CS-215 Assignment 2

Teja Bale and Janaki Ram Puli

Autumn 2021

Contents

1 Question 1

1.1 Generating Random points in Ellipse

To Run : `one_a`

Results present as `one/one_a.png`

Algorithm for generating random points (in 2D) distributed uniformly inside the ellipse:
Generally we know that random point on the ellipse $(a\cos\theta, b\sin\theta)$.
The random point inside the ellipse is given by $(ar\cos\theta, br\sin\theta)$. Here r is in the range [0, 1] and ar and br denotes the length of major and minor axis and they both change uniformly so r in the range of 0 to 1 we can get every possible point uniformly for a random θ. since change in r changes the distance of the point for fixed θ it is same as generating a random point within a circle (uniformly).

Why sqrt? : Let us take a circle of unit radius for simplicity. Since we know that circle is an ellipse with a=b=r, we can easily see that whatever we derive in case of circle should be applicable to ellipse as well. Since the circumference of a circle(2πr) we have a linear function of r and as r_max = 1 $PDF(X) = 2x$. So $CDF = x^2$, by inverse transform method as we saw in first assignment $y = x^2$, our new function y will be $\sqrt{x}$. So r is given by: $r = \sqrt{(v)}$; v is the random variable generated in the range [0,1]. Square Root is used here to ensure uniformity in drawn points inside ellipse, So that center won't get crowded.

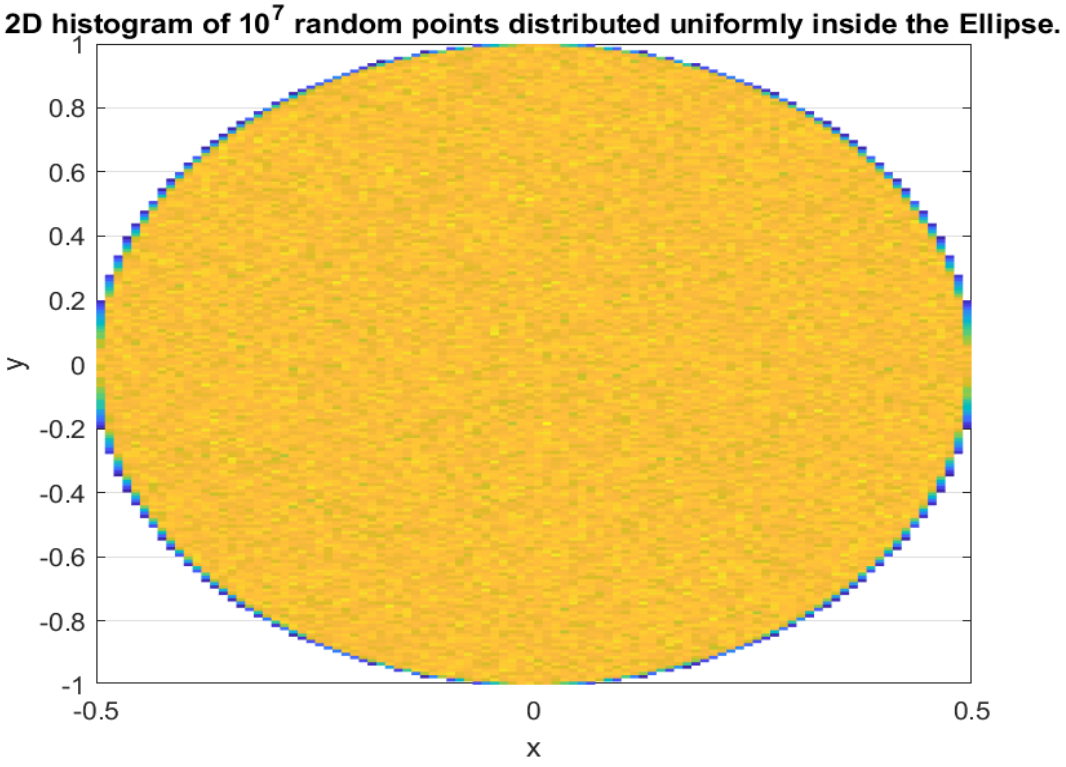


Figure 1: 2D-Histogram (with [0.01 0.01] bins) of Ellipse

Observation : Here we can see as the N value increases the Total number of random points increases and It covers almost all area inside the ellipse and we can clearly see the shape of the ellipse with a=0.5 and b=1

To Run : `one_b`

Results present as `one/one_b.png`

1.2 Generating Random points in Triangle

Algorithm for generating random points (in 2D) distributed uniformly inside the triangle:

Given one vertex(O) of the triangle is at the origin and Let the other vertex of the triangle are A and B. Construct a Parallelogram $OACB$. To pick points uniformly distributed inside the triangle, instead pick random points(x) inside the $OACB$.

$$\mathbf{x} = u\mathbf{a} + v\mathbf{b}$$

where u and v are uniform variates in the interval [0,1], which gives points uniformly distributed in $OACB$. The points not in the triangle OAB interior can transformed into the corresponding point inside the triangle by reflection along side AB.

In this problem: $\mathbf{x} = u(\pi, 0) + v(\pi/3, \exp(1))$
random point $\mathbf{x} = (\pi(u + v/3), v\exp(1))$
if $u + v > 1$ then the point is in the outside the triangle so we take reflection along side AB then the point is obtained by $u \to 1 - u$ and $v \to 1 - v$.

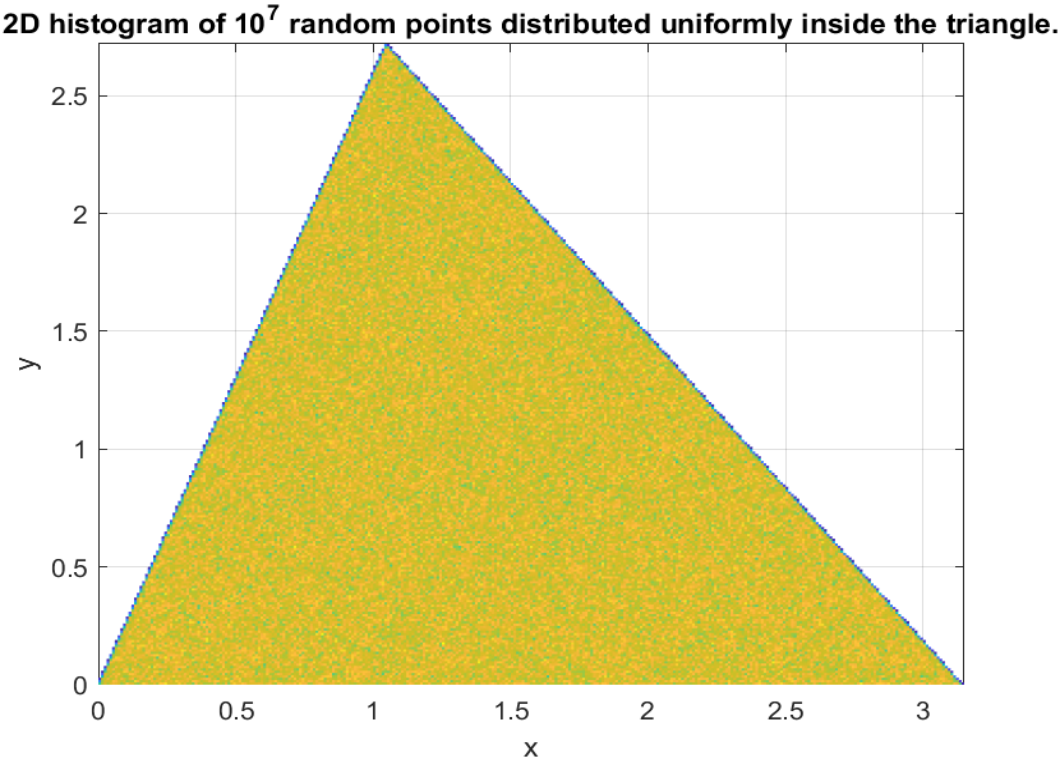


Figure 2: 2D-Histogram (with [0.01 0.01] bins)

Observation : Here we can see as the N value increases the Total number of random points increases and It covers almost all area inside the triangle and we can clearly see the shape of the triangle

2 Question 2

To Run : `two`

Results present as `two/covariance_error.png,mean_error.png,N=10^<i>.png(i=1 to 5)`

Using **randn()** function in **MATLAB** we can generate Generate a multivariate **Gaussian** random numbers. This would give us N numbers that are centered around zero and are independent of each other. we need to transform this into the Gaussian distribution described by the mean(μ) and given covariance matrix(C).
The Gaussian distribution we have at the moment is Distributed around a sphere. To move towards the Covariance matrix we want, we would need to squash this spherical distribution and to sync up with the mean we add μ The new distribution is

$$X = (\sqrt{l} \cdot V \cdot x) + \mu$$

Here, X is the transformed random numbers. l is the diagonal matrix made up of the eigenvalues of C and V is the matrix of eigen vectors (each column is an eigen vector of C).

The Maximum likelihood estimation for mean for Gaussian Distribution is given by

$$\hat{\mu} = \sum_{i=1}^{n} \frac{x_i}{n}$$

The Maximum likelihood estimation for Covariance Matrix for Gaussian Distribution is given by

$$\hat{C} = \frac{1}{N} \sum_{\alpha=1}^{n} (x_\alpha - \bar{x})(x_\alpha - \bar{x})'$$

Where here x_α is a vector (x_i, y_i) Repeated the experiment 100 times for each N, and found the error measure $||\mu - \hat{\mu}_N||_2 / ||\mu||_2$ and $||C - \hat{C}_N||_{Fro}|| / ||C||_{Fro}$ 100 times, then plotted the boxplot for all N = 10 to 10^5 using a $\log_{10}(N)$ scale

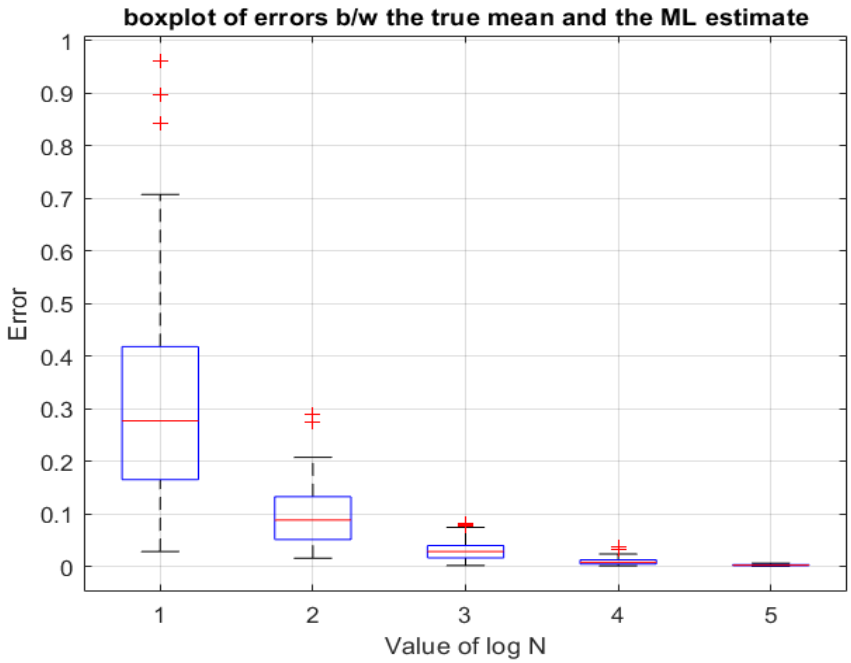


Figure 3: boxplot of the error between the true mean μ and the ML estimate $\hat{\mu}_N$

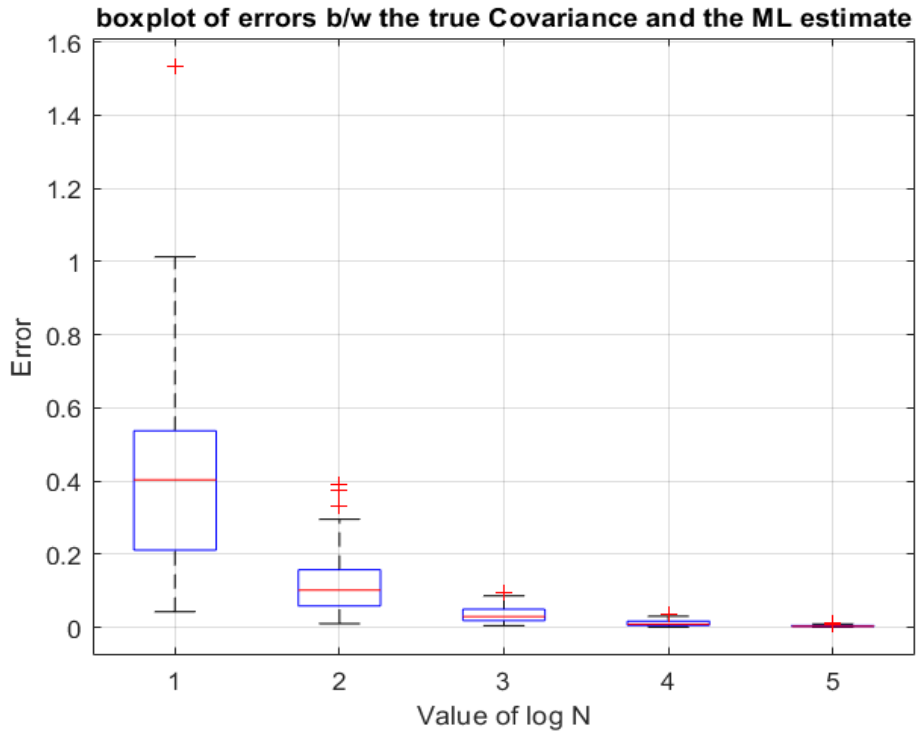
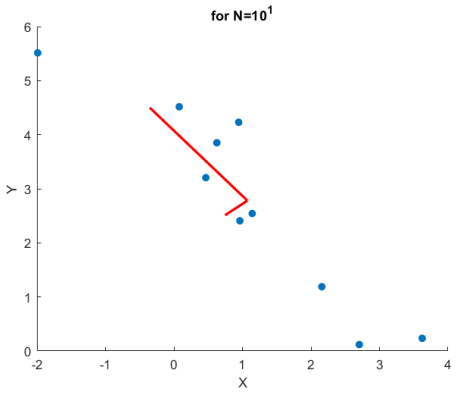
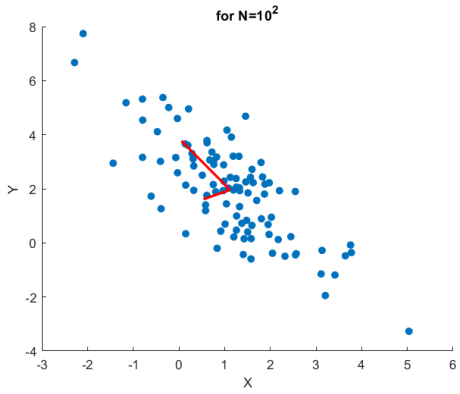


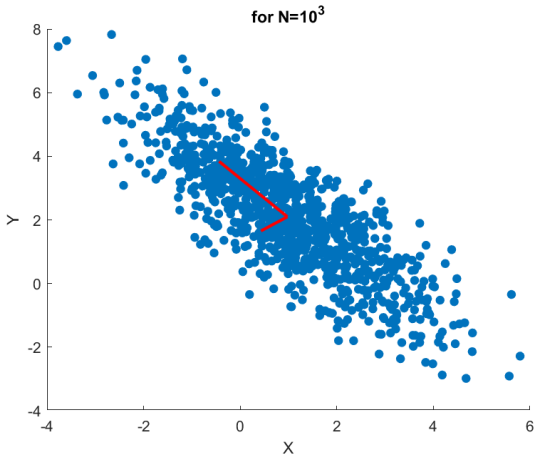
Figure 4: Boxplot of the error between the true covariance C μ and the ML estimate $\hat{C}_N$



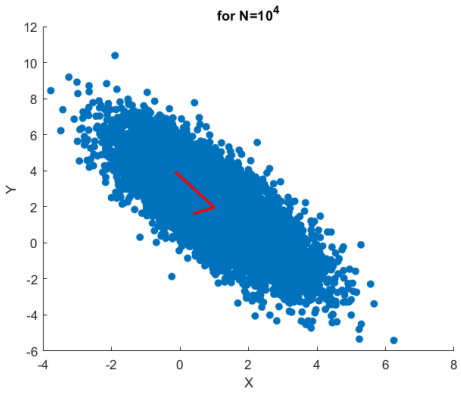
(a) scatter plot of the generated data for N=10



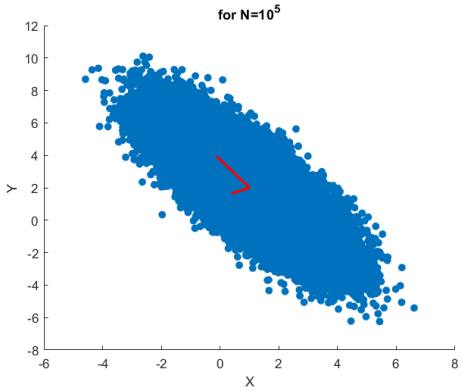
(b) scatter plot of the generated data for N=10^2



(a) scatter plot of the generated data for N=10^3



(b) scatter plot of the generated data for N=10^4



(a) scatter plot of the generated data for N=10^5

Observation : In the Boxplot of Errors of both ML estimate of mean and covariance from there true values, we can see that the size(length) of the box is decreases as N increases and when N reaches very Large value like 10^5 we can observe that the box has almost zero size and the error is zero. So to get Mean value or covariance value from the data by using ML estimations very close to their true values we have to take Very Large N values to diminish the Error which is almost Zero.

3 Question 3

To Run : `three`

Results present as `three/three_a.png,three_b.png`

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss.

For given Random variables X and Y we have to do PCA, for this first we have to standerdize the given matrix by this formula

$$x_{new} = \frac{x - \mu}{\sigma}$$

Here μ and σ are mean and standard deviation for each feature. For this matrix we have to find covariance matrix(C). In this case Covariance Matrix is given by

$$C = \begin{bmatrix} Cov(X,X) & Cov(X,Y) \\ Cov(Y,X) & Cov(Y,Y) \end{bmatrix} \tag{1}$$

Where

$$Cov(X,Y) = \frac{\sum(x_i - \bar{x}) * (y_i - \bar{y})}{N}$$

After finding covariance matrix we found eigen values and eigen vector and taken maximum eigen value corresponding eigen vector as principal component. Now, Projected the data in the direction of principal component and obtained the linear relation ship between random variables X and Y. std-data*eigen'*eigen added to mean gives us the required relationship between the Random variables X and Y.

We have plotted two plots showing the scatter plot and line overlayed.

Plot of Data-Set1 :

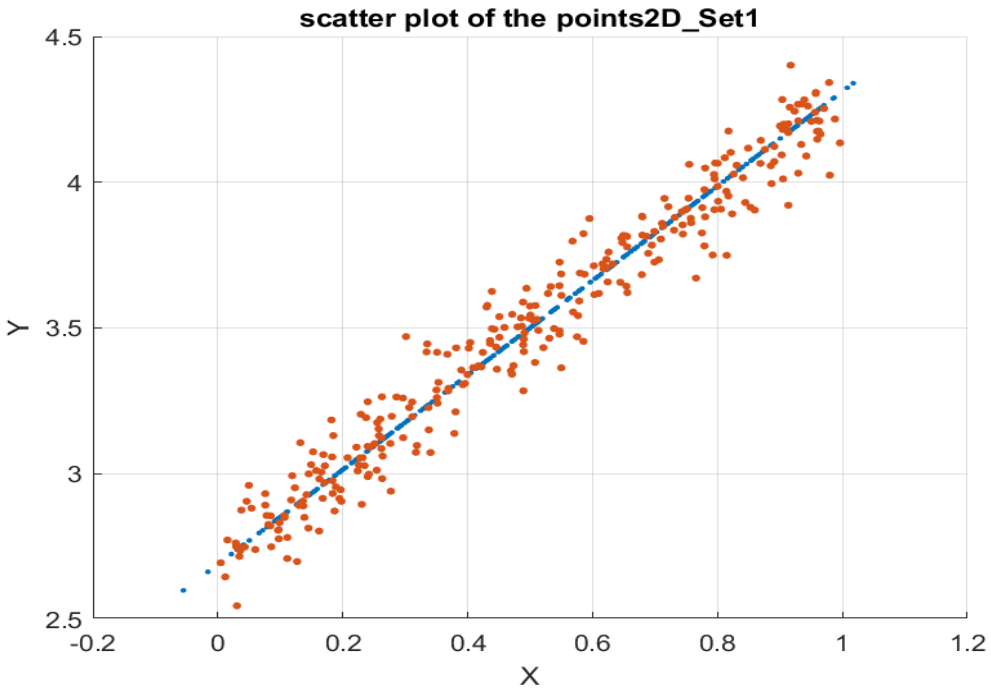


Figure 8: scatter plot of the points2D_set1

Plot of Data-Set2:

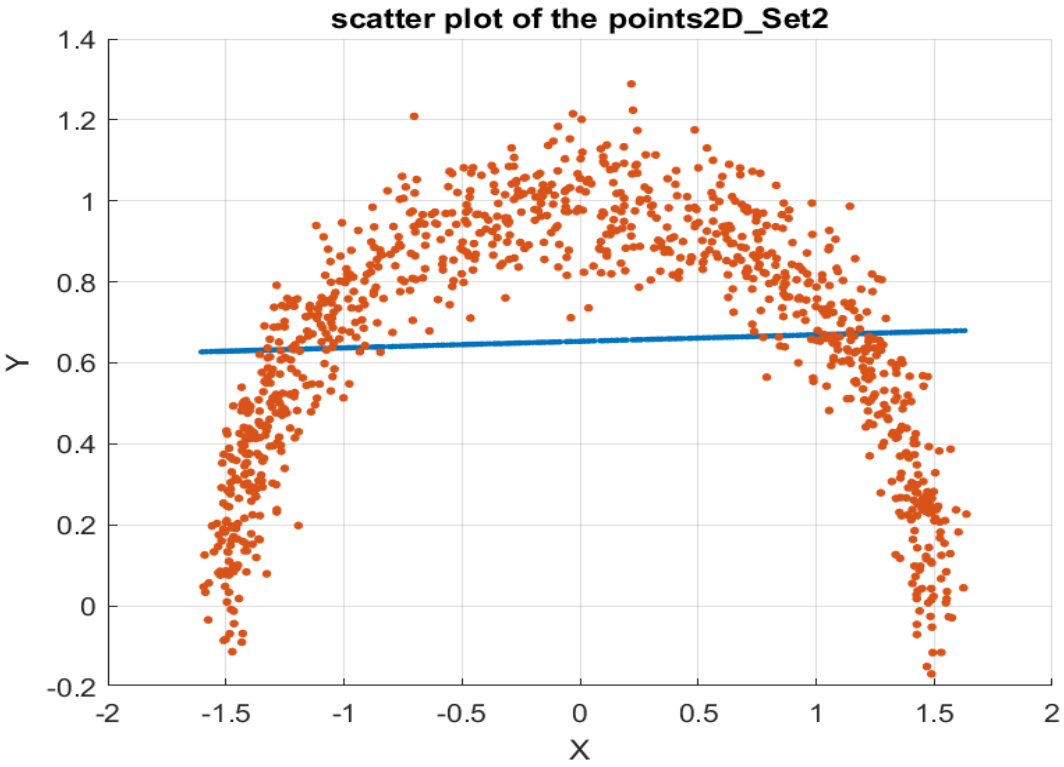


Figure 9: scatter plot of the points2D_set2.

From above two figures we can conclude that our method of PCA is giving good approximation in case1 than case2. It is due to the linearity in data1 whereas data2 is non-linear and we cannot really get a good linear approximation as original data itself is non-linear.

The plot for data set 2: There is nearly no correlation between the 2 random variables. For almost half the dataset, Y increases with increase in X, and for the remaining part, Y decreases with increase in Y. Wheras in case of data set 1 there is positive correleation between X and Y.So application of PCA in order to find a linear relationship between X and Y is more useful for dataset 1.

Covaraince Matrix in case of dataset1:

$$C_1 = \begin{bmatrix} 0.0824 & 0.1295 \\ 0.1295 & 0.2127 \end{bmatrix}$$

Covaraince Matrix in dataset2:

$$C_2 = \begin{bmatrix} 1.1034 & 0.0163 \\ 0.0163 & 0.0979 \end{bmatrix}$$

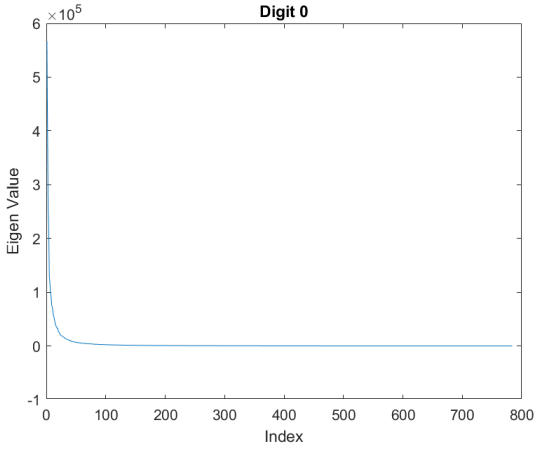
4 Question 4

4.1 Part a

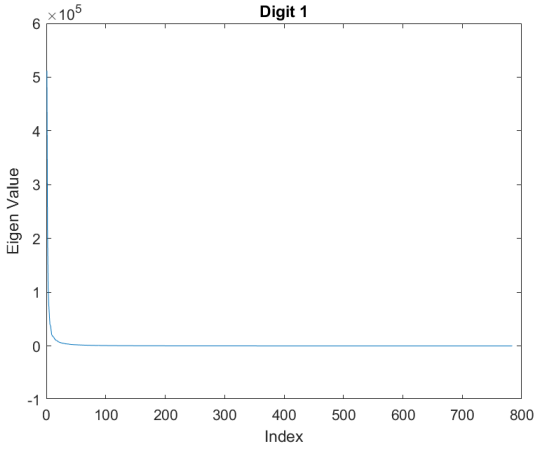
To Run : `four_a`

Results present as `four/Digit_<i>.png (i = 0 to 9)`

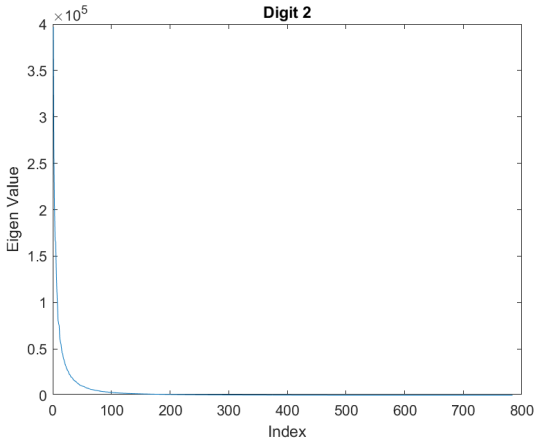
For each digit, we calculated eigen values using eig() function in Matlab and then stored all those eigen values and mapped them to corresponding eigrn vectors, after sorting in descending order and plot those eigen values. So the plots for all digits are shown below



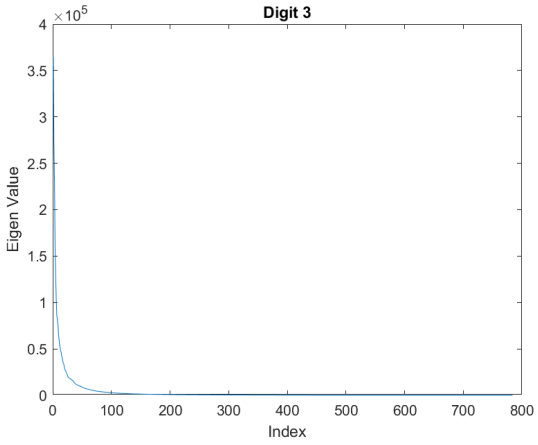
(a) Eigen Values plot for digit 0



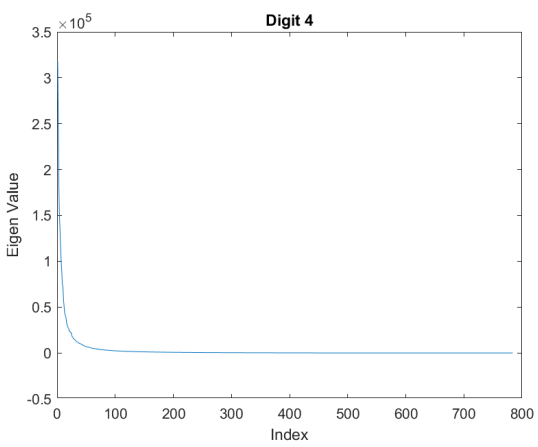
(b) Eigen Values plot for digit 1



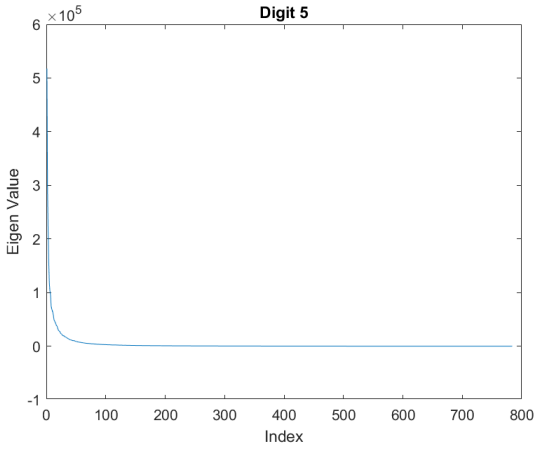
(a) Eigen Values plot for digit 2



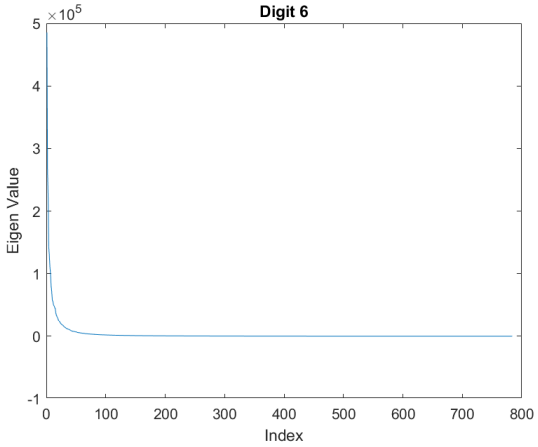
(b) Eigen Values plot for digit 3



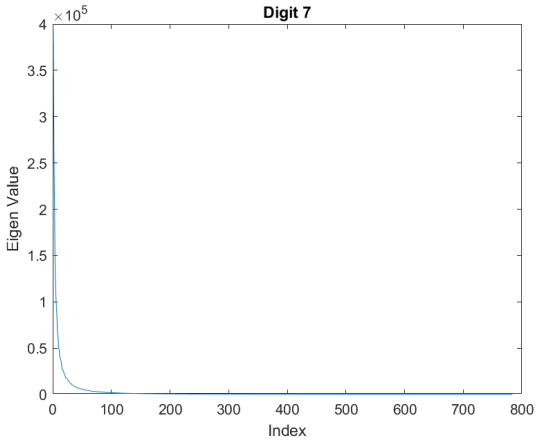
(a) Eigen Values plot for digit 4



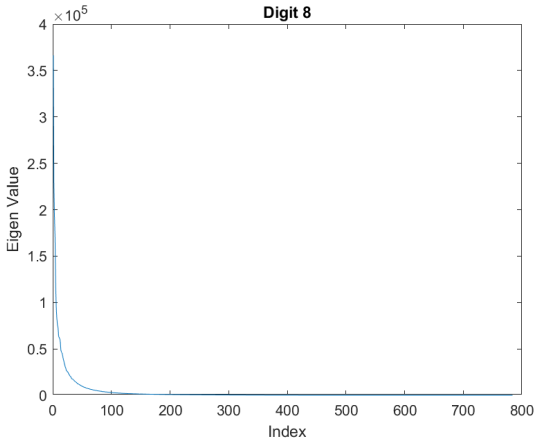
(b) Eigen Values plot for digit 5



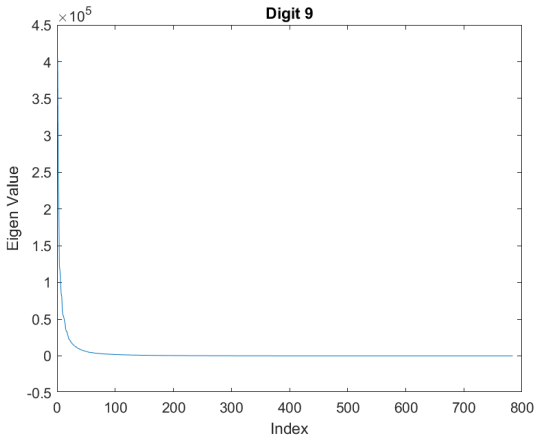
(a) Eigen Values plot for digit 6



(b) Eigen Values plot for digit 7



(a) Eigen Values plot for digit 8



(b) Eigen Values plot for digit 9

Observations : From above graphs we can conclude that number of principal/significant modes of variation are far less than 28^2. If we observe sorted eigen values data the decrease in eigen values is rapid and around 50-80th index we have a drop of order by 2. So eigen of magnitude in oder 10^5 are very less (5-10). Hence number of principal/significant modes of variation are far less than 28^2.On an average the number of eigen values that are significant compared to maximum eigen values is around **70**

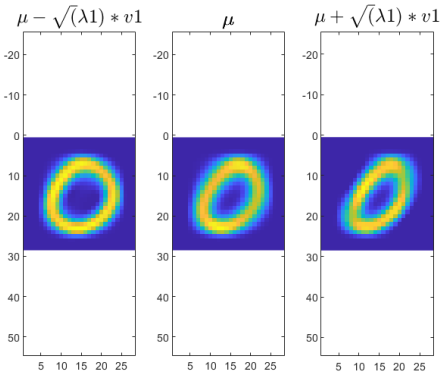
Eigenvalues are the variance of principal components. Here most of the eigen values are very low compared to highest value, that suggests there is little to no variance in the matrix, which means there are chances of high collinearity in data. Let us take 1 as example, first to notice all the pixels are not covered in writing 1 in different ways. So most of the uncovered pixels will have 0 variance and also for some covered pixels they are almost same with some difference. Therefore we can see rapid decrease in eigen values in graph.

4.2 Part b

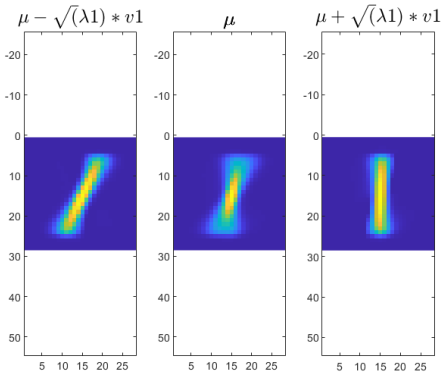
To Run : `four_b`

Results present as `four/<i>.png (i = 0 to 9)`

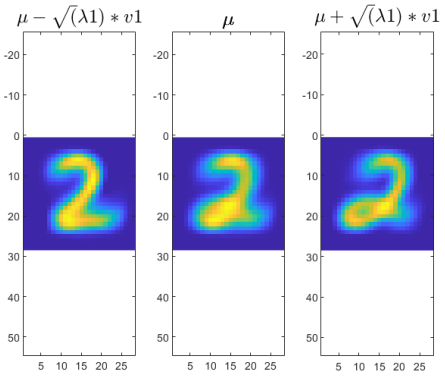
We observed that the images present on either sides are slightly inclined to the mean image,and the image left to the mean image that is the image corresponding to $\mu + \lambda v1$ of digit 1 from the given data set is the way how people write the digit 1.



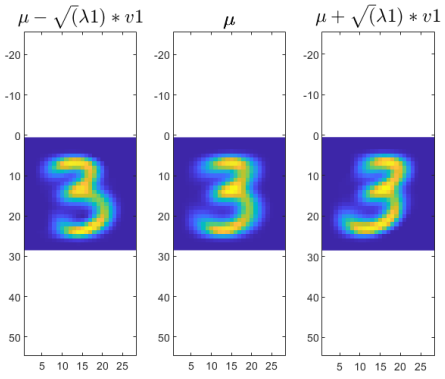
(a) Principal mode of variation of 0 around mean



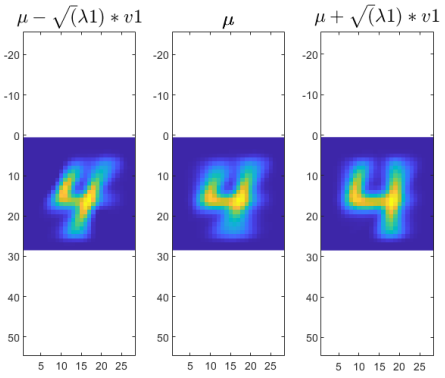
(b) Principal mode of variation of 1 around mean



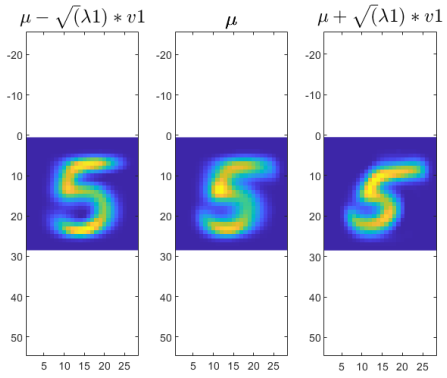
(a) Principal mode of variation of 2 around mean



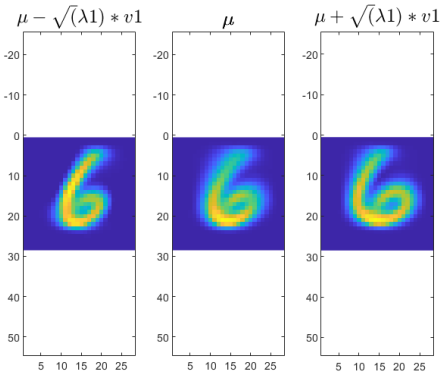
(b) Principal mode of variation of 3 around mean



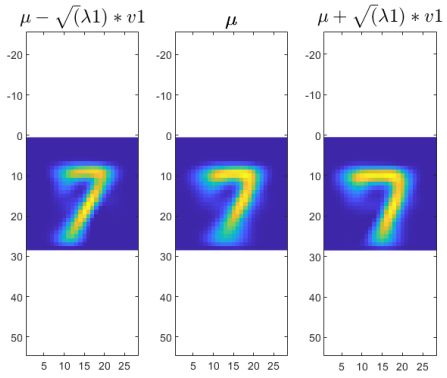
(a) Principal mode of variation of 4 around mean



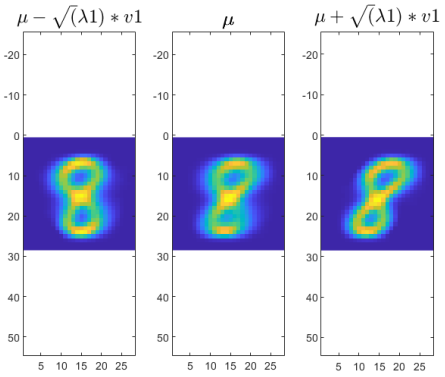
(b) Principal mode of variation of 5 around mean



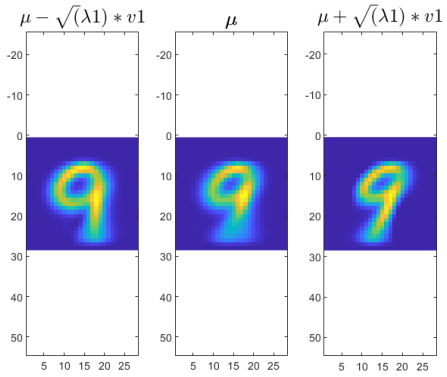
(a) Principal mode of variation of 6 around mean



(b) Principal mode of variation of 7 around mean



(a) Principal mode of variation of 8 around mean



(b) Principal mode of variation of 9 around mean

In this we have shown three images of all numbers side by side in order $\mu - \sqrt{\lambda}v1$, μ and $\mu + \sqrt{\lambda}v1$. We know that $\mu \pm \sqrt{\lambda}v1$ these two images corresponds to principal mode of variation which has more contribution of eigen vector corresponding to maximum eigen value.

Observation: We can see in all cases μ one picture is less clear. This is beacuse it has equal contributions from all eigen vectors where as other two have more contribution from V_1.

5 Question 5

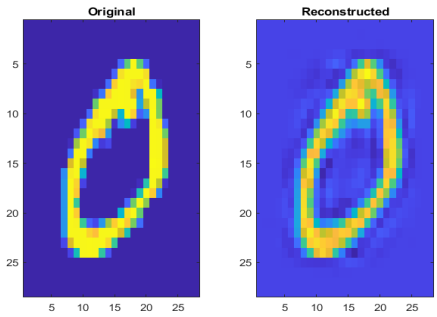
To Run : `five`

Results present as `five/five_<i>.png (i = 0 to 9)`

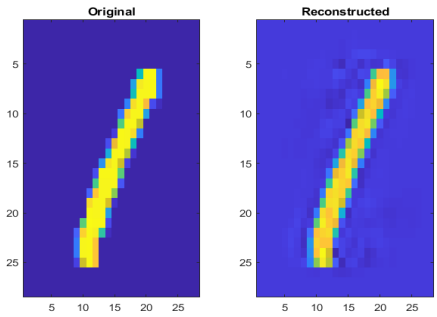
As of now, for each digit, each 28*28 pixel image is represented using 28^2 coordinate values in the Euclidean space of dimension 28^2. We are now re-representing the images using only 84 coordinates in a 84-dimensional basis for some 84-dimensional hyperplane within the original Euclidean space, such that the chosen 84-dimensional hyperplane maximizes the total dispersion of the original data (for the chosen digit) within the hyperplane.

Key idea points involved in this algorithm :

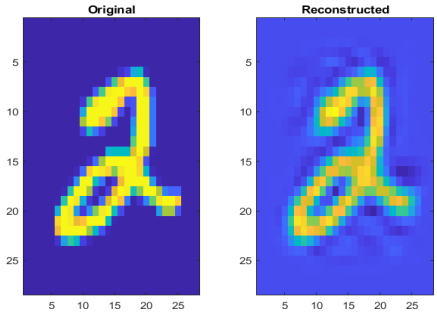
- We will make all our images zero centered, subtracting the average image from each image in the matrix for zero centering.
- Now, after making the images zero-centered, we have calculated the covariance matrix.
- Now let us obtain eigen values and eigen vectors.
- Sort the eigen values and map to corresponding eigen vectors.
- Take first 84 components of sorted eigen vectors. These 84 components are the Principal Components.
- These 84 images will be called Eigenfaces, and one can represent any image as a linear combination of these 84 images.
- Now reduce dimension to 84 by doing dot product `eigen*original*eigen'`.



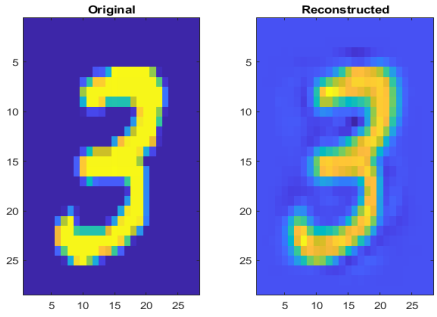
(a) Original and the Reconstructed for 0



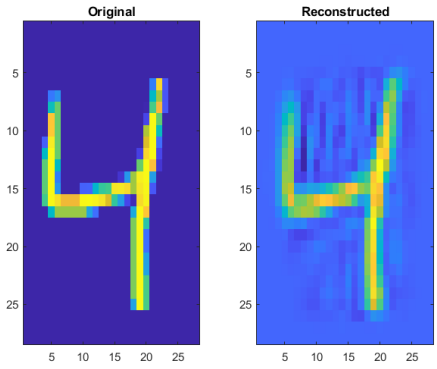
(b) Original and the Reconstructed for 1



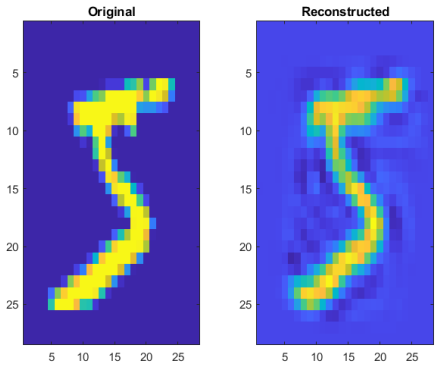
(a) Original and the Reconstructed for 2



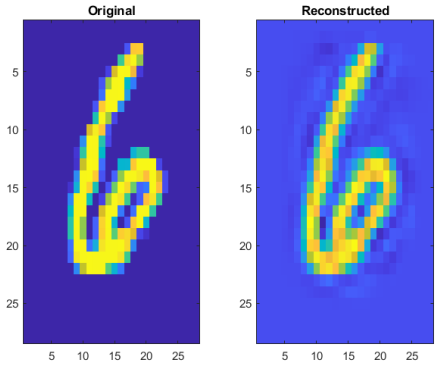
(b) Original and the Reconstructed for 3



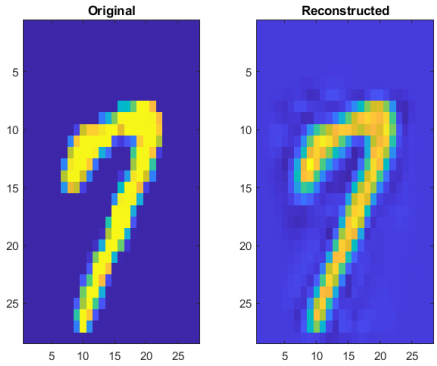
(a) Original and the Reconstructed for 4



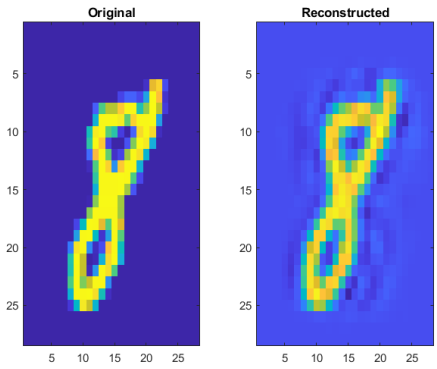
(b) Original and the Reconstructed for 5



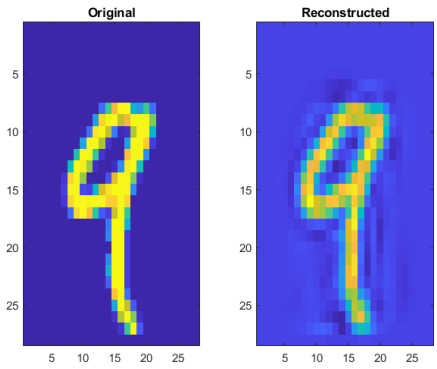
(a) Original and the Reconstructed for 6



(b) Original and the Reconstructed for 7



(a) Original and the Reconstructed for 8



(b) Original and the Reconstructed for 9

6 Question 6

6.1 Part a

To Run : `six_a`

Results present as `six/6_a_EigenValues.png,6_a_fruits.png`

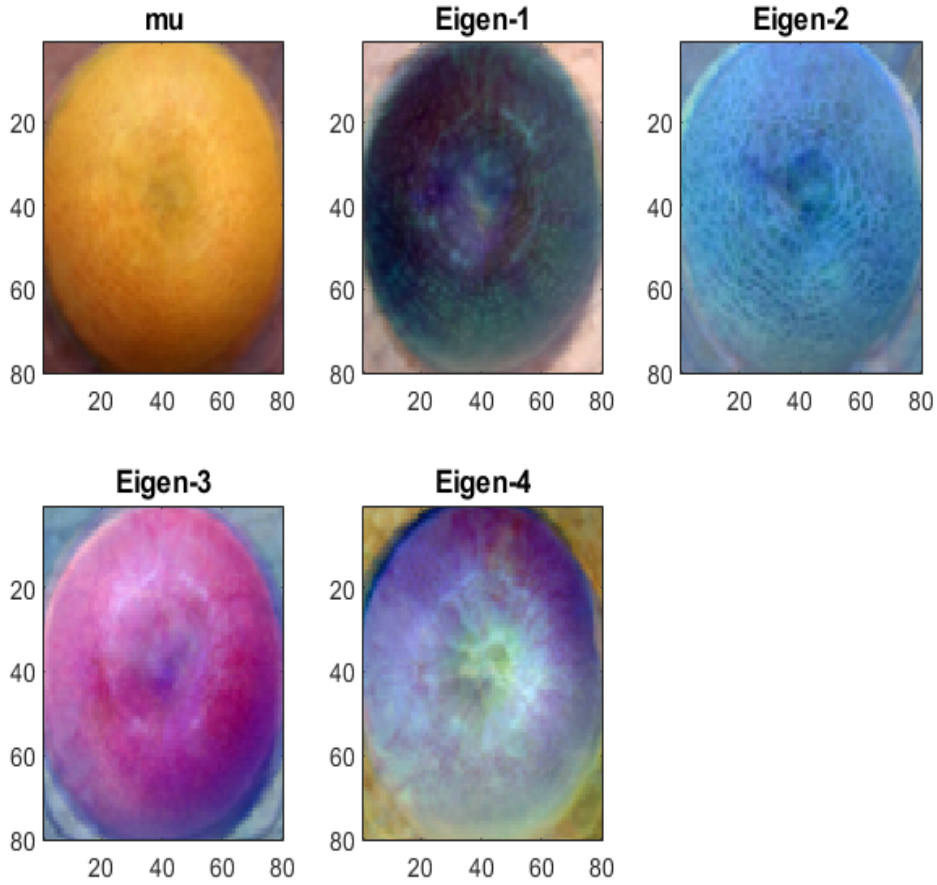


Figure 25: Mean and the Eigenvectors as images

6.2 Part b

To Run : `six_b`

Results present as `six/6_b_<i>.png (i = 1 to 16)`

Frobenius norm :

$||A||_n$ denotes the Frobenius norm of an n×n matrix A = $(a_{ij})_{i,j=1}^{n}$ and is given by

$$\left(\sum_{i,j=1}^{n} ||a_{ij}|| \right)^{\frac{1}{2}}$$

On changing matrix to linear vector doesnot effect Frobenius norm value. Let M be the original image vector and N be the new closest representation of that image Vector.

Let V1, V2, V3, V4 be the top 4 principal eigen vectors. N will be a linear combination of this and mean. We also know that original image is sum of linea r combination of all eigen vectors. Let,

$$N = a1 * \bar{mu} + a2 * \bar{V1} + a3 * \bar{V2} + a4 * \bar{V3} + a5 * \bar{V4}$$
$$\bar{mu} = \sum_{i=1}^{19200} ui * \bar{V_i}$$
$$N = (a1 * u1 + a2)\bar{V1} + (a1 * u2 + a3)\bar{V2} + (a1 * u3 + a4)\bar{V3} + (a1 * u4 + a5)\bar{V5}$$
$$M = \sum_{i=1}^{19200} m_i * V_i$$
$$||M - N||^2 = \sum_{i=1}^{4}[(m_i - a_{i+1} - a1 * u_i)]^2 + \sum_{i=5}^{19200}[(m_i - a1 * u_i)]^2$$

Now we have a1, a2, a3, a4, a5 as variables to make norm minimun we can make first four terms as zero since we have five variables let us take now only a1 as variable.

Making first 4 terms as zeros gives us :

$$a2 = m_1 - a1u1$$
$$a3 = m_2 - a2u2$$
$$a4 = m_3 - a3u3$$
$$a5 = m_4 - a4u4$$

We are now left with

$$\sum_{i=5}^{19200} [m_i - u1 * a1]^2$$

Where a1 is a variable, now we have to minimize it for that we can take partial derivative with respect to a1 and find a1.

$$\sum_{i=5}^{19200} 2 * (m_i - u_i * a1)u_i = 0$$
$$\sum_{i=5}^{19200} (m_i * u_i) = a1 * \left(\sum_{i=5}^{19200} u_i^2\right)$$

We also know that,

$$\sum_{i=1}^{19200} u_i^2 = \bar{mu}.\bar{mu}$$
$$\sum_{i=1}^{19200} (m_i * u_i) = \bar{M}.\bar{mu}$$

So,

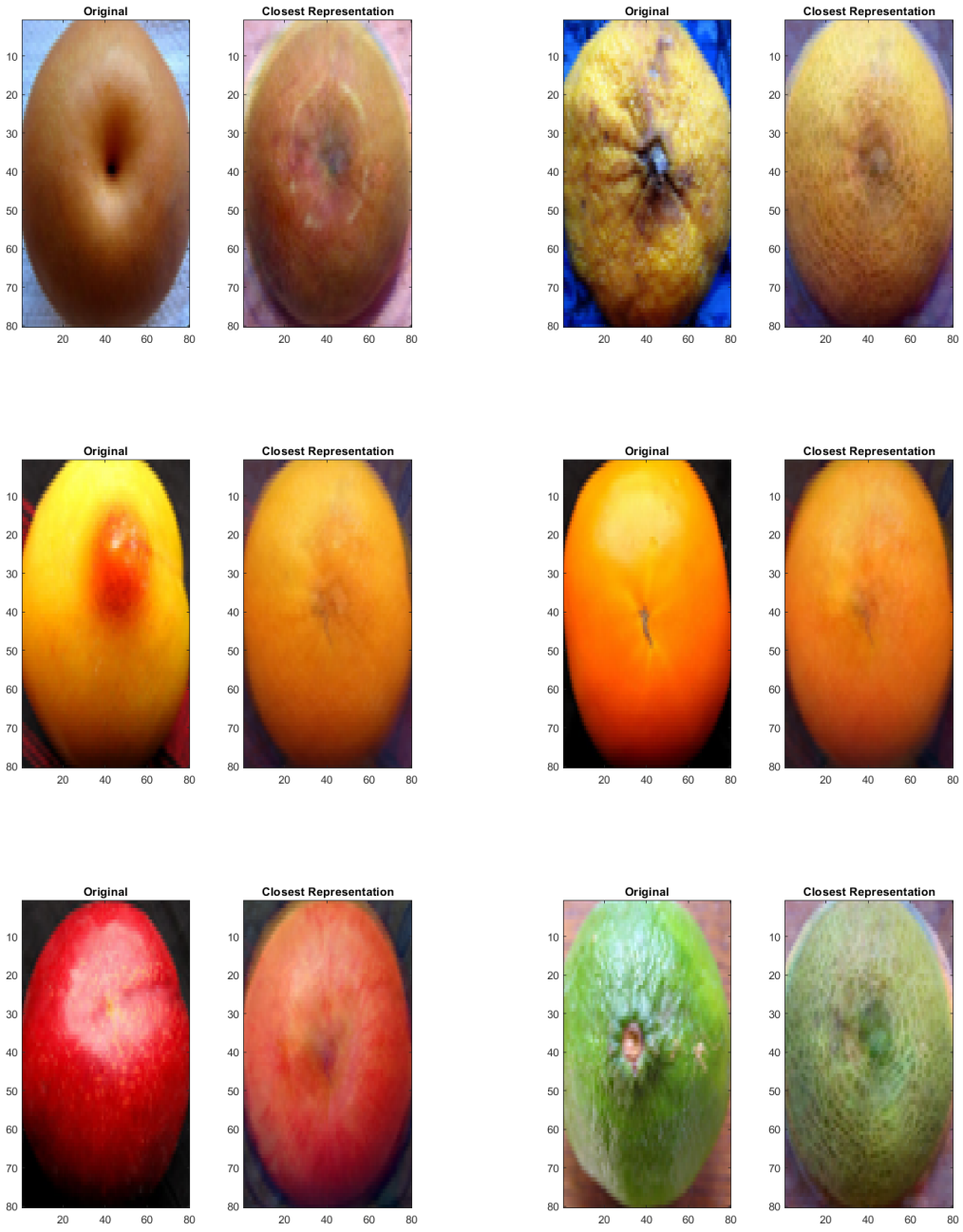
$$a_i = \frac{\bar{M}.\bar{mu} - \sum_{i=1} 4(m_i * u_i)}{\bar{mu}.\bar{mu} - \sum_{i=1}^{4} \sum_{i=1} 4u_i^2}$$

$$a2 = m_1 - a1u1$$
$$a3 = m_2 - a2u2$$
$$a4 = m_3 - a3u3$$
$$a5 = m_4 - a4u4$$

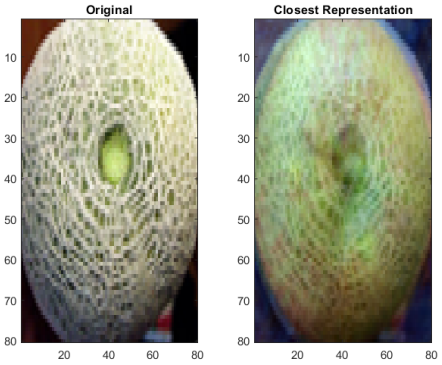
Now,

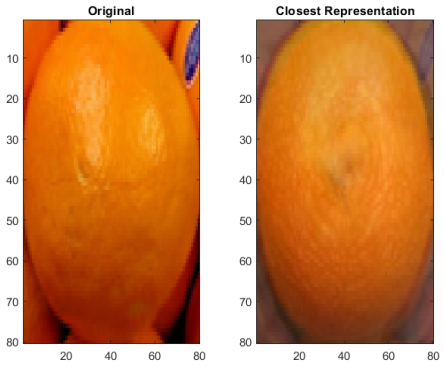
$$N = a1 * \bar{mu} + a2 * \bar{V1} + a3 * \bar{V2} + a4 * \bar{V3} + a5 * \bar{V4}$$

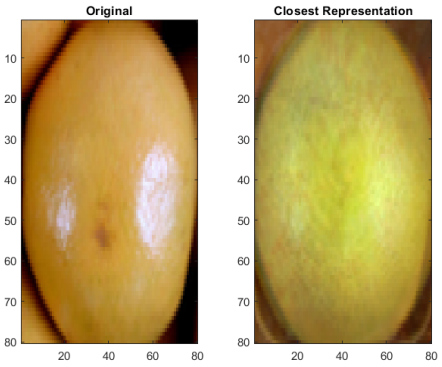
Where a1, a2, a3, a4, a5 are produced by above description and N is the newly generated closest vector representation of M.

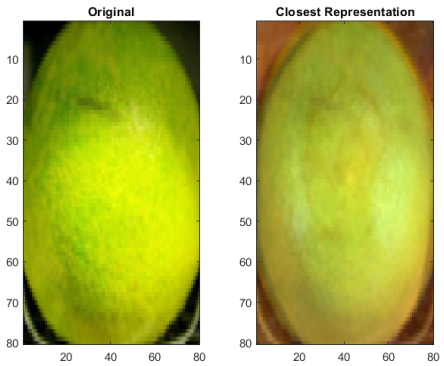


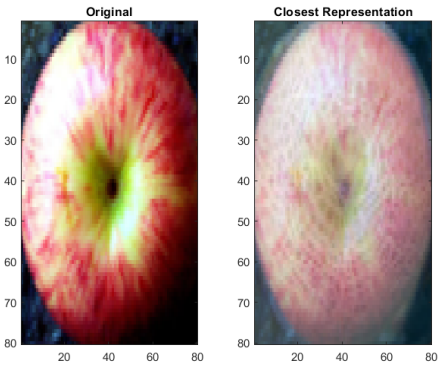

Original
Closest Representation

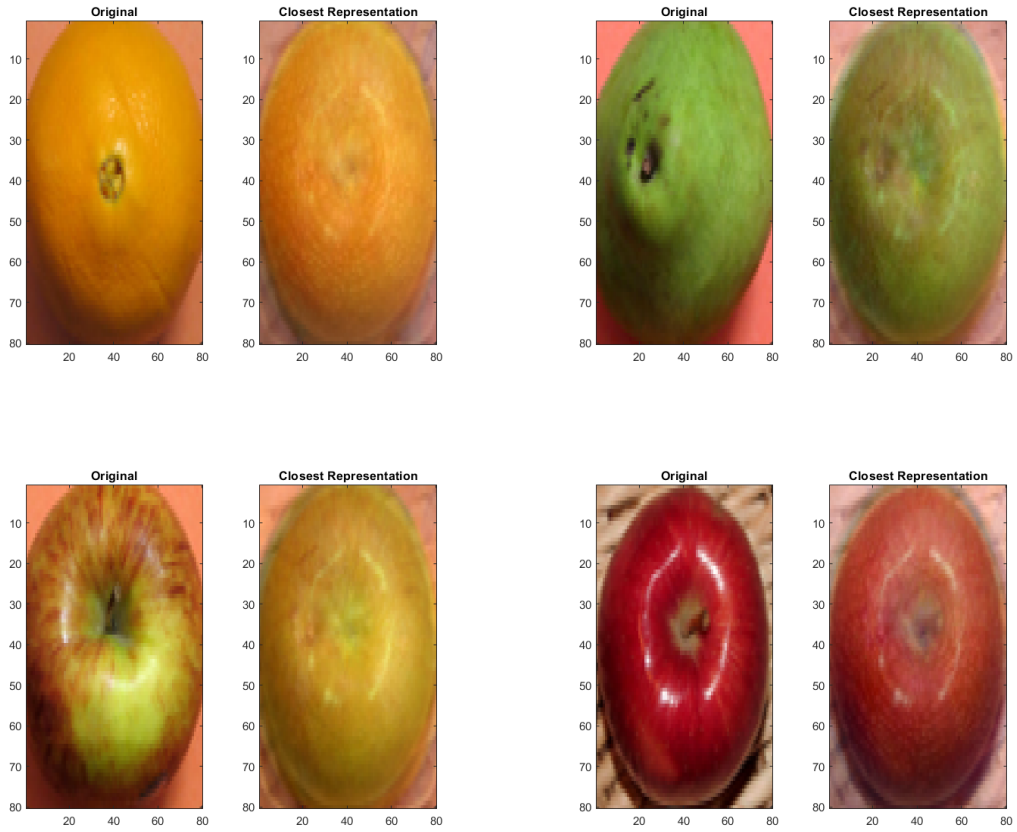

Original
Closest Representation


Original
Closest Representation


Original
Closest Representation


Original
Closest Representation


Original
Closest Representation



The value of **Frobenius norm** of difference of matrices obtained is around **4500** when taken in pixel values. If divided by 255 it gives around **18**, which can be taken of relative error.

6.3 Part c

To Run : `six_c`

Results present as `six/6_c_fruits.png`

For this question we have generated a MVG with the help mean mu from data and with the help of top 4 principal eigen vectors.

We know that, If

$$\Sigma = UDU' = UD^{1/2}(UD^{1/2})T$$

is an eigendecomposition where the columns of U are unit eigenvectors and D is a diagonal matrix of the eigenvalues, then we have

$$\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$$
$$\mathbf{X} \sim \mu + UD^{1/2}\mathcal{N}(0, I)$$

After obtaining X we rescaled, reshaped and plotted three such generated images.

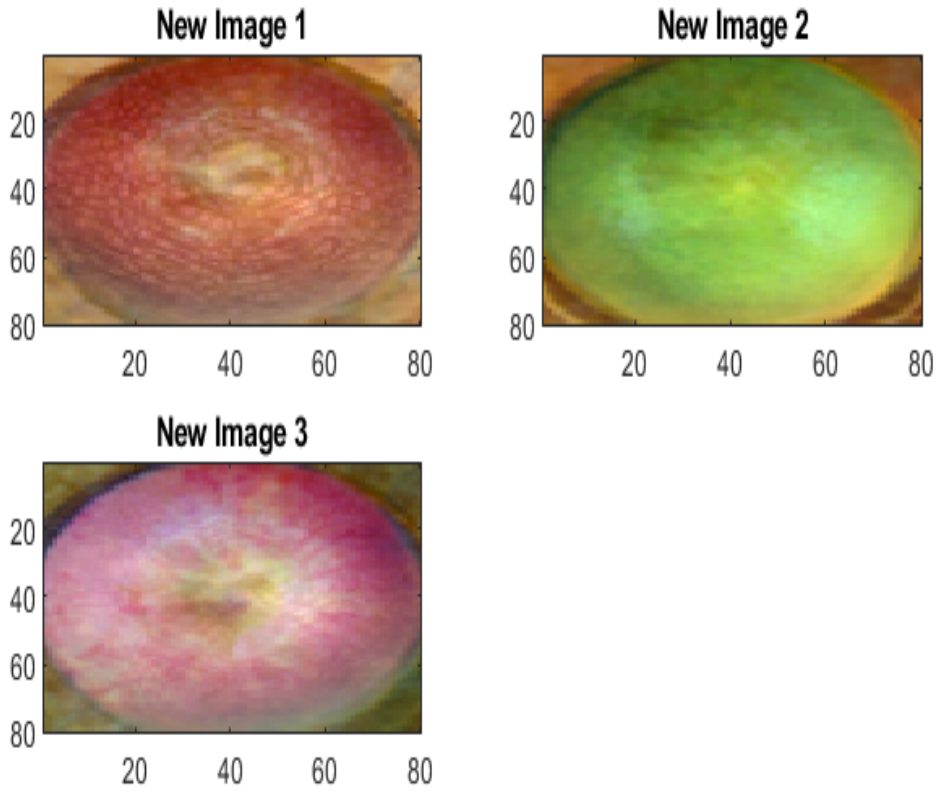


Figure 34: Newly generated fruits