ENPM673 Project 1

Ankur Mahesh Chavan (achavan1@umd.edu)

Problem 1.1:

In the given video, a red ball is thrown against a wall. Assuming that the trajectory of the ball follows the equation of a parabola: Detect and plot the pixel coordinates of the center point of the ball in the video.

(Hint: Read the video using OpenCV's inbuilt function. For each frame, filter the red channel)

Solution:

Pipeline or approach:

1. Imported the necessary libraries: OpenCV (cv2), NumPy, Matplotlib, and Seaborn.
2. Read a video named 'ball.mov' using the OpenCV's VideoCapture() method and stored it in the variable cap.
3. Defined empty NumPy arrays to store the x and y coordinates of the centroid of the red ball.
4. Defined variables to store the size of the video frame.
5. Used a while loop to loop through each frame of the video and perform the following operations:

a. Read the frame using the cap.read() method and store the returned values in the variables ret and frame.

b. Check if the video has ended, if it has, break out of the loop. Stored the shape of video frame using shape function.

c. Convert the BGR color space of the frame to the HSV color space using cv2.cvtColor() method.

d. Threshold the frame in the red range using cv2.inRange() method and store the returned binary mask in the variable mask.

e. Create a Numpy array of all the red pixels in the frame by finding the indices of the pixels that have a value of 255 in the mask using np.argwhere() method and store it in the variable red_pixels.

f. Calculate the centroid of the red ball using the mean of the x and y coordinates of the red pixels using np.mean() method. Then, append the calculated x and y coordinates to the x_vals and y_vals arrays, respectively. Finally, draw a green circle around the centroid using cv2.circle() method.

g. Display the original frame using cv2.imshow() method.

h. Check if the 'q' key is pressed, if it is, break out of the loop.

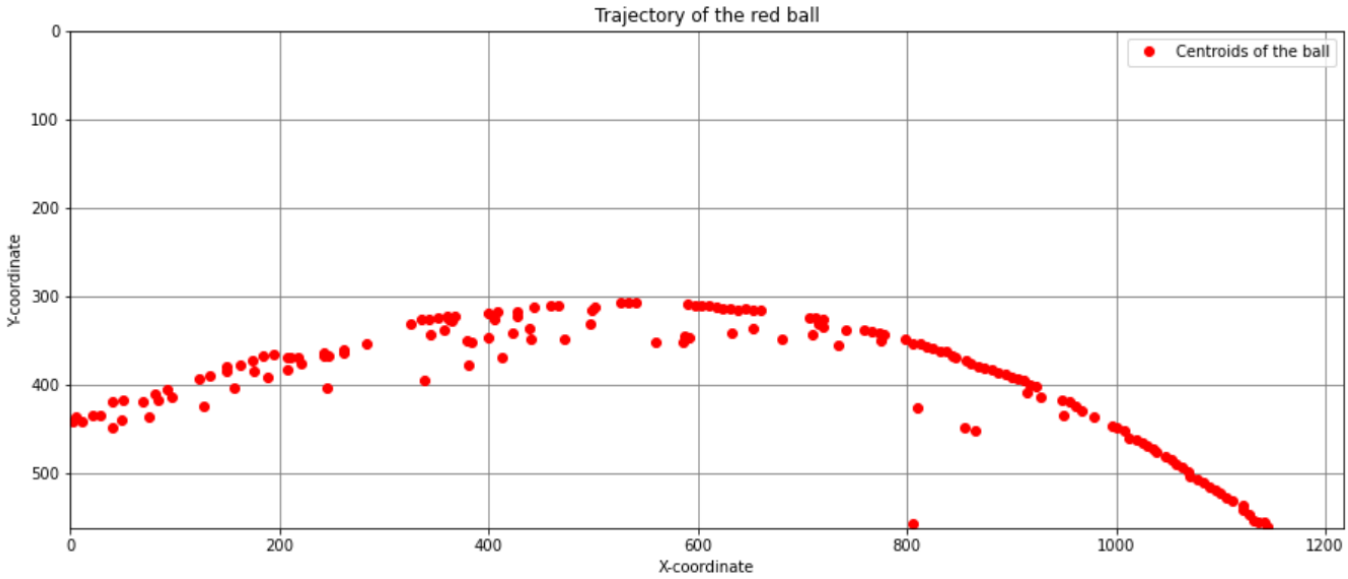
6. Then released the video and destroy all windows opened using cap.release() and cv2.destroyAllWindows() methods, respectively.
7. Set the figure size of the plot using plt.figure() method. Set the axis limits to match the size of the video frame using plt.xlim() and plt.ylim() functions.
8. Plotted the trajectory of the ball and the fitted curve using plt.plot() method. In this case, we plot the x_vals against y_vals. Displayed the plot using plt.show() method.

Problems Encountered:

1. I am new to Opencv, so first not knowing the syntax of how to read a video was my problem. I browsed through Google and stack overflow to get used to basic Opencv function regarding video reading.

2. Initially I didn't convert the image from RGB to HSV so I was not able to detect the red pixels properly. And even after converting to HSV finding the proper threshold for mask was a try and error problem. I was able to find an appropriate threshold to get rid of maximum noise.

Results of Problem 1.1:



Code: Submitted in another file

Problem 1.2:

Use Standard Least Squares to fit a curve to the extracted coordinates. For the estimated parabola you must,

a. Print the equation of the curve.

b. Plot the data with your best fit curve.

After collecting the x and y coordinates of the centroid for each frame in the video, used least square method to fit a curve of the centroid points.

Pipeline or approach:

- Defined the degree of the polynomial as 2.
- Created Numpy arrays for the x and y coordinates of the centroid.
- Constructed the matrix A and the vector b for solving the Normal Equations.
- Looped over each row in A and set its values based on the corresponding x coordinate of the centroid.
- Reshaped the y_vals array to match the dimensions of b.
- Solved the Normal Equations by computing ATA and ATb, and using them to compute the coefficients of the polynomial curve. Printed the equation of the fitted curve.
- Evaluated the polynomial at a range of x values using np.linspace() to generate 100 equally spaced values between the minimum and maximum x coordinates of the centroid.
- Created an array of zeros to store the y coordinates of the fitted curve and compute these values by looping over each degree of the polynomial and adding the corresponding term to y_range.
- Plotted the centroids of the ball as dots, and the fitted curve on the same plot.

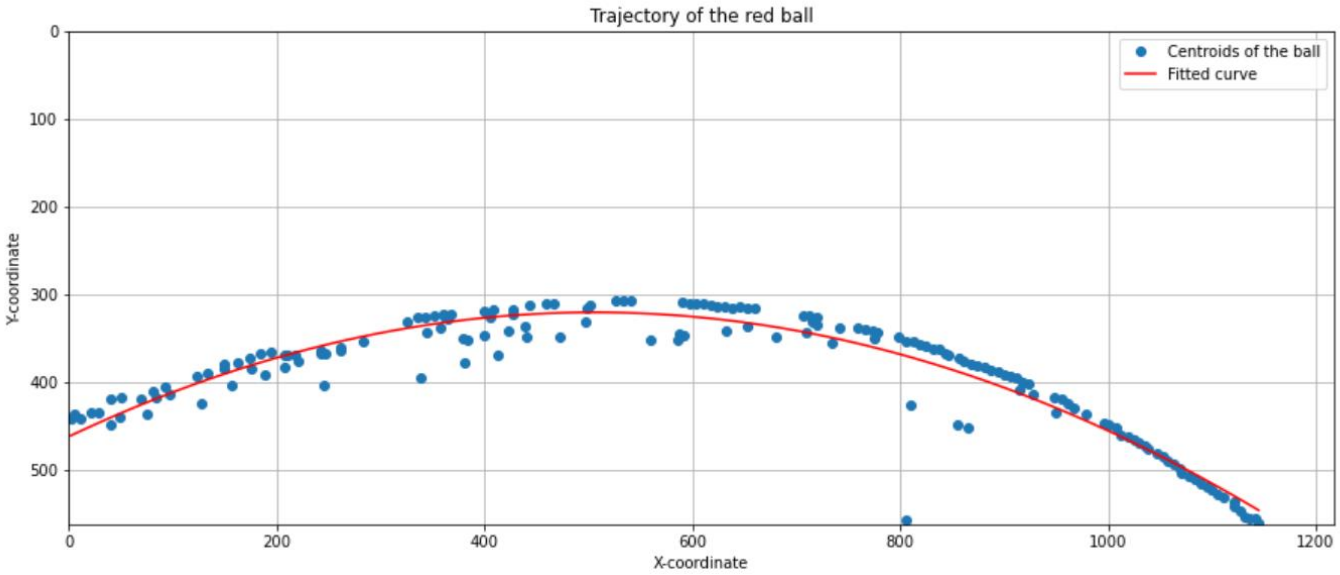
Problems encountered:

After attending the lecture the math behind least square method was pretty straight forward, so didn't encountered much problems.

Results:

Displayed the equation and plotted the fitted curve.

```
Equation of the fitted curve: y = [0.00055211]x^2 + [-0.55881395]x + [462.14774512]
```



Code: Submitted in another file

Problem 1.3:

Assuming that the origin of the video is at the top-left of the frame as shown below, compute the x-coordinate of the ball's landing spot in pixels, if the y-coordinate of the landing spot is defined as 300 pixels greater than its first detected location.

Pipeline or approach:

The x-coordinate of the ball when first detected would be the min of x_vals, in our case the min of x_vals is 2, and the y value corresponding to this x value is 442. As per the question the new y-coordinate will be 442+300 = 742. Now we need to find the corresponding x-coordinate which will be our landing spot. We find the x-coordinate by putting y-coordinate in the obtained parabolic equation "y = ([0.00055211])x^2 + ([-0.55881395])x + ([462.14774512])"

Problems encountered: No problems encountered.

Results:

```
Ball first detected at x = 2
Index of the first detected value: 1
Corresponding y coordinate: 442
x1 = 1379.5617824710619
x2 = -367.41936519914134
x-coordinate of desired landing spot is: 1379.5617824710619
```

Code: Submitted in another file

Problem 2.1:

Given are two csv files, pc1.csv and pc2.csv, which contain noisy LIDAR point cloud data in the form of (x, y, z) coordinates of the ground plane. 1. Using pc1.csv:

a. Compute the covariance matrix.

b. Assuming that the ground plane is flat, use the covariance matrix to compute the magnitude and direction of the surface normal.

2.1.a

Pipeline or approach:

1. Imported the necessary libraries, read the CSV file containing the data, and assigned column names to the data.
2. Checked if there are any missing values in the data.
3. Extracted the three columns of the data into three separate NumPy arrays.
4. Computed the mean of each column by using the NumPy function **mean()**.
5. The covariances between each pair of columns are computed using the formula for covariance. A covariance matrix is computed as a 3x3 NumPy array. The math to calculate the covariance matrix was explained in the lecture, so I referred to the class ppts for this. Finally printed the desired matrix.

Problems encountered:

Initially I faced some problems calculating the covariance matrix even though I knew the formula, I was struggling with the code. After checking through various resources I was able to compute the matrix.

Results:

```
Covariance Matrix:
[[ 33.7500586   -0.82513692 -11.39434956]
 [ -0.82513692  35.19218154 -23.23572298]
 [-11.39434956 -23.23572298  20.62765365]]
```

Code: Submitted in another file

2.1.b

Pipeline or approach:

1. Extracted the three columns from the dataframe df and assigned them to x, y, and z variables.
2. Calculated the eigenvalues and eigenvectors of the covariance matrix using NumPy's linalg.eig function.
3. The eigenvectors represent the directions of maximum variance in the dataset, and the corresponding eigenvalues represent the amount of variance along those directions. Identified the eigenvector corresponding to the minimum eigenvalue, as this is the direction of minimum variance or the surface normal.
4. Printed the min_eigenvector which is surface normal vector. Calculated the direction using tan inverse.

Problems encountered: No problems encountered in this section

Results:

```
Surface Normal: [0.28616428 0.53971234 0.79172003]
Tan inverse: 1.0832689010438281
```

Code: Submitted in another file

Problem 2.2.a:

In this question, you will be required to implement various estimation algorithms such as Standard Least Squares, Total Least Squares and RANSAC.

a. Using pc1.csv and pc2, fit a surface to the data using the standard least square method and the total least square method. Plot the results (the surface) for each method and explain your interpretation of the results.

Pipeline or approach: For standard least square method for pc1.csv

The code performs a standard least square method on a dataset 'pc1.csv'. The ultimate goal is to fit a plane to the dataset and plot it in a 3D scatter plot. Here are the detailed steps:

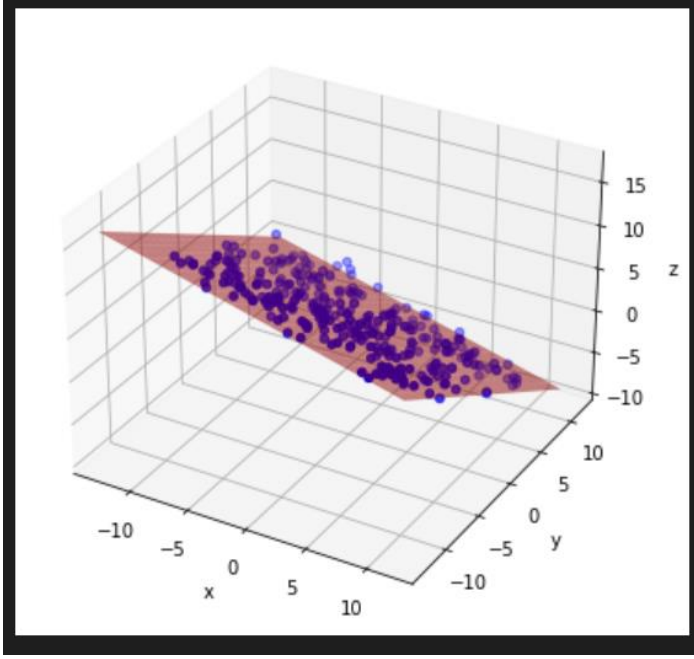
1. First, the code reads the CSV file 'pc1.csv' using the Pandas function **pd.read_csv**. The file contains three columns of data, which are assigned to **xs**, **ys**, and **zs** variables.
2. Then I initialized empty lists **tmp_A** and **tmp_b**. For each row in the dataset, the code appends a list containing the values of **xs**, **ys**, and a constant **1** to **tmp_A**, and appends the value of **zs** to **tmp_b**.
3. The lists **tmp_A** and **tmp_b** are converted to NumPy matrices **A** and **b**, respectively.
4. Then calculated the coefficients of the plane using the standard least squares method. The coefficients are calculated as **fit = (A.T * A).I * A.T * b**, where **A.T** is the transpose of matrix **A**, **A.I** is the inverse of matrix **A**, ***** denotes matrix multiplication, and **fit** contains the coefficients of the plane.
5. Printed the solution as an equation.
6. Then set the limits of the x and y axes of the 3D scatter plot using the **get_xlim** and **get_ylim** functions.
7. Calculated the z values of the plane using the coefficients of the plane and the x and y values of the mesh grid.
8. The code creates a 3D scatter plot using Matplotlib's **plt.subplot** function with the argument **projection='3d'**. Then plotted the cloud data and the fitted plane in the same figure.

Problems encountered:

I first faced problems while implementing the least square method for the parabola. When I figured that problem I was able to fit a surface to the given cloud data without much problems other than few syntax errors I knew what I had to do. Syntax errors were resolved by checking through various resources like stack overflow.

Results:

```
solution:
-0.353955 x + -0.668551 y + 3.202554 = z
```



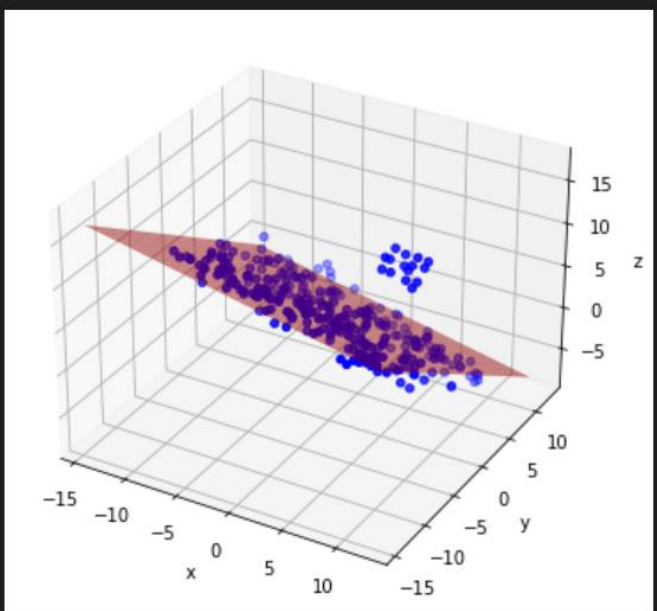
Code: Submitted in another file

Problem 2.2.a: For standard least square method for pc2.csv

The same approach and code were used to fit the curve for pc2.csv data.

Results:

```
solution:
-0.251884 x + -0.671737 y + 3.660257 = z
```



Interpretation of least square method for pc1.csv and pc2.csv:

As we can see some outliers in the pc2.csv data, the equation of the plane will change as the least square method gets affected due to outliers. From the printed equations for both the data files we can there is a slight difference in the coefficients of the equation, this is caused due to the outliers. So the plane for data2 is slightly different from data1 due to outliers. To better treat the outliers we try Total least square method which calculates the orthogonal distances.

Problem 2.2.a: For total least square method for pc1.csv

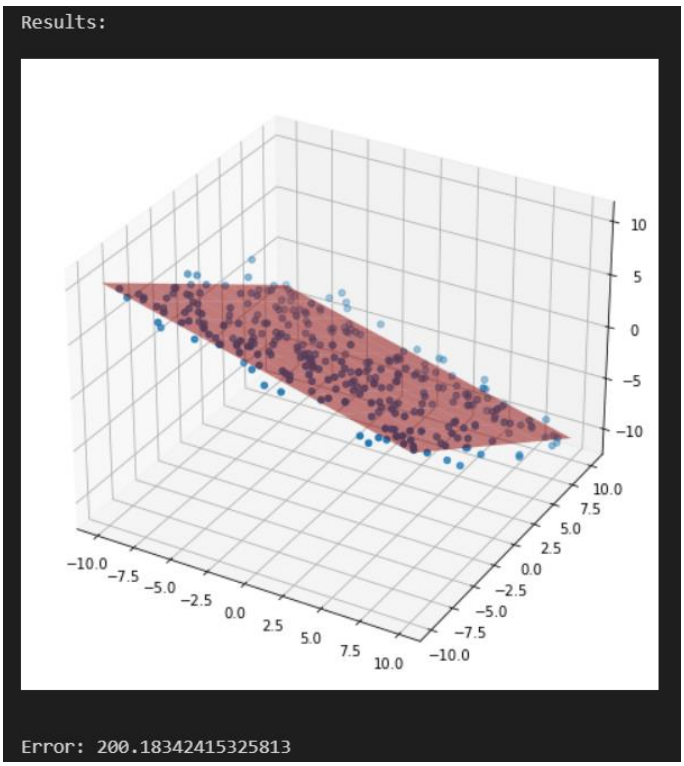
Pipeline or approach: For Total least square method for pc1.csv

1. Imported the required libraries - NumPy, Pandas, Matplotlib, and Seaborn.
2. Read the point cloud data from a CSV file into a Pandas DataFrame.
3. Extracted the x, y, and z coordinates from the DataFrame into separate NumPy arrays.
4. Calculated the centroid of the x, y, and z coordinates.
5. Shifted the data to the centroid by subtracting the x, y, and z coordinates by the centroid.
6. Calculated the matrix A from the shifted data, where each row is a point in 3D space.
7. Computed the eigenvalues and eigenvectors of the product A transpose and A.
8. Found the eigenvector corresponding to the smallest eigenvalue, which represents the normal vector of the fitted plane.
9. Calculated the coefficients of the plane equation by extracting the components of the eigenvector corresponding to the smallest eigenvalue. These components represent the coefficients of x, y, and z in the plane equation.
10. Calculate the distance of the plane from the origin (d)
11. Plotted the original point cloud data and the fitted plane in 3D using Matplotlib.
12. Calculated the mean squared error (MSE) by computing the sum of the squared errors between the predicted z-coordinates and the actual z-coordinates of the point cloud data and printed the results.

Problems encountered:

First to understand the TLS I had to read many articles and papers. After that I had to find an easy way to implement TLS without using any functions. Once I got the approach, implementation was easy. Only problems were syntax related.

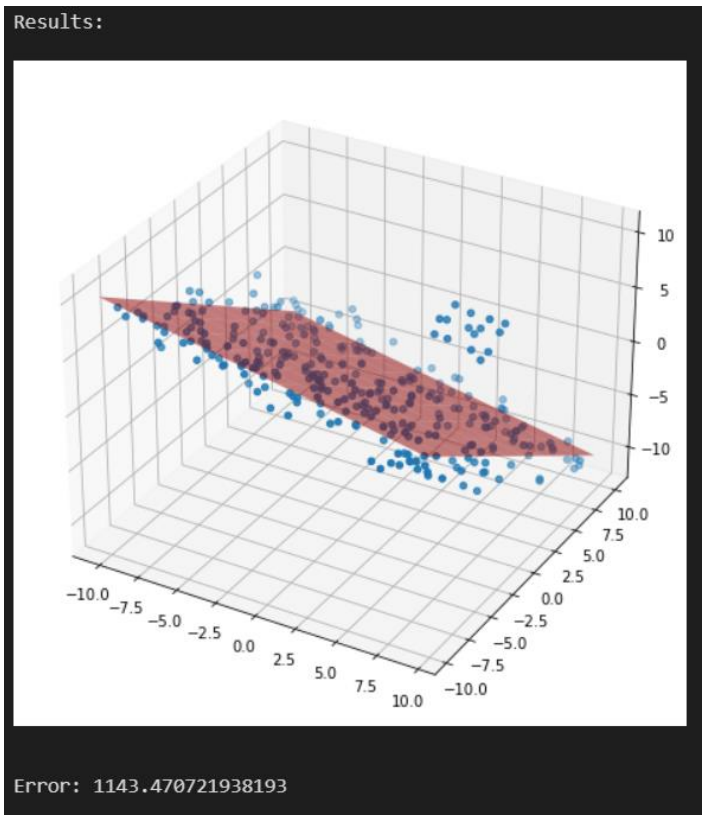
Results:



Problem 2.2.a: For total least square method for pc2.csv

The same approach and code were used to fit the curve for pc2.csv data.

Results:



Code: Submitted in another file

Interpretation of Total square method for pc1.csv and pc2.csv:

As we know there are outliers in pc2.csv, that causes the plane to shift slightly. As compared the pc1.csv the error is more in pc2.csv due to outliers.

Problem 2.2.b:

Additionally, fit a surface to the data using RANSAC. You will need to write RANSAC code from scratch. Briefly explain all the steps of your solution, and the parameters used. Plot the output surface on the same graph as the data. Discuss which graph fitting method would be a better choice of outlier rejection.

RANSAC for pc1.csv:

Approach for RANSAC:

1. Imported the necessary libraries: numpy, pandas, matplotlib.
2. Read the csv file and assigned the names to columns.
3. Extracted the columns A, B, and C from the dataframe.
4. Converted the columns to arrays using the numpy array function.
5. Defined the parameters for RANSAC method: number of iterations, threshold distance, and the minimum number of inliers required to be considered as the best model.
6. Set the initial best inliers to -1.
7. Looped through the number of iterations defined and did following operations in each loop:

   a. Randomly select three points from the X, Y, and Z arrays.

   b. Create a matrix A and a matrix B from the equation of the plane using the selected points.

   c. Solve the equation of the plane to get the coefficients.

   d. Calculate the distance of all points from the plane using the coefficients.

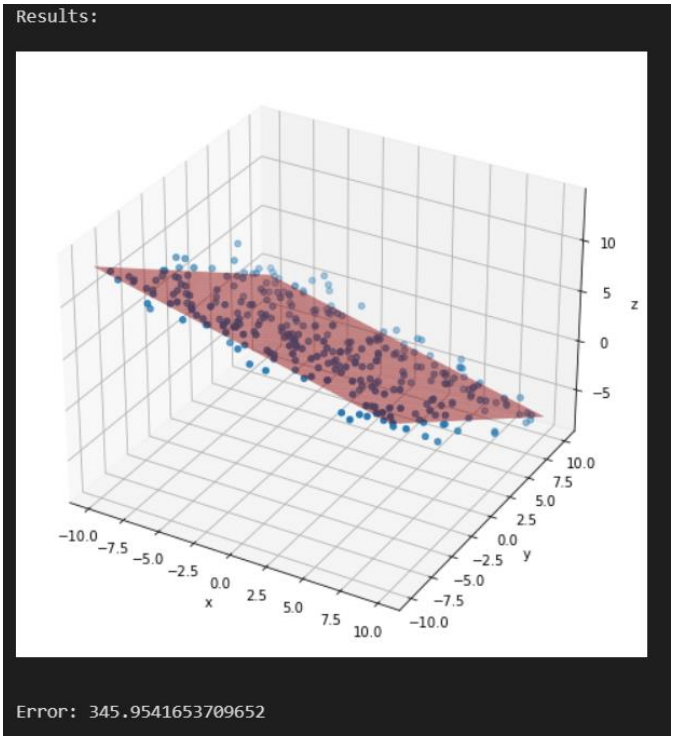
   e. Count the number of inliers that are within the threshold distance.

   f. If the number of inliers is greater than the previous best model, update the best model's inliers & coefficients.
8. Then extracted the coefficients of the best model.
9. Plotted the original data points and the fitted plane using matplotlib's 3D scatter and surface plots.
10. Calculated the mean squared error of the fitted plane.

Problems encountered:

Initially I had to read various articles and papers to understand RANSAC. After understanding I had some problems while writing the code for the loop. I resolved the problem by referring to different resources. Like other problems I faced some syntax issues here also which were also resolved.

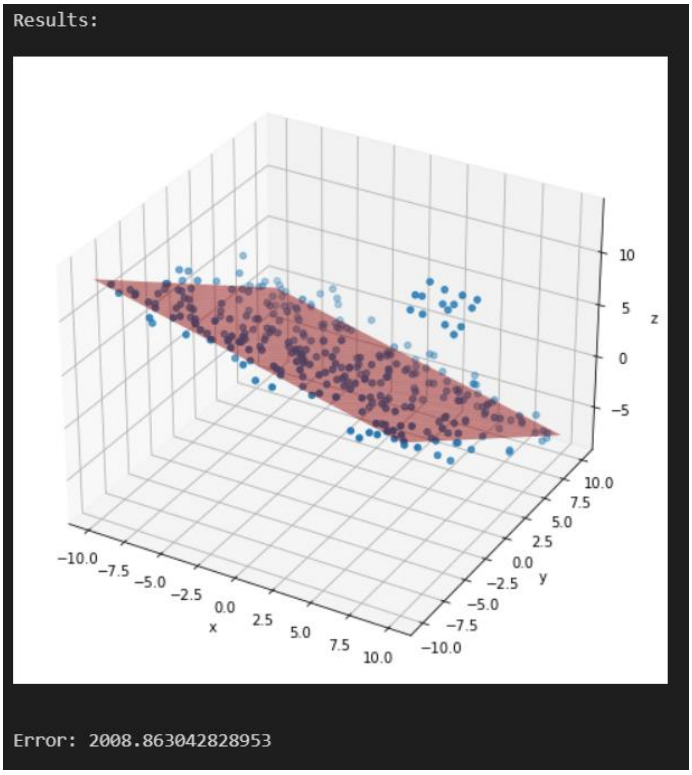
Results:



RANSAC for pc2.csv:

Same approach and code was used for pc2.csv

Results:



Code submitted in separate file.

Interpretation of results:

Like other estimation methods pc2.csv had more errors due to outliers. RANSAC has more errors as compared to total least square method because it is a random process, it will only get the best model within its iterations. Even after increasing the number of iterations, the error remains large, and it also takes more computational energy as compared to TLS. RANSAC can give the best results but its random, and it's not necessary that it will give overall best results, it will only give best result within its iterations.

Conclusion:

From the errors we can conclude that Total Least square gives the overall best result considering all the data points. TLS can be applied in case when all variables are independent.

RANSAC gives the best results within its iterations, it takes more computational energy and gives diff error each time due to its random nature.

Least square is a simple estimation method used when we have an independent variable and dependent variable.