Machine Learning 2ST129 26605 HT2023 Assignment 6

Anonymous Student

December 17, 2023

Contents

General Information

- Time used for reading: 3 hours
- Time used for basic assignment: 18 hours
- Time used for extra assignment: 10 hours
- Good with lab: It was good that generally for the different methods that the functions were implemented as separate function and then combined in the end, which makes it easier to understand whats going on.
- Things to improve with lab: I think in general there were too many questions to answer, although most of them were relatively simple it just felt that some of the things did not really connect and I just did some of them without really reflecting about them . Maybe less but more difficult questions would have been better.

```r
#Libraries
 library(tidyverse)
 library(xtable)
 library(tensorflow)
 library(keras)
 library(bsda)
```

```r
data("iris")
data("faithful")
library(uuml)
data("mixture_data")
```

1 Task 1

1.1 1.1

First we just visualize the faithful data

```r
plot(faithful)
```

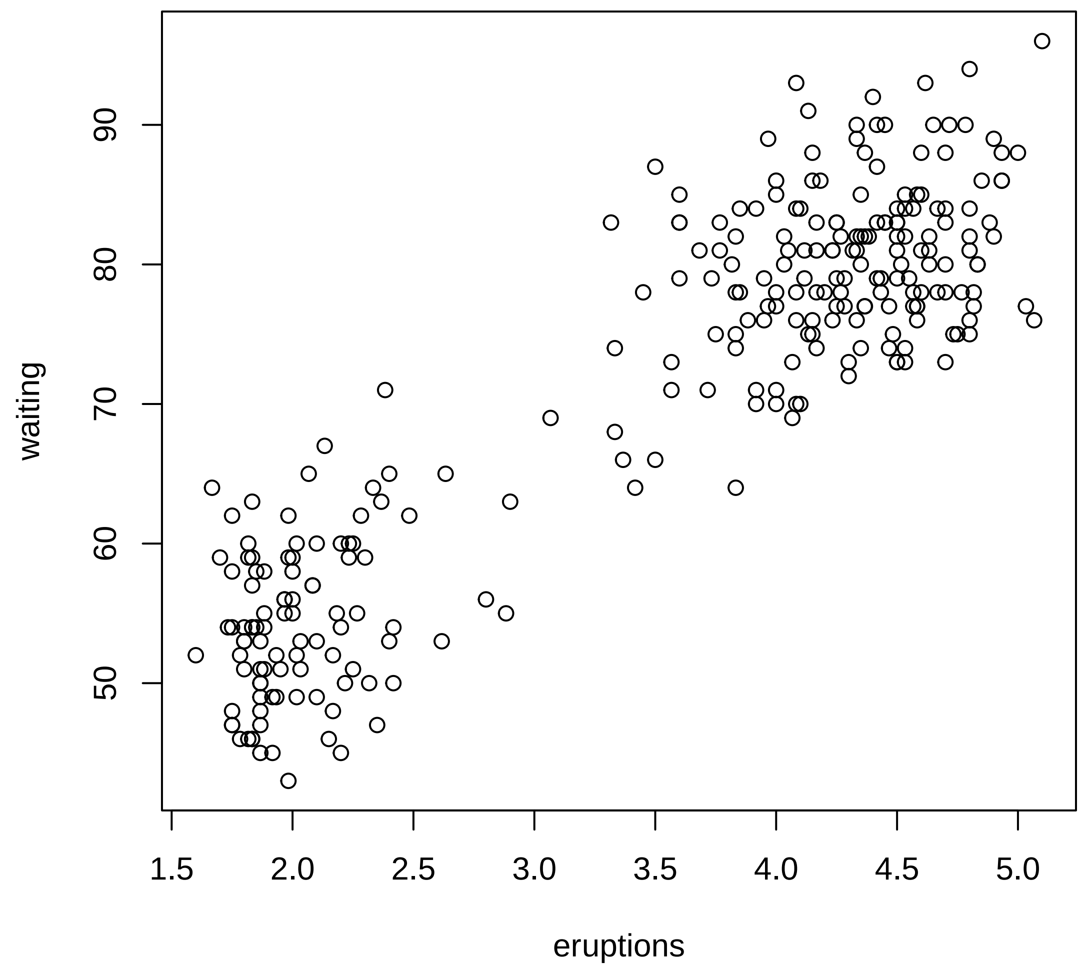


Figure 1: faithful data

Visually, there seems to be two clusters that I can think of in the corners.

1.2 1.2

Now we visualize the iris data set with the species as color.

```
iris %>%
  ggplot() +
  geom_point(aes(x=Petal.Length, y= Petal.Width, color=Species))
```

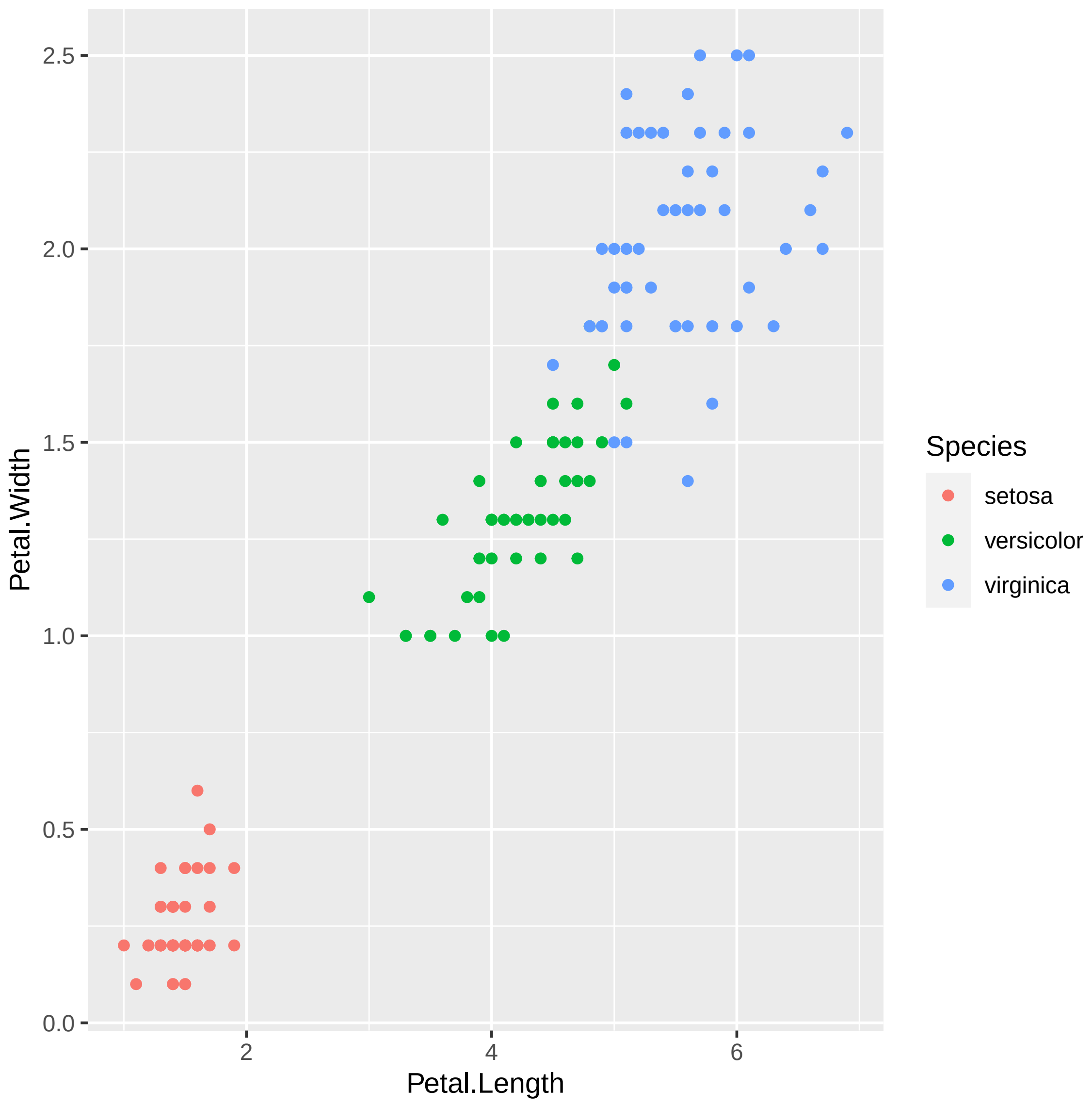


Figure 2: iris data

1.3 1.3

Now we visualize the faithful data again with the hypothesized clusters as colors. Here i based on on eruptions <= 3

```
index <- faithful$eruptions <= 3
plot(faithful)
points(faithful$eruptions[index], faithful$waiting[index], col="red")
points(faithful$eruptions[!index], faithful$waiting[!index], col="blue")
```

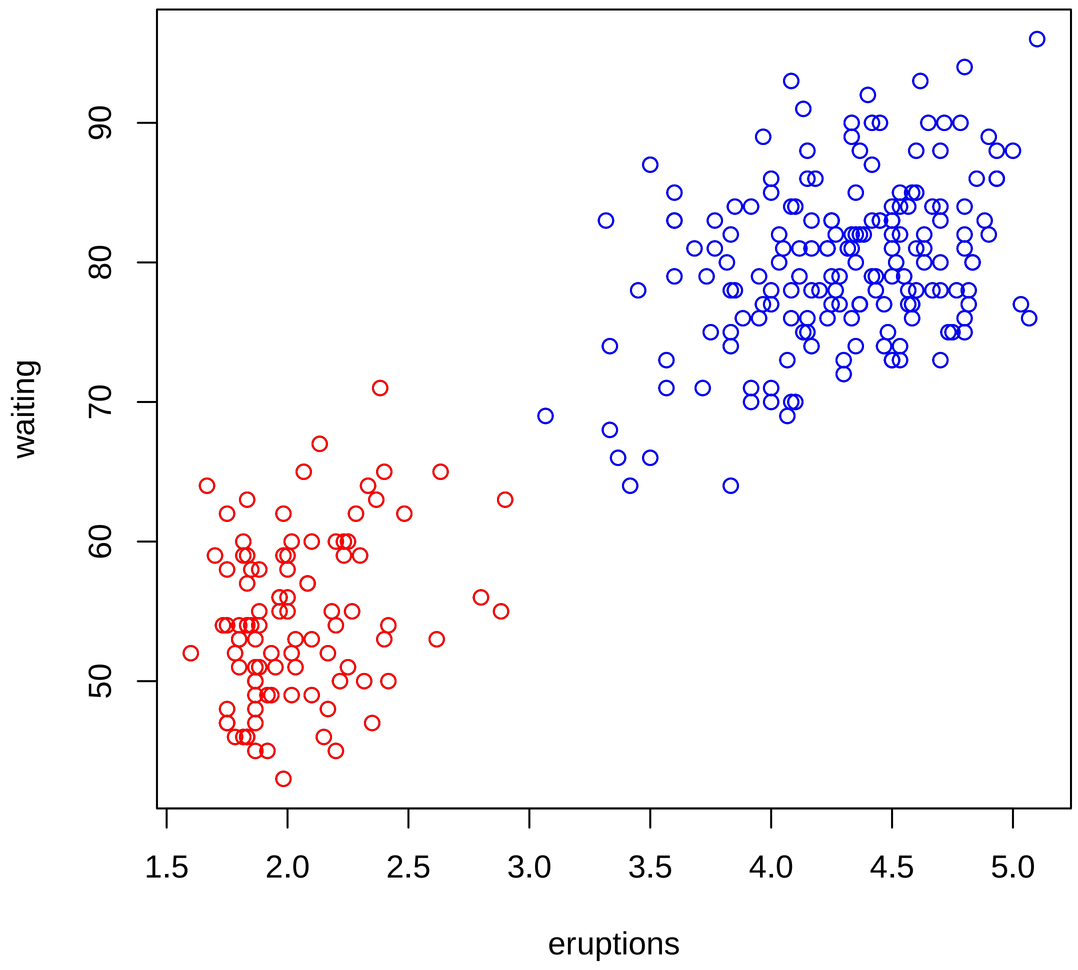


Figure 3: Hypothesized clusters for eruptions

1.4 1.4

Now to implement the different parts of the k-means algorithm. First step 1.

```r
#' @param X n X p matrix
#' @param c p x 1 vector of cluster assignments
#' n is the number of rows, p is the number of cols
compute_cluster_means <- function(X, C) {
  means <- t(sapply(sort(unique(C)), FUN = function(i){
    index <- which(C == i)
    colMeans(X[index,])
}))
colnames(means) <- colnames(X)
return(means)
}

set.seed(4711)
X <- as.matrix(faithful)
C <- sample(1: 3, nrow(X), replace = TRUE)
```

```
m <- compute_cluster_means(X,C)
m

     eruptions  waiting
[1,]  3.416354 70.57576
[2,]  3.571106 71.44706
[3,]  3.487659 70.72727
```

1.5 1.5

Now to implement the second step.

```
#function to computer the squared distance given cluster means for one obs
#assuming the number of columns match .
cluster_dist <- function(x_i, cluster_means){
  res <- t(apply(cluster_means, MARGIN=1, FUN= function(row){x_i - row}))
  sums <- rowSums(res^2)
  min_cluster <- which.min(sums)
  return(min_cluster)
}

#' @param X n×p (design) matrix X
#' @param m  K×p matrix m with one cluster mean per row
#' @return  n × 1 vector of cluster assignments
compute_cluster_encoding <- function(X, means) {
new_clusters <- sapply(1:nrow(X), FUN = function(i){ cluster_dist(X[i,], means)})
return(new_clusters)
}

C <- compute_cluster_encoding(X, m)
C[1: 10]

 [1] 2 1 2 1 2 1 2 2 1 2
```

1.6 1.6

Now we use the functions to implement algorithm 14.1 in Hastie et al

```
k_means <- function(X,k){
  clusters <- sample(1:k, nrow(X), replace=TRUE) #initialize cluster assignments
  while(TRUE){
      c_means <- compute_cluster_means(X, clusters)
    new_clusters <- compute_cluster_encoding(X, c_means)
    if(identical(new_clusters, clusters)){
      break
    }
    clusters <- new_clusters
  }
  return(clusters)
}
```

1.7 1.7

Now to implement function to compute the k-means withing-point scatter

```
set.seed(4711)
X <- as.matrix(faithful)
C <- sample(1: 3, nrow(X), replace = TRUE)

k_means_W <- function(X,C){
  c_means <- compute_cluster_means(X,C)
   within_cluster_sums<- sapply(1:nrow(c_means), FUN=function(i){
    current_cluster_m <- c_means[i, ]
    index <- which(C == i)
    x_subset <- X[index,]
     cluster_distances <- apply(x_subset, 1,
                                function(row) sum((row - current_cluster_m)^2))
     sum(cluster_distances)
})
   #cluster_distances are the unweighted sums within each cluster
   #hence here we weight them by the cluster size which are sorted in order
   aggregated_res <- sum(within_cluster_sums * table(C))
   return(aggregated_res)
}


k_means_W(X,C)

[1] 4601439
```

1.8 1.8

Now we run it a couple of times for both faithful and iris with K=2 and k=5. First we just create a wrapper function that will be able to be used in a `mapply()` function. Then we first start with the faithful data

```
#wrapper function to perform the k_means and k_means_W
#disclaimer: i know usually we should not set seed inside functions, but since
# the goal here is to explicitly test different seeds i put in the the function
#to make it easier to test
k_means_wrapper <- function(seed, X, k ){
  set.seed(seed)
  result <- k_means(X,k)
  wps <- k_means_W(X, result)

  list <- list("Clusters" = result, "WPS" = wps)
  return(list)

}

#here we use hte k_means_wrapper 10 times where the arguments are each of the
# ordered pairs. So for the first 5 times we have 2 clusters and then 3
# clusters for the last 5. Each time a new seed but same X data.

multi_res_faithful <- mapply(FUN= k_means_wrapper,
                             seed=c(1,2,3,4,5,6,7,8,9,10), X=list(X),
                            k=c(rep(2,5), rep(3,5)), SIMPLIFY = FALSE)
merged_res <- do.call(rbind, multi_res_faithful)

merged_res

      Clusters    WPS
 [1,] integer,272 1282259
 [2,] integer,272 1282259
 [3,] integer,272 1282259
 [4,] integer,272 1282259
 [5,] integer,272 1282259
 [6,] integer,272 543213
 [7,] integer,272 532868.4
 [8,] integer,272 577257.4
 [9,] integer,272 543213
[10,] integer,272 532868.4

#the clusters column contains the cluster indexes for each obs.

which.min(merged_res[,2])

[1] 7

which.max(merged_res[,2])

[1] 1
```

Looking at the results for WPS, then we see that the minimum WPS is for the seventh row, whereas the maximum value is the same for all the first 5 rows corresponding to the results which only used 2 clusters.

Next we do the same thing for this iris data.

```
iris_data <- as.matrix(iris[,-5])
multi_res_iris <- mapply(FUN= k_means_wrapper,
                         seed=c(11,12,13,14,15,17,18,19,20,22),
                         X=list(iris_data), k=c(rep(2,5),rep(3,5)),
                         SIMPLIFY = FALSE)

merged_res_iris <- do.call(rbind, multi_res_iris)
merged_res_iris

      Clusters    WPS
 [1,] integer,150 13521.46
 [2,] integer,150 13521.46
 [3,] integer,150 13521.46
 [4,] integer,150 13521.46
 [5,] integer,150 13521.46
 [6,] integer,150 4084.43
 [7,] integer,150 4084.43
 [8,] integer,150 4084.43
 [9,] integer,150 4084.43
[10,] integer,150 4084.43

#the clusters column contains the cluster indexes for each obs.

which.min(merged_res_iris[,2])

[1] 6

which.max(merged_res_iris[,2])

[1] 1
```

Looking at the results for Iris, we get the all the WPS are the sames given the same number of clusters such that all results for 2 clusters are 13521.46 whereas the results for 3 clusters are all 4084.

1.9 1.9

Now we visualize the clusters corresponding to the best and worst WPS. Where they are tied I just chose one at random.

```
plot_func <- function(X, clusters, var1, var2){
  df <- cbind(X, "clusters"= clusters) %>%
    as.data.frame()
  df$clusters <- as.factor(df$clusters)
  x <- ensym(var1)
  y = ensym(var2)
  ggplot(df ) +
    geom_point(aes(x=!!x, y= !!y, color=clusters))
  # "!!" is for injecting the variables from strings with ensym
}
```

```
#here we are sapplying over the indexes corresponding to best and worse WPS
plots_eruptions <- sapply(c(1,7), FUN = function(i){plot_func(X,
                                              multi_res_faithful[[i]]$Clusters,
  var1="eruptions", var2="waiting")}, simplify = FALSE)
gridExtra::grid.arrange(plots_eruptions[[1]], plots_eruptions[[2]])
```

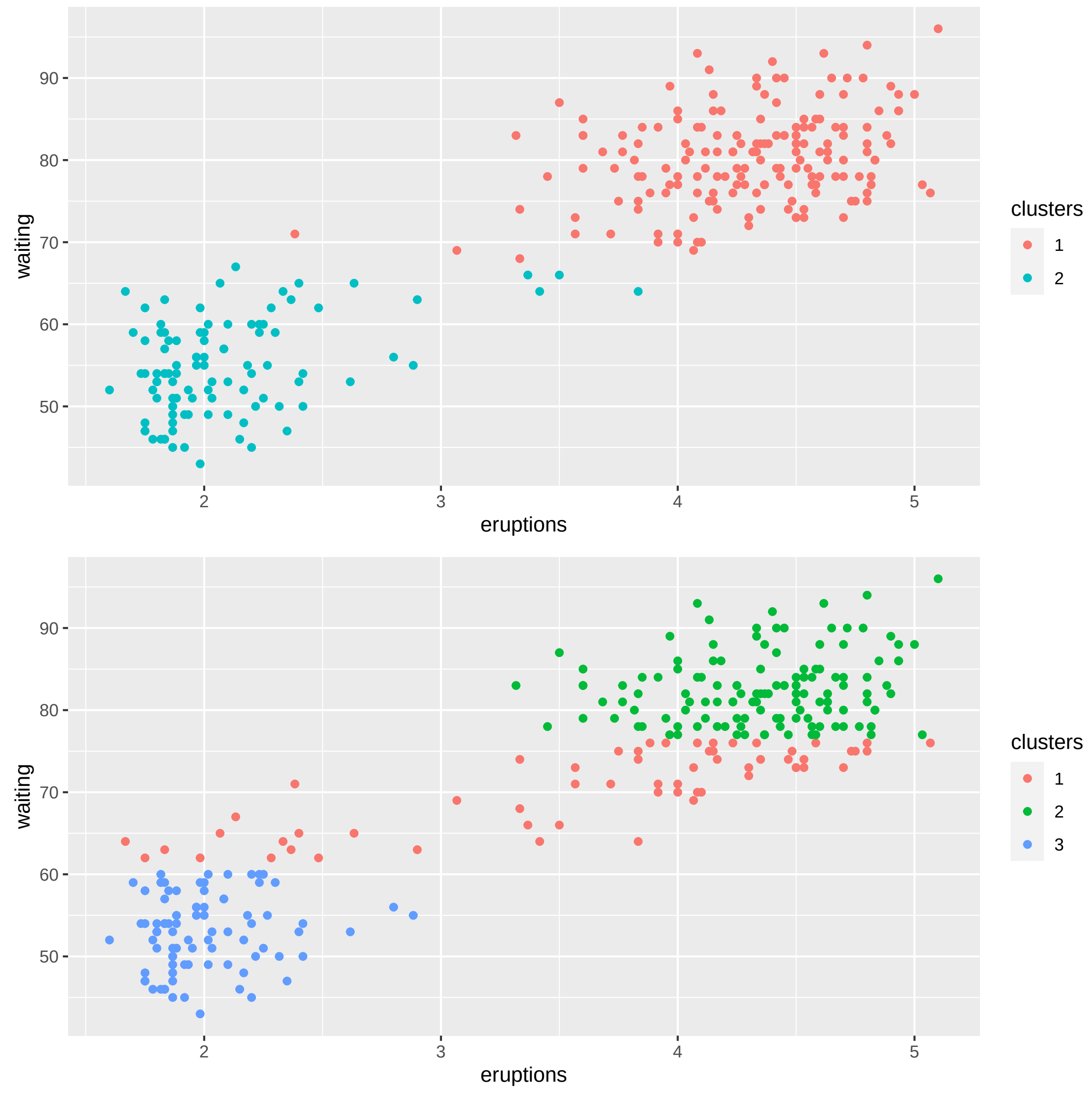


Figure 4: Clusters for eruption

```
plots_iris <- sapply(c(6,1), FUN = function(i){plot_func(as.matrix(iris),
                                                multi_res_iris[[i]]$Clusters,
  var1="Petal.Length", var2="Petal.Width")}, simplify = FALSE)

gridExtra::grid.arrange(plots_iris[[1]], plots_iris[[2]])
```

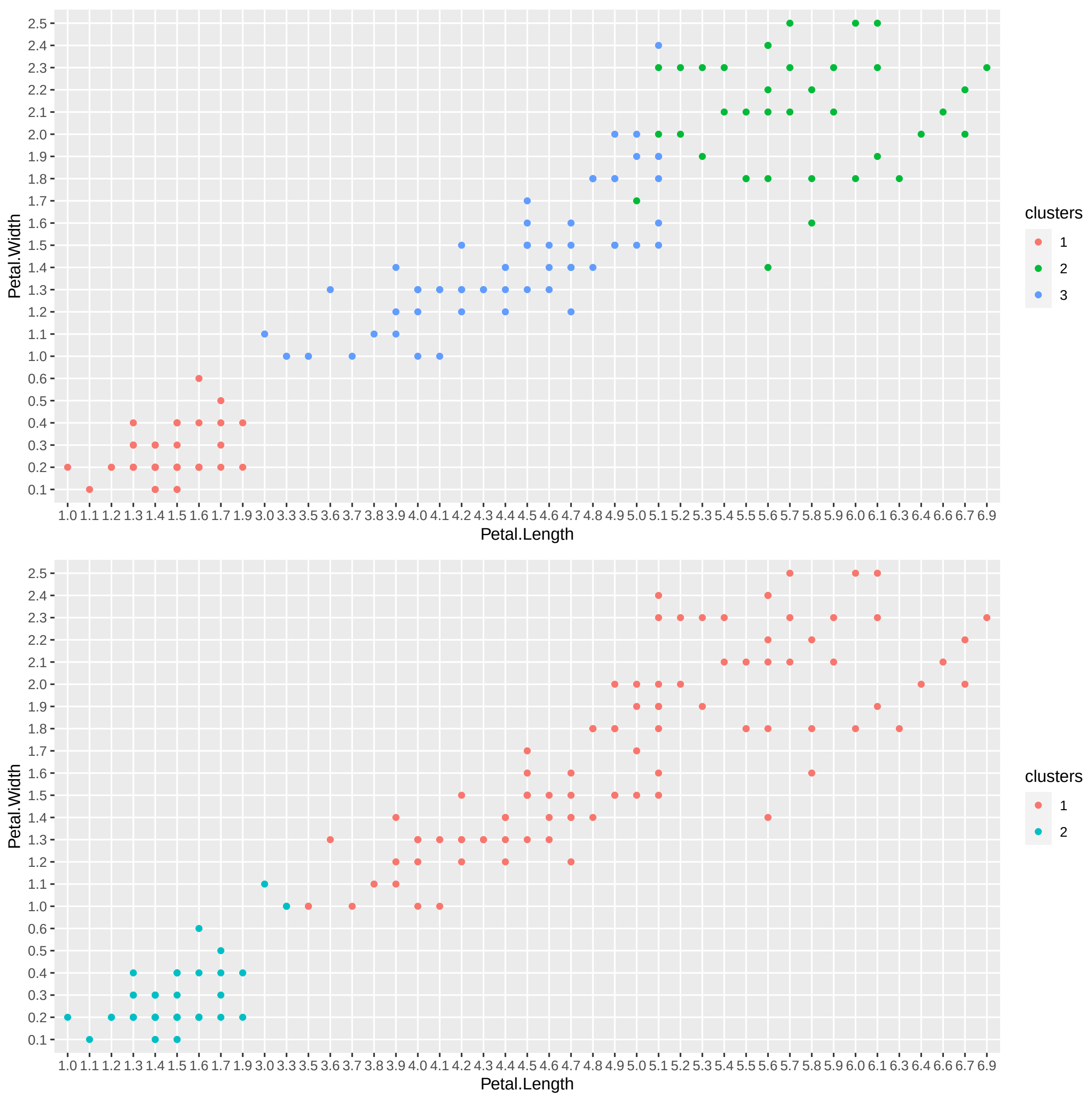


Figure 5: Clusters for petal length

1.10 1.10

Looking at the results both from 1.8 and 1.9, then we can say that for both of the two data sets then when comparing all 10 results for the different clusters and seed, then the results seem to be similar to each other and follow a similar pattern. That is, for example in both data sets, when comparing the results for all clusters with size 2, or for all clusters with size 3, then those results given the same cluster size were similar to each other with respect to the WPS. But also that 3 clusters were always better in terms of lower WPS.

For example, for the eruption data with 2 clusters they all had the same WPS, and for 3 clusters they were not all the same but relatively similar to each other still, but when comparing 3 clusters to 2 clusters, then 3 clusters had lower WPS.

Likewise for the iris data, the 2 cluster results had the same WPS, as well as the the 3 clusters having the same WPS. but when comparing 2 clusters to 3 clusters, than 3 clusters always had lower WPS.

Looking at the plots of the clusters for the different data. Then we can see that for figure 4 the subplot with 2 clusters is similar to the one proposed in task 1.3, where we see two groups with, on average, one group with low values of eruptions and waiting in contrast to the other group with higher values. Looking at the plot for the one with 3 clusters, then we see that the third cluster is the one with lowest eruptions and waiting, on average. The second cluster seem to have the highest eruptions and waiting, whereas the first cluster seen ti be more spread out with regards to eruptions, but in somewhere in between the two other clusters with regard to waiting.

As for the iris plots, we see the for the 2 cluster plot, then one group is considerably smaller with relatively low values of petal length and petal width, whereas the other group contains a larger spread of values. For the subplot with 3 clusters then it can bee seen that the first cluster there almost corresponds to the second cluster from the previous plot. But now we also have one cluster which is somewhere in the middle of petal length and petal width, whereas the third cluster has large values of petal length and petal width.

2 Task 2

2.1 2.1

For this task we implement a function that simulate data from the probabilistic PCA model

We have that:

$$\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^\top + \sigma^2 \boldsymbol{I}\right) \tag{1}$$

or equivalently

$$\mathbf{x} = \boldsymbol{W}\mathbf{h} + \boldsymbol{b} + \sigma\mathbf{z} \tag{2}$$

Hence we can just simulate $\boldsymbol{X}$ directly from the multivariate normal distribution.

```r
pPCA <- function(W,b, sigma2, n_sims){
  sigma2_diag <- diag(length(W)) #creating a diagonal matrix
  diag(sigma2_diag) <- sigma2 #replacing diagonals with sigma2
mv_sigma <- matrix(W) %*% t(matrix(W)) + sigma2_diag
X <- rmvnorm(n=n_sims, mean=matrix(b), sigma=mv_sigma )
colnames(X) <- sapply(1:ncol(X), FUN = function(i){ paste0("X",i)})
attributes(X)$W <- W
attributes(X)$b <- b
return(X)
}
```

2.2 2.2

Now we simulate 300 observations and plot them.

```r
set.seed(1337)
W <- c(-1,3)
b <- c(0.5, 2)
results <- pPCA(W,b,sigma2=1,n_sim=300)
```

```r
plot(results)
```

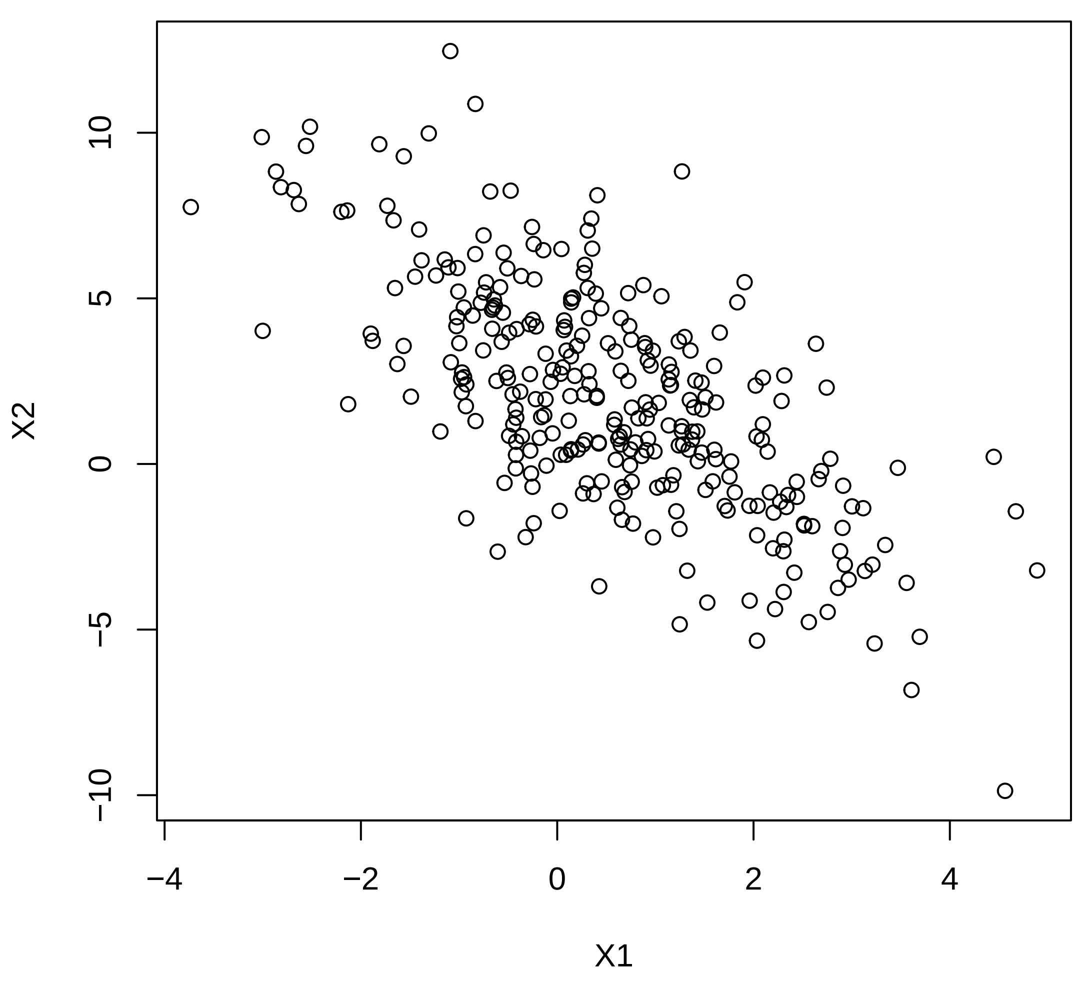


Figure 6: Visualization of 300 simulated observations

2.3 2.3

As described by the equation in 2.1, this model assumes that $\boldsymbol{x}$ follows a multivariate normal distribution as

$$\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^{\top} + \sigma^2 \boldsymbol{I}\right)$$

Hence the argument b in the code corresponds to the mean vector $\boldsymbol{b}$ and that the terms $\boldsymbol{W}\boldsymbol{W}^T + \sigma^2\boldsymbol{I}$ corresponds to the variance matrix. Also we can say that the argument W corresponds to the factor loadings in the model recalling that it could also be represented on the form:

$$\mathbf{x} = \boldsymbol{W}\mathbf{h} + \boldsymbol{b} + \sigma\mathbf{z}$$

where $\boldsymbol{h}$ here represents the latent variables.

2.4 2.4

Now we test some other parameter values for W and b and visualize the results.

```r
W_list <- list(c(1,2), c(-3,3),
               c(2,-5), c(0,0),
               c(6,10), c(-5,15))

b_list <- list(c(1,3), c(-2,5),
               c(0.8, 6), c(-1, 1),
               c(2, -3), c(10, 3))
set.seed(1337)
multi_res <- mapply(FUN=pPCA, W=W_list, b=b_list, n_sims=300, sigma2=1, SIMPLIFY = FALSE)
```

```r
par(mfrow=c(2,3))
lapply(multi_res, FUN = function(i){
  W <- attributes(i)$W
  b <- attributes(i)$b
main <- paste("W = (", paste(W, collapse = ", "), ")", ", ",
              "b = (", paste(b, collapse = ", "), ")" )
  plot(i, main=main)
})
```

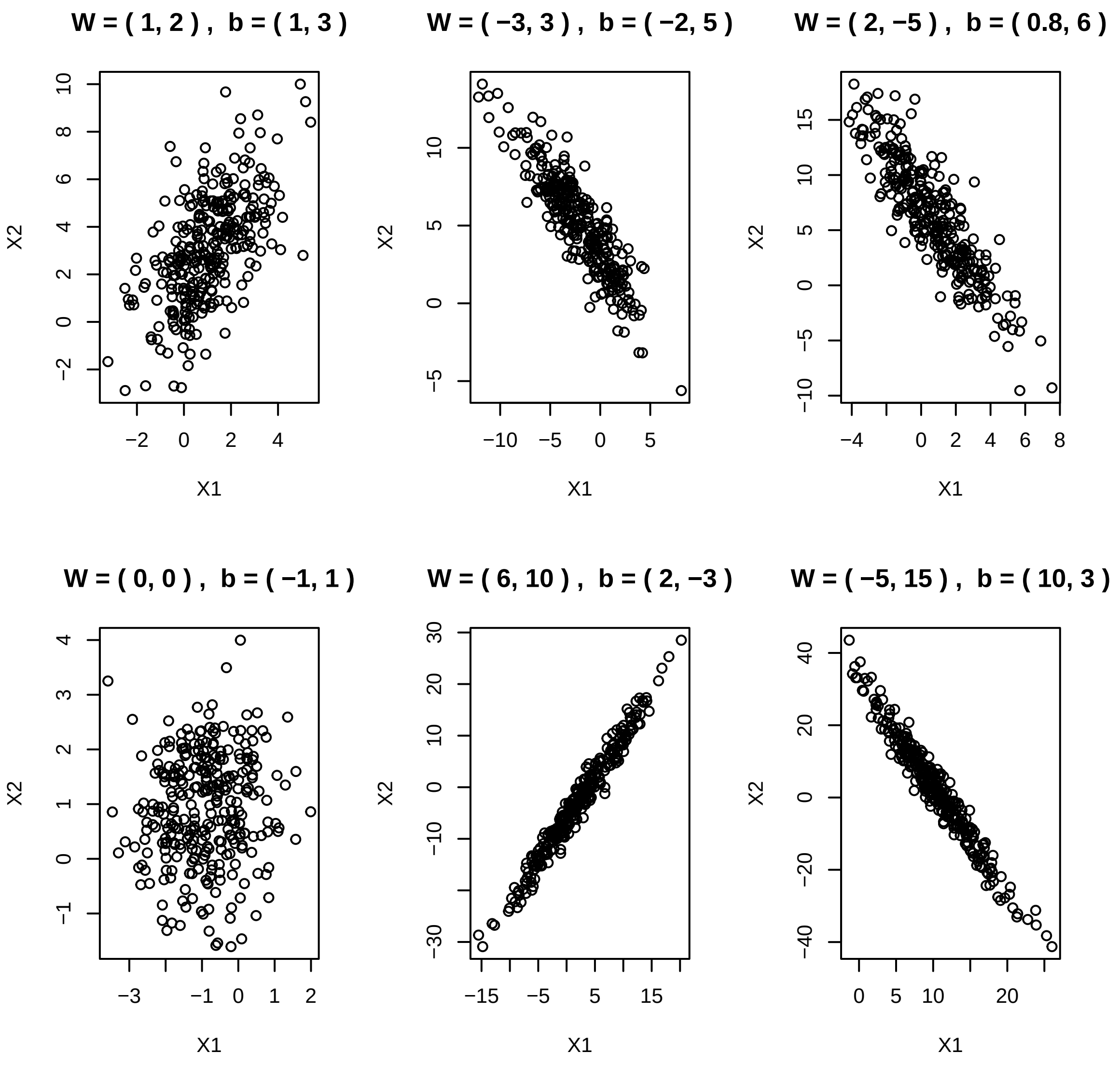


Figure 7: Results for different values of W and b

```
par(mfrow=c(1,1))
```

2.5 2.5

Now we run PCA on the simulated data from task 2.2.

```
pr_res <- prcomp(results, center=TRUE, scale= FALSE)
pr_res$rotation[,1] * pr_res$sdev[1] * -1

       X1         X2
-1.107786  3.414624
```

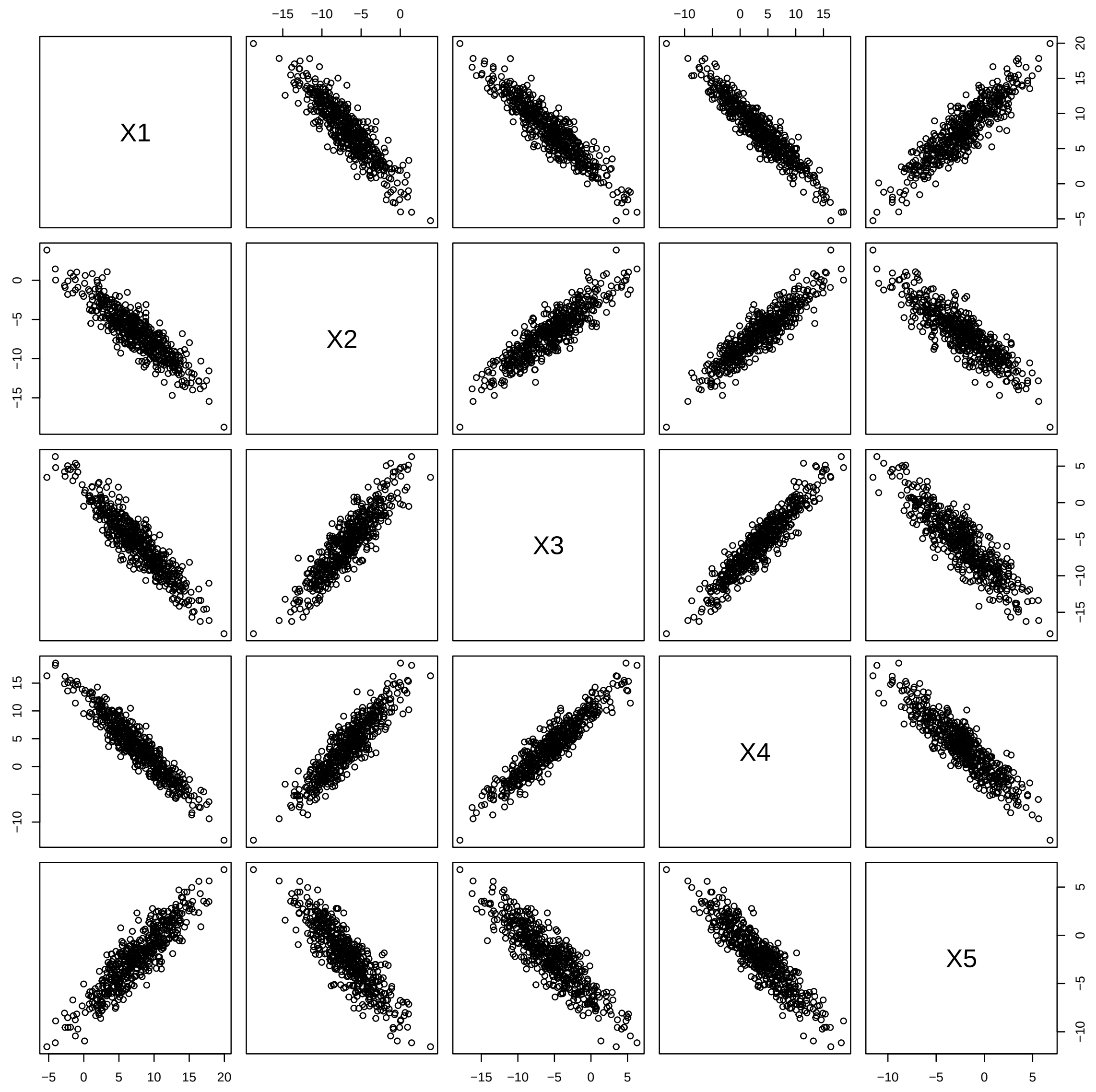
2.6 2.6

Looking at the results, it can be seen that this corresponds to the parameters in $\boldsymbol{W}$, seeing as they are close but not exactly the same .

2.7 2.7

For this task we want to simulate new data $\boldsymbol{x}$ with five dimensions. We can do that by reusing the previously defined function but increasing the dimension for $\boldsymbol{W}$ and $\boldsymbol{b}$. I will just simulate some values for them but also make sure in the end that we get both positive and negative correlations

```
set.seed(1337)
new_W = sample(-5:5, 5, replace=FALSE)
new_b <- sample(-10:10, 5, replace=TRUE)
print(new_W)

[1]  4 -3 -4 -5  3

print(new_b)

[1]  8 -7 -6  3 -2

new_results  <- pPCA(W = new_W, b = new_b, sigma2 = 1.3, n=500)
pairs(new_results)
```

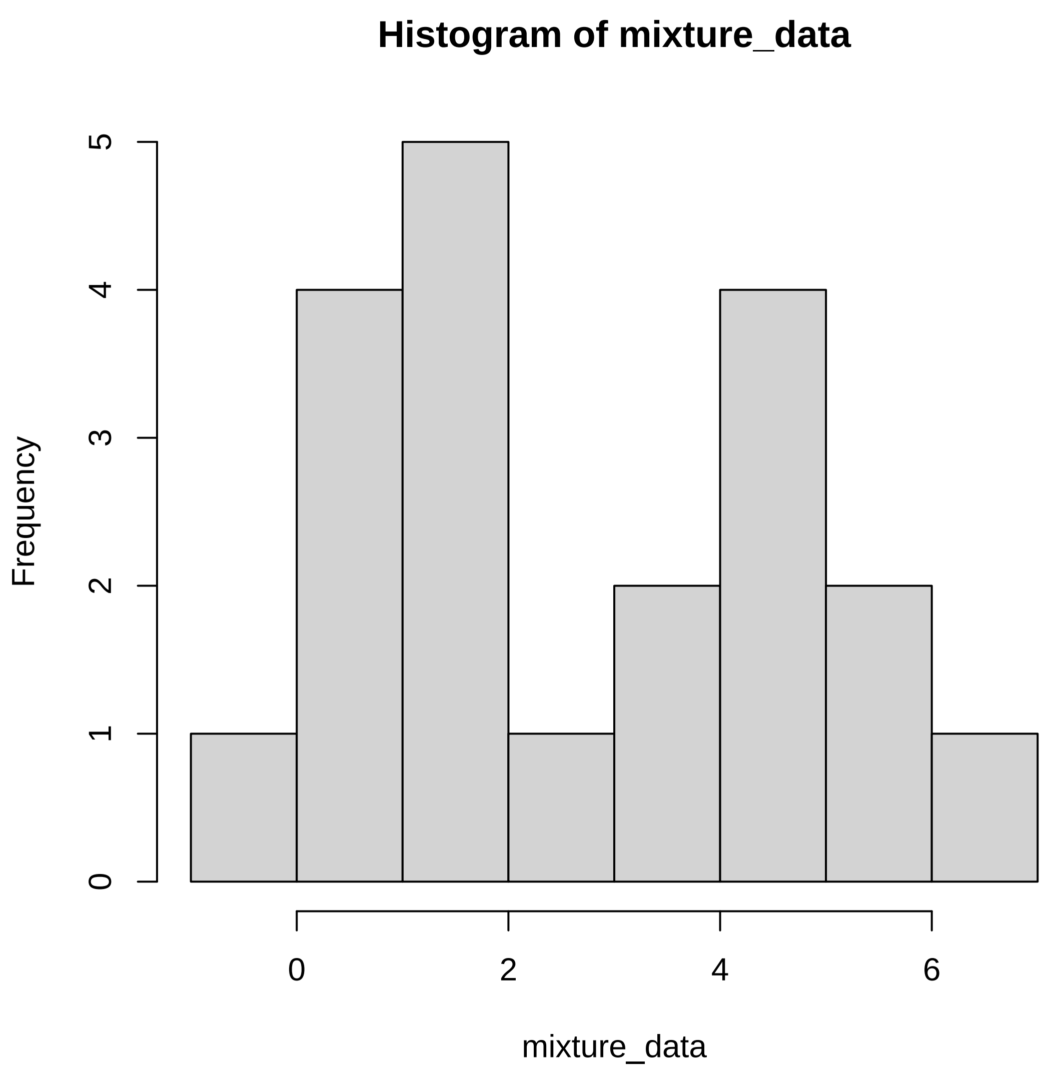


3 Task 3

```
library(uuml)
data("mixture_data")
theta0 <- list(mu_1 = 4.12, mu_2 = 0.94, sigma_1 = 2, sigma_2 = 2, pi = 0.5)
theta0
```

```
$mu_1
[1] 4.12

$mu_2
[1] 0.94

$sigma_1
[1] 2

$sigma_2
[1] 2
```

```
$pi
[1] 0.5

hist(mixture_data)
```



3.1 3.1

First we create a function to simulate data from a univariate mixture model.

```
r_uni_two_comp<- function(n, theta){
  #sample delta for each obs
  deltas <- rbinom(n=n, 1, p=theta$pi)
  y_1 <- rnorm(n=n, mean=theta$mu_1, sd= theta$sigma_1)
  y_2 <- rnorm(n=n, mean=theta$mu_2, sd=theta$sigma_2)
  Y <- (1-deltas) * y_1 + deltas*y_2
  return(Y)
  #samply y_i from the relevant component
}
head(r_uni_two_comp(n=100, theta=theta0))

[1] 2.2514313 2.9134916 4.7977395 3.1977811 2.7223639 0.3170157
```

3.2 3.2

Now to simulate 200 observations using $\mu_1 = -2, \mu_2 = 1.5, \sigma_1 = 2, \sigma_2 = 1, \pi = 0.3$

```r
set.seed(1337)
thetas <-list(mu_1=-2, mu_2=1.5,
              sigma_1=2, sigma_2=1,
              pi=0.3)

observations <- r_uni_two_comp(200, thetas)
hist(observations)
```

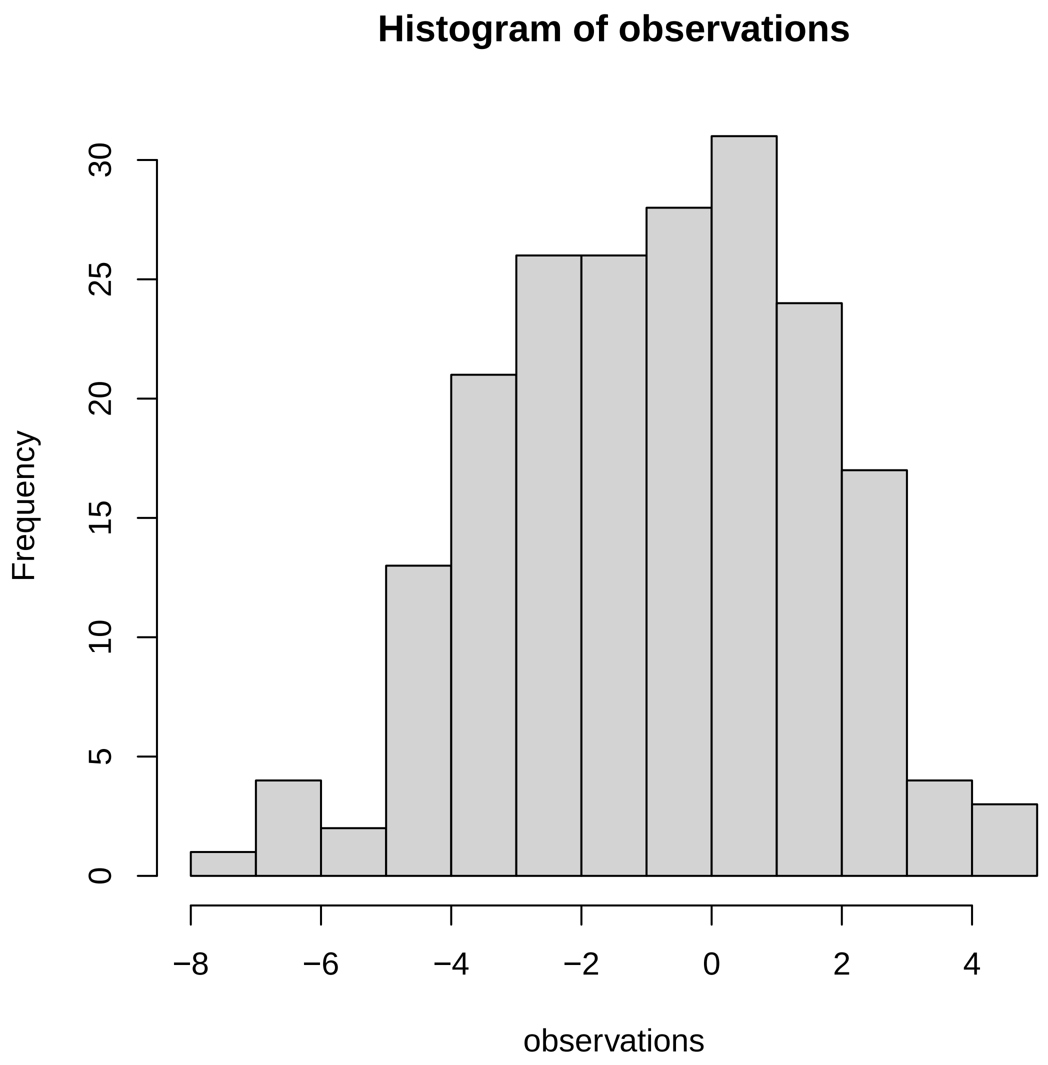


Figure 8: Distribution for simulated observations

3.3 3.3

Now we implement a function to compute the density values for a given set of parameters theta for values of x.

```r
d_uni_two_comp <- function(x, theta){
  dens_1 <- dnorm(x, mean=theta$mu_1, sd = theta$sigma_1)
  dens_2 <- dnorm(x, mean=theta$mu_2, sd=theta$sigma_2)
  g_x <- (1-theta$pi) * dens_1 + theta$pi*dens_2
  return(g_x)
}
```

3.4 3.4

Now we visualize the density for the mixture model.

```
plot_density <- function(x, theta,...){
  densities <- d_uni_two_comp(x, theta)
  plot(x=x, y=densities, col="blue", ...)
}

plot_density(x=seq(-4,4, by=0.01), theta=thetas)
```

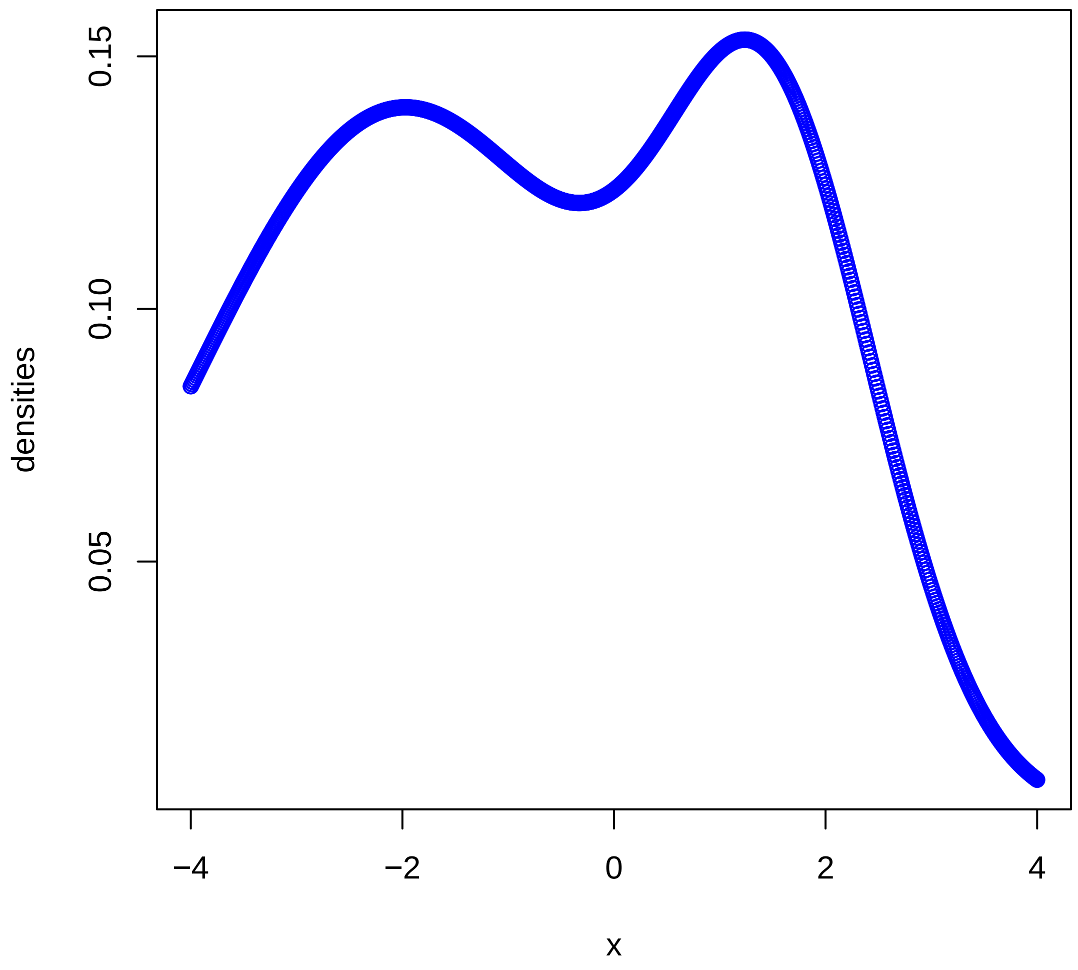


Figure 9: Density for the mixture model from -4 to 4

3.5 3.5

now we visualize the eruptions variable in the faithful data

```
hist(faithful$eruptions)
```

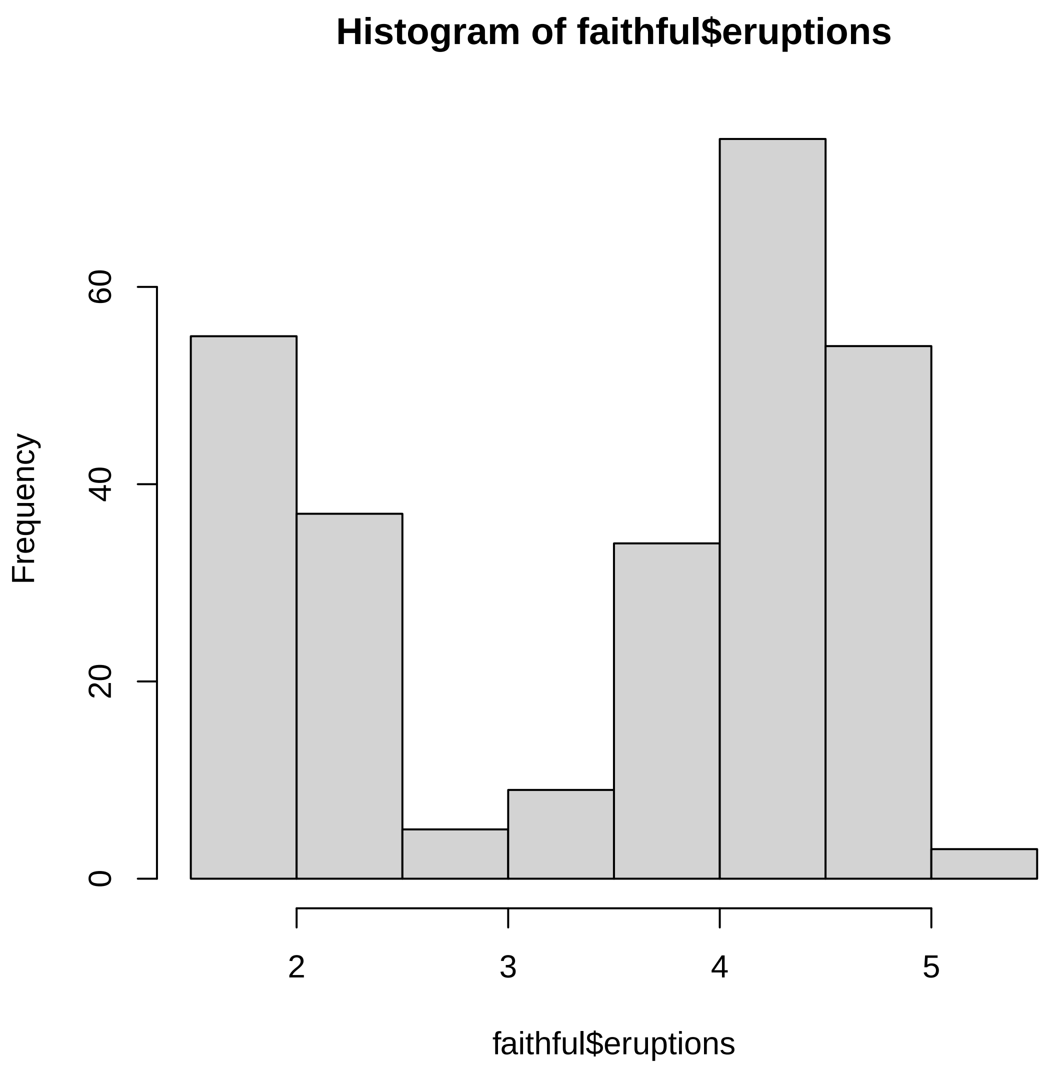


Figure 10: Distribution of eruptions

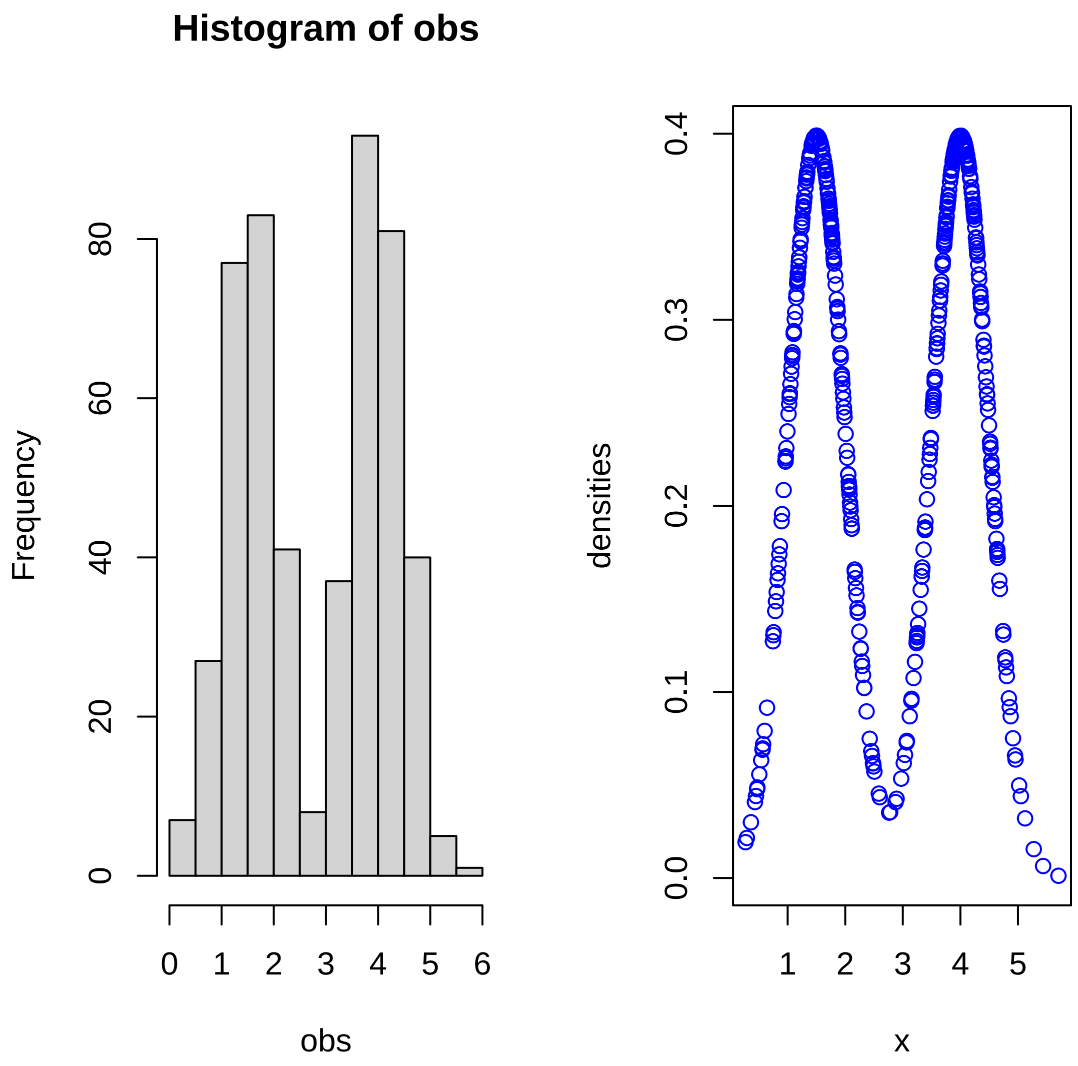
3.6 3.6

Based on the figure 10 we see that there are two peaks. One around 2 and one around 4.5 which could be indication of what means values to use. Testing for some different values of parameters of theta, the following one gives somewhat reasonable results.

```
set.seed(1337)
theta3<- list(pi=0.5, "mu_1" = 4, "mu_2" = 1.5, sigma_1 = 0.5, sigma_2 = 0.5)
print(theta3)

$pi
[1] 0.5

$mu_1
```

```
[1] 4

$mu_2
[1] 1.5

$sigma_1
[1] 0.5

$sigma_2
[1] 0.5
```

```
obs <- r_uni_two_comp(n=500,theta=theta3)
par(mfrow=c(1,2))
hist(obs)
plot_density(x=obs, theta3)
```



```
par(mfrow=c(1,1))
```

3.7 3.7

Now we implement a function that returns a vector of gamma values for each row in $\boldsymbol{X}$

```r
e_uni_two_comp <- function(data, theta){
  pi <- theta$pi
  mu1 <- theta$mu_1
  mu2 <- theta$mu_2
  sigma1 <- theta$sigma_1
  sigma2 <- theta$sigma_2
  numerator <- pi*dnorm(data, mean=mu2, sd=sigma2)
  denominator <- (1-pi) *dnorm(data, mean=mu1, sd=sigma1) + numerator
  gamma_values <- numerator / denominator
  return(gamma_values)
}

gamma <- e_uni_two_comp(mixture_data, theta0)
t(head(gamma))
```

```
          [,1]      [,2]      [,3]     [,4]      [,5]      [,6]
[1,] 0.9106339 0.8716861 0.7797225 0.664564 0.6484311 0.5178799
```

3.8 3.8

The gamma values are the irresponsibility for the observations. These responsibilities are obtained by the estimates of the parameters used in the function such that the responsibilities are assigned according to the relative density of the training under both models. For example, the first three obtained gamma values were 0.91, 0.87, and 0.78. These corresponds to the probabilities or weights that the first three observations belong to group 2 since component 2 is used as the base line component or so to speak .

3.9 3.9

Now we implement a function to return a list with the parameters given X and gamma.

```r
max_uni_two_comp <- function(x, gamma){
  mu_1 <- sum((1-gamma)*x) / sum(1-gamma)
  mu_2 <- sum(gamma*x) / sum(gamma)
  sigma_1_square <- sum ( (1-gamma) * (x-mu_1)^2 ) / sum(1-gamma)
  sigma_2_square <- sum( gamma*(x-mu_2)^2 ) / sum(gamma)
  pi <- mean(gamma)

  results <- list("mu_1"= mu_1, "mu_2"=mu_2, "sigma_1"= sqrt(sigma_1_square),
                  "sigma_2"= sqrt(sigma_2_square), "pi"=pi)
  return(results)
}

max_uni_two_comp(mixture_data, gamma)
```

```
$mu_1
[1] 3.842941

$mu_2
[1] 1.450413

$sigma_1
[1] 1.700666

$sigma_2
[1] 1.47168

$pi
[1] 0.4883709
```

3.10 3.10

Next we implement the log-likelihood as a function.

```
ll_uni_two_comp <- function(x, theta){
  pi <- theta$pi
  mu1 <- theta$mu_1
  mu2 <- theta$mu_2
  sigma1 <- theta$sigma_1
  sigma2 <- theta$sigma_2
  log_lik <- log((1-pi)*(dnorm(x, mean=mu1, sd=sigma1)) +
                   pi*dnorm(x, mean=mu2, sd=sigma2))
  total_sum <- sum(log_lik)
  return(total_sum)
}
ll_uni_two_comp(mixture_data, theta0)
```

```
[1] -43.1055
```

3.11 3.11

Now to combine the implemented function to an EM algorithm.

```
#' @param X  n x P data matrix
em_uni_two_comp <- function(X, theta0, iter){
  gamma_matrix <- matrix(NA, ncol=nrow(X), nrow=iter)
  theta_matrix <- matrix(NA, ncol=length(theta0), nrow=iter)
  log_l_matrix <- matrix(NA, ncol=1, nrow=iter)
  colnames(theta_matrix) <- names(theta0)
  #initialize first theta
   theta <- theta0
  for (i in 1:iter){
    gamma_values <- e_uni_two_comp(X, theta)
    theta <- max_uni_two_comp(X, gamma=gamma_values)
    l_lik <- ll_uni_two_comp(x=X, theta=theta)
    gamma_matrix[i,] <- gamma_values
    log_l_matrix[i,] <- l_lik
    theta_matrix[i,] <- do.call(cbind, theta)
    cat("Iter", i, "Log Lik: ",l_lik, fill = TRUE)

  }
   return(list("gamma" = gamma_matrix, "log_l"= log_l_matrix, "theta"= theta_matrix))
}
em_res <- em_uni_two_comp(mixture_data, theta0, iter = 3)
```

```
Iter 1 Log Lik:  -41.53247
Iter 2 Log Lik:  -41.11211
Iter 3 Log Lik:  -40.48348
```

```
em_res
```

```
$gamma
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
[1,] 0.9106339 0.8716861 0.7797225 0.6645640 0.6484311 0.5178799 0.2796799
[2,] 0.9178650 0.8894927 0.8167940 0.7116103 0.6955099 0.5528707 0.2519108
[3,] 0.9454468 0.9246043 0.8625704 0.7544583 0.7362906 0.5618554 0.1858035
           [,8]       [,9]      [,10]     [,11]     [,12]     [,13]     [,14]
[1,] 0.19920828 0.13010282 0.08432370 0.8769274 0.8361354 0.7700157 0.6627895
[2,] 0.15222465 0.07724154 0.03725366 0.8933796 0.8624511 0.8085676 0.7098561
[3,] 0.08785076 0.03210448 0.01096926 0.9275816 0.9028954 0.8548465 0.7524975
         [,15]     [,16]     [,17]      [,18]      [,19]       [,20]
[1,] 0.6411479 0.3606832 0.2202775 0.16169773 0.10099206 0.050519768
```

```
[2,] 0.6881296 0.3569185 0.1774257 0.10978815 0.05066204 0.015114749
[3,] 0.7278372 0.3091756 0.1103649 0.05404032 0.01723768 0.002916854

$log_l
          [,1]
[1,] -41.53247
[2,] -41.11211
[3,] -40.48348

$theta
         mu_1     mu_2  sigma_1  sigma_2        pi
[1,] 3.842941 1.450413 1.700666 1.471680 0.4883709
[2,] 3.967199 1.322309 1.641096 1.313247 0.4887533
[3,] 4.100188 1.179100 1.527945 1.145685 0.4880674
```

3.12 3.12

Now to test in on the mixture data for 20 iterations.

```
em_res_20 <- em_uni_two_comp(mixture_data, theta0 , iter = 20)
```

```
Iter 1 Log Lik:  -41.53247
Iter 2 Log Lik:  -41.11211
Iter 3 Log Lik:  -40.48348
Iter 4 Log Lik:  -39.80459
Iter 5 Log Lik:  -39.3837
Iter 6 Log Lik:  -39.19062
Iter 7 Log Lik:  -39.07533
Iter 8 Log Lik:  -38.9989
Iter 9 Log Lik:  -38.95411
Iter 10 Log Lik:  -38.93147
Iter 11 Log Lik:  -38.92116
Iter 12 Log Lik:  -38.91669
Iter 13 Log Lik:  -38.91478
Iter 14 Log Lik:  -38.91397
Iter 15 Log Lik:  -38.91363
Iter 16 Log Lik:  -38.91348
Iter 17 Log Lik:  -38.91342
Iter 18 Log Lik:  -38.91339
Iter 19 Log Lik:  -38.91338
Iter 20 Log Lik:  -38.91338
```

```
round(em_res_20$theta[c(1,5,10,15,20),], digits=3)
```

```
      mu_1  mu_2 sigma_1 sigma_2    pi
[1,] 3.843 1.450   1.701   1.472 0.488
[2,] 4.340 0.997   1.246   0.890 0.498
[3,] 4.609 1.051   0.948   0.875 0.544
[4,] 4.651 1.079   0.909   0.897 0.553
[5,] 4.655 1.083   0.905   0.900 0.554
```

The results for the parameters are pretty similiar, though they are not exactly equal for all decimals. Now we can plot the log-likelihood as well.

```
plot(em_res_20$log_l, col="green", type="l", xlab="Iteration",
     ylab="Observed Data Log-likelihood")
points(em_res_20$log_l, col="green", pch=1, cex=1.5)
```

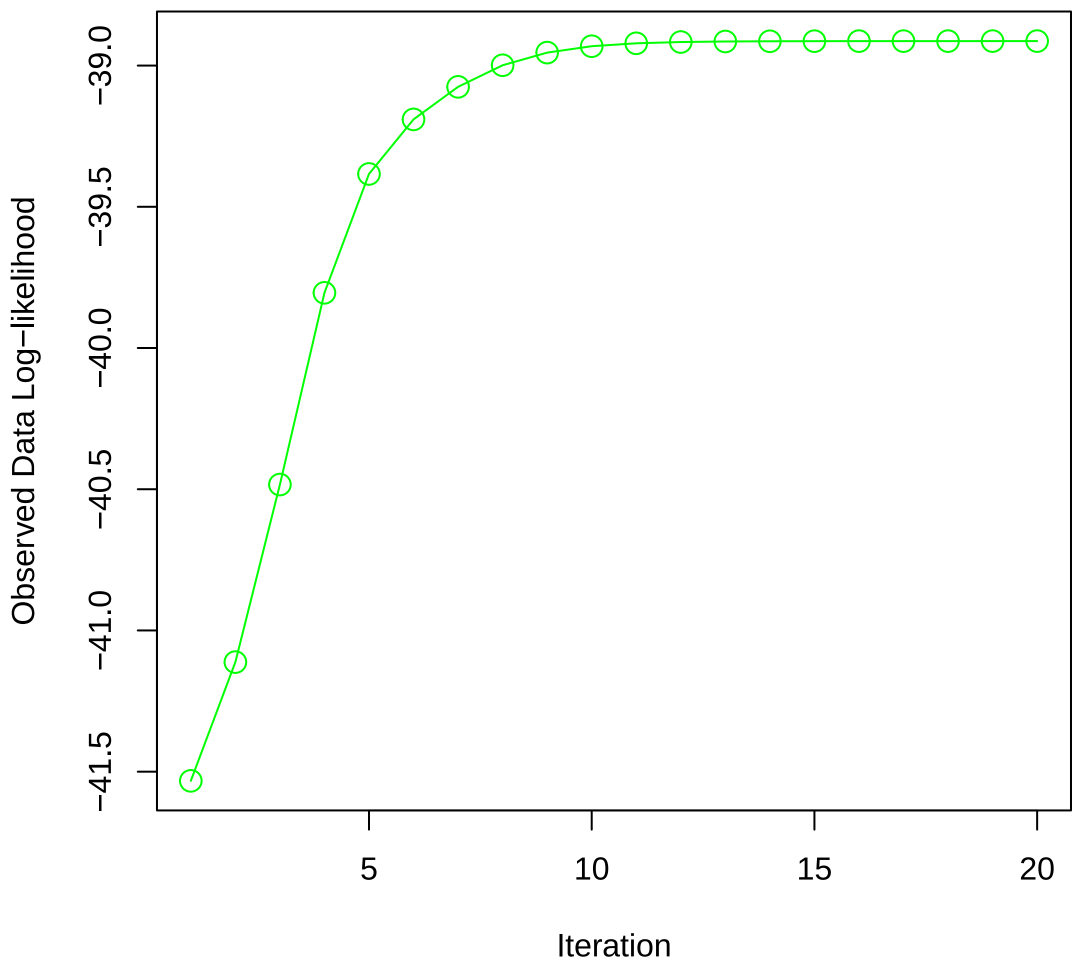


Figure 11: EM algorithm: observed data log-likelihood as a function of the iteration number.

The plot is also pretty similar to the one in Hastie et al.

3.13 3.13

Next we run the EM algorithm on the eruptions variable as well as Petal.Length.

```
eruption_res <-em_uni_two_comp(as.matrix(faithful$eruptions), theta0 , iter = 20)
```

```
Iter 1 Log Lik:  -418.3729
Iter 2 Log Lik:  -416.5957
Iter 3 Log Lik:  -412.7431
Iter 4 Log Lik:  -402.6956
Iter 5 Log Lik:  -373.5664
Iter 6 Log Lik:  -320.9953
Iter 7 Log Lik:  -293.9944
Iter 8 Log Lik:  -280.3981
Iter 9 Log Lik:  -277.4294
Iter 10 Log Lik:  -276.816
Iter 11 Log Lik:  -276.567
Iter 12 Log Lik:  -276.4501
Iter 13 Log Lik:  -276.3969
Iter 14 Log Lik:  -276.3743
Iter 15 Log Lik:  -276.3654
Iter 16 Log Lik:  -276.362
Iter 17 Log Lik:  -276.3607
Iter 18 Log Lik:  -276.3603
Iter 19 Log Lik:  -276.3601
Iter 20 Log Lik:  -276.3601
```

```
petal_res<- em_uni_two_comp(as.matrix(iris$Petal.Length), theta0 , iter = 20)
```

```
Iter 1 Log Lik:  -293.4476
Iter 2 Log Lik:  -289.0347
Iter 3 Log Lik:  -277.2853
Iter 4 Log Lik:  -251.7706
Iter 5 Log Lik:  -217.0564
Iter 6 Log Lik:  -200.5795
Iter 7 Log Lik:  -200.5788
Iter 8 Log Lik:  -200.5788
Iter 9 Log Lik:  -200.5788
Iter 10 Log Lik:  -200.5788
Iter 11 Log Lik:  -200.5788
Iter 12 Log Lik:  -200.5788
Iter 13 Log Lik:  -200.5788
Iter 14 Log Lik:  -200.5788
Iter 15 Log Lik:  -200.5788
Iter 16 Log Lik:  -200.5788
Iter 17 Log Lik:  -200.5788
Iter 18 Log Lik:  -200.5788
Iter 19 Log Lik:  -200.5788
Iter 20 Log Lik:  -200.5788
```

```
#estimated parameters for iter 1,5,10,15, and 20
eruption_res$theta[c(1,5,10,15,20),]
```

```
         mu_1     mu_2   sigma_1   sigma_2        pi
[1,] 3.825971 2.839693 1.0036998 1.1024940 0.3428935
[2,] 4.176372 2.396603 0.6629985 0.8462387 0.3868979
[3,] 4.285734 2.032232 0.4184247 0.2570452 0.3540938
[4,] 4.274590 2.019929 0.4351905 0.2377294 0.3489691
[5,] 4.273436 2.018705 0.4369244 0.2357767 0.3484463
```

```
petal_res$theta[c(1,5,10,15,20),]
```

```
         mu_1     mu_2   sigma_1   sigma_2        pi
[1,] 4.453807 2.358893 1.4580177 1.4540669 0.3321410
[2,] 4.924221 1.510104 0.8114996 0.3437946 0.3415879
[3,] 4.904976 1.461750 0.8232177 0.1716566 0.3331109
[4,] 4.904976 1.461750 0.8232177 0.1716566 0.3331109
[5,] 4.904976 1.461750 0.8232177 0.1716566 0.3331109
```

3.14 3.14

Lastly we visualize the densities for the two data sets using the parameters estimated with the EM-algorithm

```
#Extracting thetas from last row of results
eruption_theta <- eruption_res$theta[nrow(eruption_res$theta),]
#converting thetas to list
eruption_theta_list <- list()
```

```
for(i in 1:5){
  eruption_theta_list[[names(eruption_theta)[i]]] <- eruption_theta[i]
}

par(mfrow=c(1,2))
hist(faithful$eruptions)
plot_density(x=faithful$eruptions, theta=eruption_theta_list,
             main="Densities for Eruption data")
```

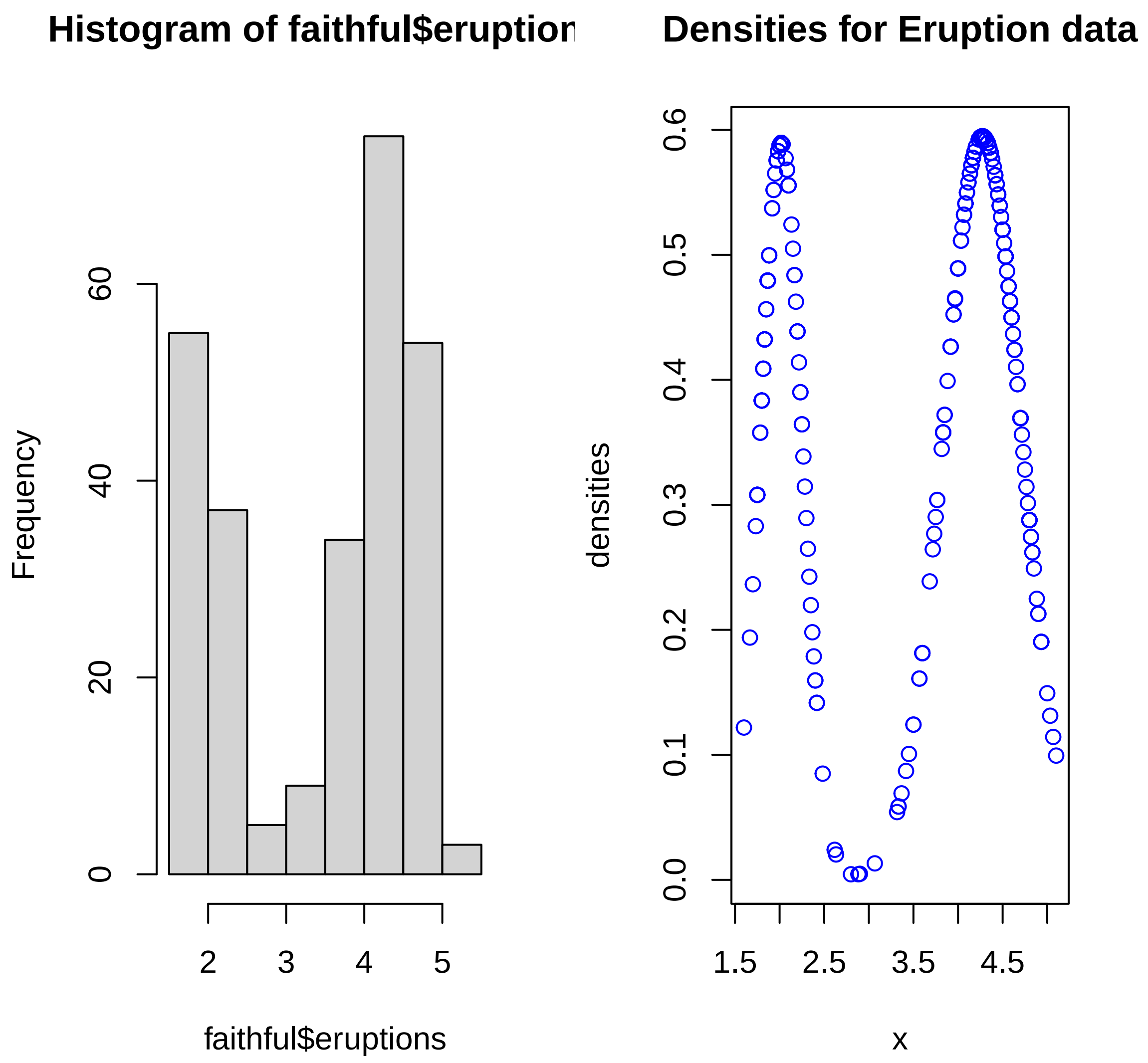


Figure 12: Densities for eruptions

```
petal_theta <- petal_res$theta[nrow(petal_res$theta),]
petal_theta_list <- list()
for(i in 1:5){
  petal_theta_list[[names(petal_theta)[i]]] <- petal_theta[i]
}
par(mfrow=c(1,2))
hist(iris$Petal.Length)
plot_density(x=iris$Petal.Length, theta=petal_theta_list,
             main="Densities for Petal length data")
```

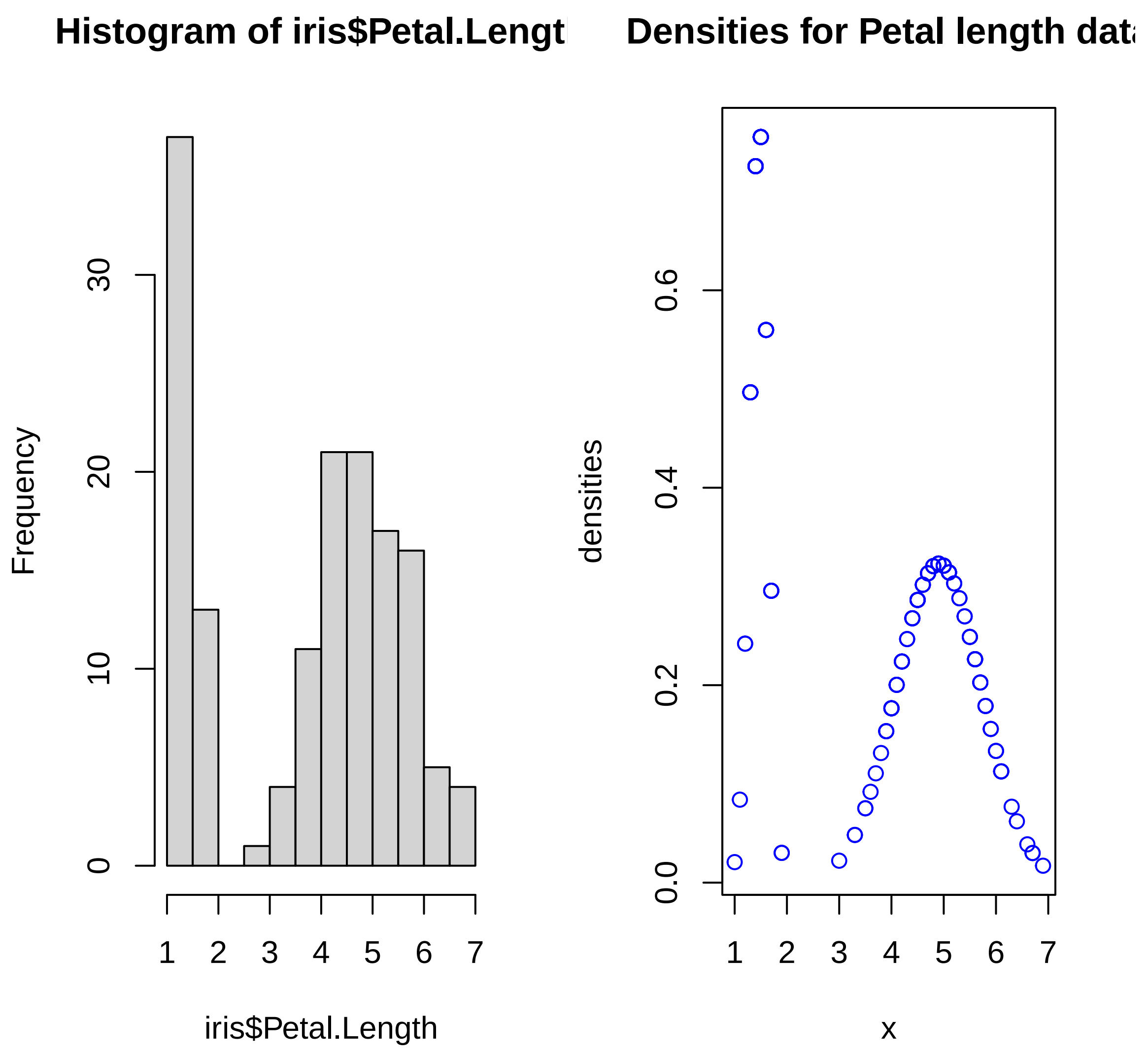


Figure 13: densities for petal length