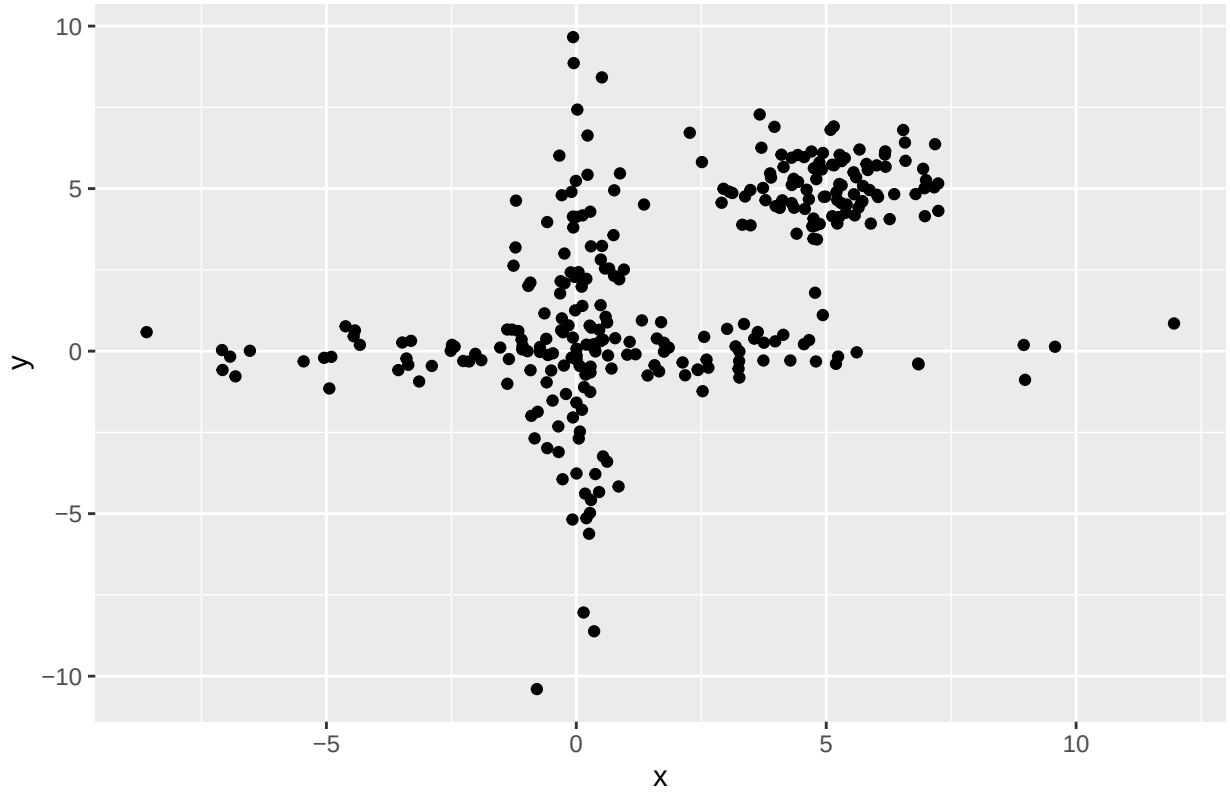
Labb2

Anton Holm

2021-02-15

Part A

Figure 1: Plot over the data



In Figure 1 we can see that we have a few outliers in the data set. However, they don't seem to be all too extreme and as such I begin to check if a KNN graph could be used. I use the `mstknnclust` packages only to check for a good number of neighbors since they use the same method as suggested in class, i.e. testing for each value $K = 1, 2, ...$ until we have a completely connected graph. However, the output of the edges and their weight are in a bad format and as such, I write my own code to produce the graph, weight matrix etc, using this value of K.

The function `generate.knn` from the package `msknnclust` gives us the given $K = 5$. Since this K is not too large, we will have a completely connected graph without any major problems with short circuits and as such, I will stick with normal KNN. We can also see in Figure 2 that the second eigenvalue is small, but not zero and as such we have a connect graph, may it be a soft connection. However, one could also consider mutual-KNN in this scenario since we do not seem to have any extreme outliers. To produce the similarity

matrix in this case I used euclidean distance. This is reasonable since we manage a small K and as such, we will mostly connect to very local vertices and any surface looked open very locally is "flat" and we will not have any short circuits due to the local connections. Thus I also chose the weight to be $1/||x_i - x_j||^2$ using euclidean distance. We could have also chosen the gaussian kernel, but then we also need to search for the optimal σ which gives us so many options to chose from and the probability of getting a good match is much lower.

Below we produce the unnormalized lapplacian and perform the algorithm for unnormalized spectral clustering. In plotting the data in their new representation y_i I use the two eigenvectors that have the two smallest eigenvalues (not including the vector connected to the eigenvalue equal to zero since this does not include any information regarding cluster structure). However, as we can see in Figure 2, we have an eigengap between the second and third smallest eigenvalue which gives us a hint that we could plot the data in one dimensions instead, only using the eigenvector connected to the smallest eigen value seperated from zero. We plot the data using both eigenvectors (Figure 3) and we plot the data using only the eigenvector with the smallest eigenvalue (Figure 4). We can see in Figure 3 it looks like we might have 3 clusters which might give us problem with using K-mean with the assumption of only 2 clusters. It is also hard to know if we are in the euclidean space in this case which is one of the biggest assumptions needed for the K-means algorithm to work at all. However, if we look at Figure 4, first of all we are in 1 dimension and ofcourse hence we are working with euclidean distance. The data does not exactly resemble gaussian which is another assumption needed for K-mean. However, the distance between the two clusters are quite large (compared to within clusters) and we are in the euclidean space which speaks in favor of the K-mean algorithm. One problem however is that the data is very unbalanced which could make problems for the K-mean algorithm. At the same time, the data is clearly linearly seperable. I try the K-mean algorithm assuming 2 clusters in the 2D space and see that the number of data in each cluster is almost the same (see Table 1). This is not the case for the original data and as such the K-mean algorithm has failed. However would we instead use other combinations of eigvevectors the result could have been different. However, since working in one dimension actually is enough in this case I will do so instead of searching for combinations of eigenvectors. For the one dimensional case I plot the result of the k-mean algorithm in Figure 5. Clearly the algorithm works in this projected space and we see in Figure 6 that the K-mean clustering algorithm has separated the data perfectly in the original space.

Figure 2: Eigenvalues of unnormalized laplacian

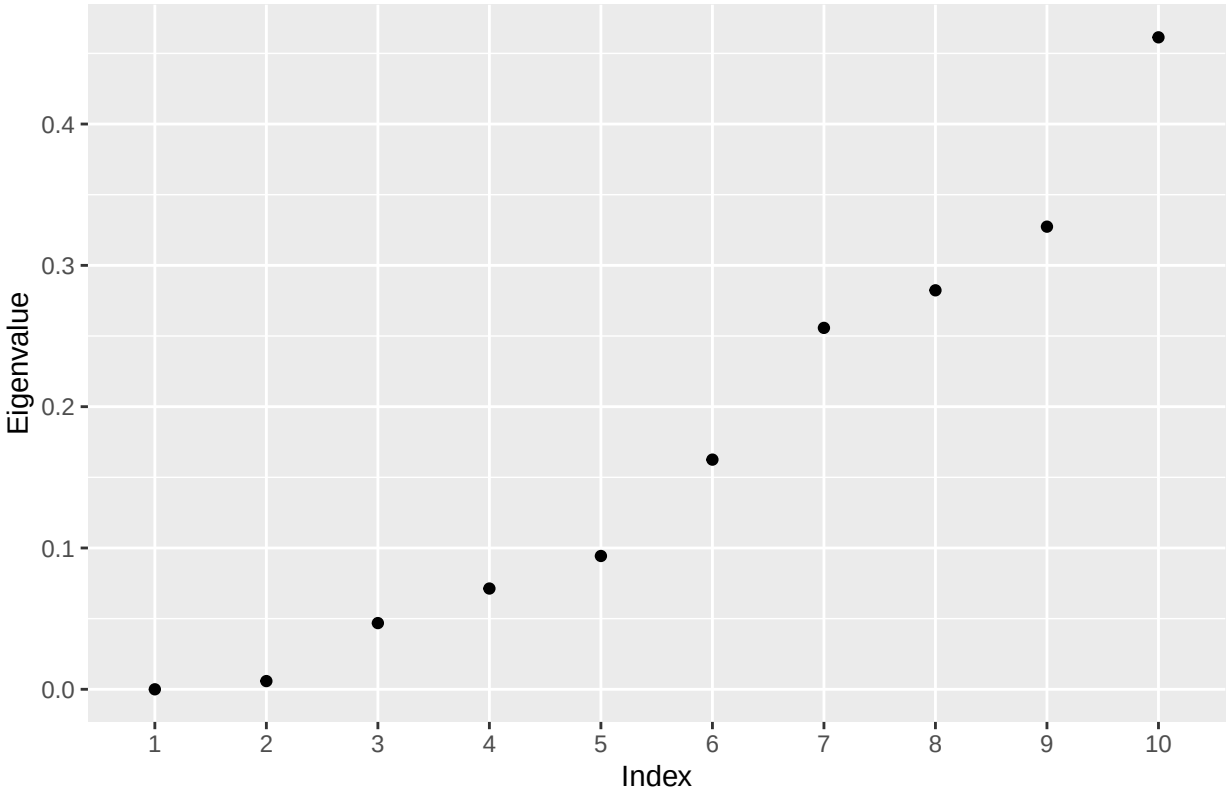


Figure 3: Data plotted using eigenvectors

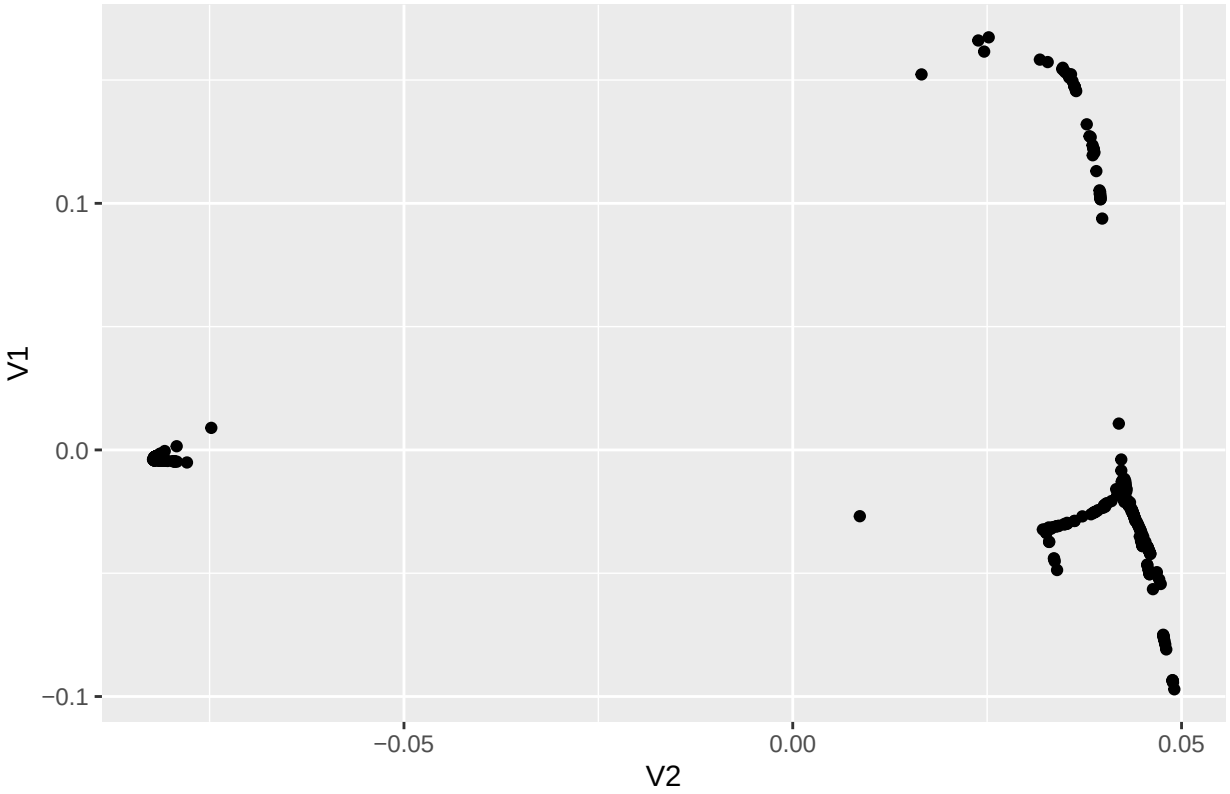
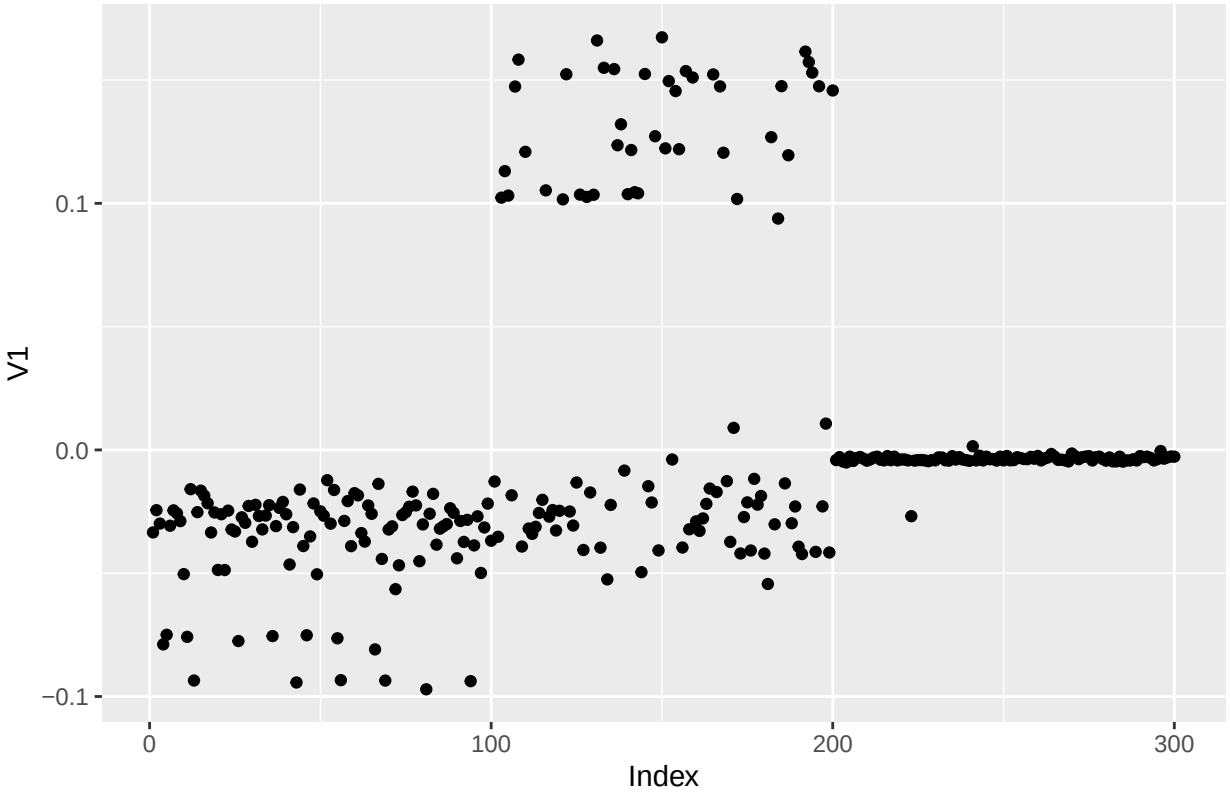


Figure 4: Data plotted using a single eigenvector




```
## Warning in kable_pipe(x = structure(c("Cluster 1", "143", "Cluster2", "157": The
## table should have a header (column names)
```

Table 1: Table 1: Number of data in each cluster

Cluster 1	Cluster2
143	157

Figure 5: Result of Kmean on data using one eigenvector

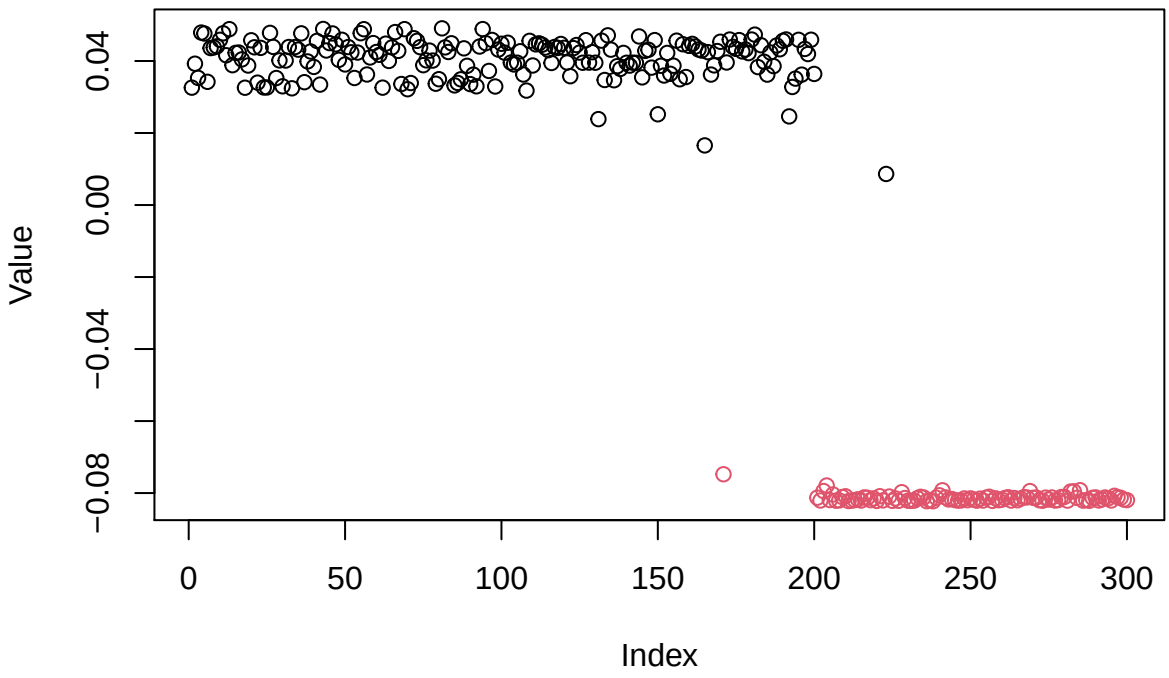


Figure 6: Result of Kmean on data reprojected onto original space

Now we perform the normalized spectral clustering algorithm using the same similarity matrix as before by the same agruments as before. The results looks quite similar to when using the unnormalized spectral clustering algorithm and as such all the above arguments also apply here.

Figure 7: Eigenvalues of Normalized Symmetric Laplacian

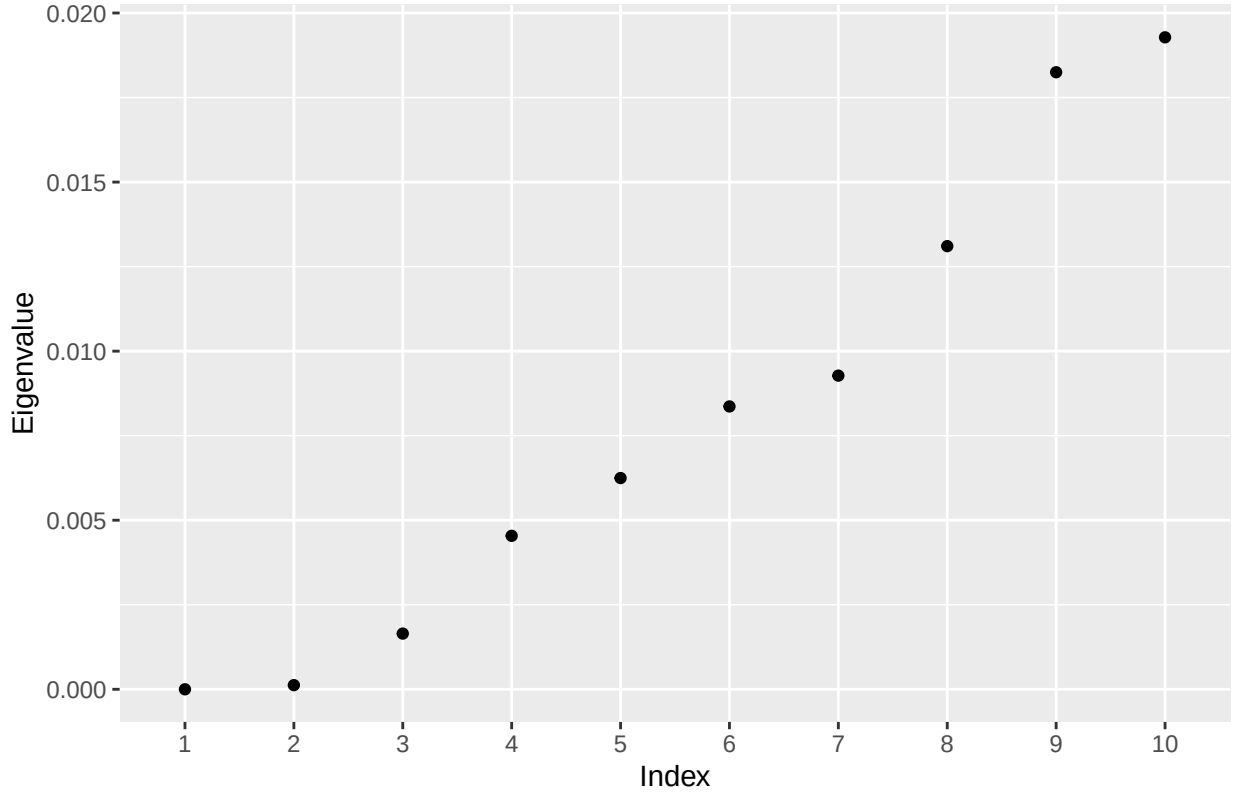


Figure 8: Data plotted using a single eigenvector of L_Sym

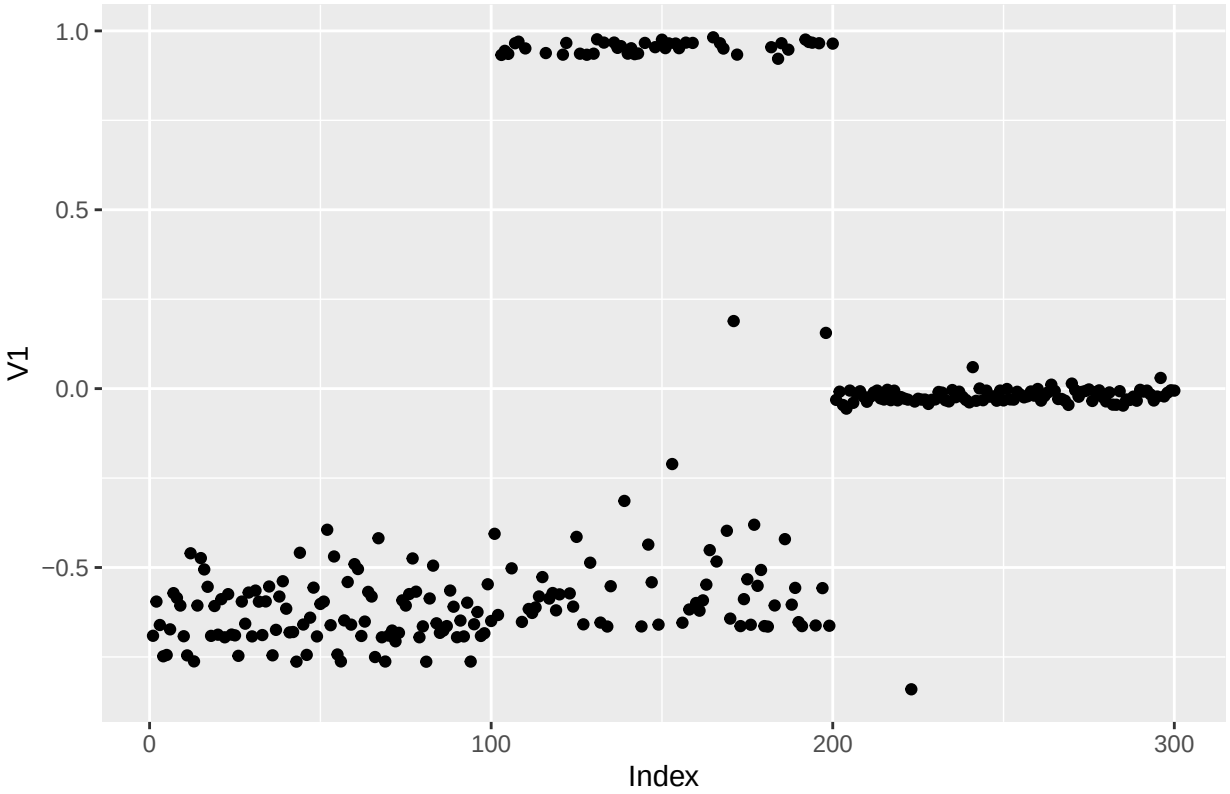


Figure 9: Result of Kmean on data using one eigenvector of L_Sym

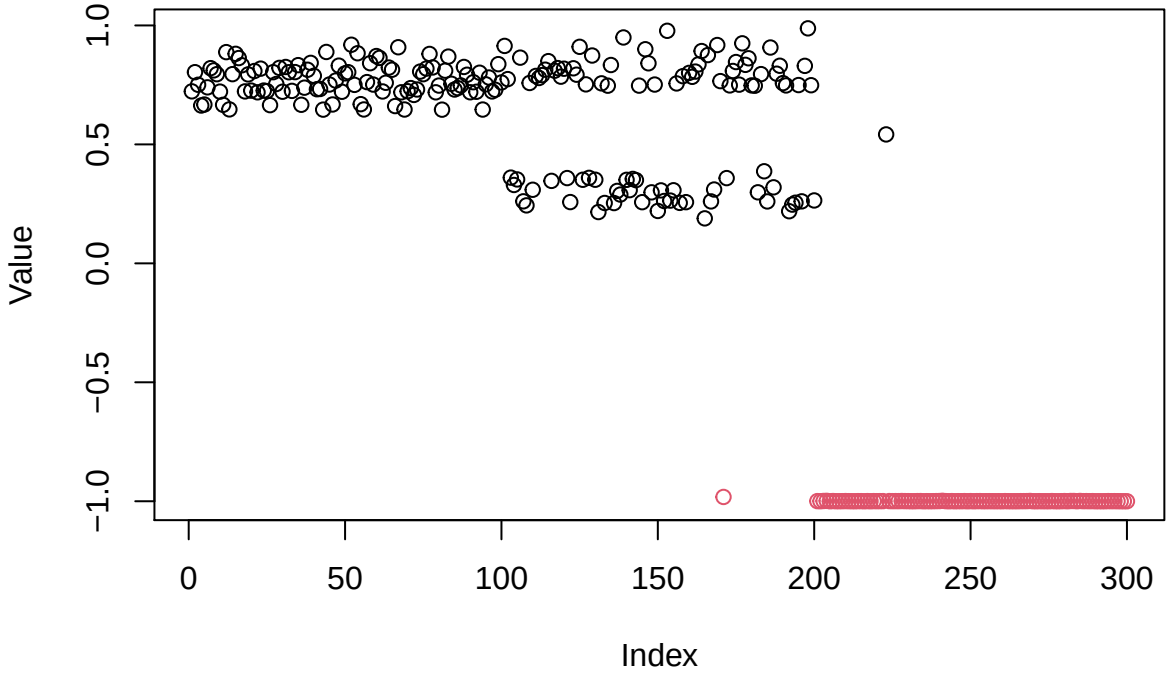


Figure 10: Result of Kmean on data reprojected onto original space

As we can see, both the unnormalized Laplacian L and the symmetric normalized Laplacian L_{Sym} worked well to separate the clusters. However this is a simple example and for more complex data we should consider the normalized version since we want the degree of vertices to matter. The unnormalized case does not take this into consideration but treat each data point equally, no matter of its degree. We can actually see by comparing Figure 4 and Figure 8 that the between cluster distance is drastically increased for the normalized case (keep in mind the ratio of the y axis) which is also a reason to why it is preferable to use the normalized laplacian.

Part B

I follow the same method as for Part A in creating the graph and see that the number of neighbors K becomes equal to 4. However, when I look at the eigenvalues of the Laplacian I see that we still have 2 eigenvalues equal to zero which mean we are most likely not fully connected. Hence I try to increase K to be equal to 5 and get a distinct eigengap. In Figure 11 the inversed eigenvalues are shown since this represents the variation in different directions of the CTD embedding (new x-coordinates from class). This is still a small K as in Part A and hence we should not have problems with short circuits and we can assume a locality that implies euclidean distance when calculating the distance. Therefor I continue to use normal KNN graphs, in this case with $K = 5$. In Figure 12 we see the swiss roll when projected using the smallest and fifth smallest eigenvectors not counting the one with eigenvalue equal to zero. It look close to an hourglass but we can see the four corners most likely ressembling the four corners of the swiss roll. The large bubbles/holes are to no surprice since we only have 2000 datapoints which is not enough when trying to unfold this complex structure into two dimensions. However we do have a slight problem in the right curvature in the middle of the figure. Since using the CTD embedding results in us having euclidean distance, it looks like two points are close even though they lie on different sides of this curve. This is not something we want in our projection but it was the best embedding I could find after searching through all combinations of the eigenvectors with

the 10 smallest eigenvalues. The reason (at least one of them) that the swiss roll in this case is not unfolding into a square or rectangle is due to the ratio of the swissroll. The length of the swiss roll is much larger than that of the width which makes it though to unfold it perfectly.

Figure 11: Eigenvalue of unnormalized laplacian

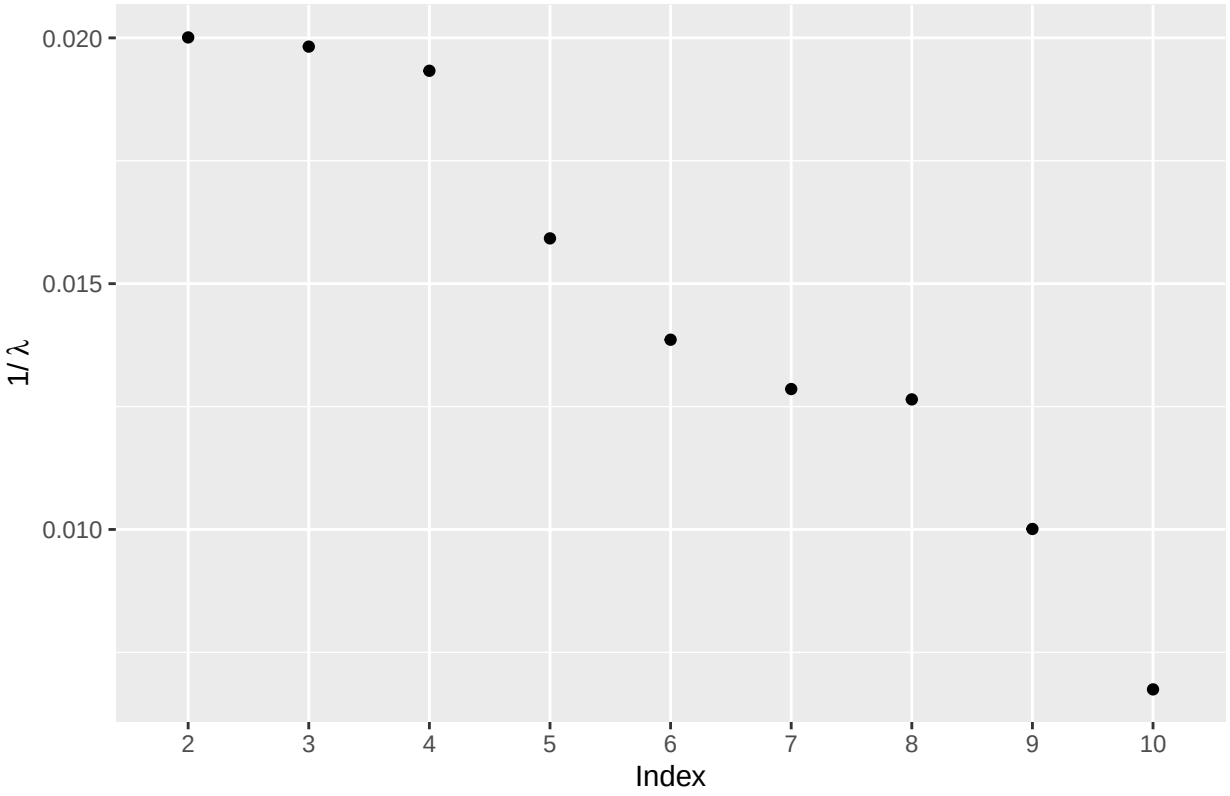
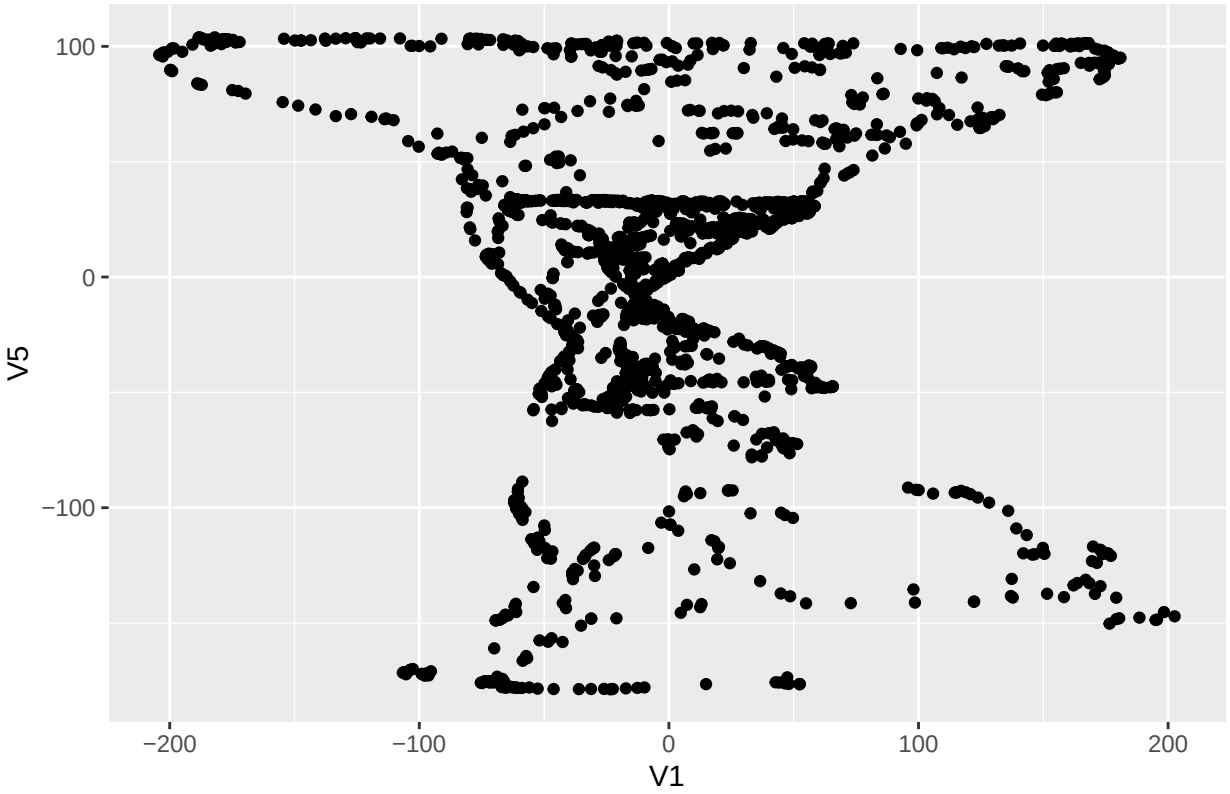


Figure 12: Swissroll on CTD embedding (unnormalized)



Below we construct the CTD embedding for the normalized case. In Figure 14 we once again can see the four corners of the swiss rollin the bottom and upper left and right corners of the figure. In this case it looks like we don't have such a large problem with intrusion as we did in the unnormalized case and the data is more compact, i.e. they are mapped onto a more compact surface. In the case of CTD embedding the same reasoning follows from Task A that we prefer to use the normalized symmetric laplacian over the unnormalized laplacian since then we will take degrees of points into consideration.

Now in order to really test which two eigenvectors to use and which method is the best we should do some validation. One way we can validate our results is by using the shepard rank diagram as in the previous project, especially in this case to check if we have a problem with intrusion, something we want to minimize. Yet another argument to use the normalized laplacian is that the eigenvalues are not changing too much if we would say use half of the data points instead. The same can not be said for the unnormalized laplacian.

Figure 13: Eigenvalues for the normalized Laplacian

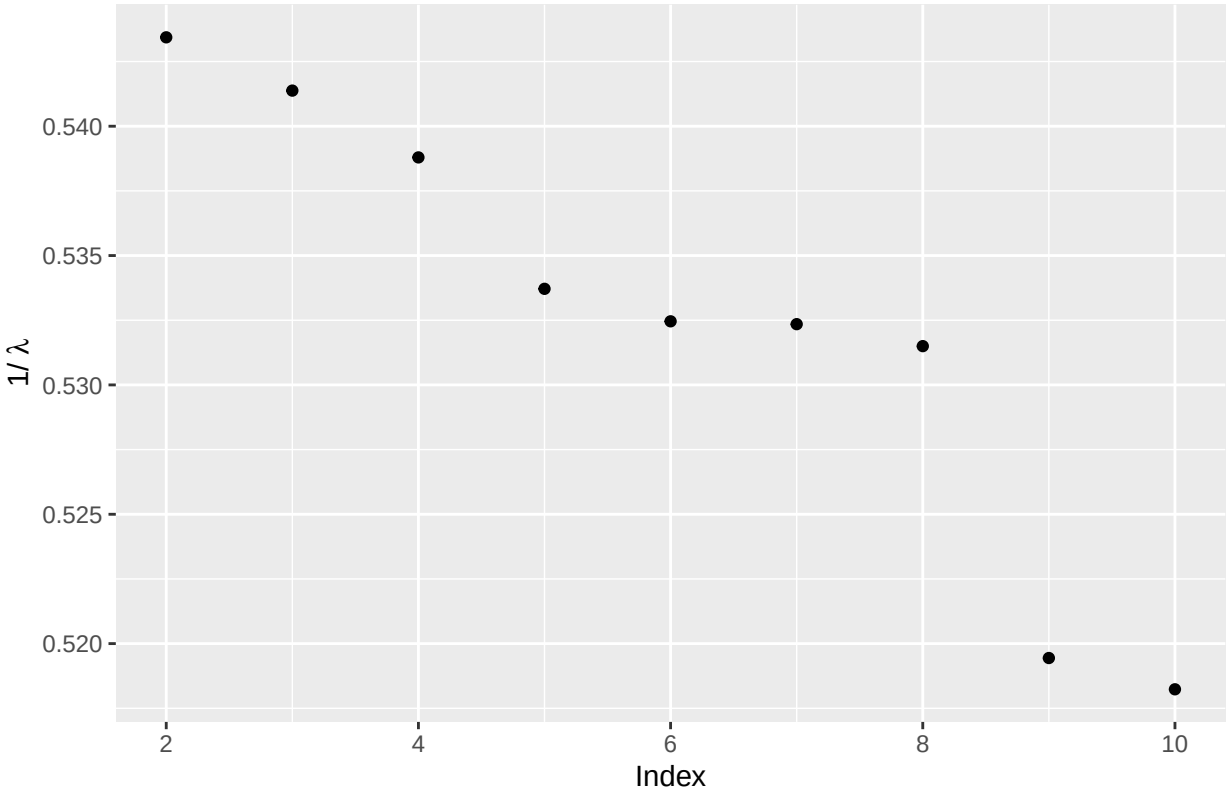
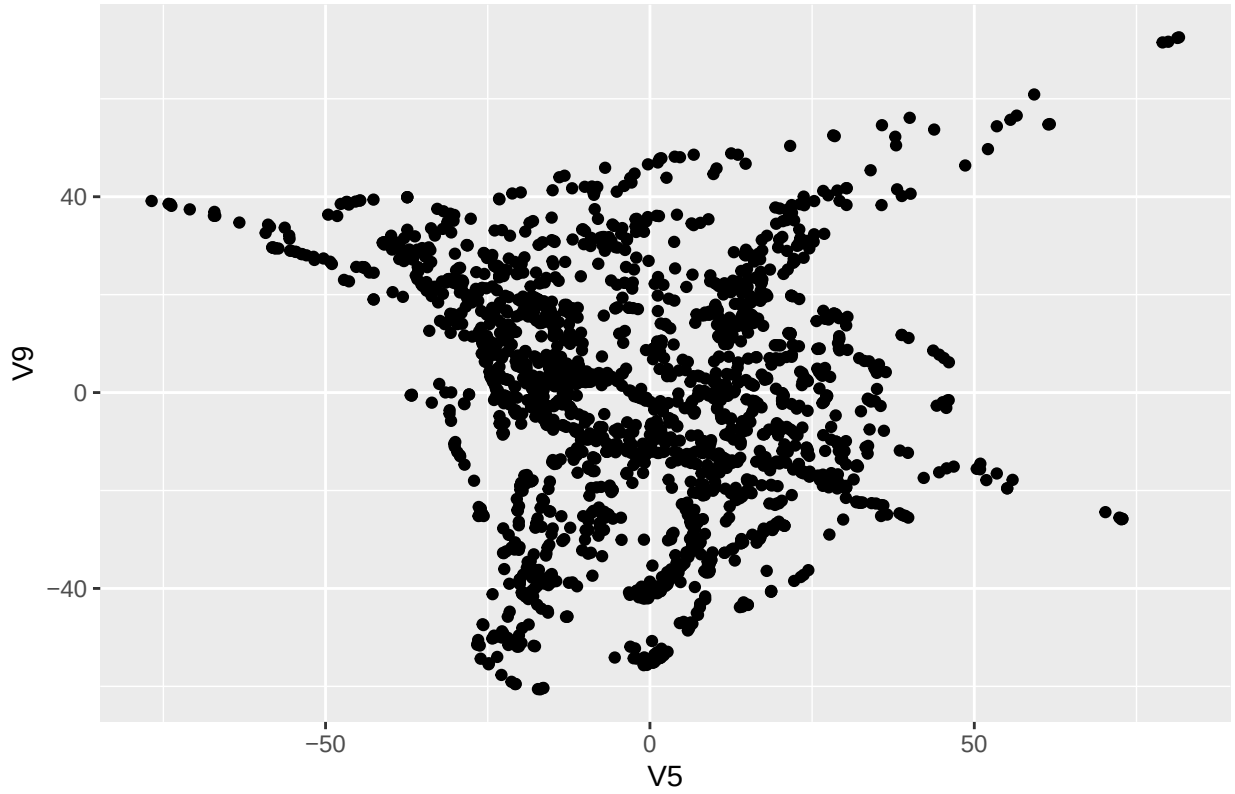


Figure 14: Swissroll on CTD embedding (normalized)



Code Chunks

Code for plotting the star data

```
set.seed(931031)
#Loads the data
star_df <- read.table("Star_Data.txt", header = FALSE, sep = " ") %>%
  select(V1, V5) %>%
  rename('x' = V1, 'y' = V5)

#Plots the data
star_df %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  labs(title = "Figure 1: Plot over the data")
```

Code to construct Graphs

```
####################
# Creating similarity matrix and find number of neighbors to use
####################

#Construct the similarity matrix
star_dist <- as.matrix(dist(star_df, method = "euclidean"))
```

```r
#Construct weight matrix
star_sim <- star_dist %>%
  as.data.frame() %>%
  mutate_all(~((1/.)^2)) %>%
  mutate_all(~ifelse(. == Inf, 0, .)) %>%
  data.matrix()
#Create a complete graph
star_graph <- generate.complete.graph(1:nrow(star_df), star_dist)
#Create the KNN graph, K chosen as the minimum K where the graph is completely connected
star_knn <- generate.knn(star_graph)
#Gives optimal K based on the criteria above
opt_k = star_knn$k

#Creates the weight matrix
#Use the edge weight as 1/||x_i - x_j||^2 as suggested in class as to not have to also find a suitable
#Ref to Code: http://www.di.fc.ul.pt/~jpn/r/spectralclustering/spectralclustering.html
make.affinity <- function(S, n.neighboors) {
  N <- length(S[,1])
    A <- matrix(rep(0,N^2), ncol=N)
    for(i in 1:N) { # for each line
      # only connect to those points with larger similarity (exclude yourself)
      best.similarities <- sort(S[i,])[2:n.neighboors+1]
      for (s in best.similarities) {
        j <- which(S[i,] == s)
        #To know which edges to keep. Weights are added later
        A[i,j] <- 1
        A[j,i] <- 1 # to make an undirected graph, i.e, the matrix becomes symmetric
      }
    }
  return(A)}

W_rough <- make.affinity(star_dist, 5)
```

Code for unnormalized spectral clustering part

```r
######################
# Calculates the unnormalized laplacian and perform the spectral cluster analysis.
######################

#Constructs the adjacency matrix
W <- star_sim * W_rough
W[is.nan(W)] <- 0
#Constructs the degree matrix
D <- colSums(W) %>% diag()
#Unnormalized Laplacian
L_unnorm <- D - W
#Computes Eigenvalues and Eigenvectors of the Unnormalized Laplacian
ev <- eigen(L_unnorm, symmetric = TRUE)
eig_vec <- ev$vectors[,298:299] %>% as.data.frame()
eig_val <- ev$values[298:299]

ev$values %>%
  tibble(eig = .) %>%
```

```
  arrange(eig) %>%
  slice_head(n = 10) %>%
  ggplot(aes(x = factor(1:10), y = eig)) +
  geom_point() +
  labs(title = "Figure 2: Eigenvalues of unnormalized laplacian", x = "Index", y = "Eigenvalue")

ggplot(data = eig_vec, aes(x = V2, y = V1)) +
  geom_point() +
  labs(title = "Figure 3: Data plotted using eigenvectors")

ggplot(data = eig_vec, aes(x = 1:300, y = V1)) +
  geom_point() +
  labs(title = "Figure 4: Data plotted using a single eigenvector", x = "Index")

#Performs Kmean
kmean <- kmeans(eig_vec, centers = 2)
nreachclust <- rbind(c("Cluster 1", "Cluster2"), kmean$size)
knitr::kable(nreachclust, caption = "Table 1: Number of data in each cluster")
kmean2 <- kmeans(eig_vec$V2, centers = 2)
plot(eig_vec$V2, col = kmean2$cluster, main = "Figure 5: Result of Kmean on data using one eigenvector"

star_df %>%
  mutate(cluster = kmean2$cluster) %>%
  ggplot(aes(x = x, y = y, col = as.factor(cluster))) +
  geom_point() +
  labs(title = "Figure 6: Result of Kmean on data reprojected onto original space")
```

Code for normalized spectral clustering part

```
#################
# Normalized Spectral Clustering
################

#Calculates the Symmetric Normalized Laplacian
L_Sym <- sqrt(solve(D)) %*% L_unnorm %*% sqrt(solve(D))

#Computes Eigenvalues and Eigenvectors of the Unnormalized Laplacian
ev2 <- eigen(L_Sym, symmetric = TRUE)
eig_vec2 <- ev2$vectors[,298:299]
eig_val2 <- ev2$values[298:299]

ev2$values %>%
  tibble(eig = .) %>%
  arrange(eig) %>%
  slice_head(n = 10) %>%
  ggplot(aes(x = factor(1:10), y = eig)) +
  geom_point() +
  labs(title = "Figure 7: Eigenvalues of Normalized Symmetric Laplacian", x = "Index", y = "Eigenvalue"

#Normalize row sums to have norm 1
summing_rows <- sqrt(rowSums(eig_vec2^2))
U <- eig_vec2/summing_rows
```

```r
ggplot(data = as.data.frame(U), aes(x = 1:300, y = V1)) +
  geom_point() +
  labs(title = "Figure 8: Data plotted using a single eigenvector of L_Sym", x = "Index")

#Performs Kmean
kmean3 <- kmeans(as.data.frame(U)$V2, centers = 2)
plot(as.data.frame(U)$V2, col = kmean3$cluster, main = "Figure 9: Result of Kmean on data using one eig


star_df %>%
  mutate(cluster = kmean3$cluster) %>%
  ggplot(aes(x = x, y = y, col = as.factor(cluster))) +
  geom_point() +
  labs(title = "Figure 10: Result of Kmean on data reprojected onto original space")
```

Code for CTD Embedding using unnormalized laplacian

```r
#########################
#CTD Embedding using the unnormalized Laplacian
#########################

swiss_df <- read.table("Swiss_Roll.txt") %>%
  rename(x = V1, y = V2, z = V3)

#Construct the similarity matrix
swiss_dist <- as.matrix(dist(swiss_df, method = "euclidean"))
#Construct weight matrix
swiss_sim <- swiss_dist %>%
  as.data.frame() %>%
  mutate_all(~((1/.)^2)) %>%
  mutate_all(~ifelse(. == Inf, 0, .)) %>%
  data.matrix()

#Create a complete graph
swiss_graph <- generate.complete.graph(1:nrow(swiss_df), swiss_dist)
#Create the KNN graph, K chosen as the minimum K where the graph is completely connected
swiss_knn <- generate.knn(swiss_graph)
#Gives optimal K based on the criteria above
opt_k = swiss_knn$k

W_rough_swiss <- make.affinity(swiss_dist, 5)

#Constructs the adjacency matrix
W_swiss <- swiss_sim * W_rough_swiss
W_swiss[is.nan(W_swiss)] <- 0

#Constructs the degree matrix
D_swiss <- colSums(W_swiss) %>% diag()

#Unnormalized Laplacian
L_unnorm_swiss <- D_swiss - W_swiss

vol_swiss <- sum(colSums(W_swiss))
```

```r
ev_swiss <- eigen(L_unnorm_swiss, symmetric = TRUE)
eigval_swiss <- ev_swiss$values
eigvec_swiss <- ev_swiss$vectors

#Eigvalues
ggplot(as.data.frame(eigval_swiss) %>% arrange(eigval_swiss) %>% slice(1991:1999), aes(x = as.factor(2:
  geom_point() +
  labs(title = "Figure 11: Eigenvalue of unnormalized laplacian", x = "Index", y = bquote("1/" ~ lambda

# Calculates the new CTD embedding (x-coordinates from class)
L_unnorm_swiss_xcoord <- sqrt(vol_swiss)/sqrt(eigval_swiss[c(1995,1999)]) * eigvec_swiss[,c(1995,1999)]

ggplot(data = as.data.frame(L_unnorm_swiss_xcoord) %>% rename(V5 = V2), aes(x = V1, y = V5)) +
  geom_point() +
  labs(title = "Figure 12: Swissroll on CTD embedding (unnormalized)")
```

Code for CTD Embedding using the normalized laplacian

```r
####################
# CTD Embedding using the symmetric normalized Laplacian
####################

L_norm_swiss <- sqrt(solve(D_swiss)) %*% L_unnorm_swiss %*% sqrt(solve(D_swiss))

ev_swiss2 <- eigen(L_norm_swiss, symmetric = TRUE)
eigval_swiss2 <- ev_swiss2$values
eigvec_swiss2 <- ev_swiss2$vectors

#Eigen values
ggplot(as.data.frame(eigval_swiss2) %>% arrange(eigval_swiss2) %>% slice(1991:1999), aes(x = 2:10, y =
  geom_point() +
  labs(title = "Figure 13: Eigenvalues for the normalized Laplacian", x = "Index", y = bquote("1/" ~ la

#CTD Embedding
L_unnorm_swiss_xcoord2 <- sqrt(vol_swiss) * 1/sqrt(eigval_swiss2[c(1991,1995)]*colSums(W_swiss)) * eigv
ggplot(data = as.data.frame(L_unnorm_swiss_xcoord2) %>% rename(V5 = V1, V9 = V2), aes(x = V5, y = V9)) +
  geom_point() +
  labs(title = "Figure 14: Swissroll on CTD embedding (normalized)")
```