数値解析 レポート 1

自分でプログラムを書く計算問題(5),(6),(7)に取り組んだ。
計算環境：

CPU: Intel Core i5, メモリ: 8 GB, OS: macOS, プログラミング言語: Python 3.7.3

(5)
LU 分解のプログラムを以下のように実装した。

```
import numpy as np

def LU(A):
    A = np.array(A,dtype=np.float64)
    n = len(A)
    p = np.empty(n)
    for i in range(n):
        p[i] = i
    for k in range(n-1):
        m = k
        for l in range(k,n): #枢軸選択、[k,n-1]にあるmのうち|A[l,k]|が最大になるmを探す
            if abs(A[l,k]) > abs(A[m,k]):
                m = l
        p_m,p_k = p[m],p[k] #p[m],p[k]の交換
        p[m],p[k] = p_k,p_m
        A_m,A_k = np.array(A[m]),np.array(A[k]) #A[m],A[k]の交換
        A[m],A[k] = A_k,A_m
        w = 1.0/A[k,k]
        for i in range(k+1,n):
            A[i,k] = A[i,k]*w #PAの狭義下三角部分を作成
            for j in range(k+1,n): #前進消去
                A[i,j] -= A[i,k]*A[k,j]
    L=np.zeros([n,n])
    U=np.zeros([n,n])
    for i in range(n-1): #L,Uを作成
        L[i,i] = 1
        U[i,i] = A[i,i]
        for j in range(i+1,n):
            L[j,i] = A[j,i]
            U[i,j] = A[i,j]
    L[n-1,n-1] = 1
    U[n-1,n-1] = A[n-1,n-1]
    return p,L,U
```

1.
3 つの行列に対して先のプログラムと下記の LU 分解をチェックするプログラムを実行し、正しく LU 分解が計算できることを確認した。

（LU 分解をチェックするプログラム）

```
def LU_check(A): #PA,LUを出力する関数
    p,L,U = LU(A)
    n = len(A)
    PA = np.empty([n,n])
    for i in range(n):
        PA[i] = A[p[i]]
    return (PA,np.dot(L,U))
```

・A: $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

結果

print(LU(A)):

```
(array([ 2.,  0.,  1.]),
array([[ 1.        ,  0.        ,  0.        ],
       [ 0.14285714,  1.        ,  0.        ],
       [ 0.57142857,  0.5       ,  1.        ]]),
array([[  7.00000000e+00,   8.00000000e+00,   9.00000000e+00],
       [  0.00000000e+00,   8.57142857e-01,   1.71428571e+00],
       [  0.00000000e+00,   0.00000000e+00,   1.11022302e-16]]))
```

print(LU_check(A))

```
(array([[ 7.,  8.,  9.],
       [ 1.,  2.,  3.],
       [ 4.,  5.,  6.]]),
array([[ 7.,  8.,  9.],
       [ 1.,  2.,  3.],
       [ 4.,  5.,  6.]]))
```

・B: $\begin{pmatrix} 3 & 3 & -5 & -6 \\ 1 & 2 & -3 & -1 \\ 2 & 3 & -5 & -3 \\ -1 & 0 & 0 & 1 \end{pmatrix}$

結果

print(LU(B))

```
(array([ 0.,  1.,  2.,  3.]),
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [ 0.33333333,  1.        ,  0.        ,  0.        ],
       [ 0.66666667,  1.        ,  1.        ,  0.        ],
       [-0.33333333,  1.        ,  1.        ,  1.        ]]),
array([[ 3.        ,  3.        , -5.        , -6.        ],
       [ 0.        ,  1.        , -1.33333333,  1.        ],
       [ 0.        ,  0.        , -0.33333333,  0.        ],
       [ 0.        ,  0.        ,  0.        , -2.        ]]))
```

print(LU_check(B))

```
(array([[ 3.,  3., -5., -6.],
       [ 1.,  2., -3., -1.],
```

```
       [ 2.,  3., -5., -3.],
       [-1.,  0.,  0.,  1.]]),
array([[ 3.,  3., -5., -6.],
       [ 1.,  2., -3., -1.],
       [ 2.,  3., -5., -3.],
       [-1.,  0.,  0.,  1.]]))
```

・C: $\begin{pmatrix} 2 & 1 & 3 & 0 & -1 \\ 1 & 3 & 0 & 2 & 1 \\ -1 & 4 & 1 & -2 & 2 \\ -3 & -2 & 1 & -4 & 3 \\ 1 & 2 & 4 & -1 & -2 \end{pmatrix}$

結果

print(LU(C))

```
(array([ 3.,  2.,  4.,  1.,  0.]),
array([[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00],
       [  3.33333333e-01,   1.00000000e+00,   0.00000000e+00,
          0.00000000e+00,   0.00000000e+00],
       [ -3.33333333e-01,   2.85714286e-01,   1.00000000e+00,
          0.00000000e+00,   0.00000000e+00],
       [ -3.33333333e-01,   5.00000000e-01,  -1.33992434e-17,
          1.00000000e+00,   0.00000000e+00],
       [ -6.66666667e-01,  -7.14285714e-02,   8.96551724e-01,
         -7.93103448e-01,   1.00000000e+00]]),
array([[-3.        , -2.        ,  1.        , -4.        ,  3.        ],
       [ 0.        ,  4.66666667,  0.66666667, -0.66666667,  1.        ],
       [ 0.        ,  0.        ,  4.14285714, -2.14285714, -1.28571429],
       [ 0.        ,  0.        ,  0.        ,  1.        ,  1.5       ],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  3.4137931 ]]))
```

print(LU_check(C))

```
(array([[-3., -2.,  1., -4.,  3.],
       [-1.,  4.,  1., -2.,  2.],
       [ 1.,  2.,  4., -1., -2.],
       [ 1.,  3.,  0.,  2.,  1.],
       [ 2.,  1.,  3.,  0., -1.]]),
array([[-3., -2.,  1., -4.,  3.],
       [-1.,  4.,  1., -2.,  2.],
       [ 1.,  2.,  4., -1., -2.],
       [ 1.,  3.,  0.,  2.,  1.],
       [ 2.,  1.,  3.,  0., -1.]]))
```

2.

与えられた行列 A について 1 段ごとに消去が進む様子を示すと以下のようになる。

```
[[ 10.    2.    3.   -1. ]
 [  0.2   9.6  -2.6   0.2]
 [ -0.1   3.2  10.3   3.9]
 [  0.2  -1.4   4.4  10.2]]
[[ 10.          2.          3.         -1.        ]
 [  0.2         9.6        -2.6         0.2       ]
 [ -0.1         0.33333333 11.16666667  3.83333333]
 [  0.2        -0.14583333  4.02083333 10.22916667]]
```

```
[[ 10.          2.           3.          -1.        ]
 [  0.2         9.6         -2.6          0.2       ]
 [ -0.1         0.33333333  11.16666667   3.83333333]
 [  0.2        -0.14583333   0.36007463   8.8488806 ]]
```

LU_check(A)を実行すると正しく LU 分解が実行できていることが確認できた。

```
(array([[ 10.,   2.,   3.,  -1.],
       [  2.,  10.,  -2.,   0.],
       [ -1.,   3.,  10.,   4.],
       [  2.,  -1.,   5.,  10.]]),
array([[ 10.,   2.,   3.,  -1.],
       [  2.,  10.,  -2.,   0.],
       [ -1.,   3.,  10.,   4.],
       [  2.,  -1.,   5.,  10.]]))
```

(6)
CG 法のプログラムを以下のように実装した。

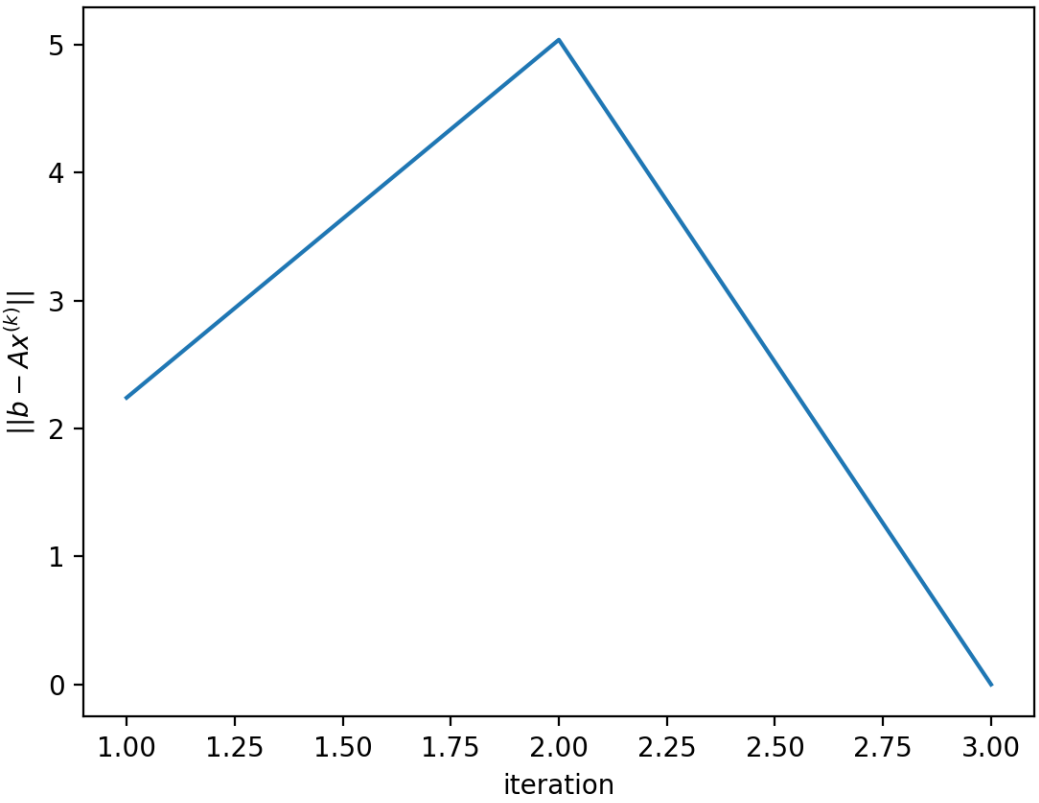
```
import numpy as np

def CG(A,b):
    n = len(b)
    x = np.zeros(n)
    r = b-np.dot(A,x)
    p = np.array(r)
    re = □ #グラフ作成用に残差を記録
    maxiter = 1000
    for k in range(maxiter):
        Ap = np.dot(A,p) #Apを予め計算
        alpha = np.dot(r,p)/np.dot(p,Ap)
        x += alpha*p #x,rを更新
        r += -alpha*np.dot(A,p)
        beta = -np.dot(r,Ap)/np.dot(p,Ap)
        p = r+beta*p #pを更新
        re.append(np.linalg.norm(b-np.dot(A,x)))
        if np.linalg.norm(b-np.dot(A,x))<1.0/10000*np.linalg.norm(b):
            return x,re
            break
    else:
        return "no convergence"
```

4 つの実対称正定値行列に対して実際に計算を行い、収束の様子を図示すると以下のようになった。

1)

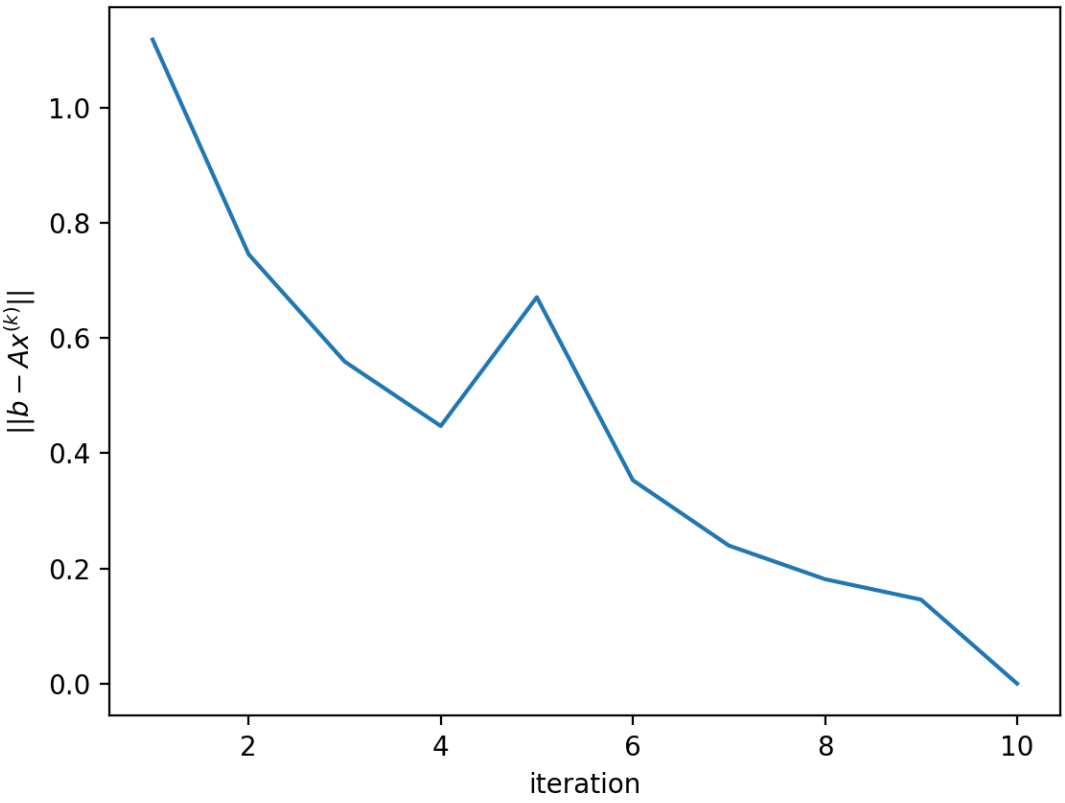
$$\begin{pmatrix} 2 & 1 & 2 \\ 1 & 3 & -1 \\ 2 & 1 & 4 \end{pmatrix} x = \begin{pmatrix} 1 \\ 4 \\ 2 \end{pmatrix}$$

を CG 法で計算すると、$x = (-13.5 \quad 9 \quad 9.5)^{\mathrm{T}}$ となり、収束の様子は以下の通り。

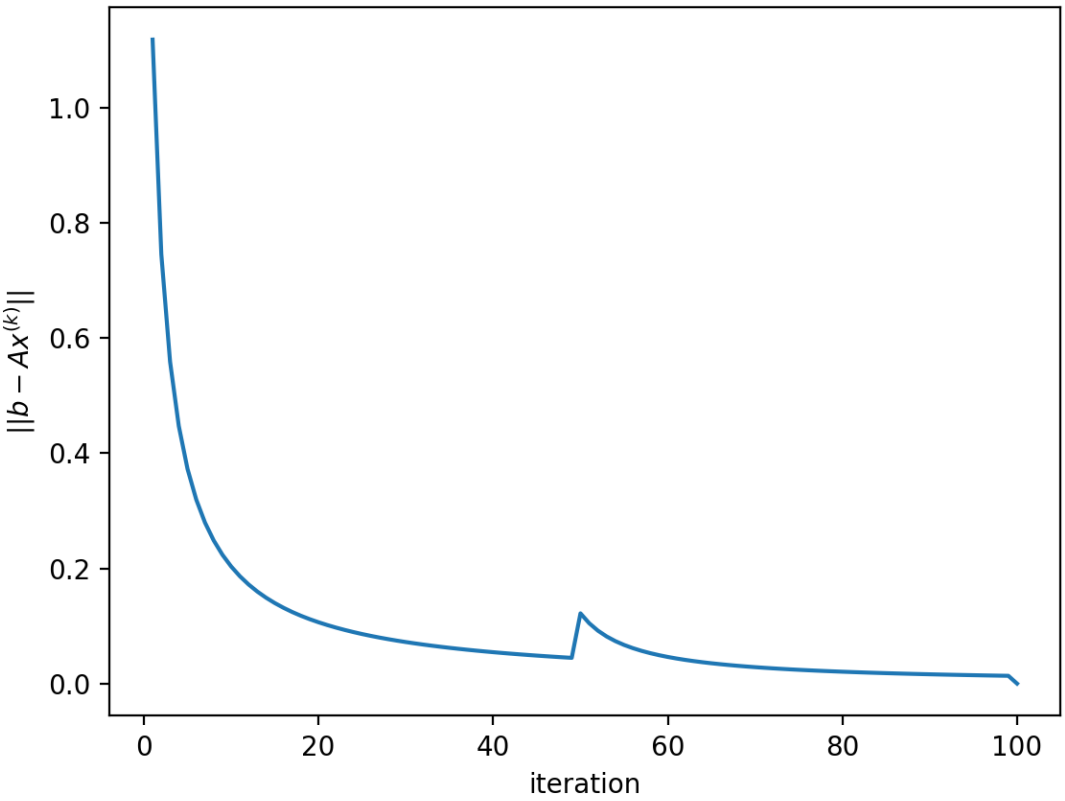


2) 授業スライド 2.1 にあったように、1 次元 Laplace 方程式を、微分の差分近似をして行右列計算にしたものを実際に計算した。境界条件を $u(0) = 1, u(1) = 2$ とし、$N = 10, 100, 1000$ とすると以下のような結果が得られた。

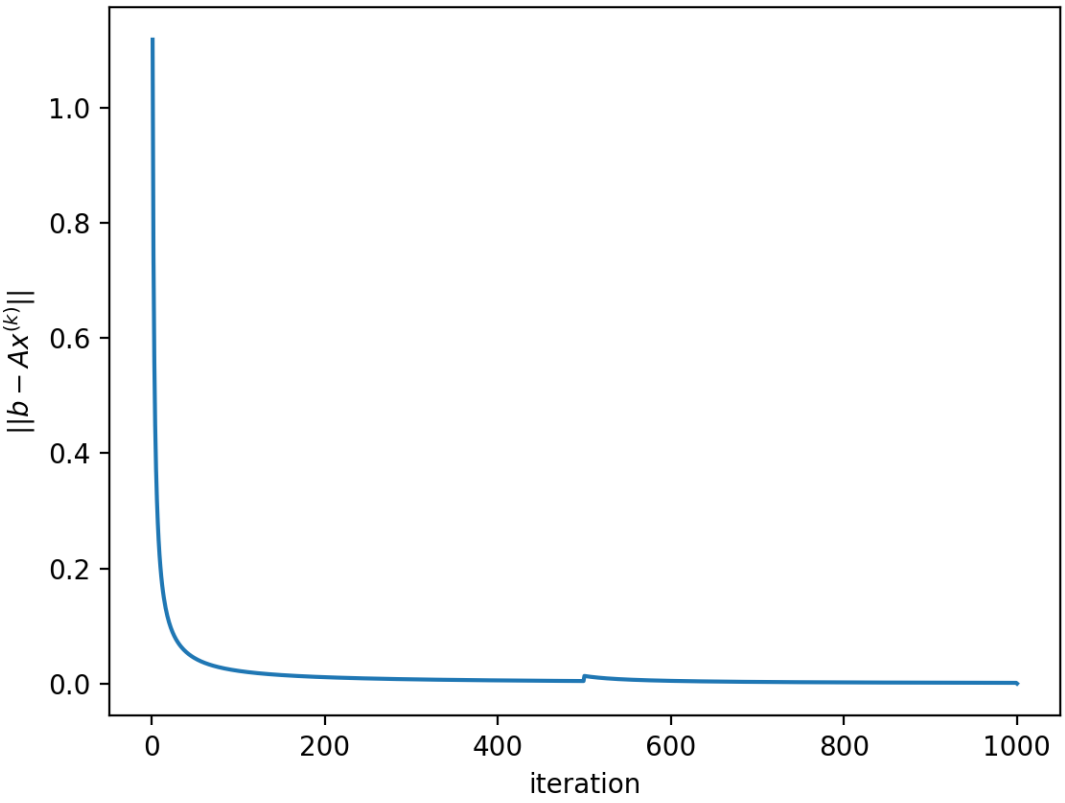
$\underline{N = 10 \text{ のとき}}$



$N = 100$ のとき



$\underline{N = 1000 \text{ のとき}}$



以上のグラフより、CG 法はかなり収束が早いことが分かった。残差が小さい時には残差のオーダーが下がりにくくなっているので、どの精度で答えを得たいかに注意しつつなるべく反復の回数が少なくなるように停止条件を設定するのが良いと考えた。

(7)
べき乗法を以下のように実装した。

```
import numpy as np

def PM(A):
    A = np.array(A,dtype=np.float64)
    maxiter = 10000
    x = np.ones(len(A)) #初期ベクトル
    for i in range(maxiter):
        former_x = x
        y = np.dot(A,former_x)
        x = y/np.linalg.norm(y)
        if np.linalg.norm(x-former_x)<1.0/1000:
            m = np.argmax(abs(x)) #絶対値最大の成分のインデックスを取得
            return y[m]/former_x[m] #成分比として固有値を計算
```

```
            break
    else:
        return "no convergence"
```

上で実装したべき乗法が正しく回っているかの確認用の $A_3 = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 3 \\ 2 & 2 & 1 \end{pmatrix}$ も合わせて、A_1, A_2, A_3 について計算を実行したところ以下のようになった。

入力：

```
print(PM(A1))
print(PM(A2))
print(PM(A3))
```

出力

```
10.0
no convergence
4.0
```

この結果の解釈は以下のようになる。

A_1, A_2 の固有値はそれぞれ $\lambda = 10, \frac{5 \pm \sqrt{31}i}{2}$、$\lambda = 1, \pm\sqrt{3}$ であるから、それぞれの絶対値最大の固有値は$\lambda = 10$、$\lambda = \pm\sqrt{3}$ となる。前者は絶対値最大の固有値が 1 つのみであり、べき乗法が正しく回るが、後者は絶対値最大の固有値が 2 つあるために、x が固有値 $\lambda = \pm\sqrt{3}$ の 2 つの固有ベクトルで張られる空間内を回転してしまい収束しない。

また、A_3については固有値が$\lambda = -1, 2, 4$ となるから正しく計算できていることが確認できた。初めは固有値をベクトル$x^{(k)}, y^{(k+1)}$の第 1 成分を比較することで計算していたが、出てきた固有値が 2 となってしまったため、授業で教わったように絶対値最大の比較に修正した。これは固有ベクトルの第 1 成分が 0 となってしまうからだと分かり、絶対値最大の比較が良いことの理由が実感できた。