数値計算-後期期末レポート

HI4 45 番 山口惺司

2025 年 1 月 30 日

課題 4.1

4.1.a 問題:

偏微分方程式 1(ラプラス方程式)

図 1 のような三角形の境界を S とし, 三角形で囲まれた領域を R とする問題領域 $G = R \bigcap S$ がある. 領域 R 内はラプラス方程式

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \tag{1}$$

を満たしており, 境界条件は以下である.

$$\begin{cases} \phi(x,y) = 20x, (x,y) \in S_0 \\ \phi(x,y) = 20y, (x,y) \in S_1 \\ \phi(x,y) = 160, (x,y) \in S_2 \end{cases} \tag{2}$$

刻み幅を 2 として領域 R 内で定義されたラプラス方程式の解$\phi_{2.2}, \phi_{2.4}, \phi_{4.2}$(図 1) を求めよ.(解 : $\phi_{2.2} = 80, \phi_{2.4} = 120, \phi_{4.2} = 120$)

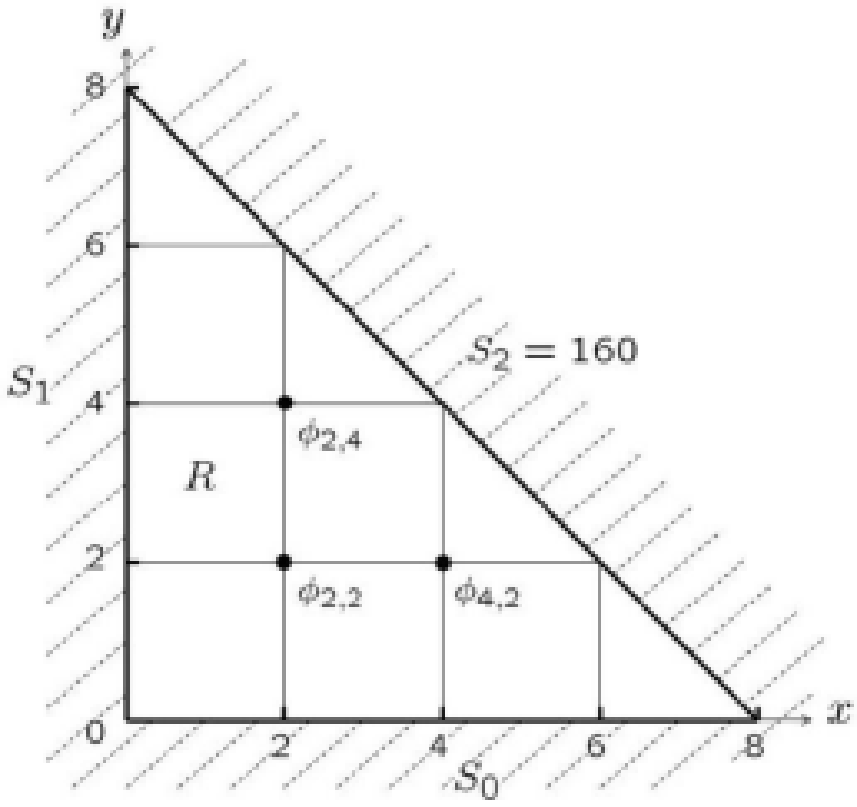

図 1　偏微分方程式 1

4.1.b アルゴリズム:

図 1 と境界条件より以下のような行列ができる.

$$\begin{pmatrix} -4 & 1 & 1 \\ 1 & -4 & 0 \\ 1 & 0 & -4 \end{pmatrix} \begin{pmatrix} \phi_{22} \\ \phi_{42} \\ \phi_{24} \end{pmatrix} = \begin{pmatrix} -80 \\ -400 \\ -400 \end{pmatrix}$$

これを逆行列法を用いて解く. 逆行列法のアルゴリズムは図 2 に示す.

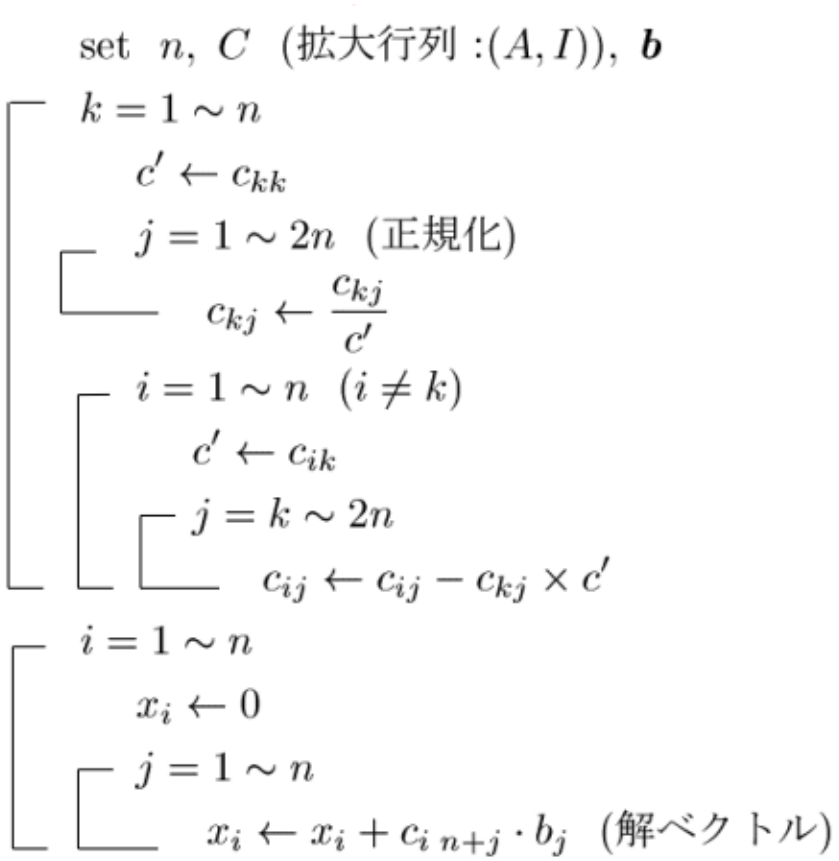

図 2　逆行列を用いた解法アルゴリズム

4.1.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. Python プログラム

```
1 #4-1
2 import numpy as np
3 import math
```

```
import sympy as sym
import matplotlib.pyplot as plt
import copy

#逆行列による連立1次方程式の解法
def InverseMatrix(a, b, x):
    n = len(a)
    a1 = copy.deepcopy(a)
    C = a1

    unit = np.identity(n) #単位行列の作成

    for i in range(n): #拡大行列の作成
        for j in range(n):
            C[i].append(unit[i, j])

    for k in range(n):
        Ctmp = C[k][k]
        for j in range(2 * n):
            C[k][j] /= Ctmp #正規化
        for i in range(n):
            if i != k:
                Ctmp = C[i][k]
                for j in range(k, 2 * n):
                    C[i][j] = C[i][j] - C[k][j] * Ctmp

    for i in range(n):
        x[i] = 0
        for j in range(n):
            x[i] = round(x[i] + C[i][n+j] * b[j], 2)

#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)

def main():
    a = [[-4, 1, 1],[1, -4, 0],[1, 0, -4]]
```

```
    b = [-80, -400, -400]
    x = [0] * len(a)
    InverseMatrix(a, b, x)

    print("解:", x)
    print("解を式に代入:", assignment(a, x))

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

4.1.d 実行結果:

実行結果を図 3 に示す.

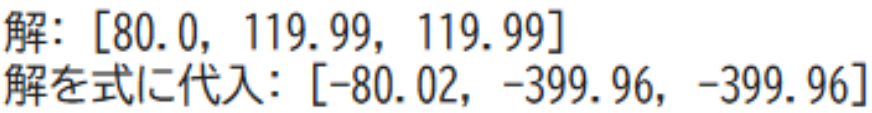

図 3　4.1 実行結果

4.1.e 考察:

実行結果を見ると, 求めた解を式に代入したところ, 4.1.b の式と一致していることが分かる. よってプログラムは正しいと言える.

課題 4.2

4.2.a 問題:

偏微分方程式 2(ラプラス方程式)
図 4 のような領域 R 内で定義されたラプラス方程式

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \tag{3}$$

の解を求める. ただし, 境界条件は以下である.

$$\begin{cases} \phi(x,y) = 10x, (x,y) \in S_1 \\ \phi(x,y) = 10y, (x,y) \in S_2 \\ \phi(x,y) = 30, (x,y) \in S_3 \end{cases} \tag{4}$$

刻み幅を 1 として領域 R 内で定義されたラプラス方程式の解$\phi_{1.1}, \phi_{2.1}, \phi_{1.2}$(図 4) を求めよ.(解 : $\phi_{1.1} = 17.14, \phi_{2.1} = 24.29, \phi_{1.2} = 24.29$)

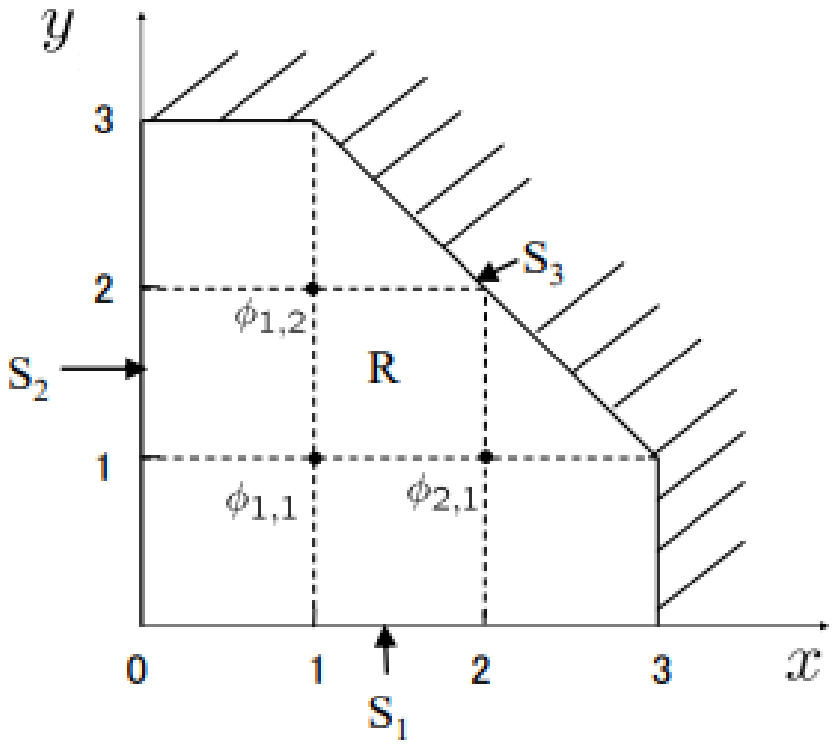

図 4　偏微分方程式 2

4.2.b アルゴリズム:

図 4 と境界条件より以下のような行列ができる.

$$\begin{pmatrix} -4 & 1 & 1 \\ 1 & -4 & 0 \\ 1 & 0 & -4 \end{pmatrix} \begin{pmatrix} \phi_{11} \\ \phi_{21} \\ \phi_{12} \end{pmatrix} = \begin{pmatrix} -20 \\ -80 \\ -80 \end{pmatrix}$$

これを逆行列法を用いて解く. 逆行列法のアルゴリズムは図 2 に示めした通りである.

4.2.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 2　演習課題. Python プログラム

```
1 #4-2
2 import numpy as np
3 import math
4 import sympy as sym
5 import matplotlib.pyplot as plt
6 import copy
```

```
7
8  #逆行列による連立1次方程式の解法
9  def InverseMatrix(a, b, x):
10     n = len(a)
11     a1 = copy.deepcopy(a)
12     C = a1
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
20     for k in range(n):
21         Ctmp = C[k][k]
22         for j in range(2 * n):
23             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, 2 * n):
28                     C[i][j] = C[i][j] - C[k][j] * Ctmp
29
30     for i in range(n):
31         x[i] = 0
32         for j in range(n):
33             x[i] = round(x[i] + C[i][n+j] * b[j], 2)
34
35 #連立方程式に解析解を代入する関数
36 def assignment(a, x):
37     n = len(a)
38     ans = [0] * n
39     for i in range(n):
40         for j in range(n):
41             ans[i] += a[i][j] * x[j]
42     return(ans)
43
44 def main():
45     a = [[-4, 1, 1],[1, -4, 0],[1, 0, -4]]
46     b = [-20, -80, -80]
47     x = [0] * len(a)
48     InverseMatrix(a, b, x)
```

```
49
50      print("解:", x)
51      print("解を式に代入:", assignment(a, x))
52
53  if __name__ == "__main__":
54      main()
```

4.2.d 実行結果:

実行結果を図 5 に示す.

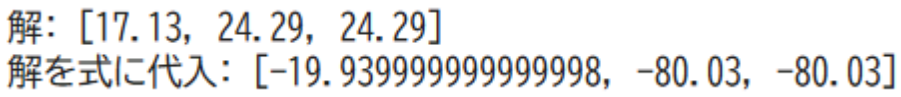

図 5　4.2 実行結果

4.2.e 考察:

実行結果を見ると, 求めた解を式に代入したところ, 4.2.b の式と一致していることが分かる. よってプログラムは正しいと言える.

課題 4.3

4.3.a 問題:

行列 $\mathrm{A} = \begin{pmatrix} 4 & 1 \\ 1 & 0 \end{pmatrix}$ の絶対値最大の固有値とその固有ベクトルをべき乗法により求めよ.

解：固有値 $\lambda_1 = 4.236$, 固有ベクトル $\mathbf{u}_1 = \begin{pmatrix} 0.973 \\ 0.230 \end{pmatrix}$
（ただし初期値により固有ベクトルの向きが反対になる場合もある. ）

4.3.b アルゴリズム:

べき乗法のアルゴリズムを図 6 に示す.

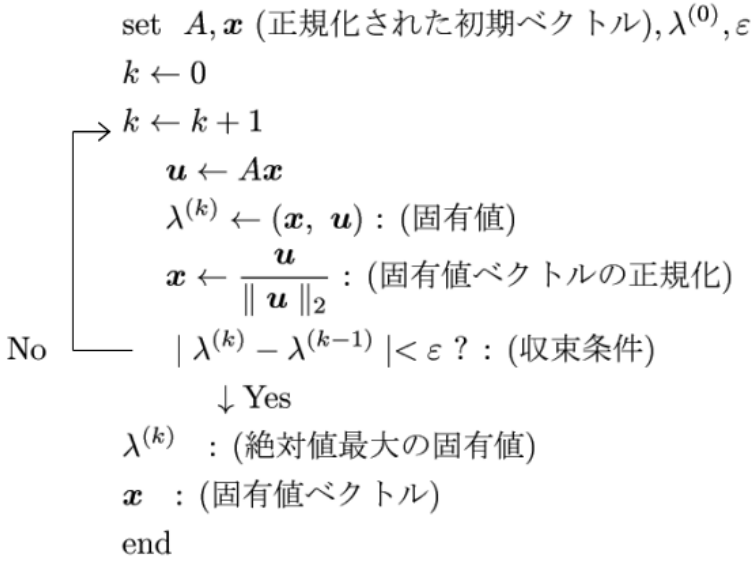

図 6　べき乗法 (Power Method) のアルゴリズム

4.3.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 3　演習課題. Python プログラム

```
#4-3
import numpy as np

def powerMethod(A, x, e):
    l = [1]
    k = 0
    while True:
        k += 1
        u = np.dot(A, x)
        l.append(np.dot(x.T, u))
        x = np.dot(u, 1 / np.sqrt(np.dot(u.T, u)))
        if abs(l[k] - l[k-1]) < e:
            break

    return l[-1], x

def main():
    A = np.array([
        [4, 1],
        [1, 0]])
```

```
21      e = 10E-5
22      x = np.array([1, 1])
23      l, u = powerMethod(A, x, e)
24      print("固有値:", l)
25      print("固有ベクトル:", u)
26
27  if __name__ == "__main__":
28      main()
```

4.3.d 実行結果:

実行結果を図 7 に示す.

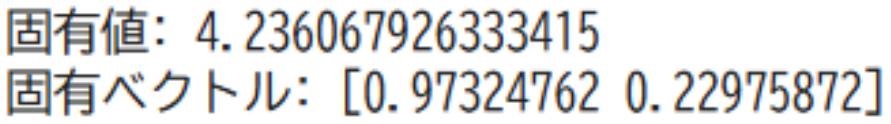

図 7　4.3 実行結果

4.3.e 考察:

実行結果を見ると誤差はほとんどないことがわかる.
固有値問題は $Ax = \lambda x$ を満たすため, これを今回求めた解に当てはめると

$$
\begin{aligned}
(\text{左辺}) = Au_1 &= \begin{pmatrix} 4 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0.97324762 \\ 0.22975872 \end{pmatrix} \\
&= \begin{pmatrix} 4.1227492 \\ 0.97324762 \end{pmatrix} \\
(\text{右辺}) &= \lambda_1 u_1 \\
&= 4.236067926333415 \begin{pmatrix} 4.1227492 \\ 0.97324762 \end{pmatrix} \\
&= \begin{pmatrix} 4.1227430274623314752223 \\ 0.9732735445874197236288 \end{pmatrix}
\end{aligned}
$$

(左辺) ≒ (右辺) より求めは解は正しいと言える.

課題 4.4

4.4.a 問題:

行列 $\mathrm{A} = \begin{pmatrix} 1 & 3 & 2 \\ 3 & 5 & -1 \\ 2 & -1 & 3 \end{pmatrix}$ の絶対値最大の固有値とその固有ベクトルをべき乗法により求めよ.

解：固有値 $\lambda_1 = 6.607$, 固有ベクトル $\mathbf{u}_1 = \begin{pmatrix} -0.4776 \\ -0.8783 \\ -0.02128 \end{pmatrix}$

（ただし初期値により固有ベクトルの向きが反対になる場合もある. ）

4.4.b アルゴリズム:

べき乗法のアルゴリズムは図 6 に示した通りである.

4.4.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 4　演習課題. Python プログラム

```
#4-4
import numpy as np

def powerMethod(A, x, e):
    l = [1]
    k = 0
    while True:
        k += 1
        u = np.dot(A, x)
        l.append(np.dot(x.T, u))
        x = np.dot(u, 1 / np.sqrt(np.dot(u.T, u)))
        if abs(l[k] - l[k-1]) < e:
            break

    return l[-1], x

def main():
    A = np.array([
        [1, 3, 2],
        [3, 5, -1],
```

```
21          [2, -1, 3]])
22      e = 10E-5
23      x = np.array([1, 1, 1])
24      l, u = powerMethod(A, x, e)
25      print("固有値:", l)
26      print("固有ベクトル:", u)
27
28  if __name__ == "__main__":
29      main()
```

4.4.d 実行結果:

実行結果を図 8 に示す.

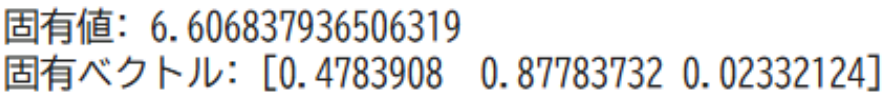

図 8　4.4 実行結果

4.4.e 考察:

実行結果を見ると誤差はほとんどないことがわかる.
固有値問題は $Ax = \lambda x$ を満たすため, これを今回求めた解に当てはめると

$$
\begin{aligned}
(\text{左辺}) = Au_1 &= \begin{pmatrix} 1 & 3 & 2 \\ 3 & 5 & -1 \\ 2 & -1 & 3 \end{pmatrix} \begin{pmatrix} 0.4783908 \\ 0.87783732 \\ 0.02332124 \end{pmatrix} \\
&= \begin{pmatrix} 3.15854524 \\ 5.80103776 \\ 0.148908 \end{pmatrix} \\
(\text{右辺}) &= \lambda_1 u_1 \\
&= 6.606837936506319 \begin{pmatrix} 0.4783908 \\ 0.87783732 \\ 0.02332124 \end{pmatrix} \\
&= \begin{pmatrix} 3.1606504859156071514652 \\ 5.7997289078570372340 2508 \\ 0.1540796531583686269 1556 \end{pmatrix}
\end{aligned}
$$

(左辺) ≒ (右辺) より求めは解は正しいと言える.

課題 4.5

4.5.a 問題:

行列 $\mathrm{A} = \begin{pmatrix} 3 & 0 & 0 & 1 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 3 \end{pmatrix}$ の絶対値最大の固有値とその固有ベクトルをべき乗法により求めよ.

解：固有値 $\lambda_1 = 4.000$, 固有ベクトル $\mathbf{u}_1 = \begin{pmatrix} 0.707 \\ 0 \\ 0 \\ 0.707 \end{pmatrix}$

4.5.b アルゴリズム:

べき乗法のアルゴリズムは図 6 に示した通りである.

4.5.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 5　演習課題. Python プログラム

```
#4-5
import numpy as np

def powerMethod(A, x, e):
    l = [1]
    k = 0
    while True:
        k += 1
        u = np.dot(A, x)
        l.append(np.dot(x.T, u))
        x = np.dot(u, 1 / np.sqrt(np.dot(u.T, u)))
        if abs(l[k] - l[k-1]) < e:
            break

    return l[-1], x

def main():
    A = np.array([
```

```
19          [3, 0, 0, 1],
20          [0, 3, 0, 0],
21          [0, 0, 1, 0],
22          [1, 0, 0, 3]
23         ])
24     e = 10E-5
25     x = np.array([1, 1, 1, 1])
26     n = 2
27     l, u = powerMethod(A, x, e)
28     print("固有値:", l)
29     print("固有ベクトル:", u)
30
31 if __name__ == "__main__":
32     main()
```

4.5.d 実行結果:

実行結果を図 9 に示す.

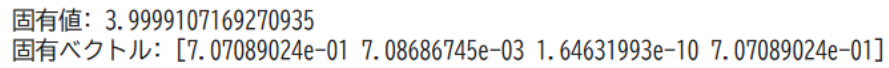

図 9　4.5 実行結果

4.5.e 考察:

実行結果を見ると誤差はほとんどないことがわかる.
固有値問題は $Ax = \lambda x$ を満たすため, これを今回求めた解に当てはめると

$$
\begin{aligned}
(\text{左辺}) = Au_1 &= \begin{pmatrix} 3 & 0 & 0 & 1 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} 7.07089024e-01 \\ 7.08686745e-03 \\ 1.64631993e-10 \\ 7.07089024e-01 \end{pmatrix} \\
&= \begin{pmatrix} 2.828356096 \\ 0.02126060235 \\ 1.64631993e-10 \\ 2.828356096 \end{pmatrix} \\
(\text{右辺}) &= \lambda_1 u_1 \\
&= 3.9999107169270935 \begin{pmatrix} 7.07089024e-01 \\ 7.08686745e-03 \\ 1.64631993e-10 \\ 7.07089024e-01 \end{pmatrix} \\
&= \begin{pmatrix} 2.8282929649191188220717 44 \\ 0.028346837062696782948256575 \\ 6.5851327314976623860234 55e-10 \\ 2.8282929649191188220717 44 \end{pmatrix}
\end{aligned}
$$

(左辺) ≒ (右辺) より求めは解は正しいと言える.

課題 4.8

4.8.a 問題:

課題 4.3 の問題において, 2 番目の固有値と固有ベクトルを求めよ.

解: $\lambda_2 = -0.236$, $\mathbf{u}_2 = \begin{pmatrix} -0.230 \\ 0.973 \end{pmatrix}$

4.8.b アルゴリズム:

最大固有値以外の固有値の求め方

簡単のため以下の行列 A は実対称行列とする. 実対象行列の固有値はすべて実数であることが知られていることを利用する.

絶対値が 2 番目に大きい固有値 λ_2, 固有ベクトル $\mathbf{u}_2$ の求め方は,

$$W_1 = A - \lambda_1 \mathbf{u}_1 \mathbf{u}_1^T$$

として, この W_1 をべき乗法に適用すればよい. なぜなら, $\mathbf{u}_1^T\mathbf{u}_1 = 1$ と正規化した $\mathbf{u}_1$ を用いて,

$$W_1\mathbf{u}_1 = A\mathbf{u}_1 - \lambda_1\mathbf{u}_1\mathbf{u}_1^T\mathbf{u}_1 = \lambda_1\mathbf{u}_1 - \lambda_1\mathbf{u}_1 = 0\cdot\mathbf{u}_1 = 0$$

$\mathbf{u}_1^T\mathbf{u}_i = 0 \quad (i = 2, \ldots, n)$ より

$$W_1\mathbf{u}_i = A\mathbf{u}_i - \lambda_1\mathbf{u}_1\mathbf{u}_1^T\mathbf{u}_i = A\mathbf{u}_i = \lambda_i\mathbf{u}_i \quad (i = 2, \ldots, n)$$

となり, λ_1 に対応する固有値が 0 となるほかは, A と同じ固有値, 固有ベクトルを有し W_1 の固有値は $\{0, \lambda_2, \ldots, \lambda_n\}$ となるからである. 絶対値が 3 番目に大きい固有値, 固有ベクトルを求める場合は,

$$W_2 = A - \lambda_1\mathbf{u}_1\mathbf{u}_1^T - \lambda_2\mathbf{u}_2\mathbf{u}_2^T$$

として, 同様にべき乗法を適用して求めることが可能である.

これらを利用して n 個の固有値と固有ベクトルを求める場合は固有値の条件が

$$|\lambda_1| > |\lambda_2| > |\lambda_3| > \cdots > |\lambda_n|$$

となる必要がある.

4.8.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 6　演習課題. Python プログラム

```
#4-8
import numpy as np

def powerMethod(A, x, e, n):
    for i in range(n):
        l = [1]
        k = 0
        while True:
            k += 1
            u = np.dot(A, x)
            l.append(np.dot(x.T, u))
            x = np.dot(u, 1 / np.sqrt(np.dot(u.T, u)))
            if abs(l[k] - l[k-1]) < e:
                break

        A = A - l[-1] * np.outer(x, x)

    return l[-1], x

```

```
def main():
    A = np.array([
        [4, 1],
        [1, 0]])
    e = 10E-5
    x = np.array([1, 1])
    n = 2
    l2, u2 = powerMethod(A, x, e, 2)
    print("2番目の固有値λ 2:", l2)
    print("2番目の固有ベクトルu2:", u2)

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

4.8.d 実行結果:

実行結果を図 10 に示す.

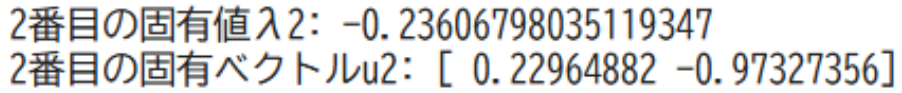

図 10　4.8 実行結果

4.8.e 考察:

1 つ目の解析解は 4.3 と同じなので正しいと言える. 2 つ目の求めた固有値, 固有ベクトルは解析解とほとんど誤差がないため正しいと言える.

課題 4.9

4.9.a 問題:

課題 4.4 の問題において, 2 番目と 3 番目の固有値と固有ベクトルを求めよ.

解: $\lambda_2 = -4.068$, $\mathbf{u}_2 = \begin{pmatrix} 0.3699 \\ -0.2230 \\ 0.9019 \end{pmatrix}$, $\lambda_3 = -1.674$, $\mathbf{u}_3 = \begin{pmatrix} 0.7969 \\ -0.4228 \\ -0.4314 \end{pmatrix}$

4.9.b アルゴリズム:

最大固有値以外の固有値, 固有ベクトルの求め方は 4.8.b アルゴリズムで示している.

4.9.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 7　演習課題. Python プログラム

```
#4-10
import numpy as np

def powerMethod(A, x, e, n):
    for i in range(n):
        l = [1]
        k = 0
        while True:
            k += 1
            u = np.dot(A, x)
            l.append(np.dot(x.T, u))
            x = u / np.linalg.norm(u)
            if abs(l[k] - l[k-1]) < e:
                break

        A = A - l[-1] * np.outer(x, x)

    return l[-1], x

def main():
    A = np.array([
        [3, 0, 0, 1],
        [0, 3, 0, 0],
        [0, 0, 1, 0],
        [1, 0, 0, 3]
        ])
    e = 10E-5
    x = np.array([1, 1, 1, 0])
    l2 , u2 = powerMethod(A, x, e, 2)
    l3 , u3 = powerMethod(A, x, e, 3)
    l4 , u4 = powerMethod(A, x, e, 4)
    print("2番目の固有値λ 2:", l2)
    print("2番目の固有ベクトルu2:", u2)
    print("3番目の固有値λ 3:", l3)
    print("3番目の固有ベクトルu3:", u3)
    print("4番目の固有値λ 4:", l4)
    print("4番目の固有ベクトルu4:", u4)
```

```
38
39 if __name__ == "__main__":
40     main()
```

4.9.d 実行結果:

実行結果を図 11 に示す.

```
2番目の固有値λ2: 4.067622286766513
2番目の固有ベクトルu2: [-0.36818337  0.2262115  -0.90181449]
3番目の固有値λ3: -1.6744733569445374
3番目の固有ベクトルu3: [-0.79692183  0.42283314  0.43142524]
```

図 11　4.9 実行結果

4.9.e 考察:

1 つ目の解析解は 4.4 と同じなので正しいと言える. 2, 3 つ目の求めた固有値, 固有ベクトルは解析解とほとんど誤差がないため正しいと言える.

課題 4.10

4.10.a 問題:

課題 4.5 の問題において, 2 番目と 3 番目と 4 番目の固有値と固有ベクトルを求めよ.

解: $\lambda_2 = -3$, $\mathbf{u}_2 = \begin{pmatrix} 0 \\ 1.000 \\ 0 \\ 0 \end{pmatrix}$, $\lambda_3 = 2.000$, $\mathbf{u}_3 = \begin{pmatrix} -0.707 \\ 0 \\ 0 \\ 0.707 \end{pmatrix}$, $\lambda_4 = 1.000$, $\mathbf{u}_4 = \begin{pmatrix} 0 \\ 0 \\ 1.000 \\ 0 \end{pmatrix}$

4.10.b アルゴリズム:

最大固有値以外の固有値, 固有ベクトルの求め方は 4.8.b アルゴリズムで示している.

4.10.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 8　演習課題. Python プログラム

```
1 #4-10
2 import numpy as np
3
4 def powerMethod(A, x, e, n):
```

```
    for i in range(n):
        l = [1]
        k = 0
        while True:
            k += 1
            u = np.dot(A, x)
            l.append(np.dot(x.T, u))
            x = u / np.linalg.norm(u)
            if abs(l[k] - l[k-1]) < e:
                break

        A = A - l[-1] * np.outer(x, x)

    return l[-1], x

def main():
    A = np.array([
        [3, 0, 0, 1],
        [0, 3, 0, 0],
        [0, 0, 1, 0],
        [1, 0, 0, 3]
        ])
    e = 10E-5
    x = np.array([1, 1, 1, 0])
    print(powerMethod(A, x, e, 2))
    print(powerMethod(A, x, e, 3))
    print(powerMethod(A, x, e, 4))

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

4.10.d 実行結果:

実行結果を図 12 に示す.

```
2番目の固有値λ2: 3.00008434281163
2番目の固有ベクトルu2: [ 7.21581668e-03 -9.99943410e-01 -8.60456327e-10  7.81726616e-03]
3番目の固有値λ3: 2.0000002713391183
3番目の固有ベクトルu3: [ 7.07106038e-01 -6.37976249e-04  1.01153879e-06 -7.07107236e-01]
4番目の固有値λ4: 1.0000000000020464
4番目の固有ベクトルu4: [ 1.43057014e-06  1.29127648e-09 -1.00000000e+00 -1.43049496e-06]
```

図 12　4.10 実行結果

4.10.e 考察:

1 つ目の解析解は 4.4 と同じなので正しいと言える. 2, 3 4 つ目の求めた固有値, 固有ベクトルは解析解とほとんど誤差がないため正しいと言える.