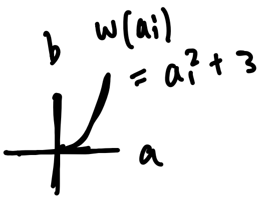
ECE 532 Assignment 3 - DEVIN BRESSER



1. *Polynomial fitting.* Suppose we observe pairs of points (a_i, b_i), $i = 1, \dots, m$ 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 $i = 1, 2, \dots, m$.

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

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

   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 $p = 1, 2, 3$.

1.) a.) $w(a)$ is degree - p polynomial

$$w(a_i) = x_0 + x_1 a_i + \cdots + x_p a_i^p$$

$$w(a_i) = \begin{bmatrix} 1 & a_i & \cdots & a_i^p \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{bmatrix} = b_i$$

$x_0 + x_1 a_1 + x_2 a_1^2 + x_3 a_3^3$

b.) $\underline{A}\, \underline{x} = \underline{d}$

$p = 3$

4 cols

$$\rightarrow \underbrace{\begin{bmatrix} 1 & a_1 & \cdots & a_1^p \\ 1 & a_2 & \cdots & a_2^p \\ \vdots & \vdots & & \vdots \\ 1 & a_m & \cdots & a_m^p \end{bmatrix}}_{m \times (p+1)} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}}_{(p+1) \times 1} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}}_{m \times 1}$$

$\underline{A}$ is a stack of features of the given a_i's.

The # of cols of $\underline{A}$ equals 1 more than the desired polynomial degree p.

The # of rows of $\underline{A}$ equals the number of samples m.

1c

In [52]:
```python
### Problem 1(c) Devin Bresser ###
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('polydata.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])

#print(f"a values: {in_data['a']} \nb values: {in_data['b']}")
a = in_data['a'];
d = in_data['b'];
```

In [48]:
```python
import numpy as np

def create_matrix(_a, _p):
    """
    create a matrix based on the ndarray 'a' and power 'p'
    input: list of input datapoints a, desired highest power p
    output: a matrix as shown in problem 1b.)

    """
    m = len(_a)
    matrix = np.zeros((m, _p+1))

    for i in range(m):
        for j in range(_p+1):
            matrix[i][j] = _a[i] ** j

    return np.around(matrix,5)
```

In [49]:
```python
def least_squares(_A,_d):
    """
    find the least squares solution x0 to the set of equations Ax=d
    input: feature matrix A, result vector d
    output: least squares weight vector x0

    """
    x0 = np.linalg.inv(_A.T @ _A) @ _A.T @ _d
    error = np.linalg.norm(_A @ x0 - _d)

    return x0
```

In [50]:
```python
[A_p1, A_p2, A_p3] = [create_matrix(in_data['a'],1),
                      create_matrix(in_data['a'],2),
                      create_matrix(in_data['a'],3)];
```

In [53]:
```python
[x0_p1, x0_p2, x0_p3] = [least_squares(A_p1,d),
                         least_squares(A_p2,d),
                         least_squares(A_p3,d)];
```

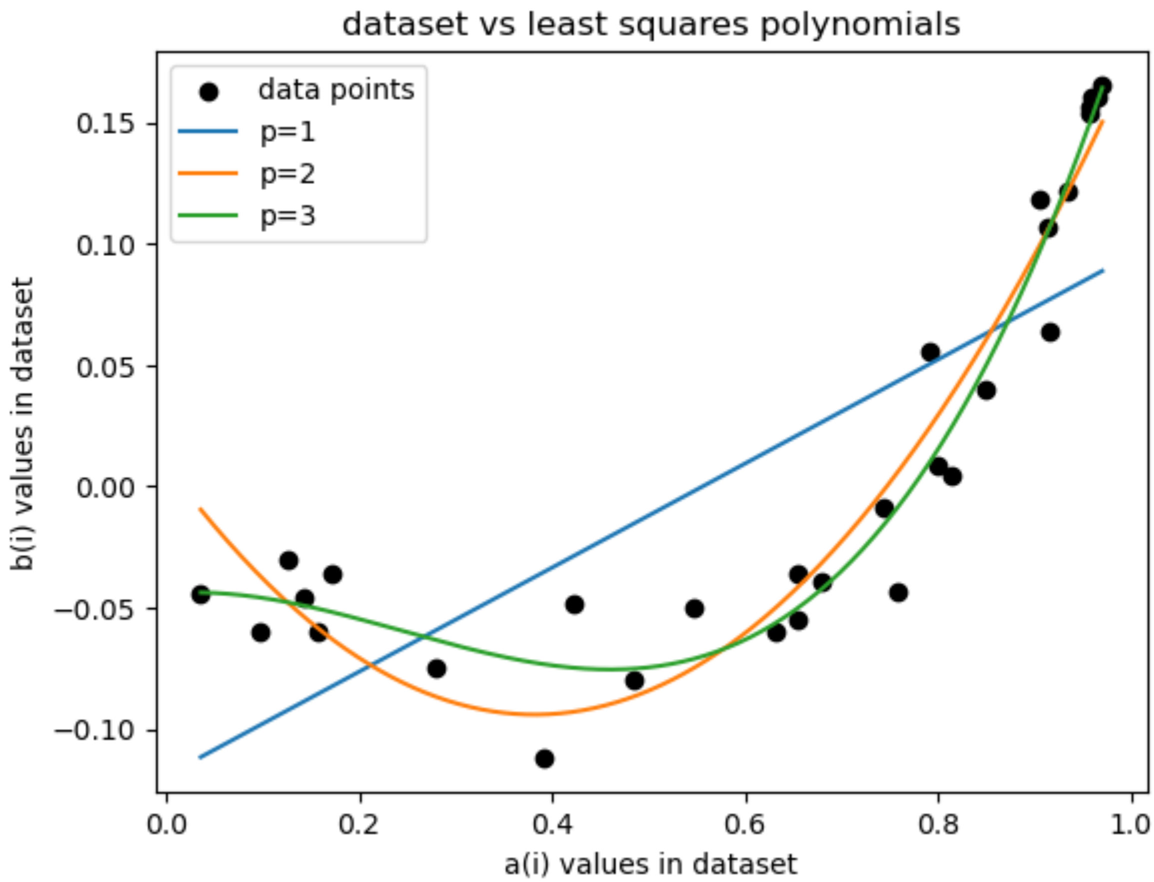
In [76]:
```python
import matplotlib.pyplot as plt
# plot the points
plt.scatter(a,d,color="black", label="data points")

# create smooth curves along the x-axis for each polynomial
x_dense = np.linspace(min(a), max(a), 400)
# polyval requires exponents to be descending
y_p1 = np.polyval(x0_p1[::-1], x_dense)
y_p2 = np.polyval(x0_p2[::-1], x_dense)
y_p3 = np.polyval(x0_p3[::-1], x_dense)
```

```
# plot the polynomial curves
plt.plot(x_dense, y_p1, label='p=1')
plt.plot(x_dense, y_p2, label='p=2')
plt.plot(x_dense, y_p3, label='p=3')

# add plot info
plt.xlabel('a(i) values in dataset')
plt.ylabel('b(i) values in dataset')
plt.title('dataset vs least squares polynomials')
plt.legend(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.

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 $\boldsymbol{X}$.

2a.) $\min_{\underline{w}} \|\underline{x} - \underline{T}\underline{w}\|_2^2$, $\underline{T}$ is $n \times r$ matrix of orthonormal columns.

$\Rightarrow \|\underline{x} - \underline{T}\underline{w}\|^2 = (\underline{x} - \underline{T}\underline{w})^T(\underline{x} - \underline{T}\underline{w})$ $\quad T^{-1}\underline{T}\underline{w} \exists \hat{\underline{x}}$

$= (\underline{x}^T - (\underline{T}\underline{w})^T)(\underline{x} - \underline{T}\underline{w})$

$= (\underline{x}^T - w^T T^T)(\underline{x} - \underline{T}\underline{w})$

$= \underline{x}^T\underline{x} - \underline{w}^T\underline{T}^T\underline{x} - \underline{x}^T\underline{T}\underline{w} + \underline{w}^T \cancelto{\underline{I}}{\underline{T}^T\underline{T}} \underline{w}$

$= \underline{x}^T\underline{x} - \underline{w}^T\underline{T}^T\underline{x} - \underline{x}^T\underline{T}\underline{w} + \underline{w}^T\underline{w}$

$\rightarrow$ Use gradient to find critical point

$\nabla_{\underline{w}} \left(\cancelto{0}{\underline{x}^T\underline{x}} - \underline{w}^T\underline{T}^T\underline{x} - \underline{x}^T\underline{T}\underline{w} + \underline{w}^T\underline{w} \right) = 0$

$\Rightarrow -\underline{T}^T\underline{x} - \underline{x}^T\underline{T} + 2\underline{w} = 0$

$\rightarrow 2\underline{w} = \underline{T}^T\underline{x} + \underline{x}^T\underline{T}$

$\rightarrow 2\underline{w} = 2\underline{T}^T\underline{x}$

$\rightarrow \underline{w} = \underline{T}^T\underline{x}$

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 $\boldsymbol{X}$.

$$\underline{X} = [\, \underline{x}_1 \ \underline{x}_2 \ \cdots \ \underline{x}_p \,]$$

$$\rightarrow \min_{\underline{w}_i} \| x_i - \underline{T}\,\underline{w}_i \|_2^2$$

every column weighted by T^T.

$$\underline{w}_i = \underline{T}^T \underline{x}_i \quad \rightarrow \quad \underline{W} = [\, \underline{T}^T \underline{x}_1 \ \ \underline{T}^T \underline{x}_2 \ \cdots \ \underline{T}^T \underline{x}_p \,]$$

$$\rightarrow \underline{W} = \underline{T}^T \underline{X}$$

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]:

```
# 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 [ ]:

3e.)

Swapped rank 2

```
[81]: # 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.
```

```
[[7.         4.97938144 2.93814433 7.25773196 7.2371134  4.19587629
  1.93814433]
 [3.         8.3814433  5.1443299  5.73195876 9.11340206 4.87628866
  4.1443299 ]
 [8.         4.12886598 2.38659794 7.63917526 6.76804124 4.0257732
  1.38659794]
 [2.         9.23195876 5.69587629 5.35051546 9.58247423 5.04639175
  4.69587629]
 [9.         3.27835052 1.83505155 8.02061856 6.29896907 3.8556701
  0.83505155]]

 [[ 0.00000000e+00 -9.79381443e-01 -9.38144330e-01  7.42268041e-01
  -2.37113402e-01 -1.95876289e-01  6.18556701e-02]
 [ 8.88178420e-16  6.18556701e-01 -1.44329897e-01  2.68041237e-01
   8.86597938e-01  1.23711340e-01  8.55670103e-01]
 [ 0.00000000e+00 -1.28865979e-01  6.13402062e-01 -6.39175258e-01
  -7.68041237e-01 -2.57731959e-02 -3.86597938e-01]
 [ 6.66133815e-16 -2.31958763e-01  3.04123711e-01 -3.50515464e-01
  -5.82474227e-01 -4.63917526e-02 -6.95876289e-01]
 [-1.77635684e-15  7.21649485e-01  1.64948454e-01 -2.06185567e-02
   7.01030928e-01  1.44329897e-01  1.64948454e-01]]
mean of the rank 2 residual: 0.364064801178203
```

Swapped rank 3

```
3]: # 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.
```

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

 [[ 1.50990331e-14  1.90958360e-14 -5.33333333e-01  5.33333333e-01
   5.33333333e-01  1.37667655e-14  4.66666667e-01]
 [-8.43769499e-15 -1.24344979e-14 -4.00000000e-01  4.00000000e-01
   4.00000000e-01 -7.10542736e-15  6.00000000e-01]
 [ 1.77635684e-15  2.22044605e-15  6.66666667e-01 -6.66666667e-01
  -6.66666667e-01  1.77635684e-15 -3.33333333e-01]
 [ 4.21884749e-15  5.32907052e-15  4.00000000e-01 -4.00000000e-01
  -4.00000000e-01  3.55271368e-15 -6.00000000e-01]
 [-1.24344979e-14 -1.42108547e-14 -1.33333333e-01  1.33333333e-01
   1.33333333e-01 -9.76996262e-15 -1.33333333e-01]]
mean of the rank 3 residual: 0.2438095238095271
```

The rank 2 approximation changes because the taste vectors created via G-S are in order of the columns of the data. So the 2nd taste vector is Based off Jennifer's ratings rather than Jake's when they are swapped.

So the span is just the span of $G\text{-}S\left(\begin{matrix} \vdots & \text{Jennifer} \end{matrix}\right)$

The rank 3 approximation does not change because it captures the exact same amount of the variance in X.

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

a) Is $\boldsymbol{Q} \succ 0$?

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

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

4a.) $\underline{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$

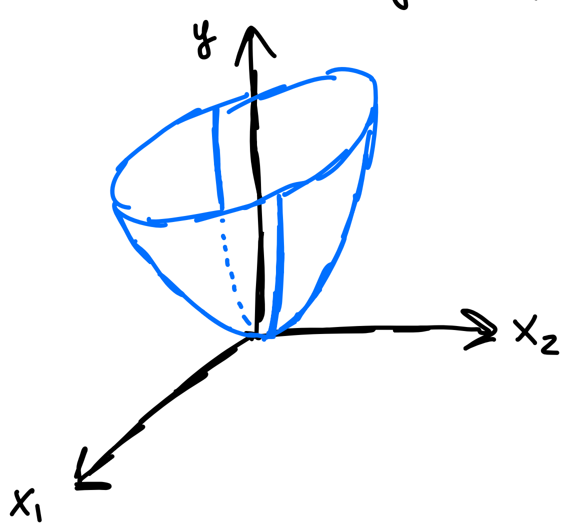
→ Eigenvalues of diagonal matrix are just the diagonal elements.

→ $\lambda_1 = 1, \quad \lambda_2 = 2$

→ All $\lambda > 0$, so $\underline{Q}$ is positive definite.

b.) $y = \underline{x}^T \underline{Q} \, \underline{x} = [x_1 \;\; x_2] \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$y = [x_1 + 2x_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + 2x_2^2$



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

positive symmetric iff $\underline{v}^T \underline{A}\, \underline{v} > 0$ for all $\underline{v} \neq 0$.

$\rightarrow \quad \underline{v}^T \underline{Q}\,\underline{P}\,\underline{Q}\,\underline{v} \overset{?}{>} 0$

Let $\underline{y} = \underline{Q}\,\underline{v}$

$$\underbrace{\underline{v}^T \underline{Q}}_{\underline{y}^T}\,\underline{P}\,\underbrace{\underline{Q}\,\underline{v}}_{\underline{y}} = \underline{y}^T \underline{P}\,\underline{y} \overset{?}{>} 0$$

since $\underline{P}$ is positive definite, $\underline{y}^T \underline{P}\,\underline{y} > 0$ for all $\underline{y} \neq 0$.

Thus $\underline{v}^T \underline{Q}\,\underline{P}\,\underline{Q}\,\underline{v} = \underline{y}^T \underline{P}\,\underline{y}$ satisfies the same property.

Thus $\underline{Q}\,\underline{P}\,\underline{Q} \succ 0$.