Kamakshi Jain
Rohan Singh
Abhinav Modi

ENPM 673
February 14, 2019

Homework 1

Part 1

Eigenvector and Eigenvalues : Eigenvectors are special vectors that maintain their direction upon linear transformation by any matrix(T). Mathematically,

$$T * v = \lambda * v \tag{1}$$

where λ is a scalar, A.K.A. Eigenvalue.
As, seen by the equation, on application of transformation(T) matrix, the vector simply scales and does not undergo any kind of rotation. So essentially, eigenvectors can be used to find axis of rotation of a given dataset.
A few important characteristics of Eigenvector are:

1. They are linearly independent
2. These dont change their orientation after linear transformation

Implementation

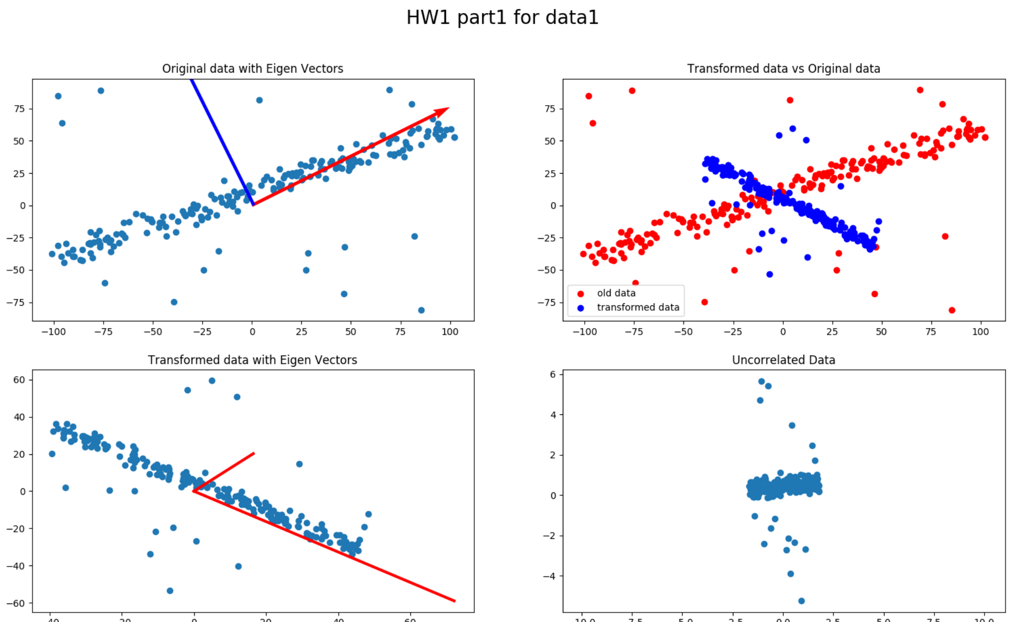
Covariance is a measure of how much two variables vary together.Using covariance we can calculate entries of the covariance matrix which is a square matrix.The diagonal of this matrix is variance and others are covariances.

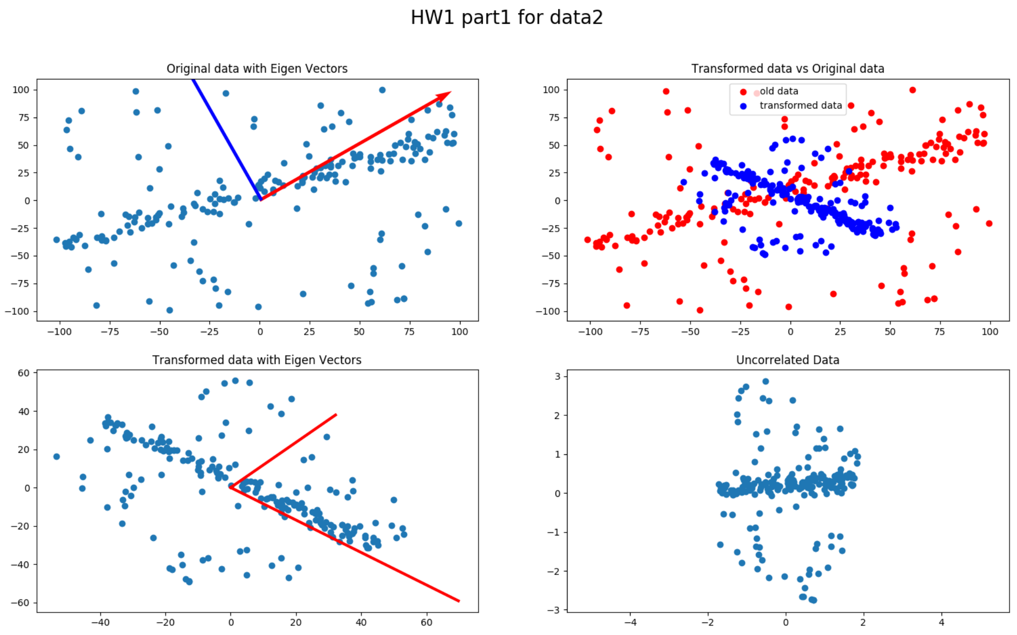
We want to show how linear transformation affects the original data set and its covariance matrix.By using the mean values at the origin, the covariance matrix of the uncorrelated data is found.

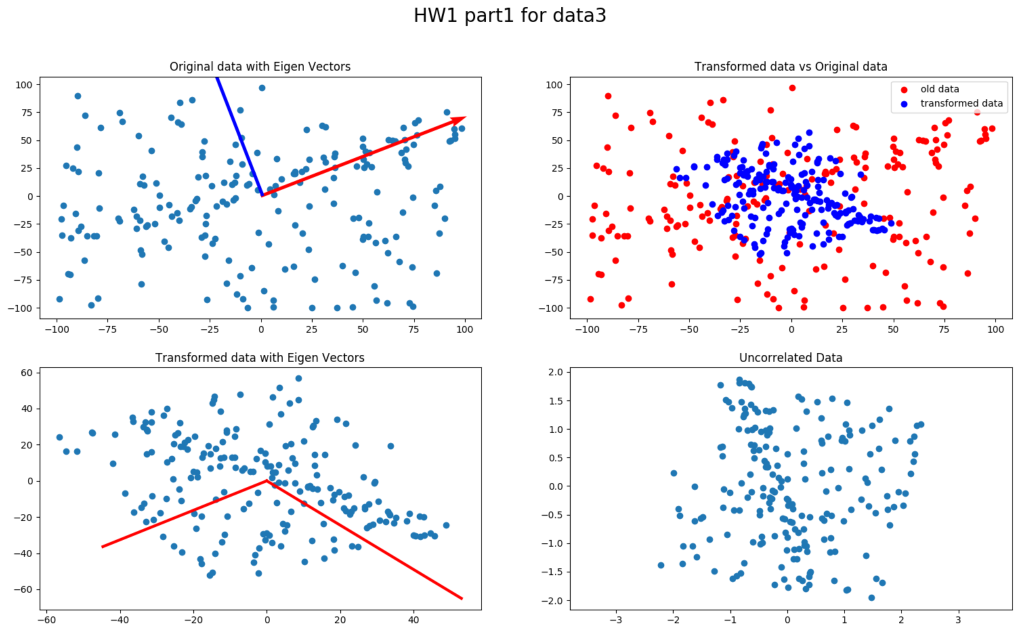
The original data is scaled by 0.5 in x-axis and 0.4 in y-axis. Further, this scaled data is rotated by 60 degrees.What we observe is that, from the covariance matrix of the transformed data set , we can extract the scaling matrix and the rotation matrix by eigen decomposition of the covariance matrix.

The eigen vectors are vectors representing the largest variance of the data and the eigen value gives the magnitude of variance in the corresponding direction.Thus, eigen vector represents the rotation matrix and the squareroot of eigen value gives a scaling matrix.

Finally, by using the inverse of transformation matrix we can remove correlation data.

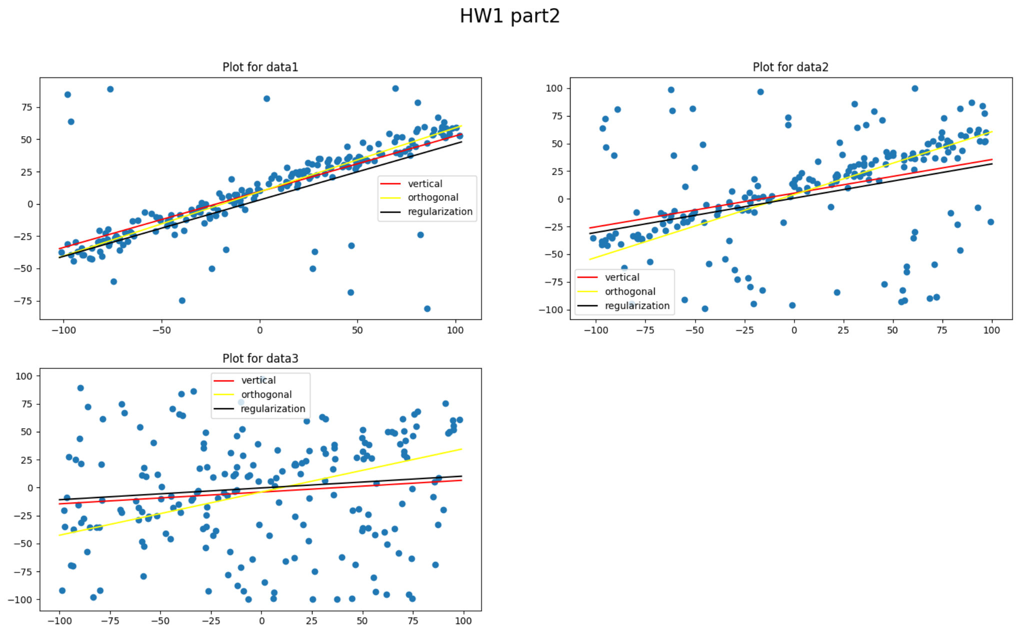

HW1 part1 for data1
Original data with Eigen Vectors
Transformed data vs Original data
old data
transformed data
Transformed data with Eigen Vectors
Uncorrelated Data


HW1 part1 for data2
Original data with Eigen Vectors
Transformed data vs Original data
old data
transformed data
Transformed data with Eigen Vectors
Uncorrelated Data



Part 2

We used Least Squares line fitting using (a)vertical distance and (b)orthogonal distances.



- **For Vertical distance :**
  The goal is to minimize the sum of the squared vertical distance between the y data value and the corresponding y values on the fitted line.
  The error function is : $\sum_{i=1}^{n}(y_i - mx_i - b)^2$. Further for matrix method, The line equation is XB=Y. where B= $(X^T X)^{-1}(X^T Y)$

- **For Orthogonal distance :**
  The goal is to minimize the orthogonal/perpendicular distance from the The error function is data points to the fitted line.
  $\sum_{i=1}^{n}(ax_i + by_i - d)^2$.

Conclusion

For all the three data sets, it has been observed that the orthogonal distance method produces a better fit.

Part 3

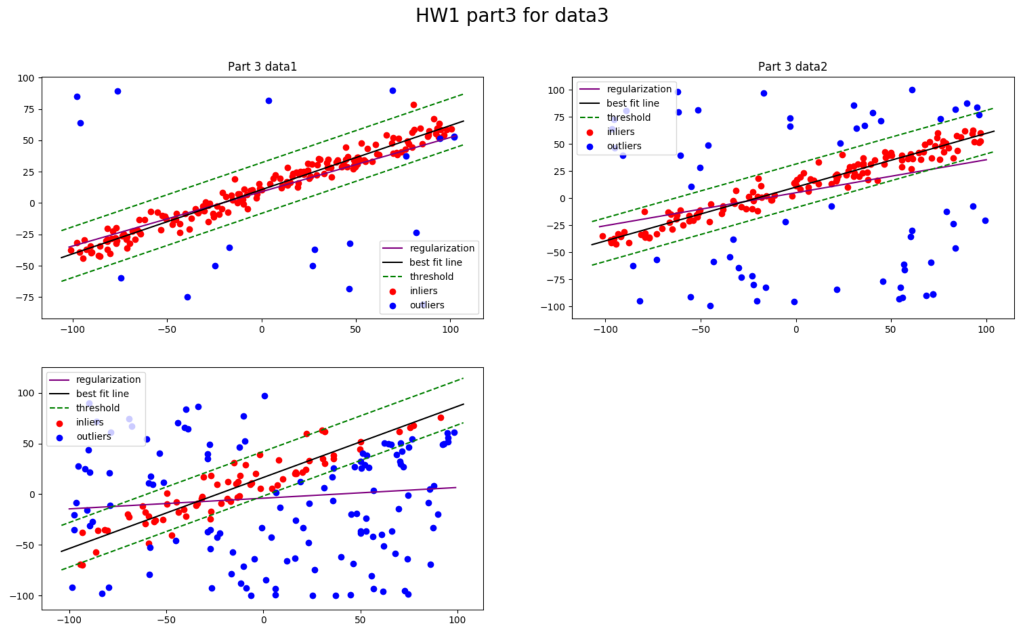
We now look at outlier rejection methods. We see and compare the response of our line fitting to the varied levels of noise(outliers) in our 3 datasets by applying (a)Total Least Squares, (a)Least Squares with Regularization and (b) RANSAC. The response of Total Least Squares is shown above.

1. **DATASET 1** This dataset compromises few outliers.On observing the lines formed by the 3 methods, we can observes that the line fitted by RANSAC is a better fit. Here RANSAC works with outlier ratio threshold e = 0.1 (ratio of outliers to total number of points), N = 3 (number of iterations) and p = 0.99 (at least one random sample is free from outliers).

2. **DATASET 2** This dataset has some noise. Total Least Squares fits a line closer in direction but with an offset from center of the data. Least Squares with Regularization is more centred after some tuning but its direction isn't as close to the direction of the most dense data. RANSAC clearly outperforms all other methods .Here RANSAC works with e = 0.5, N = 18 and p = 0.99.

3. **DATASET 3** This dataset is very noisy. Least Squares, least square with regularization and total least square, all fail. RANSAC works but only on increasing e and N, here the parameters selected are: e = 0.7, N = 18 and p = 0.99.

4. **Method Limitations :**



(a) After working with these datasets, we have realised using Total Least Squares is only helpful when outlier ratio is < 0.1. For more noisy data, regularization or outlier rejection with RANSAC is a better bet.

(b) Least Squares with Regularization requires a lot of tuning even for mildly noisy data and still doesn't give good results.

(c) There are too many parameters to tune for RANSAC. And figuring out how each of those affect the algorithm took us some time.

(d) Number of iterations increase as we increase outlier ratio threshold e. We also observed that having a high **e** sometimes meant some of the inliers (points within distance threshold) got left out and fine tuning took time.

(e) Rarely, but surely, we found that our randomly generated point pairs were not good enough to fit the threshold conditions for RANSAC and thus we had to run the program again to find a better fit. Increasing N or "smartly choosing" random samples might improve the situation but that would slow down the algorithm or make it complex to implement.

5. **An Interesting Problem :** Working on different setups turned out to be a little difficult for our group. While running RANSAC on python2, we found that the condition for outlier ratio seemed to turn positive even when it was clearly false. The same code worked perfectly fine on python3. Hours of scratching our brains and suspecting the random number generator, we found python2 rounded the same variable to 0 while python3 didn't.