ENPM673
HW-1
Nitesh Jha
Section: 0101

Problem 1

1. Focal Length = 25mm
   Camera sensor width = 14mm
   field of view = 2* arctan(d/2f)
   = 2 * arctan (14/(2*25))
   = 31.28 deg

As the sensor is square shaped, the horizontal and the vertical FOV will be the same.

2. object width a = 5 cm
   distance from sensor = 20m
   Using similarity of triangles and taking proportions:
   5/(20*100) = h / 25
   h = 6.25 * 10^(-2) mm
   The image height is 6.25*10^(-2)mm.
   A 5MP camera has 5*10^(6) pixels and the sensor width is 14mm. The total area of sensor is (14*14)mm^2. Thus, taking proportions again using image area to find pixels occupied by image:
   (14*14) / (5*10^(6)) = ((6.25 * 10^(-2))^2) / P
   P = 99.65 pixels ~ 100pixels
   Thus the object will occupy 100 pixels in the image.

Problem 2

Steps to fit the parabola:
1. Read one frame at a time to process the image as described in the following steps
2. Create a mask which checks two conditions:
   a) red channel value in the frames greater than 150(can be shifted higher as well, but works with this). This is to capture the red ball in the frame
   b) The background is white, which has R value higher than 150, so to filter this out, we impose a condition of having B channel value lesser than 150 (as white has B value as 255).
   Only pixels which satisfy both conditions are set to the original image's R value (255), rest are set to zero.

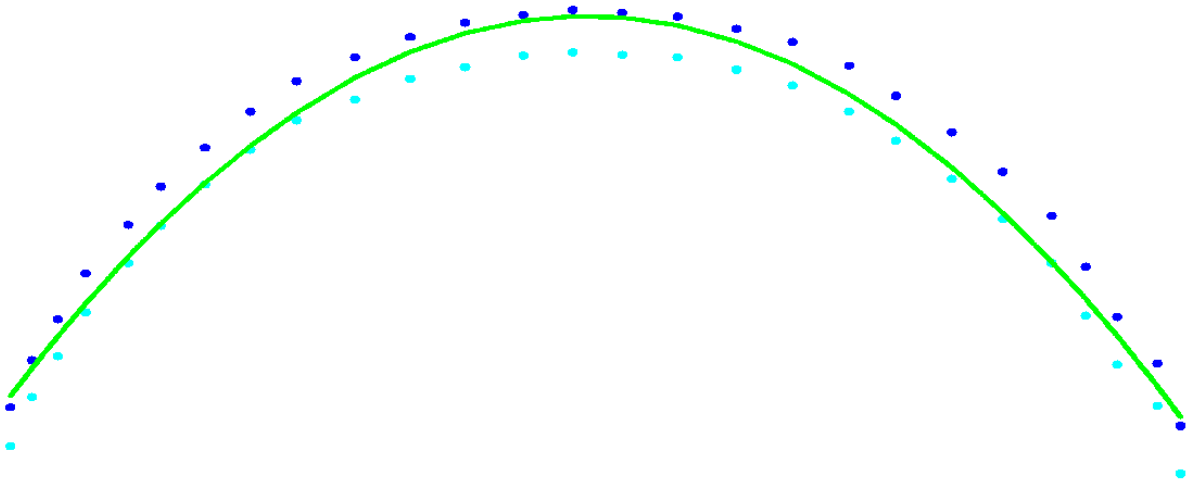
Result of this is as shown above

3. Obtain the coordinates of the pixels where the mask is white (ball locations).
4. Take the extreme values in x and y directions.
5. Find ball center by taking mean of the leftmost point on the ball and the rightmost point.

The above figure shows top (blue) and bottom(light blue) points.
6. Append these two points (top and bottom) shown above to arrays ; the x coordinates (ball center x) and y coordinates for both the topmost point and bottom-most point. This step is optional as we can alternatively just find ball center's x and y coordinates and append one point for one frame. Instead, in the implemented approach, two points, i.e. topmost and bottom-most points are appended into respective x and y arrays for every frame. In the x array, append three values: x^2, x, 1: corresponding to the parabola equation ax^2 + bx + c = y.
7. Fit a 2 degree curve using standard least squares:
   a) Use the below formula to find the pseudoinverse of the x_array matrix:

$$A^{\dagger} = (A^T A)^{-1} A^T$$

Here, A is the x_array matrix of shape (num_points,3).
   b) Then find the parameters if the parabola using the formula:
      X = pseudoinverse(A) * Y
here, X is the parameter array [a,b,c]
8. Use the curve parameters (a,b,c of ax^2 + bx +c = y) to find y coordinate points in the range of x coordinates of the ball.
9. Draw the curve using cv.polylines on the generated points.

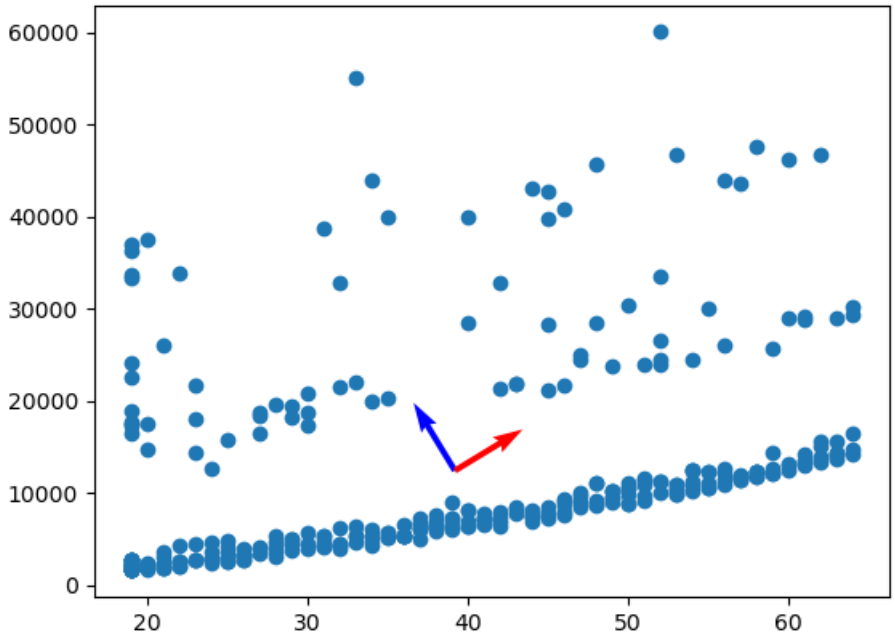


Problem 3:

1. To find the covariance matrix:

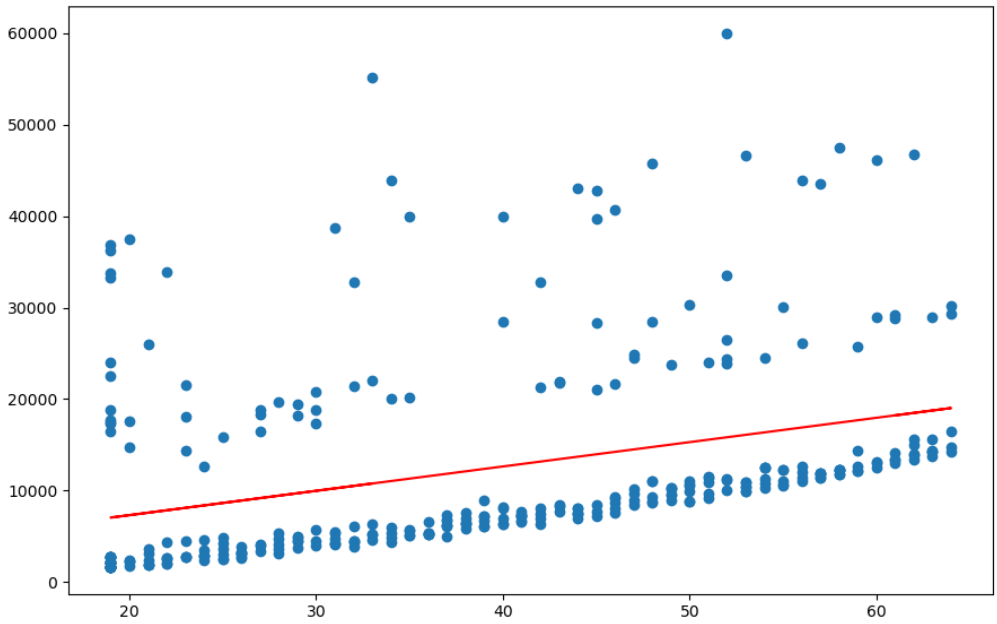
The data is first normalized by dividing the arrays with the max value of array. This is important as the x and y values are not in similar scale and would result in the eigenvectors that do not represent the variation in values. The mean values of age and charges are then calculated. Then, age-age_mean and charges-charges_mean is computed and an element-wise multiplication is performed between these vectors to get covariance values. For the variance of age and charges, an element wise multiplication is performed between age-age_mean and its transpose. This is repeated for variance of charges. Then, these values are stacked to get the covariance matrix.

The eigenvalues and eigenvectors are found using np.linalg.eig(cov_mtx), and plotted along with the data points.



2 and 3.

Standard Least Squares is found using the same approach as mentioned in **Problem 2,** only difference being that here we augment a column of ones in the age vector, to account for the term b in equation a*x + b = y.



We see that the fit line has an offset from the trend of the points due to the outliers. This is because it takes into account the vertical distances only, and outliers with high y-value skew the fit.

Total Least Squares :

The error term is the perpendicular distances of the points from the fit line.

The parameters of total least squares is found by find the solution to

$$\frac{dE}{dN} = 2(U^T U)N = 0$$

To obtain the fit line, the matrix U is calculated which is given by:

U= [age-age_mean ; charges-charges_mean]

To find solution of U.transpose * U * N =0 under constraint magnitude(N) = 1 is found out by calculating the eigen vectors of

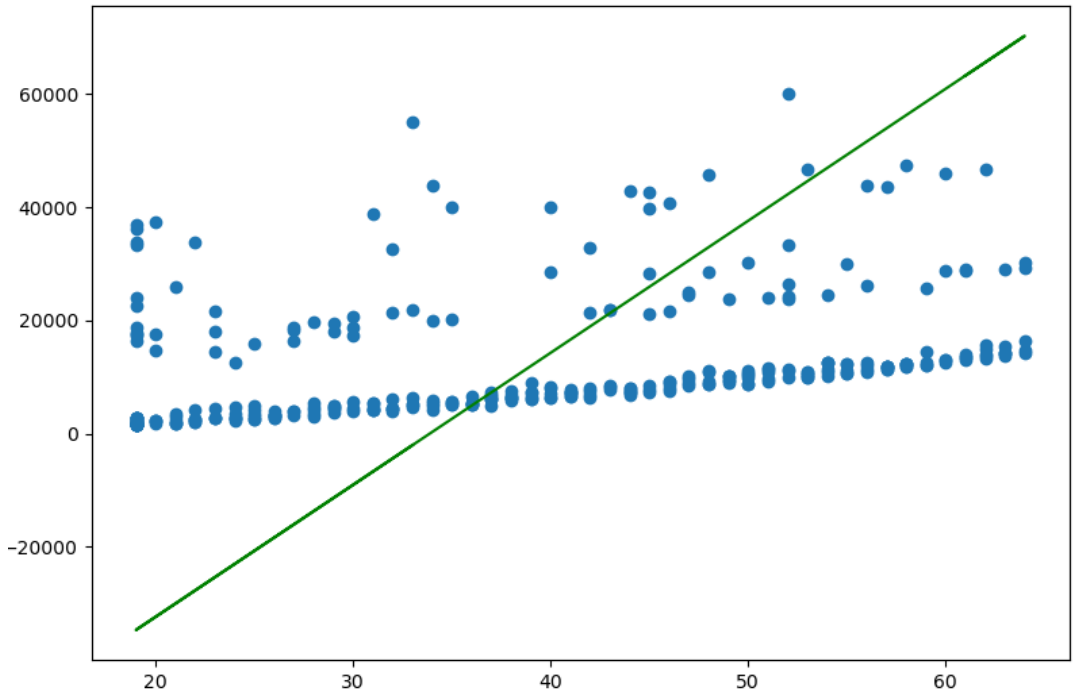
((U.transpose)*U).transpose * (U.transpose*U)

The parameters of the line a,b can be obtained by finding the eigenvector corresponding to the smallest eigenvalue. This is found by using np.linalg.eig( ) and sorting in reverse order of eigenvalues.

The constant d in the equation a*x + b*y = d is then calculated using the equation:

d = a* x_mean + b* y_mean

This gives all parameters of the line. This line is plotted with the data as shown:

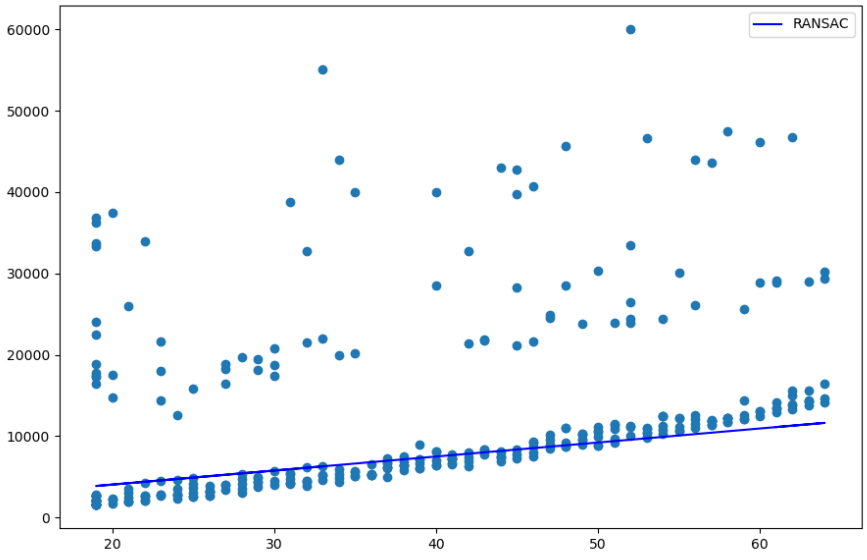


We see that as the fit line does take into account the perpendicular distances between points, the slope is skewed more by the outliers and it results in the line being in the ‘center’ of the data, but it fails to capture the general trend of the data as a result. This, again, is due to high amount of outliers in the data. This method gives a worse fit compared to standard least squares, and will perform better than special cases (when the data shows a more vertical trend).

RANSAC:

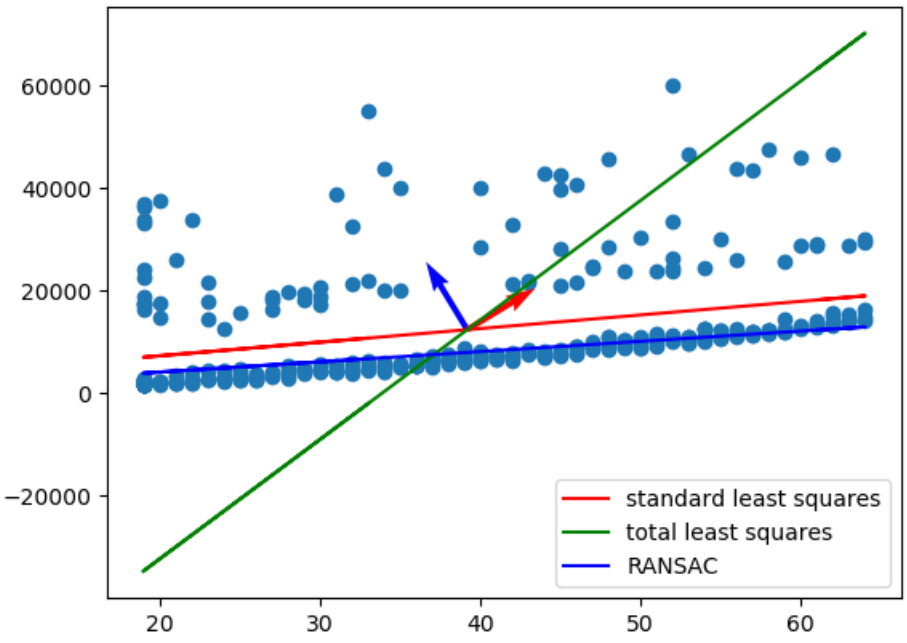
The following parameters were used for RANSAC:
s = 2 (as we need 2 points to define a line)
e = 0.4 (the probability to get a bad point)
p = 0.9999 (desired accuracy)
thresh = 10 (the distance within which a point is classified as an inlier and an outlier outside this threshold)

The number of iterations to perform is calculated by the formula :
N = log(1-e)/(log(1-(1-e)^s)
For these values, iteration count comes out as 20
In each of these iterations, two randomly points are selected and line of form p*x+q*y+r = 0 is formed. Then, distances of all points from this line is calculated and each data point is classified as inlier or outlier depending on their distance. The total count of inliers is calculated.
This is done for each iteration and the line with the maximum inliers is taken as the fit line for RANSAC.



The result captures the actual trend in the data accurately because of the random iterative approach it takes. When selecting points to fit a line, the probability of selecting an outlier is proportional to the amount of outliers present in the data. This percentage is provided by eyeballing. If the percentage of outliers is higher, this will result in a higher amount of iterations to check for best fit lines, hence increasing chances of selecting two points which accurately capture the trend.
However, this may fail sometimes when an outlier is randomly selected at every iteration. A solution to this may be to run the RANSAC iterations multiple times to find best fit across the runs.

Combined result:



Problem 4

A rectangular matrix can be represented as three matrix components:
A = U*S*V.t
Consider A.T * A:

A.T * A = V * S.T * U.T * U * S * V.T
As U is an orthogonal matrix, U.T * U = I
Thus, A.T * A = V * S.T * S * V.T
This is of the form of eigen decomposition. Hence, V is composed of eigenvectors of A.T * A and S^2 is the eigenvalues of A.T * A.

Similarly, consider A * A.T:

A * A.T = U*S*V.T * V * S.T * U.T
As V is also orthogonal, V*V.T = I
Hence, A * A.T = U*S * S.T * U.T.
This is of the form of eigen decomposition. Hence, U is composed of eigenvectors of A* A.T and S^2 is the eigenvalues of A * A.T

The SVD of rectangular matrix A is found by finding eigenvectors and eigenvalues of A.transpose*A and A*A.T as shown above.
A = U * S * V.T
U : matrix consisting of columns which are eigenvectors of A*A.T, in decreasing order of eigenvalues
S : diagonal matrix consisting of square root of eigenvalues of A*A.T. Here, the last column of S will be padded with zeros as the shape required is 8x9.
V : matrix consisting of columns which are eigenvectors of A.T*A, in decreasing order of eigenvalues

The eigenvectors and eigenvalues can be calculated using function np.linalg.eig( ).
A : 8 x 9
U : 8 x 8
S : 8 x 9
V : 9 x 9

Then, the homogenous matrix H is given by the last column of V.T (smallest value of A.T* A), reshaped to (3x3).

```
Homography matrix is:
[[ 5.31056350e-02 -4.91718843e-03 6.14648552e-01]
[ 1.77018784e-02 3.93375075e-03 7.86750146e-01]
[ 2.36025045e-04 -4.91718843e-05 7.62164204e-03]]
```