3a)

In [1]:

```
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([[ 1.0,4.0,7.0,2.0,8.0,7.0,4.0,2.0],[1.0,9.0,3.0,5.0,6.0,10.0,5.0,5.0],[1
    X =  np.array([[ 4.0,7.0,2.0,8.0,7.0,4.0,2.0],[9.0,3.0,5.0,6.0,10.0,5.0,5.0],[4.0,8.0,3
    A1 = gram_schmidt(B1)
    print(A1.round(5))
    t1 = A1[:,[0]]
    print(t1)
```

```
[[ 0.44721 -0.36515 -0.63246 -0.5164   0.       0.       0.       0.     ]
 [ 0.44721  0.54772  0.31623 -0.3873   0.       0.       0.       0.5    ]
 [ 0.44721 -0.36515  0.        0.6455   0.       0.       0.       0.5    ]
 [ 0.44721  0.54772 -0.31623  0.3873   0.       0.       0.      -0.5    ]
 [ 0.44721 -0.36515  0.63246 -0.1291   0.       0.       0.      -0.5    ]]
[[0.4472136]
 [0.4472136]
 [0.4472136]
 [0.4472136]
 [0.4472136]]
```

Since the first column is just a column of ones, on applying the first step of Gram-Schmidt orthogonalization, we obtain an orthonormal vector which normalizes the column of ones. Thus, we get t1 as defined in the question as the first column.

3b)

In [2]:

```
# rank 1 approximation
print("t1.T@t1 = \n",t1.T@t1)
#solution for W
Wrank1 = t1.T@X
print("\nt1.T@X = \n",Wrank1.round(5))

#RESIDUAL CALCULATION
residual1 = X - t1@Wrank1

print("\n X - t1@Wrank1 = \n",residual1)
```

```
t1.T@t1 =
 [[1.]]

t1.T@X =
 [[13.41641 12.96919  8.04984 15.20526 17.44133  9.8387   5.81378]]

 X - t1@Wrank1 =
 [[-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]]
```

3c)

In [3]:

```
# rank 2 approximation
t12 = A1[:,:2]

print("t12.T@t12 = \n",(t12.T@t12).round(5))
#solution for W
Wrank2 = t12.T@X
print("\nt12.T@X = \n",Wrank2.round(5))

#RESIDUAL CALCULATION
residual2 = X - t12@Wrank2

print("\n X - t12@Wrank2 = \n",residual2.round(5))
```

```
t12.T@t12 =
 [[ 1. -0.]
 [-0.  1.]]

t12.T@X =
 [[13.41641 12.96919  8.04984 15.20526 17.44133  9.8387   5.81378]
 [ 5.47723 -6.02495  3.46891 -2.37346  3.10376  1.09545  3.46891]]

 X - t12@Wrank2 =
 [[ 0.      -1.      -0.33333  0.33333  0.33333  0.       0.66667]
 [ 0.       0.5     -0.5      0.5      0.5      0.       0.5    ]
 [ 0.       0.       0.66667 -0.66667 -0.66667  0.      -0.33333]
 [ 0.      -0.5      0.5     -0.5     -0.5      0.      -0.5    ]
 [ 0.       1.      -0.33333  0.33333  0.33333  0.      -0.33333]]
```

Since Jennifer (who's ratings are in column 2 of X) heavily prefers scifi over romantic movies, unlike Jake (who's ratings are in column 1 of X), we see that that the addition of a second column heavily alters the approximation.

3d)

In [4]:

```
# rank 3 approximation
t123 = A1[:,:3]

print("t123.T@t123 = \n",(t123.T@t123).round(5))
#solution for W
Wrank3 = t123.T@X
print("\nt123.T@X = \n",Wrank3.round(5))

#RESIDUAL CALCULATION
residual3 = X - t123@Wrank3

print("\n X - t123@Wrank3 = \n",residual3.round(5))
```

```
t123.T@t123 = 
 [[ 1. -0.  0.]
 [-0.  1. -0.]
 [ 0. -0.  1.]]

t123.T@X = 
 [[13.41641 12.96919  8.04984 15.20526 17.44133  9.8387   5.81378]
 [ 5.47723 -6.02495  3.46891 -2.37346  3.10376  1.09545  3.46891]
 [ 0.       1.58114 -0.31623  0.31623  0.31623  0.       -0.31623]]

 X - t123@Wrank3 = 
 [[ 0.       0.      -0.53333  0.53333  0.53333  0.       0.46667]
 [ 0.      -0.      -0.4      0.4      0.4      0.       0.6     ]
 [ 0.       0.       0.66667 -0.66667 -0.66667  0.      -0.33333]
 [ 0.       0.       0.4     -0.4     -0.4      0.      -0.6     ]
 [-0.      -0.      -0.13333  0.13333  0.13333 -0.      -0.13333]]
```

We can see that as we increase the rank in the approximation, with every column that is added from the taste profile matrix, the error goes to zero in those columns.

3e)

In [5]:

```
B1 = np.array([[ 1.0,7.0,4.0,2.0,8.0,7.0,4.0,2.0],[1.0,3.0,9.0,5.0,6.0,10.0,5.0,5.0],[1.0,8
X =  np.array([[ 7.0,4.0,2.0,8.0,7.0,4.0,2.0],[3.0,9.0,5.0,6.0,10.0,5.0,5.0],[8.0,4.0,3.0,7
A1 = gram_schmidt(B1)
t12 = A1[:,:2]

print("t12.T@t12 = \n",(t12.T@t12).round(5))
Wrank2 = t12.T@X
print("\nt12.T@X = \n",Wrank2.round(5))

residual2 = X - t12@Wrank2

print("\n X - t12@Wrank2 = \n",residual2.round(5))
t123 = A1[:,:3]

print("\nt123.T@t123 = \n",(t123.T@t123).round(5))
Wrank3 = t123.T@X
print("\nt123.T@X = \n",Wrank3.round(5))

residual3 = X - t123@Wrank3

print("\n X - t123@Wrank3 = \n",residual3.round(5))
```

```
t12.T@t12 =
 [[1. 0.]
 [0. 1.]]

t12.T@X =
 [[12.96919 13.41641  8.04984 15.20526 17.44133  9.8387   5.81378]
 [ 6.22896 -5.29783 -3.43556  2.376   -2.92183 -1.05957 -3.43556]]

 X - t12@Wrank2 =
 [[ 0.       -0.97938 -0.93814  0.74227 -0.23711 -0.19588  0.06186]
 [ 0.        0.61856 -0.14433  0.26804  0.8866   0.12371  0.85567]
 [ 0.       -0.12887  0.6134  -0.63918 -0.76804 -0.02577 -0.3866 ]
 [ 0.       -0.23196  0.30412 -0.35052 -0.58247 -0.04639 -0.69588]
 [ 0.        0.72165  0.16495 -0.02062  0.70103  0.14433  0.16495]]

t123.T@t123 =
 [[ 1.  0.  0.]
 [ 0.  1. -0.]
 [ 0. -0.  1.]]

t123.T@X =
 [[12.96919 13.41641  8.04984 15.20526 17.44133  9.8387   5.81378]
 [ 6.22896 -5.29783 -3.43556  2.376   -2.92183 -1.05957 -3.43556]
 [ 0.        1.39032  0.57467 -0.2966   1.09372  0.27806  0.57467]]

 X - t123@Wrank3 =
 [[ 0.        0.       -0.53333  0.53333  0.53333  0.        0.46667]
 [-0.       -0.       -0.4      0.4      0.4     -0.        0.6     ]
 [ 0.        0.        0.66667 -0.66667 -0.66667  0.       -0.33333]
 [ 0.        0.        0.4     -0.4     -0.4      0.       -0.6     ]
 [-0.       -0.       -0.13333  0.13333  0.13333 -0.       -0.13333]]
```

We can see that the rank-2 approximations are completely different to the first case where Jake and Jennifer's values were unaltered. This is because, in the Gram-Schmidt process, we first normalize the first vector and then use the results of that normalization to compute the next normal vector from the next column. Order matters here.

But since only those two columns are switched, the remaining result is unaltered, and the Gram-Schmidt process is the same. Thus, the rank-3 approximation is the same in both cases.

In [ ]:

```

```