ENPM 673 - Perception of Autonomous Robots

Pradeep Gopal
Sahana Anbazhagan
Srikumar Muralidharan

February 8, 2020

1 Problem 1

Assume that you have a camera with a resolution of 5MP where the camera sensor is square shaped with a width of $14mm$. It is also given that the focal length of the camera is $15mm$.

1.1 Part 1

Compute the **Field of View** of the camera in the **horizontal** and **vertical** direction.

Solution:

The following information is available to us:

We have a 5MP camera ($5x10^6$ pixels), with a square shaped sensor of width $14mm$. So, we can compute the sensor area to be $196mm^2$. Also, since the sensor is square and there is no specific aspect ratio mentioned in the question, we take the aspect ratio to be $1:1$.
Also, since the aspect ratio is 1:1, the **vertical** and **horizontal field of view** have the same values.

Considering a thin lens, we can further equate the values of the focal length to the distance between the image and the lens, for an object placed at infinity.

Assumptions for thin lens include the following:

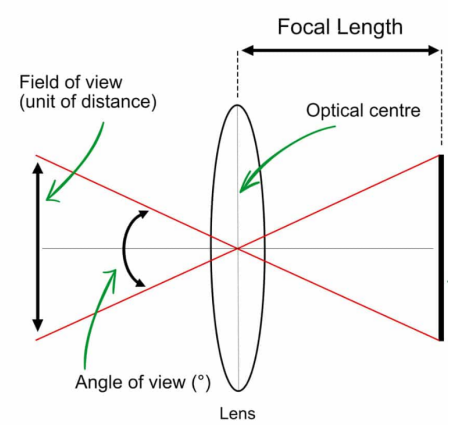

Figure 1: An image of Field of view

- Lens surfaces are spherical.
- Incoming light rays make a small angle with the optical axis.
- The lens thickness is small compared to the radii of curvature.
- The refractive index is the same for the media on both sides of the lens.

Refering to the image shown for this example, we can infer that the **Field of view (FOV)** can be represented as a function of focal length and width of screen. As mentioned previously, the values of horizontal and vertical FOV are equal and is given by the equation:

$$\theta = 2\text{arctan}(\frac{D_i}{2f}) \tag{1}$$

where θ is the angular FOV, D_i is width of square sensor ($14mm$) and f is focal length of thin lens ($15mm$). Substituting the values for each of them, we obtain the following results:

$$\theta = 2\text{arctan}(\frac{14}{2*15}) \tag{2}$$

$$\theta = 2\text{arctan}(0.4667) \tag{3}$$

$$\theta = 50.03^\circ \tag{4}$$

Thus, both the **horizontal and vertical FOV** have a value of 50.03°.

1.2 Part 2

Assuming you are detecting a square shaped object with width $5cm$, placed at a distance of $20m$ from the camera, compute the minimum number of pixels that the object will occupy in the image.

Solution:

We have to compute the number of pixels over a spread of the sensor area. Since the sensor is a square sheet of width $14mm$, we can easily say that the sensor area is $196mm^2$. Also, it is given that camera has a resolution of $5MP$, which comes up to be 5x10^6 pixels spread over the sensor area. So, for a unit area, we have:

$$\frac{5*10^6}{196} pixels/mm^2 \tag{5}$$

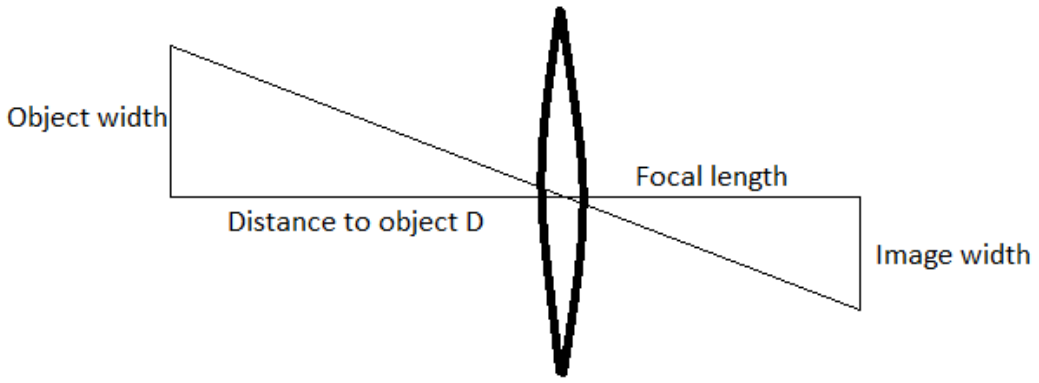

Figure 2: Object is at infinity, image is at focus

Clearly, the object is placed at a distance of $20m$ from the lens. Assuming its a thin lens with the properties stated above, and with a focal length of $15mm$, the object which is placed at a distance of $2m$ from the lens can be considered as an object placed at infinity. So, the image is formed at focus. Also, from the image, it can inferred as 2 similar triangles. So, the width of the image can be found out with the information provided.

$$\frac{\text{Width of object}}{\text{Distance of object from lens}} = \frac{\text{Width of image}}{\text{Focal length}} \tag{6}$$

$$\frac{50mm}{20000mm} = \frac{\text{Width of image}}{15mm} \tag{7}$$

Therefore, the image comes out to be a square of side $0.0375mm$. The resultant area of the image is 0.0375^2mm^2. Multiplying the same with the equation 5 framed above, we can compute the minimum number of pixels required to represent the image on the sensor screen.

$$\text{Minimum number of pixels} = \frac{5 * 10^6 * 0.0375 * 0.0375}{196}\text{pixels} \tag{8}$$

$$\text{Minimum number of pixels} = 36\text{pixels (rounded off from 35.87)} \tag{9}$$

So, we require a minimum of **36 pixels** to make sure that the image is completely visible.

2 Problem 2

Two files of 2D data points are provided in the form of CSV files (Dataset1 and Dataset2). The data represents measurements of a projectile with different noise levels and is shown in figure 1. Assuming that the projectile follows the equation of a parabola:

2.1 Part 1

Find the best method to fit a curve to the given data for each case. You have to plot the data and your best fit curve for each case. Submit your code along with the instructions to run it.

Solution:

Attaching below the codes and the output graphs

Code for dataset 1 with Least Square method

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

""" To read and plot the input data from the csv file """
data = pd.read_csv('data_1.csv')
x_axis = data.iloc[:,0]
y_axis = data.iloc[:,1]

"""
Building the model for this dataset
Calculating the A and B matrices to determine X,
B = inverse((transpose(X)*X)) * (transpose(X)*Y)
For a simple square matrix we can calculate the same as,
X = inverse(A)*B
```

```
where A is the coefficient matrix (consists of the x^n terms obtained from the csv file)
      Y is the data obtained from the csv file
      X is the final expected matrix that will give us the coefficients of the equation (unknowns)
"""
x1 = sum(x_axis)
x2 = sum(np.power(x_axis, 2))
x3 = sum(np.power(x_axis, 3))
x4 = sum(np.power(x_axis, 4))
xy = sum(x_axis * y_axis)
x2y = sum(np.power(x_axis, 2) * y_axis)
y = sum(y_axis)
A = ([x4,x3,x2], [x3, x2, x1], [x2, x1, 250])
B = ([x2y, xy, y])
A_inv = np.linalg.inv(A)
X = np.matmul(A_inv, B)

""" Printing the obtained values of the coefficients of the equation a, b, c """
print("The value of a is", X[0])
print("The value of b is", X[1])
print("The value of c is", X[2])

""" Plotting the original graph along with the curve fitted graph """
Y_pred = X[0] * (np.power(x_axis, 2)) + X[1] * x_axis + X[2] # The final equation of a parabola
#print(Y_pred) # To print the values of the above equation
plt.scatter(x_axis,y_axis, label = 'Initial dataset') # Graph with only the data points
plt.plot(x_axis, Y_pred, 'blue', label = 'Curve Fitting') # Graph with the curve fitted to the data
plt.legend()
plt.show()
```

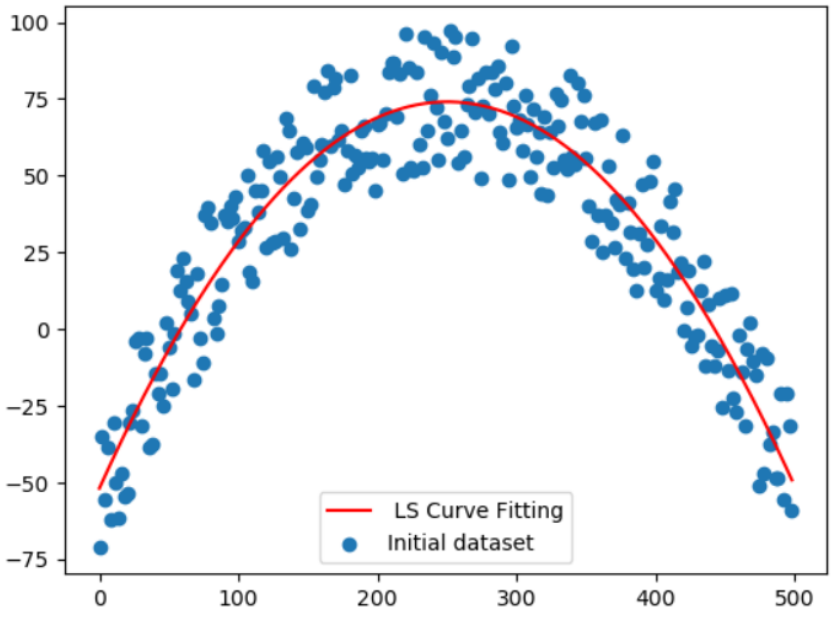

Figure 3: Output showing the dataset 1 with the curve fitted using LS method

Code for dataset 2 with Least Square method

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

""" To read and plot the input data from the csv file """
data = pd.read_csv('data_2.csv')
x_axis = data.iloc[:,0]
y_axis = data.iloc[:,1]

"""
Building the model for this dataset
Calculating the A and B matrices to determine X,
B = inverse((transpose(X)*X)) * (transpose(X)*Y)
For a simple square matrix we can calculate the same as,
X = inverse(A)*B
where A is the coefficient matrix (consists of the x^n terms obtained from the csv file)
      Y is the data obtained from the csv file
      X is the final expected matrix that will give us the coefficients of the equation (unknowns)
"""
x1 = sum(x_axis)
x2 = sum(np.power(x_axis, 2))
x3 = sum(np.power(x_axis, 3))
x4 = sum(np.power(x_axis, 4))
xy = sum(x_axis * y_axis)
x2y = sum(np.power(x_axis, 2) * y_axis)
y = sum(y_axis)
A = ([x4,x3,x2], [x3, x2, x1], [x2, x1, 250])
B = ([x2y, xy, y])
A_inv = np.linalg.inv(A)
X = np.matmul(A_inv, B)

""" Printing the obtained values of the coefficients of the equation a, b, c """
print("The value of a is", X[0])
print("The value of b is", X[1])
print("The value of c is", X[2])

""" Plotting the original graph along with the curve fitted graph """
Y_pred = X[0] * (np.power(x_axis, 2)) + X[1] * x_axis + X[2] # The final equation of a parabola
#print(Y_pred) # To print the values of the above equation
plt.scatter(x_axis,y_axis, label = 'Initial dataset') # Graph with only the data points
plt.plot(x_axis, Y_pred, 'red', label = 'Curve Fitting') # Graph with the curve fitted to the data
plt.legend()
plt.show()
```

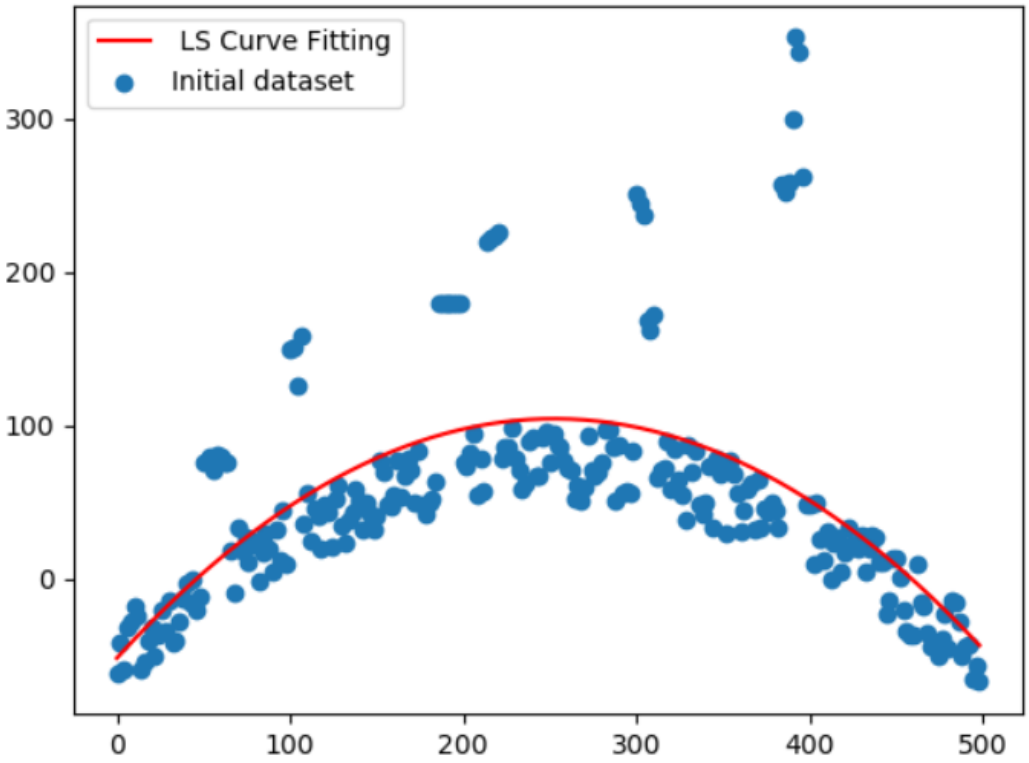

Figure 4: Output showing the dataset 2 with the curve fitted using LS method

Code for dataset 1 using LS with regularization

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

""" To read the input data from the csv file """
data = pd.read_csv('data_1.csv')
x = data.iloc[:,0]
y = data.iloc[:,1]

""" This is a scalar value multiplied to the identity matrix to obtain the curve using Least square with
R = ([5, 0, 0], [0, 5, 0], [0, 0, 5])

"""
Calculating the x matrix and determining the transpose of the same to bring it to determine B,
B = inverse((transpose(X)*X) + R) * (transpose(X)*Y)
where X is the coefficient matrix
      Y is the data obtained from the csv file
      R = aI, a is any scalar value and I is an identity matrix
      B is the final expected matrix that will give us the coefficients of the equation
"""
x_s = np.power(x,2)
x_m = np.transpose([x_s, x, np.ones(np.shape(x))])
#print(x_m)

"""
From B we obtained coefficients a, b, c in the equation y = a * (x**2) + b * x + c
We obtain the coeffients as follows:
a = -1.90461499e-03  b = 9.43431225e-01  c = -4.40364232e+01
"""
B = np.matmul(np.linalg.inv(np.matmul(np.transpose(x_m), x_m) + R),np.matmul(np.transpose(x_m),y))
print(B)

"""
The final equation for the parabolic dataset is obtained as Y = XB, since B is a 1x3 matrix in the previ
take the transpose of B to multiply it with the same
"""
y_new = np.matmul(x_m,np.transpose(B))

"""
Plotting and displaying the original dataset and the plotting the best fit line
"""
plt.scatter(x, y, label = 'XY_data')
plt.plot(x, y_new, 'r', label = 'Curve Fitting')
plt.legend()
plt.show()
```

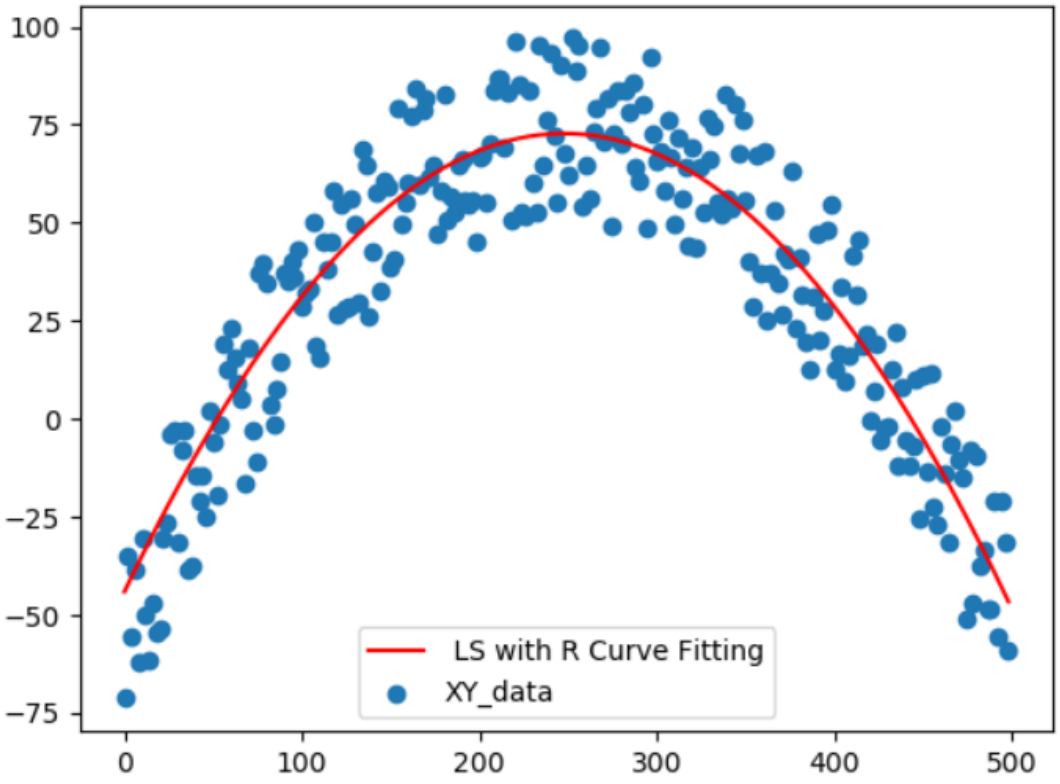

Figure 5: Output showing the dataset 2 with the curve fitted using LS method

Code for dataset 2 using LS with regularization

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

""" To read the input data from the csv file """
data = pd.read_csv('data_2.csv')
x = data.iloc[:,0]
y = data.iloc[:,1]

""" This is a scalar value multiplied to the identity matrix to obtain the curve using Least square with
R = ([8, 0, 0], [0, 8, 0], [0, 0, 8])

"""
Calculating the x matrix and determining the transpose of the same to bring it to determine B,
B = inverse((transpose(X)*X) + R) * (transpose(X)*Y)
where X is the coefficient matrix
      Y is the data obtained from the csv file
      R = aI, a is any scalar value and I is an identity matrix
      B is the final expected matrix that will give us the coefficients of the equation
"""
x_s = np.power(x,2)
x_m = np.transpose([x_s, x, np.ones(np.shape(x))])
#print(x_m)

"""
From B we obtained coefficients a, b, c in the equation y = a * (x**2) + b * x + c
We obtain the coeffients as follows:
a = -2.29745229e-03  b = 1.14536716e+00 c = -4.04561982e+01
"""
B = np.matmul(np.linalg.inv(np.matmul(np.transpose(x_m), x_m) + R),np.matmul(np.transpose(x_m),y))
print(B)

"""
The final equation for the parabolic dataset is obtained as Y = XB, since B is a 1x3 matrix in the previ
take the transpose of B to multiply it with the same
"""
y_new = np.matmul(x_m,np.transpose(B))

"""
Plotting and displaying the original dataset and the plotting the best fit line
"""
plt.scatter(x, y, label = 'XY_data')
plt.plot(x, y_new, 'r', label = 'Curve Fitting')
plt.legend()
plt.show()
```

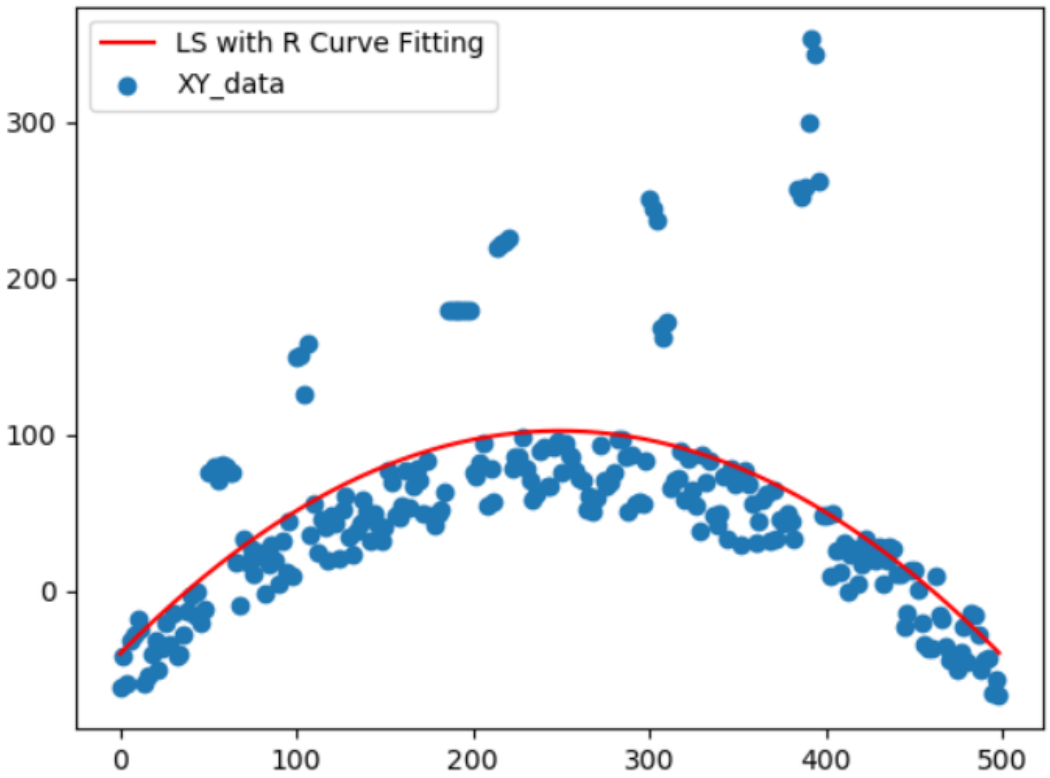

Figure 6: Output showing the dataset 2 with the curve fitted using LS method

Code for dataset 1 using Total Least Square method

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

""" To read and plot the input data from the csv file """
data = pd.read_csv('data_1.csv')
x_axis = data.iloc[:, 0]
y_axis = data.iloc[:, 1]

x_bar = 0
x_sq_bar = 0
y_bar = 0

for j in range(250):
    x_bar += x_axis[j]
    y_bar += y_axis[j]
    x_sq_bar += np.power(x_axis[j], 2)
U = np.zeros((250, 3))
for i in range(250):
    U[i][0] = np.power(x_axis[i], 2) - (x_sq_bar / 250)
    U[i][1] = np.power(x_axis[i], 1) - (x_bar / 250)
    U[i][2] = np.power(y_axis[i], 1) - (y_bar / 250)

capU = np.dot(np.transpose(U), U)
g = [0.0015, -0.7360, 0.6770]

V = np.dot(np.transpose(capU), capU)

lamATA, V_ATA = np.linalg.eig(V)
V_ATA = np.around(V_ATA, decimals=9)

G = V_ATA[:, 2]

a = g[0]
b = g[1]
c = g[2]

d = (a * x_sq_bar / 250) + (b * x_bar / 250) + (c * y_bar / 250)
new_y = (d / c) - (a * np.power(x_axis, 2) / c) - (b * x_axis / c)

plt.scatter(x_axis, y_axis, label='Given_data')
plt.plot(x_axis, new_y, "r", label='TLS_Fitting')
plt.legend()
plt.show()
```

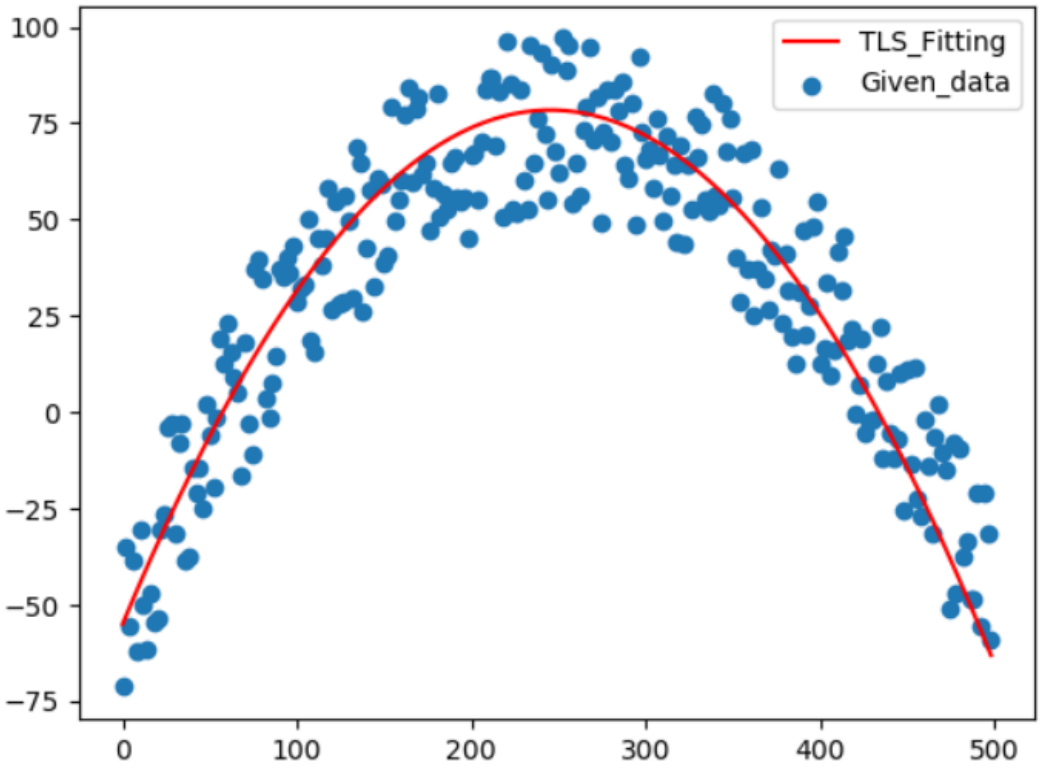

Figure 7: Output showing the dataset 2 with the curve fitted using LS method

Code for dataset 2 using Total Least Square method

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

""" To read and plot the input data from the csv file """
data = pd.read_csv('data_2.csv')
x_axis = data.iloc[:, 0]
y_axis = data.iloc[:, 1]

x_bar = 0
x_sq_bar = 0
y_bar = 0

for j in range(250):
    x_bar += x_axis[j]
    y_bar += y_axis[j]
    x_sq_bar += np.power(x_axis[j], 2)
U = np.zeros((250, 3))
for i in range(250):
    U[i][0] = np.power(x_axis[i], 2) - (x_sq_bar / 250)
    U[i][1] = np.power(x_axis[i], 1) - (x_bar / 250)
    U[i][2] = np.power(y_axis[i], 1) - (y_bar / 250)

capU = np.dot(np.transpose(U), U)
g = [0.0015, -0.7360, 0.6770]

V = np.dot(np.transpose(capU), capU)

lamATA, V_ATA = np.linalg.eig(V)
V_ATA = np.around(V_ATA, decimals=9)

G = V_ATA[:, 2]

a = g[0]
b = g[1]
c = g[2]

d = (a * x_sq_bar / 250) + (b * x_bar / 250) + (c * y_bar / 250)
new_y = (d / c) - (a * np.power(x_axis, 2) / c) - (b * x_axis / c)

plt.scatter(x_axis, y_axis, label='Given_data')
plt.plot(x_axis, new_y, "r", label='TLS_Fitting')
plt.legend()
plt.show()
```

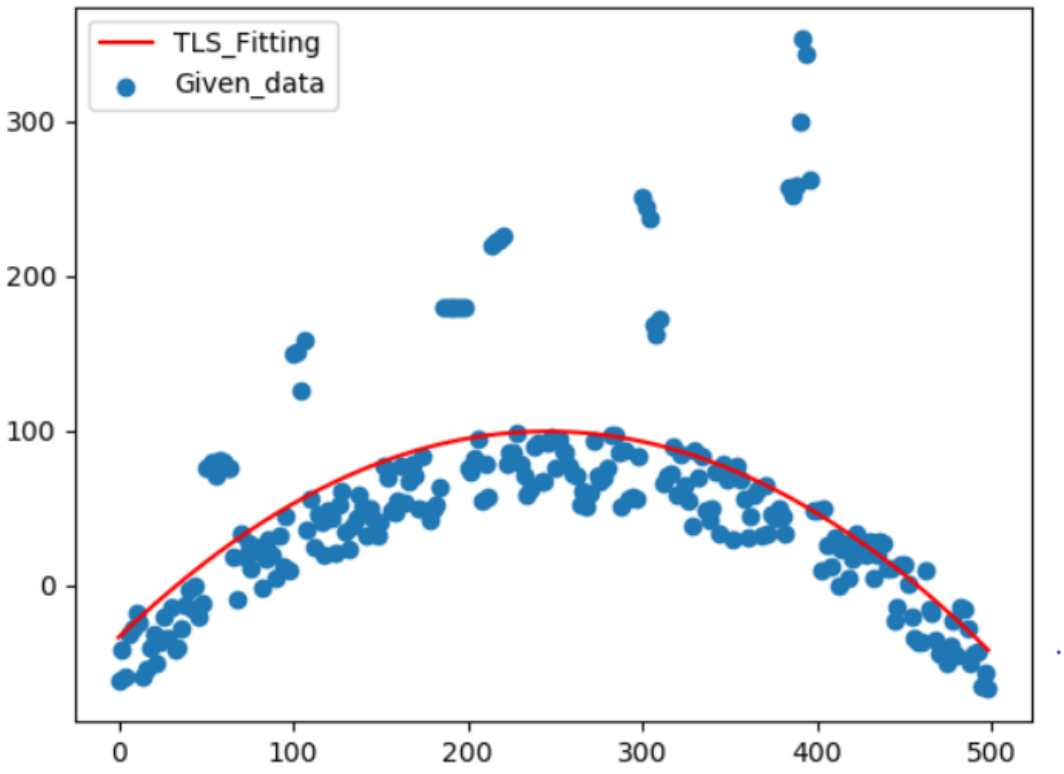

Figure 8: Output showing the dataset 2 with the curve fitted using LS method

For RANSAC implementation or also known as outlier rejection technique implementation, we import the dataset from the given .csv files and store the values in 2 arrays. For a fixed number of iterations (80 in our case), we are going to randomly choose 3 values from the original x array and with their index values, we choose corresponding values from the original y array. Using the Least-square model fitting method (explained earlier), we calculate the equation of a parabola that connects these three random x and y coordinates, for each iteration. As was done in LS method, we calculate A and B matrices and eventually calculate coefficients of the parabola equation and fit these 3 random pairs of points. Now, with the parabola equation, we can try to calculate the vertical distance between the other points in the data set and corresponding vertical point on parabola. This will constitute the error, which we will try to minimize. We try to take the absolute value of vertical distance and add it up in a variable called "threshold" in our case. After this, we find the average value of "threshold" and store it an array called "thr", the index being the iteration number. Now, we use this "thr" array to calculate the upper and lower boundaries for inliers. Our goal is to maximize the number of inliers and use the iteration that has the highest count of inliers for model fitting as it represents the data set best. In our case, we consider a particular dataset to be an inlier if its vertical distance from the parabola fits in the band of "thr[that iteration]" +/- 10. We have global variables that are used to keep track of the best fit in all the iterations and coefficients of parabola equation. If an iteration gives us the maximum number of inliers count, that iteration's parabola equation is stored and is used to print the best fit curve for our datasets.

Code for dataset 1 using RANSAC method

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv('data_1.csv')
x_axis = data.iloc[:, 0]
y_axis = data.iloc[:, 1]
thr = np.zeros((250, 1))
n_inliers = np.zeros((250, 1))
best_inlier_count = 0
for i in range(80):
    no_of_inliers = 0
    x = np.random.choice(x_axis, 3)
    y = [y_axis[x[0] / 2], y_axis[x[1] / 2], y_axis[x[2] / 2]]
    A = np.zeros((3, 3))
    B = np.zeros((3, 1))
    for j in range(3):
        A[0][0] += np.power(x[j], 4)
        A[0][1] += np.power(x[j], 3)
        A[0][2] += np.power(x[j], 2)
        A[1][2] += x[j]
        B[0][0] += np.power(x[j], 2) * y[j]
        B[1][0] += x[j] * y[j]
        B[2][0] += y[j]
    A[1][0] = A[0][1]
    A[1][1] = A[0][2]
    A[2][0] = A[1][1]
    A[2][1] = A[1][2]
    A[2][2] = 3
    x_i = np.dot(np.linalg.inv(A), B)
    threshold = 0
    y_est = np.zeros((250, 1))
    for k in range(250):
        y_est[k] = x_i[0] * np.power(x_axis[k], 2) + x_i[1] * x_axis[k] + x_i[2]
```

```
        threshold += abs(y_axis[k] - y_est[k])
    thr[i] = threshold / 250
    for l in range(250):
        if (abs(y_axis[l] - y_est[l]) + 10 > thr[i] and abs(y_axis[l] - y_est[l]) - 10 < thr[i]):
            no_of_inliers += 1
    n_inliers[i] = no_of_inliers;
    print("No. of inliers for this case" + str(n_inliers[i]))
    if (best_inlier_count < n_inliers[i]):
        best_inlier_count = n_inliers[i]
        best_x_i = x_i

        # The next four lines can be easily hidden to make it look less cluttered
    # but for our understanding, we have printed plots for each of the iterations.
    y = x_i[0] * np.power(x_axis, 2) + x_i[1] * np.power(x_axis, 1) + x_i[2]
    plt.scatter(x_axis, y_axis)
    plt.plot(x_axis, y, 'r')
    plt.show()

print("The best fit that we got till now:")
print("Maximum number of inliers: " + str(best_inlier_count))
best_y = best_x_i[0] * np.power(x_axis, 2) + best_x_i[1] * np.power(x_axis, 1) + best_x_i[2]
plt.scatter(x_axis, y_axis)
plt.plot(x_axis, best_y, 'r')
plt.show()
```

```
The best fit that we got till now:
Maximum number of inliers: [192.]
```

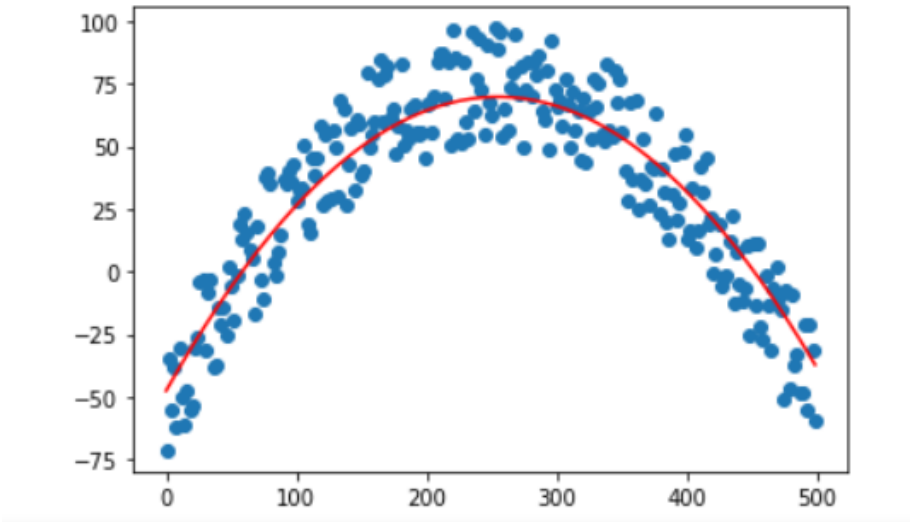

Figure 9: Output showing the dataset 2 with the curve fitted using LS method

Code for dataset 2 using RANSAC method

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv('data_2.csv')
x_axis = data.iloc[:, 0]
y_axis = data.iloc[:, 1]
thr = np.zeros((250, 1))
n_inliers = np.zeros((250, 1))
best_inlier_count = 0
for i in range(80):
    no_of_inliers = 0
    x = np.random.choice(x_axis, 3)
    y = [y_axis[x[0] / 2], y_axis[x[1] / 2], y_axis[x[2] / 2]]
    A = np.zeros((3, 3))
    B = np.zeros((3, 1))
    for j in range(3):
        A[0][0] += np.power(x[j], 4)
        A[0][1] += np.power(x[j], 3)
        A[0][2] += np.power(x[j], 2)
        A[1][2] += x[j]
        B[0][0] += np.power(x[j], 2) * y[j]
        B[1][0] += x[j] * y[j]
        B[2][0] += y[j]
    A[1][0] = A[0][1]
    A[1][1] = A[0][2]
    A[2][0] = A[1][1]
    A[2][1] = A[1][2]
    A[2][2] = 3
    x_i = np.dot(np.linalg.inv(A), B)
    threshold = 0
    y_est = np.zeros((250, 1))
    for k in range(250):
        y_est[k] = x_i[0] * np.power(x_axis[k], 2) + x_i[1] * x_axis[k] + x_i[2]
        threshold += abs(y_axis[k] - y_est[k])
    thr[i] = threshold / 250
    for l in range(250):
        if (abs(y_axis[l] - y_est[l]) + 10 > thr[i] and abs(y_axis[l] - y_est[l]) - 10 < thr[i]):
            no_of_inliers += 1
    n_inliers[i] = no_of_inliers;
    print("No. of inliers for this case" + str(n_inliers[i]))
    if (best_inlier_count < n_inliers[i]):
        best_inlier_count = n_inliers[i]
        best_x_i = x_i

        # The next four lines can be easily hidden to make it look less cluttered
    # but for our understanding, we have printed plots for each of the iterations.
    y = x_i[0] * np.power(x_axis, 2) + x_i[1] * np.power(x_axis, 1) + x_i[2]
    plt.scatter(x_axis, y_axis)
    plt.plot(x_axis, y, 'r')
    plt.show()
```

```
print("The best fit that we got till now:")
print("Maximum number of inliers: " + str(best_inlier_count))
best_y = best_x_i[0] * np.power(x_axis, 2) + best_x_i[1] * np.power(x_axis, 1) + best_x_i[2]
plt.scatter(x_axis, y_axis)
plt.plot(x_axis, best_y, 'r')
plt.show()
```

```
The best fit that we got till now:
Maximum number of inliers: [48.]
```

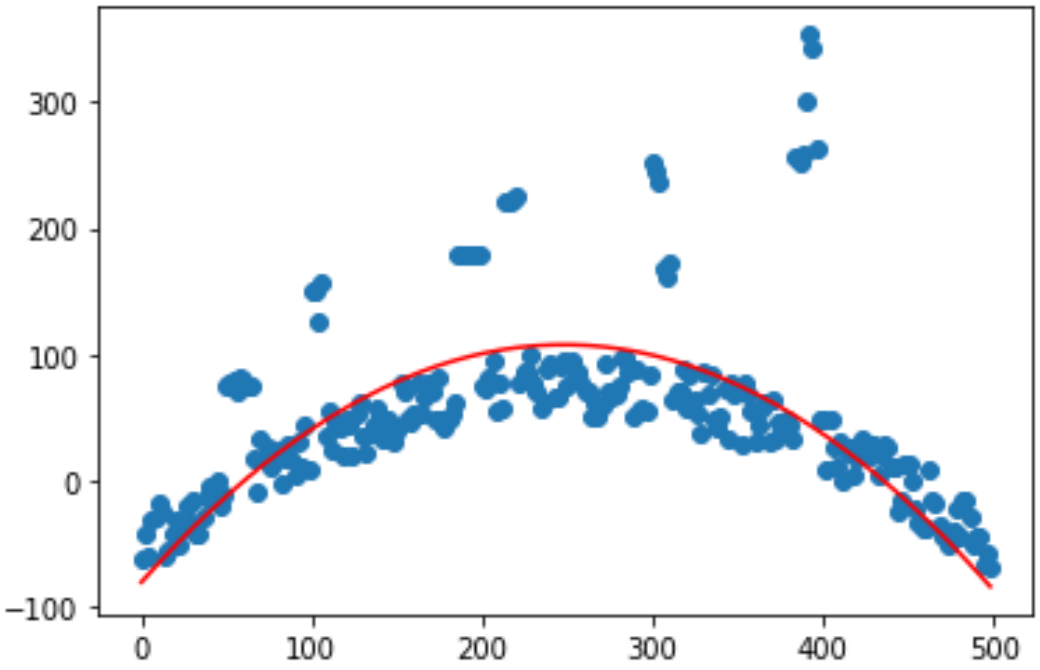

Figure 10: Output showing the dataset 2 with the curve fitted using LS method

2.2 Part 2

Briefly explain all the steps of your solution and discuss why your choice of outlier rejection technique is best for that case.
Solution:

We have implemented Least Square fitting, Total Least Square Fitting, Least Square Fitting with Regularization and outlier rejection technique.
In the Least Square Fitting method it can be observed that the first data set already has a best fit curve since there are no outliers. The second dataset has a few outliers which are not taken care of by this method and it gives a curve that fits closest to the data.
The method employed here is:
First the equation of a parabola is taken as follows−

$$y = ax^2 + bx + c \tag{10}$$

We know that the error is nothing but the (actual − observed). This leads us to the following error equation:

$$E = \sum_{i=n}^{n} [y_i - f(x_i)]^2 \tag{11}$$

Substituting the parabola equation in the above equation we obtain:

$$E = \sum_{i=n}^{n} [y_i - (ax_i^2 + bx_i + c)]^2 \tag{12}$$

Now to find an optimal solution we need to minimize a, b and c. This is done by partially differentiating the error equation by a,b and c and equation them to 0. We will obtain 3 equations, and we will have 3 unknown variables.

$$\frac{\partial E}{\partial a} = 0 \qquad \frac{\partial E}{\partial b} = 0 \qquad \frac{\partial E}{\partial c} = 0 \tag{13}$$

$$\implies \sum_{i=n}^{n} \quad 2[y_i - (ax_i^2 + bx_i + c)](-x_i^2) = 0 \tag{14}$$

$$\implies \sum_{i=n}^{n} \quad (x_i^2)[y_i - ax_i^2 - bx_i - c)] = 0 \tag{15}$$

$$\implies \sum_{i=n}^{n} \quad [y_i x_i^2 - ax_i^4 - bx_i^3 - cx_i^2] = 0 \tag{16}$$

$$\sum_{i=n}^{n} \quad y_i x_i^2 = \sum_{i=n}^{n} \quad ax_i^4 + \sum_{i=n}^{n} \quad bx_i^3 + \sum_{i=n}^{n} \quad cx_i^2 \tag{17}$$

Similarly we obtain the below equations after partial differentiation with respect to b, c

$$\sum_{i=n}^{n} \quad y_i x_i = \sum_{i=n}^{n} \quad ax_i^3 + \sum_{i=n}^{n} \quad bx_i^2 + \sum_{i=n}^{n} \quad cx_i \tag{18}$$

$$\sum_{i=n}^{n} \quad y_i = \sum_{i=n}^{n} \quad ax_i^2 + \sum_{i=n}^{n} \quad bx_i + \sum_{i=n}^{n} \quad cn \tag{19}$$

(Since sum of numbers from 1 to n is n and so we get cn as the last term)
We can hence write equations 17, 18 and 19 as matrices and solve them to find the unknown a, b and c values.

$$\begin{bmatrix} \sum_{i=n}^{n} x_i^4 & \sum_{i=n}^{n} x_i^3 & \sum_{i=n}^{n} x_i^2 \\ \sum_{i=n}^{n} x_i^3 & \sum_{i=n}^{n} x_i^2 & \sum_{i=n}^{n} x_i \\ \sum_{i=n}^{n} x_i^2 & \sum_{i=n}^{n} x_i & n \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum_{i=n}^{n} y_i x_i^2 \\ \sum_{i=n}^{n} y_i x_i \\ \sum_{i=n}^{n} y_i \end{bmatrix}$$

(Here n = 250)

It has come to form

$$AX = B \tag{20}$$

Now the unknowns can be calculated using

$$X = A^{-1}B \tag{21}$$

The curve seems to fit perfectly for the first data set. But in case of data set 2, the outliers are not accounted for and this is the reason this is not the most suitable method.

Similarly, while using LS with regularization we add an I matrix multiplied by a scalar value which also conditions the sensitivity of the matrix. We have multiplied the I matrix by 5 and 8 for data sets 1 and 2 respectively. We have represented the same as the R matrix. This still does not eliminate the outliers in case of data set 1.

In the Total Least Square Fitting method it also can be observed that the first data set already has a best fit curve since there are no outliers. The second dataset has a few outliers which are not taken care of by this method and it gives a curve that fits closest to the data.
The matrices $\mathrm{U}^T U = 0, \textit{can be derived as follows}$:

First the equation of a parabola is taken as follows−

$$d = ax^2 + bx + cy \tag{22}$$

We know that the error is nothing but the (actual − observed). This leads us to the following error equation:

$$E = \sum_{i=n}^{n} [ax^2 + bx + cy - d]^2 \tag{23}$$

Differentiating the above equation with respect to d and equating it to 0, we get an equation for d as follows,

$$d = \frac{a}{n} \sum_{i=n}^{n} [x^2] + \frac{b}{n} \sum_{i=n}^{n} [x] + \frac{c}{n} \sum_{i=n}^{n} [y] \tag{24}$$

where the mean of x and y values can be replaced with $\overline{x} and \overline{y}$

$$d = \frac{a}{n} \sum_{i=n}^{n} [x^2] + b\overline{x} + c\overline{y} \tag{25}$$

Substituting the equation of d back into eq.23, we get

$$E = \sum_{i=n}^{n} [a(x^2 - \frac{x^2}{n}) + b(x - \overline{x}) + c(y - \overline{y})]^2 \tag{26}$$

$$= \left\| \left(\begin{bmatrix} {x_1}^2 - \sum_{i=n}^{n} \frac{{x_1}^2}{n} & x_1 - \overline{x} & y_1 - \overline{y} \\ \cdot & \cdot & \\ \cdot & \cdot & \\ \cdot & \cdot & \\ {x_\mathrm{n}}^2 - \sum_{i=n}^{n} \frac{{x_\mathrm{n}}^2}{n} & x_\mathrm{n} - \overline{x} & y_1 - \overline{y} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \right) \right\|^2 \tag{27}$$

$$E = (UN)^T(UN) \tag{28}$$

Now to find an optimal solution we need to minimize a, b and c. This is done by partially differentiating the error equation by N whose elements are a,b and c.

$$\frac{\partial E}{\partial N} = 2(U^T U)N = 0 \tag{29}$$

We can solve the above equation by using SVD or we can use the

$$U^T U \tag{30}$$

matrix and then use Eigen value decomposition. We have chosen SVD and found out the solution for the N matrix.

This is done by finding the homography matrix of the U matrix, which is nothing but the N matrix which has a,b and c as its elements. The homography matrix of the Umatrix is the last column of the matrix formed by the orthogonal eigen vectors of transpose(U).U matrix. The curve seems to fit perfectly for the first data set. But in case of data set 2, the outliers are not accounted for and this is the reason this is not the most suitable method.

Clearly, RANSAC is the best for eliminating outliers as we have defined the way to classify a point as inlier and what needs to be taken as a meaningful data set value and what is considered an outlier. Noises get classified as outliers and does not affect the actual dataset. So, it is an efficient and effective method of filtering out gaussian noises that affect our dataset.

3 To compute SVD of a matrix A and to find its Homography matrix

$$A = \begin{pmatrix} -x1 & -y1 & -1 & 0 & 0 & 0 & x1*xp1 & y1*xp1 & xp1 \\ 0 & 0 & 0 & -x1 & -y1 & -1 & x1*yp1 & y1*yp1 & yp1 \\ -x2 & -y2 & -1 & 0 & 0 & 0 & x2*xp2 & y2*xp2 & xp2 \\ 0 & 0 & 0 & -x2 & -y2 & -1 & x2*yp2 & y2*yp2 & yp2 \\ -x3 & -y3 & -1 & 0 & 0 & 0 & x3*xp3 & y3*xp3 & xp3 \\ 0 & 0 & 0 & -x3 & -y3 & -1 & x3*yp3 & y3*yp3 & yp3 \\ -x4 & -y4 & -1 & 0 & 0 & 0 & x4*xp4 & y4*xp4 & xp4 \\ 0 & 0 & 0 & -x4 & -y4 & -1 & x4*yp4 & y4*yp4 & yp4 \end{pmatrix} \tag{31}$$

where,

index	x	y	xp	yp
1	5	5	100	100
2	150	5	200	80
3	150	150	220	80
4	5	150	100	200

Substituting the values, the A matrix becomes

$$A = \begin{bmatrix} -5 & -5 & -1 & 0 & 0 & 0 & 500 & 500 & 100 \\ 0 & 0 & 0 & -5 & -5 & -1 & 500 & 500 & 100 \\ -150 & -5 & -1 & 0 & 0 & 0 & 30000 & 1000 & 200 \\ 0 & 0 & 0 & -150 & -5 & -1 & 12000 & 400 & 80 \\ -150 & -150 & -1 & 0 & 0 & 0 & 33000 & 33000 & 220 \\ 0 & 0 & 0 & -150 & -150 & -1 & 12000 & 12000 & 80 \\ -5 & -150 & -1 & 0 & 0 & 0 & 500 & 15000 & 100 \\ 0 & 0 & 0 & -5 & -150 & -1 & 1000 & 30000 & 200 \end{bmatrix} \tag{32}$$

To compute SVD for Matrix A ,we need to calculate eigen values and vectors of $\mathrm{A}A^{\mathrm{T}}$ and $\mathrm{A} = \mathrm{U}\sum V^{T}$

The columns of U $(size : m \times m)$*are orthogonal eigenvectors of* $A\mathrm{A}^{\mathrm{T}}$

The columns of V (size:n×n)*are orthogonal eigenvectors of* $\mathrm{A}^{\mathrm{T}}\mathrm{A}$

Eigenvalues 1 ... λr *of* $A\mathrm{A}^{\mathrm{T}}\mathrm{V}$ are the eigenvalues of

$\sigma i = \sqrt{\lambda i}$

$\sum = diag(\sigma_1 ... \sigma_{\mathrm{r}})$

U Matrix

$$A = \begin{bmatrix} -5 & -5 & -1 & 0 & 0 & 0 & 500 & 500 & 100 \\ 0 & 0 & 0 & -5 & -5 & -1 & 500 & 500 & 100 \\ -150 & -5 & -1 & 0 & 0 & 0 & 30000 & 1000 & 200 \\ 0 & 0 & 0 & -150 & -5 & -1 & 12000 & 400 & 80 \\ -150 & -150 & -1 & 0 & 0 & 0 & 33000 & 33000 & 220 \\ 0 & 0 & 0 & -150 & -150 & -1 & 12000 & 12000 & 80 \\ -5 & -150 & -1 & 0 & 0 & 0 & 500 & 15000 & 100 \\ 0 & 0 & 0 & -5 & -150 & -1 & 1000 & 30000 & 200 \end{bmatrix} \quad (33)$$

$$A^T = \begin{bmatrix} -5 & 0 & -150 & 0 & -150 & 0 & -5 & 0 \\ -5 & 0 & -5 & 0 & -150 & 0 & -150 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 \\ 0 & -5 & 0 & -150 & 0 & -150 & 0 & -5 \\ 0 & -5 & 0 & -5 & 0 & -150 & 0 & -150 \\ 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 500 & 500 & 30000 & 12000 & 33000 & 12000 & 500 & 1000 \\ 500 & 500 & 1000 & 400 & 33000 & 12000 & 15000 & 30000 \\ 100 & 100 & 200 & 80 & 220 & 80 & 100 & 200 \end{bmatrix} \quad (34)$$

$$AA^T = \begin{bmatrix} 510051 & 510000 & 15520776 & 6208000 & 33023501 & 12008000 & 7760776 & 15520000 \\ 510000 & 510051 & 15520000 & 6208776 & 33022000 & 12009501 & 7760000 & 15520776 \\ 15520776 & 15520000 & 901062526 & 360416000 & 1023067251 & 372016000 & 30021501 & 60040000 \\ 6208000 & 6208776 & 360416000 & 144188926 & 409217600 & 148829651 & 12008000 & 24017501 \\ 33023501 & 33022000 & 1023067251 & 409217600 & 2178093401 & 792017600 & 511545251 & 1023044000 \\ 12008000 & 12009501 & 372016000 & 148829651 & 792017600 & 288051401 & 186008000 & 372039251 \\ 7760776 & 7760000 & 30021501 & 12008000 & 511545251 & 186008000 & 225282526 & 450520000 \\ 15520000 & 15520776 & 60040000 & 24017501 & 1023044000 & 372039251 & 450520000 & 901062526 \end{bmatrix} \quad (35)$$

The Eigen Values and the Eigen Vectors of the above 8x8 Matrix is given by,

$$EigenValues(AA^T) = \begin{bmatrix} 3625833630 & 1012800110 & 68065 & 34677.6193 & 21201.2335 & 3706.48899 & 0.656490959 & 15.2001454 \end{bmatrix} \quad (36)$$

$$U = EigenVectors(AA^T) = \begin{bmatrix} 0.0118 & 0.0003 & 0.0516 & -0.4661 & -0.2603 & -0.0678 & 0.0108 & -0.8411 \\ 0.0118 & 0.0003 & 0.0872 & -0.4594 & -0.2491 & -0.0886 & 0.7655 & 0.3542 \\ 0.3587 & 0.6549 & -0.0135 & -0.4651 & 0.1701 & 0.2936 & -0.2784 & 0.1823 \\ 0.1435 & 0.2620 & 0.4454 & 0.1361 & -0.5008 & -0.5875 & -0.2731 & 0.1529 \\ 0.7750 & 0.0227 & -0.4085 & 0.2849 & 0.0320 & -0.2352 & 0.2627 & -0.1597 \\ 0.2818 & 0.0082 & 0.6922 & 0.3159 & 0.0114 & 0.5019 & 0.2466 & -0.1696 \\ 0.1846 & -0.3168 & -0.2485 & -0.0347 & -0.6983 & 0.4673 & -0.2524 & 0.1816 \\ 0.3693 & -0.6336 & 0.2889 & -0.3933 & 0.3189 & -0.1750 & -0.2614 & 0.1526 \end{bmatrix} \tag{37}$$

V Matrix

$$A = \begin{bmatrix} -5 & -5 & -1 & 0 & 0 & 0 & 500 & 500 & 100 \\ 0 & 0 & 0 & -5 & -5 & -1 & 500 & 500 & 100 \\ -150 & -5 & -1 & 0 & 0 & 0 & 30000 & 1000 & 200 \\ 0 & 0 & 0 & -150 & -5 & -1 & 12000 & 400 & 80 \\ -150 & -150 & -1 & 0 & 0 & 0 & 33000 & 33000 & 220 \\ 0 & 0 & 0 & -150 & -150 & -1 & 12000 & 12000 & 80 \\ -5 & -150 & -1 & 0 & 0 & 0 & 500 & 15000 & 100 \\ 0 & 0 & 0 & -5 & -150 & -1 & 1000 & 30000 & 200 \end{bmatrix} \tag{38}$$

$$A^T = \begin{bmatrix} -5 & 0 & -150 & 0 & -150 & 0 & -5 & 0 \\ -5 & 0 & -5 & 0 & -150 & 0 & -150 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 \\ 0 & -5 & 0 & -150 & 0 & -150 & 0 & -5 \\ 0 & -5 & 0 & -5 & 0 & -150 & 0 & -150 \\ 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 500 & 500 & 30000 & 12000 & 33000 & 12000 & 500 & 1000 \\ 500 & 500 & 1000 & 400 & 33000 & 12000 & 15000 & 30000 \\ 100 & 100 & 200 & 80 & 220 & 80 & 100 & 200 \end{bmatrix} \tag{39}$$

$$A^T A = \begin{bmatrix} 45050 & 24025 & 310 & 0 & 0 & 0 & -9455000 & -5177500 & -64000 \\ 24025 & 45050 & 310 & 0 & 0 & 0 & -5177500 & -7207500 & -49500 \\ 310 & 310 & 4 & 0 & 0 & 0 & -64000 & -49500 & -620 \\ 0 & 0 & 0 & 45050 & 24025 & 310 & -3607500 & -2012500 & -25500 \\ 0 & 0 & 0 & 24025 & 45050 & 310 & -2012500 & -6304500 & -42900 \\ 0 & 0 & 0 & 310 & 310 & 4 & -25500 & -42900 & -460 \\ -9455000 & -5177500 & -64000 & -3607500 & -2012500 & -25500 & 2278750000 & 1305800000 & 15530000 \\ -5177500 & -7207500 & -49500 & -2012500 & -6304500 & -42900 & 1305800000 & 2359660000 & 16052000 \\ -64000 & -49500 & -620 & -25500 & -4290 & -460 & 15530000 & 16052000 & 171200 \end{bmatrix} \tag{40}$$

The Eigen values and Eigen vectors of the above 9x9 Matrix is given by,

$$EigenValues(A^TA) = \begin{bmatrix} 3625833630 & 1012800110 & 68065.1927 & 34677.6193 & 21201.2335 \\ 3706.48899 & 15.2001454 & 0.656490976 & -4.76010921e-10 & \end{bmatrix} \tag{41}$$

$$V = Eigenvectors(A^TA) = \begin{bmatrix} 0.0028 & 0.0031 & -0.2464 & -0.1586 & 0.1752 & 0.1767 & -0.9137 & 0.1203 & 0.0531 \\ 0.0024 & -0.0013 & -0.3770 & 0.1766 & -0.6895 & 0.5903 & 0.0529 & 0.0022 & -0.0049 \\ 0.0000 & 0.0000 & -0.0024 & -0.0037 & -0.0052 & 0.0075 & -0.0660 & -0.7860 & 0.6146 \\ 0.0011 & 0.0012 & 0.6612 & 0.3412 & -0.5017 & -0.2325 & -0.3721 & 0.0426 & 0.0177 \\ 0.0016 & -0.0029 & 0.5743 & -0.0710 & 0.3145 & 0.7499 & 0.0620 & -0.0046 & -0.0039 \\ 0.0000 & -0.0000 & 0.0058 & -0.0022 & -0.0029 & -0.0057 & 0.1225 & 0.6049 & 0.7868 \\ -0.6961 & -0.7180 & -0.0001 & -0.0038 & -0.0025 & -0.0002 & -0.0044 & 0.0006 & 0.0002 \\ -0.7180 & 0.6961 & 0.0016 & -0.0038 & -0.0025 & 0.0037 & 0.0006 & -0.0000 & -0.0000 \\ -0.0062 & 0.0000 & -0.1735 & 0.9067 & 0.3783 & 0.0622 & -0.0252 & 0.0025 & 0.0076 \end{bmatrix} \tag{42}$$

Sigma Matrix

The Eigen values of $AA^T = The Eigenvalues of A^TA. Therefore, lets\ consider\ the\ Eigenvalues\ of\ AA^T.$

$$EigenValues(AA^T) = \begin{bmatrix} 3625833630 & 1012800110 & 68065 & 34677.6193 & 21201.2335 & 3706.48899 & 0.656490959 & 15.200145 \end{bmatrix} \tag{43}$$

we know that, $\sigma_i = \sqrt{\lambda_i}$

Sorting the Eigen values in Descending order and finding the square roots, we get

$$\sqrt{\lambda} = \begin{bmatrix} 60214.9 & 31824.52 & 260.9 & 186.22 & 145.61 & 60.89 & 3.898 & 0.8102 \end{bmatrix} \tag{44}$$

Creating a Diagonal matrix using these values, we get a 8x8 Matrix as follows,

$$\sqrt{\lambda} = \begin{bmatrix} 60214.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 31824.52 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 260.9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 186.22 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 145.61 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 60.89 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3.898 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8102 \end{bmatrix} \tag{45}$$

The Dimensions of the $\sigma matrix must be (mxn)$.
Therefore we append a column of zeroes to the above matrix to get (8x9) mxn dimensions.

$$\sigma = \begin{bmatrix} 60214.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 31824.52 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 260.9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 186.22 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 145.61 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 60.89 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3.898 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8102 & 0 \end{bmatrix} \tag{46}$$

SVD

Now that we have the U, V and $\sigma Matrices, we\ can\ do\ the\ singular\ value\ decomposition$
A = U$\sum V^T$

$$U = \begin{bmatrix} 0.0118 & 0.0003 & 0.0516 & -0.4661 & -0.2603 & -0.0678 & 0.0108 & -0.8411 \\ 0.0118 & 0.0003 & 0.0872 & -0.4594 & -0.2491 & -0.0886 & 0.7655 & 0.3542 \\ 0.3587 & 0.6549 & -0.0135 & -0.4651 & 0.1701 & 0.2936 & -0.2784 & 0.1823 \\ 0.1435 & 0.2620 & 0.4454 & 0.1361 & -0.5008 & -0.5875 & -0.2731 & 0.1529 \\ 0.7750 & 0.0227 & -0.4085 & 0.2849 & 0.0320 & -0.2352 & 0.2627 & -0.1597 \\ 0.2818 & 0.0082 & 0.6922 & 0.3159 & 0.0114 & 0.5019 & 0.2466 & -0.1696 \\ 0.1846 & -0.3168 & -0.2485 & -0.0347 & -0.6983 & 0.4673 & -0.2524 & 0.1816 \\ 0.3693 & -0.6336 & 0.2889 & -0.3933 & 0.3189 & -0.1750 & -0.2614 & 0.1526 \end{bmatrix} \tag{47}$$

$$\sigma = \begin{bmatrix} 60214.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 31824.52 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 260.9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 186.22 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 145.61 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 60.89 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3.898 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8102 & 0 \end{bmatrix} \tag{48}$$

V =

-0.0028	-0.0031	0.2464	0.1586	-0.1752	-0.1767	0.9137	-0.1203	-0.0531
-0.0024	0.0013	0.3770	-0.1766	0.6895	-0.5903	-0.0529	-0.0022	0.0049
0	0	0.0024	0.0037	0.0052	-0.0075	0.0660	0.7860	-0.6146
-0.0011	-0.0012	-0.6612	-0.3412	0.5017	0.2325	0.3721	-0.0426	-0.0177
-0.0016	0.0029	-0.5743	0.0710	-0.3145	-0.7499	-0.0620	0.0046	0.0039
0	0	-0.0058	0.0022	0.0029	0.0057	-0.1225	-0.6049	-0.7868
0.6961	0.7180	0.0001	0.0038	0.0025	0.0002	0.0044	-0.0006	-0.0002
0.7180	-0.6961	-0.0016	0.0038	0.0025	-0.0037	-0.0006	0	0
0.0062	0	0.1735	-0.9067	-0.3783	-0.0622	0.0252	-0.0025	-0.0076

Homography Matrix

Multiplying U matrix, Sigma Matrix and the Transpose of V matrix, we get

```
A =

   1.0e+04 *

   -0.0005 | -0.0005   -0.0001         0         0         0    0.0501    0.0503    0.0100
         0          0         0   -0.0005   -0.0005   -0.0001    0.0501    0.0503    0.0100
   -0.0148   -0.0004   -0.0000   -0.0001    0.0001    0.0000    2.9999    0.1000    0.0201
    0.0001    0.0000    0.0000   -0.0150   -0.0005   -0.0001    1.2002    0.0400    0.0081
   -0.0148   -0.0149    0.0000   -0.0000    0.0002    0.0001    3.3003    3.3004    0.0222
    0.0001    0.0000    0.0000   -0.0150   -0.0149   -0.0001    1.2000    1.2002    0.0081
   -0.0005   -0.0150   -0.0001    0.0000    0.0001    0.0000    0.0498    1.4999    0.0100
         0    0.0000    0.0000   -0.0005   -0.0149   -0.0001    0.1002    3.0002    0.0200
```

Homography Matrix

The homography matrix is nothing but the last column of the V matrix, formed by the orthogonal Eigen vectors of the $AA^T matrix$.

$$H = \begin{bmatrix} 5.310e-02 & -4.900e-03 & 6.146e-01 \\ 1.770e-02 & -3.900e-03 & 7.868e-01 \\ 2.000e-04 & -0.000e+00 & 7.600e-03 \end{bmatrix} \tag{49}$$

where, H is the Homography matrix.
we know that, A.X = 0
we can check that by multiplying the A matrix with the Homogeneous matrix t verify the same.

$$A.X = \begin{bmatrix} 0.0044 & 0.0042 & -1.0351 & -0.4143 & 0.4274 & 0.1512 & 0.7149 & 1.4297 \end{bmatrix} \tag{50}$$

The values of A.X Matrix are approximately equal to zero.