Assignment 3

1. *Polynomial fitting.* Suppose we observe pairs of points (a_i, b_i), $i = 1, \ldots, m$ representing measurements from a scientific experiment. The variables a_i are the experimental conditions and the b_i correspond to the measured response in each condition. We fit a degree $p < m$ polynomial to these data. In other words, we want to find the coefficients of a degree p polynomial $w(a)$ so that $w(a_i) \approx b_i$ for $i = 1, 2, \ldots, m$.

a) Suppose $w(a)$ is a degree p polynomial. Write the general expression for $w(a_i) = b_i$.

$$w(a_i) = x_0 + x_1 a_i + x_2 a_i^2 + \ldots + x_{p-1} a_i^{p-1} + x_p a_i^p = b_i$$

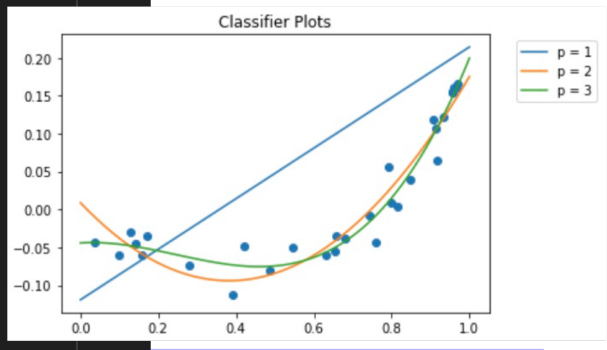
b) Express the $i = 1, \ldots, m$ equations as a system in matrix form $\boldsymbol{Ax} = \boldsymbol{d}$ while defining $\boldsymbol{A}$ and $\boldsymbol{d}$. What is the form/structure of $\boldsymbol{A}$ in terms of the given a_i?

$$A = \begin{bmatrix} 1 & a_1 & \cdots & a_1^{p-1} & a_1^p \\ \vdots & & & & \vdots \\ 1 & a_m & \cdots & a_m^{p-1} & a_m^p \end{bmatrix} \quad x = \begin{bmatrix} x_0 \\ \vdots \\ x_p \end{bmatrix} \quad d = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

A is a $m \times p+1$ matrix, where $m < p$, so $m \leq p+1$

c) Write a script to find the least-squares model fit to the $m = 30$ data points in `polydata.mat`. Plot the points and the polynomial fits for $p = 1, 2, 3$.

```
from scipy.sparse.construct import vstack
from google.colab import drive
#drive.mount('/content/drive')
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('/content/drive/My Drive/Colab Notebooks/polydata.mat')

a = in_data['a']
b = in_data['b']

# for p=1 (1+x)
A=np.column_stack(([1]*len(a), a))
x=np.linalg.inv(np.transpose(A)@A)@np.transpose(A)@b

# for p=2 (1+x+x^2)
A_2=np.column_stack((A, a**2))
x_2=np.linalg.inv(np.transpose(A_2)@A_2)@np.transpose(A_2)@b

# for p=3 (1+x+x^2+x^3)
A_3=np.column_stack((A_2, a**3))
x_3=np.linalg.inv(np.transpose(A_3)@A_3)@np.transpose(A_3)@b

plt.scatter(a, b)
plt.title('Classifier Plots')
l = np.linspace(0, 1, 100)
plt.plot(x, label='p = 1')

y_2 = np.array([np.sum(np.array([x_2[i]*(j**i) for i in range(len(x_2))])) for j in l])
plt.plot(l, y_2, label='p = 2')

y_3 = np.array([np.sum(np.array([x_3[i]*(j**i) for i in range(len(x_3))])) for j in l])
plt.plot(l, y_3, label='p = 3')

plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')

plt.show()
```



2. Least Squares Approximation of Matrices.

a) Derive the solution to least-squares problem $\min_{\boldsymbol{w}} \|\boldsymbol{x} - \boldsymbol{T}\boldsymbol{w}\|_2^2$ when $\boldsymbol{T}$ is an n-by-r matrix of orthonormal columns. Your solution should not involve a matrix inverse.

$Tw = x \longrightarrow$ typical solution $(T^TT)^{-1}T^Tx = w$

If columns of a matrix are orthonormal, then $T^TT = I$.

So the solution becoms $(I)^{-1}T^Tx = w$

$\downarrow$

$T^Tx = w$

b) Let $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_p \end{bmatrix}$ be an n-by-p matrix. Use the least-squares problems $\min_{\boldsymbol{w}_i} \|\boldsymbol{x}_i - \boldsymbol{T}\boldsymbol{w}_i\|_2^2$ to find $\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_p \end{bmatrix}$ in the approximation $\boldsymbol{X} \approx \boldsymbol{T}\boldsymbol{W}$. Your solution should express $\boldsymbol{W}$ as a function of $\boldsymbol{T}$ and $\boldsymbol{X}$.

$x_i = Tw_i \longrightarrow w_i = T^Tx_i \longrightarrow W = T^TX$

3. We return to the movies rating problem of Activity 5. The ratings 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

A matrix $\boldsymbol{X}$ containing this data is available in the file `movie.mat` and the csv file `movie.csv`. Our goal is to approximate $\boldsymbol{X}$ using r "tastes", the columns of $\boldsymbol{T}$, that is, $\boldsymbol{X} \approx \boldsymbol{T}\boldsymbol{W}$ where $\boldsymbol{T}$ is 5-by-r. You will use a Gram-Schmidt orthogonalization code to find a set of tastes that approximate the ratings. A script that implements Gram-Schmidt orthogonalization is available.

Define a 5-by-r taste matrix $\boldsymbol{T} = \begin{bmatrix} \boldsymbol{t}_1 & \boldsymbol{t}_2 & \cdots & \boldsymbol{t}_r \end{bmatrix}$ with orthonormal columns and the r-by-7 weight matrix

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{17} \\ w_{21} & w_{22} & \cdots & w_{27} \\ & & \vdots & \\ w_{r1} & w_{r2} & \cdots & w_{r7} \end{bmatrix}$$

a) In Activity 5 you found the baseline (average) rating for each friend by requiring the first basis vector in the taste matrix to be

$$\boldsymbol{t}_1 = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

You have noticed by now that the first vector in the Gram-Schmidt procedure is a scaled version of the first vector of the matrix, so you decide to define an augmented matrix $\tilde{\boldsymbol{X}} = \begin{bmatrix} \mathbf{1} & \boldsymbol{X} \end{bmatrix}$ where $\mathbf{1}$ is a column vector containing five unity entries. Apply a Gram-Schmidt orthogonalization code to $\tilde{\boldsymbol{X}}$ to find a set of orthonormal basis vectors. Is the first basis vector you obtain equal to $\boldsymbol{t}_1$?

$$X = \left[\begin{array}{c:ccccccc} 1 & 4 & 7 & 2 & 8 & 7 & 4 & 2 \\ 1 & 9 & 3 & 5 & 6 & 10 & 5 & 5 \\ 1 & 4 & 8 & 3 & 7 & 6 & 4 & 1 \\ 1 & 9 & 2 & 6 & 5 & 9 & 5 & 4 \\ 1 & 4 & 9 & 2 & 8 & 7 & 4 & 1 \end{array}\right]$$

$\tilde{X} =$

```
[[ 4.47213590e-01 -3.65148336e-01 -6.32455111e-01 -5.16397297e-01
   8.46741617e-01 -8.46742570e-01 -8.46742332e-01 -7.27982879e-01]
 [ 4.47213590e-01  5.47722578e-01  3.16227555e-01 -3.87298763e-01
  -2.50066072e-01  2.50063717e-01  2.50064343e-01  4.70365405e-01]
 [ 4.47213590e-01 -3.65148336e-01  3.90354842e-07  6.45497501e-01
  -1.01331025e-01  1.01333857e-01  1.01333089e-01  3.42496514e-01]
 [ 4.47213590e-01  5.47722578e-01 -3.16228002e-01  3.87298137e-01
   9.67276767e-02 -9.67266411e-02 -9.67268124e-02 -3.38535190e-01]
 [ 4.47213590e-01 -3.65148336e-01  6.32455945e-01 -1.29099414e-01
  -4.48186547e-01  4.48185831e-01  4.48185951e-01  1.29951343e-01]]
```

First basis = 0.44721359 → $\frac{1}{\sqrt{5}}$ = 0.44721359

The first basis vector is equal to t_1.

b) Use your solution to the preceding problem in this homework assignment to find the rank-1 approximation of $\boldsymbol{X}$ using only $\boldsymbol{t}_1$. That is, find $\boldsymbol{W}$ so that $\boldsymbol{X} \approx \boldsymbol{t}_1\boldsymbol{W}$. Use $\boldsymbol{W}$ to compute $\boldsymbol{t}_1\boldsymbol{W}$. This gives you each friend's baseline ratings. Also compute the residual error $\boldsymbol{X} - \boldsymbol{t}_1\boldsymbol{W}$.

$$W = T^T X \longrightarrow W = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} X$$

$W =$

```
[[13.4164077  12.96919411  8.04984462 15.20526206 17.44133002  9.83869898
   5.81377667]]
```

$$t_1 W = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} W = \begin{bmatrix} 6 & 5.8 & 3.6 & 6.8 & 7.8 & 4.4 & 2.6 \\ 6 & 5.8 & 3.6 & 6.8 & 7.8 & 4.4 & 2.6 \\ 6 & 5.8 & 3.6 & 6.8 & 7.8 & 4.4 & 2.6 \\ 6 & 5.8 & 3.6 & 6.8 & 7.8 & 4.4 & 2.6 \\ 6 & 5.8 & 3.6 & 6.8 & 7.8 & 4.4 & 2.6 \end{bmatrix}$$

$$X - t_1 W = \begin{bmatrix} -2 & 1.2 & -1.6 & 1.2 & -0.8 & -0.4 & -0.6 \\ 3 & -2.8 & 1.4 & -0.8 & 2.2 & 0.6 & 2.4 \\ -2 & 2.2 & -0.6 & 0.2 & -1.8 & -0.4 & -1.6 \\ 3 & 3.8 & 2.4 & -1.8 & 1.2 & 0.6 & 1.4 \\ -2 & 3.2 & -1.6 & 1.2 & -0.8 & -0.4 & -1.6 \end{bmatrix}$$

c) Now find a rank-2 approximation using $\boldsymbol{T} = \begin{bmatrix} \boldsymbol{t}_1 & \boldsymbol{t}_2 \end{bmatrix}$. That is, find $\boldsymbol{W}$ so that $\boldsymbol{X} \approx \boldsymbol{TW}$. Use $\boldsymbol{W}$ to compute $\boldsymbol{TW}$. This gives you a rank-2 approximation to the ratings. Also compute the residual error $\boldsymbol{X} - \boldsymbol{TW}$. How does $\boldsymbol{t}_2$ relate to the distinction between sci-fi and and romance movie preferences?

$$t_2 = \begin{bmatrix} -0.37 \\ 0.55 \\ -0.37 \\ 0.55 \\ -0.37 \end{bmatrix} \quad W = T^T X \rightarrow W = \begin{bmatrix} \frac{1}{\sqrt{5}} & -0.37 \\ \frac{1}{\sqrt{5}} & 0.55 \\ \frac{1}{\sqrt{5}} & -0.37 \\ \frac{1}{\sqrt{5}} & 0.55 \\ \frac{1}{\sqrt{5}} & -0.37 \end{bmatrix}^T X$$

$W =$

```
[[13.4164077  12.96919411  8.04984462 15.20526206 17.44133002  9.83869898
   5.81377667]
 [ 5.47722638 -6.02494717  3.46891001 -2.37346336  3.10376227  1.09544575
   3.46890986]]
```

$$TW = \begin{bmatrix} 4 & 8 & 2.3 & 7.6 & 6.6 & 4 & 1.3 \\ 9 & 2.5 & 5.5 & 5.5 & 9.5 & 5 & 4.5 \\ 4 & 8 & 2.3 & 7.6 & 6.6 & 4 & 1.3 \\ 9 & 2.5 & 5.5 & 5.5 & 9.5 & 5 & 4.5 \\ 4 & 8 & 2.3 & 7.6 & 6.6 & 4 & 1.3 \end{bmatrix}$$

$X - TW =$

```
[[ 2.39303045e-07 -9.99999292e-01 -3.33333197e-01  3.33333966e-01
   3.33333814e-01  2.98609233e-07  6.66666724e-01]
 [-4.07825688e-07  4.99999734e-01 -5.00000247e-01  4.99999635e-01
   4.99999515e-01 -2.66107286e-07  4.99999811e-01]
 [ 2.39303045e-07  7.08417570e-07  6.66666803e-01 -6.66666034e-01
  -6.66666186e-01  2.98609233e-07 -3.33333276e-01]
 [-4.07825688e-07 -5.00000266e-01  4.99999753e-01 -5.00000365e-01
  -5.00000485e-01 -2.66107286e-07 -5.00000189e-01]
 [ 2.39303045e-07  1.00000071e+00 -3.33333197e-01  3.33333966e-01
   3.33333814e-01  2.98609233e-07 -3.33333276e-01]]
```

In t_2 the values corresponding with sci fi movies are negative while the values corresponding with romance movies are positive. This reflects how people who rated sci fi movies high will generally rate romance movies low and vice versa, since the values for each genre have different signs.

d) Now find a rank-3 approximation using $\boldsymbol{T} = \begin{bmatrix} \boldsymbol{t}_1 & \boldsymbol{t}_2 & \boldsymbol{t}_3 \end{bmatrix}$. That is, find $\boldsymbol{W}$ so that $\boldsymbol{X} \approx \boldsymbol{TW}$. Use $\boldsymbol{W}$ to compute $\boldsymbol{TW}$. This gives you a rank-3 approximation to the ratings. Also compute the residual error $\boldsymbol{X} - \boldsymbol{TW}$. Qualitatively discuss the effect of increasing the rank of the approximation on the residual error.

$W = T^T X \rightarrow W =$

```
[[ 1.34164077e+01  1.29691941e+01  8.04984462e+00  1.52052621e+01
   1.74413300e+01  9.83869898e+00  5.81377667e+00]
 [ 5.47722638e+00 -6.02494717e+00  3.46891001e+00 -2.37346336e+00
   3.10376227e+00  1.09544575e+00  3.46890986e+00]
 [ 8.75965952e-07  1.58114752e+00 -3.16227397e-01  3.16234728e-01
   3.16231715e-01  2.66410530e-06 -3.16228119e-01]]
```

$$TW = \begin{bmatrix} 4 & 7 & 2.5 & 7.4 & 6.5 & 4 & 1.5 \\ 9 & 3 & 5.4 & 5.6 & 9.6 & 5 & 4.4 \\ 4 & 8 & 2.3 & 7.6 & 6.6 & 4 & 1.3 \\ 9 & 2 & 5.6 & 5.4 & 9.4 & 5 & 4.6 \\ 4 & 9 & 2 & 7.8 & 6.8 & 4 & 1.1 \end{bmatrix}$$

$X - TW =$

```
[[ 7.93312188e-07  5.53502079e-06 -5.33332831e-01  5.33338236e-01
   5.33336178e-01  1.98353624e-06  4.66666634e-01]
 [-6.84830260e-07 -2.67979418e-06 -4.00000430e-01  3.99997500e-01
   3.99998333e-01 -1.10857079e-06  5.99999856e-01]
 [ 2.39302703e-07  9.12089817e-08  6.66666926e-01 -6.66666157e-01
  -6.66666310e-01  2.98608193e-07 -3.33333152e-01]
 [-1.30820725e-07  2.85363706e-06  3.99999795e-01 -3.99998089e-01
  -3.99999161e-01  5.76357410e-07 -6.00000375e-01]
 [-3.14706829e-07 -5.43759796e-06 -1.33333300e-01  1.33329432e-01
   1.33331185e-01 -1.38632000e-06 -1.33332922e-01]]
```

Increasing rank generally decreases the error, meaning the predicted values are closer to the actual. What is interesting is how the rank 2 approximation was very accurate for columns 1 and 6, and the rank 3 approximation is very accurate for columns 1, 2, and 6, so the each rank increase accurately predicts an extra column.

e) Suppose you interchange the order of Jake and Jennifer so that Jennifer's ratings are in the first column of $\boldsymbol{X}$ and Jake's ratings are in the second column. Does the rank-2 approximation change? Why or why not? Does the rank-3 approximation change? Why or why not?

The rank 2 approximation does change. Because the matrix T for the rank 2 matrix only contains t_1, which gives the baseline, and t_2, which is the basis vector derived from the first column, if the first column of X is changed then the resulting approximation would also look different. The rank 3 approximation, on the other hand, stays very similar. Rank 3 uses t_1, t_2, and t_3, so the basis vectors from the first two columns are used, thus the swap doesn't affect the result much.

4. Let $Q = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$.

a) Is $Q \succ 0$? (positive definite): symmetric matrix where $x^T A x > 0$ for every column $x \neq 0$ in $\mathbb{R}^n$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$x^T Q x = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

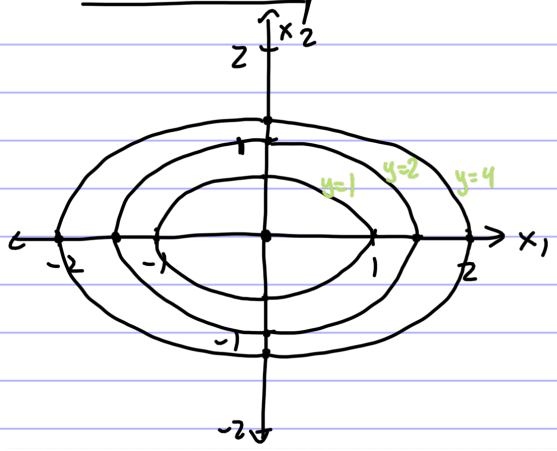
$$= \begin{bmatrix} x_1 & 2x_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + 2x_2^2 > 0$$

Because both x values are squared and added together, there is no way to get a negative number, so Q is positive definite.

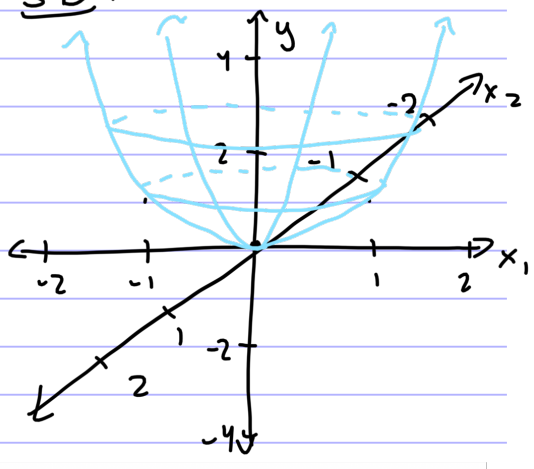
b) Sketch the surface $y = x^T Q x$ where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. If you find 3-D sketching too difficult, you may draw a contour map with labeled contours.

Sketch $y = x_1^2 + 2x_2^2$

Contour map:



3D:



5. Suppose $\mathbf{P} \succ 0$ and $\mathbf{Q} \succ 0$ are (symmetric) positive definite $n \times n$ matrices. Prove that $\mathbf{QPQ} \succ 0$.

P and Q both fulfill $x^T A x > 0$ and they are both symmetric.

We want $x^T(ABA)x > 0$.

Since A is symmetric, $A = A^T$ so $x^T A^T B A x > 0$

This means $(Ax)^T B (Ax) > 0$.

Let $y = Ax \longrightarrow y^T B y > 0$, which shows that ABA is positive definite.

Scripts for Problem 3

```
import numpy as np
from scipy.io import loadmat
from scipy.sparse.construct import vstack
import math
from google.colab import drive
#drive.mount('/content/drive')

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_data = loadmat('/content/drive/My Drive/Colab Notebooks/movie.mat')
    M=in_data['M']
    M=np.column_stack(([1]*5, M))
    M = M.astype('float32')

    B = gram_schmidt(M)

    print(B)
    print(1/math.sqrt(5))

    # part b
    M=in_data['M']
    W=np.atleast_2d(B[:, 0])@M
    print(W)
    base=np.atleast_2d(B[:, 0]).transpose()@W
    print(base)
    print(M-base)
```

```
[[ 4.47213590e-01 -3.65148336e-01 -6.32455111e-01 -5.16397297e-01
```

```
   8.46741617e-01 -8.46742570e-01 -8.46742332e-01 -7.27982879e-01]
  [ 4.47213590e-01  5.47722578e-01  3.16227555e-01 -3.87298763e-01
   -2.50066072e-01  2.50063717e-01  2.50064343e-01  4.70365405e-01]
  [ 4.47213590e-01 -3.65148336e-01  3.90354842e-07  6.45497501e-01
   -1.01331025e-01  1.01333857e-01  1.01333089e-01  3.42496514e-01]
  [ 4.47213590e-01  5.47722578e-01 -3.16228002e-01  3.87298137e-01
    9.67276767e-02 -9.67266411e-02 -9.67268124e-02 -3.38535190e-01]
  [ 4.47213590e-01 -3.65148336e-01  6.32455945e-01 -1.29099414e-01
   -4.48186547e-01  4.48185831e-01  4.48185951e-01  1.29951343e-01]]
0.4472135954999579
[[13.4164077  12.96919411  8.04984462 15.20526206 17.44133002  9.83869898
   5.81377667]]
[[5.99999986 5.79999986 3.59999991 6.79999984 7.79999981 4.39999989
  2.59999994]
 [5.99999986 5.79999986 3.59999991 6.79999984 7.79999981 4.39999989
  2.59999994]
 [5.99999986 5.79999986 3.59999991 6.79999984 7.79999981 4.39999989
  2.59999994]
 [5.99999986 5.79999986 3.59999991 6.79999984 7.79999981 4.39999989
  2.59999994]
 [5.99999986 5.79999986 3.59999991 6.79999984 7.79999981 4.39999989
  2.59999994]]
[[-1.99999986  1.20000014 -1.59999991  1.20000016 -0.79999981 -0.39999989
  -0.59999994]
 [ 3.00000014 -2.79999986  1.40000009 -0.79999984  2.20000019  0.60000011
   2.40000006]
 [-1.99999986  2.20000014 -0.59999991  0.20000016 -1.79999981 -0.39999989
  -1.59999994]
 [ 3.00000014 -3.79999986  2.40000009 -1.79999984  1.20000019  0.60000011
   1.40000006]
 [-1.99999986  3.20000014 -1.59999991  1.20000016 -0.79999981 -0.39999989
  -1.59999994]]


# part c
M=in_data['M']
T = B[:,[0, 1]]
print(T)
W_2=np.transpose(T)@M
print(W_2)
rank_two=T@W_2
print(rank_two)
print(M-rank_two)

[[ 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]]
[[ 0.44721359 -0.36514834]
 [ 0.44721359  0.54772258]
 [ 0.44721359 -0.36514834]
 [ 0.44721359  0.54772258]
 [ 0.44721359 -0.36514834]]
[[13.4164077  12.96919411  8.04984462 15.20526206 17.44133002  9.83869898
   5.81377667]
 [ 5.47722638 -6.02494717  3.46891001 -2.37346336  3.10376227  1.09544575
```

```
   3.46890986]]
 [[3.99999976 7.99999929 2.3333332  7.66666603 6.66666619 3.9999997
   1.33333328]
  [9.00000041 2.50000027 5.50000025 5.50000037 9.50000048 5.00000027
   4.50000019]
  [3.99999976 7.99999929 2.3333332  7.66666603 6.66666619 3.9999997
   1.33333328]
  [9.00000041 2.50000027 5.50000025 5.50000037 9.50000048 5.00000027
   4.50000019]
  [3.99999976 7.99999929 2.3333332  7.66666603 6.66666619 3.9999997
   1.33333328]]
 [[ 2.39303045e-07 -9.99999292e-01 -3.33333197e-01  3.33333966e-01
    3.33333814e-01  2.98609233e-07  6.66666724e-01]
  [-4.07825688e-07  4.99999734e-01 -5.00000247e-01  4.99999635e-01
    4.99999515e-01 -2.66107286e-07  4.99999811e-01]
  [ 2.39303045e-07  7.08417570e-07  6.66666803e-01 -6.66666034e-01
   -6.66666186e-01  2.98609233e-07 -3.33333276e-01]
  [-4.07825688e-07 -5.00000266e-01  4.99999753e-01 -5.00000365e-01
   -5.00000485e-01 -2.66107286e-07 -5.00000189e-01]
  [ 2.39303045e-07  1.00000071e+00 -3.33333197e-01  3.33333966e-01
    3.33333814e-01  2.98609233e-07 -3.33333276e-01]]


# part d
M=in_data['M']
T = B[:,[0, 1, 2]]
print(T)
W_3=np.transpose(T)@M
print(W_3)
rank_three=T@W_3
print(rank_three)
print(M-rank_three)

 [[ 4.47213590e-01 -3.65148336e-01 -6.32455111e-01]
  [ 4.47213590e-01  5.47722578e-01  3.16227555e-01]
  [ 4.47213590e-01 -3.65148336e-01  3.90354842e-07]
  [ 4.47213590e-01  5.47722578e-01 -3.16228002e-01]
  [ 4.47213590e-01 -3.65148336e-01  6.32455945e-01]]
 [[ 1.34164077e+01  1.29691941e+01  8.04984462e+00  1.52052621e+01
    1.74413300e+01  9.83869898e+00  5.81377667e+00]
  [ 5.47722638e+00 -6.02494717e+00  3.46891001e+00 -2.37346336e+00
    3.10376227e+00  1.09544575e+00  3.46890986e+00]
  [ 8.75965952e-07  1.58114752e+00 -3.16227397e-01  3.16234728e-01
    3.16231715e-01  2.66410530e-06 -3.16228119e-01]]
 [[3.99999921 6.99999446 2.53333283 7.46666176 6.46666382 3.99999802
   1.53333337]
  [9.00000068 3.00000268 5.40000043 5.6000025  9.60000167 5.00000111
   4.40000014]
  [3.99999976 7.99999991 2.33333307 7.66666616 6.66666631 3.9999997
   1.33333315]
  [9.00000013 1.99999715 5.6000002  5.39999809 9.39999916 4.99999942
   4.60000038]
  [4.00000031 9.00000544 2.1333333  7.86667057 6.86666881 4.00000139
   1.13333292]]
 [[ 7.93312188e-07  5.53502079e-06 -5.33332831e-01  5.33338236e-01
    5.33336178e-01  1.98353624e-06  4.66666634e-01]
  [-6.84830260e-07 -2.67979418e-06 -4.00000430e-01  3.99997500e-01
```

```
   3.99998333e-01 -1.10857079e-06  5.99999856e-01]
 [ 2.39302703e-07  9.12089817e-08  6.66666926e-01 -6.66666157e-01
  -6.66666310e-01  2.98608193e-07 -3.33333152e-01]
 [-1.30820725e-07  2.85363706e-06  3.99999795e-01 -3.99998089e-01
  -3.99999161e-01  5.76357410e-07 -6.00000375e-01]
 [-3.14706829e-07 -5.43759796e-06 -1.33333300e-01  1.33329432e-01
   1.33331185e-01 -1.38632000e-06 -1.33332922e-01]]


# part e
M=in_data['M']
M[:, [1, 0]] = M[:, [0, 1]]
print(M)

M=np.column_stack(([1]*5, M))
M = M.astype('float32')

B = gram_schmidt(M)

M=in_data['M']
print(M)

T = B[:,[0, 1]]
print(T)
W_3=np.transpose(T)@M
print(W_3)
rank_three=T@W_3
print(rank_three)
print(M-rank_three)

[[ 7  4  2  8  7  4  2]
 [ 3  9  5  6 10  5  5]
 [ 8  4  3  7  6  4  1]
 [ 2  9  6  5  9  5  4]
 [ 9  4  2  8  7  4  1]]
[[ 7  4  2  8  7  4  2]
 [ 3  9  5  6 10  5  5]
 [ 8  4  3  7  6  4  1]
 [ 2  9  6  5  9  5  4]
 [ 9  4  2  8  7  4  1]]
[[ 0.44721359  0.19264843]
 [ 0.44721359 -0.44951293]
 [ 0.44721359  0.35318872]
 [ 0.44721359 -0.61005324]
 [ 0.44721359  0.5137291 ]]
[[12.96919411 13.4164077   8.04984462 15.20526206 17.44133002  9.83869898
   5.81377667]
 [ 6.22896536 -5.29783055 -3.43556288  2.37599745 -2.92183347 -1.05956587
  -3.43556294]]
[[7.00000023 4.97938114 2.93814413 7.25773201 7.23711319 4.1958762
  1.93814415]
 [2.9999994  8.38144318 5.14432985 5.73195826 9.11340173 4.87628845
  4.1443299 ]
 [8.00000018 4.12886585 2.38659785 7.63917534 6.76804118 4.02577318
  1.38659785]
 [1.99999936 9.23195856 5.69587618 5.35051489 9.58247379 5.04639149
```

```
  4.69587624]
 [9.0000006  3.27835016 1.8350513  8.02061886 6.29896895 3.85567008
  0.8350513 ]]
[[-2.30848674e-07 -9.79381141e-01 -9.38144134e-01  7.42267994e-01
  -2.37113195e-01 -1.95876197e-01  6.18558537e-02]
 [ 5.99419026e-07  6.18556816e-01 -1.44329845e-01  2.68041737e-01
   8.86598266e-01  1.23711547e-01  8.55670104e-01]
 [-1.83163854e-07 -1.28865847e-01  6.13402153e-01 -6.39175345e-01
  -7.68041180e-01 -2.57731770e-02 -3.86597850e-01]
 [ 6.44553022e-07 -2.31958556e-01  3.04123816e-01 -3.50514889e-01
  -5.82473792e-01 -4.63914891e-02 -6.95876244e-01]
 [-5.99573116e-07  7.21649841e-01  1.64948696e-01 -2.06188600e-02
   7.01031053e-01  1.44329922e-01  1.64948703e-01]]
```