CS/ECE/ME532 Period 10 Activity

Estimated Time:

P1: 25 mins

P2: 25 mins

Preambles

In [81]:
```
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 [82]:
```
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,-
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 [83]:
```
# 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 [84]:
```python
def dist(x, y):
    """
    this function takes in two 1-d numpy as input an outputs
    Euclidean the distance between them
    """
    return np.linalg.norm(x - y) ## Fill in the blank: Recall the 'distance'

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 num
    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
        # 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 cen
                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 [85]:
```python
# k-means with 1 cluster
centroids, C = kMeans(A, 1) ## Fill in the blank: call the "kMeans" algorith
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 [86]:
```
# Construct rank-1 approximation using cluster

A_hat_1 = np.tile(centroids, (1, 6))
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.

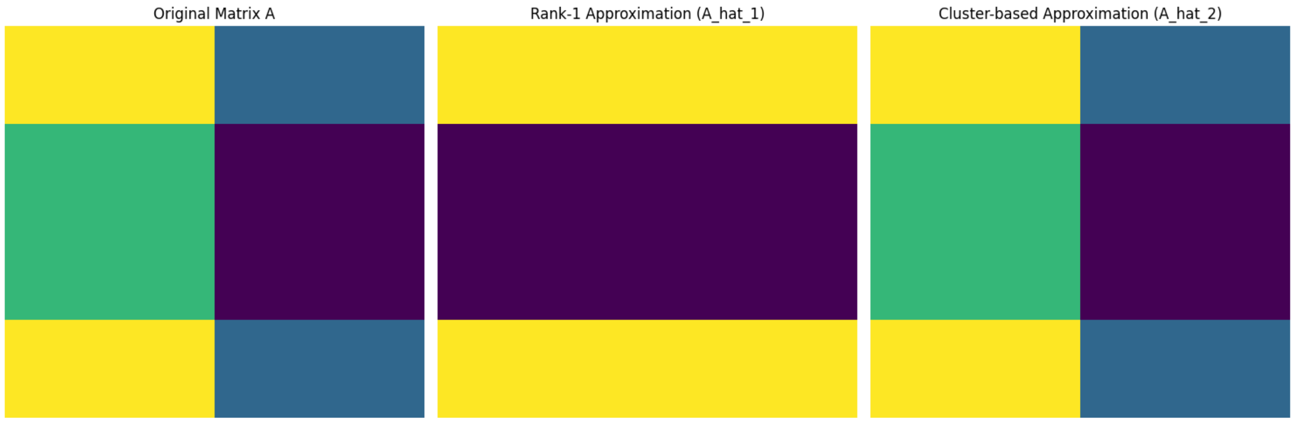
In [87]:
```
# k-means with 2 cluster
centroids, C = kMeans(A, 2) ## Fill in the blank: call the "kMeans" method w
print('A = \n', A)
print('centroids = \n', centroids)
print('centroid assignment = \n', C)

A_hat_2 = np.array([centroids[:, C[i]] for i in range(len(A[0]))]).T
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 =
 [[-1.00 3.00]
 [-3.00 1.00]
 [-3.00 1.00]
 [-1.00 3.00]]
centroid assignment =
 [1 1 1 0 0 0]
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 [88]:

```python
# Write code to compare A_hat_1 and A_hat_2 to the original matrix A
error_A_hat_1 = np.linalg.norm(A - A_hat_1, 'fro')
error_A_hat_2 = np.linalg.norm(A - A_hat_2, 'fro')

# Print the errors
print(f"Frobenius Norm (Error) for A_hat_1: {error_A_hat_1:.4f}")
print(f"Frobenius Norm (Error) for A_hat_2: {error_A_hat_2:.4f}")

# Visualize the original matrix and approximations
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Displaying A
axes[0].imshow(A, aspect='auto', cmap='viridis')
axes[0].set_title('Original Matrix A')
axes[0].axis('off')

# Displaying A_hat_1
axes[1].imshow(A_hat_1, aspect='auto', cmap='viridis')
axes[1].set_title('Rank-1 Approximation (A_hat_1)')
axes[1].axis('off')

# Displaying A_hat_2
axes[2].imshow(A_hat_2, aspect='auto', cmap='viridis')
axes[2].set_title('Cluster-based Approximation (A_hat_2)')
axes[2].axis('off')

plt.tight_layout()
plt.show()
```

```
Frobenius Norm (Error) for A_hat_1: 9.7980
Frobenius Norm (Error) for A_hat_2: 0.0000
```



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 $\boldsymbol{V}$.

A.shape = 4x6

U.shape = 4x4

S.shape = 4x6

V^T.shape = 6x6

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

A.shape = 4x6

U.shape = 4x4

S.shape = 4x4

V^T.shape = 4x6

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 [89]:
```
# 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
np.fill_diagonal(S_matrix, s) ## Fill in the diagonal entries of S_matrix wi
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 [90]:
```
# 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

# 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 [91]:
```python
# i)
U, s, VT = np.linalg.svd(A, full_matrices=False)
S_skinny = np.diag(s)
print(U@S_skinny@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 [92]:

```
# 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?

They have the same values but are just different sizes

In [93]:

```
print("S_normal\n", S_matrix)
print("S_skinny\n", S_skinny)
```

```
S_normal
 [[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]]
S_skinny
 [[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]]
```

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

In [94]:
```
# Assuming U from the SVD of A
rank_A = np.linalg.matrix_rank(A)
column_space_basis = U[:, :rank_A]
print('Orthonormal basis for the column space of A: \n', column_space_basis)
```

```
Orthonormal basis for the column space of A:
 [[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 [95]:
```
# Assuming VT from the SVD of A
row_space_basis = VT[:rank_A, :]
print('Orthonormal basis for the row space of A: \n', row_space_basis)
```

```
Orthonormal basis for the row space of A:
 [[0.41 0.41 0.41 -0.41 -0.41 -0.41]
 [0.41 0.41 0.41 0.41 0.41 0.41]]
```

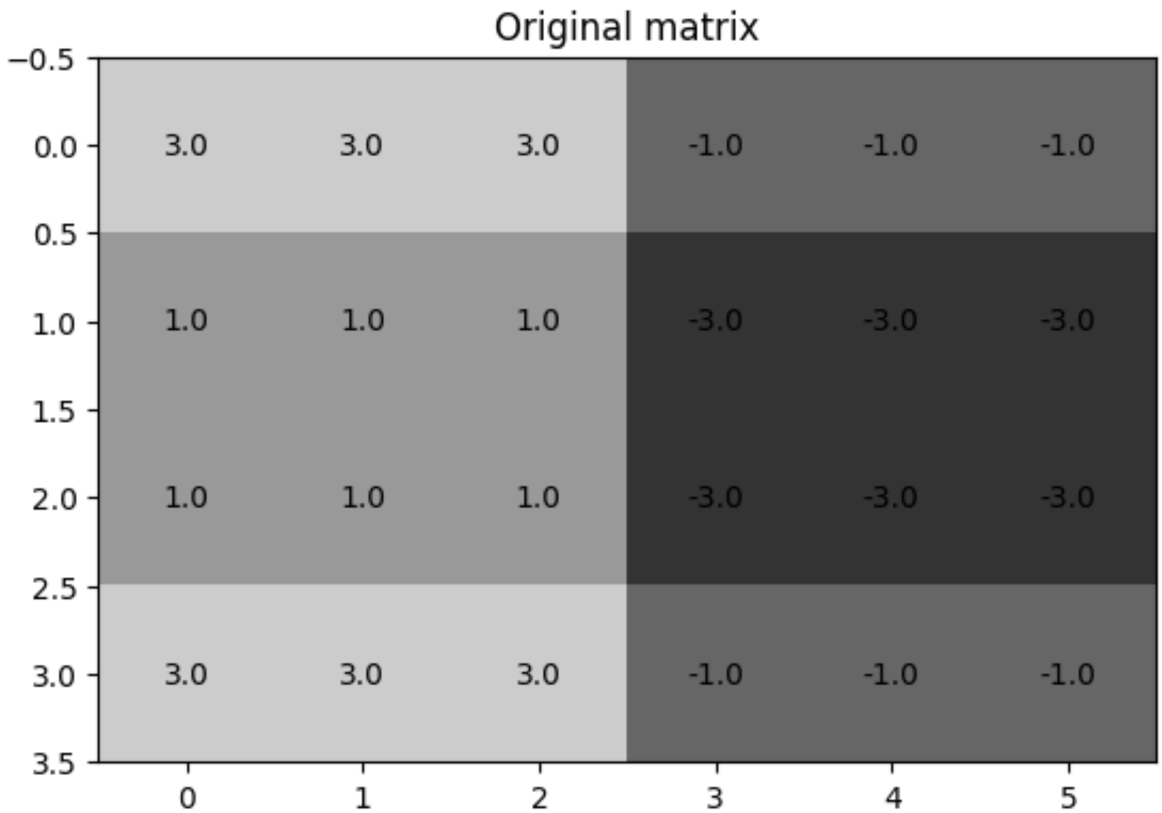
h) Define the rank-r approximation to $\boldsymbol{A}$ as $\boldsymbol{A}_r = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$ where σ_i is the ith singular value with left singular vector $\boldsymbol{u}_i$ and right singular vector $\boldsymbol{v}_i$.

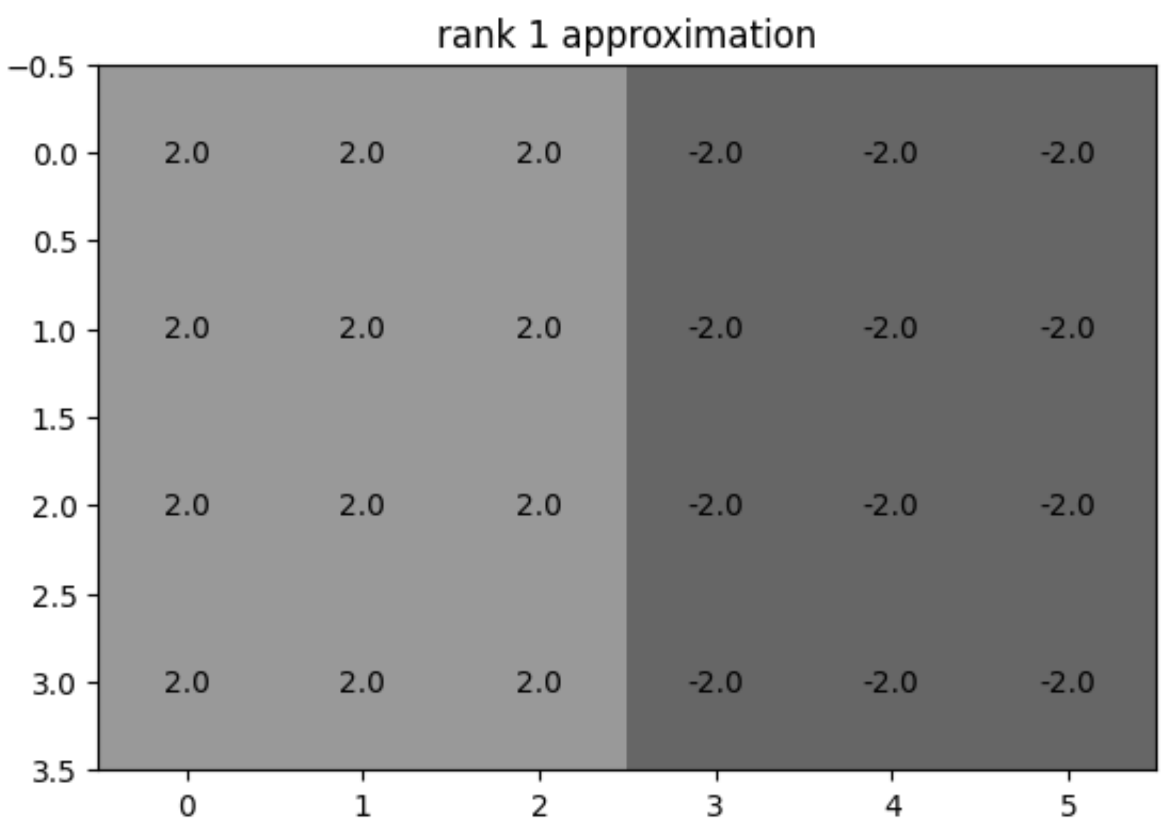
i. Find the rank-1 approximation $\boldsymbol{A}_1$. How does $\boldsymbol{A}_1$ compare to $\boldsymbol{A}$?

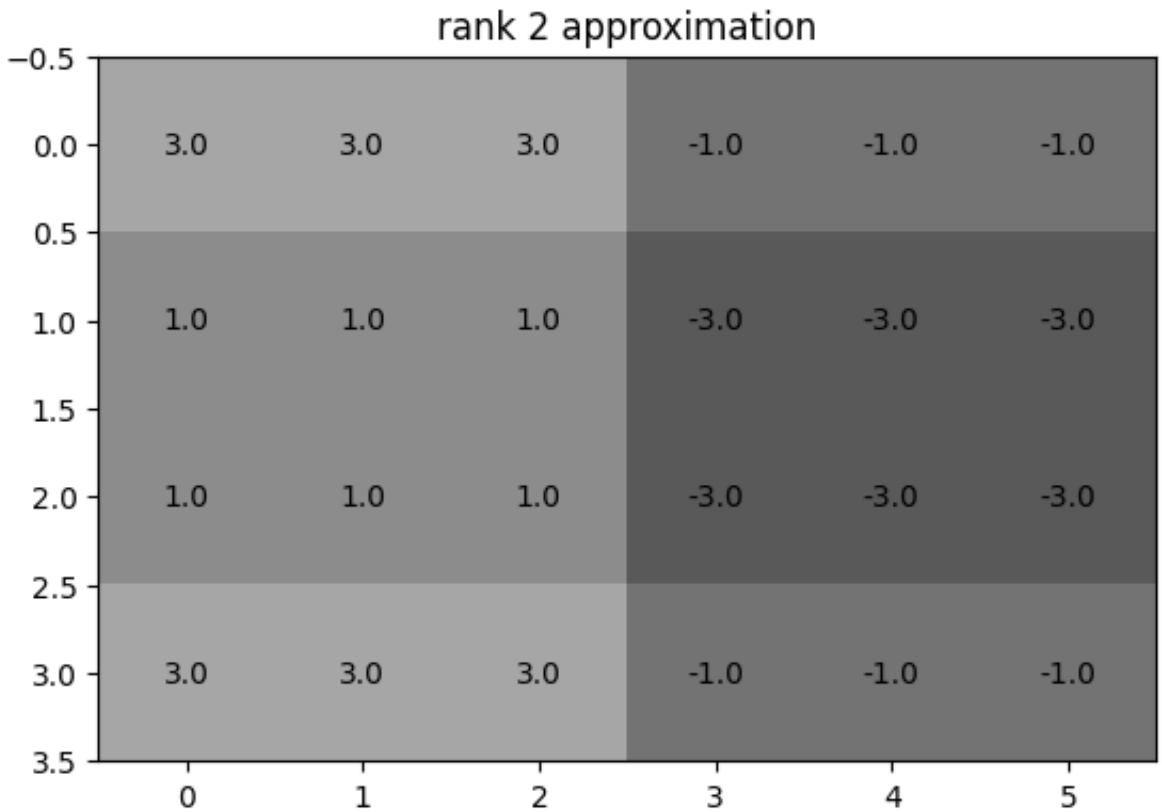
ii. Find the rank-2 approximation $\boldsymbol{A}_2$. How does $\boldsymbol{A}_2$ compare to $\boldsymbol{A}$?

In [96]:
```
## 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
    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', cma
    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.

The smallest dim S can be is either 4x4 or 2x2.

In [97]:
```
U_economy, s_economy, VT_economy = np.linalg.svd(A, full_matrices=False)

# Economy SVD dimensions
print('U (economy) shape:', U_economy.shape)
print('U (economy) shape:', U_economy)
print('S (economy) shape:', len(s_economy))
print("S (economy) \n", np.diag(s_economy))
print('VT (economy) shape:', VT_economy.shape)
print('VT (economy) shape:', VT_economy)

# Minimal SVD using only the non-zero singular values
r = 2  # Rank of the matrix A

# Create the diagonal matrix S (2x2) with the first r singular values
S_minimal = np.diag(s_economy[:r])

# Create the U and V matrices for the minimal economy SVD
U_minimal = U_economy[:, :r]
VT_minimal = VT_economy[:r, :]

# Print the dimensions and matrices
```

```python
print('U (minimal economy) shape:', U_minimal.shape)
print('S (minimal economy) shape:', S_minimal.shape)
print('VT (minimal economy) shape:', VT_minimal.shape)
```

```
U (economy) shape: (4, 4)
U (economy) shape: [[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]]
S (economy) shape: 4
S (economy)
 [[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]]
VT (economy) shape: (4, 6)
VT (economy) shape: [[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]]
U (minimal economy) shape: (4, 2)
S (minimal economy) shape: (2, 2)
VT (minimal economy) shape: (2, 6)
```

In [ ]: