Δ Item 2

$$w = (A^T A)^{-1} A^T d$$

$$\rightarrow V \Sigma^{-1} U^T y$$

b) $y = \begin{bmatrix}1\\0\\0\\1\end{bmatrix}$

$$w = \underbrace{\frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix}}_{2\times2} \underbrace{\frac{1}{r}\begin{bmatrix}r&0\\0&1\end{bmatrix}}_{2\times2} \underbrace{\frac{1}{2}\begin{bmatrix}1&1&1&1\\1&-1&-1&1\end{bmatrix}}_{2\times4} \underbrace{\begin{bmatrix}1\\0\\0\\1\end{bmatrix}}_{4\times1}$$

$$= \frac{1}{\cancel{2}\sqrt{2}r}\begin{bmatrix}r&1\\r&-1\end{bmatrix}\begin{bmatrix}\cancel{2}1\\\cancel{2}1\end{bmatrix} = \frac{1}{\sqrt{2}r}\begin{bmatrix}r+1\\r-1\end{bmatrix}$$

y is on the span(U), so the error is 0

$$\|w\|_2^2 = \frac{1}{2r^2}\begin{bmatrix}r^2+2r+1\\r^2-2r+1\end{bmatrix} = \frac{\cancel{2}r^2+\cancel{2}}{\cancel{2}r^2} = \frac{r^2+1}{r^2}$$

$r \to 0$, $w \to \infty$

c)

$$w = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}\frac{1}{2}\begin{bmatrix}1&1&1&1\end{bmatrix}\begin{bmatrix}1\\0\\0\\1\end{bmatrix}$$

$$= \frac{1}{\cancel{2}\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$$

$$\rightarrow \|w\|_2^2 = \left(\frac{1}{\sqrt{2}}\right)^2 + \left(\frac{1}{\sqrt{2}}\right)^2 = 1$$

$$Xw = UU^T y$$

$$\frac{1}{\cancel{4}_{2}}\underbrace{\begin{bmatrix}1\\1\\1\\1\end{bmatrix}}_{\cancel{1\times4}\ 4\times1}\underbrace{\begin{bmatrix}1&1&1&1\end{bmatrix}}_{\cancel{4\times1}\ 1\times4}\underbrace{\begin{bmatrix}1\\0\\0\\1\end{bmatrix}}_{4\times1} - \begin{bmatrix}1\\0\\0\\1\end{bmatrix}$$

~~$= \begin{bmatrix}1\\2\\2\\1\end{bmatrix} - [illegible] = 0 = 1+4+4+1$~~

$$= \begin{bmatrix}\frac{1}{2}-1\\\frac{1}{2}\\\frac{1}{2}\\\frac{1}{2}-1\end{bmatrix} = \frac{1}{4}\times4 = 1$$

activity-11

March 5, 2024

0.1 CS/ECE/ME 532 - Activity 11 Item 1

Preamble

[1]:
```
import numpy as np # numpy
from pprint import pprint as pprint # pretty print
from scipy.io import loadmat # load & save data
from scipy.io import savemat
import matplotlib.pyplot as plt # plot
from mpl_toolkits import mplot3d
np.set_printoptions(formatter={'float': lambda x: "{0:0.2f}".format(x)})
```

K-means has some 'random' components in it. You will get different results depending on your luck. Even when you run an identical code, you will see some different results from your peers. So… we need the following line of code to start with:

[2]:
```
np.random.seed(2)
```

Indeed, one may be tempted to try so many random seeds until you get a good performance!

Don't do that! Some subfields in ML are suffering from a "reproduction crisis" partially due to this. See these for more details: - Deep Reinforcement Learning that Matters - This AI researcher is trying to ward off a reproducibility crisis - Artificial Intelligence Confronts a 'Reproducibility' Crisis

---

0.1.1 1. K-means and SVD for rating prediction

We return to the movies rating problem considered previously. The movies and ratings from your friends on a scale of 1-10 are:

Movie	Jake	Jennifer	Jada	Theo	Ioan	Bo	Juanita
Star Trek	4	7	2	8	7	4	2
Pride and Prejudice	9	3	5	6	10	5	5
The Martian	4	8	3	7	6	4	1
Sense and Sensibility	9	2	6	5	9	5	4
Star Wars: Empire Strikes	4	9	2	8	7	4	1

Run the following code block to create a numpy array X

```
[3]: X = np.array([
         [4,7,2,8,7,4,2],
         [9,3,5,6,10,5,5],
         [4,8,3,7,6,4,1],
         [9,2,6,5,9,5,4],
         [4,9,2,8,7,4,1],
         ], float)
     print(X)
```

```
[[4.00 7.00 2.00 8.00 7.00 4.00 2.00]
 [9.00 3.00 5.00 6.00 10.00 5.00 5.00]
 [4.00 8.00 3.00 7.00 6.00 4.00 1.00]
 [9.00 2.00 6.00 5.00 9.00 5.00 4.00]
 [4.00 9.00 2.00 8.00 7.00 4.00 1.00]]
```

`float` is necessary as the array will only hold integers otherwise

Also, we load the K-mean algorithm we implemented in the last activity.

```
[4]: def dist(x, y):
         return (x-y).T@(x-y)

     def kMeans(X, K, maxIters = 20):
         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_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
```

Note that `(x-y).T@(x-y)` is the squared L^2 norm of $x - y$: since x and y are 1-d numpy arrays, the .T does not actually impact the code.

1 a) Use the K-means algorithm to represent the columns of X with two clusters.

```
[5]: centroids_2, C_2 = kMeans(X, 2) ## Fill in the blank: call the "kMeans"␣
      ↪algorithm with proper input arguments
     print('centroids = \n', centroids_2)
     print('centroid assignment = \n', C_2)
```

```
centroids =
 [[3.00 7.33]
 [6.00 6.33]
 [3.00 7.00]
 [6.00 5.33]
 [2.75 8.00]]
centroid assignment =
 [0 1 0 1 1 0 0]
```

1 b) Express the rank-2 approximation to X based on this cluster as TW^T where the columns of T contains the cluster centers and W is a vector of ones and zeros. Compare the rank-2 clustering approximation to the original matrix.

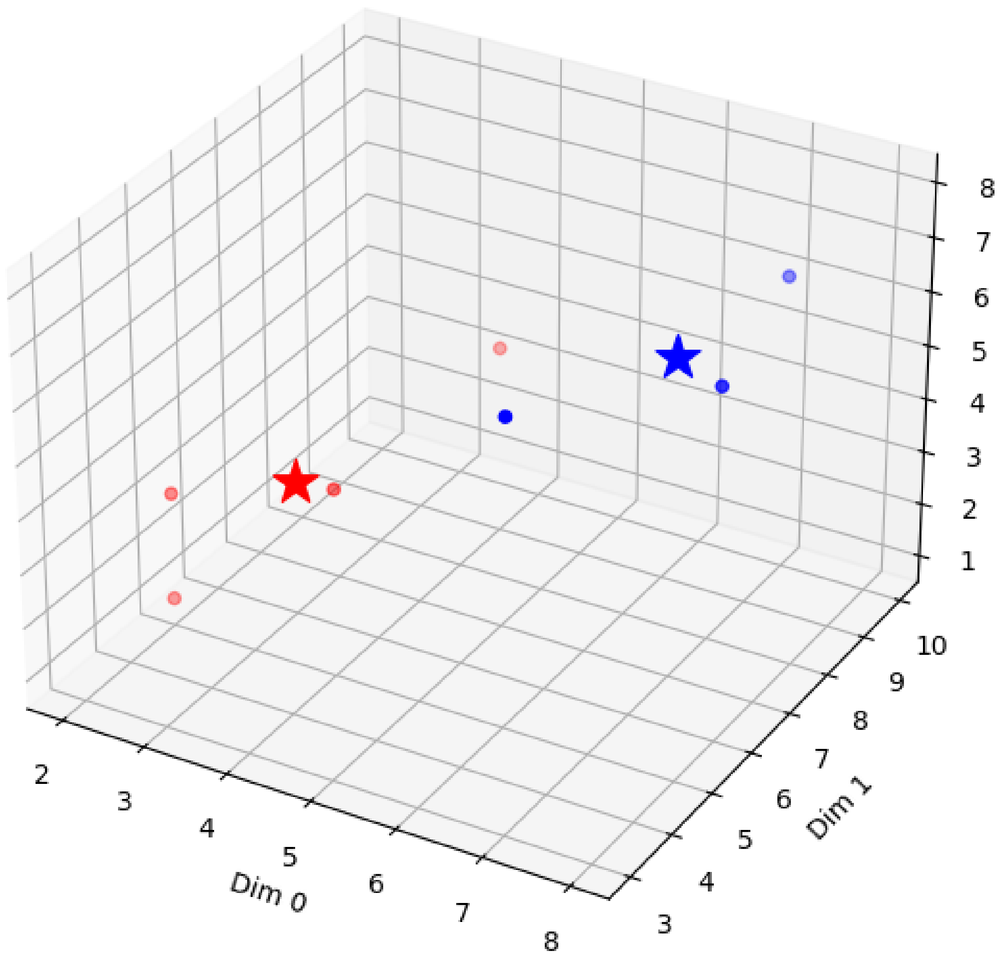
[13]:
```python
# Construct rank-2 approximation using cluster
# X (5x7) = T (5x2) @ WT (2x7)
WT = np.zeros((2,7))
for ind, v in enumerate(C_2):
    # print(ind, v)
    if v == 0:
        WT[0, ind] = 1
    else:
        WT[1, ind] = 1
# print(WT)
X_hat_2 = centroids_2 @ WT ## Fill in the blank
print('Rank-2 Approximation = \n', X_hat_2)
```

```
Rank-2 Approximation =
 [[3.00 7.33 3.00 7.33 7.33 3.00 3.00]
 [6.00 6.33 6.00 6.33 6.33 6.00 6.00]
 [3.00 7.00 3.00 7.00 7.00 3.00 3.00]
 [6.00 5.33 6.00 5.33 5.33 6.00 6.00]
 [2.75 8.00 2.75 8.00 8.00 2.75 2.75]]
```

1 c) Play with the following code! You can pick three dimensions to look at by modifying `coordinates_to_plot`. Just have fun with it.

[6]:
```python
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection = "3d")
coordinates_to_plot = [0,1,2]
color_array = np.array(['red', 'blue'])
ax.scatter3D(
            X[coordinates_to_plot[0],:], # x
            X[coordinates_to_plot[1],:], # y
            X[coordinates_to_plot[2],:], # z
            c=color_array[C_2] # color depends on cluster idx
            )

for i in range(2):
    ax.scatter3D(
            centroids_2[coordinates_to_plot[0],i], # x
            centroids_2[coordinates_to_plot[1],i], # y
            centroids_2[coordinates_to_plot[2],i], # y
            marker='*', # star instead of circle
            s=300, # size
            c=color_array[i] # color
            )

ax.set_xlabel('Dim %d'%coordinates_to_plot[0])
ax.set_ylabel('Dim %d'%coordinates_to_plot[1])
ax.set_zlabel('Dim %d'%coordinates_to_plot[2])
```

```
[6]: Text(0.5, 0, 'Dim 2')
```



[14]:
```
color_array[C_2]
```

[14]:
```
array(['red', 'blue', 'red', 'blue', 'blue', 'red', 'red'], dtype='<U4')
```

1 d) Repeat a)–c) with $K = 3$.

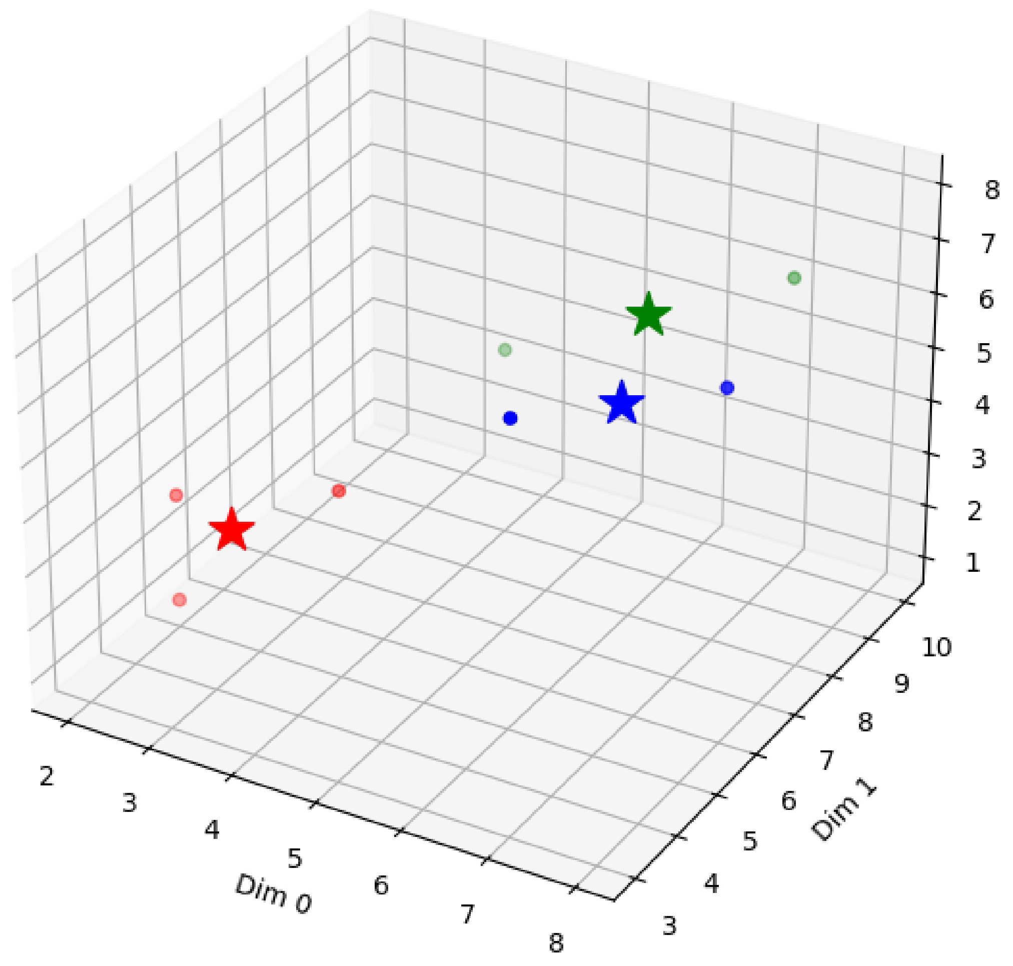
[15]:
```
centroids_3, C_3 =  kMeans(X, 3) ## Fill in the blank
print('centroids = \n', centroids_3)
print('centroid assignment = \n', C_3)
```

```
centroids =
 [[2.67 7.50 5.50]
 [5.00 4.50 9.50]
 [2.67 7.50 5.00]
 [5.00 3.50 9.00]
 [2.33 8.50 5.50]]
centroid assignment =
 [2 1 0 1 2 0 0]
```

[20]:
```
# Construct rank-3 approximation using cluster
# X (5x7) = T (5x3) @ WT (3x7)
WT = np.zeros((3, 7))
for ind, v in enumerate(C_3):
    # print(ind, v)
    if v == 0:
        WT[0, ind] = 1
    elif v == 1:
        WT[1, ind] = 1
    else:
        WT[2, ind] = 1
# print(WT)
X_hat_3 =  centroids_3 @ WT ## Fill in the blank
print('Rank-3 Approximation = \n', X_hat_3)
```

```
Rank-3 Approximation =
 [[5.50 7.50 2.67 7.50 5.50 2.67 2.67]
 [9.50 4.50 5.00 4.50 9.50 5.00 5.00]
 [5.00 7.50 2.67 7.50 5.00 2.67 2.67]
 [9.00 3.50 5.00 3.50 9.00 5.00 5.00]
 [5.50 8.50 2.33 8.50 5.50 2.33 2.33]]
```

[21]:
```
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
coordinates_to_plot = [0,1,2]
color_array = np.array(['red', 'blue', 'green'])
ax.scatter3D(
            X[coordinates_to_plot[0],:], # x
            X[coordinates_to_plot[1],:], # y
            X[coordinates_to_plot[2],:], # z
            c=color_array[C_3] # color depends on cluster idx
            )

for i in range(3):
    ax.scatter3D(
            centroids_3[coordinates_to_plot[0],i], # x
            centroids_3[coordinates_to_plot[1],i], # y
            centroids_3[coordinates_to_plot[2],i], # z
            marker='*', # star instead of circle
            s=300, # size
            c=color_array[i] # color
            )

ax.set_xlabel('Dim %d'%coordinates_to_plot[0])
ax.set_ylabel('Dim %d'%coordinates_to_plot[1])
ax.set_zlabel('Dim %d'%coordinates_to_plot[2]);
```



1 e) SVD can be also used to find T and W such that $X \approx TW$. Assume that you are given the SVD of X, i.e., $X = USV^T$. Find SVD-based T and W as a function of U, S, V (In an equation form, not numbers.) Recall that T is a 5-by-r matrix with orthonormal columns. T = U

W = S_matrix @ VT

1 f) Find T, W and the rank-r approximation to X for $r = 2$. What aspects of the ratings does the first taste vector capture? What about the second taste vector? The first taste vector seems to have an average values, while the second may represent different genres of movies, containing positive and negative values.

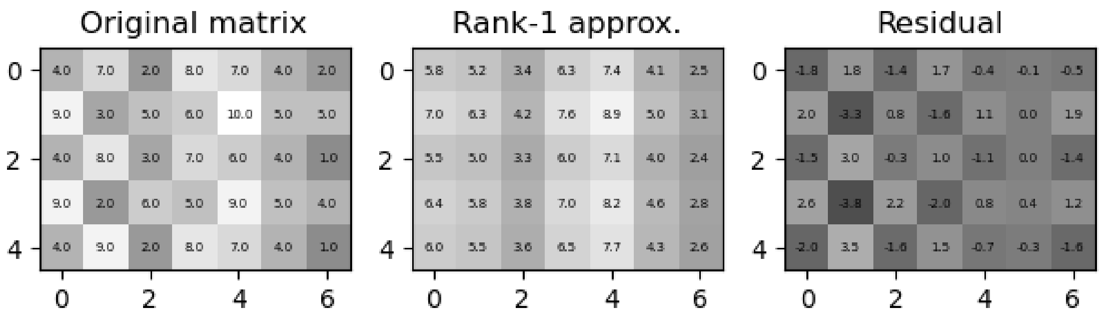
```
[31]: U, s, VT = np.linalg.svd(X, full_matrices=True)
S_matrix = np.zeros_like(X)
np.fill_diagonal(S_matrix, s)
# X (5x7) = U (5x5) @ S_matrix (5x7) @ VT (7*7)
## Fill in the blank using U, S_matrix, and VT
T = U
W = S_matrix @ VT
# print(W)
for r in range(0,2):
    print('Rank %d matrices:'%(r+1))
    T_r = T[:,0:r+1] ## Choose the first r columns of T
    W_r = W[0:r+1,:] ## Choose the first r rows of W
    print(T_r)
    print(W_r)
    print('---')
```

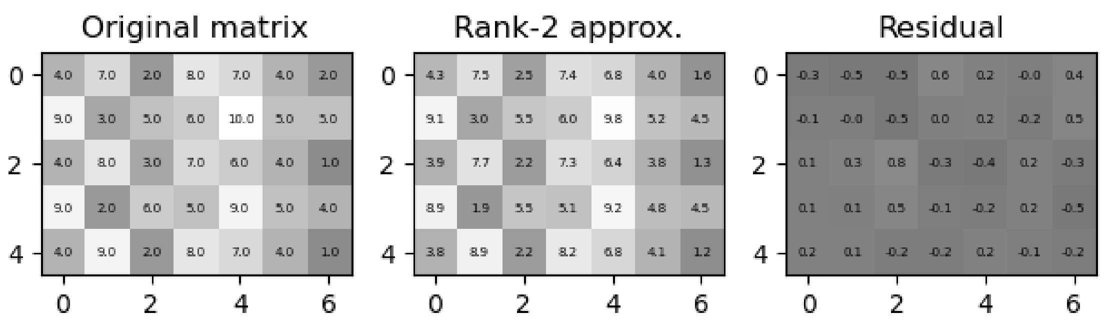
```
Rank 1 matrices:
[[-0.42]
 [-0.51]
 [-0.40]
 [-0.47]
 [-0.44]]
[[-13.77 -12.52 -8.24 -15.02 -17.65 -9.89 -6.07]]
---
Rank 2 matrices:
[[-0.42 -0.32]
 [-0.51 0.47]
 [-0.40 -0.37]
 [-0.47 0.55]
 [-0.44 -0.48]]
[[-13.77 -12.52 -8.24 -15.02 -17.65 -9.89 -6.07]
 [4.49 -7.06 2.93 -3.46 1.80 0.40 3.06]]
---
```

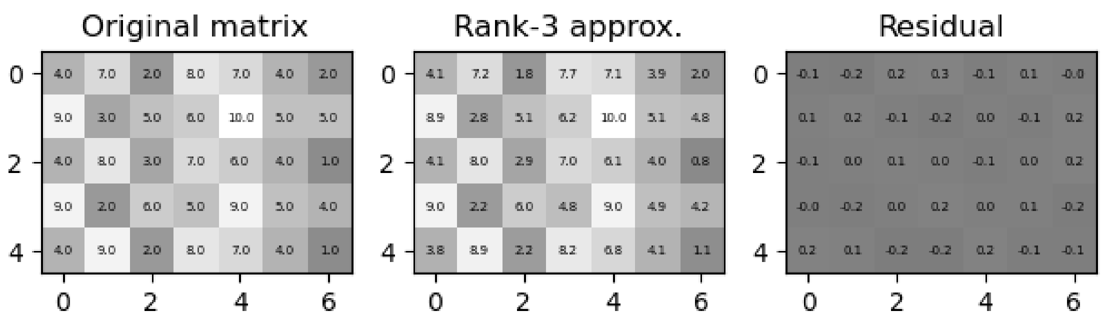
1 g) The following code visualizes the rank-r approximation for an increasing value of r. When does the approximation become exact? Why? The rank-5 approximation become exacet, because we only have 5 weights. Because s has only 5 values, showing the rank of A is five, we use rank-5 approximation to get the exact X.

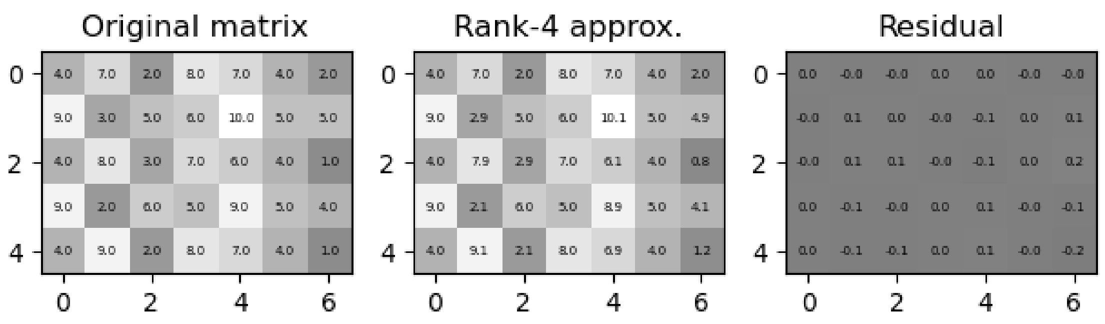
[32]:
```
print(s)
for r in range(0,7):
    T_r = T[:,0:r+1] ## Choose the first r columns of T
    W_r = W[0:r+1,:] ## Choose the first r rows of W
    X_rank_r_approx = T_r@W_r
    fig, ax = plt.subplots(1,3,figsize=(7.5, 7))
    for (j,i),label in np.ndenumerate(X):
        ax[0].text(i,j,np.round(label,1),ha='center',va='center', size=5)
    im = ax[0].imshow(X, vmin=-10, vmax=10, interpolation='none', cmap='gray')
    for (j,i),label in np.ndenumerate(X_rank_r_approx):
        ax[1].text(i,j,np.round(label,1),ha='center',va='center', size=5)
    im = ax[1].imshow(X_rank_r_approx, vmin=-10, vmax=10, interpolation='none',␣
 ↪cmap='gray')
    for (j,i),label in np.ndenumerate(X-X_rank_r_approx):
        ax[2].text(i,j,np.round(label,1),ha='center',va='center', size=5)
    im = ax[2].imshow(X-X_rank_r_approx, vmin=-10, vmax=10,␣
 ↪interpolation='none', cmap='gray')

    ax[0].set_title("Original matrix")
    ax[1].set_title("Rank-%d approx." %(r+1))
    ax[2].set_title("Residual")
```

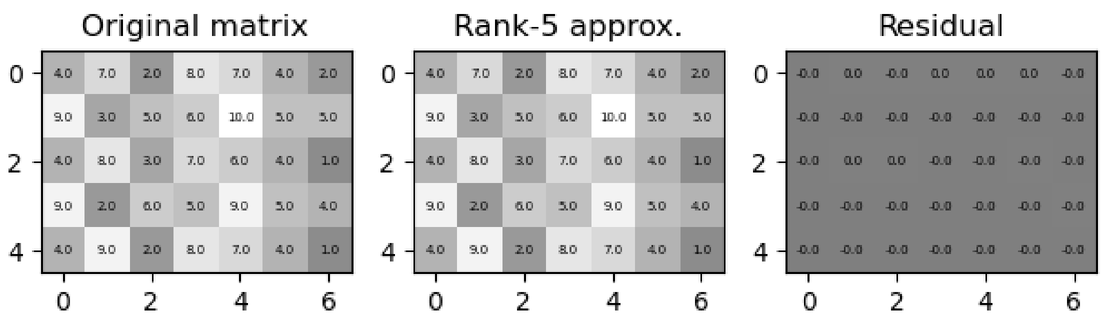
```
[32.95 10.17 1.79 0.70 0.41]
```

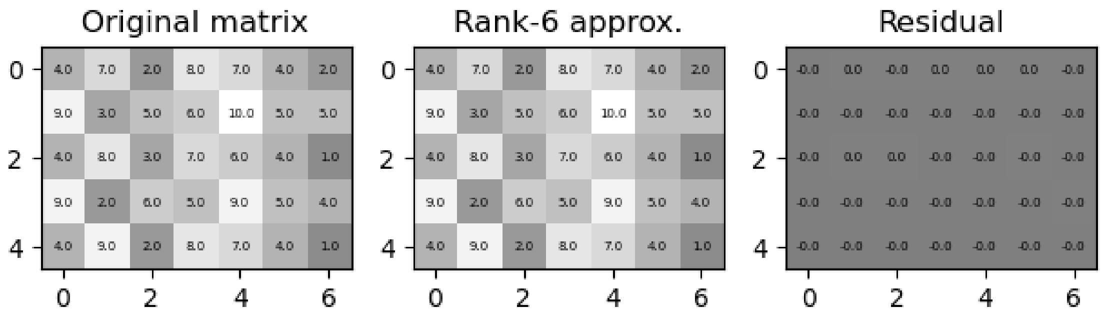


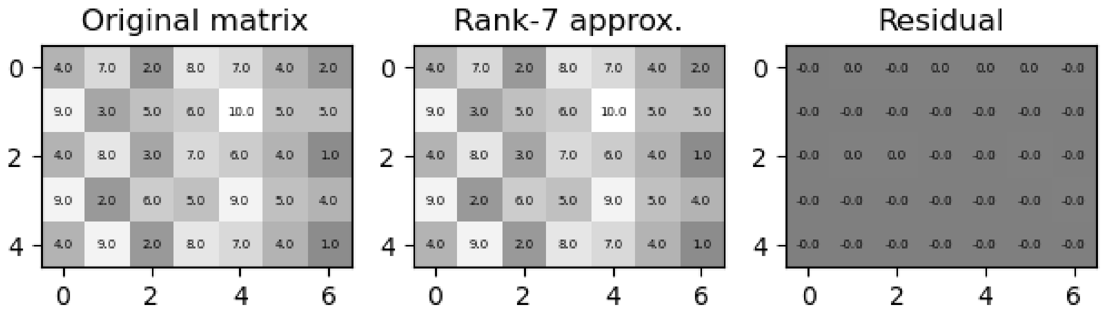

Original matrix
Rank-2 approx.
Residual


Original matrix
Rank-3 approx.
Residual


Original matrix
Rank-4 approx.
Residual


Original matrix
Rank-5 approx.
Residual





1 g) Your friend Jon rates Star Trek 6 and Pride and Prejudice 4. Assume a two-column taste matrix T. Formulate a system of equations that can find Jon's weight vector. Write down the least square solution. We have [6],[4] calculated by T @ w, so the taste profile would be 2x2 and w would be 2x1.

1 h) Using this weight vector, how can we predict Jon's ratings for all five movies, including the remaining three movies? We have taste profile (5x2) and weight (2x1), so we can calculate Jon's ratings (5x1).

1 i) Predict Jon's ratings for all the five movies with different choices of the taste matrix.

- Choice 1: T is the two centroids of the K-means result with $K = 2$
- Choice 2: T is the first two centroids of the K-means result with $K = 3$
- Choice 3: T is the first two SVD-based taste vectors

[33]:
```
y = np.array([[6],[4]])

## Choice 1: K-means (K=2) based taste matrix T
print('K-means with K=2')
```

```
T = centroids_2[:,0:2] # fill in the blank
T_12 = T[0:2,:]
print(T@np.linalg.inv(T_12.T@T_12)@T_12.T@y)

## Choice 2: K-means (K=3) based taste matrix T
print('')
print('K-means with K=3')
T = centroids_3[:,0:2] # fill in the blank
T_12 = T[0:2,:]
print(T@np.linalg.inv(T_12.T@T_12)@T_12.T@y)

## Choice 3: SVD-based taste matrix T
print('')
print('SVD with two vectors')
T = U[:,0:2] # fill in the blank
T_12 = T[0:2,:]
print(T@np.linalg.inv(T_12.T@T_12)@T_12.T@y)
```

```
K-means with K=2
[[6.00]
 [4.00]
 [5.68]
 [3.04]
 [6.73]]

K-means with K=3
[[6.00]
 [4.00]
 [6.00]
 [3.24]
 [6.72]]

SVD with two vectors
[[6.00]
 [4.00]
 [6.01]
 [3.23]
 [6.83]]
```