Activity 10, ECE 532, Ayandeep Hazra

1. a) (In the pdf)

b) (In the pdf)

c) We see that the Rank 1 approximation is not only is it very far-off from the value of the centroid, but also does not capture the correct sign.

For columns (1 to 3) the true value of the centroid is 2, but the Rank-1 approx gives us 1.

For Columns (4 to 6), the true value of the centroid is -2, but the Rank-1 approx gives us 1 again (WRONG SIGN)

d) We see that the Rank 2 approximation is a little better as it includes the correct sign of the centroid for all columns in A, even the ones that were missed.

2. a) A is a 4×6 matrix

Thus as $A = USV^T$

U has dimensions 4×4

S has dimensions 4×6

V has dimensions 6×6.

b) In the skinny case of SVD,

for $A = USV^T$

U has dimensions 4×4

S has dimensions 4×4

V has dimensions 6×4

c) i) We see from the code that

$A = USV^T$

ii) We see that $U^TU = I$, thus cols of U is orthonormal by definition

Also, $V^TV = I$, thus V's cols. orthonormal by definition.

iii) Again $UU^T = I$ and $VV^T = 1$,
thus the rows of U & V are
orthonormal by definition.

iv) First left singular vector

$$\begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix}$$

Largest singular value

$= 9.7979$

v) Rank $\{A\} = 2$

d) i) From the code, we see that
$A = USV^T$ holds.

ii) As before $U^TU = I$ & $V^TU = I$, thus
the columns of U, V are orthonormal.

iii) $UU^T = I$ & $VV^T = I$, thus the
rows of U, V are orthonormel.

e) we see that even for the skinny case,

first left singular vector = $\begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix}$

& largest singular value $= 9.7979$

Thus the values are the same. We expect this as theres only one unique way to $\overset{\text{eigen}}{\text{decompose}}$ a matrix, regardless of the approach taken.

f) In economy SVD, $A = USV^T$, where $A \in \mathbb{R}^{m \times n}$, $U \in \mathbb{R}^{m \times r}$, $S \in \mathbb{R}^{r \times r}$, $V^T \in \mathbb{R}^{r \times m}$, and r is the rank of A.

If we let $B = SV^T$

Thus $A = UB$ so that each column of A is a linear combination of the rows of U.

cols(A) is in span of cols(U)

Scanned with CamScanner

since U is orthonormal.
thus the first r columns of U form an orthonormal basis for the space spanned by the cols of A, r depends on rank approximation. thus orthonormal bas is

$$= \begin{bmatrix} -0.5 & -0.5 \\ -0.5 & 0.5 \\ -0.5 & 0.5 \\ -0.5 & -0.5 \end{bmatrix}$$

9) similarly for $\boxed{A = USV^T}$

let $B = US$

thus $A = BV^T$

& since V is orthonormal, V^T is orthonormal by definition.

Thus each row of A is a linear comb. of the columns of V^T.

Thus given some B, first r rows of V^T acts also the orthonormal basis for the space spanned by the rows of A. r depends on rank approx.

Thus, orthonormal basis is
$$\begin{bmatrix} -0.5 & -0.5 \\ -0.5 & 0.5 \\ -0.71 & -0.03 \\ -0.03 & 0.71 \end{bmatrix}$$

Scanned with CamScanner

h) i) The rank 1 approx generally gets the notion of a line that divides the +ve values from the -ve values in the matrix. But it gets all the values as the same absolute value, + or -.

ii) The rank 2 approximation correctly defines A exactly. (All values in all rows & columns).

i) Since A has dimensions 4×6

S can have minimum dimensions of 4×4

where $\left\{ A_{4\times6} = U_{4\times4} \, S_{4\times4} \, V^T_{4\times6} \right\}$

CS/ECE/ME532 Period 10 Activity

Estimated Time:

P1: 25 mins

P2: 25 mins

Preambles

In [1]:

```
import numpy as np # numpy
from scipy.io import loadmat # load & save data
from scipy.io import savemat
import matplotlib.pyplot as plt # plot
np.set_printoptions(formatter={'float': lambda x: "{0:0.2f}".format(x)})
```

Q1. K-means

Let $A = \begin{bmatrix} 3 & 3 & 3 & -1 & -1 & -1 \\ 1 & 1 & 1 & -3 & -3 & -3 \\ 1 & 1 & 1 & -3 & -3 & -3 \\ 3 & 3 & 3 & -1 & -1 & -1 \end{bmatrix}$. Use the provided script to help you complete the problem.

In [2]:

```
A = np.array([[3,3,3,-1,-1,-1],[1,1,1,-3,-3,-3],[1,1,1,-3,-3,-3],[3,3,3,-1,-1,-1]], float)
rows, cols = A.shape
print('A = \n', A)
```

```
A =
 [[3.00 3.00 3.00 -1.00 -1.00 -1.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [3.00 3.00 3.00 -1.00 -1.00 -1.00]]
```

In [3]:

```
# numpy iterates over the 0th dimension first (over the rows)

for each_entry in A:
    print(each_entry) # This prints iterates the "rows" of A

for each_entry in A.transpose():
    print(each_entry) # This prints iterates the "columns" of A
```

```
[3.00 3.00 3.00 -1.00 -1.00 -1.00]
[1.00 1.00 1.00 -3.00 -3.00 -3.00]
[1.00 1.00 1.00 -3.00 -3.00 -3.00]
[3.00 3.00 3.00 -1.00 -1.00 -1.00]
[3.00 1.00 1.00 3.00]
[3.00 1.00 1.00 3.00]
[3.00 1.00 1.00 3.00]
[-1.00 -3.00 -3.00 -1.00]
[-1.00 -3.00 -3.00 -1.00]
[-1.00 -3.00 -3.00 -1.00]
```

a) Understand the following implementation of the k-means algorithm and fill in the blank to define the distance function.

In [4]:

```
def dist(x, y):
    """
    this function takes in two 1-d numpy as input an outputs
    Euclidean the distance between them
    """
    return np.sqrt((x-y).T@(x-y))## Fill in the blank: Recall the 'distance' function used

def kMeans(X, K, maxIters = 20):
    """
    this implementation of k-means takes as input (i) a matrix X
    (with the data points as columns) (ii) an integer K representing the number
    of clusters, and returns (i) a matrix with the K columns representing
    the cluster centers and (ii) a list C of the assigned cluster centers
    """
    X_transpose = X.transpose()
    centroids = X_transpose[np.random.choice(X.shape[0], K)]
    for i in range(maxIters):
        # Cluster Assignment step
        C = np.array([np.argmin([dist(x_i, y_k) for y_k in centroids]) for x_i in X_transpo
        #np.array([np.argmin([dist(x_k,y_k) for y_k in centroids]) for x_i in X_transpose])
        # Update centroids step
        for k in range(K):
            if (C == k).any():
                centroids[k] = X_transpose[C == k].mean(axis = 0)
            else: # if there are no data points assigned to this certain centroid
                centroids[k] = X_transpose[np.random.choice(len(X))]
    return centroids.transpose() , C
```

b) Use the K-means algorithm to represent the columns of A with a single cluster.

In [5]:

```
# k-means with 1 cluster
centroids, C =  kMeans(A, 1)## Fill in the blank: call the "kMeans" algorithm with proper i
print('A = \n', A)
print('centroids = \n', centroids)
print('centroid assignment = \n', C)
```

```
A =
 [[3.00 3.00 3.00 -1.00 -1.00 -1.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [3.00 3.00 3.00 -1.00 -1.00 -1.00]]
centroids =
 [[1.00]
 [-1.00]
 [-1.00]
 [1.00]]
centroid assignment =
 [0 0 0 0 0 0]
```

c) Construct a matrix $\hat{A}_{r=1}$ whose i-th column is the centroid corresponding to the i-th column of A. Note that this can be viewed as a rank-1 approximation to A. Compare the rank-1 approximation to the original matrix and explain the nature of the approximation in terms of the properties of the K-means algorithm.

In [6]:

```
# Construct rank-1 approximation using cluster
centroids_transposed = centroids.transpose() # transpose "centroids" to iterate over column
A_hat_1 = centroids.transpose()[C,C] # Fill in the blank: pick the columns of centroids ind
print('Rank-1 Approximation, \n A_hat_1 = \n', A_hat_1)
```

```
Rank-1 Approximation,
 A_hat_1 =
 [1.00 1.00 1.00 1.00 1.00 1.00]
```

d) Repeat b) and c) with $K = 2$. Compare the rank-2 approximation to the original matrix and explain the nature of the approximation in terms of the properties of the K-means algorithm.

In [7]:

```
# k-means with 2 cluster
centroids, C = kMeans(A, 2) ## Fill in the blank: call the "kMeans" method with proper inpu
print('A = \n', A)
print('centroids = \n', centroids)
print('centroid assignment = \n', C)
centroids_transposed = centroids.transpose() # transpose "centroids" to iterate over column
A_hat_2 = centroids.transpose()[C,C] # Fill in the blank: pick the columns of centroids ind
print('Rank-2 Approximation \n', A_hat_2)
```

```
A =
 [[3.00 3.00 3.00 -1.00 -1.00 -1.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [3.00 3.00 3.00 -1.00 -1.00 -1.00]]
centroids =
 [[3.00 -1.00]
 [1.00 -3.00]
 [1.00 -3.00]
 [3.00 -1.00]]
centroid assignment =
 [0 0 0 1 1 1]
Rank-2 Approximation
 [3.00 3.00 3.00 -3.00 -3.00 -3.00]
```

In [8]:

```
# Write code to compare A_hat_1 and A_hat_2 to the original matrix A
```

Q2. SVD

Again let $A = \begin{bmatrix} 3 & 3 & 3 & -1 & -1 & -1 \\ 1 & 1 & 1 & -3 & -3 & -3 \\ 1 & 1 & 1 & -3 & -3 & -3 \\ 3 & 3 & 3 & -1 & -1 & -1 \end{bmatrix}$. Now consider the singular value decomposition (SVD) $A = USV^T$

a) If the full SVD is computed, find the dimensions of U, S, and V.

b) Find the dimensions of U, S, and V in the economy or skinny SVD of A.

c) The Python and NumPy command `U, s, VT = np.linalg.svd(A, full_matrices=True)` computes the singular value decomposition, $A = USV^T$ where U and V are matrices with orthonormal columns comprising the left and right singular vectors and S is a diagonal matrix of singular values.

i. Compute the SVD of A. Make sure $A = USV^T$ holds.

ii. Find U^TU and V^TV. Are the columns of U and V orthonormal? Why? *Hint:* compute U^TU.

iii. Find UU^T and VV^T. Are the rows of U and V orthonormal? Why?

iv. Find the left and right singular vectors associated with the largest singular value.

v. What is the rank of A?

In [9]:

```
# i)
U, s, VT = np.linalg.svd(A, full_matrices=True)
S_matrix = np.zeros_like(A) ## Fill in the blank: Size of S should be equal to size of ???
np.fill_diagonal(S_matrix, s) ## Fill in the diagonal entries of S_matrix with ???
print(U@S_matrix@VT)
print(A)
```

```
[[3.00 3.00 3.00 -1.00 -1.00 -1.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [3.00 3.00 3.00 -1.00 -1.00 -1.00]]
[[3.00 3.00 3.00 -1.00 -1.00 -1.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [3.00 3.00 3.00 -1.00 -1.00 -1.00]]
```

In [10]:

```python
# ii)
print('UTU: \n', U@U.T) # i. Printing U^T*U
print('VTV: \n', VT@VT.T) # i. Printing V^T*V

# iii)
print('UUT: \n', U.T@U) # i. Printing U*U^T
print('VVT: \n', VT.T@VT) # i. Printing V*V^T

# iv)
print('First left singular vector: \n', U[:,[0]])
print('Largest singular value:', s[0])

# v)
print(np.sum(np.abs(s)>1e-6))
```

```
UTU:
 [[1.00 -0.00 -0.00 -0.00]
 [-0.00 1.00 -0.00 -0.00]
 [-0.00 -0.00 1.00 -0.00]
 [-0.00 -0.00 -0.00 1.00]]
VTV:
 [[1.00 0.00 -0.00 0.00 -0.00 0.00]
 [0.00 1.00 -0.00 -0.00 -0.00 -0.00]
 [-0.00 -0.00 1.00 -0.00 -0.00 -0.00]
 [0.00 -0.00 -0.00 1.00 -0.00 -0.00]
 [-0.00 -0.00 -0.00 -0.00 1.00 0.00]
 [0.00 -0.00 -0.00 -0.00 0.00 1.00]]
UUT:
 [[1.00 -0.00 0.00 -0.00]
 [-0.00 1.00 -0.00 0.00]
 [0.00 -0.00 1.00 -0.00]
 [-0.00 0.00 -0.00 1.00]]
VVT:
 [[1.00 0.00 -0.00 -0.00 -0.00 -0.00]
 [0.00 1.00 0.00 -0.00 -0.00 -0.00]
 [-0.00 0.00 1.00 0.00 -0.00 -0.00]
 [-0.00 -0.00 0.00 1.00 -0.00 -0.00]
 [-0.00 -0.00 -0.00 -0.00 1.00 -0.00]
 [-0.00 -0.00 -0.00 -0.00 -0.00 1.00]]
First left singular vector:
 [[-0.50]
 [-0.50]
 [-0.50]
 [-0.50]]
Largest singular value: 9.797958971132713
2
```

d) The Python and NumPy command `U, s, VT = np.linalg.svd(A, full_matrices=False)` computes the economy or skinny singular value decomposition, $A = USV^T$ where U and V are matrices with orthonormal columns comprising the left and right singular vectors and S is a square diagonal matrix of singular values.

i. Compute the SVD of A. Make sure $A = USV^T$ holds.

ii. Find $U^T U$ and $V^T V$. Are the columns of U and V orthonormal? Why? *Hint:* compute $U^T U$.

iii. Find UU^T and VV^T. Are the rows of U and V orthonormal? Why?

In [11]:

```
# i)
U, s, VT = np.linalg.svd(A, full_matrices=False)
S_matrix = np.diag(s)
print(U@S_matrix@VT)
print(A)
```

```
[[3.00 3.00 3.00 -1.00 -1.00 -1.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [3.00 3.00 3.00 -1.00 -1.00 -1.00]]
[[3.00 3.00 3.00 -1.00 -1.00 -1.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [1.00 1.00 1.00 -3.00 -3.00 -3.00]
 [3.00 3.00 3.00 -1.00 -1.00 -1.00]]
```

In [12]:

```
# ii)
print('UTU: \n', U.T@U) # i. Printing U^T*U
print('VTV: \n', VT@VT.T) # i. Printing V^T*V

# iii)
print('UUT: \n', U@U.T) # i. Printing U*U^T
print('VVT: \n', VT.T@VT) # i. Printing V*V^T
```

```
UTU:
 [[1.00 -0.00 0.00 -0.00]
 [-0.00 1.00 -0.00 0.00]
 [0.00 -0.00 1.00 -0.00]
 [-0.00 0.00 -0.00 1.00]]
VTV:
 [[1.00 0.00 -0.00 0.00]
 [0.00 1.00 -0.00 -0.00]
 [-0.00 -0.00 1.00 -0.00]
 [0.00 -0.00 -0.00 1.00]]
UUT:
 [[1.00 -0.00 -0.00 -0.00]
 [-0.00 1.00 -0.00 -0.00]
 [-0.00 -0.00 1.00 -0.00]
 [-0.00 -0.00 -0.00 1.00]]
VVT:
 [[1.00 0.00 -0.00 -0.00 -0.00 -0.00]
 [0.00 1.00 0.00 -0.00 -0.00 -0.00]
 [-0.00 0.00 1.00 0.00 -0.00 -0.00]
 [-0.00 -0.00 0.00 0.33 0.33 0.33]
 [-0.00 -0.00 -0.00 0.33 0.33 0.33]
 [-0.00 -0.00 -0.00 0.33 0.33 0.33]]
```

e) Compare the singular vectors and singular values of the economy and full SVD. How do they differ?

In [13]:

```
#2e
print('First left singular vector: \n', U[:,[0]])
print('Largest singular value:', s[0])

print(np.sum(np.abs(s)>1e-6))
```

```
First left singular vector: 
 [[-0.50]
 [-0.50]
 [-0.50]
 [-0.50]]
Largest singular value: 9.797958971132713
2
```

f) Identify an orthonormal basis for the space spanned by the columns of A.

In [14]:

```
U, S, VT = np.linalg.svd(A, full_matrices=False)
print("The orthonormal basis of space spanned by the columns of A is the first 2 columns of
print(U[:, 0:2])
```

```
The orthonormal basis of space spanned by the columns of A is the first 2 co
lumns of U
[[-0.50 -0.50]
 [-0.50 0.50]
 [-0.50 0.50]
 [-0.50 -0.50]]
```

g) Identify an orthonormal basis for the space spanned by the rows of A.

In [15]:

```
U, S, VT = np.linalg.svd(A, full_matrices=False)
print("The orthonormal basis of space spanned by the rows of A is the first 2 columns of V"
V = VT.T
print(V[:, 0:2])
```

```
The orthonormal basis of space spanned by the rows of A is the first 2 colum
ns of V
[[-0.41 -0.41]
 [-0.41 -0.41]
 [-0.41 -0.41]
 [0.41 -0.41]
 [0.41 -0.41]
 [0.41 -0.41]]
```

h) Define the rank-r approximation to A as $A_r = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ where σ_i is the ith singular value with left singular vector u_i and right singular vector v_i.

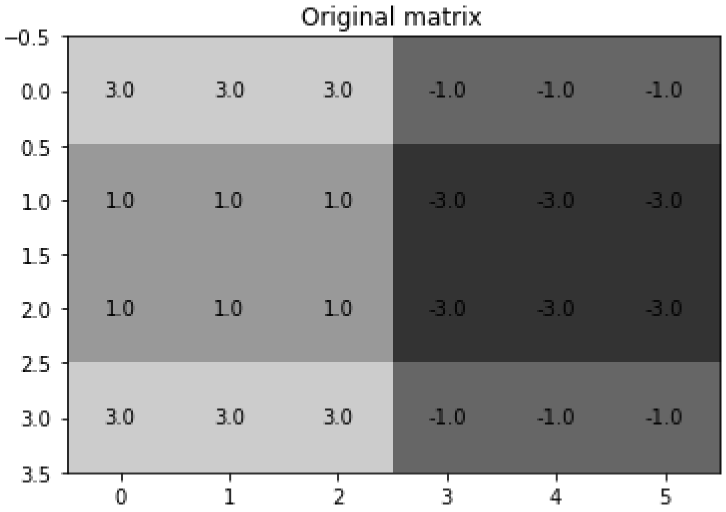
i. Find the rank-1 approximation A_1. How does A_1 compare to A?

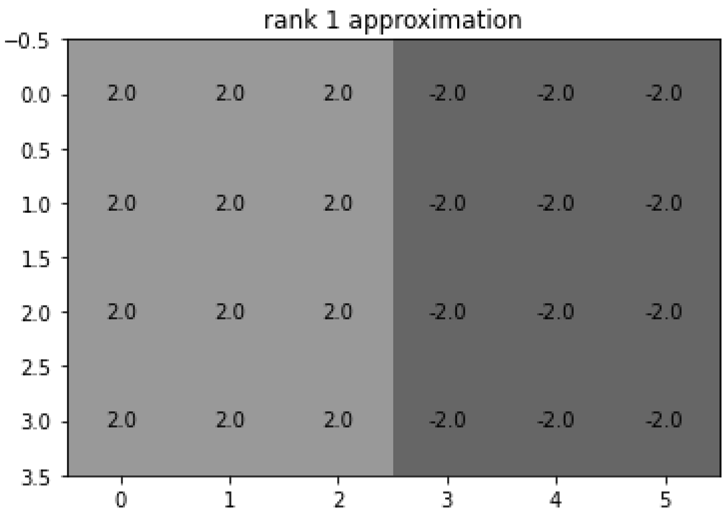
ii. Find the rank-2 approximation A_2. How does A_2 compare to A?

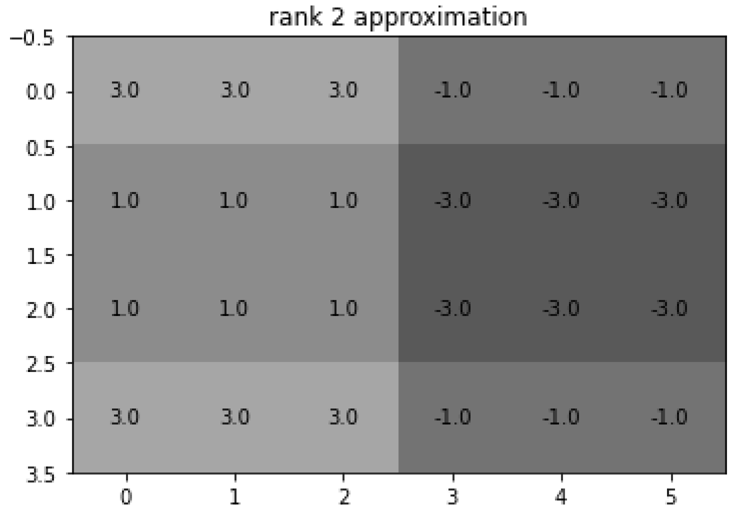
In [16]:

```
import matplotlib.pyplot as plt
## display the original matrix using a heatmap
plt.figure(num=None)
for (j,i),label in np.ndenumerate(A):
    plt.text(i,j,np.round(label,1),ha='center',va='center')
plt.imshow(A, vmin=-5, vmax=5, interpolation='none', cmap='gray')
plt.title('Original matrix' )

## display the rank-r approximations using a heatmap
for r in range(1,3):
    ## Fill in the blank: choose the first r colummns of U, first r singular values, etc...
    A_rank_r_approx = U[:,:r]@S_matrix[:r,:r]@VT[:r,]
    plt.figure(num=None)
    for (j,i),label in np.ndenumerate(A_rank_r_approx):
        plt.text(i,j,np.round(label,1),ha='center',va='center')
    plt.imshow(A_rank_r_approx, vmin=-10, vmax=10, interpolation='none', cmap='gray')
    plt.title('rank ' + str(r) + ' approximation'  )
```







i) The economy SVD is based on the dimension of the matrices and does not consider the rank of the matrix. What is the smallest economy SVD (minimum dimension of the square matrix S) possible for the matrix A? Find U, S, and V for this minimal economy SVD.

In [17]:

```
U, S, VT = np.linalg.svd(A, full_matrices=False)
V = VT.T

U = U[:, 0:2]
Sigma = np.zeros((2,2))
np.fill_diagonal(Sigma,S[0:2])
VT = VT[0:2,:]

print("U:\n", U)
print("Sigma:\n", Sigma)
print("VT:\n", VT)
```

```
U:
 [[-0.50 -0.50]
 [-0.50 0.50]
 [-0.50 0.50]
 [-0.50 -0.50]]
Sigma:
 [[9.80 0.00]
 [0.00 4.90]]
VT:
 [[-0.41 -0.41 -0.41 0.41 0.41 0.41]
 [-0.41 -0.41 -0.41 -0.41 -0.41 -0.41]]
```

In [ ]: