In [1]:

```
import numpy as np
from scipy.io import loadmat

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

    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 centroids]) 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 certain centroid
                centroids.append( X[np.random.choice(len(X))] )
    return np.array(centroids) , C

# Load data for activity
in_data = loadmat('Period11Activity.mat')
X = in_data['X']

rows, cols = np.shape(X)
```

In [2]:

```
# k-means with 2 clusters
centroids, C = kMeans(X.transpose(), K = 2)

print('X = ', X, sep="\n", end='\n\n')
print('centroid assigned = ',C, sep="\n", end='\n\n')
print('centroids =', centroids.T.round(3), sep="\n", end='\n\n')
```

```
X =
[[ 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]]

centroid assigned =
[1 0 0 0 1 0 0]

centroids =
[[4.6 5.5]
 [4.8 9.5]
 [4.6 5. ]
 [4.4 9. ]
 [4.8 5.5]]
```

In [10]:

```
# Construct rank-2 approximation using clusters

Xhat_2 = np.zeros((rows,cols),float)
for i in range(cols):
    Xhat_2[:,i]=centroids.transpose()[:,C[i]]

print('Rank-2 Approximation = ', Xhat_2.round(3), sep="\n", end='\n\n'
)

print('Original Matrix = ', X.round(3), sep="\n", end='\n\n')
```

```
Rank-2 Approximation =
[[5.5 4.6 4.6 4.6 5.5 4.6 4.6]
 [9.5 4.8 4.8 4.8 9.5 4.8 4.8]
 [5.  4.6 4.6 4.6 5.  4.6 4.6]
 [9.  4.4 4.4 4.4 9.  4.4 4.4]
 [5.5 4.8 4.8 4.8 5.5 4.8 4.8]]

Original Matrix =
[[ 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]]
```

In [11]:

```
# k-means with 3 clusters
# Add code here . . .
centroids, C = kMeans(X.transpose(), K = 3)

print('centroid assigned = ',C, sep="\n", end='\n\n')
print('centroids =', centroids.T.round(3), sep="\n", end='\n\n')
```

```
centroid assigned =
[1 2 0 2 1 0 0]

centroids =
[[2.667 5.5   7.5  ]
 [5.    9.5   4.5  ]
 [2.667 5.    7.5  ]
 [5.    9.    3.5  ]
 [2.333 5.5   8.5  ]]
```

In [13]:
```
# Construct rank-3 approximation using clusters
# Add code here
Xhat_3 = np.zeros((rows,cols),float)
for i in range(cols):
    Xhat_3[:,i]=centroids.transpose()[:,C[i]]

print('Rank-3 Approximation = ', Xhat_3.round(3), sep="\n", end='\n\n'
)

print('Original Matrix = ', X.round(3), sep="\n", end='\n\n')
```

```
Rank-3 Approximation =
[[5.5   7.5   2.667 7.5   5.5   2.667 2.667]
 [9.5   4.5   5.    4.5   9.5   5.    5.   ]
 [5.    7.5   2.667 7.5   5.    2.667 2.667]
 [9.    3.5   5.    3.5   9.    5.    5.   ]
 [5.5   8.5   2.333 8.5   5.5   2.333 2.333]]

Original Matrix =
[[ 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]]
```

In [6]:
```
U,s,VT = np.linalg.svd(X,full_matrices=False)

print('U = ',U.round(3), sep="\n", end='\n\n')
print('Singular Values = ',s.round(3), sep="\n", end='\n\n')
print('V^T = ',VT.round(3), sep="\n", end='\n\n')
```

```
U =
[[-0.419 -0.319  0.565 -0.634  0.043]
 [-0.506  0.469  0.402  0.428 -0.424]
 [-0.402 -0.372 -0.582 -0.106 -0.592]
 [-0.466  0.552 -0.424 -0.318  0.444]
 [-0.436 -0.485 -0.019  0.55   0.521]]

Singular Values =
[32.952 10.165  1.788  0.699  0.407]

V^T =
[[-0.418 -0.38  -0.25  -0.456 -0.536 -0.3   -0.184]
 [ 0.441 -0.695  0.289 -0.34   0.177  0.04   0.301]
 [-0.191 -0.286 -0.664  0.329  0.3   -0.142  0.472]
 [ 0.329  0.445 -0.363 -0.626  0.276 -0.301  0.062]
 [ 0.174 -0.306 -0.252  0.121  0.385 -0.023 -0.806]]
```

3ci

X = [u1 u2 ... ur] W = S@V^T, taking the first r rows.

In [21]:
```
# svd rank-1 approximation
T = U[:,0]
print('Taste for rank-1 = ', T.round(3), sep="\n", end='\n\n')

W = s[0]*VT[0,:]
print('Weights for rank-1 = ',W.round(3), sep="\n", end='\n\n')

X_1 = s[0]*U[:,[0]]@VT[[0],:]
print("Rank-1 Approximation = ",X_1.round(3), sep="\n", end='\n\n')
print("first taste captures the average for each user.")
```

```
Taste for rank-1 =
[-0.419 -0.506 -0.402 -0.466 -0.436]

Weights for rank-1 =
[-13.773 -12.521  -8.24  -15.017 -17.647  -9.886  -6.068]

Rank-1 Approximation =
[[5.766 5.242 3.45  6.286 7.387 4.139 2.54 ]
 [6.964 6.331 4.167 7.593 8.923 4.999 3.068]
 [5.538 5.035 3.313 6.038 7.095 3.975 2.44 ]
 [6.419 5.835 3.84  6.998 8.224 4.607 2.828]
 [6.006 5.461 3.594 6.549 7.696 4.311 2.646]]

first taste captures the average for each user.
```

In [22]:
```
# svd rank-2 approximation
T = U[:,0:2]

print('Taste for rank-2 = ', T.round(3), sep="\n", end='\n\n')

W = s[0:2]@VT[0:2,:]
print('Weights for rank-2 = ', W.round(3), sep="\n", end='\n\n')

X_2 = s[0:2]*U[:,0:2]@VT[0:2,:]
print('Rank-2 Approximation = ',Xhat_2.round(3), sep="\n", end='\n\n')
print("second taste cpatures their preference towards sci-fi movies.")
```

```
Taste for rank-2 =
[[-0.419 -0.319]
 [-0.506  0.469]
 [-0.402 -0.372]
 [-0.466  0.552]
 [-0.436 -0.485]]

Weights for rank-2 =
[ -9.285 -19.581  -5.306 -18.475 -15.845  -9.483  -3.009]

Rank-2 Approximation =
[[5.5 4.6 4.6 4.6 5.5 4.6 4.6]
 [9.5 4.8 4.8 4.8 9.5 4.8 4.8]
 [5.  4.6 4.6 4.6 5.  4.6 4.6]
 [9.  4.4 4.4 4.4 9.  4.4 4.4]
 [5.5 4.8 4.8 4.8 5.5 4.8 4.8]]

second taste cpatures their preference towards sci-fi movies.
```

In [18]:
```
# Use svd to predict Jon's ratings

# first two tastes
T = U[:,:2]

# tastes for which we have ratings
G = T[:2,:2]

# ratings for first two movies
y = np.array([[6], [4]])

# use first two movies to find weights for tastes
a = np.linalg.inv(G.transpose()@G)@G.transpose()@y

# now use weights and tastes to predict all ratings
Jon_ratings = T@a

print('Jon ratings =',Jon_ratings.round(3), sep="\n", end='\n\n')
```

```
Jon ratings =
[[6.   ]
 [4.   ]
 [6.014]
 [3.231]
 [6.832]]
```

2

see other pdf

In [ ]:
```

```