1

In [1]:
```
import numpy as np
from scipy.io import loadmat
from scipy.io import savemat
import matplotlib.pyplot as plt

def kMeans(X, K, maxIters = 20, plot_progress = None):

    centroids = X[np.random.choice(len(X), K)]
    for i in range(maxIters):
        # Cluster Assignment step
        C = np.array([np.argmin([(x_i-y_k)@(x_i-y_k) for y_k in centro
ids]) for x_i in X])
        # Update centroids step
        centroids = []
        for k in range(K):
            if (C == k).any():
                centroids.append( X[C == k].mean(axis = 0) )
            else: # if there are no data points assigned to this certa
in centroid
                centroids.append( X[np.random.choice(len(X))] )
        if plot_progress != None: plot_progress(X, C, np.array(centroi
ds))
    return np.array(centroids) , C

# Define A for activity
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]])

rows = np.array(A.shape)[0]
cols = np.array(A.shape)[1]
```

In [2]:
```
# k-means with 1 cluster
centroids, C = kMeans(A.transpose(), K = 1)
print('A = ')
print(A)
print('centroid assigned = ',C)
print('centroids')
print(centroids.transpose())
```

```
A =
[[ 3  3  3 -1 -1 -1]
 [ 1  1  1 -3 -3 -3]
 [ 1  1  1 -3 -3 -3]
 [ 3  3  3 -1 -1 -1]]
centroid assigned =  [0 0 0 0 0 0]
centroids
[[ 1.]
 [-1.]
 [-1.]
 [ 1.]]
```

In [3]:
```
# Construct rank-1 approximation using cluster
Ahat_1 = centroids.transpose()@np.ones((1,cols),float)

print('Rank-1 Approximation')
print(Ahat_1)
```

```
Rank-1 Approximation
[[ 1.  1.  1.  1.  1.  1.]
 [-1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1.]
 [ 1.  1.  1.  1.  1.  1.]]
```

W_T = [1 1 1 1 1 1]

In [4]:
```
# k-means with 2 clusters
centroids, C = kMeans(A.transpose(), K = 2)

print('A = ')
print(A)
print('centroid assigned = ',C)
print('centroids')
print(centroids.transpose())
```

```
A =
[[ 3  3  3 -1 -1 -1]
 [ 1  1  1 -3 -3 -3]
 [ 1  1  1 -3 -3 -3]
 [ 3  3  3 -1 -1 -1]]
centroid assigned =  [1 1 1 0 0 0]
centroids
[[-1.  3.]
 [-3.  1.]
 [-3.  1.]
 [-1.  3.]]
```

In [5]:
```
# Construct rank-2 approximation using clusters

Ahat_2 = np.zeros((rows,cols),float)
for i in range(cols):
    Ahat_2[:,i]=centroids.transpose()[:,C[i]]

print('Rank-2 Approximation')
print(Ahat_2)
```

```
Rank-2 Approximation
[[ 3.  3.  3. -1. -1. -1.]
 [ 1.  1.  1. -3. -3. -3.]
 [ 1.  1.  1. -3. -3. -3.]
 [ 3.  3.  3. -1. -1. -1.]]
```

W_T = [[1 1 1 0 0 0][0 0 0 1 1 1]]

2

2a

A = 4 by 6

Therefore U is 4 by 4

S is 4 by 6

V is 6 by 6

2b

U is 4 by 4

S is 4 by 4

V tranpose is 4 by 6 -> V is 6 by 4.

2c

In [24]:
```
U,S,VT = np.linalg.svd(A,full_matrices=True)
np.set_printoptions(suppress=True)
print("U = \n ", U)
print("S = \n ", S)
print("VT = \n ", VT)
```

```
U =
  [[-0.5        -0.5        -0.70415281  0.06456638]
 [-0.5         0.5        -0.06456638 -0.70415281]
 [-0.5         0.5         0.06456638  0.70415281]
 [-0.5        -0.5         0.70415281 -0.06456638]]
S =
  [9.79795897 4.89897949 0.         0.        ]
VT =
  [[-0.40824829 -0.40824829 -0.40824829  0.40824829  0.40824829  0.4
0824829]
 [-0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.408
24829]
 [ 0.78860564 -0.57753998 -0.21106566  0.          0.          0.
]
 [-0.21158404 -0.5771605   0.78874454  0.         -0.         -0.
]
 [ 0.          0.         -0.         -0.57735027  0.78867513 -0.211
32487]
 [ 0.          0.         -0.         -0.57735027 -0.21132487  0.788
67513]]
```

i

In [20]:
```
UT_U = U.T @ U
VT_V = VT @ VT.T
np.set_printoptions(suppress=True)
print("UT_U = \n ", UT_U)
print("VT_V = \n ", VT_V)
```

```
UT_U =
  [[ 1. -0.  0. -0.]
 [-0.  1. -0.  0.]
 [ 0. -0.  1.  0.]
 [-0.  0.  0.  1.]]
VT_V =
  [[ 1.  0. -0.  0. -0. -0.]
 [ 0.  1. -0. -0.  0.  0.]
 [-0. -0.  1. -0. -0. -0.]
 [ 0. -0. -0.  1. -0. -0.]
 [-0.  0. -0. -0.  1.  0.]
 [-0.  0. -0. -0.  0.  1.]]
```

Both are orthonormal, since UT_U and VT_V both Identity Matrices.

ii

In [22]:
```
U_UT = U @ U.T
V_VT = VT.T @ VT
print("U_UT = \n ", U_UT)
print("V_VT = \n ", V_VT)
```

```
U_UT =
  [[ 1. -0. -0. -0.]
 [-0.  1. -0. -0.]
 [-0. -0.  1. -0.]
 [-0. -0. -0.  1.]]
V_VT =
  [[ 1.  0.  0. -0. -0. -0.]
 [ 0.  1. -0. -0. -0. -0.]
 [ 0. -0.  1.  0. -0. -0.]
 [-0. -0.  0.  1. -0. -0.]
 [-0. -0. -0. -0.  1. -0.]
 [-0. -0. -0. -0. -0.  1.]]
```

By the same reason, they are also orthonormal.

iii

The left singluar vector, U1.Transpose = [-0.5 -0.5 -0.5 -0.5]

The right singular vector, V1.Transpose = [-0.40824829 -0.40824829 -0.40824829 0.40824829 0.40824829 0.40824829]

iv

The rank of A is 2

2d

In [25]:
```
U,S,VT = np.linalg.svd(A,full_matrices=False)
np.set_printoptions(suppress=True)
print("U = \n ", U)
print("S = \n ", S)
print("VT = \n ", VT)
```

```
U =
  [[-0.5        -0.5        -0.70415281  0.06456638]
 [-0.5         0.5        -0.06456638 -0.70415281]
 [-0.5         0.5         0.06456638  0.70415281]
 [-0.5        -0.5         0.70415281 -0.06456638]]
S =
  [9.79795897 4.89897949 0.         0.        ]
VT =
  [[-0.40824829 -0.40824829 -0.40824829  0.40824829  0.40824829  0.4
0824829]
 [-0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.408
24829]
 [ 0.78860564 -0.57753998 -0.21106566  0.          0.          0.
]
 [-0.21158404 -0.5771605   0.78874454  0.         -0.         -0.
]]
```

In [27]:
```
UT_U = U.T @ U
VT_V = VT @ VT.T
np.set_printoptions(suppress=True)
print("UT_U = \n ", UT_U)
print("VT_V = \n ", VT_V)
```

```
UT_U =
  [[ 1. -0.  0. -0.]
 [-0.  1. -0.  0.]
 [ 0. -0.  1.  0.]
 [-0.  0.  0.  1.]]
VT_V =
  [[ 1.  0. -0.  0.]
 [ 0.  1. -0. -0.]
 [-0. -0.  1. -0.]
 [ 0. -0. -0.  1.]]
```

Since they are both identity matrices, they are both orthonormal.

In [26]:
```
U_UT = U @ U.T
V_VT = VT.T @ VT
print("U_UT = \n ", U_UT)
print("V_VT = \n ", V_VT)
```

```
U_UT =
  [[ 1. -0. -0. -0.]
 [-0.  1. -0. -0.]
 [-0. -0.  1. -0.]
 [-0. -0. -0.  1.]]
V_VT =
  [[ 1.          0.          0.         -0.         -0.         -0.
]
 [ 0.          1.         -0.         -0.         -0.         -0.
]
 [ 0.         -0.          1.          0.         -0.         -0.
]
 [-0.         -0.          0.          0.33333333  0.33333333  0.333
33333]
 [-0.         -0.         -0.          0.33333333  0.33333333  0.333
33333]
 [-0.         -0.         -0.          0.33333333  0.33333333  0.333
33333]]
```

rows in U are still orthonormal , since the matrix product an Identity Matrix

Rows in V is not orthonormal, since V@VT is not an identity matrix.

2e

Their singualr values are identical, however, VT of the full version has 2 more vectors that are not present in the economy version. Nonetheless, since the respective singluar values for these vectors are 0, they are not relevant.

2f

orthonormal basis spanned by columns: {u1, u2}, where

u1^T = [-0.5 -0.5 -0.5 -0.5]

u2^T = [-0.5 0.5 0.5 -0.5]

orthonormal basis spanned by rows: {v1, v2}, where

v1^T = [-0.40824829 -0.40824829 -0.40824829 0.40824829 0.40824829 0.40824829]

v2^T = [-0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.40824829]

2h

In [30]:
```
# recalculating SVD
U,S,VT = np.linalg.svd(A,full_matrices=True)
np.set_printoptions(suppress=True)
print("U = \n ", U)
print("S = \n ", S)
print("VT = \n ", VT)
```

```
U =
  [[-0.5        -0.5        -0.70415281  0.06456638]
 [-0.5         0.5        -0.06456638 -0.70415281]
 [-0.5         0.5         0.06456638  0.70415281]
 [-0.5        -0.5         0.70415281 -0.06456638]]
S =
  [9.79795897 4.89897949 0.         0.        ]
VT =
  [[-0.40824829 -0.40824829 -0.40824829  0.40824829  0.40824829  0.4
0824829]
 [-0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.408
24829]
 [ 0.78860564 -0.57753998 -0.21106566  0.          0.          0.
]
 [-0.21158404 -0.5771605   0.78874454  0.         -0.         -0.
]
 [ 0.          0.         -0.         -0.57735027  0.78867513 -0.211
32487]
 [ 0.          0.         -0.         -0.57735027 -0.21132487  0.788
67513]]
```

In [57]:
```
# rank 1 approximation
A_head = S[0] * U[:, 0:1]@VT[0:1, :]
print(A_head)
```

```
[[ 2.  2.  2. -2. -2. -2.]
 [ 2.  2.  2. -2. -2. -2.]
 [ 2.  2.  2. -2. -2. -2.]
 [ 2.  2.  2. -2. -2. -2.]]
```

In [60]:
```
# rank 2 approximation
A_head = S[0:2] * U[:, 0:2]@VT[0:2, :]
print(A_head)
```

```
[[ 3.  3.  3. -1. -1. -1.]
 [ 1.  1.  1. -3. -3. -3.]
 [ 1.  1.  1. -3. -3. -3.]
 [ 3.  3.  3. -1. -1. -1.]]
```

2i

min dimension is 2

In [61]:
```
U_head = U[:, 0:2]
S_head = S[0:2]
VT_head = VT[0:2, :]
print("U_head = \n", U_head)
print("S_head = \n", S_head)
print("VT_head = \n", VT_head)
```

```
U_head =
 [[-0.5 -0.5]
 [-0.5  0.5]
 [-0.5  0.5]
 [-0.5 -0.5]]
S_head =
 [9.79795897 4.89897949]
VT_head =
 [[-0.40824829 -0.40824829 -0.40824829  0.40824829  0.40824829  0.40
824829]
 [-0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.40824829 -0.408
24829]]
```

In [ ]: