EE412 Foundation of Big Data Analytics, Fall 2019
HW2

Name: Cao Viet Hai Nam

Student ID: 20200817

Discussion Group (People with whom you discussed ideas used in your answers):

On-line or hardcopy documents used as part of your answers: for the second HW problem
https://drive.google.com/drive/folders/1DqINCMOimp7pD9R_4eX52G7P5jEu-hVG?usp=sharing

Answer to Problem 1:

a/ Let us choose the following set of strings: {sacker, sacher, sacfer, sacper, sacjer, saxoxo, laxoxo}

Below is the table calulating the edit distance between pairs of string.

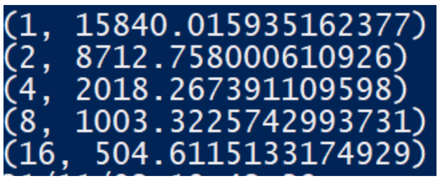
	sacker	sacher	sacfer	sacper	sacjer	saxoxo	laxoxo
sacker	0	1	1	1	1	4	5
sacher	1	0	1	1	1	4	5
sacfer	1	1	0	1	1	4	5
sacper	1	1	1	0	1	4	5
sacjer	1	1	1	1	0	4	5
saxoxo	4	4	4	4	4	0	1
laxoxo	5	5	5	5	5	1	0
Sum of distances	13	13	13	13	13	17	26

- If we choose the clustroid by minimizing the sum of the distances to the other points we can choose **sacker** (or sacher, or sacfer, or sacper, or sacjer) as clusteroid.

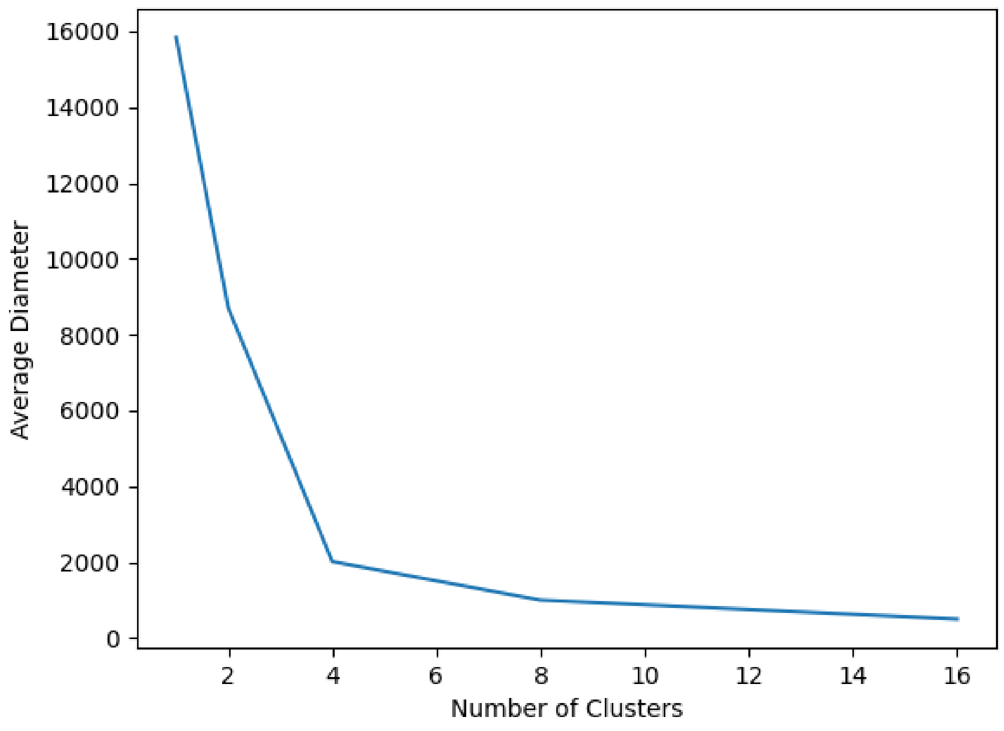
- If we choose the clustroid by minimizing the maximum distance to the other points, we can choose **saxoxo** as clusteroid (maximum distance = 4)

b/

- I run kmeans for k=1,2,4,8,16 which yields the result printed in form (k, average diameter)

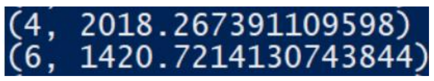


```
(1, 15840.015935162377)
(2, 8712.758000610926)
(4, 2018.267391109598)
(8, 1003.3225742993731)
(16, 504.6115133174929)
```

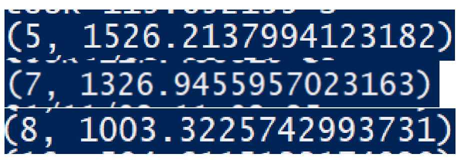


- I find two values 8 and 16 where there is little decrease in average diameter and concludes k lies between 8/2 = 4 and 8. Which means k lie in [4,5,6,7,8].

- I run kmeans with k=4,5,6,7,8 and yield:



```
(4, 2018.267391109598)
(6, 1420.7214130743844)
```



```
(5, 1526.2137994123182)
(7, 1326.9455957023163)
(8, 1003.3225742993731)
```

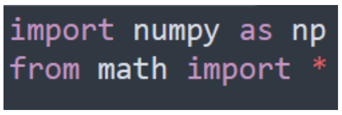
- We have (4+8)/2 = 6 , since there is not much to change between 6 and 8 ("not much to change" here is relative, here I simply realize 6->8 changes less than 6-> 4), the true k value lies between 4 and 6. We do it again, (4+6)/2 = 5 and there is not much to change between 5 and 6. Then, k lies between 4 and 5. Therefore, I choose **k=5 as the optimal value.**

Answer to Problem 2

(All the code in this problem can be find in this link: https://drive.google.com/drive/folders/1DqINCMOimp7pD9R_4eX52G7P5jEu-hVG?usp=sharing)

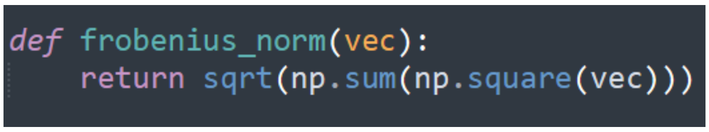
I. Exercise 11.1.7 (file hw2_2a.py in the link)

- First import numpy and math library



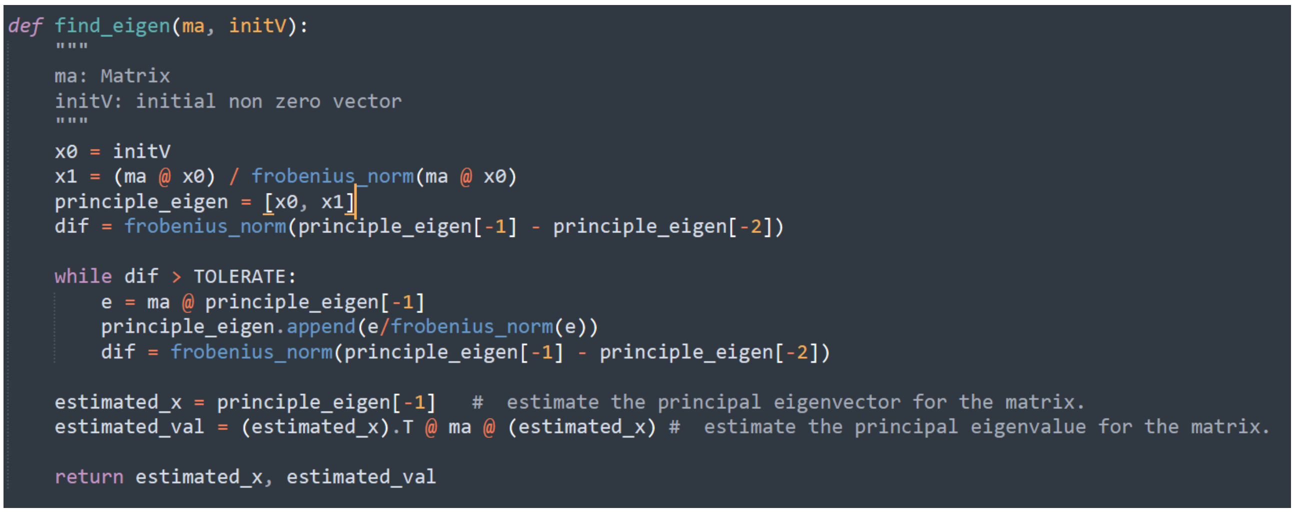
```python
import numpy as np
from math import *
```

- Define function to calculate frobenius norm of a vector.



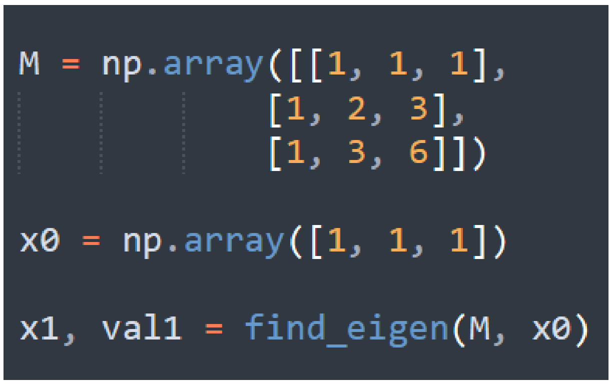
```python
def frobenius_norm(vec):
    return sqrt(np.sum(np.square(vec)))
```

- Define function to estimate eigenvectors and eigenvalues of matrix **ma** with initial vector **initV**



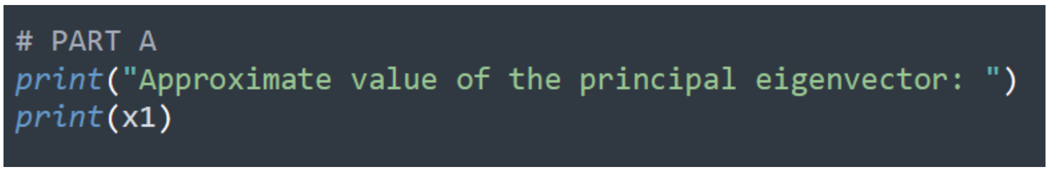
```python
def find_eigen(ma, initV):
    """
    ma: Matrix
    initV: initial non zero vector
    """
    x0 = initV
    x1 = (ma @ x0) / frobenius_norm(ma @ x0)
    principle_eigen = [x0, x1]
    dif = frobenius_norm(principle_eigen[-1] - principle_eigen[-2])

    while dif > TOLERATE:
        e = ma @ principle_eigen[-1]
        principle_eigen.append(e/frobenius_norm(e))
        dif = frobenius_norm(principle_eigen[-1] - principle_eigen[-2])

    estimated_x = principle_eigen[-1]   #  estimate the principal eigenvector for the matrix.
    estimated_val = (estimated_x).T @ ma @ (estimated_x) #  estimate the principal eigenvalue for the matrix.

    return estimated_x, estimated_val
```

a) Starting with a vector of three 1's, use power iteration to find an approximate value of the principal eigenvector.

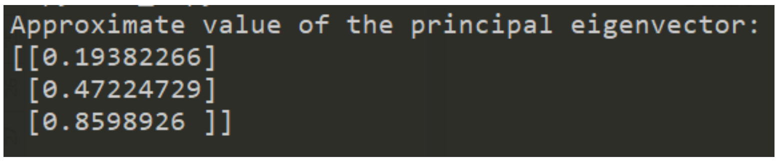


```python
M = np.array([[1, 1, 1],
              [1, 2, 3],
              [1, 3, 6]])

x0 = np.array([1, 1, 1])

x1, val1 = find_eigen(M, x0)
```

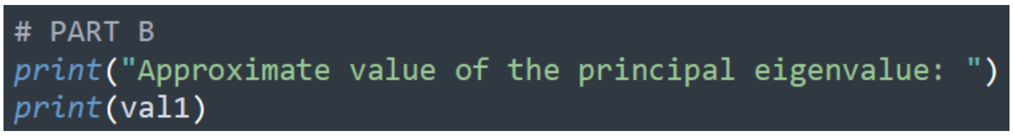
- M is the given matrix, x0 is initial vector.
- Using find_eigen function we have x1,val1 which is an approximate value of principal eigenvector.


```python
# PART A
print("Approximate value of the principal eigenvector: ")
print(x1)
```

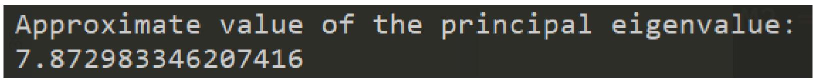
RESULT:


```
Approximate value of the principal eigenvector:
[[0.19382266]
 [0.47224729]
 [0.8598926 ]]
```

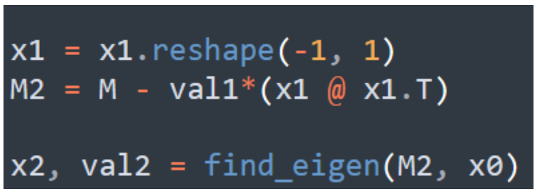
(b) Compute an estimate the principal eigenvalue for the matrix.


```python
# PART B
print("Approximate value of the principal eigenvalue: ")
print(val1)
```

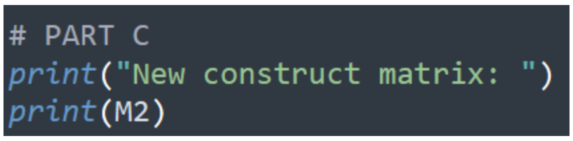
RESULT:


```
Approximate value of the principal eigenvalue:
7.872983346207416
```

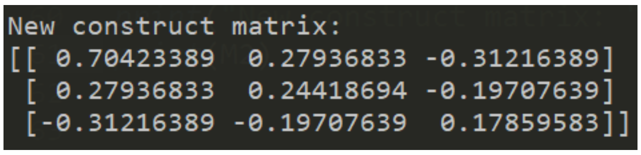
(c) Construct a new matrix by subtracting out the effffect of the principal eigenpair, as in Section 11.1.3.


```python
x1 = x1.reshape(-1, 1)
M2 = M - val1*(x1 @ x1.T)

x2, val2 = find_eigen(M2, x0)
```

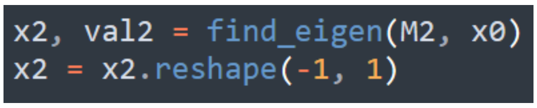
- Construct new matrix M2 and substracting out the effect of principle eigenpair (x1, val1)

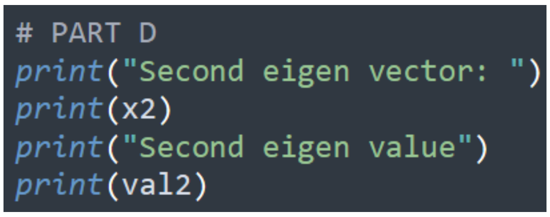

```python
# PART C
print("New construct matrix: ")
print(M2)
```

RESULT


```
New construct matrix:
[[ 0.70423389  0.27936833 -0.31216389]
 [ 0.27936833  0.24418694 -0.19707639]
 [-0.31216389 -0.19707639  0.17859583]]
```

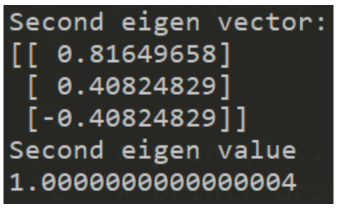
(d) From your matrix of (c), find the second eigenpair for the original matrix of Exercise 11.1.5.


```python
x2, val2 = find_eigen(M2, x0)
x2 = x2.reshape(-1, 1)
```



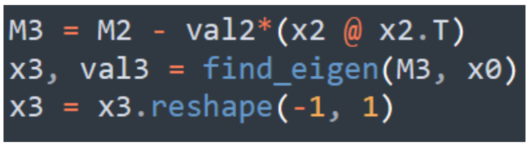
```
# PART D
print("Second eigen vector: ")
print(x2)
print("Second eigen value")
print(val2)
```

<u>RESULT</u>

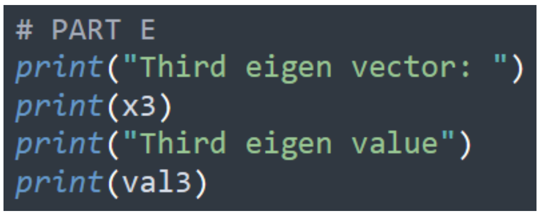


```
Second eigen vector:
[[ 0.81649658]
 [ 0.40824829]
 [-0.40824829]]
Second eigen value
1.0000000000000004
```

(e) Repeat (c) and (d) to find the third eigenpair for the original matrix.

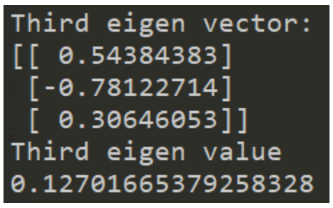


```
M3 = M2 - val2*(x2 @ x2.T)
x3, val3 = find_eigen(M3, x0)
x3 = x3.reshape(-1, 1)
```



```
# PART E
print("Third eigen vector: ")
print(x3)
print("Third eigen value")
print(val3)
```

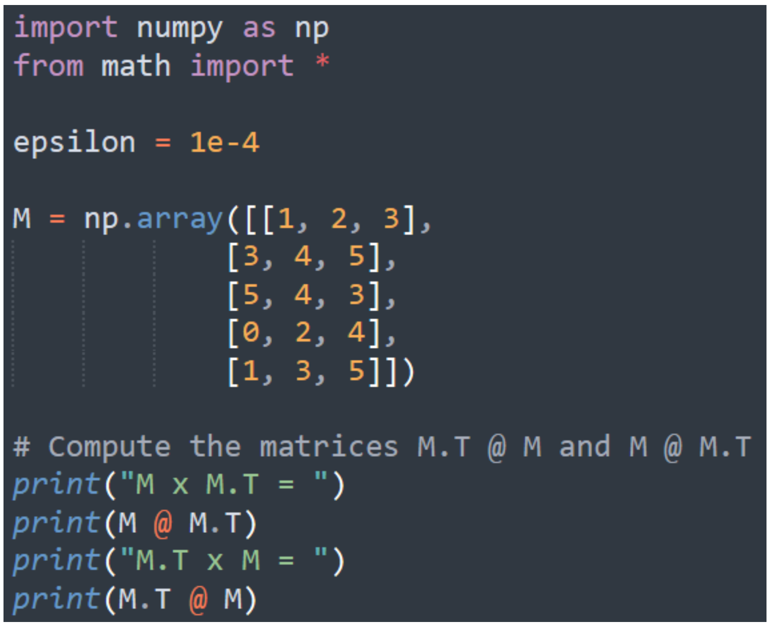
<u>RESULT:</u>



```
Third eigen vector:
[[ 0.54384383]
 [-0.78122714]
 [ 0.30646053]]
Third eigen value
0.12701665379258328
```

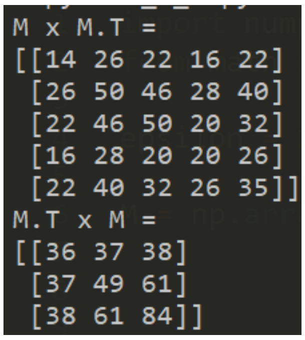
II. <u>Exercise 11.3.1 (file hw2_2b.py in the link)</u>

(a) Compute the matrices *MTM* and *MMT* .



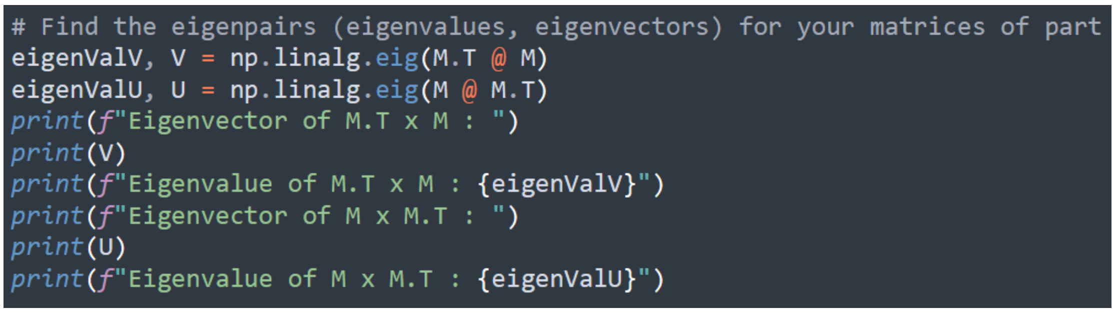
```python
import numpy as np
from math import *

epsilon = 1e-4

M = np.array([[1, 2, 3],
              [3, 4, 5],
              [5, 4, 3],
              [0, 2, 4],
              [1, 3, 5]])

# Compute the matrices M.T @ M and M @ M.T
print("M x M.T = ")
print(M @ M.T)
print("M.T x M = ")
print(M.T @ M)
```

RESULTS:



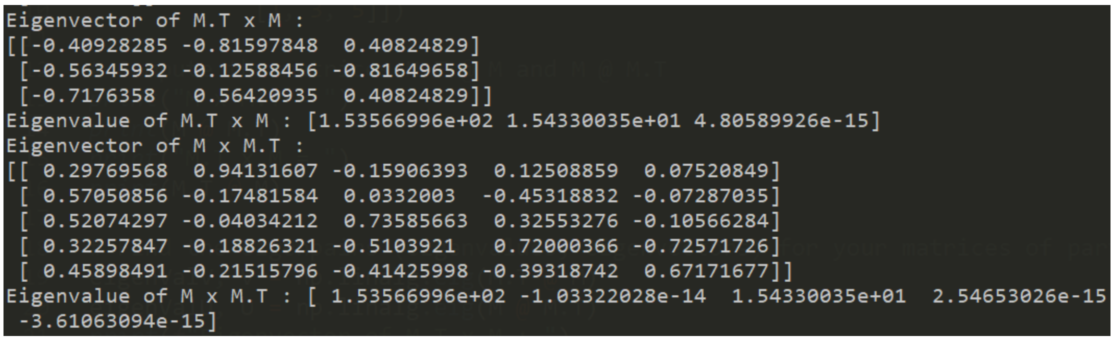
```
M x M.T =
[[14 26 22 16 22]
 [26 50 46 28 40]
 [22 46 50 20 32]
 [16 28 20 20 26]
 [22 40 32 26 35]]
M.T x M =
[[36 37 38]
 [37 49 61]
 [38 61 84]]
```

(b) Find the eigenpairs (eigenvalues, eigenvectors) for your matrices of part (a) using Python NumPy function (numpy.linalg.eig()).



```python
# Find the eigenpairs (eigenvalues, eigenvectors) for your matrices of part
eigenValV, V = np.linalg.eig(M.T @ M)
eigenValU, U = np.linalg.eig(M @ M.T)
print(f"Eigenvector of M.T x M : ")
print(V)
print(f"Eigenvalue of M.T x M : {eigenValV}")
print(f"Eigenvector of M x M.T : ")
print(U)
print(f"Eigenvalue of M x M.T : {eigenValU}")
```

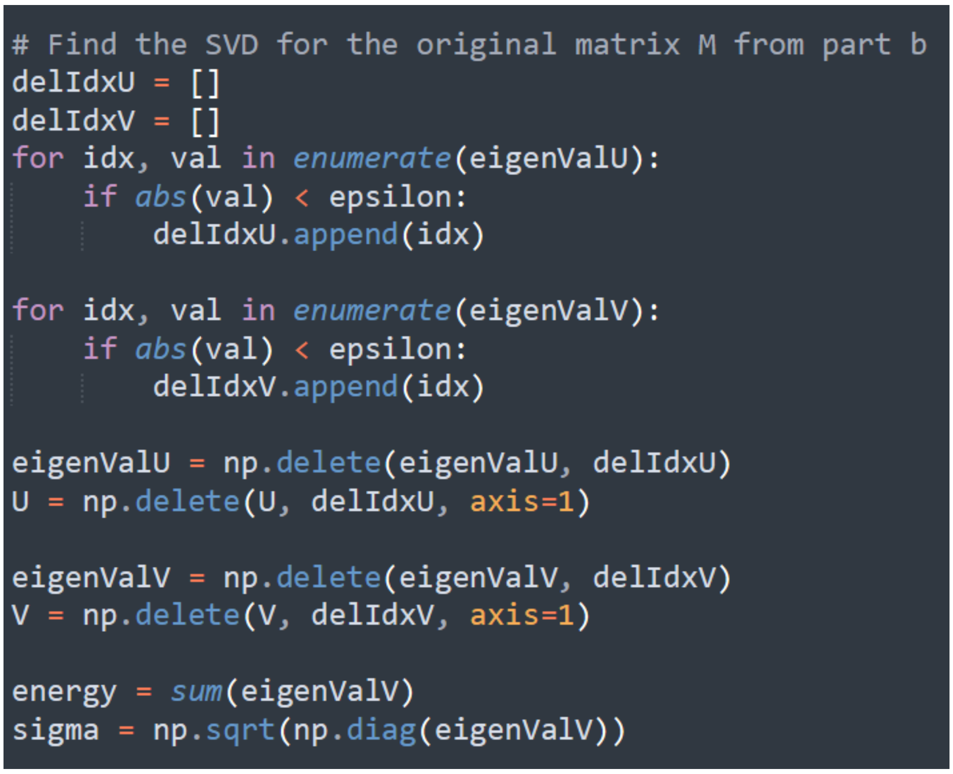
RESULTS:



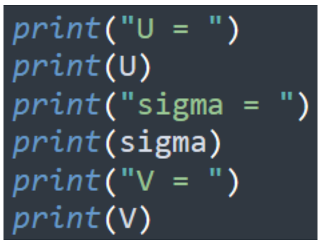
```
Eigenvector of M.T x M :
[[-0.40928285 -0.81597848  0.40824829]
 [-0.56345932 -0.12588456 -0.81649658]
 [-0.7176358   0.56420935  0.40824829]]
Eigenvalue of M.T x M : [1.53566996e+02 1.54330035e+01 4.80589926e-15]
Eigenvector of M x M.T :
[[ 0.29769568  0.94131607 -0.15906393  0.12508859  0.07520849]
 [ 0.57050856 -0.17481584  0.0332003  -0.45318832 -0.07287035]
 [ 0.52074297 -0.04034212  0.73585663  0.32553276 -0.10566284]
 [ 0.32257847 -0.18826321 -0.5103921   0.72000366 -0.72571726]
 [ 0.45898491 -0.21515796 -0.41425998 -0.39318742  0.67171677]]
Eigenvalue of M x M.T : [ 1.53566996e+02 -1.03322028e-14  1.54330035e+01  2.54653026e-15
 -3.61063094e-15]
```

(c) Find the SVD for the original matrix *M* from parts (b). Note that there are only two nonzero eigenvalues, so your matrix Σ should have only two singular values, while *U* and *V* have only two columns.

Since sigma should have only two singular value, I delete the smallest (close to 0) eigenvalue and the corresponding column in U and V. The result left with sigma having 2 singular values, while U, V have 2 columns.

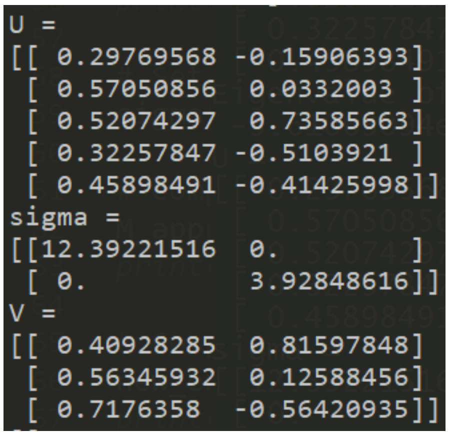


```
# Find the SVD for the original matrix M from part b
delIdxU = []
delIdxV = []
for idx, val in enumerate(eigenValU):
    if abs(val) < epsilon:
        delIdxU.append(idx)

for idx, val in enumerate(eigenValV):
    if abs(val) < epsilon:
        delIdxV.append(idx)

eigenValU = np.delete(eigenValU, delIdxU)
U = np.delete(U, delIdxU, axis=1)

eigenValV = np.delete(eigenValV, delIdxV)
V = np.delete(V, delIdxV, axis=1)

energy = sum(eigenValV)
sigma = np.sqrt(np.diag(eigenValV))
```



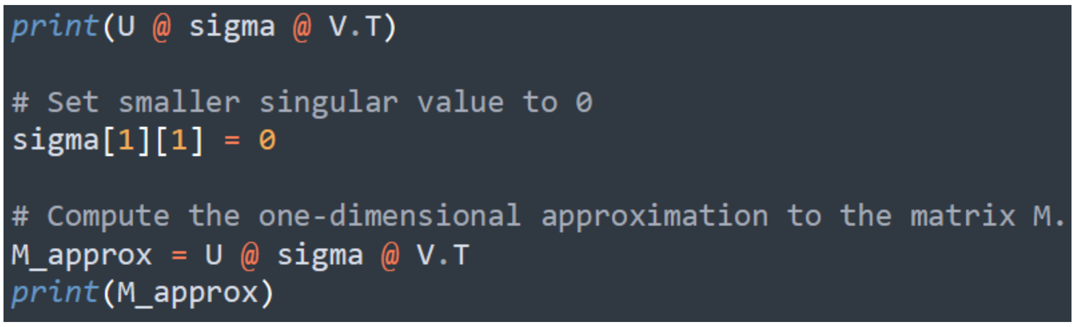
```
print("U = ")
print(U)
print("sigma = ")
print(sigma)
print("V = ")
print(V)
```

RESULTS:



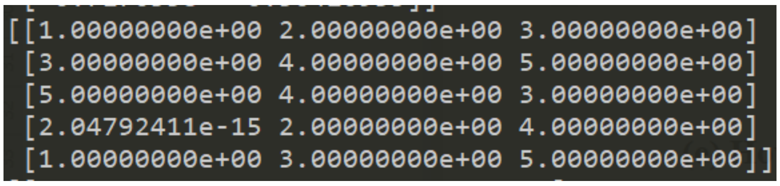
```
U =
[[ 0.29769568 -0.15906393]
 [ 0.57050856  0.0332003 ]
 [ 0.52074297  0.73585663]
 [ 0.32257847 -0.5103921 ]
 [ 0.45898491 -0.41425998]]
sigma =
[[12.39221516  0.        ]
 [ 0.          3.92848616]]
V =
[[ 0.40928285  0.81597848]
 [ 0.56345932  0.12588456]
 [ 0.7176358  -0.56420935]]
```

(d) Set your smaller singular value to 0 and compute the one-dimensional approximation to the matrix *M*.



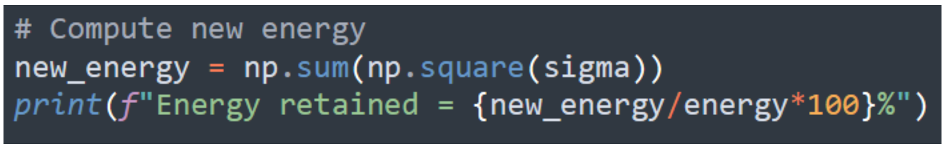
```
print(U @ sigma @ V.T)

# Set smaller singular value to 0
sigma[1][1] = 0

# Compute the one-dimensional approximation to the matrix M.
M_approx = U @ sigma @ V.T
print(M_approx)
```

RESULTS:



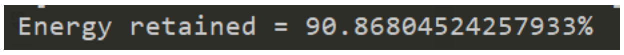
```
[[1.00000000e+00 2.00000000e+00 3.00000000e+00]
 [3.00000000e+00 4.00000000e+00 5.00000000e+00]
 [5.00000000e+00 4.00000000e+00 3.00000000e+00]
 [2.04792411e-15 2.00000000e+00 4.00000000e+00]
 [1.00000000e+00 3.00000000e+00 5.00000000e+00]]
```

(e) How much of the energy of the original singular values is retained by the onedimensional approximation? (Hint: energy = sum of the squares of the singular values)



```
# Compute new energy
new_energy = np.sum(np.square(sigma))
print(f"Energy retained = {new_energy/energy*100}%")
```

RESULTS:



```
Energy retained = 90.86804524257933%
```

Answer to Problem 3

I.

3 Recommendation systems

I) Exercise 9.3.1

a)

	a	b	c	d	e	f	g	h
A	1	1	0	1	1	0	1	1
B	0	1	1	1	1	1	1	0
C	1	0	1	1	0	1	1	1

$$\text{Jacc}(A,B) = 1 - \frac{4}{8} = \frac{1}{2}$$

$$\text{Jacc}(A,C) = 1 - \frac{4}{8} = \frac{1}{2}$$

$$\text{Jacc}(B,C) = 1 - \frac{4}{8} = \frac{1}{2}$$

b) $$\text{cosine}(A,B) = \frac{5\times 3 + 5\times 3 + 1\times 1 + 3\times 1}{\sqrt{4^2+5^2+5^2+1^2+3^2+2^2}\,\sqrt{3^2+4^2+3^2+1^2+2^2+1^2}}$$

$$= \frac{34}{\sqrt{80}\cdot 2\sqrt{10}} = \frac{17\sqrt{2}}{40} \approx 0{,}601$$

$$\text{cosine}(A,C) = \frac{4\times 2 + 5\times 3 + 3\times 5 + 2\times 3}{\sqrt{4^2+5^2+5^2+1^2+3^2+2^2}\,\sqrt{2^2+1^2+3^2+4^2+5^2+3^2}}$$

$$= \frac{44}{\sqrt{80}\cdot 8} = \frac{11\sqrt{5}}{40} \approx 0{,}615$$

$$\text{cosine}(B,C) = \frac{4\times1 + 3\times3 + 2\times4 + 1\times5}{\sqrt{3^2+4^2+3^2+1^2+2^2+1^2}\,\sqrt{2^2+1^2+3^2+4^2+5^2+3^2}}$$

$$= \frac{26}{2\sqrt{10}\,.\,8} = \frac{13\sqrt{10}}{80} \approx 0{,}514$$

c)

	a	b	c	d	e	f	g	h
A	1	1	0	1	0	0	1	0
B	0	1	1	1	0	0	0	0
C	0	0	0	1	0	1	1	1

$$\text{Jacc}(A,B) = 1 - \frac{2}{5} = \frac{3}{5}$$

$$\text{Jacc}(A,C) = 1 - \frac{2}{6} = \frac{2}{3}$$

$$\text{Jacc}(B,C) = 1 - \frac{1}{6} = \frac{5}{6}$$

d) $$\text{cosine}(A,B) = \frac{1\times1 + 1\times1}{\sqrt{1^2+1^2+0^2+1^2+0^2+0^2+1^2+0^2}\,\sqrt{0^2+1^2+1^2+1^2+0^2+0^2+0^2+0^2}}$$

$$= \frac{2}{\sqrt{4}\,.\sqrt{3}} = \frac{\sqrt{3}}{3}$$

$$\text{cosine}(A,C) = \frac{1\times1 + 1\times1}{\sqrt{1^2+1^2+0^2+1^2+0^2+0^2+1^2+0^2}\,\sqrt{0^2+0^2+0^2+1^2+0^2+1^2+1^2+1^2}}$$

$$= \frac{2}{4} = \frac{1}{2}$$

$$\text{cosine}(B,C) = \frac{1 \times 1}{\sqrt{0^2+1^2+1^2+1^2+0^2+0^2+0^2+0^2}\sqrt{0^2+0^2+0^2+1^2+0^2+1^2+1^2+1^2}}$$

$$= \frac{1}{\sqrt{3}\cdot\sqrt{4}} = \frac{\sqrt{3}}{6}.$$

e)

	a	b	c	d	e	f	g	h	AVG
A	4	5		5	1		3	2	10/3
B		3	4	3	1	2	1		7/3
C	2		1	3		4	5	3	3

$\Downarrow$ Subtracting the Avg value

	a	b	c	d	e	f	g	h
A	$\frac{2}{3}$	$\frac{5}{3}$		$\frac{5}{3}$	$-\frac{7}{3}$		$-\frac{1}{3}$	$-\frac{4}{3}$
B		$\frac{2}{3}$	$\frac{5}{3}$	$\frac{2}{3}$	$-\frac{4}{3}$	$-\frac{1}{3}$	$-\frac{4}{3}$	
C	-1		-2	0		1	2	0

f) $$\text{norm}(A) = \sqrt{\left(\frac{2}{3}\right)^2+\left(\frac{5}{3}\right)^2+\left(\frac{5}{3}\right)^2+\left(-\frac{7}{3}\right)^2+\left(-\frac{1}{3}\right)^2+\left(-\frac{4}{3}\right)^2}$$

$$= \sqrt{\frac{40}{3}}$$

$$\text{norm}(B) = \sqrt{\left(\frac{2}{3}\right)^2+\left(\frac{5}{3}\right)^2+\left(\frac{2}{3}\right)^2+\left(\frac{-4}{3}\right)^2+\left(\frac{-1}{3}\right)^2+\left(\frac{-4}{3}\right)^2}$$

$$= \sqrt{\frac{22}{3}}$$

$$\text{norm}(C) = \sqrt{(-1)^2 + (-2)^2 + 1^2 + 2^2}$$
$$= \sqrt{10}$$

Cosine (A, B)

$$= \frac{\frac{5}{3} \times \frac{2}{3} + \frac{5}{3} \times \frac{2}{3} + \left(-\frac{1}{3}\right) \times \left(-\frac{4}{3}\right) + \left(-\frac{1}{3}\right) \times \left(-\frac{4}{3}\right)}{\text{norm}(A) \times \text{norm}(B)}$$

$$= \frac{52/9}{\sqrt{\frac{40}{3}} \times \sqrt{\frac{22}{3}}} = \frac{13}{3\sqrt{55}} \approx 0{,}584$$

cosine (A, C)

$$= \frac{\frac{2}{3} \times (-1) + \left(-\frac{1}{3}\right) \times 2}{\text{norm}(A) \times \text{norm}(C)}$$

$$= \frac{-4/3}{\sqrt{\frac{40}{3}}\ \sqrt{10}} \approx -0{,}1155$$

$$\text{cosince}(B, C) = \frac{\frac{5}{3} \times (-2) + \left(-\frac{1}{3}\right) \times 1 + \left(-\frac{4}{3}\right) \times 2}{\text{norm}(B) \times \text{norm}(C)}$$

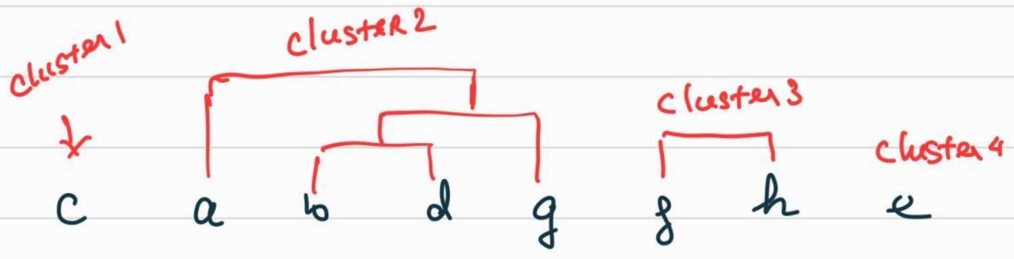
$$= \frac{-\frac{19}{3}}{\sqrt{\frac{22}{3}} \times \sqrt{10}} = \frac{-19}{2\sqrt{165}} \approx -0{,}739$$

II. Exercise 9.3.2

a)

	a	b	c	d	e	f	g	h
A	1	1	0	1	0	0	1	0
B	0	1	1	1	0	0	0	0
C	0	0	0	1	0	1	1	1

Jaccard distance	a	b	c	d	e	f	g	h
a	1	1/2	1	2/3	1	1	1/2	1
b	1/2	1	1/2	1/3	1	1	2/3	1
c	1	1/2	1	2/3	1	1	1	1
d	2/3	1/3	2/3	1	1	2/3	1/3	2/3
e	1	1	1	1	1	1	1	1
f	1	1	1	2/3	1	1	1/2	0
g	1/2	2/3	1	1/3	1	1/2	1	1/2
h	1	1	1	2/3	1	0	1/2	1



cluster / user	1	2	3	4
A		4,25	2	1
B	4	$\frac{7}{3}$	2	1
C	1	$\frac{10}{3}$	3,5	

$$\text{norm}(A) = \sqrt{4{,}25^2 + 2^2 + 1^2} = \frac{3\sqrt{41}}{4}$$

$$\text{norm}(B) = \sqrt{4^2 + \left(\frac{7}{3}\right)^2 + 2^2 + 1^2} = \frac{\sqrt{238}}{3}$$

$$\text{norm}(C) = \sqrt{1^2 + \left(\frac{10}{3}\right)^2 + 3{,}5^2} = \frac{\sqrt{877}}{6}$$

- cosine(A, B)

$$= \frac{\frac{7}{3} \times 4{,}25 + 2\times 2 + 1\times 1}{\text{norm}(A) \cdot \text{norm}(B)}$$

$$= \frac{\frac{179}{12}}{\frac{3\sqrt{41}}{4} \cdot \frac{\sqrt{238}}{3}} \approx 0{,}604$$

- cosine(B, C)

$$= \frac{4\times 1 + \frac{7}{3} \times \frac{10}{3} + 2\times 3{,}5}{\text{norm}(B)\ \text{norm}(C)}$$

$$= \frac{\frac{169}{9}}{\frac{\sqrt{238}}{3} \cdot \frac{\sqrt{877}}{6}} \approx 0{,}739$$

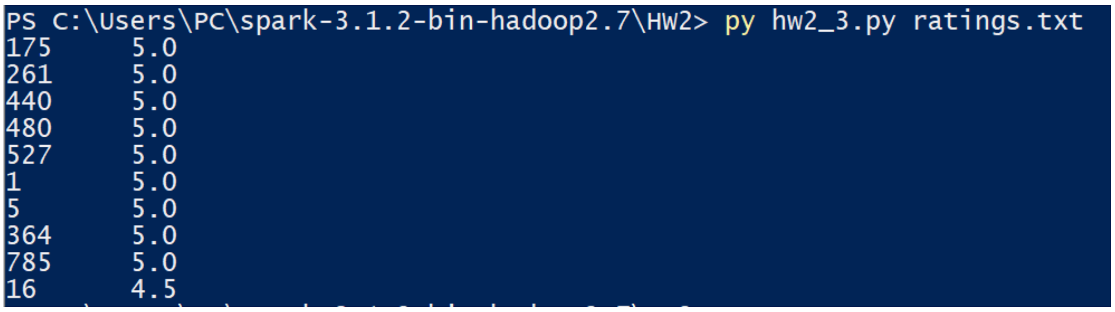
- cosine (A, C)

$$= \frac{4{,}25 \times \frac{10}{3} + 2 \times 3{,}5}{\text{norm}(A)\ \text{norm}(C)}$$

$$= \frac{\frac{127}{6}}{\frac{3\sqrt{41}}{4} \cdot \frac{\sqrt{877}}{6}} \approx 0{,}893$$

II. 3b

Below is the result with **the first 5 line** is recommendation using **user-based method** and **the last 5 line** is recommendation using **item-based method**



```
PS C:\Users\PC\spark-3.1.2-bin-hadoop2.7\HW2> py hw2_3.py ratings.txt
175     5.0
261     5.0
440     5.0
480     5.0
527     5.0
1       5.0
5       5.0
364     5.0
785     5.0
16      4.5
```

III. 3c

Note: If you run the code again, then you should delete the existed output.txt file.

I used UV decomposition in this exercise:

- **Performing the optimization**

With the initialization of U and V, I calculate the square error of their product is to M, and then try to minimize it by gradient descent.

To minimize the error, gradient can be obtained by differentiating the equation with respect to these two variables separately

Having obtained the gradient, we can now formulate the update rules of gradient descent with learning rate initialized from the start. I choose a small value for learning rate, about 0.0002 to gradually reach the local minimum.

I also introduce regularization to avoid overfitting by adding a parameter beta and modifying the squared error which in turns modifying the update rules.

Iteratively perform the operation until the error converges to its minimum.

- **Multipling updated U and V to get approximate matrix.**