STAT 542 / CS 598: Homework 5

Fall 2019, by John Moran (jfmoran2)

Due: Monday, Oct 28 by 11:59 PM Pacific Time

Contents

Question 1 [50 Points] K-Means Clustering

Let's consider coding a K-means algorithm. Perform the following:

- Load the `zip.train` (handwritten digit recognition) data from the ElemStatLearn package, and the goal is to identify clusters of digits based on only the pixels.
- [15 Points] Write your own code of k-means that iterates between two steps, and stop when the cluster membership does not change.
  - updating the cluster means given the cluster membership
  - updating the cluster membership based on cluster means

```
library(ElemStatLearn)
library(parallelDist)

set.seed(1)

data <- zip.train

data.Y <- data[, 1]
data.X <- data[, -1]

max.iterations <- 50

calc.distance <- function(X, k) {
  v <- sweep(X, 2, k, "-")
  v <- v^2
  return(rowSums(v))
}

my.kmeans <- function(X, k) {
  iterations <- 0

  centroid <- X[sample(nrow(X), k),]
  converged <- FALSE

  # stop when 99.99% of membership in current and previous clusters is the same
  stopping.criteria <- 0.0001

  while(!converged && iterations <= max.iterations) {
    if (iterations >= 2) {
      cluster.prev <- cluster
    }
```

```
    distances <- sapply(1:k, function(c) calc.distance(X,centroid[c,]))

    cluster <- apply(distances, 1, which.min)

    centroid <- t(sapply(1:k, function(c)
      apply(X[cluster == c,], 2, mean)))

    if (iterations >= 2) {
      # check cluster memberships compared to previous
      diff <- length(cluster) - sum(cluster == cluster.prev)
      membership.diff <- diff/length(cluster)

      if (membership.diff <= stopping.criteria) {
        converged <- TRUE
        print(sprintf("my.kmeans converged after %d iterations", iterations))
      }
    }

    iterations <- iterations + 1
  }

  results <- list(centroid = centroid, cluster = cluster)
  return(results)
}
```

- [10 Points] Perform your algorithm with one random initialization with $k = 5$
  - For this question, compare your cluster membership to the true digits. What are the most prevalent digits in each of your clusters?

```
n.clusters <- 5

k.means <- my.kmeans(data.X, n.clusters)
```

```
## [1] "my.kmeans converged after 39 iterations"
```

```
t1 <- table(data.Y, k.means$cluster)
```

The following tables compares the digits assigned to five clusters from the my.kmeans algorithm vs. the true digits:

```
print(t1)
```

```
##
## data.Y    1    2    3    4    5
##      0  400  764    1   22    7
##      1    1    0 1003    1    0
##      2   39   18   46  540   88
##      3    8    2    2  633   13
##      4   33    8   72   14  525
##      5  235   38    2  231   50
##      6  548   74   21    8   13
##      7    2    0    7    5  631
##      8    6    8   29  412   87
##      9    3    3   29    3  606
```

To determine what are the most prevalent digits, I chose a floor of at least 15% occurence in a specific cluster

to call it "prevalent":

```
t1 <- table(data.Y, k.means$cluster)
props <- prop.table(t1,2)
likely.digits <- sapply(1:n.clusters, function(i) names(which(props[,i] > .15)))

print("Acceptance threshold > 15% cluster is composed of these digits")
```

```
## [1] "Acceptance threshold > 15% cluster is composed of these digits"
```

```
for (i in 1:n.clusters) {
  print(sprintf("The most common digits assigned to cluster %d are: %s", i,
                paste(unlist(likely.digits[i]), collapse=" ")))
}
```

```
## [1] "The most common digits assigned to cluster 1 are: 0 5 6"
## [1] "The most common digits assigned to cluster 2 are: 0"
## [1] "The most common digits assigned to cluster 3 are: 1"
## [1] "The most common digits assigned to cluster 4 are: 2 3 8"
## [1] "The most common digits assigned to cluster 5 are: 4 7 9"
```

- [10 Points] Perform your algorithm with 10 independent initiations with $k = 5$ and record the best
  - For this question, plot your clustering results on a two-dimensional plot, where the two axis are the first two principle components of your data

```
within.cluster.variation <- function(X, clusters, n.clusters) {
  within.var <- 0
  for (k in 1:n.clusters) {
    tmpcluster <- X[clusters == k,]

    # sum up all the elements returned from parallel distance function which
    # will take advantage of multicore processors
    val <- sum(parDist(tmpcluster, method = "euclidean"))

    within.var <- within.var + (val/nrow(tmpcluster))
  }
 return(within.var)
}

min.within.var <- 1e10
n.initiations <- 10
for (i in 1:n.initiations) {
  k.means <- my.kmeans(data.X, n.clusters)
  within.var <- within.cluster.variation(data.X, k.means$cluster, n.clusters)

  if (within.var < min.within.var) {
      best.kmeans <- k.means
      min.within.var <- within.var
  }
}
```

```
## [1] "my.kmeans converged after 29 iterations"
## [1] "my.kmeans converged after 35 iterations"
## [1] "my.kmeans converged after 35 iterations"
## [1] "my.kmeans converged after 24 iterations"
## [1] "my.kmeans converged after 16 iterations"
## [1] "my.kmeans converged after 44 iterations"
```

```
## [1] "my.kmeans converged after 36 iterations"
## [1] "my.kmeans converged after 26 iterations"
## [1] "my.kmeans converged after 50 iterations"
```

```
print(sprintf("my.best kmeans within-cluster variation = %0.2f", min.within.var))
```

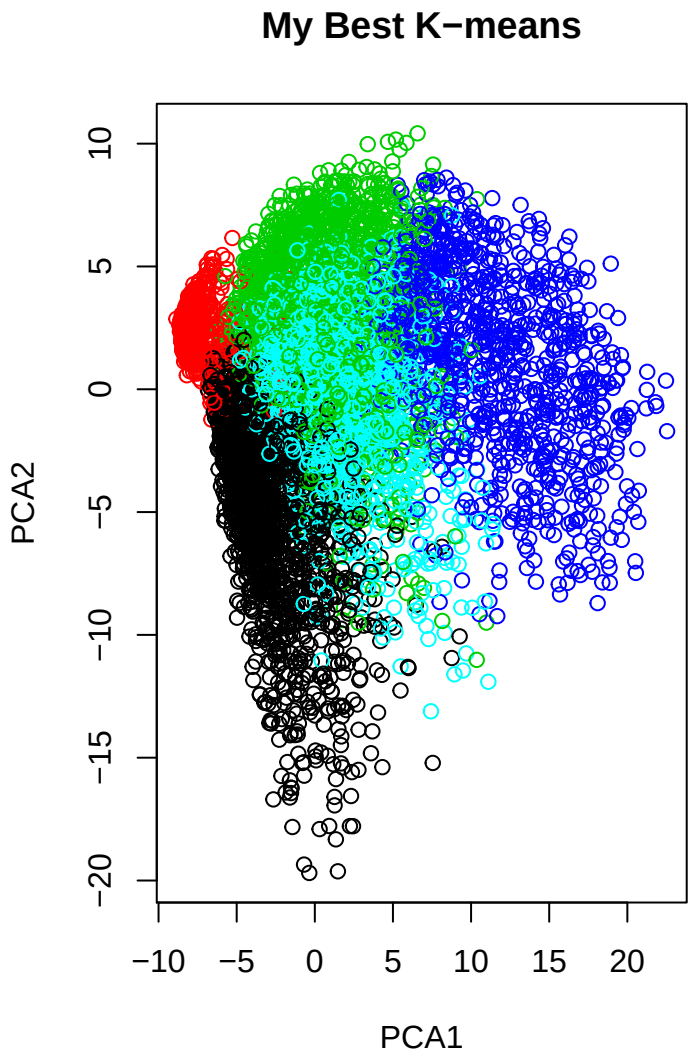
```
## [1] "my.best kmeans within-cluster variation = 46396.02"
```

The following is graph of the top two principal components:

```
pr <- prcomp(data.X, scale=TRUE)
PCA1 <- pr$x[,"PC1"]
PCA2 <- pr$x[,"PC2"]

par(mfrow=c(1,2))

plot(PCA1, PCA2, col=best.kmeans$cluster, main = "My Best K-means")
```



- [15 Points] Compare the clustering results from the above two questions with the built-in `kmeans()` function in R. Use tables/figures to demonstrate your results and comment on your findings.

```
real.kmeans <- kmeans(data.X, n.clusters, iter.max = max.iterations, nstart = n.initiations)
real.within.var <- within.cluster.variation(data.X, real.kmeans$cluster, n.clusters)
```

```
print(sprintf("R default kmeans within-cluster variation = %f", real.within.var))
```

```
## [1] "R default kmeans within-cluster variation = 46397.671785"
```

```
print(sprintf("my.best kmeans within-cluster variation = %f", min.within.var))
```

```
## [1] "my.best kmeans within-cluster variation = 46396.020449"
```

```
t.real <- table(data.Y, real.kmeans$cluster)
props.real <- prop.table(t.real,2)
likely.digits.real <- sapply(1:n.clusters, function(i) names(sort(which(props.real[,i] > .15))))

print("R Default K-means")
```

```
## [1] "R Default K-means"
```

```
print("Acceptance threshold > 15% cluster is composed of these digits")
```

```
## [1] "Acceptance threshold > 15% cluster is composed of these digits"
```

```
for (i in 1:n.clusters) {
  print(sprintf("The most common digits assigned to cluster %d are: %s", i,
                paste(unlist(likely.digits.real[i]), collapse=" ")))
}
```

```
## [1] "The most common digits assigned to cluster 1 are: 4 7 9"
## [1] "The most common digits assigned to cluster 2 are: 1"
## [1] "The most common digits assigned to cluster 3 are: 0 2 6"
## [1] "The most common digits assigned to cluster 4 are: 3 5 8"
## [1] "The most common digits assigned to cluster 5 are: 0"
```

```
t1 <- table(data.Y, best.kmeans$cluster)
props <- prop.table(t1,2)
likely.digits <- sapply(1:n.clusters, function(i) names(which(props[,i] > .15)))

print("My best K-means")
```

```
## [1] "My best K-means"
```

```
print("Acceptance threshold > 15% cluster is composed of these digits")
```

```
## [1] "Acceptance threshold > 15% cluster is composed of these digits"
```

```
for (i in 1:n.clusters) {
  print(sprintf("The most common digits assigned to cluster %d are: %s", i,
                paste(unlist(likely.digits[i]), collapse=" ")))
}
```

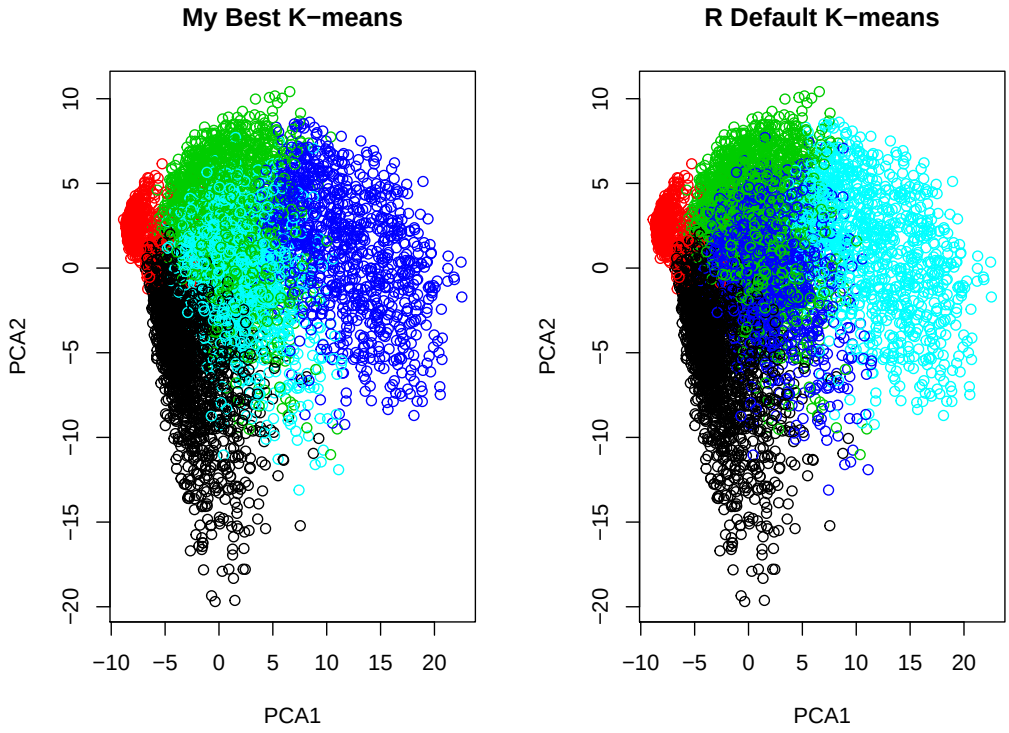
```
## [1] "The most common digits assigned to cluster 1 are: 4 7 9"
## [1] "The most common digits assigned to cluster 2 are: 1"
## [1] "The most common digits assigned to cluster 3 are: 0 2 6"
## [1] "The most common digits assigned to cluster 4 are: 0"
## [1] "The most common digits assigned to cluster 5 are: 3 5 8"
```

The final step is to compare the PCA1 and PCA2 for the default R kmeans and my best kmeans:

```
par(mfrow=c(1,2))

plot(PCA1, PCA2, col=best.kmeans$cluster, main = "My Best K-means")
```

```r
plot(PCA1, PCA2, col=real.kmeans$cluster, main = "R Default K-means")
```



The prevalent digits assigned to each cluster are the same for the default R K-means and my best K-means, though the cluster number they are assigned to is arbitrary.

The within cluster variation of my best kmeans and the default R k-means are within **0.0036%** of each other.

Question 2 [50 Points] Two-dimensional Gaussian Mixture Model

We consider an example of the EM algorithm, which fits a Gaussian mixture model to the Old Faithful eruption data. For a demonstration of this problem, see the figure provided on Wikipedia. As the end result, we will obtain the distribution parameters of the two underlying distributions. We consider the problem as follows. For this question, you are allowed to use packages that calculate the densities of normal distributions.

- We use both variables `eruptions` and `waiting`. We assume that the underlying distributions given the unobserved latent variables are both two-dimensional normal: $N(\mu_1, \Sigma_1)$ and $N(\mu_2, \Sigma_2)$, respectively, while μ_1, Σ_1, μ_2, and Σ_2 are unknow parameters that we need to solve.
- We assume that the unobserved latent variables (that indicate the membership) follow i.i.d. Bernoulli distribution, with parameter p.
- Based on the logic of an EM algorithm, we will first initiate some values of the parameters in the normal distribution. I provided a choice of them, and the normal density plots based on the initial values.
- Your goal is to write the EM algorithm that progressively updates the parameters and the latent variable distribution parameter. Eventually, we will reach a stable model fitting result that approximate the two underlying distributions, as demonstrated on the Wikipedia page. Choose a reasonable stopping criterion. To demonstrate your results, you should provide at least the following information.

```r
library(mixtools)

set.seed(1)

# load the data
```

```
faithful = as.matrix(read.table("..//data//faithful.txt"))

add.ellipse <- function(mu, sigma, ...)
{
  ellipse(mu, sigma, alpha = .05, lwd = 1, ...)
  ellipse(mu, sigma, alpha = .25, lwd = 2, ...)
}

plot.init.ellipses <- function(X, mu1, mu2, sigma1, sigma2, title) {
  par(mfrow=c(1,1))

  plot(X, main = title)
  add.ellipse(mu1, sigma1, col = "darkorange")
  add.ellipse(mu2, sigma2, col = "deepskyblue")
}

plot.ellipses <- function(X, gmm, title) {
  # plot the current fit
  par(mfrow=c(1,1))

  plot(X, main = title)
  add.ellipse(gmm$mu1, gmm$sigma1, col = "darkorange")
  add.ellipse(gmm$mu2, gmm$sigma2, col = "deepskyblue")
}

print.gmm <- function(gmm) {
  print("mu1: ")
  print(gmm$mu1)
  print("mu2: ")
  print(gmm$mu2)
  print("sigma1: ")
  print(gmm$sigma1)
  print("sigma2: ")
  print(gmm$sigma2)
  print("pi: ")
  print(gmm$hat.pi)
  print(sprintf("Converged in %d iterations", gmm$total.iterations))
}

#-------------------------------------
# my Gaussian Mixed Model EM algorithm
#-------------------------------------
my.gmm <- function(X, mu1, mu2, sigma1, sigma2, hat.pi) {

  max.iterations <- 100
  stopping.epsilon <- 1e-5

  mu1.prev <- c(1e10, 1e10)
  mu2.prev <- c(1e10, 1e10)

  d1 <- vector(mode = "list", length = max.iterations)
  d2 <- vector(mode = "list", length = max.iterations)
```

```
delta <- 1e10
iter <- 1

while (iter <= max.iterations && delta >= stopping.epsilon) {
  #----------------------------
  # E-step
  #----------------------------
  d1[[iter]] <- hat.pi * dmvnorm(X, mu1, sigma1)
  d2[[iter]] <- (1 - hat.pi) * dmvnorm(X, mu2, sigma2)
  expected.Z <- d2[[iter]]/(d1[[iter]] + d2[[iter]])

  #-----------------------------------------------------------
  # M-step
  # based on the conditional distribution, calculate the new MLE
  # of the parameters
  #-----------------------------------------------------------
  hat.pi <- mean(expected.Z)

  # save previous mu
  mu1.prev <- mu1
  mu2.prev <- mu2

  # hat_mu1 = sum( (1-ez) * x ) / sum(1-ez)
  mu1 <- colSums((1 - expected.Z) * X) / sum(1 - expected.Z)
  mu2 <- colSums(expected.Z * faithful)/ sum(expected.Z)

  sigma1 <- 0
  sigma2 <- 0
  for (i in 1:length(expected.Z)) {
    val1 <- (1 - expected.Z[i]) * (X[i,] - mu1) %*% t(X[i,] - mu1)
    sigma1 <- sigma1 + val1

    val2 <- expected.Z[i] * (X[i,] - mu1) %*% t(X[i,] - mu2)
    sigma2 <- sigma2 + val2
  }

  sigma1 <- sigma1/ sum(1-expected.Z)
  sigma2 <- sigma2/ sum(expected.Z)

  delta <- (sum((mu1 - mu1.prev)^2) + sum((mu2 - mu2.prev)^2))/2

  total.iterations <- iter
  iter <- iter + 1
}

results <- list()
results$mu1 <- mu1
results$mu2 <- mu2
results$sigma1 <- sigma1
results$sigma2 <- sigma2
results$hat.pi <- hat.pi
results$d1 <- d1
results$d2 <- d2
```

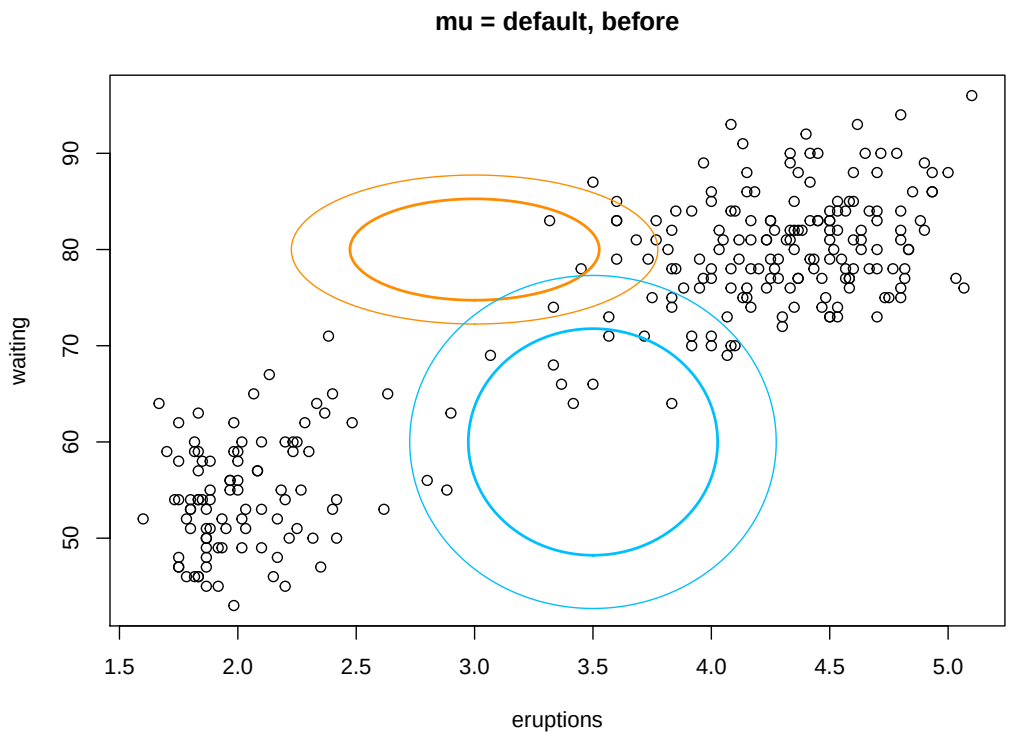
```
  results$expected.Z <- expected.Z
  results$total.iterations <- total.iterations

  return(results)
}
```

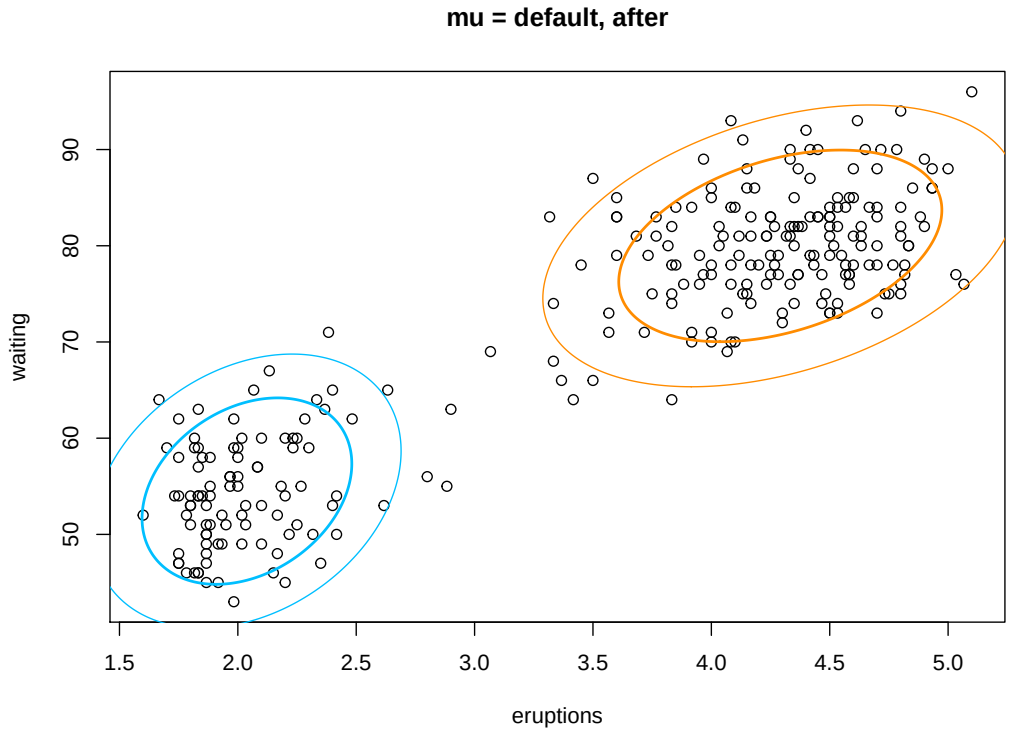
- The distribution parameters p, μ_1, Σ_1, μ_2, and Σ_2

```
# the parameters
mu1 <- c(3, 80)
mu2 <- c(3.5, 60)
sigma1 <- matrix(c(0.1, 0, 0, 10), 2, 2)
sigma2 <- matrix(c(0.1, 0, 0, 50), 2, 2)
hat.pi <- 0.5

plot.init.ellipses(faithful, mu1, mu2, sigma1, sigma2, "mu = default, before")
```




```
gmm <- my.gmm(faithful, mu1, mu2, sigma1, sigma2, hat.pi)
plot.ellipses(faithful, gmm, "mu = default, after")
```

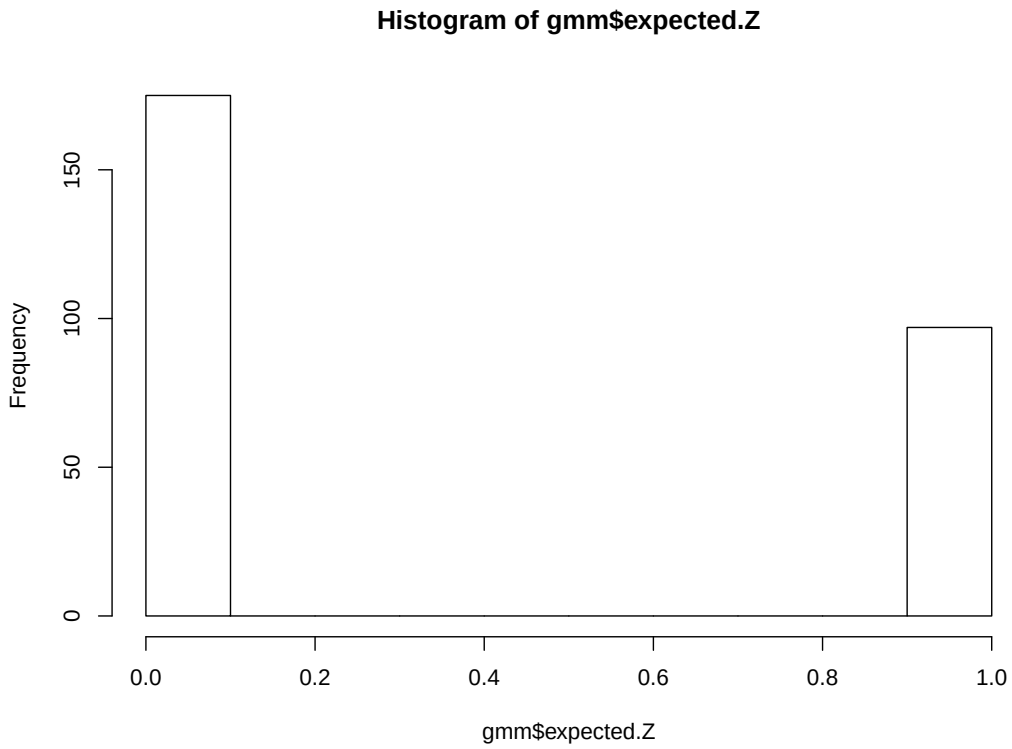



```
print.gmm(gmm)
```

```
## [1] "mu1: "
## eruptions   waiting
##  4.291298 79.987034
## [1] "mu2: "
## eruptions   waiting
##  2.038332 54.499751
## [1] "sigma1: "
##      eruptions    waiting
## [1,] 0.1679362  0.9157934
## [2,] 0.9157934 35.7802998
## [1] "sigma2: "
##      eruptions    waiting
## [1,] 0.0707754  0.4535782
## [2,] 0.4535782 33.8523549
## [1] "pi: "
## [1] 0.3566475
## [1] "Converged in 14 iterations"
```

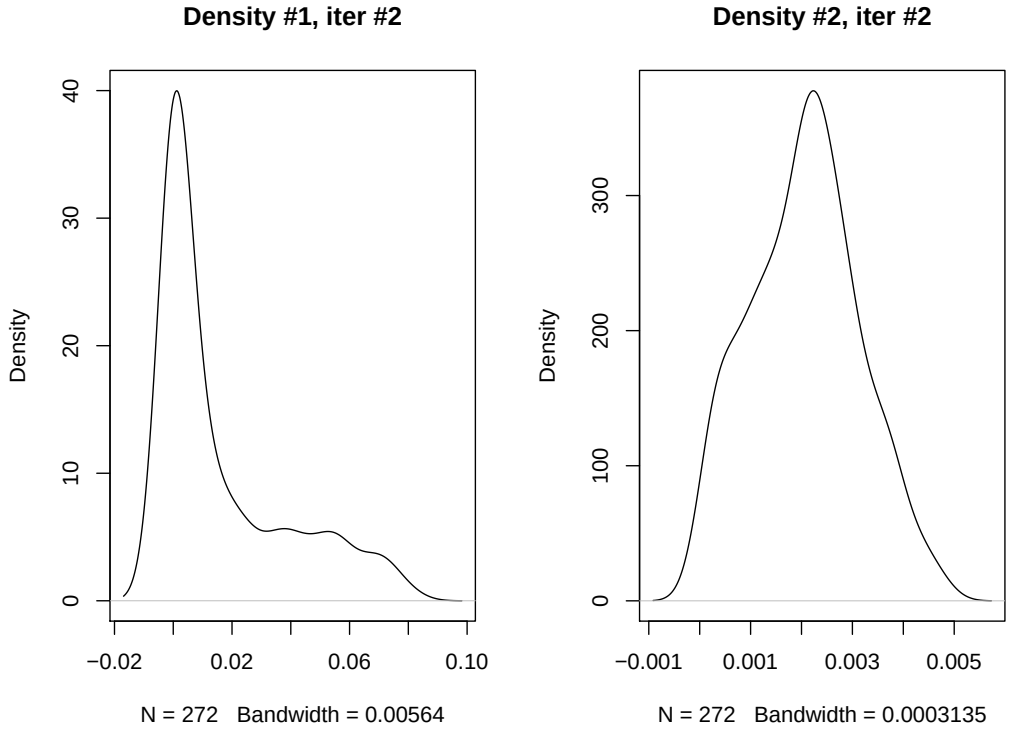
- A histogram of the underlying probabilities of the latent variables

```
# histograms
hist(gmm$expected.Z)
```



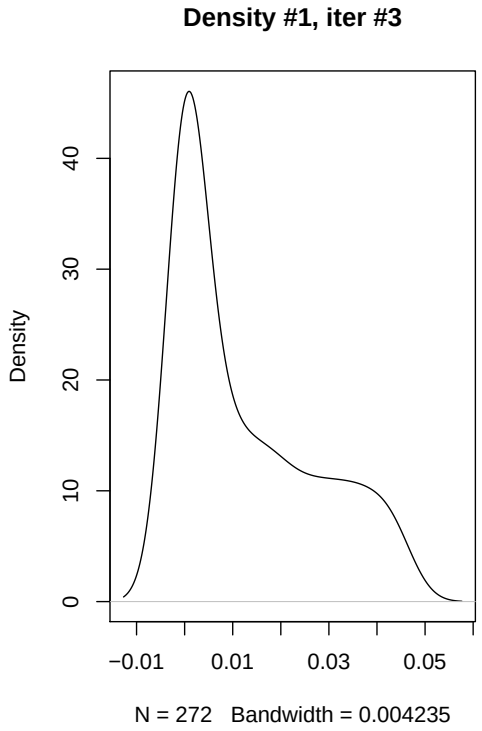
- Plot the normal densities at the 2nd, 3rd, 4th and the final iteration of your algorithm

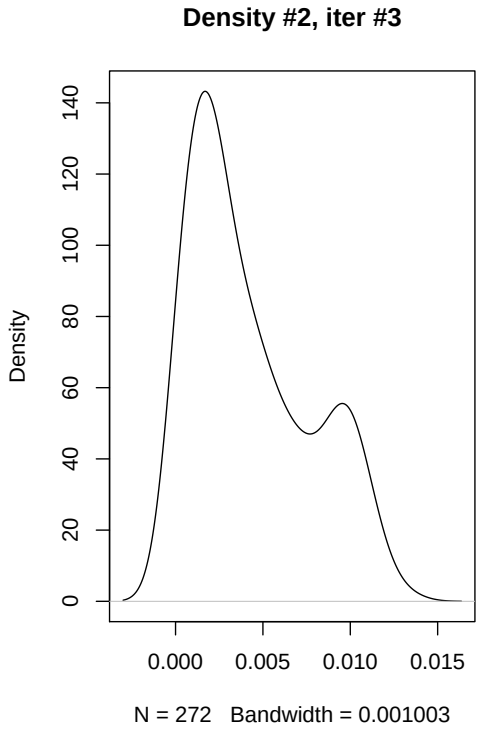
```
par(mfrow=c(1,2))
plot(density(gmm$d1[[2]]), main= "Density #1, iter #2")
plot(density(gmm$d2[[2]]), main="Density #2, iter #2")
```



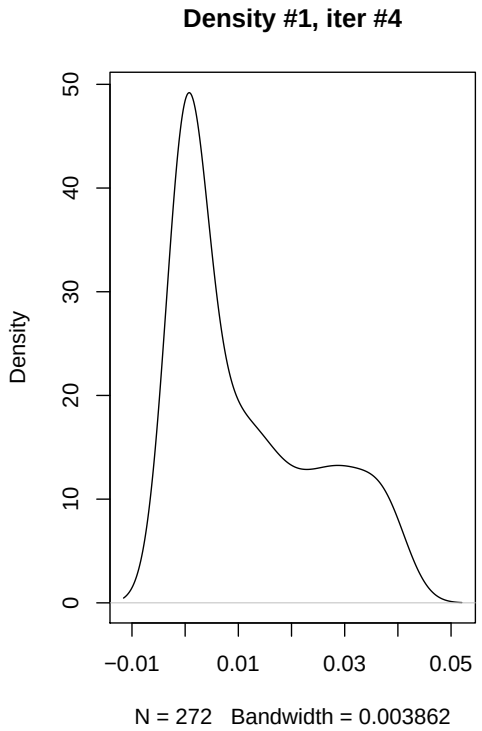

```
plot(density(gmm$d1[[3]]), main="Density #1, iter #3")
```

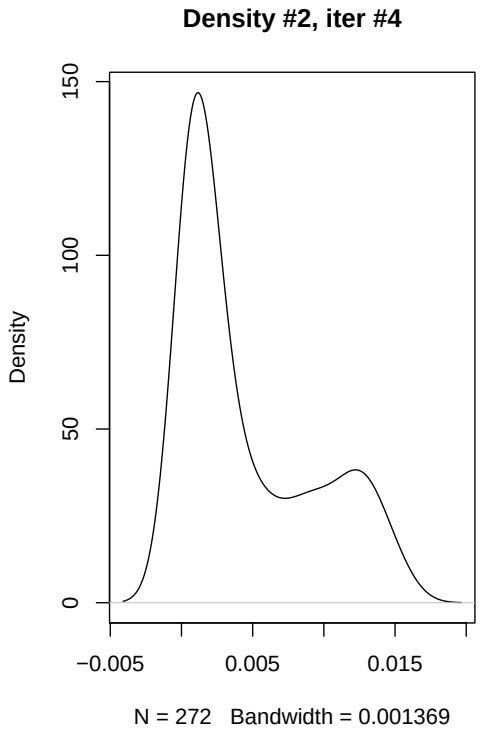
```
plot(density(gmm$d2[[3]]), main="Density #2, iter #3")
```





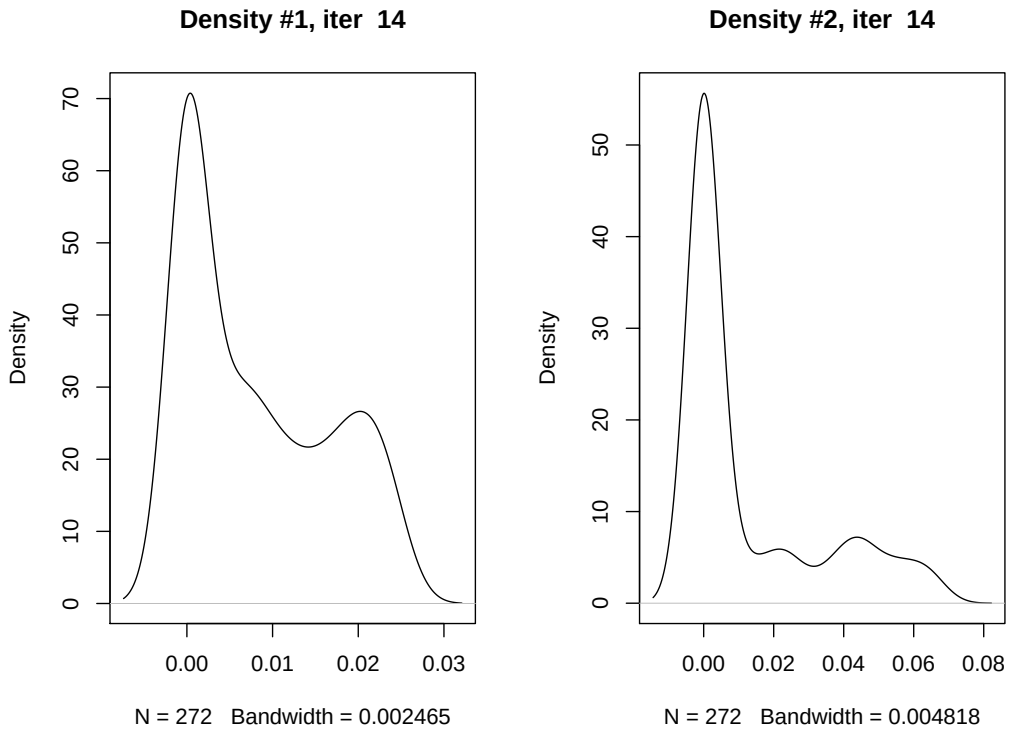

```
plot(density(gmm$d1[[4]]), main="Density #1, iter #4")
plot(density(gmm$d2[[4]]), main="Density #2, iter #4")
```






```
str1 <- paste("Density #1, iter ", gmm$total.iterations)
str2 <- paste("Density #2, iter ", gmm$total.iterations)
```

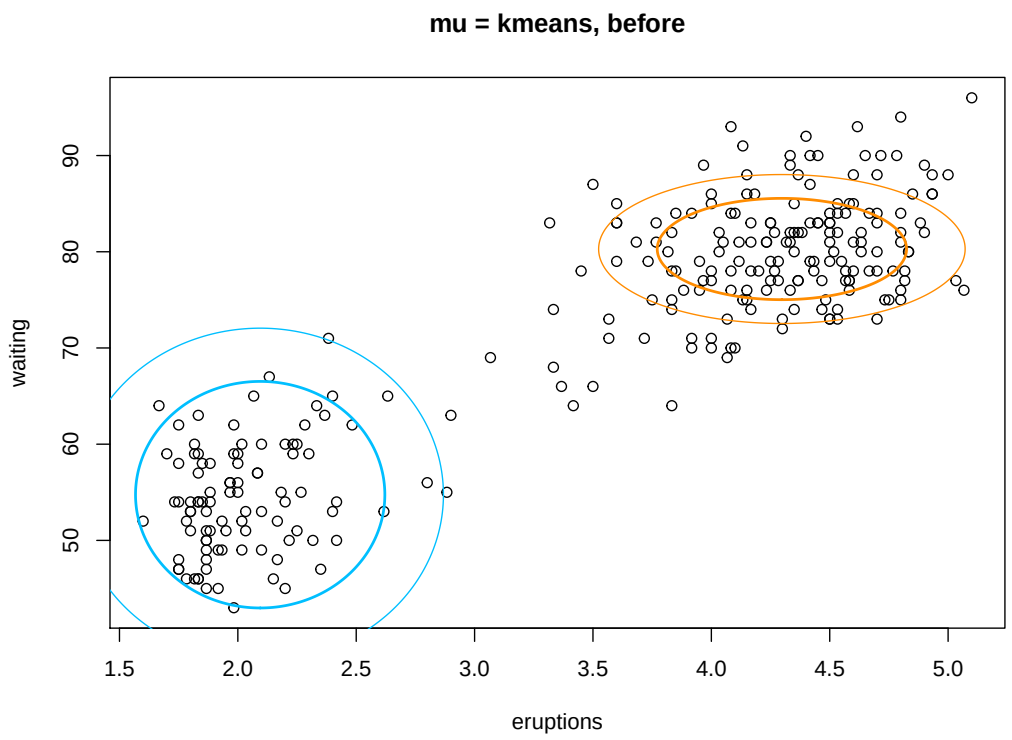
```
plot(density(gmm$d1[[gmm$total.iterations]]), main=str1)
plot(density(gmm$d2[[gmm$total.iterations]]), main=str2)
```



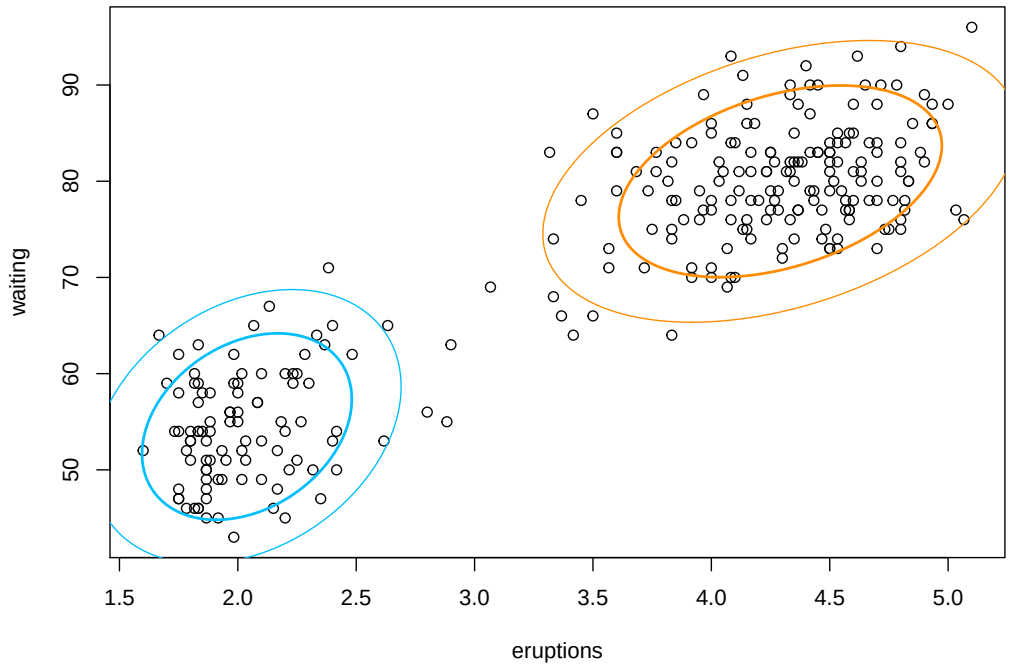
- Now, experiment a very different initial value of the parameters and rerun the algorithm. Comment on the efficiency and convergence speed of this algorithm.

In the following code, I will first show a good result, using R default kmeans to find the mus and then show a bad result, pursposefully choosing bad mus.

```
# good result
k <- kmeans(faithful, 2)
mu1 <- k$centers[1,]
mu2 <- k$centers[2,]
plot.init.ellipses(faithful, mu1, mu2, sigma1, sigma2, "mu = kmeans, before")
```




```
gmm <- my.gmm(faithful, mu1, mu2, sigma1, sigma2, hat.pi)
plot.ellipses(faithful, gmm, "mu = kmeans, after")
```



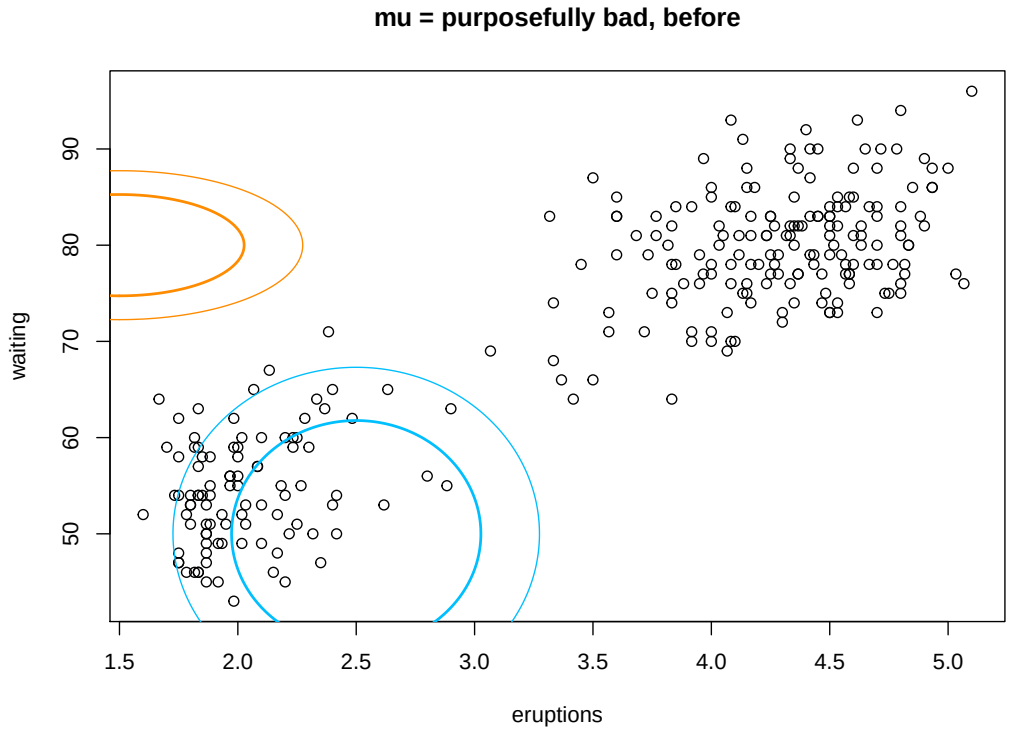

```
print.gmm(gmm)
```

```
## [1] "mu1: "
## eruptions   waiting
##  4.291303 79.987087
```

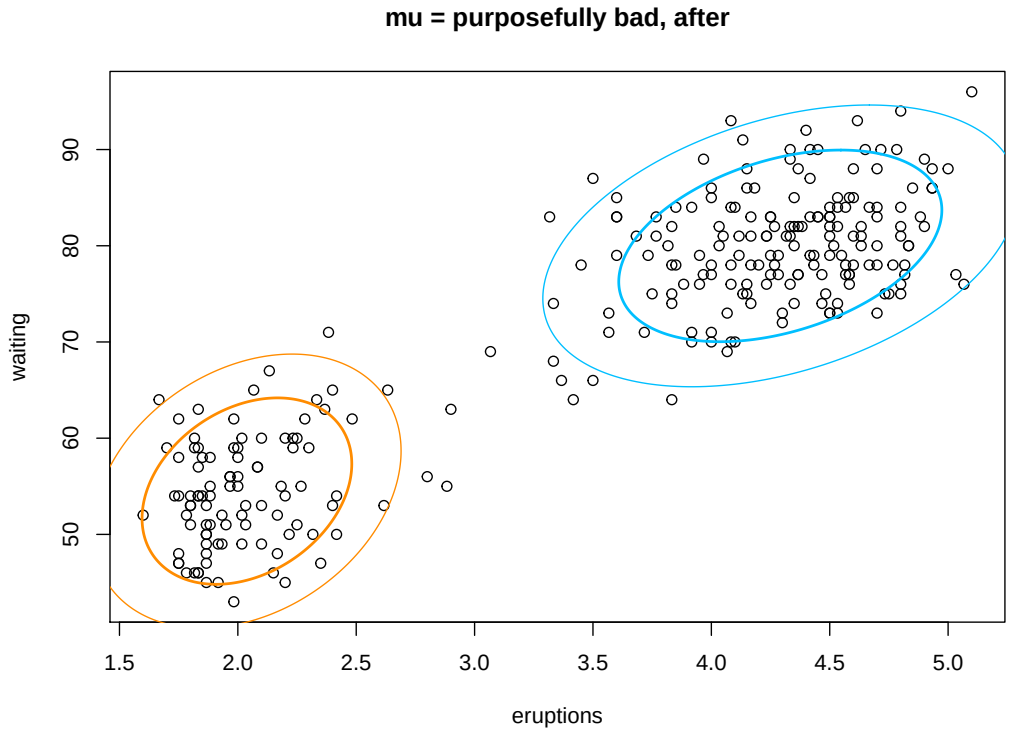
```
## [1] "mu2: "
## eruptions   waiting
##  2.038339 54.499836
## [1] "sigma1: "
##      eruptions   waiting
## [1,] 0.1679303  0.915727
## [2,] 0.9157270 35.779677
## [1] "sigma2: "
##       eruptions    waiting
## [1,] 0.07078155  0.4536584
## [2,] 0.45365844 33.8532018
## [1] "pi: "
## [1] 0.35665
## [1] "Converged in 6 iterations"
```

and now the bad choice of mus:

```
mu1 <- c(1.5,80)
mu2 <- c(2.5, 50)
plot.init.ellipses(faithful, mu1, mu2, sigma1, sigma2, "mu = purposefully bad, before")
```




```
gmm <- my.gmm(faithful, mu1, mu2, sigma1, sigma2, hat.pi)
plot.ellipses(faithful, gmm, "mu = purposefully bad, after")
```




```
print.gmm(gmm)
```

```
## [1] "mu1: "
## eruptions   waiting
##  2.038321 54.499610
## [1] "mu2: "
## eruptions   waiting
##  4.291289 79.986944
## [1] "sigma1: "
##       eruptions    waiting
## [1,] 0.07076514  0.4534445
## [2,] 0.45344454 33.8509464
## [1] "sigma2: "
##      eruptions    waiting
## [1,] 0.1679462  0.9159045
## [2,] 0.9159045 35.7813455
## [1] "pi: "
## [1] 0.6433568
## [1] "Converged in 48 iterations"
```

The algorithm is very dependent upon good first choices of mus for efficiency. For the default choices, it converges in **14 iterations**, for k-means mus it converges in **6 iterations**, for a purposefully bad choice of mus, it takes **48 iterations** to converge.