Curve Fitting with Least Squares,Total Least Squares and RANSAC : ENPM673

Rahul Karanam
Robotics Graduate Student
University of Maryland, College Park
College Park, MD
rkaranam@umd.edu

I. Introduction

We will be looking for the best fit curves and lines to many datasets in this report. The first section discusses how to calculate Field of View and the image sensor's minimum pixel count. Later sections will go through how to use Least Squares, Total Least Squares, and RANSAC to fit parabola and line equations. The final section describes Singular Value Decomposition and how to use it to find a homography matrix given four points.

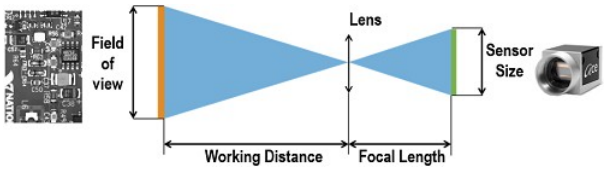


Fig. 1: Focal Length , Image and sensor size

II. Image sensor Basics

Finding the FOV :

We'll show you how to find the field of view (FOV) in both vertical and horizontal directions in this section.

Assume you have a camera with a resolution of **5MP** and a sensor width of **14mm**.

The camera's focal length is **25mm**.

$$\text{FOV} = 2 \times \arctan\left(\frac{\text{sensor size}}{2f}\right)$$

$$\text{FOV} = 2 \times \arctan\left(\frac{14}{50}\right)$$

$$\text{FOV} = 31.28^\circ \text{ (0.54 rad)}$$

where **f** is the focal length and textbfs is the sensor size, both in the same length unit, and FOV is measured in radians.

Finding the min_pixels for a object in the Image:

Given the object size and distance from the camera, we can now calculate the number of pixels in an image.

$$\text{numberofpixels} = \text{image area} \times \left(\frac{\text{camera resolution}}{sensorarea}\right)$$

First we find the image size using the below formulae:

$$\text{imagesize} = \text{focal length} \times \left(\frac{\text{object size}}{objectdistance}\right)$$

where the object distance is 20m, the object size is 5cm, and the image resolution is 5MP Mega Pixel

Now we need to identify the image and sensor area that will be used to find the smallest number of pixels.

$\text{sensorarea} = \text{sensor width} \times \text{sensor width}$

sensor area = $14 \times 14 = 196\text{mm}^2$

$\text{imagearea} = \text{image size} \times \text{image size}$

sensor area = $0.0625 \times 0.0625 = 0.003906\text{mm}^2$

$$\text{numberofpixels} = 0.003905 \times \left(\frac{5 \times 10^6}{196}\right)$$

number of pixels = 99.64 Pixels

III. Fitting a curve to a given video

We've provided two videos (ball video 1 and ball video 2) of a red ball being tracked with two separate sensors.The first video was captured by a near-perfect sensor, whereas the second was captured by a malfunctioning one.

To fit a parabola to these videos, I used opencv to obtain the coordinates of the red ball, that is, the ball's center points, and then utilized these coordinates to fit the curve against it.

In order to return frames, I utilized OpenCV to read the video.The cvtColor function is then used to convert each frame to a grayscale image. Grayscale is simple to work with because it only has one channel.Then I established a threshold for all the values between 230 and 255, which are white values.I used the threshold approach to set the value to 0.

We usually apply a threshold to these frames so that getting the colored pixel values of an item is simple.We use the findContours function to extract the contours in the image after applying the threshold.The boundaries of an item in the image are returned by this function. To display the contours, we take the result from findContours and feed it into drawContours.

As per our given image we can find two contours, ball and the boundary. To get the center position of the ball, we use minEnclosingCircle method which takes the contours as the input.

Fig. 2: Contour output

The ouput of the minEnclosingCircle is my center points of the red ball in all the frames.

Now we save all the coordinates into two arrays which we use to find the best fit using Standard Least squares.

The issue I faced was with the axis of the plot as the coordinates are calculated from the top left corner, so I had to invert the y-axis to get the correct trajectory.

Equation of parabola$-ax^2 + bx + c.$

To find the least squares solution we need to find the below parameter.

$\beta = (X^T X)^{-1} X^T y$

The below given figures represent the output of the best fit curve showing the trajectory of red ball in the both the videos.

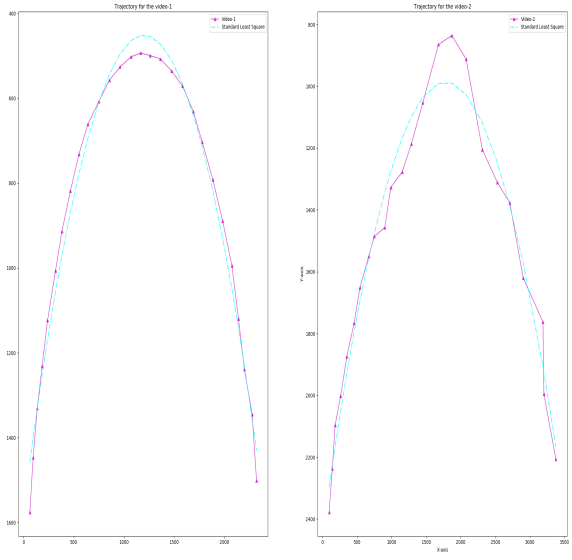


Fig. 3: Curve fitting for video-1 & video-2

Covariance Matrix:

We have two variables in our dataset: age and insurance premiums. To demonstrate how these variables are associated, we first determine the covariance matrix for the provided dataset.

The formulas below are used to find the covariance matrix.

Covariance Formula (Sample mean)

$Cov(x, y) = \sum_0^{n-1} \frac{(x-\bar{x})(y-\bar{y})}{n-1}$

Where:

X_i : the values of the X variable

Y_j : the values of the Y variable

$\bar{X}$: the mean (average) of the X variable

$\bar{Y}$: the mean (average) of the Y variable

n :the number of data p$\beta = (X^T X)^{-1} X^T y$oints

To get the covariance matrix, I used two functions: one to calculate the covariance of two variables, x and y, and another to build a matrix using the previous function.

The dataset file is an excel spreadsheet that I read and converted into two numpy arrays using pandas.

To display the data's covariance, we must construct the covariance matrix's eigen vectors using the numpy.linalg.eig method, which returns eigen values and eigen vectors. The cov matrix is (2,2). The sample data and eigen vectors are then plotted using matplotlib to highlight the variety of data using eigen vectors.

Please find below the plot of sample data points and eigen vectors associated with the covarinace matrix.

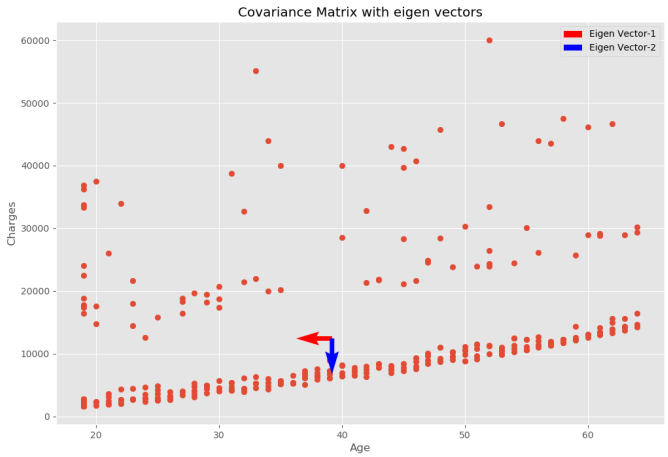


Fig. 4: Covariance Matrixc of age vs Insurance Charges

IV. Curve Fitting using LS,TLS and RANSAC

Now we'll try to fit the given data using three different methods: Least Squares(LS), Total Least Squares(TLS), and Random Sample Consensus(RANSAC), and then analyze the results.

Least Squares: To fit a line or a curve with standard least squares, we start by calculating the coefficients using the difference between dataset points and dataset's

mean.Essentially, least squares finds the best fit to the system of equations $AX = B$, where it determines the projection or best estimate of the variable b and the solution to the variable x.

To get the coefficients from the dataset, we apply the formula below. We utilize these coefficients to create a new line based on the dataset points, which will be our best fit line.We use vertical distance, or distance along the y-axis, to calculate the distance between new points.

$$\beta = (X^T X)^{-1} X^T y$$

Total Least Squares: The main difference between least square and total least square is that LS only updates the dependent variable, y, whereas TLS updates both the x and y variables because they are both susceptible to noise. We start by making a U matrix, which is made up of the difference between the mean values of both variables, x and y.

Then we look for the $U^T.U$ matrix to find the eigen vectors and eigen values, using the np.linalg.eig function.The total least squares solution is found in the vector that has the smallest eigen value, i.e. the last vector.We usually look for the orthogonal or perpendicular distance here.

$$U = \left[(X_i - \bar{X}) * (Y_i - \bar{Y})\right]$$

In order to plot the TLS best fit line , we first find a matrix d which is equal to $d = a * \bar{x} + b$ equation. After finding the d we then subtract with the new coefficients to get the new y values in order to plot.

$y_{new} = \frac{A\bar{x}+B-Ax}{B}$

where A,B are the coefficients returned by the eigen vector constituted by the smallest eigen value.

Random Sample Consensus: The following methods can provide an ideal solution, but if the data contains outliers, finding the best fit line will be difficult since the best fit line will be prone to large errors produced by the outliers.We use RANSAC, which is the best method for outlier rejection, to get the best fit while having outliers.

RANSAC's main goal is to find an ideal line while examining all of the data points.

It is an iterative approach for estimating coefficents from a set of datapoints that includes outliers.

1) To begin, we choose a minimum of two points at random, as a line requires two points. We iterate through a large number of times (n).
2) Second, we draw a line between these two points and calculate the perpendicular distance between them and the rest of the dataset's points.
3) Third, we determine the number of inliers by establishing a minimum distance threshold; typically, we choose sqrt(3.84*sigma), where sigma is the standard deviation, to cover larger data because this value will cover the majority of the data, or it can be user-specific.
4) Fourth, based on the chosen threshold, we now count the number of inliners. The solution to our best fit line is if the number exceeds the best number.
5) Finally, we repeat the procedure until the best option is found. We then draw our line with the data using the best fit solution.

n_iter: Number of iterations is given by the below formulae

$$N = \frac{log(1p)}{log(1(1e)^2)}$$

where p is the probability of success of two points choosen is the solutio to the line and e is the outlier ratio.

$$e = 1 - \frac{no.of.inliers}{totalnumbers}$$

Analysis

Least Squares

Advantages:

1) It works well when the data doesn't have any outliers and it is robust to noise as well.

Disadvantages:

1) When there are outliers, LS does not perform well and is not particularly resilient because it shifts the line towards outliers.
2) Because the error is calculated using vertical distance (i.e. along the y-axis), it may be erroneous for some data points because the distance would be closer if we used perpendicular instead of vertical.

Total Least Squares Advantages:

1) The perpendicular distance between the points is calculated, which is smaller than the least squared residual.

Disadvantages:

1) It is also prone to outliers

RANSAC

Advantages:

1) it works well with the data noise and outliers.

Disadvantages:

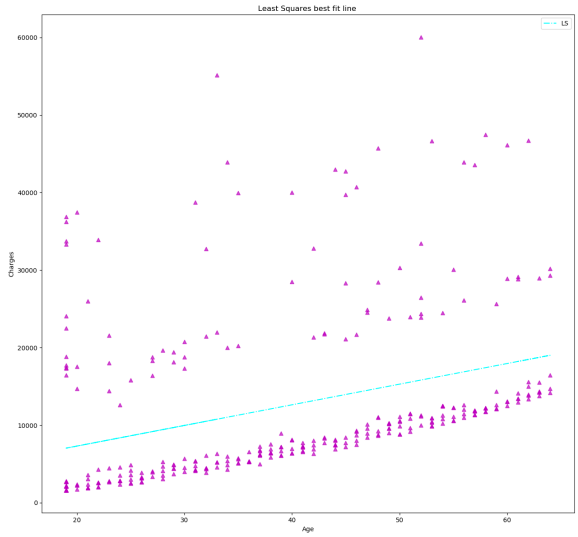
1) It is computationally expensive and and needs more time to fetch the best fit.

In terms of outliers, I believe RANSAC has performed the best of the three algorithms to identify the optimal line to match the data.TLS, on the other hand, has done a better job of covering more inliners, so it may work better if we have more data points.LS has performed a optimal solution but it might not work when outliers are present in the dataset.

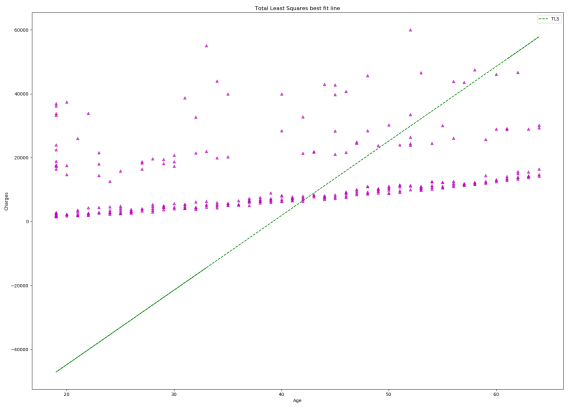
V. Homography and Singular Value Decompositon

To discover the solution to these system of equations, we utilize Singular Value Decomposition to get the homography matrix given four sets of data.

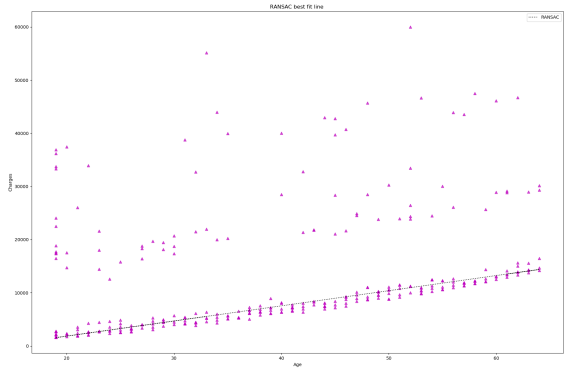
$$\underset{8\times 99\times 1}{A\ X} = 0$$



(a) $LeastSquares$



(b) $TotalLeastSquares$



(c) $RANSAC$

Fig. 5: Curve Fitting using LS,TLS and RANSAC

The dimension of the homography matrix is 3 x 3.

The solution to these system of equations is not possible since the matrix A has more unknowns than variables. We utilize singular value decomposition to get a best fit or a projection of AX, which is a best approximate to the answer, to discover a solution to this matrix. We use pseudo inverse to find the inverse of a m x n matrix. The singular value decomposition is used to find pseudo inverse.

We'll go over how to solve SVD mathematically before moving on to the Homography matrix.

Singular Value Decomposition

$\mathbf{A} = \mathbf{U\Sigma V}^*$

$$A = U\Sigma V^*$$
$$8\times9 \quad 8\times88\times99\times9$$

The pseudo inverse of a m x n or non-square matrix is found via Singular Value Decomposition. Because our A matrix is an 8 x 9 matrix, we must first locate the U, sigma, and V matrices.

The eigen vectors of $A.AT$ make up the matrix, while the eigen vectors of $A.T.A$ make up matrix V and Σ matrix consists of singular values of $A.A^T$ or $A^T.A$ based upon the size of (m,n).

U - Left Singular Vectors of A V - Right Singular Vectors of A Σ - Non-zero singular values of A (found on the diagonal entries) are the square roots of the non-zero eigenvalues of both $A.A^T$ and $A^T.A$.

The singular values should be in decreasing order.

We find the eigen vectors of $(A.A^T - \lambda * I) = 0$ or $(A^T.A - \lambda * I) = 0$.For finding out the Σ we find the eigen values and take a square root of the values and insert in the diagonal of the matrix.

Now we use SVD to discover the U,Σ,V matrix, which is the solution to the system of equations regarding finding the **H** homography matrix, which is to find the nullspace for that system of equations. The solution to those equations is the vector in the V matrix that represents the least singular value.

The key reason for finding the eigen vector with the smallest singular value is that we tend to take the inverse of the $Sigma$ matrix when determining the pseudoinverse for the matrix A, therefore the least value in the singular matrix will correspond to a greater value in the $Sigma^{-1}$ matrix. This value contains the most important data that conveys the closest approximation to the answer.

After calculating the SVD matrix from the python code, we extract the eigen vector of V which is having the smallest singular value.This is the solution to the equations or the Homography matrix.

$$H = \begin{bmatrix} 0.053106 & -0.004917 & 0.614649 \\ 0.017702 & -0.003934 & 0.786750 \\ 0.000236 & -0.000049 & 0.007622 \end{bmatrix}$$

```
if mat.shape[0] > mat.shape[1]:
            s_val=np.mat(np.sqrt(np.diag(val_u)))
            s_val=s_val[:,:mat.shape[1]]
        else:
            s_val=np.mat(np.sqrt(np.diag(val_v)))
            s_val=s_val[:mat.shape[0],:]
```

I have used the above code to get the correct singular values and it is calculated after sorted the eigen vectors of U and V in decreasing order of eigen values.

Calculating the SVD product for a m x n (n $>$ m) is a problem. It produces -1*A instead of A, although the SVD product for a m x n (m $>$ n) is correct.

Please find my github repository for the above implementations.

$https://github.com/karanamrahul/Perception$

REFERENCES

[1] https://cmsc426.github.io/math-tutorial/ransac
[2] https://datascienceplus.com/understanding-the-covariance-matrix/
[3] https://en.wikipedia.org/wiki/68
[4] https://global.oup.com/booksites/content/0199268010/samplesec3
[5] https://blog.rtwilson.com/orthogonal-distance-regression-in-python/