CS/ECE/ME532 Period 10 Activity

Estimated Time:

P1: 25 mins

P2: 25 mins

Preambles

In [2]:
```
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 $\boldsymbol{A} = \begin{bmatrix} 3 & 3 & 3 & -1 & -1 & -1 \\ 1 & 1 & 1 & -3 & -3 & -3 \\ 1 & 1 & 1 & -3 & -3 & -3 \\ 3 & 3 & 3 & -1 & -1 & -1 \end{bmatrix}$. Use the provided script to help you complete the problem.

In [18]:
```
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]], floa
rows, cols = A.shape
print('A = \n', A)
np.linalg.matrix_rank(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]]
```

Out[18]:
```
2
```

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

In [6]:
```
def dist(x, y):
    """
    this function takes in two 1-d numpy as input an outputs
    Euclidean the distance between them
    """
    return np.sqrt(np.sum((x - y)**2)) ## Fill in the blank: Recall the 'distance' funct

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
```

```python
        C = np.array([np.argmin([dist(x_i, y_k) for y_k in centroids]) for x_i in X_tran
        # 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 $\boldsymbol{A}$ with a single cluster.

In [8]:

```python
# k-means with 1 cluster
centroids, C = kMeans(A, 1) ## Fill in the blank: call the "kMeans" algorithm with prope
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 [9]:

```python
# Construct rank-1 approximation using cluster
A_hat_1 = np.repeat(centroids, 6, axis=1)
 # Fill in the blank
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]
 [-1.00 -1.00 -1.00 -1.00 -1.00 -1.00]
 [-1.00 -1.00 -1.00 -1.00 -1.00 -1.00]
 [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 [14]:

```python
# k-means with 2 cluster
centroids2, C2 = kMeans(A, 2) ## Fill in the blank: call the "kMeans" method with proper
print('A = \n', A)
print('centroids = \n', centroids2)
print('centroid assignment = \n', C2)
```

```
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]
```

In [15]:

```python
A_hat_2 = centroids2[:, C2]
 # Fill in the blank
print('Rank-2 Approximation \n', A_hat_2)

# Comment: the rank-2 approximation of A is just A because
# A was rank 2 to begin with.
```

```
Rank-2 Approximation
 [[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 [19]:

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

frobenius_diff_A_A_hat_1 = np.linalg.norm(A - A_hat_1, 'fro')
frobenius_diff_A_A_hat_2 = np.linalg.norm(A - A_hat_2, 'fro')

print(f"Frobenius norm of difference between A and A_hat_1: {frobenius_diff_A_A_hat_1:.2
print(f"Frobenius norm of difference between A and A_hat_2: {frobenius_diff_A_A_hat_2:.2
```

```
Frobenius norm of difference between A and A_hat_1: 9.80
Frobenius norm of difference between A and A_hat_2: 0.00
```

Q2. SVD

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

a) If the full SVD is computed, find the dimensions of $\boldsymbol{U}$, $\boldsymbol{S}$, and $\boldsymbol{V}$.

b) Find the dimensions of $\boldsymbol{U}$, $\boldsymbol{S}$, and $\boldsymbol{V}$ in the economy or skinny SVD of $\boldsymbol{A}$.

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

i. Compute the SVD of $\boldsymbol{A}$. Make sure $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ holds.

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

iii. Find $\boldsymbol{U}\boldsymbol{U}^T$ and $\boldsymbol{V}\boldsymbol{V}^T$. Are the rows of $\boldsymbol{U}$ and $\boldsymbol{V}$ orthonormal? Why?

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

v. What is the rank of $\boldsymbol{A}$?

In [ ]:

```python
# Problem 2a #
```

```
# A is m x n = 4 x 6

# U will be an m x m orthogonal matrix
# U will be 4 x 4 orthogonal matrix

# S will be an m x n diagonal matrix
# S will be a 4 x 6 diagonal matrix

# V^T will be an n x n orthogonal matrix
# V^T will be 6 x 6
```

In [ ]:

```
# Problem 2b #

# For skinny SVD case:

# A is still m x n = 4 x 6

# U will be an m x k matrix, where k=rank(A)
# k=rank(A)=2
# U will be an m x 2 = 4 x 2 matrix

# S will be a k x k diagonal matrix ()
# S will be a 2 x 2 diagonal matrix

# V^T will be a k x n matrix
# V^T will be a 2 x 6 matrix, and V will be a 6 x 2.
```

In [49]:

```
# Problem 2c #

# i)
U_full, s_full, VT_full = np.linalg.svd(A, full_matrices=True)
S_matrix_full = np.zeros_like(A) ## Fill in the blank: Size of S should be equal to size
np.fill_diagonal(S_matrix_full, s_full) ## Fill in the diagonal entries of S_matrix with
print(U_full@S_matrix_full@VT_full)
print(A)

A - U_full@S_matrix_full@VT_full
# So A = U*S*V^T
```

```
[[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]]
```

Out[49]:

```
array([[-0.00, 0.00, 0.00, 0.00, 0.00, 0.00],
       [0.00, 0.00, -0.00, 0.00, 0.00, 0.00],
       [0.00, 0.00, -0.00, 0.00, 0.00, 0.00],
       [0.00, 0.00, 0.00, 0.00, 0.00, 0.00]])
```

In [50]:

```
# ii)
print('UTU: \n', U_full.T@U_full) # i. Printing U^T*U
print('VTV: \n', VT_full@VT_full.T) # i. Printing V^T*V
# Comment: cols of U are orthonormal because U^T U = I
# Comment: cols of V are orthonormal because V^T V = I

# iii)
print('UUT: \n', U_full@U_full.T) # i. Printing U*U^T
print('VVT: \n', VT_full.T@VT_full) # i. Printing V*V^T
# Comment: rows of U are orthonormal because U U^T = I
# Comment: rows of V are orthonormal because V V^T = I
```

```
# 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: 9.79795897113271
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, $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ where $\boldsymbol{U}$ and $\boldsymbol{V}$ are matrices with orthonormal columns comprising the left and right singular vectors and $\boldsymbol{S}$ is a square diagonal matrix of singular values.

i. Compute the SVD of $\boldsymbol{A}$. Make sure $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ holds.

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

iii. Find $\boldsymbol{U}\boldsymbol{U}^T$ and $\boldsymbol{V}\boldsymbol{V}^T$. Are the rows of $\boldsymbol{U}$ and $\boldsymbol{V}$ orthonormal? Why?

In [51]:
```
# i)
U_small, s_small, VT_small = np.linalg.svd(A, full_matrices=False)
S_matrix_small = np.diag(s_small)
print(U_small@S_matrix@VT_small)
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 [52]:
```
# ii)
print('UTU: \n', U_small.T@U_small) # i. Printing U^T*U
print('VTV: \n', VT_small@VT_small.T) # i. Printing V^T*V
# Comment: cols of U are orthonormal because U^T U = I
# Comment: cols of V are orthonormal because V^T V = I

# iii)
print('UUT: \n', U_small@U_small.T) # i. Printing U*U^T
print('VVT: \n', VT_small.T@VT_small) # i. Printing V*V^T
# Comment: rows of U are orthonormal because U U^T = I
# Comment: rows of V are NOT orthonormal because V V^T =/= I
```

```
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 [53]:
```
print(s_small)
print(S_matrix_small)
print(s_full)
print(S_matrix_full)

# The full matrix has larger dimensions but is filled with more 0s
```

```
[9.80 4.90 0.00 0.00]
[[9.80 0.00 0.00 0.00]
 [0.00 4.90 0.00 0.00]
 [0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00]]
[9.80 4.90 0.00 0.00]
[[9.80 0.00 0.00 0.00 0.00 0.00]
 [0.00 4.90 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.00]]
```

f) Identify an orthonormal basis for the space spanned by the columns of $\boldsymbol{A}$.

In [58]:
```
# The first k (rank A) columns of the matrix U from the SVD represent
# an orthonomral basis for the column space of A.
orthonormal_basis_cols = U_full[:,:np.linalg.matrix_rank(A)]
orthonormal_basis_cols
```

Out[58]:
```
array([[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 $\boldsymbol{A}$.

In [60]:
```python
# The first k (rank A) rows of the matrix V^T from the SVD represent
# an orthonormal basis for the row space of A.
orthonormal_basis_rows = VT_full[:np.linalg.matrix_rank(A), :]
orthonormal_basis_rows
```

Out[60]:
```
array([[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 $\boldsymbol{A}$ as $\boldsymbol{A}_r = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$ where σ_i is the ith singular value with left singular vector $\boldsymbol{u}_i$ and right singular vector $\boldsymbol{v}_i$.

i. Find the rank-1 approximation $\boldsymbol{A}_1$. How does $\boldsymbol{A}_1$ compare to $\boldsymbol{A}$?

ii. Find the rank-2 approximation $\boldsymbol{A}_2$. How does $\boldsymbol{A}_2$ compare to $\boldsymbol{A}$?

In [69]:
```python
U, S, Vt = np.linalg.svd(A, full_matrices=False)

# Rank-1 approximation
A1 = S[0] * np.outer(U[:, 0], Vt[0, :])
print(f"A1: {A1}")
print(f"Frob norm diff: {np.linalg.norm(A - A1, 'fro')}")


# Rank-2 approximation
A2 = A1 + S[1] * np.outer(U[:, 1], Vt[1, :])
print(f"A2: {A2}")
print(f"Frob norm diff: {np.linalg.norm(A - A2, 'fro')}")

# A2 is just A because again, rank(A) = 2
```

```
A1: [[2.00 2.00 2.00 -2.00 -2.00 -2.00]
 [2.00 2.00 2.00 -2.00 -2.00 -2.00]
 [2.00 2.00 2.00 -2.00 -2.00 -2.00]
 [2.00 2.00 2.00 -2.00 -2.00 -2.00]]
Frob norm diff: 4.898979485566356
A2: [[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]]
Frob norm diff: 8.021313332746002e-15
```

In [70]:
```python
# (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.

# Comment: the smallest economy SVD possible for the matrix A (4x6):
# U: 4x4
# S: 4x4
# V^T: 6x6

# Compute the economy SVD of A
U, S, Vt = np.linalg.svd(A, full_matrices=False)

print("Economy SVD U:")
print(U)

print("\nEconomy SVD S:")
```

```
print(np.diag(S))

print("\nEconomy SVD V^T:")
print(Vt)
```

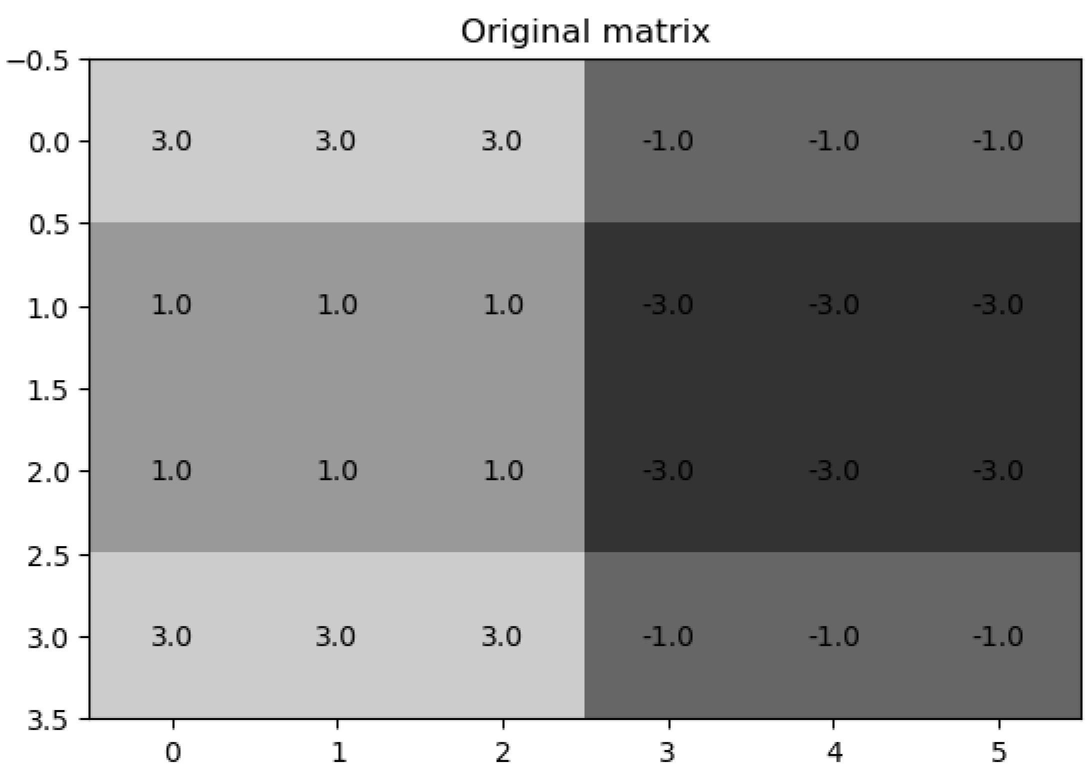
```
Economy SVD U:
[[0.50 0.50 -0.71 -0.01]
 [0.50 -0.50 0.01 -0.71]
 [0.50 -0.50 -0.01 0.71]
 [0.50 0.50 0.71 0.01]]

Economy SVD S:
[[9.80 0.00 0.00 0.00]
 [0.00 4.90 0.00 0.00]
 [0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00]]

Economy SVD V^T:
[[0.41 0.41 0.41 -0.41 -0.41 -0.41]
 [0.41 0.41 0.41 0.41 0.41 0.41]
 [-0.82 0.38 0.44 0.00 0.00 0.00]
 [-0.03 0.72 -0.69 -0.00 -0.00 -0.00]]
```

In [62]:

```
## 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' )
```

Out[62]:

```
Text(0.5, 1.0, 'Original matrix')
```



In [ ]:

```
## 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[:,2]@S_matrix[???,???]@VT[???,:]
    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 $\boldsymbol{S}$) possible for the matrix $\boldsymbol{A}$? Find $\boldsymbol{U}$, $\boldsymbol{S}$, and $\boldsymbol{V}$ for this minimal economy SVD.