HOMEWORK3

Dingcheng Yue

Problem1: (20 points)

Let

$$A = \begin{bmatrix} 1 & 4 \\ 1 & 1 \end{bmatrix}$$

Your answers to the following questions should be numeric and specific to this particular matrix, not just the general definitions.

(a) What is the characteristic polynomial of **A**?

$$(1-\lambda)^2 - 4 = 0$$

(b) What are the roots of the characteristic polynomial of **A**?

$$\lambda_1 = 3, \lambda_2 = -1$$

(c) What are the eigenvalues of **A**? (d) What are the eigenvectors of **A**?

$$\begin{aligned} \lambda_1 &= 3 & x_1 &= c\begin{bmatrix} 2 \\ 1 \end{bmatrix} \\ \lambda_1 &= -1 & x_1 &= c\begin{bmatrix} -2 \\ 1 \end{bmatrix} \end{aligned}$$

(e) Perform one iteration of power iteration on **A**, using $[1 \quad 1]^T$ as starting vector.

$x_2 = normalised(A^2x) = [1 \quad 0.4]^T$

(f) To what eigenvector of **A** will power iteration ultimately converge?

x = $\begin{bmatrix} 1 & 0.5 \end{bmatrix}^T$ (Normalised based on the infinity norm)

(g) What eigenvalue estimate is given by the Rayleigh quotient, using the vector x = $\begin{bmatrix} 1 & 1 \end{bmatrix}^T$?

λ = 3.0

(h) To what eigenvector of A would inverse iteration ultimately converge?

x = $\begin{bmatrix} -1 & 0.5 \end{bmatrix}^T$ (Normalised based on the infinity norm)

(i) What eigenvalue of A would be obtained if inverse iteration were used with shift σ = 2? (Normalised based on the infinity norm)

x = $\begin{bmatrix} 1 & 0.5 \end{bmatrix}^T$

(j) If QR iteration were applied to A, to what form would it converge: diagonal or triangular? Why?

It will converge to the triangle form. Since it is not a symmetric matrix. And QR iteration is a similar transformation will become to symmetric only if matrix before transformation is symmetric. And diagonal matrix is symmetric.

Problem 2: (10 points)

Suppose that the following partitioned matrix:

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{C} \end{bmatrix}$$

is orthogonal, where the submatrices **A** and **C** are square. Prove that **A** and **C** must be orthogonal, and **B** = **0**.

$$\begin{aligned}
M^T M &= I \\
\begin{bmatrix} A & B \\ 0 & C \end{bmatrix}^T \begin{bmatrix} A & B \\ 0 & C \end{bmatrix} &= \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \\
\begin{bmatrix} A^T A & A^T B \\ B^T A & B^T B + C^T C \end{bmatrix} &= \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \\
A^T A &= I \\
A^T B &= 0 \\
B^T A &= 0 \\
B^T B + C^T C &= I
\end{aligned}$$

Thus, A is orthonormal since $A^T A = I$
B is zero since $B = (A^T)^{-1} 0 = A0 = 0$
C is othonormal since $B^T B + C^T C = C^T C = I$

Problem 3: (10 points)

Let x be the solution to the linear least squares problem Ax ≈ b, where:

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$$

Let $r = b - Ax$ be the corresponding residual vector. Which of the following three vectors is a possible value for r? Why?

(a) $\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ (b) $\begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$ (c) $\begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$

$r = b - Ax => A^T r = A^T b - A^T Ax = 0$
Thus, only (c) is the correct answer.

Problem4: (15 points)

(a) What are the eigenvalues of the Householder transformation

$$H = I - 2\frac{vv^T}{v^T v},$$

where v is any nonzero vector?

$$det(H - \lambda I) = -\frac{v^T v}{2} det(\frac{v^T v(1-\lambda)}{2} I + vv^T) = 0$$

Since the eigenvalue for vv^T shall be 0 (since its rank 1, thus determinant is zero already) or $v^T v$
$vv^T b = \lambda b \Rightarrow v^T vv^T b = \lambda v^T b \Rightarrow v^T v = \lambda$
Thus, $\frac{v^T v(1-\lambda)}{2} = 0$ or $\frac{v^T v(1-\lambda)}{2} = v^T v$
Thus, $\lambda = 1$ or $\lambda = -1$

(b)What are the eigenvalues of the plane rotation

$$G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

where $c^2 + s^2 = 1$?

characteristic polynomial $(c - \lambda)^2 + s = 0$
Thus, $\lambda = c \pm si$

Problem 5: (15 points)

(a) Implement power iteration to compute the dominant eigenvalue and a corresponding eigenvector of the matrix

$$A = \begin{bmatrix} 2 & 3 & 2 \\ 10 & 3 & 4 \\ 3 & 6 & 1 \end{bmatrix}$$

As starting vector, take $x_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$

```python
import numpy as np
import numpy.linalg as la


def norm(x):
    return np.max(np.abs(x))

def normalise(x):
    return x/norm(x)

def shift(A, sigma):
    return A-np.eye(len(A))*sigma

def powerIter(A, x):
    while True:
        x = A.dot(x)
        x = x/norm(x)
        yield(x)

def rayleighQuotient(A, x):
    return x.dot(A).dot(x)/x.dot(x)

def estimate(A, x, iterator, error=0.0):
    # stop when it becomes stable, using infinity norm
    v_prev = x
    countinggg = 0
    for v in iterator(A, x):
        if norm(v_prev-v)<=error or countinggg>100:
            return v, rayleighQuotient(A, v)
        else:
            v_prev = v
            countinggg += 1


A = np.array([[2.0, 3.0, 2.0],[10, 3, 4],[3, 6, 1]])
x = np.array([0.0, 0.0, 1.0])

x1, lambda1 = estimate(A, x, powerIter)
print x1, lambda1
```

result: (x, λ)

```
[ 0.5 1. 0.75] 11.0
```

(b) Using any of the methods for deflation given in Section 4.5.4, deflate out the eigenvalue found in part (a) and apply power iteration again to compute the second largest eigenvalue of the same matrix.

```
def deflation(A, x1):
    v = x1.copy()
    v[0]-=la.norm(x1)
    H = np.eye(x1.size)-2*np.outer(v,v)/np.dot(v,v)
    return H.dot(A).dot(la.inv(H))[1:,1:]

A2 = deflation(A, x1)

x2, lambda2 = estimate(A2, np.array([1.0, 1.0]), powerIter)

print x2, lambda2
```

result: (something that is not important, λ)

```
[-0.7827455 1. ] -3.0
```

(c) Use a general real eigensystem library routine to compute all of the eigenvalues and eigenvectors of the matrix, and compare the results with those obtained in parts (a) and (b).

```
print la.eig(A)
```

result:

```
(array([ 11., -2., -3.]),
array([[ 3.71390676e-01, 1.82574186e-01, -4.13692033e-16],
[ 7.42781353e-01, 3.65148372e-01, -5.54700196e-01],
[ 5.57086015e-01, -9.12870929e-01, 8.32050294e-01]]))
```

Problem 6: (15 points)

(a) Implement inverse iteration with a shift to compute the eigenvalue nearest to 2, and the corresponding eigenvector, of the matrix

$$A = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

You may use an arbitrary starting vector.

```python
import numpy as np

def norm(x):
    return np.max(np.abs(x))

def shift(A, sigma):
    return A-np.eye(len(A))*sigma


def inversePowerIter(A, x):
    while True:
        x = np.linalg.solve(A, x)
        x = x/norm(x)
        yield(x)

A = np.array([[6.0, 2.0, 1.0],[2.0, 3.0, 1.0],[1.0, 1.0,
1.0]])
x = np.array([0.0, 0.0, 1.0])

def rayleighQuotient(A, x):
    return x.dot(A).dot(x)/x.dot(x)

def estimating(A, x, iterator, error=0.0):
    # stop when it becomes stable, using infinity norm
    v_prev = x
    for v in iterator(A, x):
        if norm(v_prev-v)<=error:
            return v, rayleighQuotient(A, v)
        else:
            v_prev = v

print estimating(A, x, inversePowerIter)
```

result:

```
(array([-0.04614548, -0.37492113, 1. ]), 0.57893338569105257)
```

(b) Use a real symmetric eigensystem library routine to compute all of the eigenvalues and eigenvectors of the matrix, and compare the results with those obtained in part (a)

```
import numpy.linalg as la
print la.eigvalsh(A)
```

result:

```
array([ 0.57893339, 2.13307448, 7.28799214])
```

The result indicates three eigen-vector, and indeed 2.133 is the nearest one.

(c) Write a program implementing Rayleigh quotient iteration for computing an eigenvalue and corresponding eigenvector of a matrix. Test your program on the given matrix using a random starting vector.

```
def rayleighIter(A, x):
    while True:
        sigma = rayleighQuotient(A, x)
        x = np.linalg.solve(shift(A, sigma), x)
        x = x/norm(x)
        yield(x)

print estimating(A, x, rayleighIter)
```

result:

```
(array([ 0.04614548, 0.37492113, -1. ]), 0.57893338569105257)
```

So, this rayleigh-iter solves the eigenvector near zero, and it matches the result above.

Problem 7: (15 points)

In this problem, you would be aiding a planetary scientist who's recently detected a new object, potentially a planetoid. She has provided you with the Cartesian coordinates in a fitted coordinate system (i.e., after some normalization) of the positions of the object in `planetoid.txt`. Your aim is to estimate the orbit of this object on the basis of the provided observations. You shall do so assuming the orbit to be defined by two different curves, an ellipse:

$$x^2 = ay^2 + bxy + cx + dy + e$$

and a parabola:

$$x^2 = dy + e$$

Set up a system of overdetermined system of linear equations in each case for the unknown coefficients. You should solve the resulting least-squares system using QR factorization. You can use library functions for this purpose, for example, `npla.qr` followed by `npla.solve`. Which of the two estimated trajectories that you have obtained is more likely? Quantify your claim. (Hint: residuals!) Also, back your claim by plotting the two estimated trajectories.

The first one since the second one is the serious underfitting. The residual has been printed out and it is clearly that the first model fits quite well and it also can be seen from the graph

The solution for the equation

$$x^2 = ay^2 + bxy + cx + dy + e$$

is

$$y = \frac{-(bx+d) \pm \sqrt{(bx+d)^2 - 4a(cx+e-x^2)}}{2a}$$

And solution for the equation

$$x^2 = dy + e$$

is

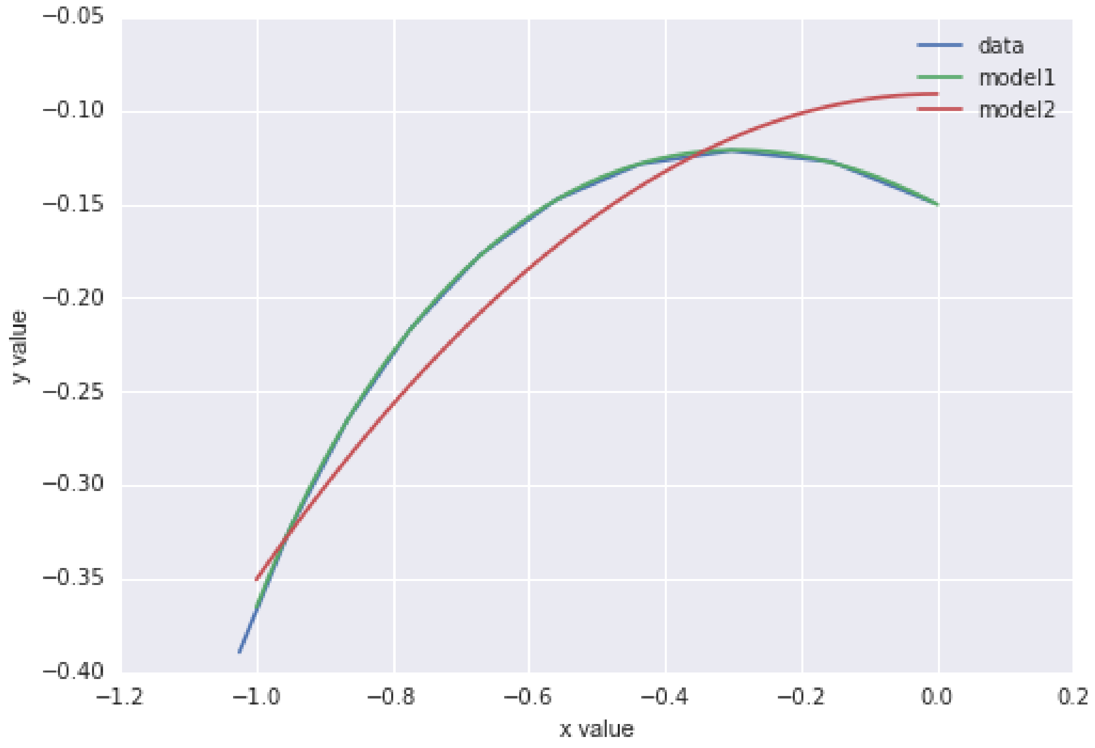
$$y = \frac{x^2 - e}{d}$$

The code is as follows:

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
# import seaborn


data = np.array([[-1.02494, -0.389269],
[-0.949898, -0.322894],
[-0.866114, -0.265256],
[-0.773392, -0.216557],
[-0.671372, -0.177152],
[-0.559524, -0.147582],
[-0.437067, -0.128618],
[-0.302909, -0.121353],
[-0.155493, -0.127348],
[-0.007464, -0.148885]])

X = data[:,0]
Y = data[:,1]

def lstsq_solve(A, x):
    Q,R = la.qr(A)
    x = la.solve(R, Q.T.dot(b))
    r = A.dot(x)-b
    return x, la.norm(r)


def plot1(para):
    x = np.linspace(-1, 0)
    a,b,c,d,e = para
    A = a
    B = b*x+d
    C = c*x+e-x**2
    y = (-B+np.sqrt(B**2-4*A*C))/(2*A)
#     pyplot.plot(x,y)
    y = (-B-np.sqrt(B**2-4*A*C))/(2*A)
    plt.plot(x,y, label="model1")

def plot2(para):
    x = np.linspace(-1, 0)
    d, e = para
    y = (x**2-e)/d
    plt.plot(x,y, label="model2")

plt.plot(X, Y, label="data")

# first model x^2=ay^2+bxy+cx+dy+e
A = np.array([Y*Y, X*Y, X, Y, np.ones(len(X))]).T
b = X*X

para, r = lstsq_solve(A, b)

plot1(para)
print r

A = np.array([Y,np.ones(len(X))]).T

b = X*X

para, r = lstsq_solve(A, b)

plot2(para)
print r


plt.legend()
plt.xlabel("x value")
plt.ylabel("y value")
plt.savefig("P7.png")
```

This is the result (Indicating the residual)

```
0.00179380916718
0.343737209495
```


data
model1
model2
y value
x value
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−0.35
−0.40
−1.2
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2