UML PROJECT 3

Ajay Anand Kumar: 013711933
(MSc Bioinformatics)

Pedro Alonsod: 013753179
(MSc Bayesian)

Instructions:

- Download the zipped file UmlProj3.zip. Extract the **UmlProj3Report.pdf** and **Exercise3.r**
- The source code can only be run in R.
- Install package "fields" in R. It has been used to compute the eucledian distance matrix in exercise 2.3-6.
- For every exercise there is function such as ex1.1 (), ex1.2 (), ex2.1 () etc. All the output and usage of these functions will be explained in this report.

Solutions:

Exercise1.1

Command: >source ("Exercise3.r")

>ex1.1 ()

Objective: The question was asked to generate 2000 samples from a two dimensional Gaussian mixture model with two clusters shown in Figure 11.5 of lecture notes.

Explanations: From theory GMM is a generative model and the data generating mechanism is:

1. The cluster index *r* was randomly chosen. Say like tossing a coin but with probability 0.7 and 0.3. These are the probability of cluster indices. The cluster indices are (-2,-1) and (2,2)
2. The data is generated by drawing a point from the multivariate Gaussian distribution with mew_r (cluster mean) and covariance matrix C_r. The covariance matrices are C_1 = diagonal (1,-1.730, 1.1907,-0.2) and C_2 = diagonal (1, 1.7321,-.7321,-0.2).

Here *r* is the latent variable and we get one realization of this latent variable through this mechanism.

The output of is as shown in Figure 1.1. One can see the Gaussian data with larger probability 0.7 with black points and cluster mean (-2, -1) marked. The other cluster with mean (2, 2) correspond to smaller probability 0.3. The corresponding function "**ex1.1 ()**" returns the mixture

model data realizations and has dimension 2 × 2000. The output figure resembles the Figure 11.5 of lecture notes. The cluster mean, cluster probabilities, and covariance matrices are the parameters that govern the properties such as spread and location of data cloud.

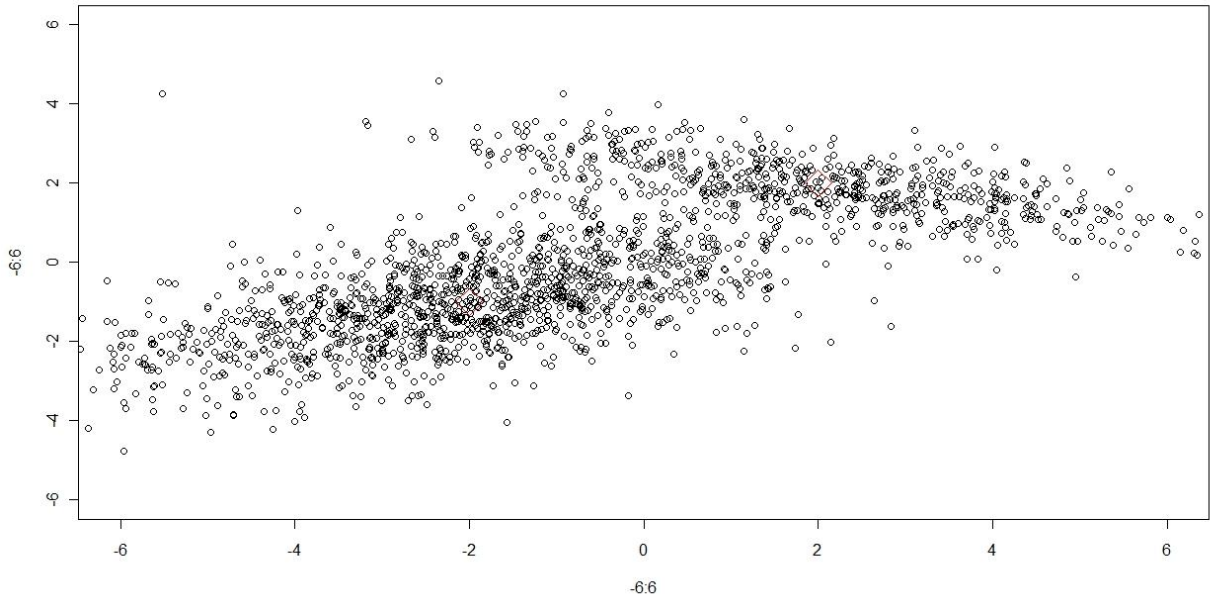


Figure 1.1: Sampling data points from Gaussian Mixture model with respective cluster probabilities and cluster means.

Exercise 1.2 and Exercise 1.3

Command: >source ("Exercise3.r")

>comb = ex1.2 ()

Objective: The aim was to implement the EM algorithm on the generated data from previous exercise and get the estimates of Cluster means, Covariance matrices, Cluster probabilities and objective function. The estimated values should be in the near range of true values.

Explanation: The heuristic EM algorithm was implemented. From lecture notes equation 11.15 to 11.19 were implemented on the data and the algorithm was run for several iterations until it converges. The convergence was checked for cluster means, Covariance matrices, and cluster probabilities. Once the convergence is reached, the objective function is calculated by using the estimated mean, estimated covariance matrices and estimated cluster probabilities and together they are used to evaluate Equation 11.21. While evaluating the logarithm of quantity q of equation 11.15 it was found that NaNs were produced. As such $\log(u+1e^{-10})*u$ was used transform the q and thus objective function was evaluated successfully. After evaluation of objective function the MAP estimate of every data point in GMM was evaluated. After

convergence q matrix (2×2000) is respective posterior probabilities of every data point. Thus for each data point the maximum probability is evaluated and it determines the membership of the data points to the respective clusters.

Results: The command comb = ex1.2 () returns a list object. The first element of list gives the objective function value. The second and third element gives the estimated cluster means. The fourth element gives the cluster probabilities. The fifth and sixth element gives the respective estimated covariance matrices corresponding to cluster probabilities. The estimated values are:

- Objective function value: -2486.989
- Cluster probabilities : 0.3109967, 0.689003
- Cluster means: (1.984422, 2.043394) and (-1.9661949, -0.9574094) respectively to corresponding cluster probabilities.
- Covariance matrices:

  C_1 = [,1] [,2]

  [1,] 3.956553 -1.0281770

  [2,] -1.028177 0.5299996

  C_2 = [,1] [,2]

  [1,] 4.292090 1.618684

  [2,] 1.618684 1.451441

The MAP estimates and the respective cluster membership for the data points is shown in Figure 1.2. The red points correspond to cluster with probability 0.69 and blue points correspond to cluster with probability 0.31. The obtained estimates of the parameters are closed to the true values with which data points are generated. However, there is difference in estimated covariance matrices. But this only corresponds to orientation of data cloud. Since we are dealing with clustering hence the important point is to get good estimation of cluster means and cluster probabilities and respective objective function which determines the cluster membership.

N.B: The EM algorithm ran for 100 iterations. The output of these parameters is printed on the screen for every iterations. The algorithm is not checked for convergence. The reason is because EM algorithm blows up several times and such we wanted to monitor values of every iterations and then choose the corresponding starting values. In several runs, the algorithm converges around 50 to 60 iterations. While running the algorithm one can visualize how the parameter value changes and then finally converges. The start point for cluster means are chosen (0,0) and covariance matrices were initialized randomly. The initial cluster probabilities chosen are (0.1 and 0.9).

Thus the desired target is achieved.

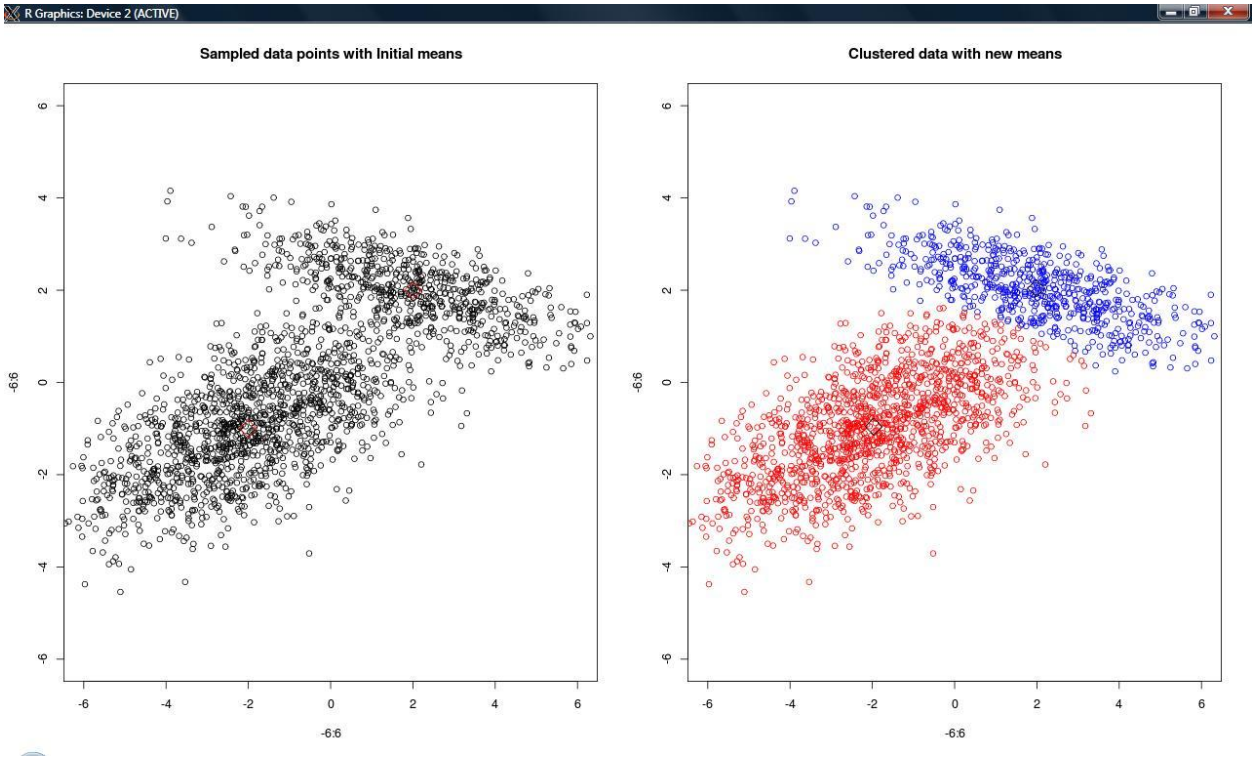


Figure1.2: Visualization of MAP estimates of data points having objective function value = -2486.989. The red points belong to cluster with probability 0.69 and blue points belong to cluster 0.31. The left figure is the original sample data from Gaussian Mixture model.

Exercise 1.4

Commands: >source("Exercise3.r")

>comb1 = ex1.4 ()

>comb2 = ex1.4 ()

>comb3 = ex1.4 ()

>comb4 = ex1.4 ()

>comb5 = ex1.4 ()

Objective: The aim is to generate an artificial data from GMM with 4 Gaussian distribution with respective means. Run EM algorithm with randomized cluster means, covariance matrices, cluster probabilities for 5 different start points. Then visualize the MAP estimates for highest and lowest objective function.

Explanations:

1. 100 data points were sample from 4 Gaussian distribution randomly. Thus, X has dimension of 2 × 100. The covariance matrices of 4 Gaussian distribution were chosen randomly and cluster means were chosen as (-2, -1), (2, 2), (4, -4), (-4, 2).
2. For EM algorithm, initial means were randomized. The covariance matrices were initialized to :
   sigma.1 = matrix(c(2.4,-0.3,-0.3,0.07),2,2)
   sigma.2 = matrix(c(1,1.16,1.16,1.7),2,2)
   sigma.3 = matrix(c(16,1.92,1.92,1),2,2)
   sigma.4 = matrix(c(13.60,-13.36,-13.36,16.7),2,2)
3. The cluster means were randomized within the range of -6 and 6.
4. Five initial start points were the initial means and hence randomized. Thus in commands five times the function ex1.4 () was run.

Results: For each randomized start point, the objective function was evaluated. The objective function were not same for all the runs. For this report, the corresponding objective function values are:

- Obj1 = -912.652
- Obj2 = -108.532
- Obj3 = -113.077
- Obj4 = -220.833
- Obj5 = -475.202

The highest objective function value is Obj2 and lowest is Obj1. The corresponding plots for all the objective function and value and respective MAP estimates are shown in Figure 1.4

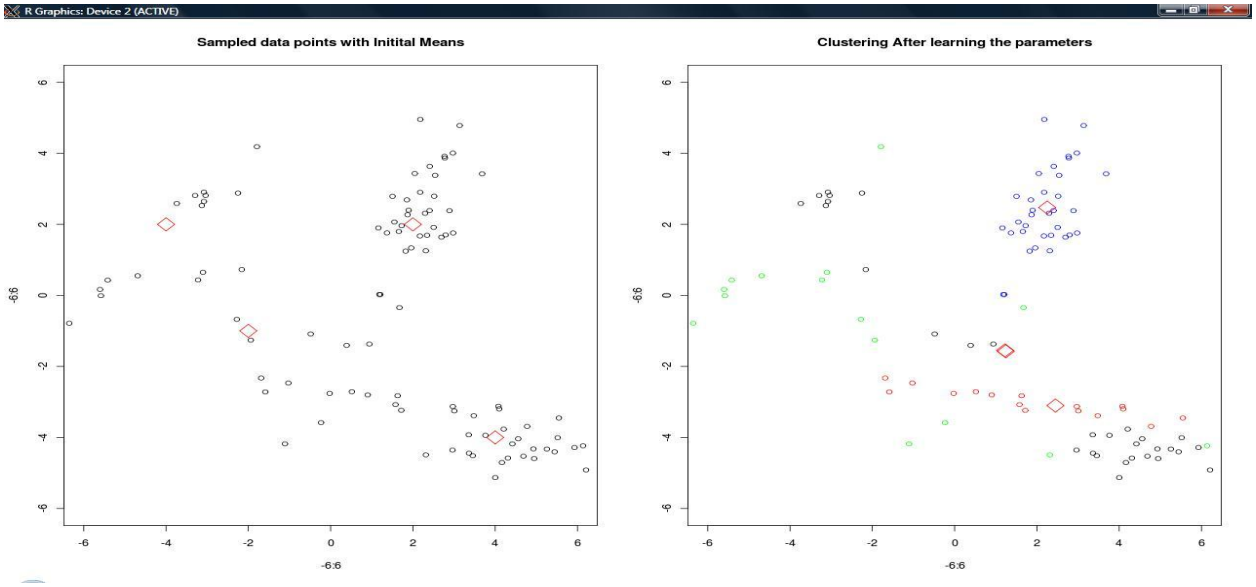


Figure1.4.1: Objective function value -912.652. The estimated cluster means are also plotted and 4 Gaussian data coloured in blue, black, green and red. Left hand figure is original sampled data points with original cluster means.

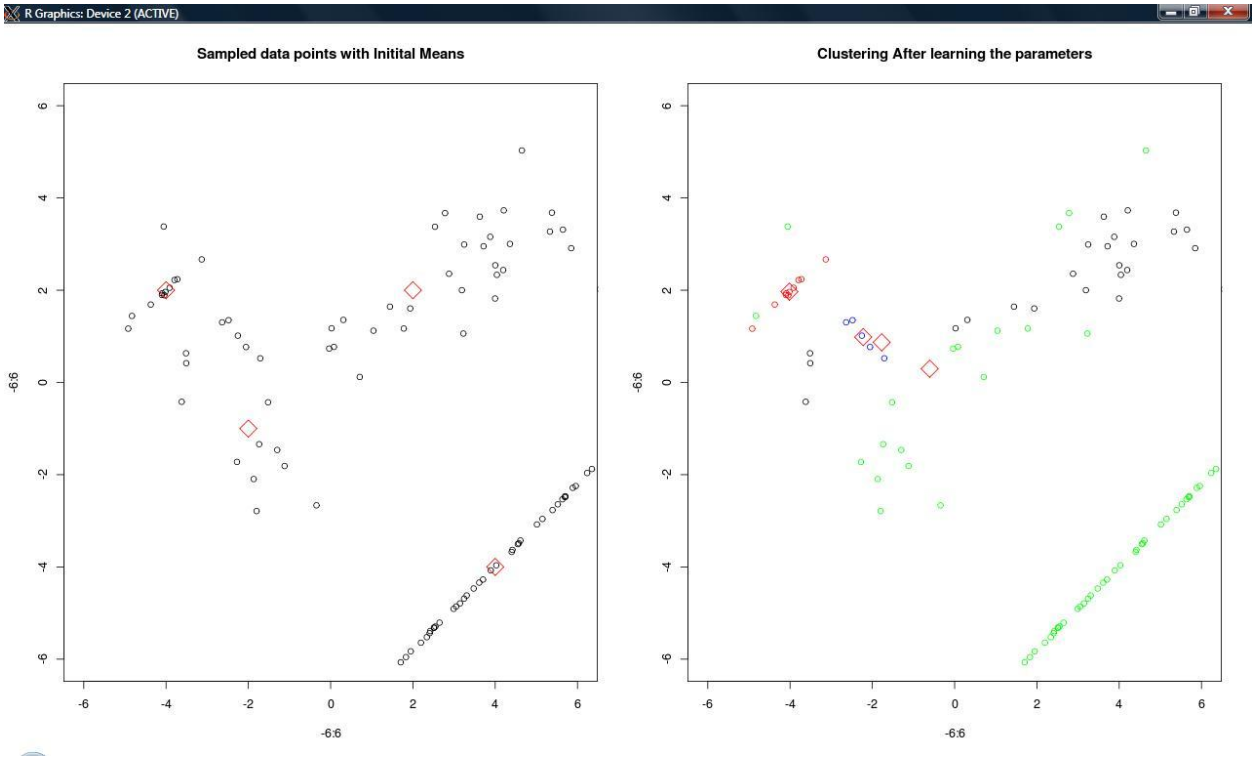


Figure 1.4.2: Objective function value: -108.532

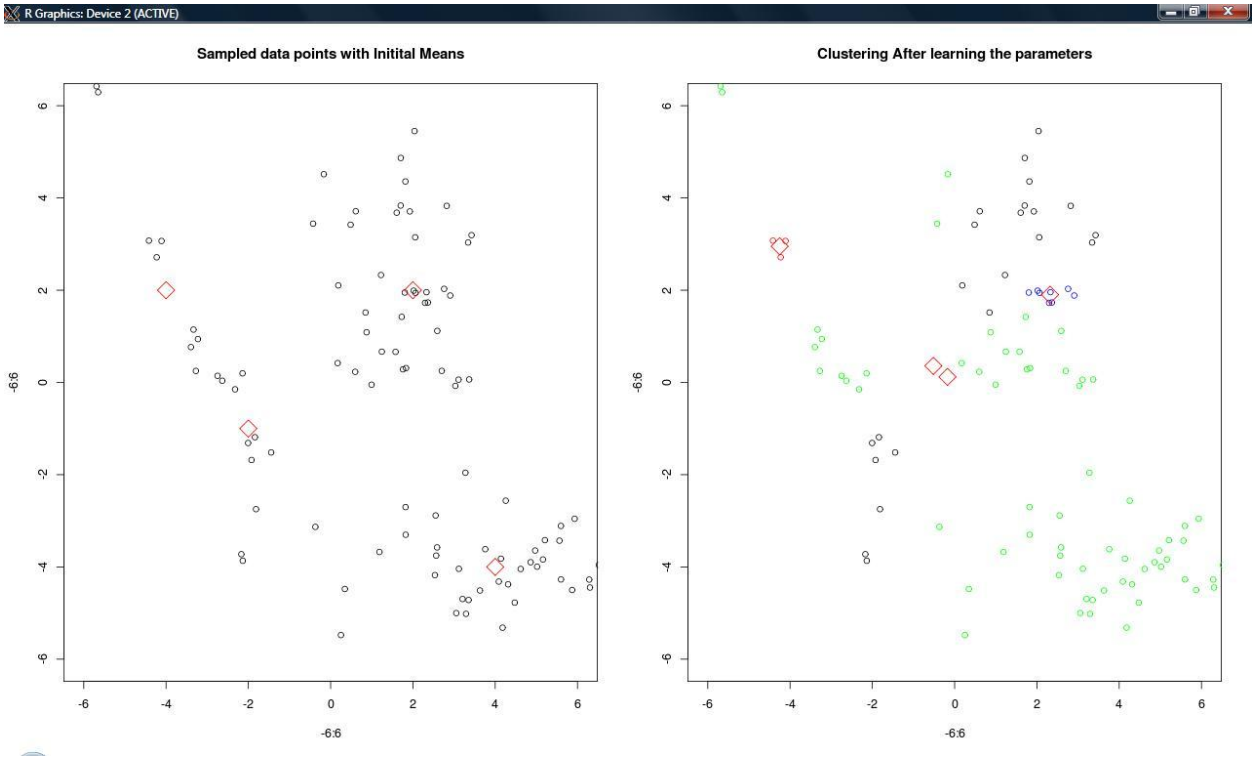


Figure 1.4.3: Objective function value: -113.077

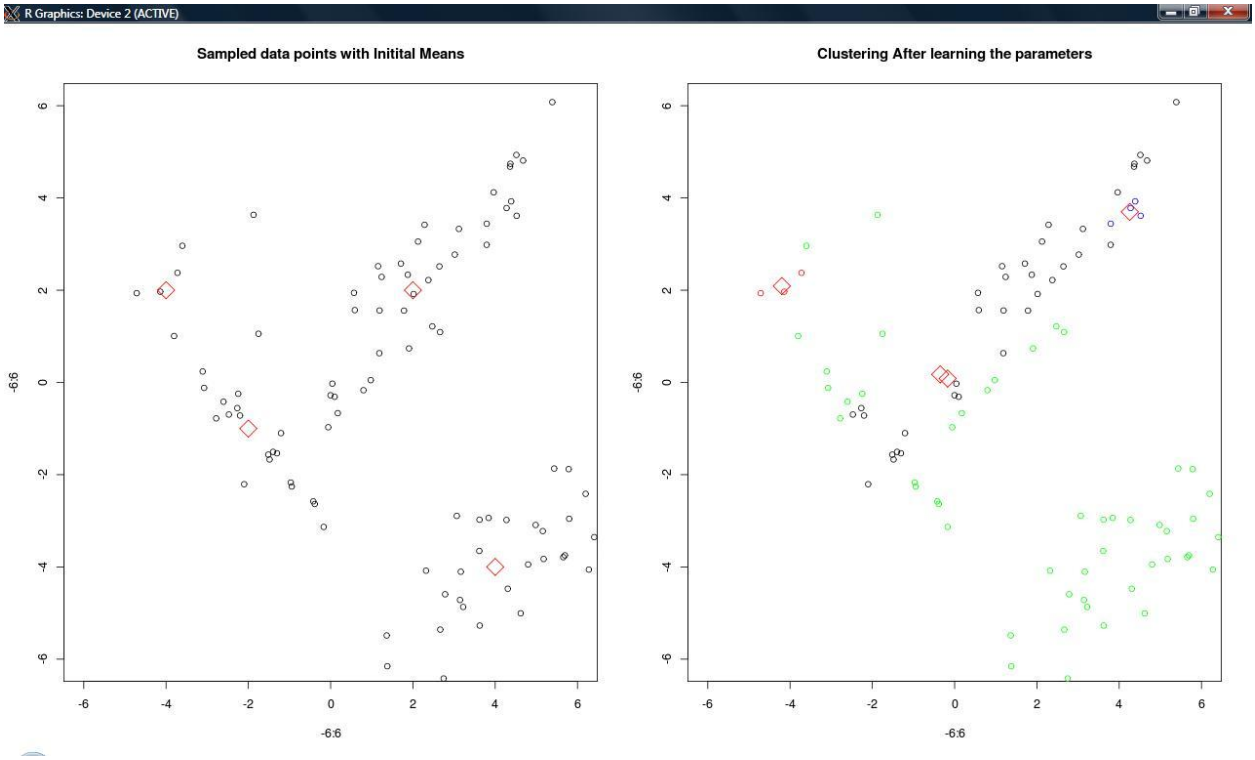


Figure 1.4.4: Objective function value: -220.833

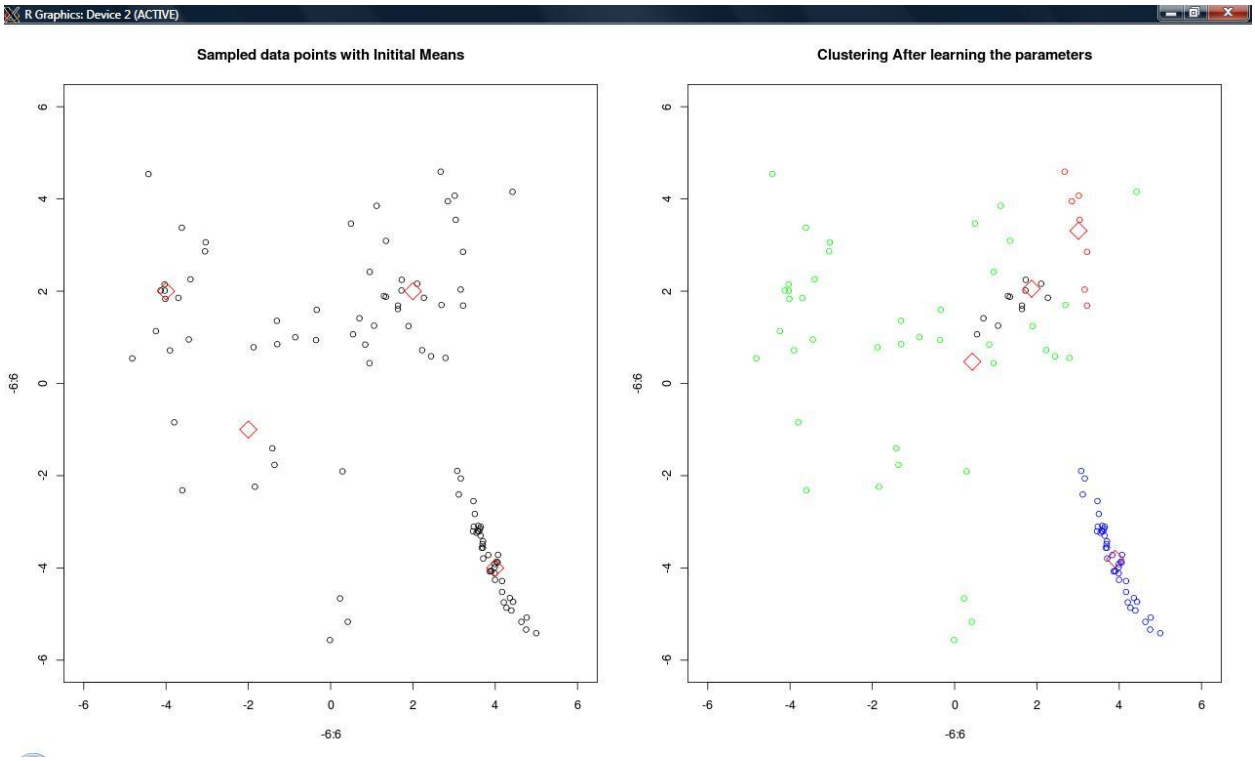


Figure 1.4.5: Objective function value: -475.202

Exercise2.1

Commands: >source("Exercise3.r")

>ex2.1()

Aim: To perform linear MDS on the two dimensional data given in data_proj.txt and then visualize the projections based on the value of projections.

Explanations: The input data is clearly a two dimensional non linear in structure. As such it cannot be meaningfully represented using linear components. Hence, if we evaluate its principal components, its' linear combination cannot represent the data. Metric MDS or linear MDS is applied to this data. Hence, using non-linear transformation, a single component can represent the data well. As such the data was represented using its first principal component. Initially, the Eigen Value Decomposition was done on this data that gives its principal components and directions. Sorting out the principal components the data was colour coded with the eigen vector corresponding to largest principal component.

Results: The obtained result is shown in Figure 2.1.

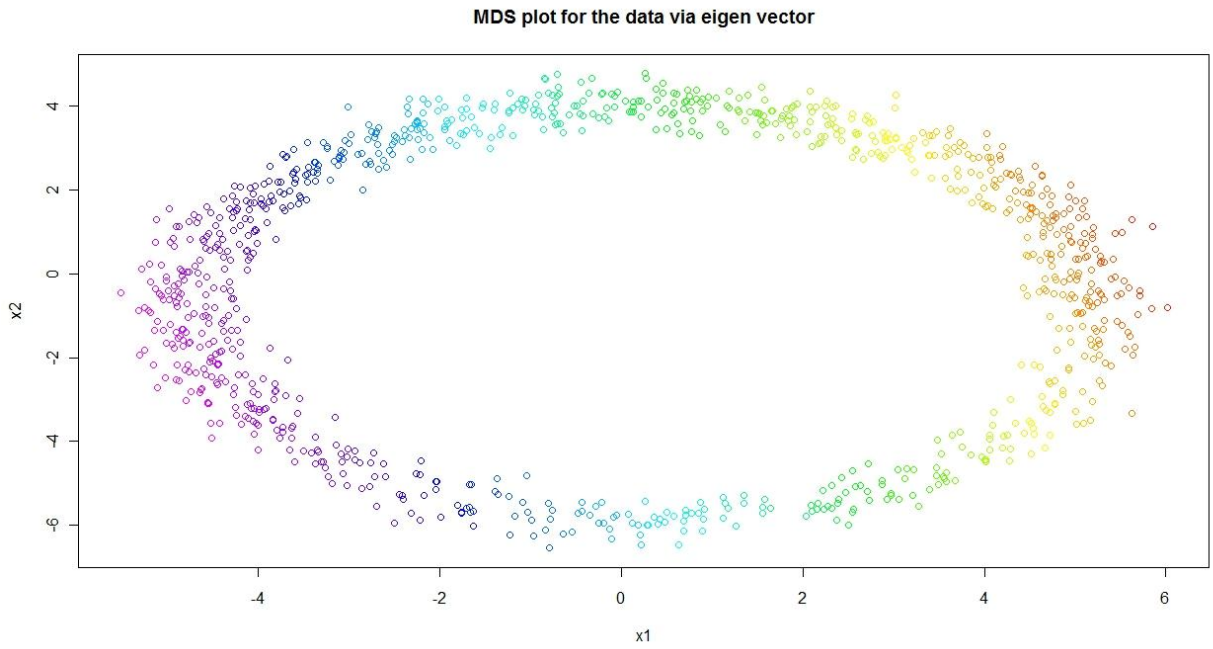


Figure 2.1: MDS for the data via the first eigen vector.

Form Figure 2.1 one can see the application of metric MDS on the data set. Colour of each data point shows the value of the principal component at that data point. This implies that metric MDS is not able to learn the nonlinear structure of the data and as such requires the nonlinear functions of the Eucliedian distances.

Exercise2.2

Commands: >source("Exercise3.r")

>ex2.2()

Aim: The aim was to do PCA on the data. Then plot the first principal component direction and compute the first principal components. Color each data point by the value of its first principal components. Then compare the resultant plot with previous exercise.

Explanations: PCA is generally performed with covariance matrix of input data. Here, first the covariance matrix was calculated and then the corresponding EVD was done. The first principal component is the given by the first highest eigen value and its direction is given by eigen vector corresponding to the highest eigen value. PCA differs from MDS in a way that in PCA covariance matrix is used whereas in MDS the distance matrix is used. After calculating the principal component weights, the non linear data was projected to these principal components.

Results:

The first 2 principal components are:

13.23061 10.94219

The projected data on these principal component directions:

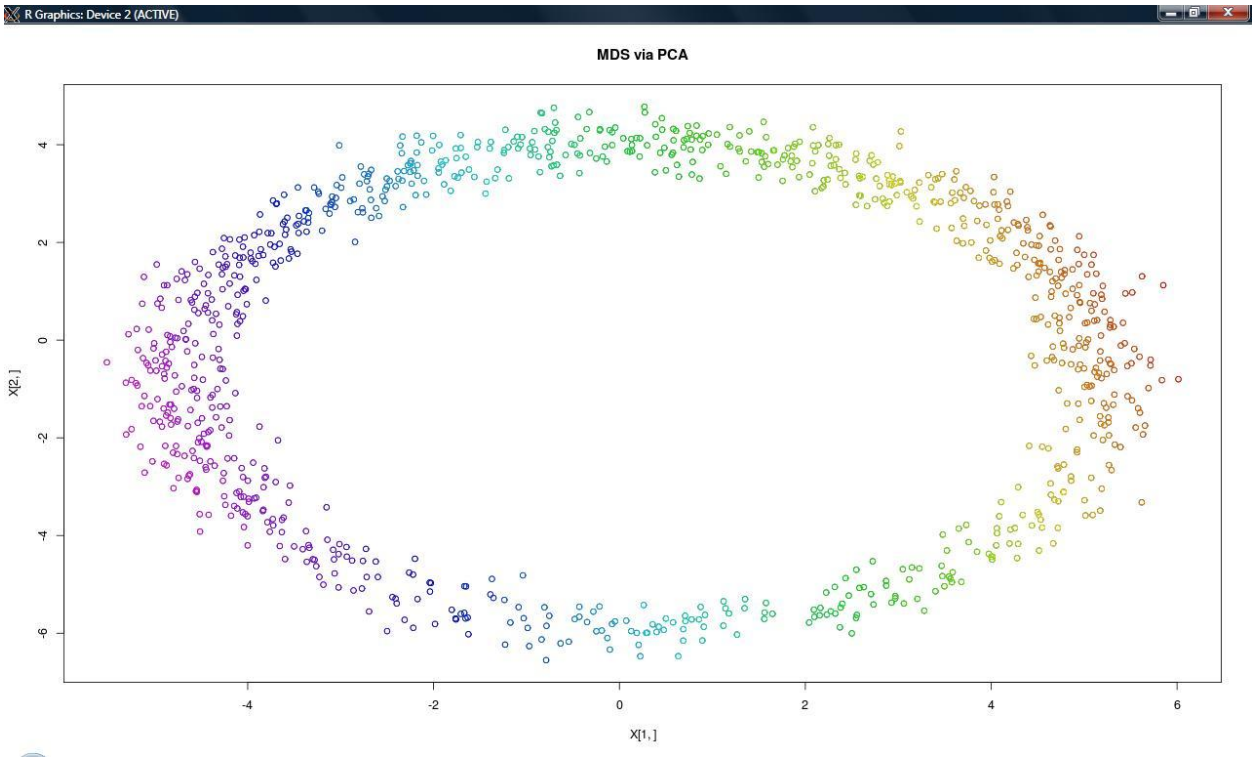


Figure 2.2: Application of linear MDS via PCA on the non linear dataset.

From Figure 2.2 and 2.1 we can see that there is no difference between linear MDS and PCA. As explained earlier linear MDS is a variant of PCA, and difference lies only in usage of distance matrix than covariance matrix. Here data cloud is of nonlinear structure as such the linear PCA is not able to learn. Hence the two figures are same. As such it is required to have some non linear function of distance matrix, for e.g usage of kernel PCA and IsoMap.

Exercise 2.3

Commands: >source("Exericse3.r")

>ex2.3()

Aim: To apply SOM algorithm on the above nonlinear data and visualize learned model vectors w_i, where i = 4 to 20.

Explanations: Self organizing Maps are unsupervised learning techniques for non-linear projections and visualization of high dimensional data. From previous exercise 2.1 and 2.2 it is clear that linear methods such metric MDS and PCA are not suitable for non-linear projections and such it is required to have non-linear methods for projections. In SOM algorithm following steps has been done:

1. First a model vector $\mathbf{w_i}$ is chosen randomly. In example case i = 1 to 7.
2. For each of data point, the closest model vector $\mathbf{w_i}$ was found and corresponding index i was kept.
3. For each $\mathbf{w_i}$, set of all those data points whose similar model vector belongs to the neighborhood set of node i. For eg. if i =5, then its neighborhood set of nodes are 4, 5, and 6. For i= 1 its neighborhood sets are 7,1,2 and for i= 7, the neighborhood sets of nodes are 6, 7, 1.
4. The new model vector $\mathbf{w_i}$ would have the values mean of all those data points collected in previous step.
5. The process from step 2 to 4 is repeated until the convergence.

Initialization of model vector is very crucial. The algorithm was run for several initializations of model vectors and numbers of these model vectors were from range 4 to 20. The initialization of these model vectors were done randomly within the range of min and max of input data matrix. If initialization is not proper then algorithm reaches a local minimum and does not converge.

Results:

Following figure are for model vector $\mathbf{w_i}$ where i = 4, 7, 10, 20. The input non-linear data is coloured according to most similar model vectors.

For i = 4 or node = 4

Figure 2.3.1 shows the 4 node vector points in black. First two node points overlap and while running the algorithm after 100 iterations it never reached convergence. On visualizing the

values of 4 model vectors in every iteration, it was found that the model vector values alternates after certain iterations. This clearly indicates that algorithm has reached local minimum and this can be treated with proper initialization values. Since, in this case the model vectors were initialized randomly within the range of min and max of input data matrix.

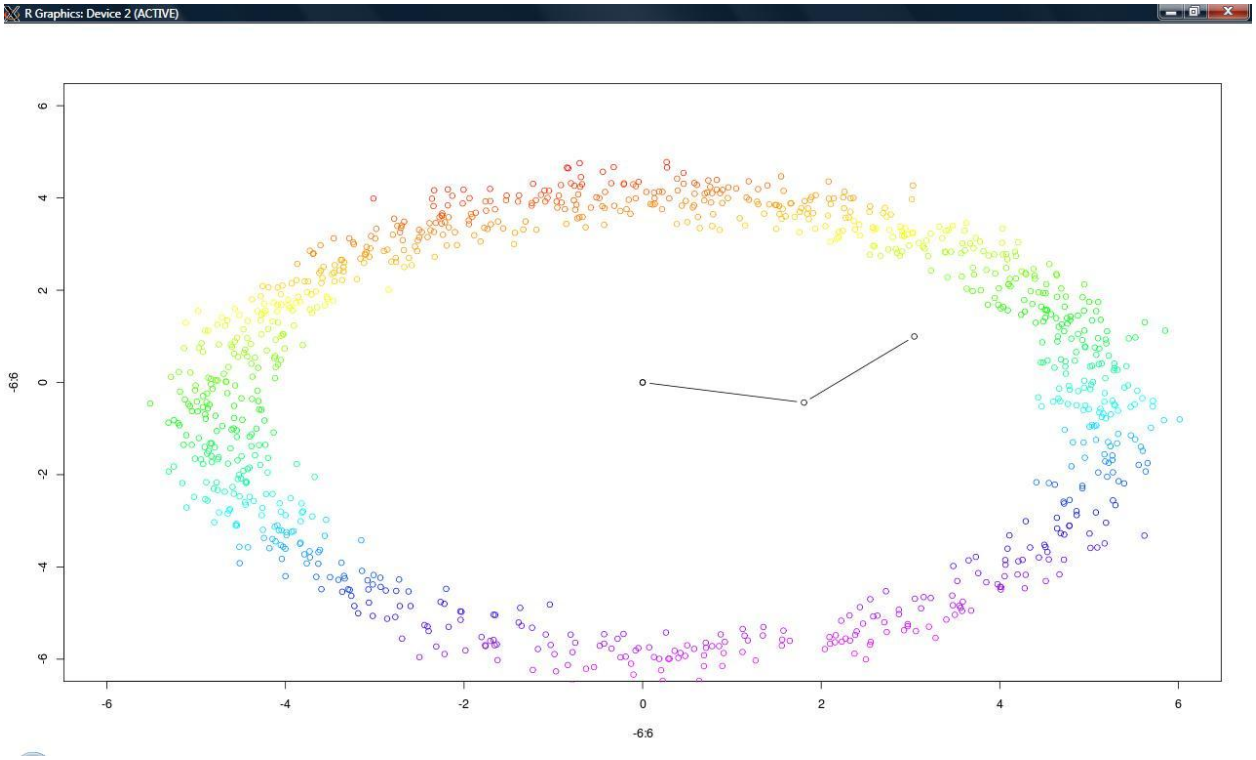


Figure 2.3.1: Visualization of model vectors of size = 4. Here first two model vector overlaps at (0, 0).

For i =1 to 7 or number of model vectors of size = 7.

In this case the algorithm was converged after 50 iterations. But interesting point to note is that bad choice of initial model vectors leads to local minimum. This resulted in **Figure 2.3.2a**. However, with another choice of initialization of random model vectors it was also found that although the algorithm converged, it also resulted in well ordered learned model vectors as shown in **Figure 2.3.2b,** and **Figure2.3.4.** From these figure it is clear that this implementation of SOM clearly depends upon choice of initial values of model vectors. Even though algorithm converges, but model vectors remains unordered and doesnot reflect the good projection of non-linear data.

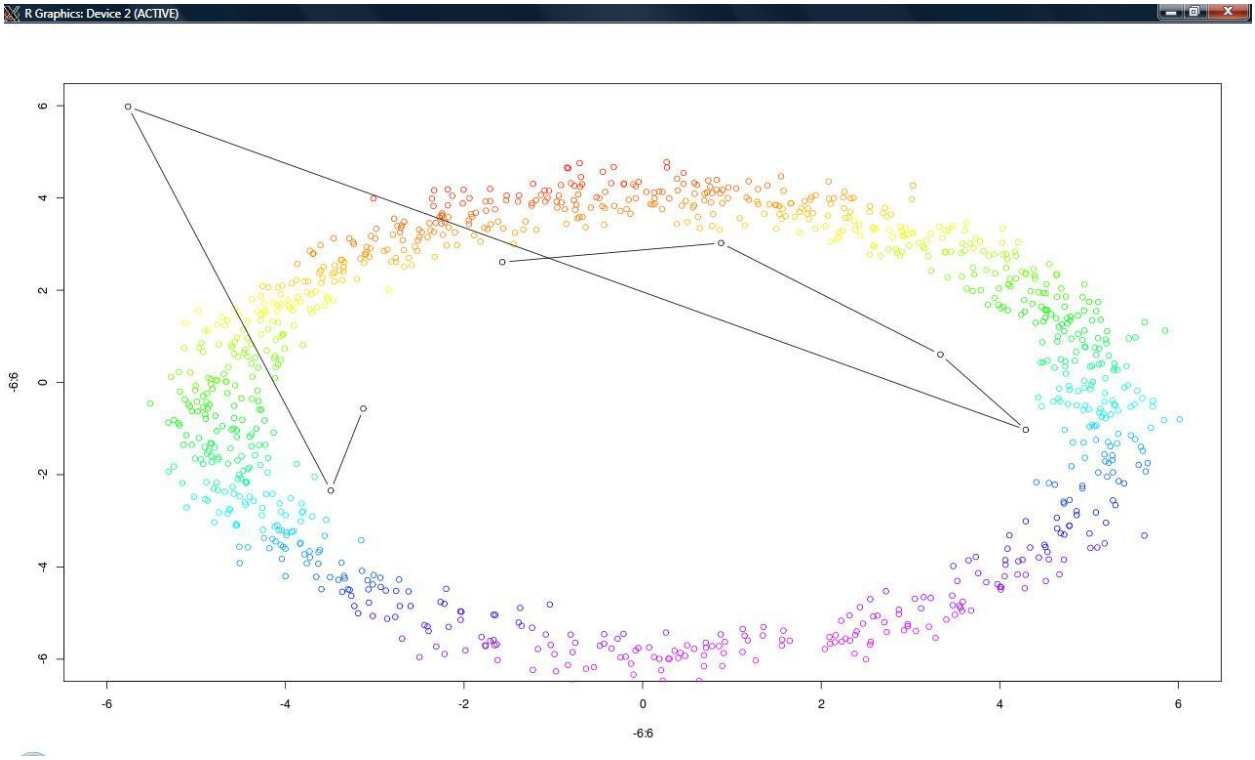


Figure2.3.2a: Visualization of learned model vectors. Due to bad choice of initial model vectors the algorithm gets stucked in local minimum.

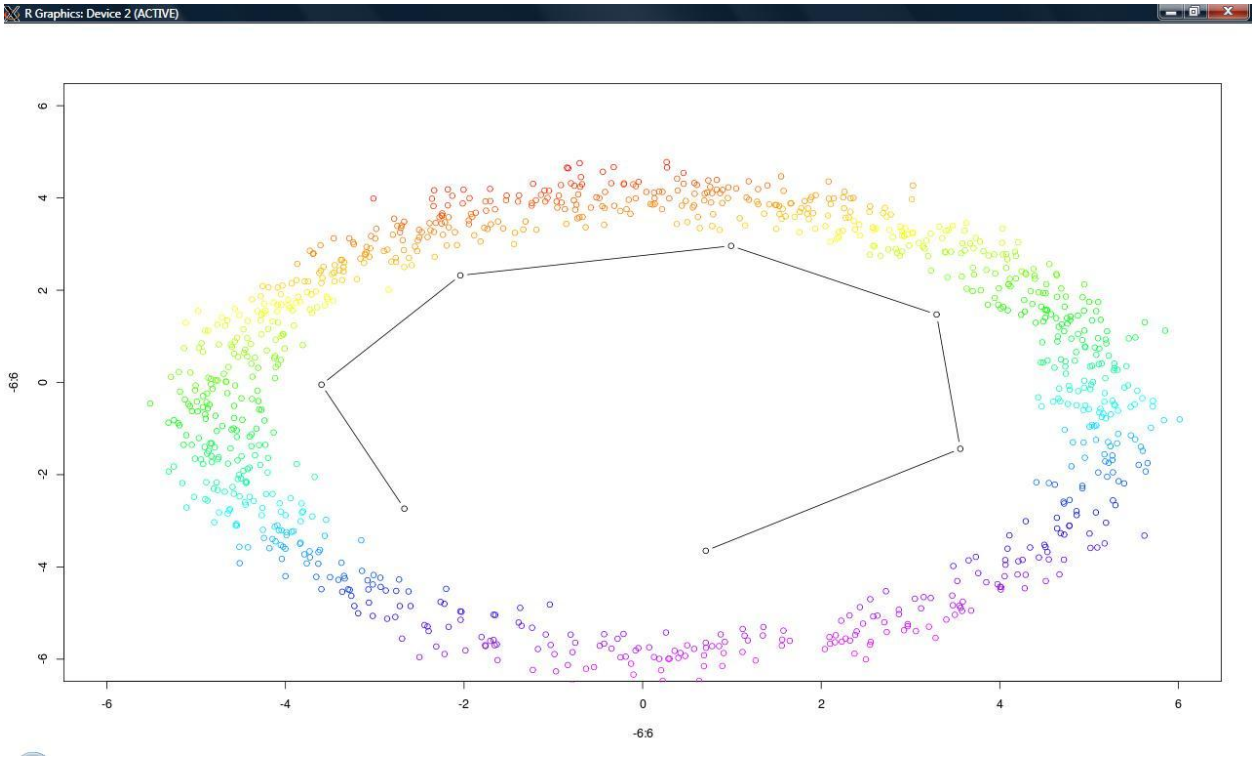


Figure 2.3.2b: Visualization of model vectors when in finds right solutions.

Similarly for i=1 to 10 or node size =10, right solution was found.

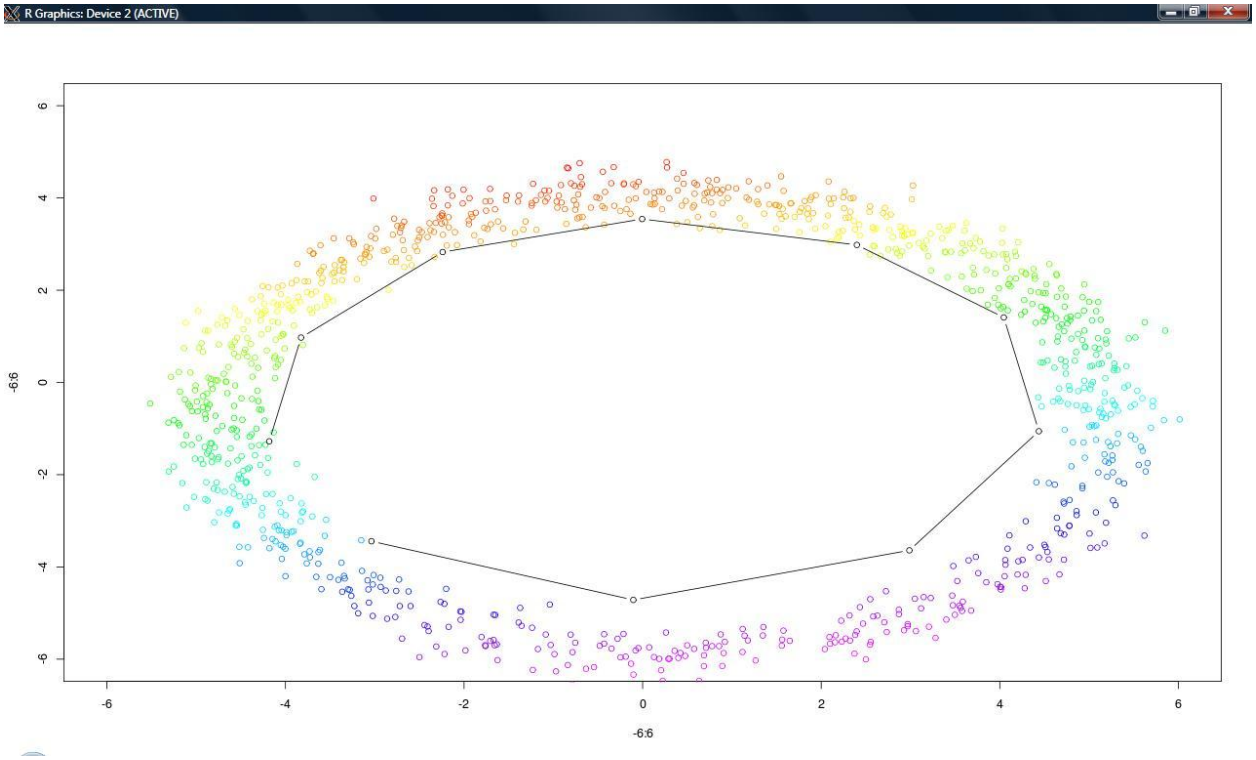


Figure 2.3.4: Visualization of learned model vectors of size 10 with good initialization of model vectors.

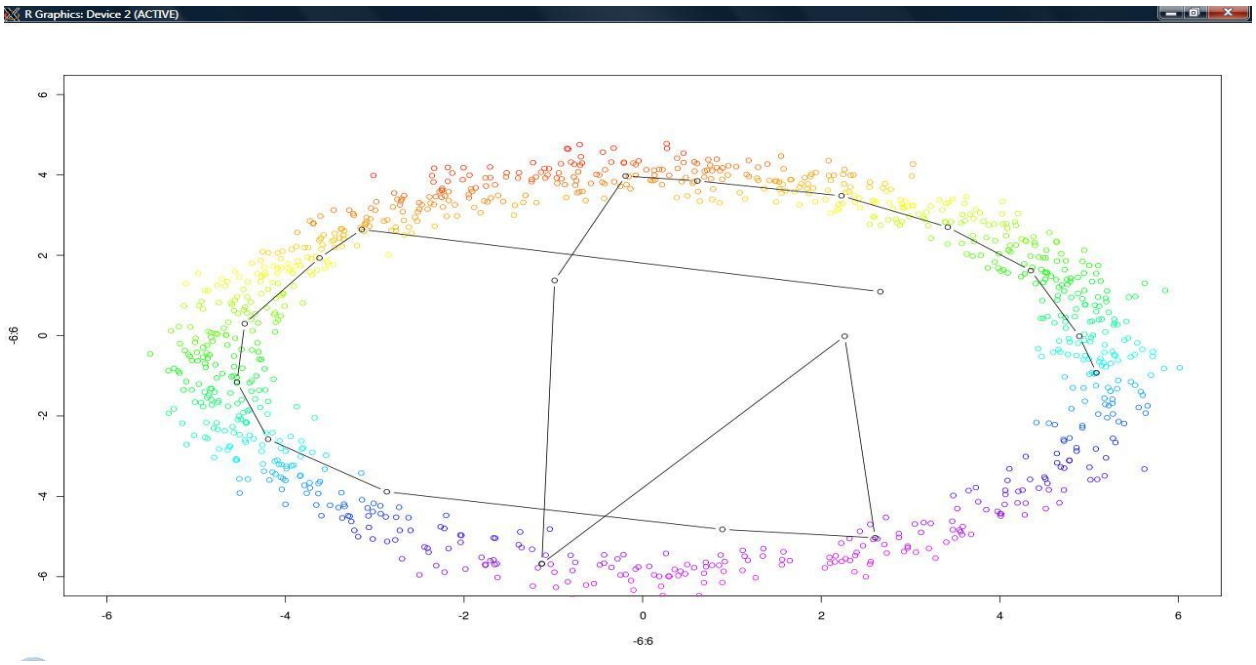


Figure 2.3.5: Visualization of model vectors of size=20. The algorithm gets stucked in local minimum and hence because of bad choice of initial values of model vectors.

Exercise2.4

Commands: >source("Exercise3.r")

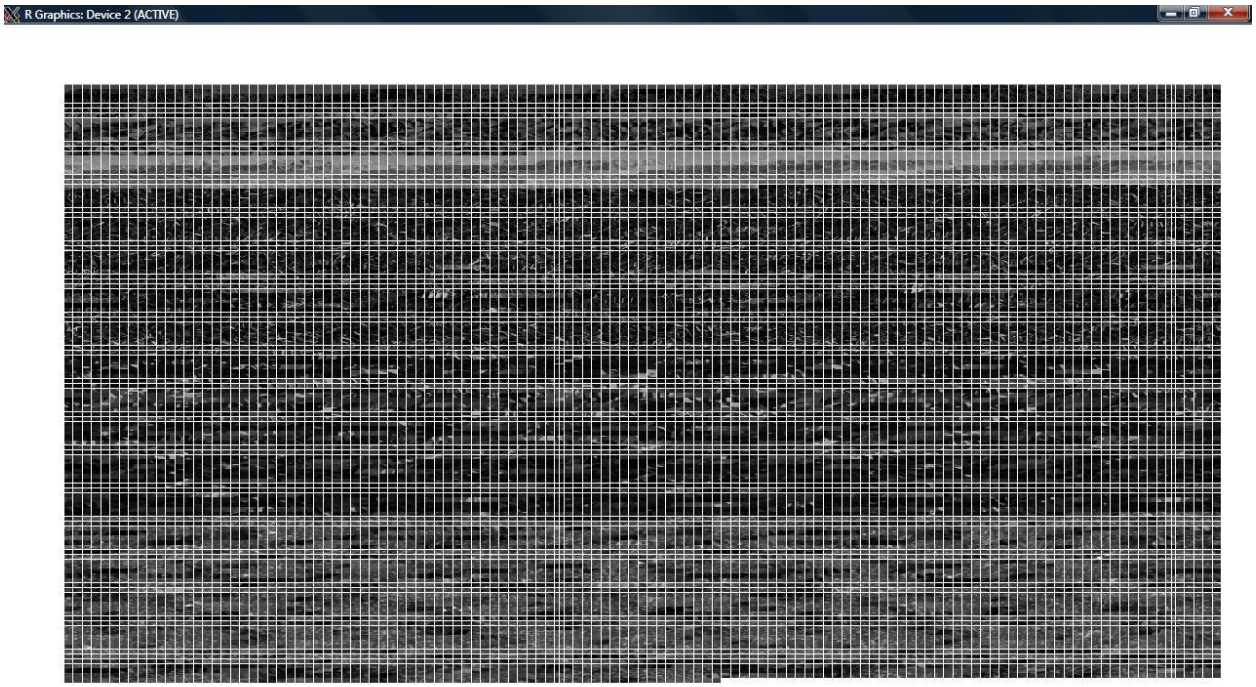
>w = ex2.4()

Aim: Load 6 images file I1, I2… I6 file and extract non-overlapping patches of size 10px × 10px. Then reshape each patch as 100 dimensional vector and put together in a data matrix of size 100 × 15696.

Explanations: The patches were extracted from every image file and reshaped into 100 × 15696 dimensional matrix. Thus each patch is a realization of 100 dimensional vector. Thus in total there were 15696 numbers of patches retrieved. Each patch has intensities value as such on sample visualization of each patch is shown in **Figure 2.4**.

Results:

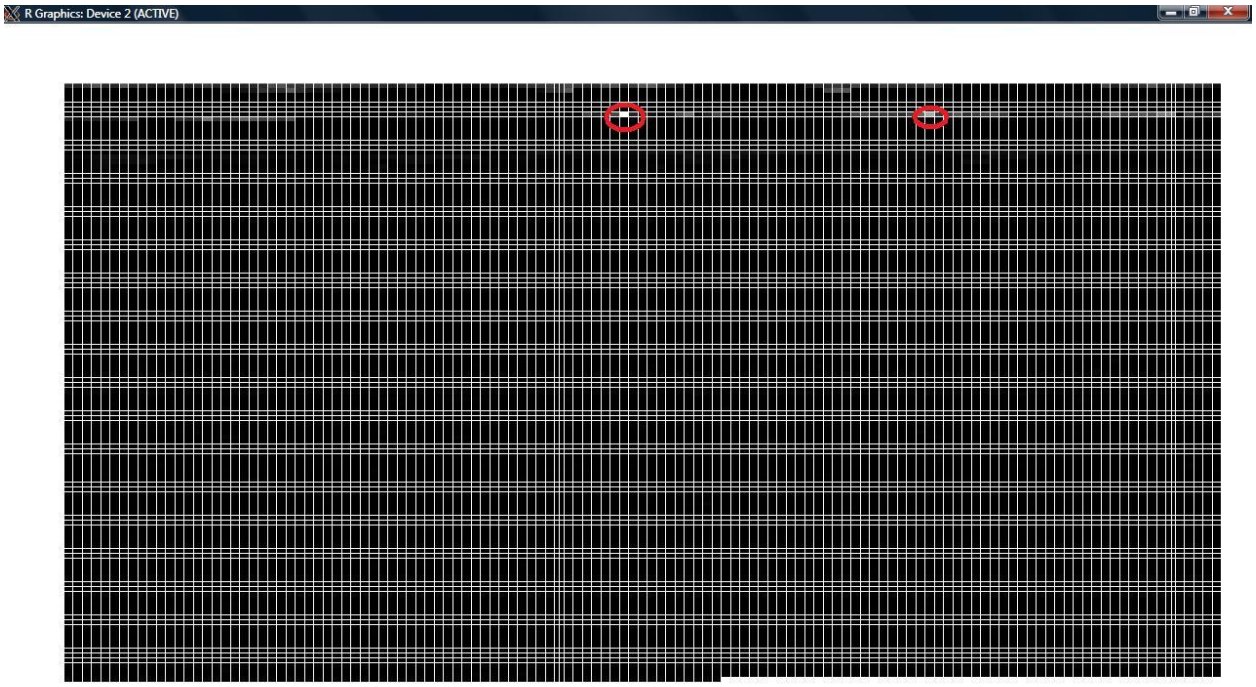
Figure 2.4.1: Visualizing these patches gives respective intensities values.



Each cell of this image is a 10px × 10px size and thus there are 15696 patches. Since in the data matrix of patches each column that represent a patch and represents the intensity values, if we substract the mean and normalize it unit variance the resultant image looks like in **Figure 2.4.2.** The images marked in red show image patches having variance equal to zero or nearby zero. This type of modeling establishes the fact that each patch is not independent of each other and shares some correlation among each other. This is different from Exercise 3 of project 2 where each of images from mixture were assumed to be independent. Statistical independence of signal is vital condition for application of ICA. But for SOM clustering there should be some

correlation between individual signal components such that they can be clustered based on their similarity or correlation.

Figure 2.4.2: Visualization after subtracting mean and normalizing to unit variance



The images marked in red show image patches having variance equal to zero or nearby zero. These patches are white and have very low values in comparison to other patches.

Exercise2.5

Commands: > source("Exercise3.r")

> w_vec = ex2.5()

> visual(w_vec)

Aim: To preprocess the patches data matrix. Then apply SOM algorithm with 10, 20 and 20 model vectors and visualize the corresponding learned **w** vector.

Explanations: As shown in previous exercise the data was preprocessed by deducting the mean from every patches and rescaling it to unit variance. Out of all the images, patch 819 has zero variance. As such, all patches were rescaled to unit variance other than 819^{th} patch. The corresponding patch has been marked in **Figure 2.4.2.** The SOM algorithm was applied to the processed patches with model vector **w** of size 10, 20 and 30. The **w** vector was initialized randomly and all steps performed for SOM were similar as in previous exercise2.3. The algorithm converged after 70 iterations and final **w** vector was obtained. Since SOM reorders the

w vector based on decreasing similarity. Moreover, values of **w** vector indicates the fact that **w1, w2, w3, w4, w5** are arranged according to decreasing order. Hence, it can be inferred that patches were arranged based on decreasing order of similarity. This is shown in **Figure 2.5.1.**

Figure2.5.1: Visualization of learned **w** vector for 10 nodes. The algorithm converged after 70 iterations.

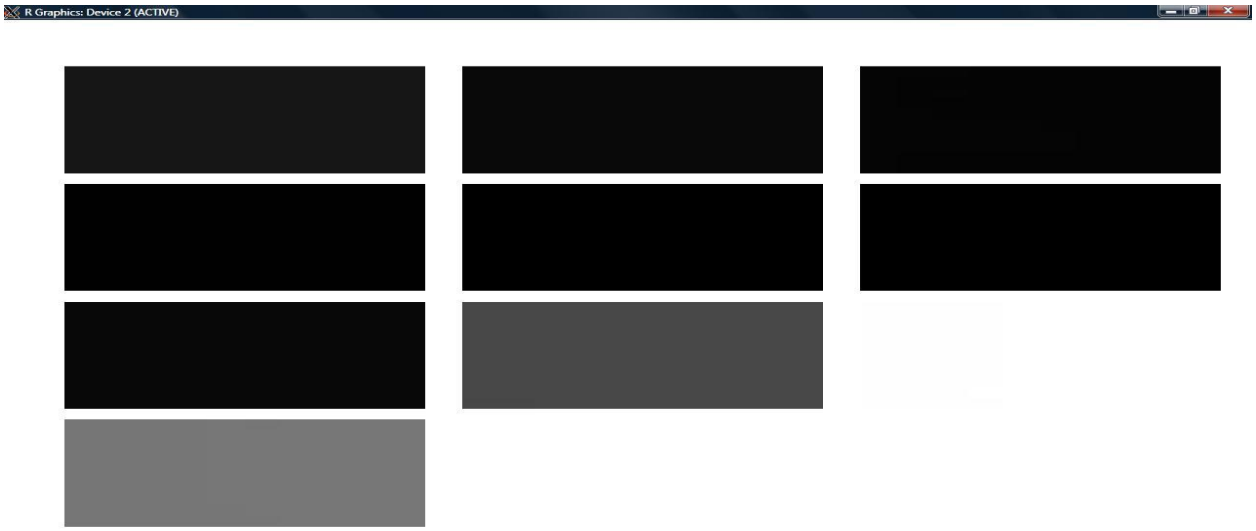


Figure 2.5.2: Visualizing the learned **w** vector with 20 model vectors. The algorithm converged after 120 iterations.

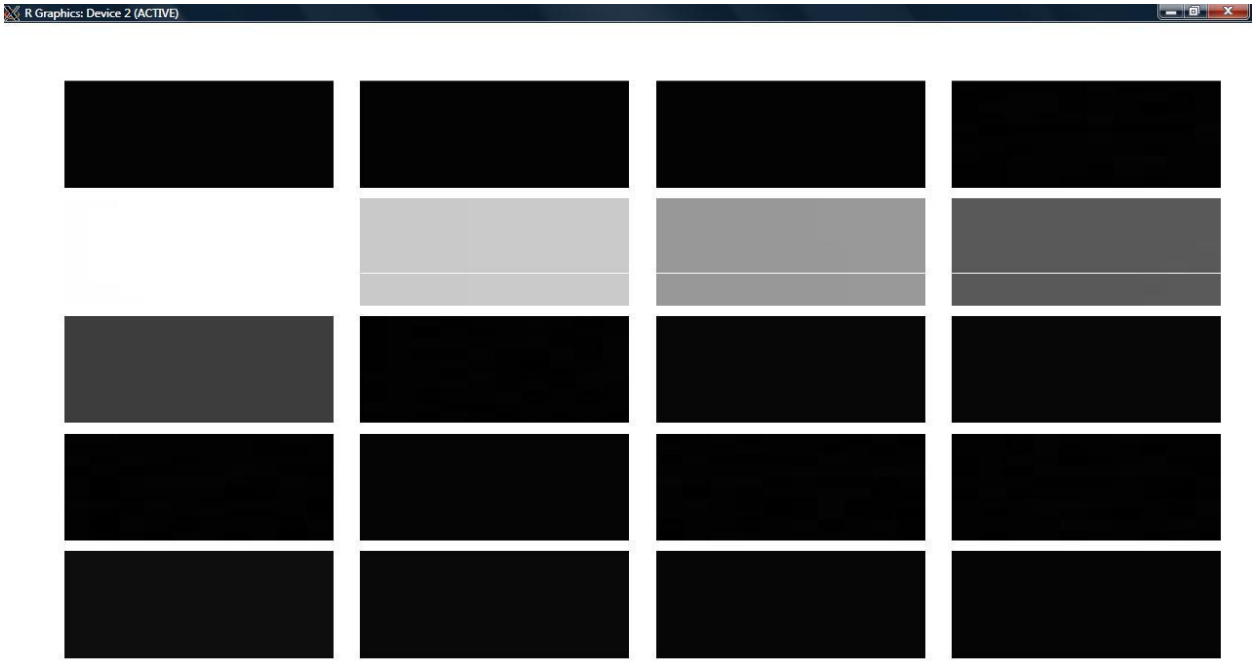
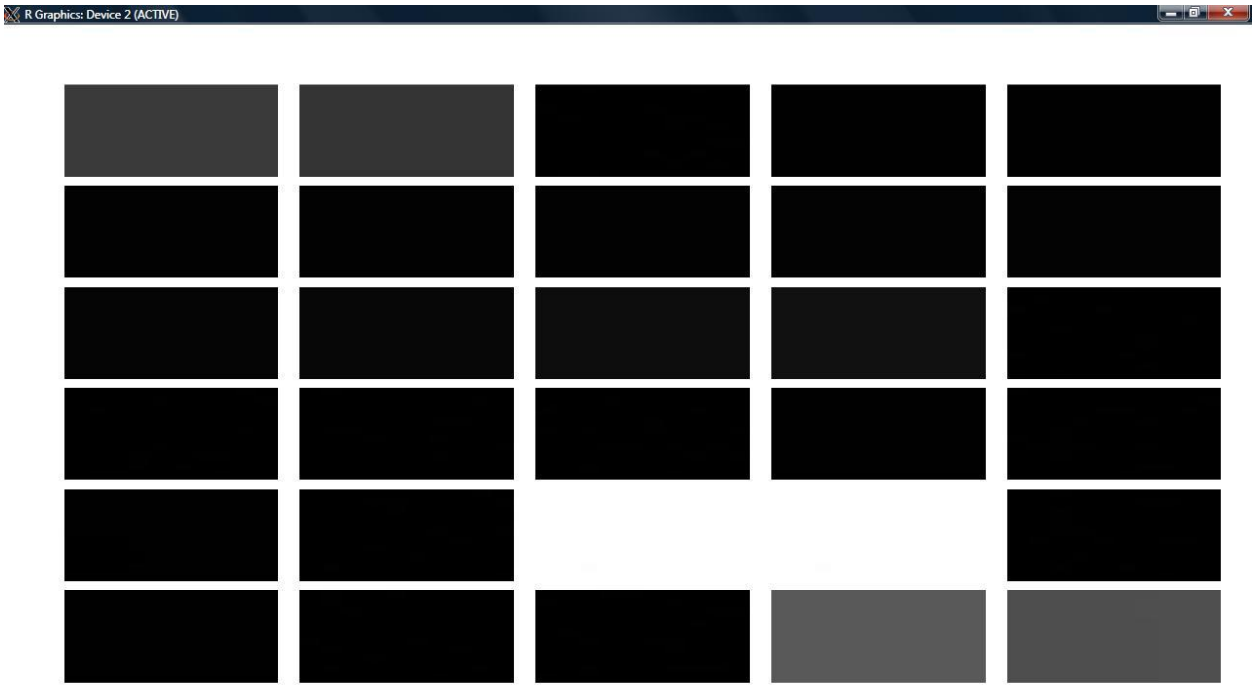


Figure 2.5.3: Visualization of learned **w** vectors for 30 nodes. The algorithm converged at 247th iteration.



From above figures it is clear that SOM algorithm implementation is getting stucked at local maxima. The model vectors corresponding to white patches in above figures have largest values with respect other model vectors for all the nodes. This indicates that algorithm reaches a local maxima due bad initialization of model vectors. However, seeing the pattern of learned **w** vectors this establishes that SOM reorders these model vectors based on decreasing similarity values.

Exercise2.6:

Aim: Set the number of model vectors to 20 and then investigate the changes when:

1. Rescaling to unit variance is omitted during preprocessing
2. Deduction of average value from the patches is removed during preprocessing.

Results:

When the rescaling is omitted during preprocessing, the following output was obtained:

Figure 2.6.1: Visualization of learned **w** vector when variance rescaling was omitted.

The SOM algorithm converged after running for 500 iterations. With omitting the variances from the data matrix during preprocessing step, the resultant image patches are not uniform related to their intensities as compared to figures of exercise2.5 where uniform intensities patches were obtained. However, the SOM still works and we can see model vectors **w** ordered according to their decreasing similarity values.

When subtraction of mean from the patches data matrix is omitted during preprocessing step, an array of 20 patches is obtained with intensities having some uniformity. However, the patches get clustered based on decreasing intensities value as shown in **Figure 2.6.2**. On inspecting the values of learned model vector **w,** it was found that values starts decreasing from **w1 to w18** and then it starts increasing. Overall, on basis of uniformity of intensities of patches they resemble similar to figures of exercise2.5 where data was processed.

Figure 2.6.2: Visualizing learned w vector with 20 nodes. The data matrix was processed without rescaling it to unit variance.