ENPM673- Homework1

Charu Sharma – 117555448
charu107@umd.edu

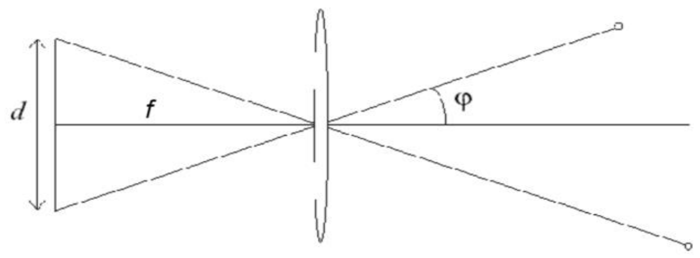
Problem 1:

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 **25mm**. 1. Compute the Field of View of the camera in the horizontal and vertical direction. 2. Assuming you are detecting a square shaped object with width 5cm, placed at a distance of 20 meters from the camera, compute the minimum number of pixels that the object will occupy in the image.

Given with the details of the camera specifications, the field of view angle can be computed. From the FOV angle we can get the vertical Field of View. And since it is a square sensor, the horizontal field of view would be the same.

Field of view depends on the focal length of the lens. The size of Field of View is governed by the size of the retina.

FOV equation, where d is the sensor width and f is the focal length of the lens,



Size of field of view governed by size of the camera retina:

$$\varphi = \tan^{-1}\left(\frac{d}{2f}\right)$$

With d = 14 mm and f = 25 mm

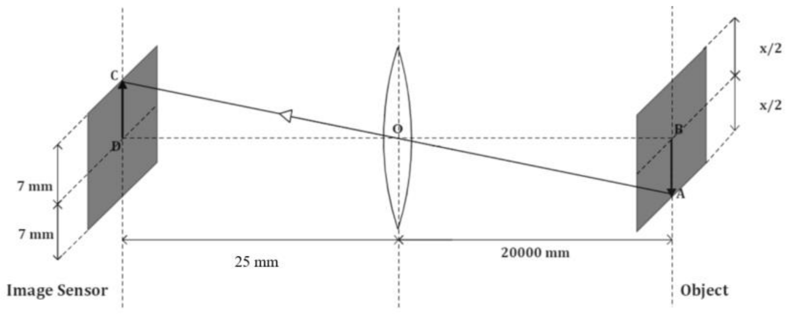
$\varphi = \tan^{-1}\left(\frac{14}{2 * 25}\right)$

$\varphi = \tan^{-1}(0.28)$

$\varphi = 15.64\ deg$

Field of view in horizontal and vertical direction is two times the FOV angle. Therefore, **31.28 deg.**

2.



Ray diagram for the given problem

Properties of similar triangles to ABO and CDO,

$$\frac{AB}{BO} = \frac{CD}{DO}$$

$$\frac{x/2}{20000} = \frac{7}{25}$$

x = 11200 mm

Field of view represents area that the camera can capture. Therefore, the FOV is square of x i.e., 125440000 mm^2

To get the number of pixels n per mm^2, we divide the resolution to the image sensor with Field of view area.

$$n = \frac{5000000}{125440000}$$

n = 0.0398 pixels/mm^2

Because it is a square sensor, Pixel size and the aspect ratio was considered the same.

Minimum number of pixels that the object occupies on the image sensor will be the area of object times n

Minimum number of pixels = area of object * n

Minimum number of pixels = 50 * 50 * 0.0398

Minimum number of pixels = 99.65 = 100 pixels

The object will occupy **100 pixels** on the image sensor.

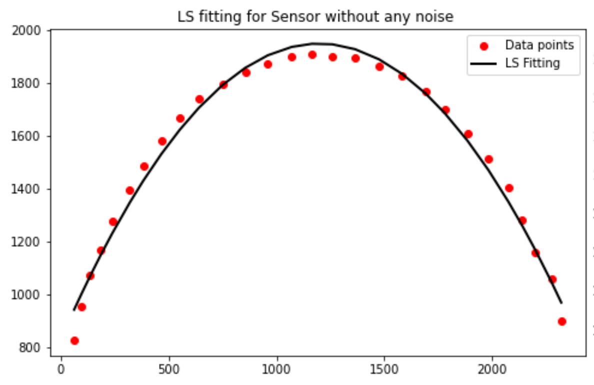
Interesting Observation: Minimum number of pixels are rounded off to the next whole number

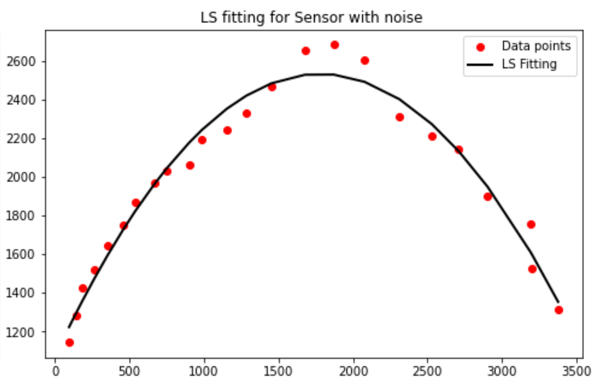
Problem 2:

A ball is thrown against a white background and a camera sensor is used to track its trajectory. We have a near perfect sensor tracking the ball in video1 and the second sensor is faulty and tracks the ball as shown in video2. Clearly, there is no noise added to the first video whereas there is significant noise in the second video. If the trajectory of the ball follows the equation of a parabola:

A. Use Standard Least Squares to fit curves to the given videos in each case. You must plot the data and your best fit curve for each case. Submit your code along with the instructions to run it.

The curve fitting with Lease Square Method (LS) for data with noise and without is given by the below images





Interesting Observation:

- Converting the images to gray scale made the tasks easier
- Computing the centers to plot the graph gave a better curve

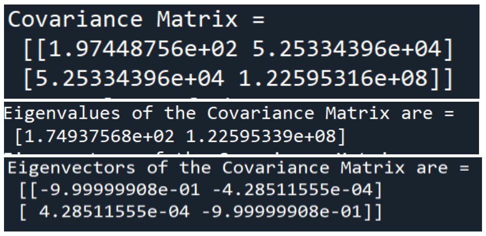
Problem 3:

In the above problem, we used the least squares method to fit a curve. However, if the data is scattered, this might not be the best choice for curve fitting. In this problem, you are given data for health insurance costs based on the person's age. There are other fields as well, but you must fit a line only for age and insurance cost data. The data is given in .csv file format and can be downloaded from here.

1. Compute the covariance matrix and find its eigenvalues and eigenvectors. Plot the eigenvectors on the same graph as the data.

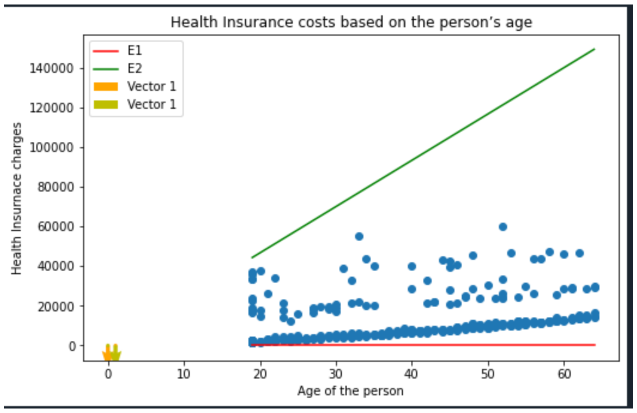
Preprocessing

Read the file with csvreader and sorted the age and health Insurance cost. Saved those data into two separate lists used that to calculate the covariance matrix.



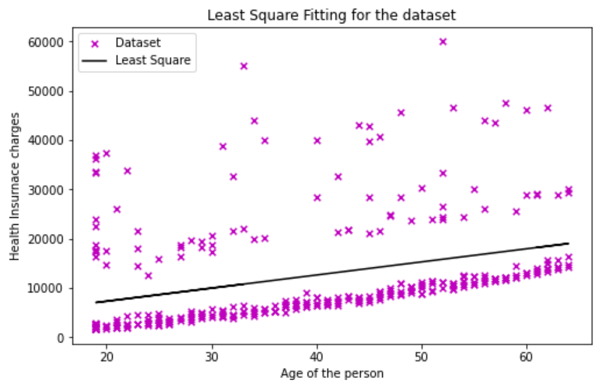
```
Covariance Matrix =
 [[1.97448756e+02 5.25334396e+04]
 [5.25334396e+04 1.22595316e+08]]
Eigenvalues of the Covariance Matrix are =
 [1.74937568e+02 1.22595339e+08]
Eigenvectors of the Covariance Matrix are =
 [[-9.99999908e-01 -4.28511555e-04]
 [ 4.28511555e-04 -9.99999908e-01]]
```

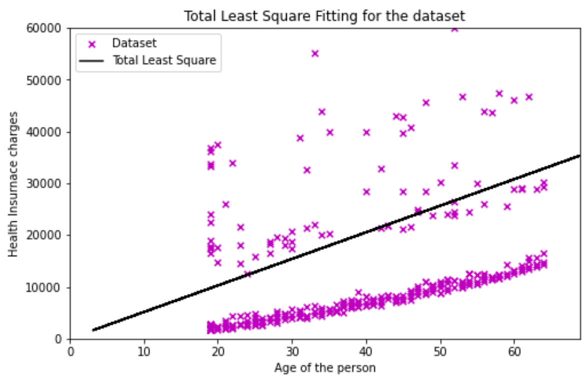
Plotting the eigenvectors

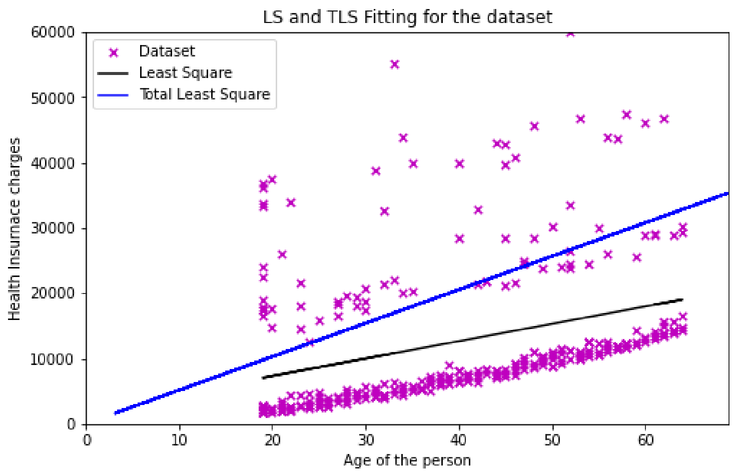


2. Fit a line to the data using linear least square method, total least square method and RANSAC. Plot the result for each method and explain drawbacks/advantages for each.

Fitting the data according to linear least square method, total least square method and RANSAC.







The TLS method has a bunch of outliers seemingly might not be a very good fitting graph for this data.

<u>Advantages:</u>

- LS method is simpler and faster than others. But it has a lot of outliers
- RANSAC genrally gives more accurate fitting

<u>Disadvantages:</u>

- RANSAC takes longer than other methods

3. Briefly explain all the steps of your solution and discuss which would be a better choice of outlier rejection technique for each case.

LS (Least Square):

The least-squares method is a crucial statistical method that is practised to find a regression line or a best-fit line for the given pattern. This method is described by an equation with specific parameters. The method of least squares is generously used in evaluation and regression. In regression analysis, this method is said to be a standard approach for the approximation of sets of equations having more equations than the number of unknowns.

The equation of least square line is given by Y = a + bX

Normal equation for ‘a’:

∑Y = na + b∑X

Normal equation for ‘b’:

∑XY = a∑X + b∑X2

Solving these two normal equations we can get the required trend line equation.

Thus, we can get the line of best fit with formula y = ax + b

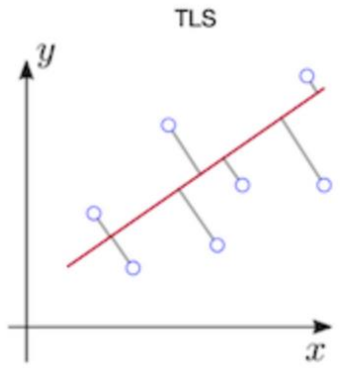
$$E = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \quad B = \begin{bmatrix} m \\ b \end{bmatrix}$$

This equation is solved by

$$\boxed{B = (X^T X)^{-1}(X^T Y)}$$

TLS (Total Least Square):



Unlike LS, TLS considers observational errors and minimizes errors for independent variable

and dependent variable. We want to minimize errors, for an independent variable and for a dependent variable. Those errors are considered as to contain both an observational error and a residual.

Geometrically, this is the perpendicular distance to the regression line.

$$\operatorname{argmin}_{E,F} \|[E\ F]\|_F, \qquad (X+E)B = Y+F$$

After solving a covariance matrix of X and Y with SVD, we ended up getting the equation below. *U* is the left singular vectors of X^TY and Σ is the diagonal matrix with singular values on its diagonal.

$$[E\ F] = -U_Y \Sigma_Y \begin{bmatrix} V_{XY} \\ V_{YY} \end{bmatrix}^* = -[X\ Y] \begin{bmatrix} V_{XY} \\ V_{YY} \end{bmatrix} \begin{bmatrix} V_{XY} \\ V_{YY} \end{bmatrix}^*.$$

The parameter vectors *B* is defined as:

$$B = -V_{XY} V_{YY}^{-1}.$$

In our implementation, we calculated the *[E F]* and add the return to *tx.* The vertically stacked vectors *[Vxy Vyy]* is the whole last column of right singular vectors of tx.T,ty. *Vxy* and *Vyy* are truncated the number of *X* variables.

The first *n* columns are for errors *E*, and *n* to last column for errors *F*. We can also calculate the estimate *Y* value through *Y = [X, E]B*,

RANSAC (Random Sample Consensus Algorithm):

When our dataset has many outliers, RANSAC delivers robust results.

The algorithm is simple:

a) We select s datapoints at random from our entire dataset and use it to estimate a curve. Here, s is the minimum number needed to fit our model.
   s = 3 for our case as we are trying to estimate a second-degree curve. (SLS has been used for this step. TLS can be used too.)
b) Next, we count the number of all the datapoints that lie within a threshold the from the curve. This count inliers_iter will be appended to the list inliers. We also store parameters b for the curve to the list all_b.
c) We repeat steps 1 and 2 again and again for t trials.
d) Our final curve is that one out of all trials, which gave maximum inliers. We run argmax on the inliers list and use the obtained index to find curve parameters from all_b.

The number of trials t can be obtained as:

$$T = \frac{log\ (1-p)}{\log\ (1-(1-e)^s)}$$

Here,
s= samples = 2 for our case
p= desired probability that we get a good sample = 0.999, this is something we get to decide.
e= outlier ratio e = 25% by simply observing the dataset, we can see that around 6/24 points are outliers

From the plots, we can see that LS algorithm gives a better fitting the TLS with this data.

<u>Interesting Observation:</u>

- Challenging to implement convergence at first. Seemingly to apply it in a loop worked amazingly well.
- It was confusing to plot the eigenvectors at first. Used plt.quiver to achieve that

- Quantitative comparison between all the methods will give better opinion on the performances

Problem 4: [20 Points]

The concept of Homography in Computer Vision is used to understand, explain, and study visual perspective, and specifically, the difference in appearance of two plane objects viewed from different points of view. Given 4 corresponding points on the two different planes, the Homography between them is computed using the following system of equations Ax = 0, where A is given by

$$A = \begin{bmatrix} -x1 & -y1 & -1 & 0 & 0 & 0 & x1*xp1 & y1*xp1 & xp1 \\ 0 & 0 & 0 & -x1 & -y1 & -1 & x1*yp1 & y1*yp1 & yp1 \\ -x2 & -y2 & -1 & 0 & 0 & 0 & x2*xp2 & y2*xp2 & xp2 \\ 0 & 0 & 0 & -x2 & -y2 & -1 & x2*yp2 & y2*yp2 & yp2 \\ -x3 & -y3 & -1 & 0 & 0 & 0 & x3*xp3 & y3*xp3 & xp3 \\ 0 & 0 & 0 & -x3 & -y3 & -1 & x3*yp3 & y3*yp3 & yp3 \\ -x4 & -y4 & -1 & 0 & 0 & 0 & x4*xp4 & y4*xp4 & xp4 \\ 0 & 0 & 0 & -x4 & -y4 & -1 & x4*yp4 & y4*yp4 & yp4 \end{bmatrix}, x = \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix}$$

For the given point correspondences,

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

Find the Homography matrix:

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix}$$

1. Show mathematically how you will compute the Singular Value Decomposition (SVD) for the matrix A. 2. Write python code to compute the SVD.

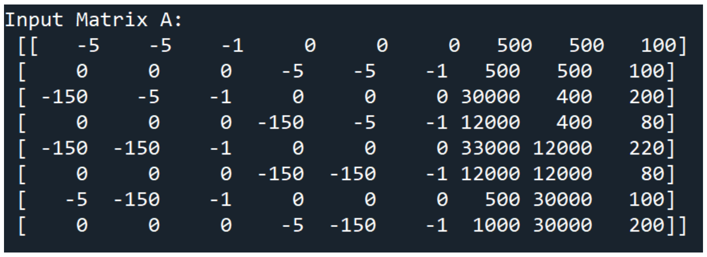
For an m× n matrix A of rank r there exists a factorization (Singular Value Decomposition = SVD) as follows:

$$A = U\Sigma V^T$$
$$U \subset R^{m\times m}$$
$$\Sigma \subset R^{m\times m}$$
$$V \subset R^{n\times n}$$

The matrices U, Σ, V are always real – this is unlike eigenvectors and eigenvalues of A which may be complex even if A is real.

The singular values are always non-negative, even though the eigenvalues may be negative.

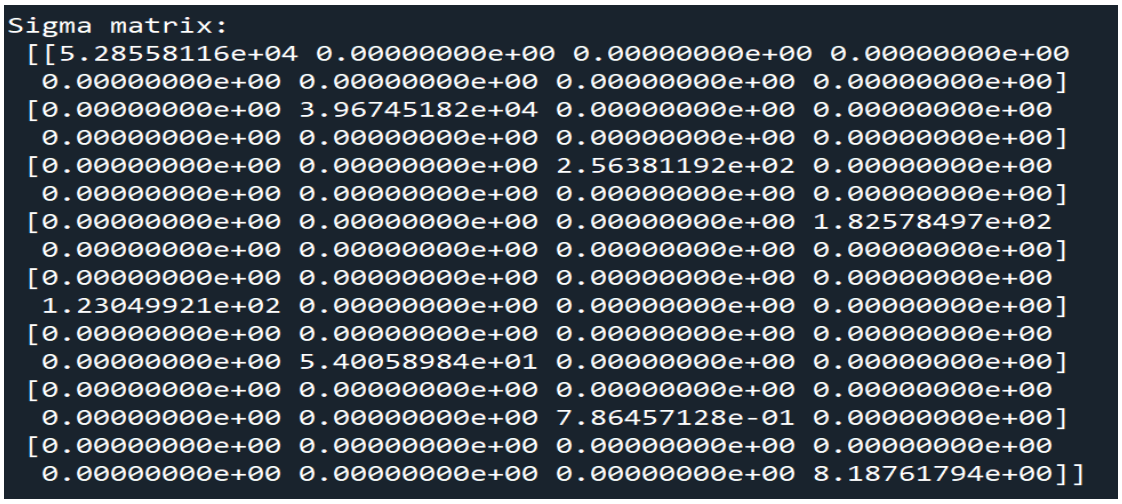
Substituting the specific values gives us the input matrix A as



```
Input Matrix A:
 [[    -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    400    200]
 [     0      0      0   -150     -5     -1  12000    400     80]
 [  -150   -150     -1      0      0      0  33000  12000    220]
 [     0      0      0   -150   -150     -1  12000  12000     80]
 [    -5   -150     -1      0      0      0    500  30000    100]
 [     0      0      0     -5   -150     -1   1000  30000    200]]
```

To find Σ:

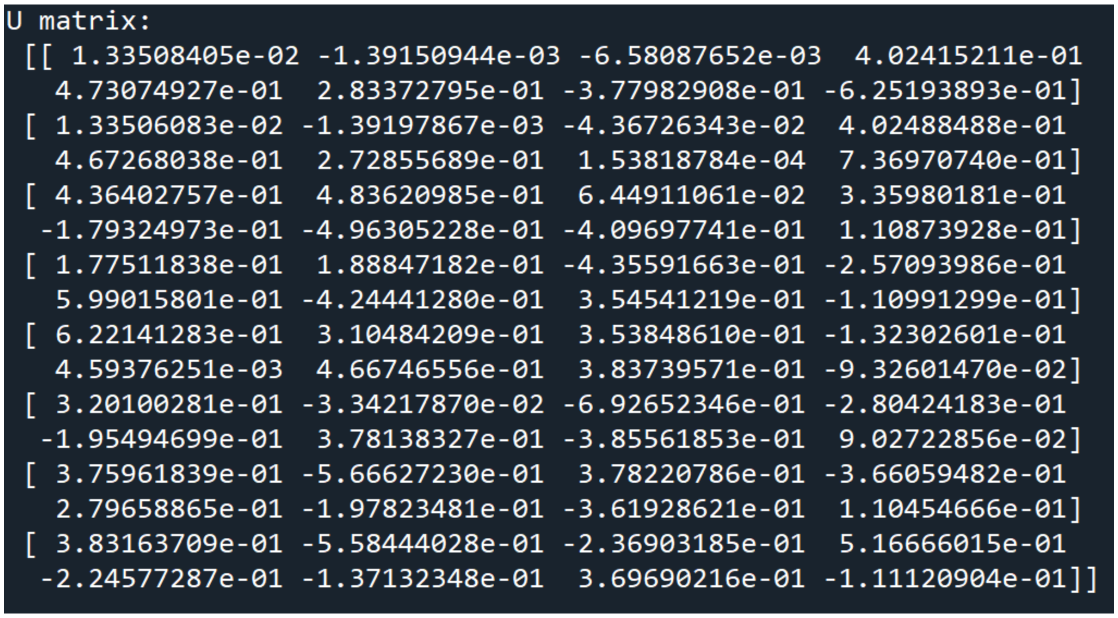
a) Compute AA^T (or A^TA)
b) Find its eigenvalues. They will be real because AA^T is a real symmetric matrix.
c) We take non-zero eigenvalues λ and take their absolute values. Square root the absolute values to obtain what are called singular values. (Even though eigenvalues)
d) Arrange these singular values in descending order and then in a diagonal matrix.This diagonal matrix is Σ (sigma).



```
Sigma matrix:
 [[5.28558116e+04 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 3.96745182e+04 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 2.56381192e+02 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 1.82578497e+02
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  1.23049921e+02 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 5.40058984e+01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 7.86457128e-01 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 8.18761794e+00]]
```

To find U:

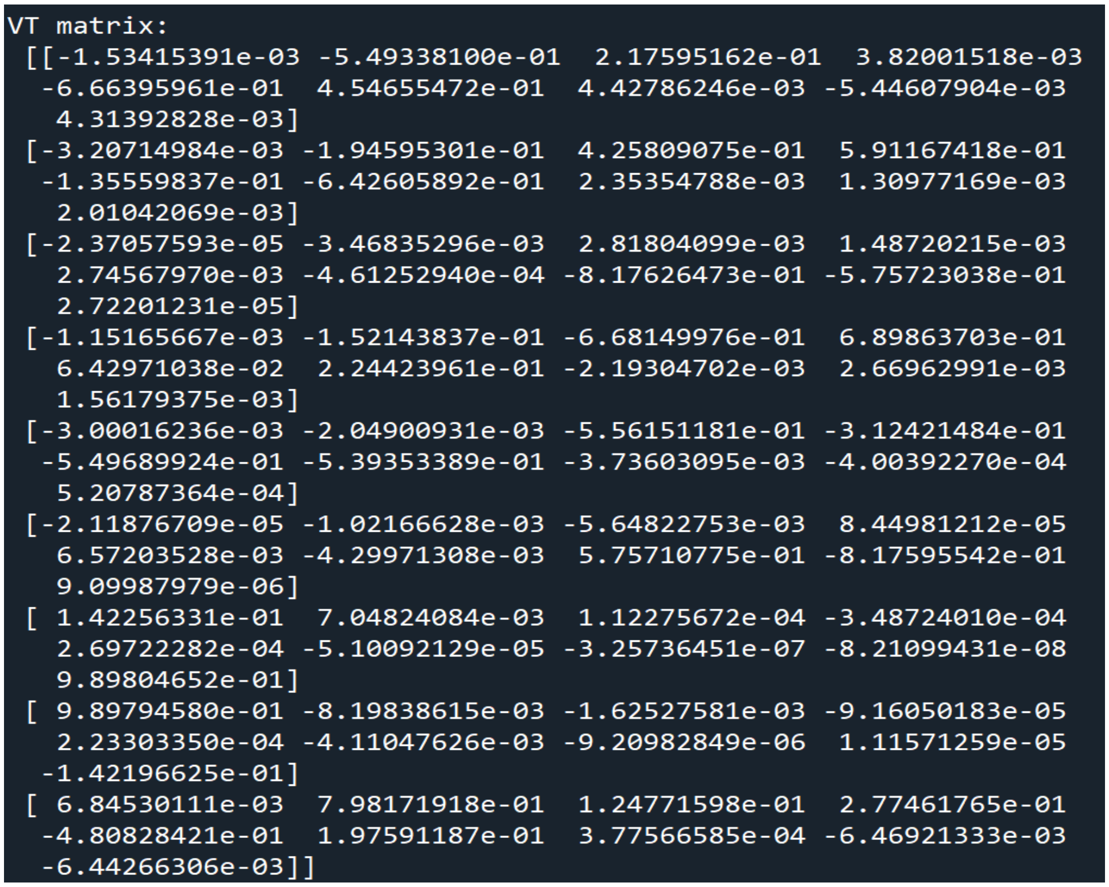
a) Find eigenvectors of AA^T (called the left singular vectors).
b) Arrange them as columns in order of singular values in Σ.



```
U matrix:
 [[ 1.33508405e-02 -1.39150944e-03 -6.58087652e-03  4.02415211e-01
    4.73074927e-01  2.83372795e-01 -3.77982908e-01 -6.25193893e-01]
  [ 1.33506083e-02 -1.39197867e-03 -4.36726343e-02  4.02488488e-01
    4.67268038e-01  2.72855689e-01  1.53818784e-04  7.36970740e-01]
  [ 4.36402757e-01  4.83620985e-01  6.44911061e-02  3.35980181e-01
   -1.79324973e-01 -4.96305228e-01 -4.09697741e-01  1.10873928e-01]
  [ 1.77511838e-01  1.88847182e-01 -4.35591663e-01 -2.57093986e-01
    5.99015801e-01 -4.24441280e-01  3.54541219e-01 -1.10991299e-01]
  [ 6.22141283e-01  3.10484209e-01  3.53848610e-01 -1.32302601e-01
    4.59376251e-03  4.66746556e-01  3.83739571e-01 -9.32601470e-02]
  [ 3.20100281e-01 -3.34217870e-02 -6.92652346e-01 -2.80424183e-01
   -1.95494699e-01  3.78138327e-01 -3.85561853e-01  9.02722856e-02]
  [ 3.75961839e-01 -5.66627230e-01  3.78220786e-01 -3.66059482e-01
    2.79658865e-01 -1.97823481e-01 -3.61928621e-01  1.10454666e-01]
  [ 3.83163709e-01 -5.58444028e-01 -2.36903185e-01  5.16666015e-01
   -2.24577287e-01 -1.37132348e-01  3.69690216e-01 -1.11120904e-01]]
```

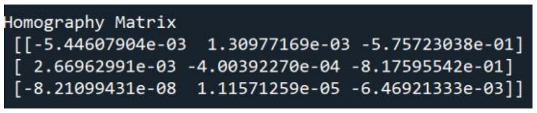
To find V:

a) Find eigenvectors of A^TA (called the right singular vectors).
b) Arrange them as columns in order of singular values in Σ.



```
VT matrix:
 [[-1.53415391e-03 -5.49338100e-01  2.17595162e-01  3.82001518e-03
   -6.66395961e-01  4.54655472e-01  4.42786246e-03 -5.44607904e-03
    4.31392828e-03]
  [-3.20714984e-03 -1.94595301e-01  4.25809075e-01  5.91167418e-01
   -1.35559837e-01 -6.42605892e-01  2.35354788e-03  1.30977169e-03
    2.01042069e-03]
  [-2.37057593e-05 -3.46835296e-03  2.81804099e-03  1.48720215e-03
    2.74567970e-03 -4.61252940e-04 -8.17626473e-01 -5.75723038e-01
    2.72201231e-05]
  [-1.15165667e-03 -1.52143837e-01 -6.68149976e-01  6.89863703e-01
    6.42971038e-02  2.24423961e-01 -2.19304702e-03  2.66962991e-03
    1.56179375e-03]
  [-3.00016236e-03 -2.04900931e-03 -5.56151181e-01 -3.12421484e-01
   -5.49689924e-01 -5.39353389e-01 -3.73603095e-03 -4.00392270e-04
    5.20787364e-04]
  [-2.11876709e-05 -1.02166628e-03 -5.64822753e-03  8.44981212e-05
    6.57203528e-03 -4.29971308e-03  5.75710775e-01 -8.17595542e-01
    9.09987979e-06]
  [ 1.42256331e-01  7.04824084e-03  1.12275672e-04 -3.48724010e-04
    2.69722282e-04 -5.10092129e-05 -3.25736451e-07 -8.21099431e-08
    9.89804652e-01]
  [ 9.89794580e-01 -8.19838615e-03 -1.62527581e-03 -9.16050183e-05
    2.23303350e-04 -4.11047626e-03 -9.20982849e-06  1.11571259e-05
   -1.42196625e-01]
  [ 6.84530111e-03  7.98171918e-01  1.24771598e-01  2.77461765e-01
   -4.80828421e-01  1.97591187e-01  3.77566585e-04 -6.46921333e-03
   -6.44266306e-03]]
```

Obtained Homography Matrix:



```
Homography Matrix
 [[-5.44607904e-03  1.30977169e-03 -5.75723038e-01]
 [ 2.66962991e-03 -4.00392270e-04 -8.17595542e-01]
 [-8.21099431e-08  1.11571259e-05 -6.46921333e-03]]
```

Interesting Observation:

- Working on Asymmetric matrix was challenging
- NumPy did not perfectly sort eigen values, which led to a bit of discrepancies in the final calculations. I have tried making it work still.