In [73]:

```
### Problem 3 Devin Bresser ###
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('movie.mat')
#loadmat() loads a matlab workspace into a python dictionary, where the names of the var
#in the dictionary.  To see what variables are loaded, uncomment the line below:
#print([key for key in in_data])

X = in_data['X']
X_swapped = X[:, [1, 0] + list(range(2, X.shape[1]))] # swap 1st and 2nd columns
X
```

Out[73]:

```
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]], dtype=uint8)
```

In [2]:

```
import numpy as np

def gram_schmidt(B):
    """Orthogonalize a set of vectors stored as the columns of matrix B."""
    # Get the number of vectors.
    m, n = B.shape
    # Create new matrix to hold the orthonormal basis
    U = np.zeros([m,n])
    for j in range(n):
        # To orthogonalize the vector in column j with respect to the
        # previous vectors, subtract from it its projection onto
        # each of the previous vectors.
        v = B[:,j].copy()
        for k in range(j):
            v -= np.dot(U[:, k], B[:, j]) * U[:, k]
        if np.linalg.norm(v)>1e-10:
            U[:, j] = v / np.linalg.norm(v)
    return U

# if __name__ == '__main__':
#     B1 = np.array([[1.0, 1.0, 0.0], [2.0, 2.0, 0.0], [2.0, 2.0, 1.0]])
#     A1 = gram_schmidt(B1)
#     print(A1)
#     A2 = gram_schmidt(np.random.rand(4,2)@np.random.rand(2,5))
#     print(A2.transpose()@A2)
```

In [3]:

```
# Problem 3a

column_of_ones = np.ones((5,1))
X_tilde = np.hstack((column_of_ones, X))
X_tilde
```

Out[3]:

```
array([[ 1.,  4.,  7.,  2.,  8.,  7.,  4.,  2.],
       [ 1.,  9.,  3.,  5.,  6., 10.,  5.,  5.],
       [ 1.,  4.,  8.,  3.,  7.,  6.,  4.,  1.],
       [ 1.,  9.,  2.,  6.,  5.,  9.,  5.,  4.],
       [ 1.,  4.,  9.,  2.,  8.,  7.,  4.,  1.]])
```

In [46]:

```
T = gram_schmidt(X_tilde)
print(np.linalg.matrix_rank(T))
print(T)
```

```
5
[[ 4.47213595e-01 -3.65148372e-01 -6.32455532e-01 -5.16397779e-01
```

```
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -8.43769499e-15]
 [ 4.47213595e-01  5.47722558e-01  3.16227766e-01 -3.87298335e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.00000000e-01]
 [ 4.47213595e-01 -3.65148372e-01  2.24693342e-15  6.45497224e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.00000000e-01]
 [ 4.47213595e-01  5.47722558e-01 -3.16227766e-01  3.87298335e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.00000000e-01]
 [ 4.47213595e-01 -3.65148372e-01  6.32455532e-01 -1.29099445e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.00000000e-01]]
```

In [9]:
```
print(1/(5**0.5))

# Problem 3a comment: Yes, the first column of U_tilde is equal to t_1.
```

```
0.4472135954999579
```

In [65]:
```
# Problem 3b

# from previous problem:
# When T is an nxr matrix of orthonormal columns,
# ||min_w x-Tw||^2 = T^T x

# In this case, T is an n=5 x r=1 matrix
# the solution ||min_w x-Tw||^2 is given by T^T x


t_1 = T[:, 0].reshape(-1, 1) # define t_1 as the first column of T
w_1 = t_1.T @ X # find minimum solution w_1
X_rank1 = t_1 @ w_1 # find rank 1 approximation of X
residual_rank1 = X - X_rank1
print(X_rank1, "\n\n", residual_rank1)
print(f"mean of the rank 1 residual: {np.mean(np.abs((residual_rank1)))}")
```

```
[[6.  5.8 3.6 6.8 7.8 4.4 2.6]
 [6.  5.8 3.6 6.8 7.8 4.4 2.6]
 [6.  5.8 3.6 6.8 7.8 4.4 2.6]
 [6.  5.8 3.6 6.8 7.8 4.4 2.6]
 [6.  5.8 3.6 6.8 7.8 4.4 2.6]]

 [[-2.   1.2 -1.6  1.2 -0.8 -0.4 -0.6]
 [ 3.  -2.8  1.4 -0.8  2.2  0.6  2.4]
 [-2.   2.2 -0.6  0.2 -1.8 -0.4 -1.6]
 [ 3.  -3.8  2.4 -1.8  1.2  0.6  1.4]
 [-2.   3.2 -1.6  1.2 -0.8 -0.4 -1.6]]
mean of the rank 1 residual: 1.5657142857142856
```

In [66]:
```
# Problem 3c

t_2 = np.hstack((t_1,T[:, 1].reshape(-1,1))) # define t_2 as first 2 cols of T
w_2 = t_2.T @ X # find minimum solution w_2
X_rank2 = t_2 @ w_2 # find rank 2 approximation of X
residual_rank2 = X - X_rank2
print(X_rank2, "\n\n", residual_rank2)
print(f"mean of the rank 2 residual: {np.mean(np.abs((residual_rank2)))}")

# Problem 3c comment:
# t_2 is comprised of two taste vectors
# The first column is just the normalization vector to get the baseline
# The second column (-0.37, 0.55, -0.37, 0.55, -0.37) represents a taste vector
# that shows a dislike of sci-fi and preference for romance movies.

# This approximation results in a pretty low residual matrix, so my conclusion
# would be that the sci-fi/romance taste vector is very important to
# explain the trends in X.
```

```
[[4.         8.         2.33333333 7.66666667 6.66666667 4.
```

```
  1.33333333]
 [9.         2.5        5.5        5.5        9.5        5.
  4.5       ]
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [9.         2.5        5.5        5.5        9.5        5.
  4.5       ]
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]]

 [[ 1.77635684e-15 -1.00000000e+00 -3.33333333e-01  3.33333333e-01
   3.33333333e-01  1.77635684e-15  6.66666667e-01]
 [-3.55271368e-15  5.00000000e-01 -5.00000000e-01  5.00000000e-01
   5.00000000e-01 -1.77635684e-15  5.00000000e-01]
 [ 1.77635684e-15  3.55271368e-15  6.66666667e-01 -6.66666667e-01
  -6.66666667e-01  1.77635684e-15 -3.33333333e-01]
 [-3.55271368e-15 -5.00000000e-01  5.00000000e-01 -5.00000000e-01
  -5.00000000e-01 -1.77635684e-15 -5.00000000e-01]
 [ 1.77635684e-15  1.00000000e+00 -3.33333333e-01  3.33333333e-01
   3.33333333e-01  1.77635684e-15 -3.33333333e-01]]
mean of the rank 2 residual: 0.35238095238095324
```

In [68]:

```python
# Problem 3d

t_3 = np.hstack((t_2,T[:, 2].reshape(-1,1))) # define t_3 as first 3 cols of T
w_3 = t_3.T @ X # find minimum solution w_3
X_rank3 = t_3 @ w_3 # find rank 3 approximation of X
residual_rank3 = X - X_rank3
print(X_rank3, "\n\n", residual_rank3)
print(f"mean of the rank 3 residual: {np.mean(np.abs(residual_rank3))}")

# Problem 3d comment:
# Increasing the rank in this case does not meaningfully reduce
# the residual error. The rank 2 approximation was a good approximation.
# Furthermore, it is difficult to interpret what the taste vector t_3
# means in terms of the data. (-0.632, 0.316, 0.224, -0.316, 0.632)
# This indicates a mild preference for Pride & Prejudice and The Martian,
# a strong preference for Star Wars, but a dislike of Star Trek and
# Sense & Sensibility. The taste vector is too specific to be useful
# to include in the approximation.
```

```
[[4.         7.         2.53333333 7.46666667 6.46666667 4.
  1.53333333]
 [9.         3.         5.4        5.6        9.6        5.
  4.4       ]
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [9.         2.         5.6        5.4        9.4        5.
  4.6       ]
 [4.         9.         2.13333333 7.86666667 6.86666667 4.
  1.13333333]]

 [[ 5.77315973e-15  3.10862447e-14 -5.33333333e-01  5.33333333e-01
   5.33333333e-01  1.19904087e-14  4.66666667e-01]
 [-5.32907052e-15 -1.55431223e-14 -4.00000000e-01  4.00000000e-01
   4.00000000e-01 -7.10542736e-15  6.00000000e-01]
 [ 1.77635684e-15  0.00000000e+00  6.66666667e-01 -6.66666667e-01
  -6.66666667e-01  1.77635684e-15 -3.33333333e-01]
 [-1.77635684e-15  1.53210777e-14  4.00000000e-01 -4.00000000e-01
  -4.00000000e-01  3.55271368e-15 -6.00000000e-01]
 [-2.66453526e-15 -3.01980663e-14 -1.33333333e-01  1.33333333e-01
   1.33333333e-01 -7.99360578e-15 -1.33333333e-01]]
mean of the rank 3 residual: 0.24380952380952686
```

In [ ]:

In [ ]: