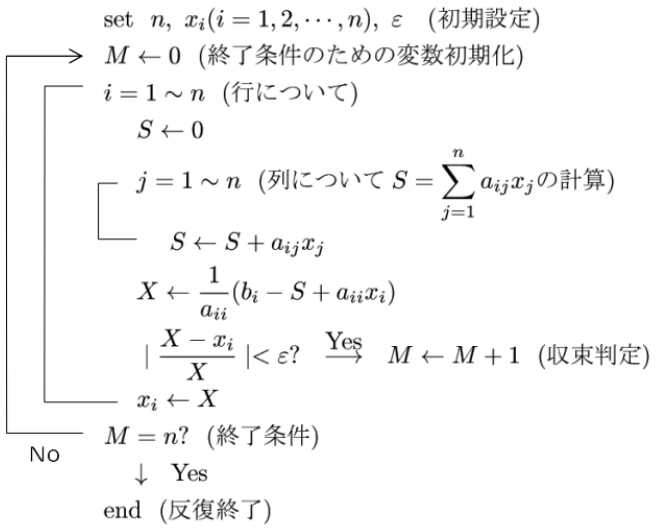

図 1: ガウス・ザイデル法のアルゴリズム

プログラム

ソースコード 1: 課題 1 ガウスザイデル法のプログラム

```
#初期設定
e=1e-5
n=2
x=[1,1]
a=[[3,-1],[2,4]]
b=[2,1]

while (1):
    M=0     #終了条件のための変数初期化
    print(x[0],x[1])
    for i in range (0,n):   #行についてのループ
        S=0
        for j in range(0,n):    #列についてのループ
            S+=a[i][j]*x[j]
        X=(b[i]-S+a[i][i]*x[i])/a[i][i]
        if abs((X-x[i])/X)<e:   #収束判定
            M+=1
        x[i]=X
    if(M==n):   #終了条件
```

```
20          break
21  print(x)     #計算結果の出力
```

実行結果

```
1, 1
1.0, -0.25
0.5833333333333334, -0.041666666666666685
0.6527777777777778, -0.0763888888888889
0.6412037037037037, -0.07060185185185186
0.6431327160493827, -0.07156635802469136
0.6428112139917695, -0.07140560699588477
0.6428647976680384, -0.0714323988340192
0.6428558670553269, -0.07142793352766347
0.6428573554907788, -0.07142867774538941
[0.6428571074182036, -0.07142855370910178]
```

考察

ガウス・ザイデル法のプログラムをソースコード 1 に示す．実行結果より，解析解にほぼ等しい結果が導出できていることが確認できる．ここで、実行結果の数値の推移を図 2 に示す．図 2 のプロットの重なりより、解析解の [0.643,-0.071] に収束していることが確認できる．

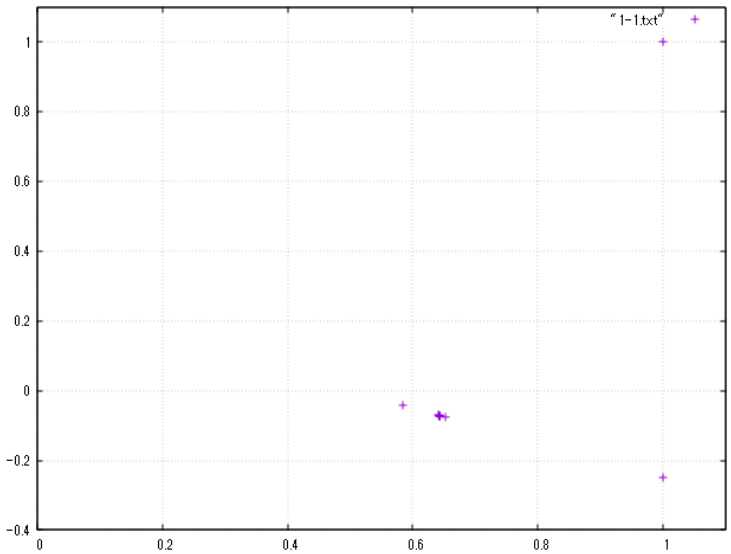

図 2: ガウス・ザイデル法における収束のプロット

課題 2　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操作が必要になる．(解析解は $\boldsymbol{x} = [-0.667, 1.444]^T$)

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

アルゴリズム

SOR 法のアルゴリズム

n 元の連立 1 次方程式をガウス・ザイデル法で解くには収束が遅く，加速が必要になる場合がある．その際，ガウス・ザイデル法の 5 式より

$$\begin{aligned} \boldsymbol{x}^{(k)} &= \boldsymbol{b} - A_U\boldsymbol{x}^{(k-1)} - A_L\boldsymbol{x}^{(k)} \\ &= \boldsymbol{x}^{(k-1)} + ((\boldsymbol{b} - A_U\boldsymbol{x}^{(k-1)} - A_L\boldsymbol{x}^{(k)}) - \boldsymbol{x}^{(k-1)}) \\ &= \boldsymbol{x}^{(k-1)} + \Delta\boldsymbol{x}^{(k)} \end{aligned}$$

とおく．ここで，$\Delta\boldsymbol{x}^{(k)}$ は $\boldsymbol{x}^{(k-1)}$ に対する $\boldsymbol{x}^{(k)}$ の変化分である．SOR 法は $\boldsymbol{x}^{(k)}$ が速く解に収束していくように加速パラメータ ω を代入して

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \omega((\boldsymbol{b} - A_U\boldsymbol{x}^{(k-1)} - A_L\boldsymbol{x}^{(k)}) - \boldsymbol{x}^{(k-1)}) \tag{6}$$

ただし，

$$0 < \omega < 2$$

とした方法である (図 3 参照)．収束の速さは加速パラメータ ω の選び方による．$\omega = 1$ のときはガウス・ザイデル法と同じである．最適な ω を定める一般的な理論はまだ確立されていない．

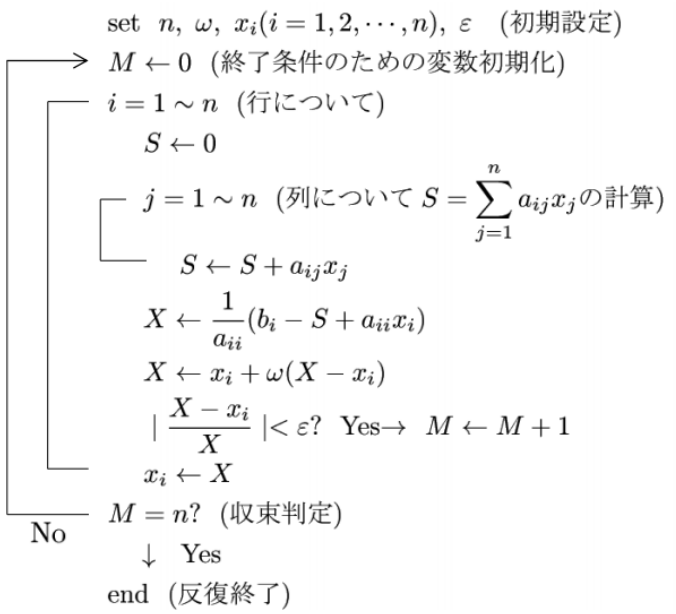

図 3: SOR 法のアルゴリズム

プログラム

ソースコード 2: 課題 2 ガウスザイデル法のプログラム

```
e=1e-5
n=2
x=[1,1]
#ピポット済み
a=[[5,3],[4,6]]
b=[1,6]

while (1):
    print(x[0],x[1])
    M=0  #終了条件のための変数初期化
    for i in range (0,n):  #行についてのループ
        S=0
        for j in range(0,n):  #列についてのループ
            S+=a[i][j]*x[j]
        X=(b[i]-S+a[i][i]*x[i])/a[i][i]
        if abs((X-x[i])/X)<e:  #収束判定
            M+=1
        x[i]=X
    if(M==n):  #終了条件
        break
print(x) #計算結果の出力
```

ソースコード 3: 課題 2 SOR 法のプログラム

```
e=1e-5
n=2
x=[1,1]
a=[[5,3],[4,6]]  #ピポット済み
b=[1,6]
omg=1.2  #加速パラメータ

while (1):
    M=0
    print(x[0],x[1])
    for i in range (0,n):
        S=0
        for j in range(0,n):
            S+=a[i][j]*x[j]
        X=(b[i]-S+a[i][i]*x[i])/a[i][i]
        X=x[i]+omg*(X-x[i])
        if abs((X-x[i])/X)<e:
            M+=1
        x[i]=X

    if(M==n):
        break
print(x)
```

実行結果

ガウス・ザイデル法

```
1 1
-0.4 1.2666666666666666
-0.5599999999999999 1.3733333333333333
-0.624 1.4160000000000001
-0.6496000000000001 1.4330666666666667
-0.65984 1.4398933333333332
-0.663936 1.4426240000000001
-0.6655744 1.4437162666666667
-0.66622976 1.4441531733333335
-0.6664919040000001 1.4443279359999999
-0.6665967615999999 1.4443978410666667
-0.6666387046400001 1.4444258030933332
-0.666655481856 1.444436987904
-0.6666621927424 1.4444414618282666
[-0.66666487709696, 1.4444432513979732]
```

SOR 法

```
1 1
```

```
-0.6799999999999999 1.5439999999999996
-0.7356799999999997 1.4797439999999997
-0.6782796799999997 1.4466749439999997
-0.6659500236799999 1.443425030144
-0.66607601696768 1.4441758075453441
-0.6665913780391117 1.4444379409222206
-0.6666770418561764 1.4444540453004968
[-0.6666715042451224, 1.4444463943359986]
```

考察

ガウス・ザイデル法のプログラムをソースコード 2 に，SOR 法のプログラムをソースコード 3 に示す．実行結果より，解析解にほぼ等しい結果が導出できていることが確認できる．ただし，手動でピボット操作をすることとする．

以下にガウスザイデル法のときと, SOR 法のときの収束のプロットを示す．図 4, 図 5 のそれぞれのプロットの重なりより、どちらも解析解の [-0.667,1.444] に収束していることが確認できる．

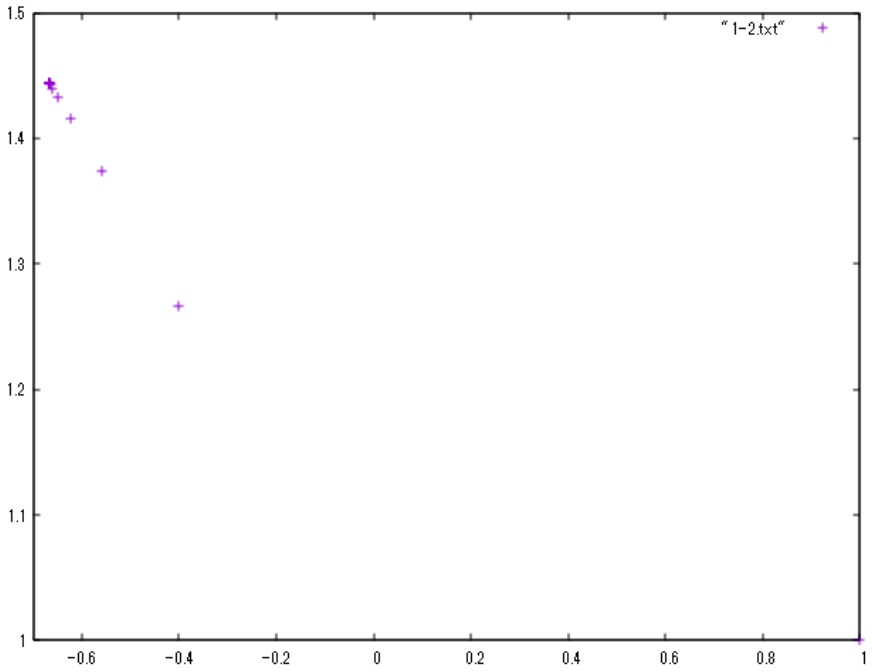

図 4: ガウスザイデル法における収束のプロット

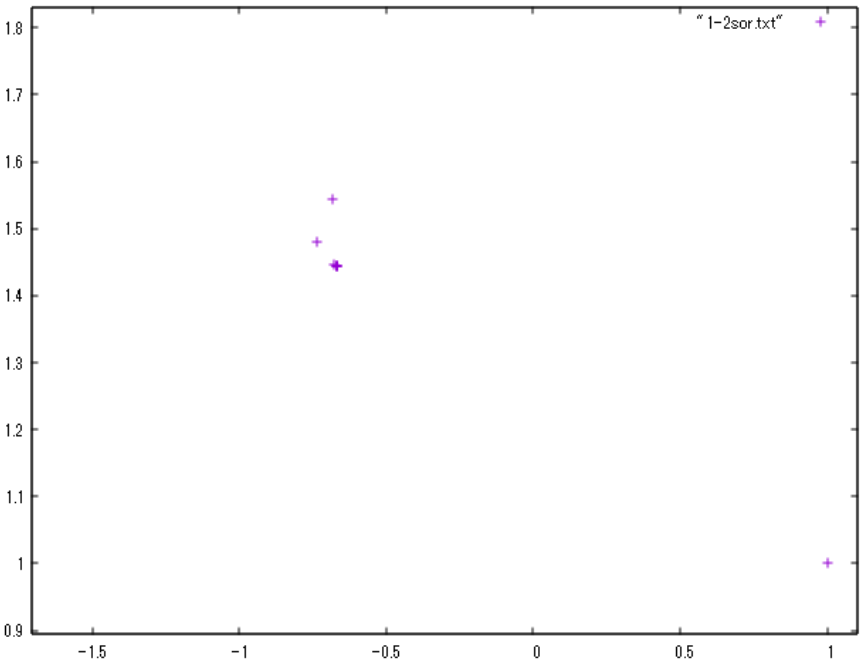

図 5: SOR 法における収束のプロット

課題 3　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 $\boldsymbol{x} = [-1, -1, 4]^T$)

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

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 に，SOR 法のアルゴリズムは課題 2 に同じ．

プログラム

ソースコード 4: 課題 3 ガウスザイデル法のプログラム

```
import math
e=1e-5
n=3
x=[1,1,1]
a=[[2,1,1],[0,2,1],[1,1,1]]
b=[1,2,2]

#ガウスザイデル法を用いた計算
while (1):
    print(x[0],x[1],x[2])
    M=0
    for i in range (0,n):
        S=0
        for j in range(0,n):
            S+=a[i][j]*x[j]
        X=(b[i]-S+(a[i][i]*x[i]))/a[i][i]
        try:
            if abs((X-x[i])/X)<e:
                M+=1
        except ZeroDivisionError:
            M+=0
        x[i]=X
    if(M==n):
        break
print(x)  #計算結果の出力
```

ソースコード 5: 課題 3 SOR 法のプログラム

```
e=1e-5
n=3
```

```
x=[1,1,1]
a=[[2,1,1],[0,2,1],[1,1,1]]
b=[1,2,2]
omg=1.2  #加速パラメータ

#SOR 法を用いた計算
while (1):
    print(x[0],x[1],x[2])
    M=0
    for i in range (0,n):
        S=0
        S2=0
        for j in range(0,n):
            S+=a[i][j]*x[j]
            S2+=a[i][j]
        X=(b[i]-S+a[i][i]*x[i])/a[i][i]
        X=x[i]+omg*(X-x[i])
        if abs((X-x[i])/X)<e:
            M+=1
        x[i]=X

    if(M==n):
        break
print(x)  #計算結果の出力
```

実行結果

ガウス・ザイデル法

```
1 1 1
-0.5 0.5 2.0
-0.75 0.0 2.75
-0.875 -0.375 3.25
-0.9375 -0.625 3.5625
-0.96875 -0.78125 3.75
-0.984375 -0.875 3.859375
-0.9921875 -0.9296875 3.921875
-0.99609375 -0.9609375 3.95703125
-0.998046875 -0.978515625 3.9765625
-0.9990234375 -0.98828125 3.9873046875
-0.99951171875 -0.99365234375 3.9931640625
-0.999755859375 -0.99658203125 3.996337890625
-0.9998779296875 -0.9981689453125 3.998046875
-0.99993896484375 -0.9990234375 3.99896240234375
-0.999969482421875 -0.999481201171875 3.99945068359375
-0.9999847412109375 -0.999725341796875 3.9997100830078125
-0.9999923706054688 -0.9998550415039062 3.999847412109375
-0.9999961853027344 -0.9999237060546875 3.999919891357422
```

```
-0.9999980926513672 -0.9999599456787109 3.999958038330078
-0.9999990463256836 -0.9999790191650391 3.9999780654907227
-0.9999995231628418 -0.9999890327453613 3.999988555908203
[-0.9999997615814209, -0.9999942779541016, 3.9999940395355225]
```

```
SOR 法
1 1 1
-0.7999999999999998 0.4 2.6799999999999997
-1.0879999999999999 -0.48799999999999966 3.7551999999999994
-1.14272 -0.9555199999999998 4.166848
-1.0982528 -1.1090048 4.215339520000001
-1.0441502720000002 -1.1074027520000003 4.1387957248
-1.0100057292799998 -1.0617968844799996 4.058403991551999
-0.9959631183871996 -1.0226830180351993 4.010694565396479
-0.993614304739328 -1.0018801356308475 3.9924544153649153
-0.9956217068925749 -0.9950966220927797 3.9903711117094427
-0.9980403523914829 -0.9952033426071096 3.9938182116564227
-0.9995608509512914 -0.9972502584724316 3.997409688977183
-1.0001834881125926 -0.9989957616918235 3.999533161969863
-1.000285742544305 -0.999920744843553 4.000341152471457
-1.0001950960678816 -1.000220542514164 4.0004305358041625
-1.0000869767604228 -1.0002142129796647 4.000275320527272
-1.0000192691764795 -1.0001223497204301 4.000114878570837
-0.9999916634749484 -1.0000444571984164 4.000020369093869
-0.9999872144422821 -1.0000033300166384 3.999984579531931
-0.999991306820719 -0.9999900817158308 3.999980750337474
[-0.999996139808842, -0.999990433859318, 3.999987738334297]
```

考察

ガウス・ザイデル法のプログラムをソースコード 4 に，SOR 法のプログラムをソースコード 5 に示す．実行結果より，どちらの解法も解析解にほぼ等しい結果が導出できていることが確認できる．

以下に前課題と同じように, ガウスザイデル法のときと,SOR 法のときの収束のプロットを示す．図 6, 図 7 のそれぞれのプロットの重なりより、どちらも解析解の [-1,-1,4] に収束していることが確認できる．

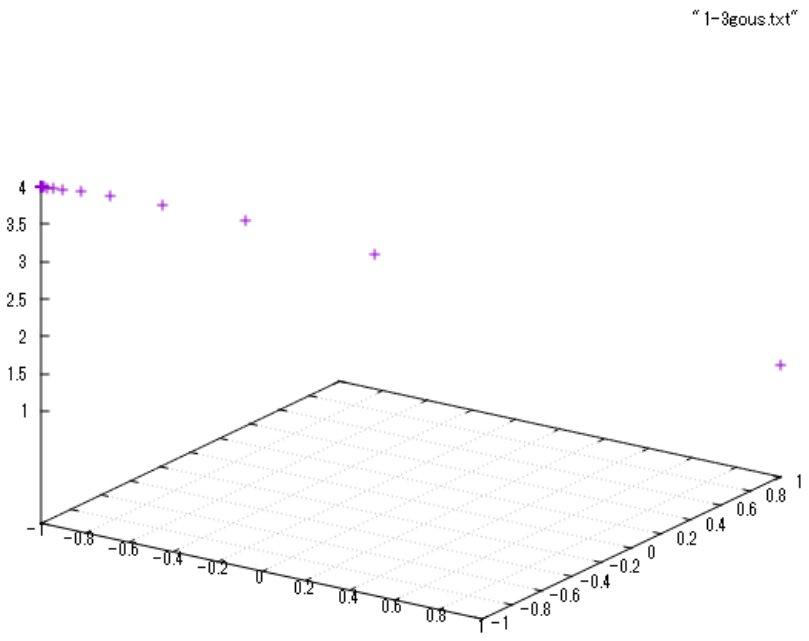

図 6: ガウスザイデル法における収束のプロット

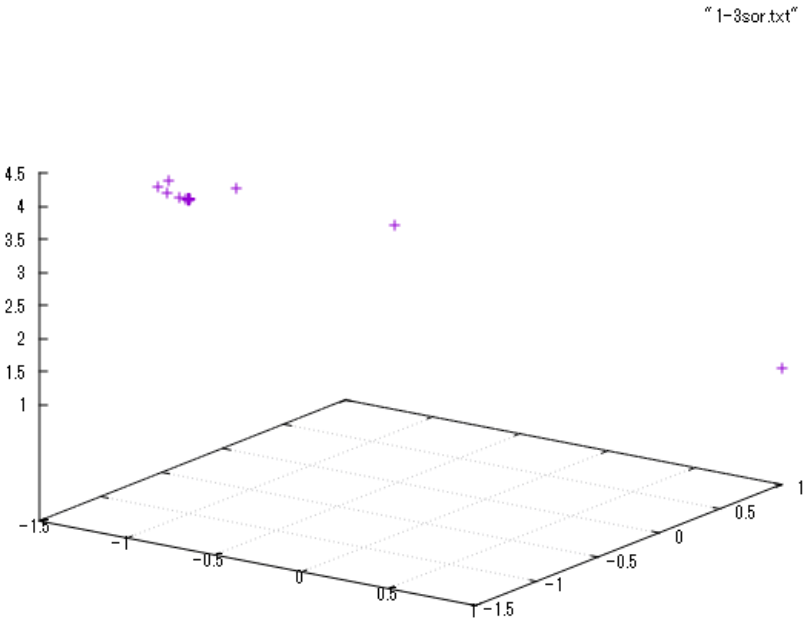

図 7: SOR 法における収束のプロット

課題 4　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 $\boldsymbol{x} = [= 1, -1, 4]^T$)

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

アルゴリズム

次の n 元の連立 1 次方程式：

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

は行列で表すと，

$$A\boldsymbol{x} = \boldsymbol{b}$$

ただし，

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

となる．$n \times (n+1)$ の拡大行列 $(A, \boldsymbol{b})$ を最初の $n \times n$ 行列の部分が対角行列，または単位行列になるまで行に関する基本操作を行って，最後の $n+1$ 列に求める解を得る方法がガウス・ジョルダン法である．
図 8 にアルゴリズムを示す．

set n, C (拡大行列 :$(A, \boldsymbol{b})$)
- $k = 1 \sim n$
  - $i = 1 \sim n$ $(i \neq k)$
    - $c' \leftarrow \dfrac{c_{ik}}{c_{kk}}$
    - $j = k \sim n+1$
      - $c_{ij} \leftarrow c_{ij} - c_{kj} \times c'$
- $i = 1 \sim n$
  - $x_i \leftarrow \dfrac{c_{i\,n+1}}{c_{ii}}$ (解ベクトル)

図 8: ガウス・ジョルダン法のアルゴリズム

プログラム

ソースコード 6: 課題 4 ガウス・ジョルダン法のプログラム

```
n=3  #次元
C=[[2,1,1,1],[0,2,1,2],[1,1,1,2]]       #拡大行列
x=[1,1,1]

#ガウスジョルダン法を用いた計算
for k in range(0,n):
    for i in range(0,n):
        if(i==k):
            continue
        cdash=C[i][k]/C[k][k]
        for j in range(k,n+1):
            C[i][j]=C[i][j]-C[k][j]*cdash
for i in range(0,n):
    x[i]=C[i][n]/C[i][i]

print(x)    #計算結果の出力
```

実行結果

```
[-1.0, -1.0, 4.0]
```

考察

まず，解析解の導出を以下に示す．

$$\begin{aligned} 2x_1 + x_2 + x_3 &= 1 \\ -)\quad x_1 + x_2 + x_3 &= 2 \\ \hline x_1 &= -1 \end{aligned}$$

$$\begin{aligned} 2x_2 + x_3 &= 2 \\ -)\quad x_2 + x_3 &= 3 \\ \hline \therefore x_2 &= -1, x_3 = 4 \end{aligned}$$

以上より，解析解は $\boldsymbol{x} = [-1, -1, 4]^T$.

ガウスジョルダン法のプログラムをソースコード 6 に示す．

実行結果より、誤差なく解析解に等しい結果を導出することができている．

課題 5　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．

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

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 9 に示す．この正規化とは，各 k 段階に入る前に拡大行列 C の k 行目の各要素を c_{kk} で割ることである．このアルゴリズムの対角要素 c_{kk} はピボットというが，このピボットで割るためここに 0 がこないよう連立方程式の行を入れ替える必要がある．また，$c_{kk} = 0$ でなくてもその絶対値が他の要素に比べて非常に小さい場合は大きな誤差を生じる場合があるため，対角要素 c_{kk} に絶対値最大のものがくるように行や列を入れ替える．このことをピボット選択という．

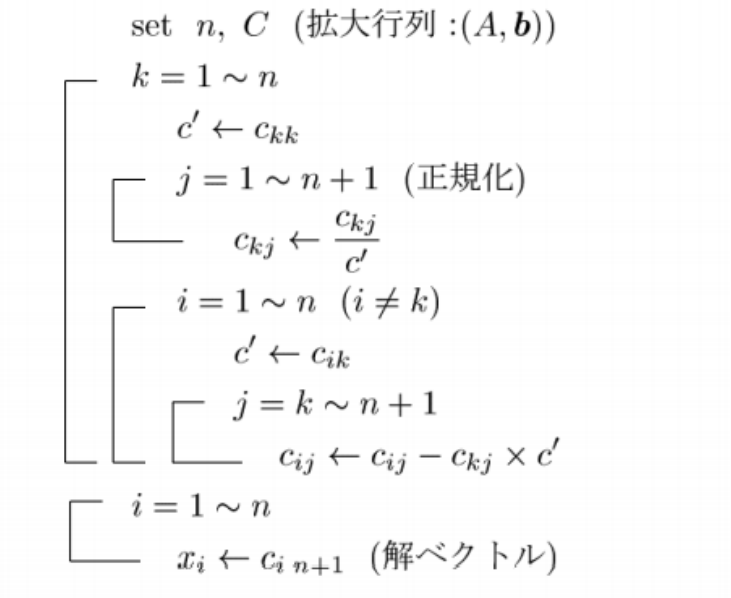

図 9: 正規化を行ったガウス・ジョルダン法のアルゴリズム

プログラム

ソースコード 7: 課題 5 正規化を行ったガウス・ジョルダン法のプログラム

```
1 n=3
2 C=[[2,1,1,1],[0,2,1,2],[1,1,1,2]]
3 x=[1,1,1]
4
5 for k in range(0,n):
6     cdash=C[k][k]
7     for j in range(0,n+1):
```

```
        C[k][j]=C[k][j]/cdash
    for i in range(0,n):
        if(i==k):
            continue
        cdash=C[i][k]/C[k][k]
        for j in range(k,n+1):
            C[i][j]=C[i][j]-C[k][j]*cdash
for i in range(0,n):
    x[i]=C[i][n]/C[i][i]

print(x)
```

実行結果

```
[-1.0, -1.0, 4.0]
```

考察

正規化を行ったガウス・ジョルダン法のプログラムをソースコード 7 に示す．実行結果より，正規化を行った場合も誤差なく解析解と等しい結果が出力できていることがわかる．

課題 6　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 $\boldsymbol{x} = [1, 2, 3, 4, 5]^T$)

$$\begin{cases} 2x_1 + 3x_2 + 4x_3 + 5x_4 + 6x_5 = 70 \\ 3x_1 + 4x_2 + 5x_3 + 6x_4 + 2x_5 = 60 \\ 4x_1 + 5x_2 + 6x_3 + 2x_4 + 3x_5 = 55 \\ 5x_1 + 6x_2 + 2x_3 + 3x_4 + 4x_5 = 55 \\ 6x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 = 60 \end{cases} \tag{7}$$

アルゴリズム

ガウス・ジョルダン法のアルゴリズムは課題 4 に同じ．

プログラム

ソースコード 8: 課題 6 ガウスジョルダン法を用いたプログラム

```
n=5
C=[[6,2,3,4,5,60],[3,4,5,6,2,60],[4,5,6,2,3,55],[5,6,2,3,4,55],[2,3,4,5,6,70]]
x=[1,1,1,1,1]

for k in range(0,n):
    for i in range(0,n):
        if(i==k):
            continue
        cdash=C[i][k]/C[k][k]
```

```
        for j in range(k,n+1):
            C[i][j]=C[i][j]-C[k][j]*cdash
for i in range(0,n):
    x[i]=C[i][n]/C[i][i]

print(x)
```

実行結果

```
[0.9999999999999988, 2.000000000000027, 2.9999999999999982, 4.000000000000001, 5.0]
```

考察

ガウス・ジョルダン法のプログラムをソースコード 8 に示す．実行結果より, 若干の誤差はあるがほぼ解析解に等しい結果が出力できていることが確認できる．

課題 7　次の連立方程式を逆行列を使って解け．(解析解は $\boldsymbol{x} = [-3, 4, -1]^T$)

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

アルゴリズム

n 元連立 1 次方程式：

$$A\boldsymbol{x} = \boldsymbol{b} \tag{8}$$

の行列 A が正則行列であればその解は逆行列を用いて

$$\boldsymbol{x} = A^{-1}\boldsymbol{b} \tag{9}$$

となる．いま求めたい逆行列を $A^{-1} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]$，ただし $\boldsymbol{x}_i \in R^n$ とおくと，

$$A[\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n] = I, \quad I : \text{単位行列} \tag{10}$$

となり，

$$A\boldsymbol{x}_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, A\boldsymbol{x}_2 = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \cdots, A\boldsymbol{x}_n = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{11}$$

を満たす $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$ を求める問題となるため前述のガウス・ジョルダンの方法が適用できる．

ガウス・ジョルダンの方法は行列 A の行と列に対して n 回操作をして単位行列：

$$U_n U_{n-1} \cdots U_1 A = I$$

にすることなのでこの n 個の行列 $U_n, U_{n-1}, \cdots, U_1$ をかけることと同じであるから，10 式の両辺に左から $U_n, U_{n-1}, \cdots, U_1$ をかけると

$$U_n U_{n-1} \cdots U_1 A[\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n] = I[\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n] = U_n U_{n-1} \cdots U_1 I \tag{12}$$

となり，右辺の値 $U_n U_{n-1} \cdots U_1$ が求めたい逆行列の値 A^{-1} となる．

この逆行列を用いた方法 (図 10 参照) は，行列 A が正則で，ピボット c_{kk} が 0 出ない場合に適用できる．ガウス・ジョルダン法と同様に対角要素 c_{kk} に絶対値最大のものがくるように行や列を入れ替える操作 (ピボット選択) が必要になる場合がある．

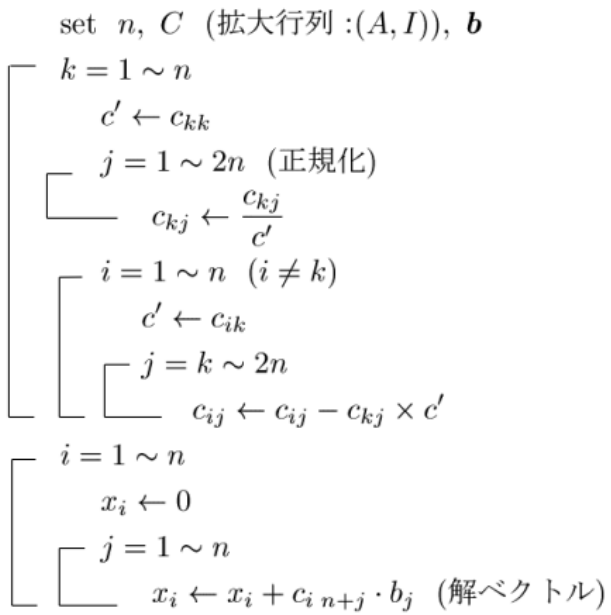

図 10: 逆行列を用いた解法アルゴリズム

プログラム

ソースコード 9: 課題 7 逆行列のプログラム

```
n=3
C=[[1,2,1,1,0,0],[4,5,6,0,1,0],[7,8,9,0,0,1]]
b=[4,2,2]
x=[1,1,1]

for k in range(0,n):
    cdash=C[k][k]
    for j in range(0,2*n):
        C[k][j]=C[k][j]/cdash
    for i in range(0,n):
        if i==k:
            continue
```

```
        cdash=C[i][k]
        for j in range(k,2*n):
            C[i][j]=C[i][j]-C[k][j]*cdash
for i in range(0,n):
    x[i]=0
    for j in range(0,n):
        x[i]=x[i]+C[i][n+j]*b[j]
print(x)
```

実行結果

```
[-3.0, 4.0, -1.0]
```

考察

逆行列のプログラムをソースコード 9 に示す．実行結果より，誤差なく解析解に等しい結果が導出できていることが確認できる．

課題 8　$f(x) = e^x$ 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．

アルゴリズム

関数 $f(x)$ が点 $x = x_0$ で無限回微分可能ならば，x_0 の近傍で

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{f^{(2)}(x_0)}{2!}(x - x_0)^2 + \frac{f^{(3)}}{3!}(x - x_0)^3 + \cdots \\ &= \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \end{aligned} \tag{13}$$

と無限級数で表現できる．これをテーラー級数展開という．特に $x_0 = 0$ のとき

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!}(x)^k \tag{14}$$

となり，マクローリン展開という．(13) 式を有限個の項 $(k = n)$ 次で打ち切って近似した式

$$f(x) \approx \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \tag{15}$$

を関数のテーラー級数展開近似という．

以下に与式 $f(x) = e^x$ のマクローリン展開の導出を示す.

$$
\begin{aligned}
f(x) &= \sum_{n=0}^{\infty} f^{(n)} \frac{x^n}{n!} \\
&= f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots + \frac{f^{(n)}(0)}{n!}x^n + \cdots \\
e^x &= 1 + 1 \cdot x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots \\
&= 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots \\
&= \sum_{n=0}^{\infty} \frac{1}{n!}x^n
\end{aligned}
$$

プログラム

ソースコード 10: 課題 8 マクローリン展開のプログラム

```
import numpy as np
import math
import sympy
from sympy.plotting import plot
sympy.init_printing(use_unicode=True)

import matplotlib.pyplot as plt
import seaborn as sns

def mac(fo,max): #マクローリン展開
  f = fx
  app = np.sum([(sympy.diff(f, x, k).subs(x, 0) / (math.factorial(k))) * ((x)**k)
    for k in range(max)])
  return app


x = sympy.symbols("x")
fx = math.e**x #元の関数 f(x)


plt.rcParams['figure.figsize'] = 10, 8

f1 = mac(fx, 2)
f2 = mac(fx, 4)
f3 = mac(fx, 6)
f4 = mac(fx, 8)
f5 = mac(fx, 10)
f6 = mac(fx, 12)
f7 = mac(fx, 14)
f8 = mac(fx, 16)
```

```
30
31 #グラフの出力
32 mcgra = sympy.plot(f1,f2,f3,f4,f5,f6,f7,f8,fx,legend=True, show=False)
33 colors = ["orange", "red", "purple", "blue", "green","deepskyblue","fuchsia","
       lawngreen"]
34
35 for i in range(9):
36   if i == 8:
37     mcgra[i].line_color="black"
38     mcgra[i].label='e^x'
39   else:
40     mcgra[i].line_color=colors[i]
41     label_i = i+1
42     mcgra[i].label='f%d'%label_i
43
44 mcgra.show()
```

実行結果

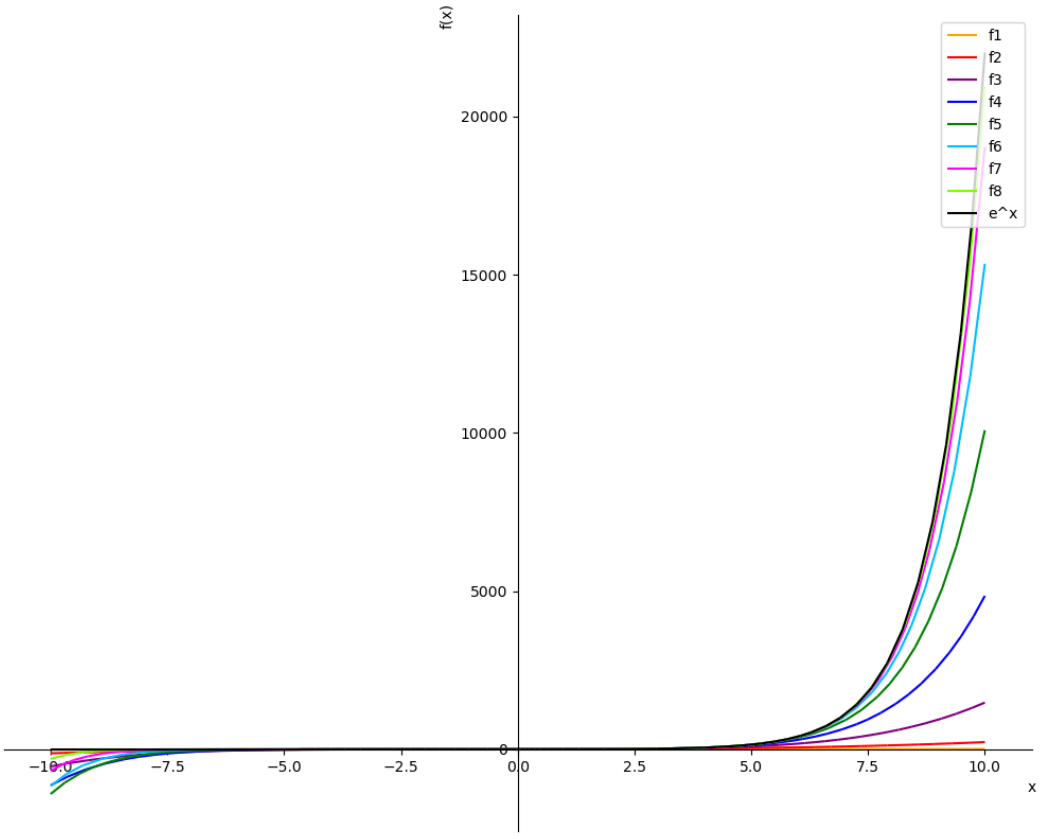

図 11: 課題 8 ソースコード 10 の実行結果

考察

マクローリン展開した関数の近似次数の違いをグラフにプロットするプログラムをソースコード 10 に示す．実行結果の図 11 より，近似次数を増やすほど，もとの関数の曲線に近づいていくことが確認できる．

課題 9　図 12 のような直流電圧 $E_1 = 100[\mathrm{V}]$，$E_2 = 100[\mathrm{V}]$，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 $V = [39.785, 48.387, 39.785, 10.753, 13.978, 10.753,$
$-10.753, -13.978, -10.753, -39.785, -48.387, -39.785]^T$**)**

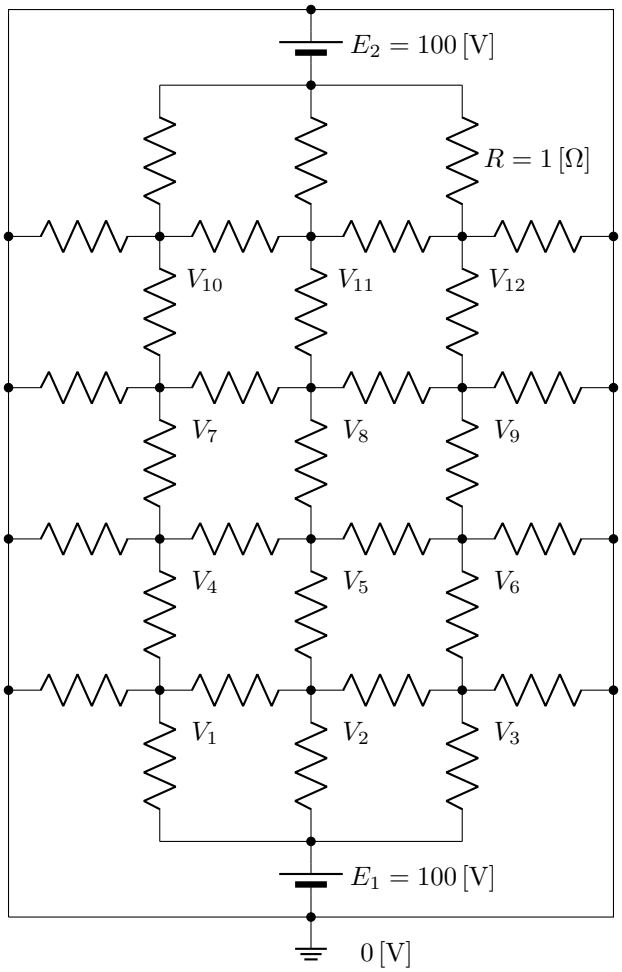

図 12: 課題 9 の回路図

アルゴリズム

ガウス・ジョルダン法のアルゴリズムは課題 4 に，掃き出し法 (逆行列) のアルゴリズムは課題 7 に同じ．

以下に課題 9 の回路図における連立 1 次方程式の導出を示す．

キルヒホッフの第 1 法則より，任意の節点 V_0 について

$$I_3 + I_4 = I_1 + I_2$$

よって，電圧と抵抗の関係から

$$\frac{V_W - V_0}{R} + \frac{V_S - V_0}{R} = \frac{V_0 - V_E}{R} + \frac{V_0 - V_N}{R}$$

となり，整理すると

$$V_E + V_N + V_W + V_S - 4V_0 = 0$$

となる．この関係を図 12 の各節点で適用する．

$E_1 = 100\mathrm{V}, E_2 = 100\mathrm{V}, R = 1\Omega$ より

$$\begin{cases}
V_2 + V_4 + 0 + 100 - 4V_1 = 0 \\
V_3 + V_5 + V_1 + 100 - 4V_2 = 0 \\
0 + V_6 + V_2 + 100 - 4V_3 = 0 \\
V_5 + V_7 + 0 + V_1 - 4V_4 = 0 \\
V_6 + V_8 + V_4 + V_2 - 4V_5 = 0 \\
0 + V_9 + V_5 + V_3 - 4V_6 = 0 \\
V_8 + V_{10} + 0 + V_4 - 4V_7 = 0 \\
V_9 + V_{11} + V_7 + V_5 - 4V_8 = 0 \\
0 + V_{12} + V_8 + V_6 - 4V_9 = 0 \\
V_{11} - 100 + 0 + V_7 - 4V_{10} = 0 \\
V_{12} - 100 + V_{10} + V_8 - 4V_{11} = 0 \\
0 - 100 + V_{11} + V_9 - 4V_{12} = 0
\end{cases}$$

プログラム

ソースコード 11: 課題 9 のプログラム

```
n=12
e=1e-10

az=[[-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]]

ag=[[-4,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],
    [1,-4,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0],
    [0,1,-4,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
    [1,0,0,-4,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0],
```

```
    [0,1,0,1,-4,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],
    [0,0,1,0,1,-4,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0],
    [0,0,0,1,0,0,-4,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,1,0,1,-4,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0],
    [0,0,0,0,0,1,0,1,-4,0,0,1,0,0,0,0,0,0,0,1,0,0,0],
    [0,0,0,0,0,0,1,0,0,-4,1,0,0,0,0,0,0,0,0,0,1,0,0],
    [0,0,0,0,0,0,0,1,0,1,-4,1,0,0,0,0,0,0,0,0,0,1,0],
    [0,0,0,0,0,0,0,0,1,0,1,-4,0,0,0,0,0,0,0,0,0,0,1]]

b=[-100,-100,-100,0,0,0,0,0,0,100,100,100]
xz=[0,0,0,0,0,0,0,0,0,0,0,0]
xg=[0,0,0,0,0,0,0,0,0,0,0,0]

#ガウスザイデル法を計算する関数
def gous(az,b,xz):
    while (1):
        M=0
        for i in range (0,n):
            S=0
            for j in range(0,n):
                S+=az[i][j]*xz[j]
            X=(b[i]-S+az[i][i]*xz[i])/az[i][i]
            if abs((X-xz[i])/X)<e:
                M+=1
            xz[i]=X
        if(M==n):
            break

#逆行列を計算する関数
def gyaku(ag,b,xg):
    for k in range(0,n):
        cdash=ag[k][k]
        for j in range(0,2*n):
            ag[k][j]=ag[k][j]/cdash
        for i in range(0,n):
            if i==k:
                continue
            cdash=ag[i][k]
            for j in range(k,2*n):
                ag[i][j]=ag[i][j]-ag[k][j]*cdash
    for i in range(0,n):
        xg[i]=0
        for j in range(0,n):
            xg[i]=xg[i]+ag[i][n+j]*b[j]

#出力
```

```
67 gous(az,b,xz)
68 gyaku(ag,b,xg)
69 print('ガウスザイデル法\n'+str(xz))
70 print('逆行列\n'+str(xg))
```

実行結果

```
ガウスザイデル法
[ 39.78501716734044,   48.38717281635068,   39.78498699749148,   10.752775173615243,
  13.978587894684814,  10.75273816817853,  -10.752622219462177, -13.978423918440258,
 -10.752650271876913, -39.78491533724592,  -48.387063648233024, -39.78492848002749]
逆行列
[ 39.78494623655914,   48.38709677419353,   39.78494623655913,   10.752688172043007,
  13.978494623655912,  10.752688172043005, -10.752688172043012, -13.978494623655912,
 -10.752688172043012, -39.78494623655914,  -48.38709677419354,  -39.78494623655914]
```

考察

プログラムをソースコード 9 に示す．実行結果より，どちらの解法も解にほぼ等しい結果が導出できていることが確認できる．解に僅かながら誤差が生じているのは，求める解の精度の違いによるものだと考えられる．

感想

1-9 まではなんとか完成させることができたが,1-10 を完成させることができなかった．自分の実力不足をひしひしと感じたのでもっと精進したい．また、LaTeX を用いて初めてレポートを作成したが非常に骨の折れる作業だった．しかしながら, 非常に綺麗なレポートを作成できたと思う．今後も積極的に使っていきたい．