PROJECT

ECE 6337 | FALL 2022

SUBMITTED BY: SHASHANK BHUSHAN

PS ID: 2093536

MAIL ID: sbhusha3@cougarnet.uh.edu

Problem 1 Solutions:

We were given task to generate multi-variate Gaussian data based random vectors using the mvnrnd() function of Statistics and Machine Learning Toolbox in MATLAB. We were provided with input parameters like mean vector, covariance matrix and number of samples as 1000 that are to be generated.

So, for first part 1(a) random 1000 samples from multi-variate Gaussian data were generated using provided mean vector, covariance matrix and number of samples as 1000 and the 3D plot was generated using scatter3() function in MATLAB as shown in the below figure 1. Regarding concluding about the shape of the data point cloud we can see it in 3D plane as more of a spherical shape with most of the data points centred as we are at centre point (0,0,0) and then as we move away from the centre point the intensity of data points in the data point cloud becomes lesser and lesser.

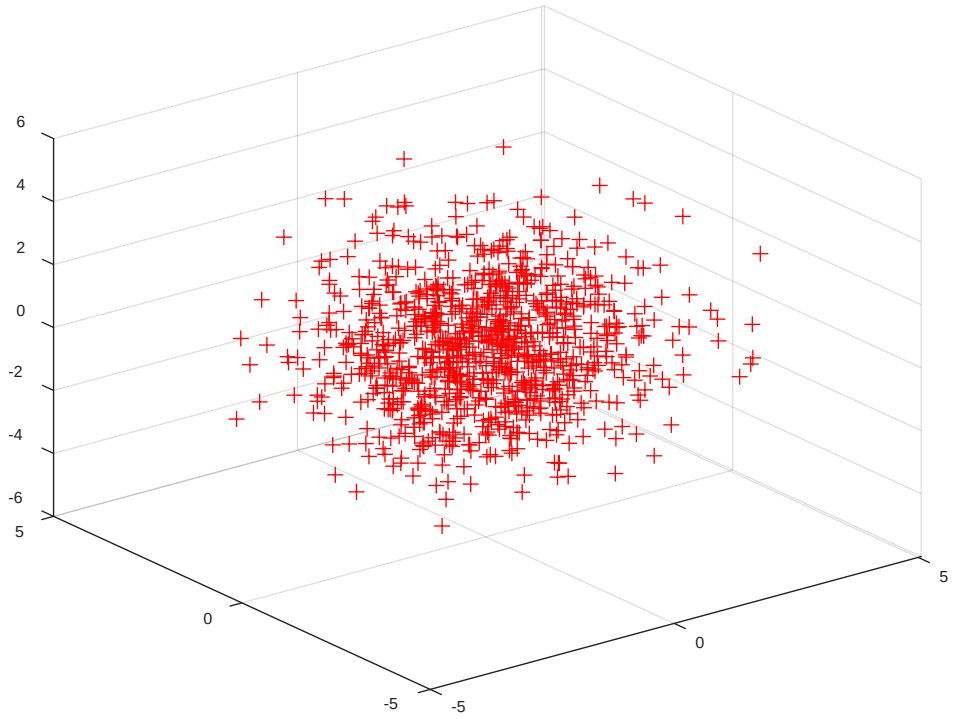

Fig 1. Data Point Cloud for 1(a)

For second part 1(b) similar random 1000 samples from multi-variate Gaussian data were generated using provided second mean vector, second covariance matrix and number of samples as 1000 and the 3D plot was generated using scatter3() function in MATLAB as shown in the below figure 2. Regarding concluding about the shape of the data point cloud we can see it in 3D plane as more of a cylindrical or oval or ellipsoidal shape with most of the data points centred around eccentric line of

this shape and then as we move away from the eccentric line the intensity of data points in the data point cloud becomes lesser and lesser.

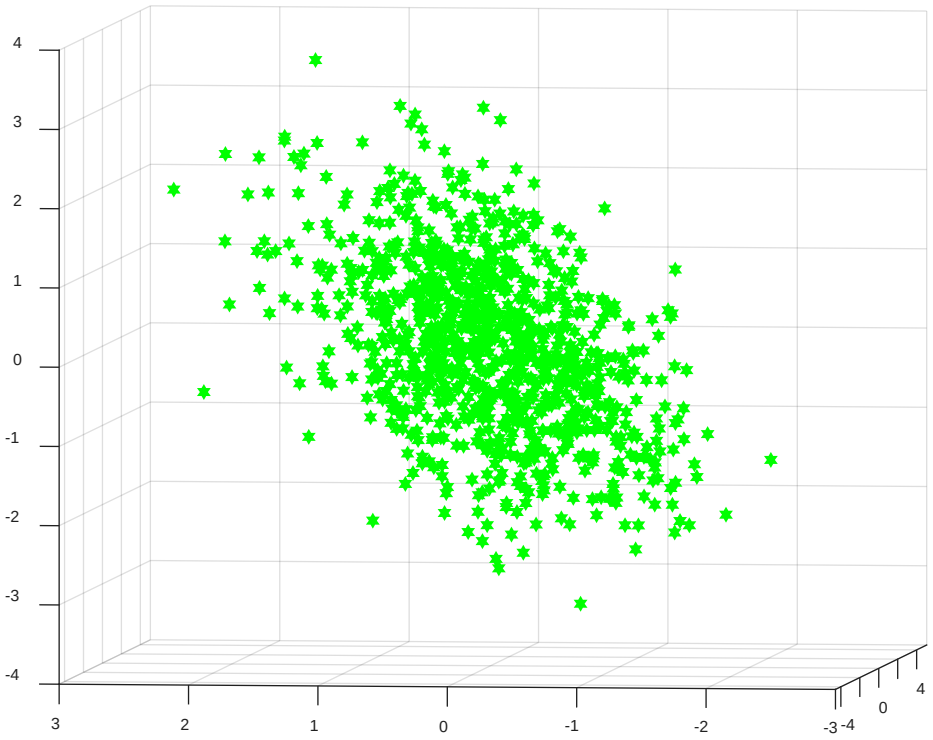

Fig 2. Data Point Cloud for 1(b)

Talking about mvnrnd() function, it gives a matrix R of n random vectors with mean vector mu and covariance matrix Sigma, all drawn from the same multivariate normal distribution. It is part of the package Statistics and Machine Learning Toolbox in MATLAB.

It is based on Multivariate Normal Distribution concept. In this case, univariate normal distribution is extended to two or more variables by the multivariate normal distribution. It has two parameters that are comparable to the mean and variance parameters of a univariate normal distribution: a mean vector and a covariance matrix. The variances for each variable are contained in the diagonal components of the covariance matrix, whereas the covariances between variables are contained in the off-diagonal elements.

The d-dimensional multivariate normal distribution's (pdf) probability density function is:

$$y = f(x, \mu, \Sigma) = \frac{1}{\sqrt{|\Sigma|(2\pi)^d}} \exp\left(-\frac{1}{2}(x\text{-}\mu)\,\Sigma^{-1}(x\text{-}\mu)'\right)$$

here x and μ are 1 x d vectors and Σ is a d x d symmetric, positive definite matrix.

Problem 2 Solutions:

This problem was a continuation of Problem 1 part a and part b. Where we used the random gaussian data generated using mvnrnd() function to feed into our custom built function as an input.

In this problem we were tasked with implementing our own function to carry out projection of

random vectors onto principal components using concepts we learned in class. We were asked to implement our own custom function ProjectData.m using the guidelines as provided.

This custom function took its input as data matrix X comprised of 1000 data samples, each as a point (vector) in the d- dimensional Euclidean space. The output of our function was needed to be another data matrix Y. As a first step inside the function we fed the input data matrix X to inbuilt covariance function cov() to find the covariance matrix covarMAT. Moving further we fed this generated covariance matrix to perform the eigen value decomposition using in-built function eig() of MATLAB to get eigen vector as P and eigen matrix as eigMAT. To ensure that our first eigen vector corresponds to the largest value and rest of the following eigen vector follow similar trend we applied in-built sort() function to eigMAT matrix keeping sorting order as 'descending'. This was then further stored in the same eigMAT matrix. Projection matrix TrPoseMAT was then obtained by selecting the first 2 columns of the eigenvector matrix that resulted in a transformation matrix TrPoseMAT. This TrPoseMAT was finally used to project data in the d-dimensional Euclidean space to a lower d2 dimensional subspace by performing a simple matrix operation as Y=TrPoseMAT.'*X.' to get the transpose of a matrix.

Once the function construction got completed, the next part was to utilize the custom-built function ProjectData to apply the transformation and project our synthetic random gaussian data that was generated in part 1(a) and part 1(b). The 2D scatter plots for data from part 1(a) and 1(b) are as follows:

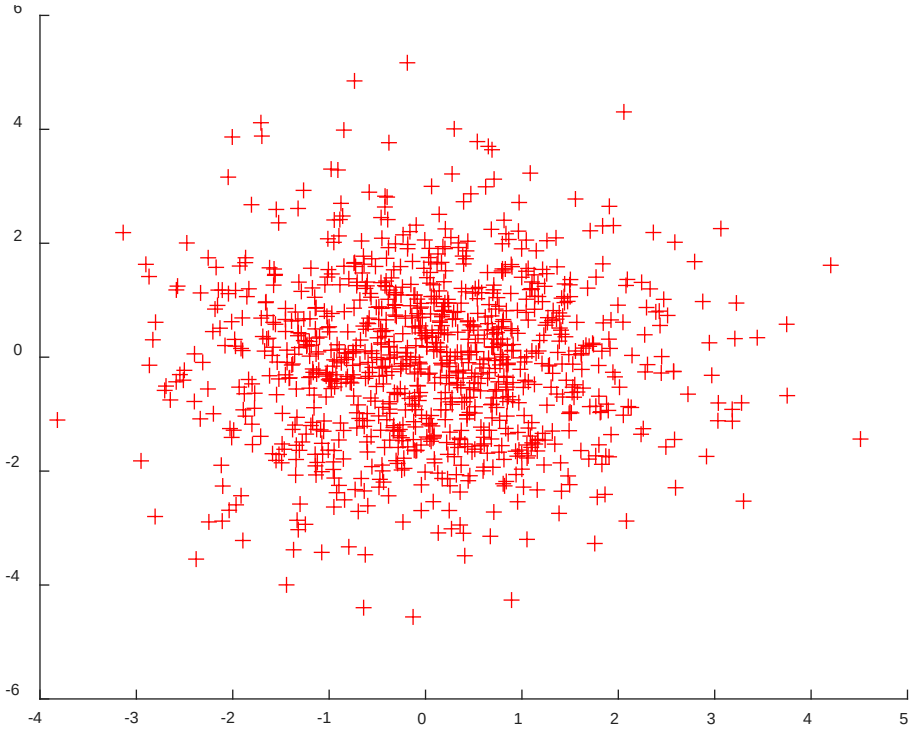

Fig 3. Scatter Plot generated for transformation of synthetic gaussian data from 1(a)

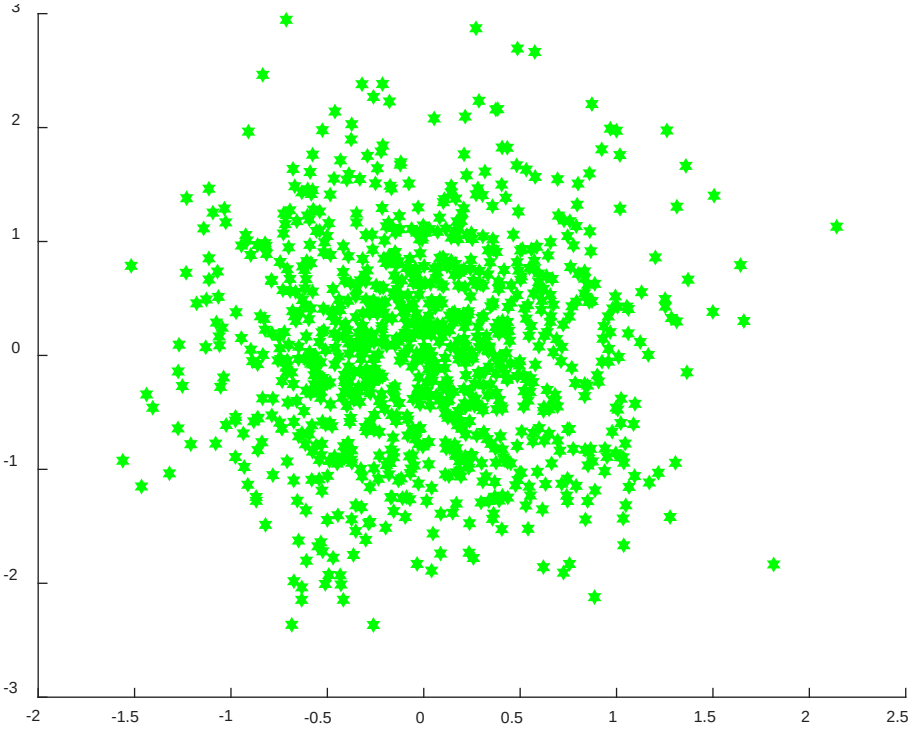

Fig 4. Scatter Plot generated for transformation of synthetic gaussian data from 1(b)

Also, the covariance matrix of projected data Y for 1(a) was calculated as:

```
Y1_cov = 2×2
    1.4851   -0.0000
   -0.0000    1.9899
```

The covariance matrix of projected data Y for 1(b) was calculated as:

```
Y2_cov = 2×2
    0.3186   -0.0000
   -0.0000    0.7878
```

On comparing the covariance matrix and scatter plots of the original data X and projected data Y it can be concluded that while we can control the data spread in 3D for original data X by controlling the mean and variance parameters in mvnrnd() function. The calculation of covariance and applying eigen value decomposition allows us to get more specific details about the sparseness of our data in a two-dimensional plane.

Problem 3 Solutions:

This problem required us to get a basic observation of how concept of random variables gets utilized in the application of digital image processing techniques like histogram equalization.

We were informed about the basics of Histogram Equalization and its corelation with random processes that is used to process contrast settings of images having a poor contrast. For this we considered intensity of pixels in images as random variables X. The histogram of intensity values would look like a rough estimate of the pdf of intensity random variables. In histogram equalization

our main target is to attain transformation of random variable X which in this case are pixel of original image to a new random variable Y using the below equation:

$$Y = g(X)$$

where g(X) is the transformation function so that after the transformation on our random variable X gets completed, the new random variable Y of the transformed image results in a histogram or a pdf which is uniform and somewhat flat. This transformation ultimately results in an image having better contrast then the image which was kept as input to the transformation function.

So, on our part we were provided with two grayscale images GrayScaleMRI.mat and geospatialImage.mat. When we loaded these images to our MATLAB code file, only GrayScaleMRI.mat file was working correctly. So, we continued with this image for histogram equalization. Initially, the input image and its respective histogram was displayed as follows:

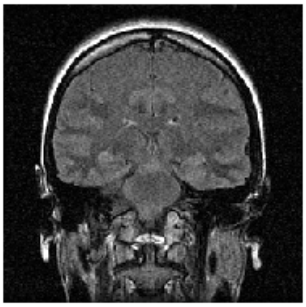

Fig 5. Input Image

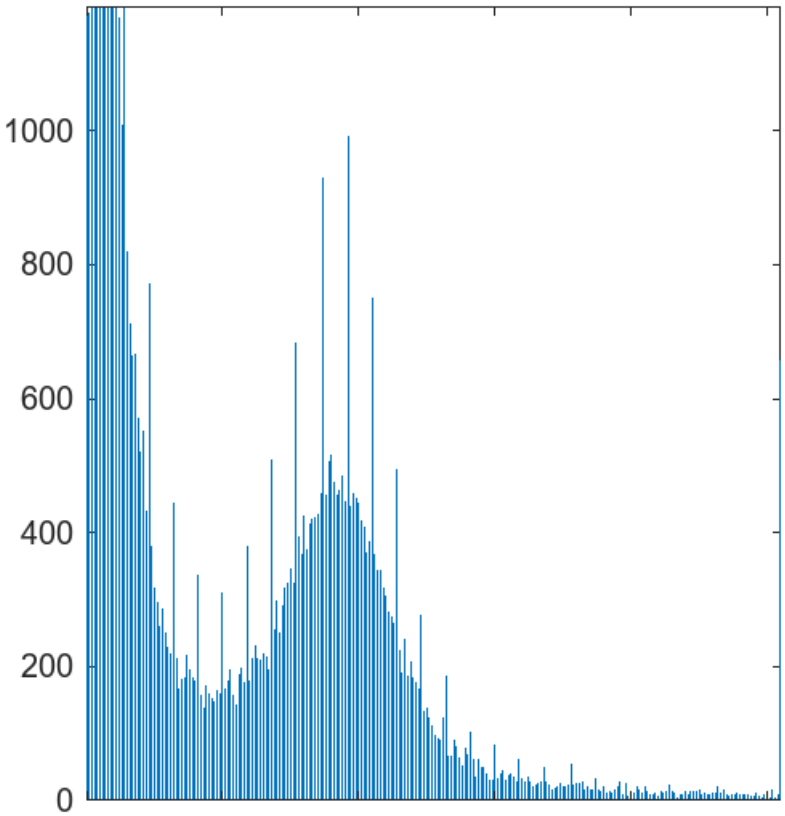

Fig 6. Histogram of Input Image

But when histeq() function with default bin size of 64 got applied to input image, the resulting image and its histogram was as follows:

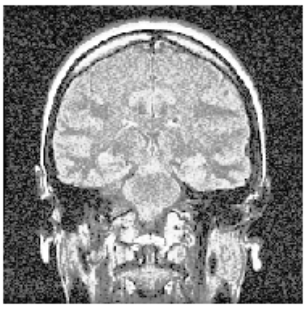

Fig 7. Resulting Image after Histogram Equalization

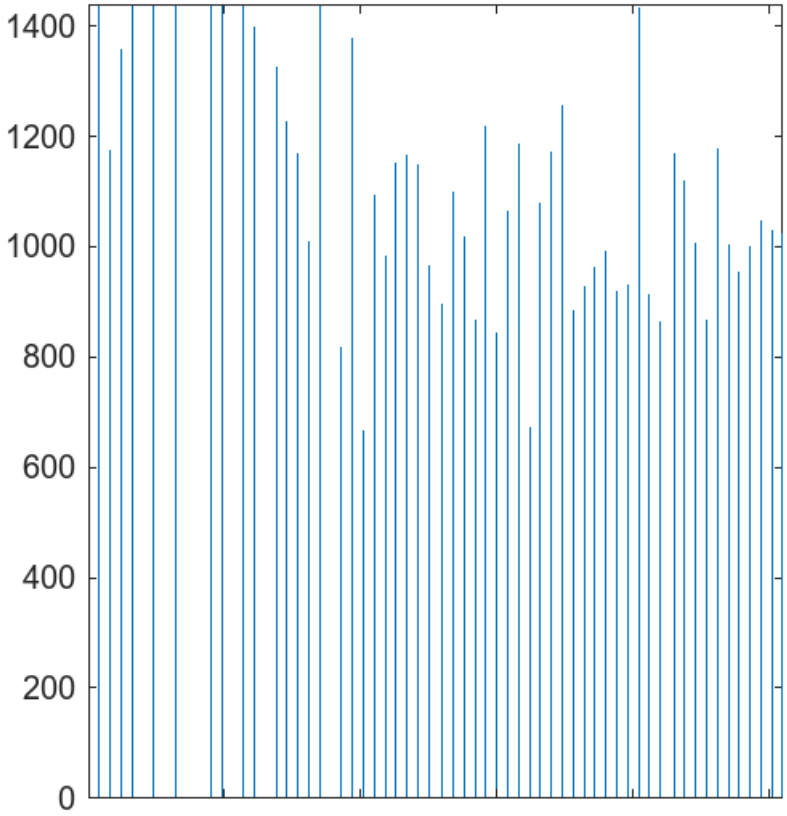

Fig 8. Resulting Histogram after applying histeq() function

Adding further to this, we tried customizing bin size by adding customizable bin values using a slider value input as nbins_sb. This resulted in customized bin-based histogram equalization whose output was as follows:

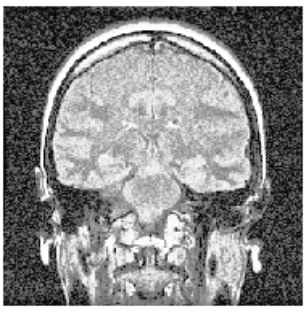

Fig 9. Resulting Image after Customized Bin Based Histogram Equalization

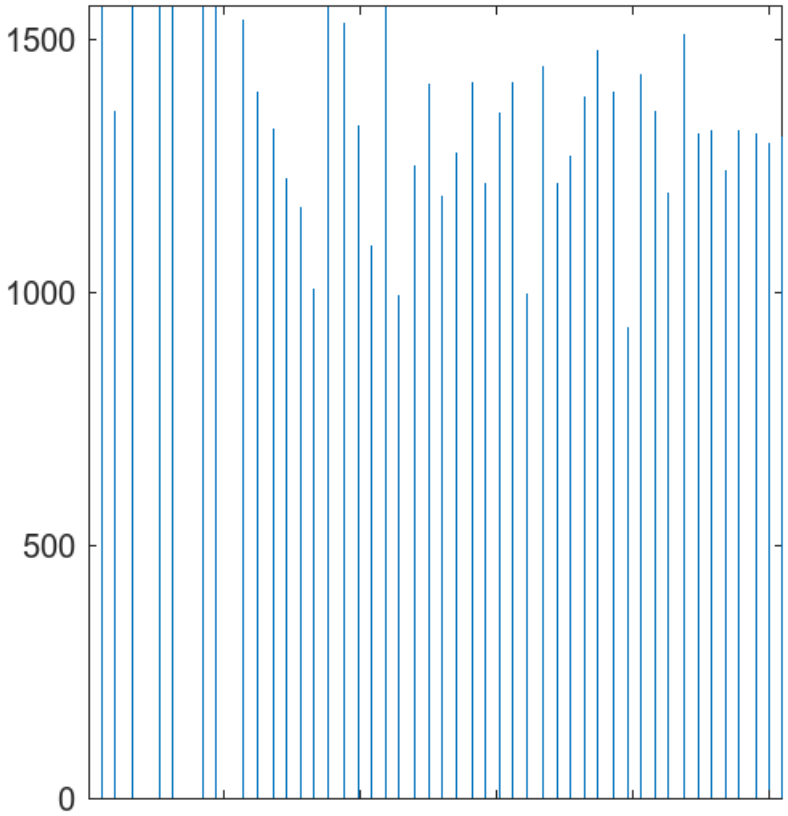

Fig 10. Resulting Histogram after applying Customized Bin Based Histogram Equalization using histeq() function

Finally, we also tried technique of histogram matching by using a “targeted” histogram as below:

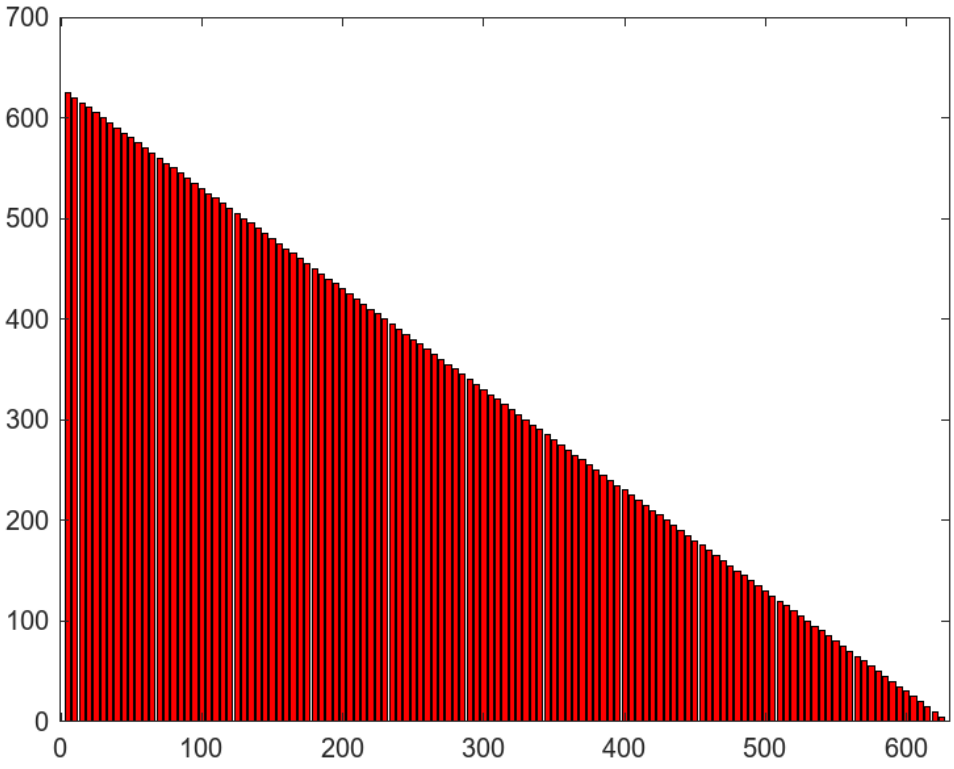

Fig 11. Targeted Histogram for Histogram Matching Technique with a target of 625

This target histogram value of 625 was then fed as a parameter to histeq() function to get a targeted equalization of our resulting image and its corresponding histogram as below:

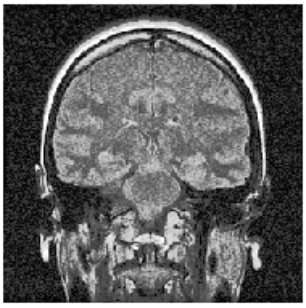

Fig 12. Resulting Image after Target Histogram Based Histogram Equalization

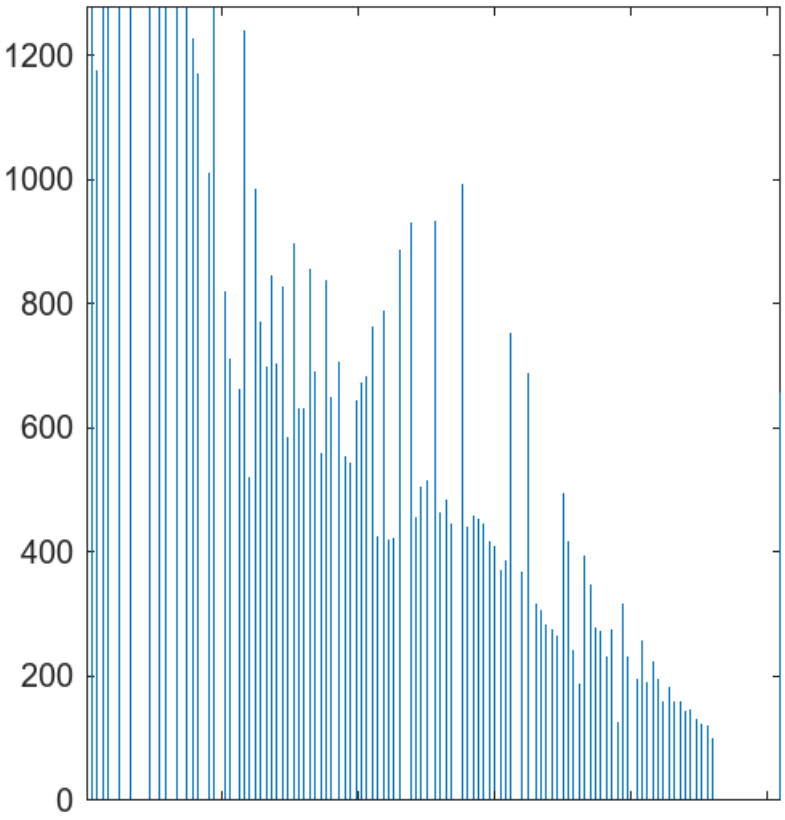

Fig 13. Resulting Histogram after applying Target Histogram Based Histogram Equalization using histeq() function