Mixture of Gaussians and the EM Algorithm

Silvana Alvarez - Sergio Quntanilla

2025-03-12

Point 1

Implement the EM algorithm to estimate the parameters of a mixture of Gaussians when we have data of any dimension and any number of classes.

Solution:

```
# source(file = "EM_function.R")
library(ggplot2)
library(mvtnorm)
library(reshape)
library(tidyverse)
```

Declaring our main function first.

```
#    EM ALGORITHM
#----------------------

# k: Number of classes (j)
# D: Dimensions
# N: Number of observations (i)
# max_iter: maximum number of iterations
# tol: convergence tolerance
library(birdring)
```

```
## Warning: package 'birdring' was built under R version 4.4.3
```

```
##
## Attaching package: 'birdring'
```

```
## The following object is masked from 'package:mvtnorm':
##
##     dmvnorm
```

```
em_gaussian_mixture <- function(X, K, max_iter = 100, tol = 1e-6) {

N <- nrow(X)  # Number of data points
D <- ncol(X)  # Number of dimensions
Alpha = NULL
```

```r
# Set initial parameters
pi <-  rep(1/K, K)    # Mixing coefficients

# mu
idx <- sample(1:N, K)
mu <- X[idx, , drop = FALSE]

# Covariance Matrix
sigma <- list()
for (j in 1:K) {
  sigma[[j]] <- diag(1, D)
}

# Initialize log-likelihood
prev_log_lik <- -Inf
log_likelihood <- c()

# Main EM loop
for (iter in 1:max_iter) {
  cat("Iteration:", iter, "\n")

#----------------------------
#   EXPECTATION (E-STEP)
#----------------------------
cat("  E-step: Computing responsibilities\n")

# Initialize prob matrix (N x K)
gamma <- matrix(0, nrow = N, ncol = K)


# Compute responsibilities for each data point and cluster
for (i in 1:N) {
  for (j in 1:K) {
    # Calculate multivariate Gaussian density
    gamma[i, j] <- pi[j] * dmvnorm(X[i, ], mean = mu[j, ], sigma = sigma[[j]])
  }
  # Normalize responsibilities for data point i
  if (sum(gamma[i, ]) > 0) {
    gamma[i, ] <- gamma[i, ] / sum(gamma[i, ])
  }
}
Alpha = cbind(Alpha,gamma)

#----------------------------
#   MAXIMIZATION (M-STEP)
#----------------------------
cat("  M-step: Updating parameters\n")

# Calculate effective number of points in each class
N_k <- colSums(gamma) # weighted sum (prob of being in each class)

# Update mixing coefficients
pi <- N_k / N
```

```r
# Update means
for (j in 1:K) {
  if (N_k[j] > 0) {
    # Weighted average of data points
    mu[j, ] <- t(gamma[, j]) %*% X / N_k[j]
  }
}

# Update covariance matrices
for (j in 1:K) {
  sigma[[j]] <- matrix(0, nrow = D, ncol = D)
  if (N_k[j] > 0) {
    for (i in 1:N) {
      diff <- X[i, ] - mu[j, ]
      sigma[[j]] <- sigma[[j]] + gamma[i, j] * (diff %*% t(diff))
    }
    sigma[[j]] <- sigma[[j]] / N_k[j]

    # Add small regularization to prevent singularity
    sigma[[j]] <- sigma[[j]] + diag(1e-6, D)
    }
  }

#--------------------------
# Check convergence
#--------------------------
# Compute log-likelihood
current_log_lik <- 0
for (i in 1:N) {
  # Sum weighted probability over all clusters
  p_xi <- 0
  for (j in 1:K) {
    p_xi <- p_xi + pi[j] * dmvnorm(X[i, ], mean = mu[j, ], sigma = sigma[[j]])
  }
  current_log_lik <- current_log_lik + log(p_xi)
}

log_likelihood <- c(log_likelihood, current_log_lik)

# Check for convergence
improvement <- abs(current_log_lik - prev_log_lik)
cat("  Log-likelihood:", current_log_lik, "Improvement:", improvement, "\n")

if (iter > 1 && improvement < tol) {
  cat("Converged after", iter, "iterations\n")
  break
}

prev_log_lik <- current_log_lik
}

# Return results
return(list(
```

```
  pi = pi,                    # Mixing coefficients
  mu = mu,                    # Means
  sigma = sigma,              # Covariance matrices
  gamma = gamma,              # Responsibilities
  Alpha = Alpha,               # Iteration tracking
  log_likelihood = log_likelihood,  # Log-likelihood history
  K = K,                      # Number of clusters
  D = D                       # Number of dimensions
))
}

#----------------
#   PREDICTION
#----------------
# Function to predict cluster assignments
predict_cluster <- function(model, X_new) {
  N_new <- nrow(X_new)
  K <- model$K

  # Calculate probabilities for each point and cluster
  probs <- matrix(0, nrow = N_new, ncol = K)
  for (i in 1:N_new) {
    for (j in 1:K) {
      probs[i, j] <- model$pi[j] * dmvnorm(X_new[i, ], model$mu[j, ], model$sigma[[j]])
    }
  }

  # Return most likely cluster
  return(max.col(probs))
}

#------------------------
#   plot   convergence
#------------------------
# Function to plot convergence
plot_convergence <- function(model) {
  plot(model$log_likelihood, type = "o",
       xlab = "Iteration", ylab = "Log-likelihood",
       main = "EM Algorithm Convergence")
}
```

```
#Example usage:
library(MASS)  # For generating multivariate normal data
```

```
##
## Attaching package: 'MASS'
```

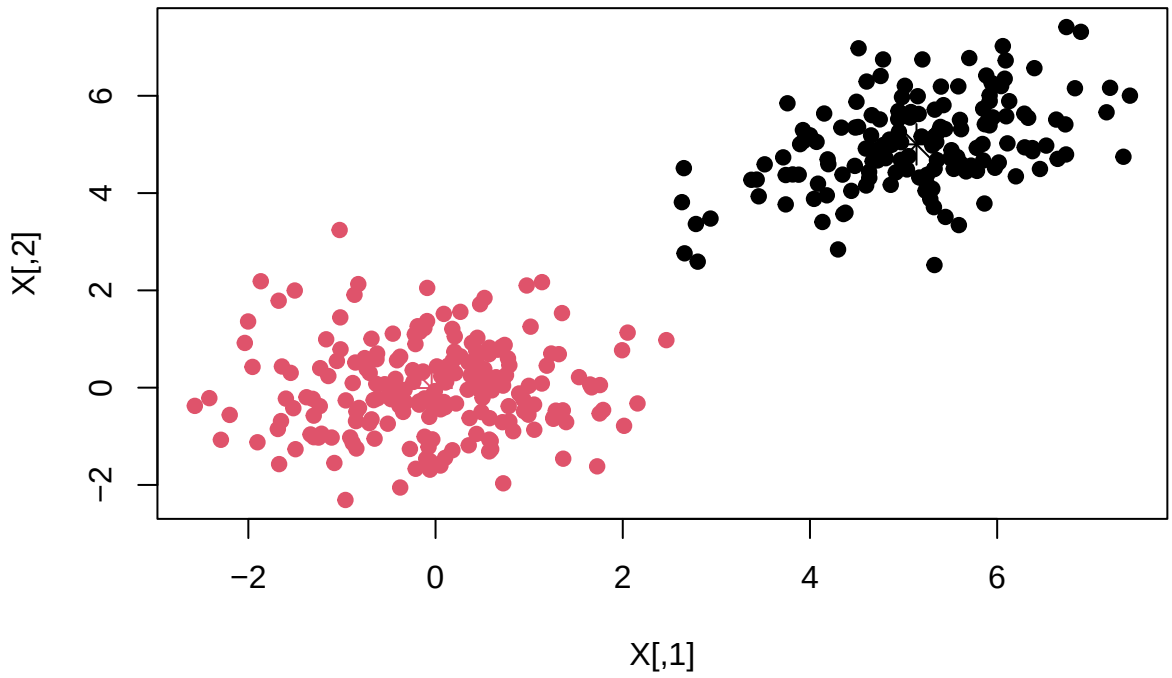
```
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
# Generate some example data
set.seed(123)
n1 <- 200
n2 <- 150
X1 <- mvrnorm(n1, mu = c(0, 0), Sigma = matrix(c(1, 0, 0, 1), 2, 2))
X2 <- mvrnorm(n2, mu = c(5, 5), Sigma = matrix(c(1, 0.5, 0.5, 1), 2, 2))
X <- rbind(X1, X2)

# Set initial parameters
K <- 2
D <- 2

# Run EM algorithm with initial parameters
model <- em_gaussian_mixture(X, K, max_iter = 50)
```

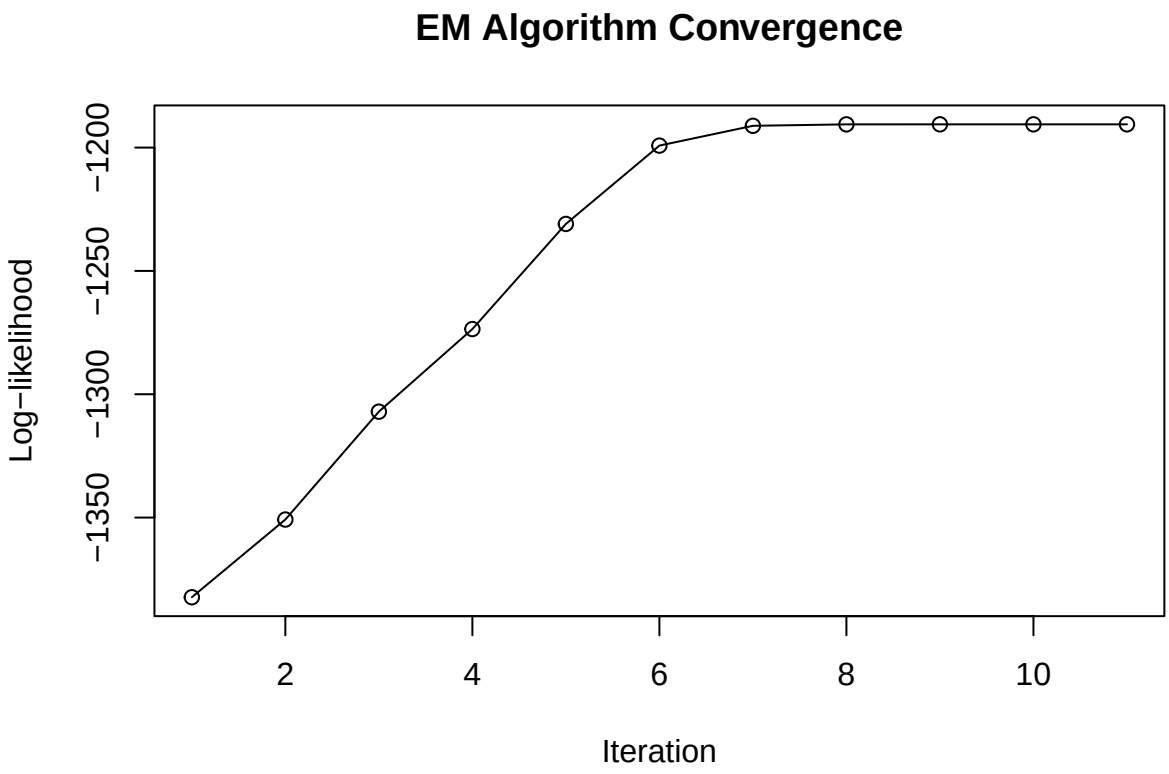
```
## Iteration: 1
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1382.304 Improvement: Inf
## Iteration: 2
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1350.8 Improvement: 31.50387
## Iteration: 3
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1307.065 Improvement: 43.73533
## Iteration: 4
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1273.576 Improvement: 33.48868
## Iteration: 5
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1230.952 Improvement: 42.62426
## Iteration: 6
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1199.193 Improvement: 31.7585
## Iteration: 7
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1191.172 Improvement: 8.021464
## Iteration: 8
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1190.582 Improvement: 0.5896712
## Iteration: 9
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1190.577 Improvement: 0.005030017
## Iteration: 10
##   E-step: Computing responsibilities
```

```
##   M-step: Updating parameters
##   Log-likelihood: -1190.577 Improvement: 1.902908e-05
## Iteration: 11
##   E-step: Computing responsibilities
##   M-step: Updating parameters
##   Log-likelihood: -1190.577 Improvement: 6.8791e-08
## Converged after 11 iterations
```

```r
# Plot results
plot(X, col = predict_cluster(model, X), pch = 19)
points(model$mu, col = 1:model$K, pch = 8, cex = 2)
```




```r
plot_convergence(model)
```



Point 2

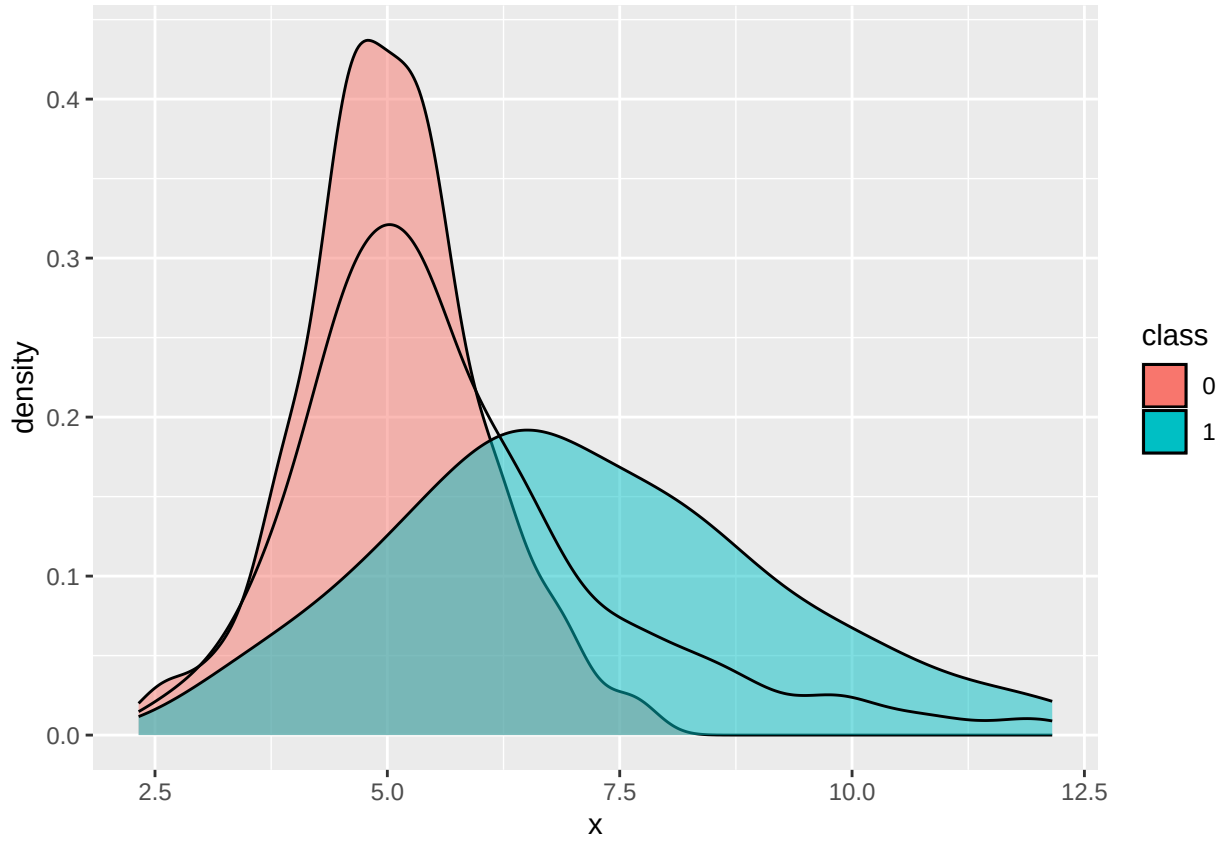
Check that it works for synthetic data generated according to a mixture of Gaussians in:

a) 1 dimensions

Solution: First, we generate our 1d synthetic data and plot it to see what to expect.

```
Mu1 = 5
Mu2 = 7
S1 = 1
S2 = 2

pi = 1/3
n = 300

set.seed(100)
data = matrix(0, n, 1)
z = rep(0,n)
for (i in 1:n){
  z[i] = rbinom(1,1,pi)
  if (z[i] ==0){
    data[i,] = rnorm(1, Mu1,S1)
```

```
  }else{
    data[i,] = rnorm(1, Mu2,S2)
  }
}

to.plot = data.frame(x = data[,1],
                     class =  as.factor(z))
ggplot(to.plot) + geom_density(aes(x=x,fill=class), alpha=.5) + geom_density(aes(x=x))
```



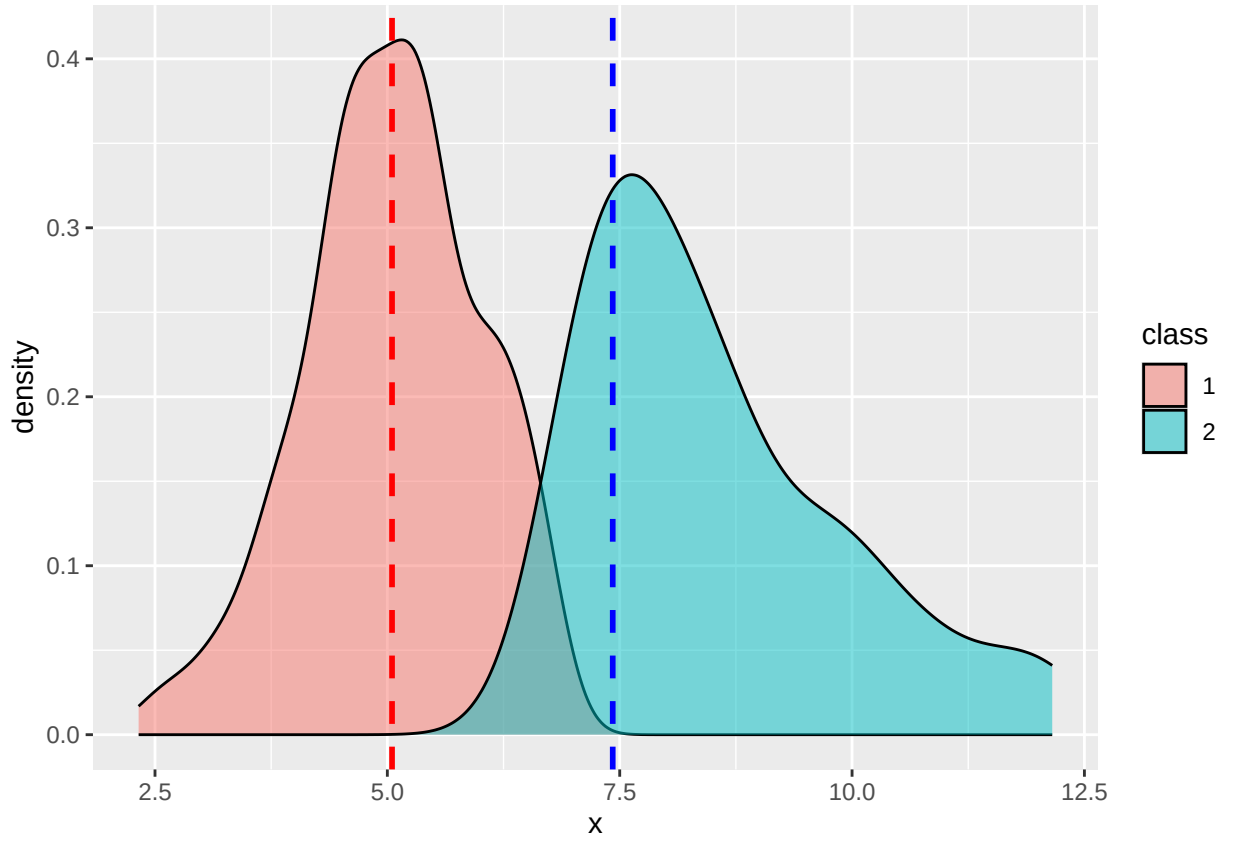
The black line through the two distributions shows the mixed distribution curve. Next, we run our function to produce a model.

```
X=data
K=2
# Run EM algorithm with initial parameters
model_1d <- em_gaussian_mixture(X, K, max_iter = 50)
```

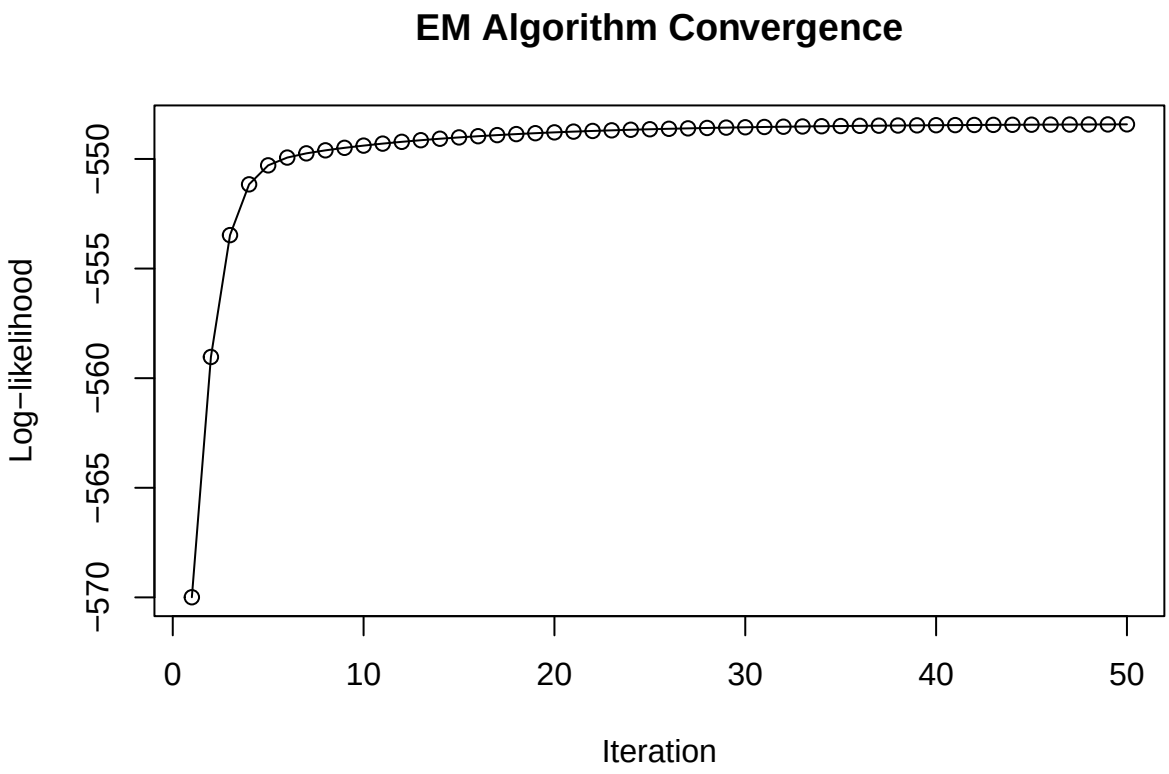
Next, we look at our output graphs. We can see this example converges very quickly, which seems to make sense when you look at how little the distribution changed in the first 8 iterations. Our approximation was reasonably accurate, so now we move on to testing in other dimensions.

```
# Plot results
to.plot = data.frame(x = data[,1],
                     class = as.factor(predict_cluster(model_1d, X)))
```

```
ggplot(to.plot) + geom_density(aes(x=x,fill=class), alpha=.5)+
  geom_vline(aes(xintercept = model_1d$mu[1]), color = "red", linetype = "dashed", size = 1)+
  geom_vline(aes(xintercept = model_1d$mu[2]), color = "blue", linetype = "dashed", size = 1)
```

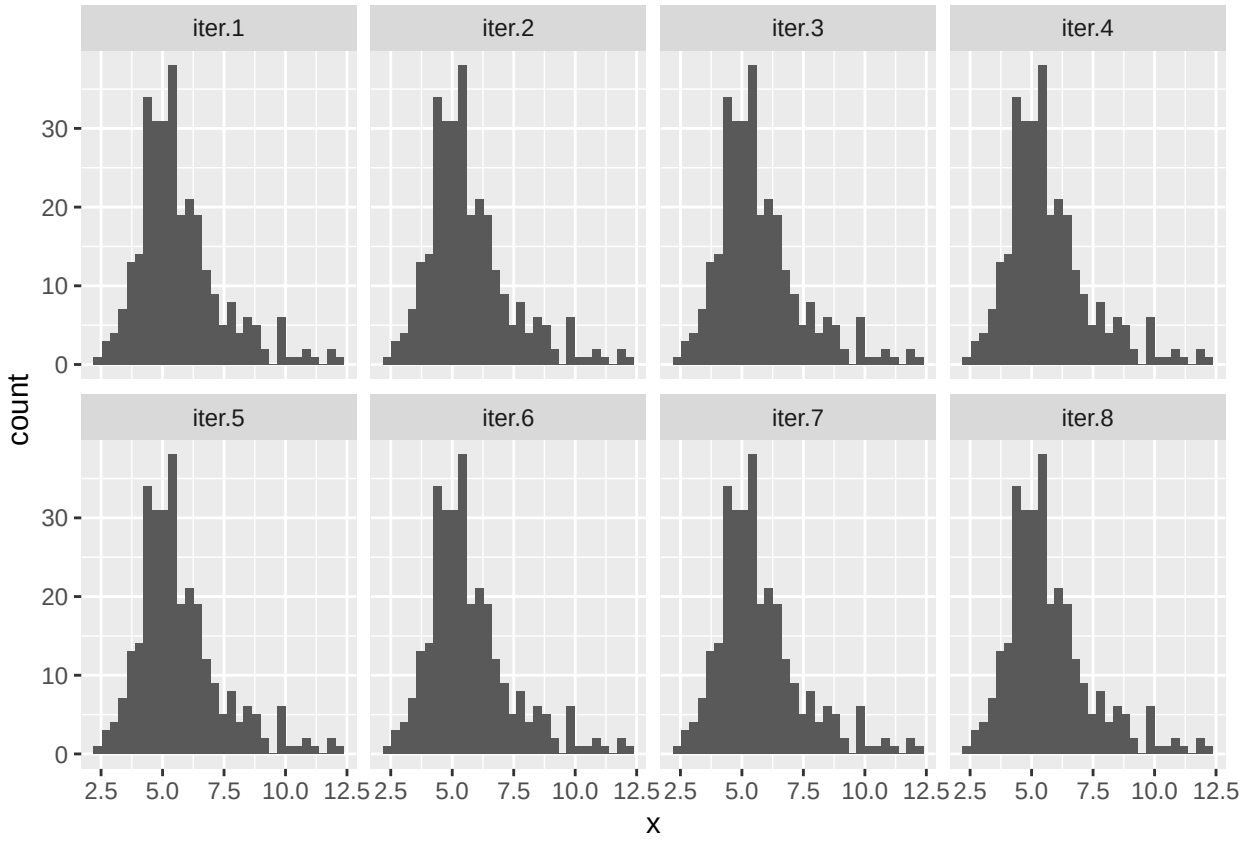



```
plot_convergence(model_1d)
```




```
to.plot = data.frame(x = data[,1],
                     iter = model_1d$Alpha[,1:8])
to.plot <- pivot_longer(to.plot, cols = -c("x"))

ggplot(to.plot)+aes(x, color = value)+
  geom_histogram()+facet_wrap(~name, nrow = 2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



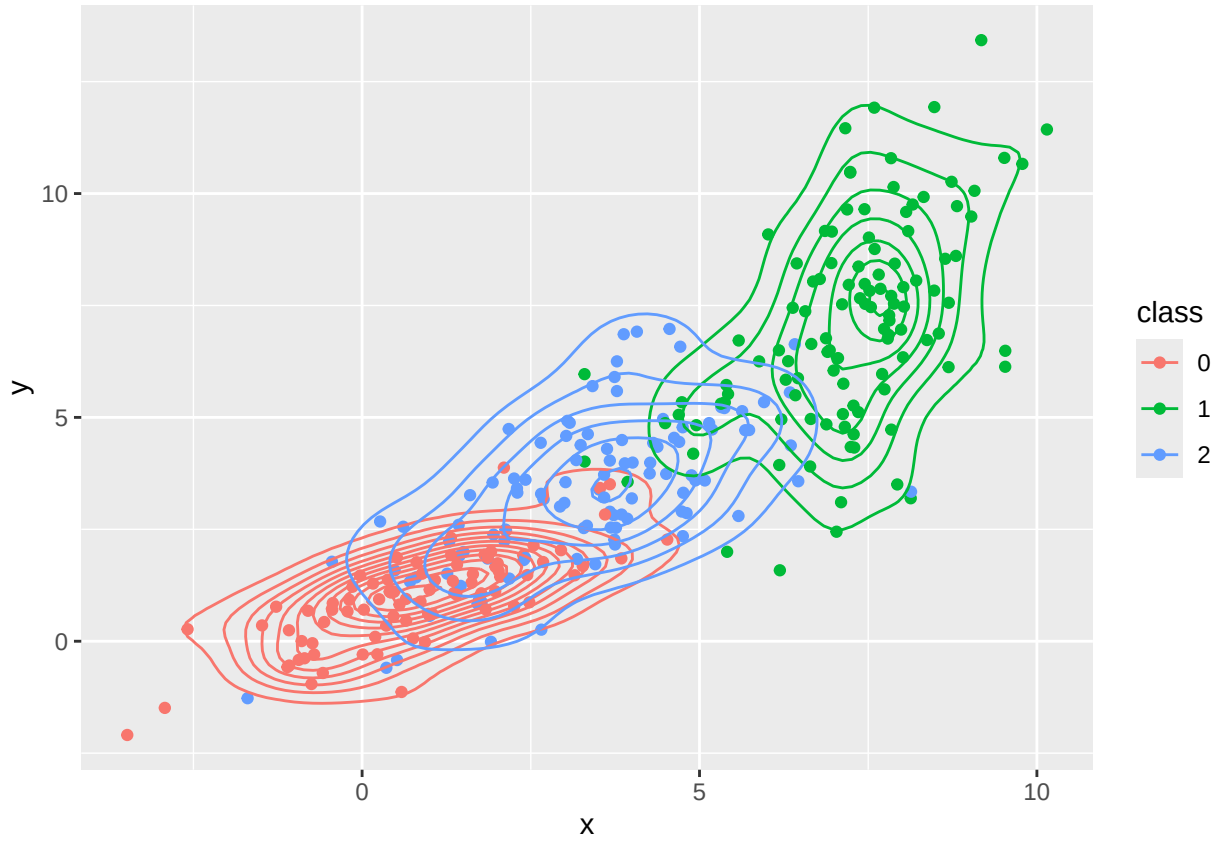
b) 2 dimensions

Solution:

In this instance, we also try with 3 groups rather than just 2 to see how it behaves. Again, we generate our data and see what it looks like.

```
library(ggplot2)
library(mvtnorm)
Mu1 = c(1,1)
Mu2 = c(7,7)
Mu3 = c(3,3)
Sigma1 = matrix(c(2, 1, 1, 1), 2,2)
Sigma2 = matrix(c(2, 2, 2, 5), 2,2)
Sigma3 = matrix(c(3, 2, 2, 3), 2,2)

pi = 1/3
n = 300

set.seed(100)
data = matrix(0, n, 2)
z = rep(0,n)
for (i in 1:n){
  z[i] = runif(1) #generate a random val to determine from what dist it comes
  if (z[i] <= 0.33){
```

```
    data[i,] = rmvnorm(1, Mu1,Sigma1)
    z[i] = 0
  }else if (z[i] >= 0.67){
    data[i,] = rmvnorm(1, Mu2,Sigma2)
    z[i] = 1
  }else {
    data[i,] = rmvnorm(1, Mu3,Sigma3)
    z[i] = 2
  }
}

to.plot = data.frame(x = data[,1],
                     y = data[,2],
                     class =  as.factor(z))
ggplot(to.plot)+ aes(x, y, color = class)+
  geom_point()+geom_density_2d()
```

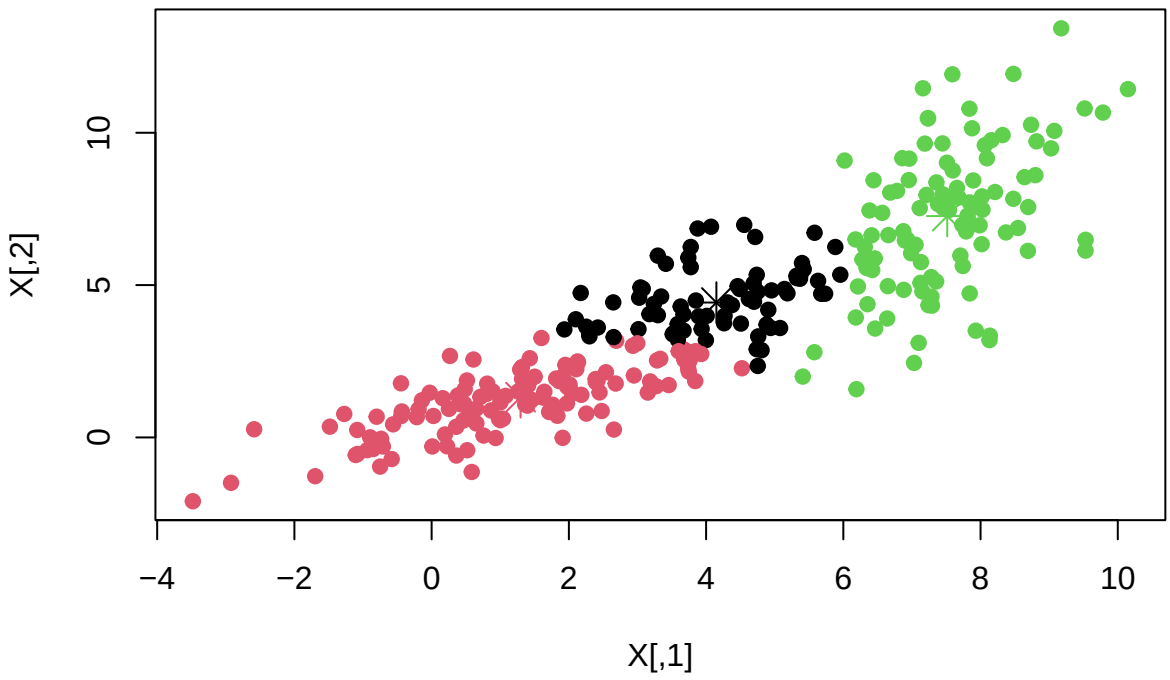


We run the model with K=3 for our 3 classes.

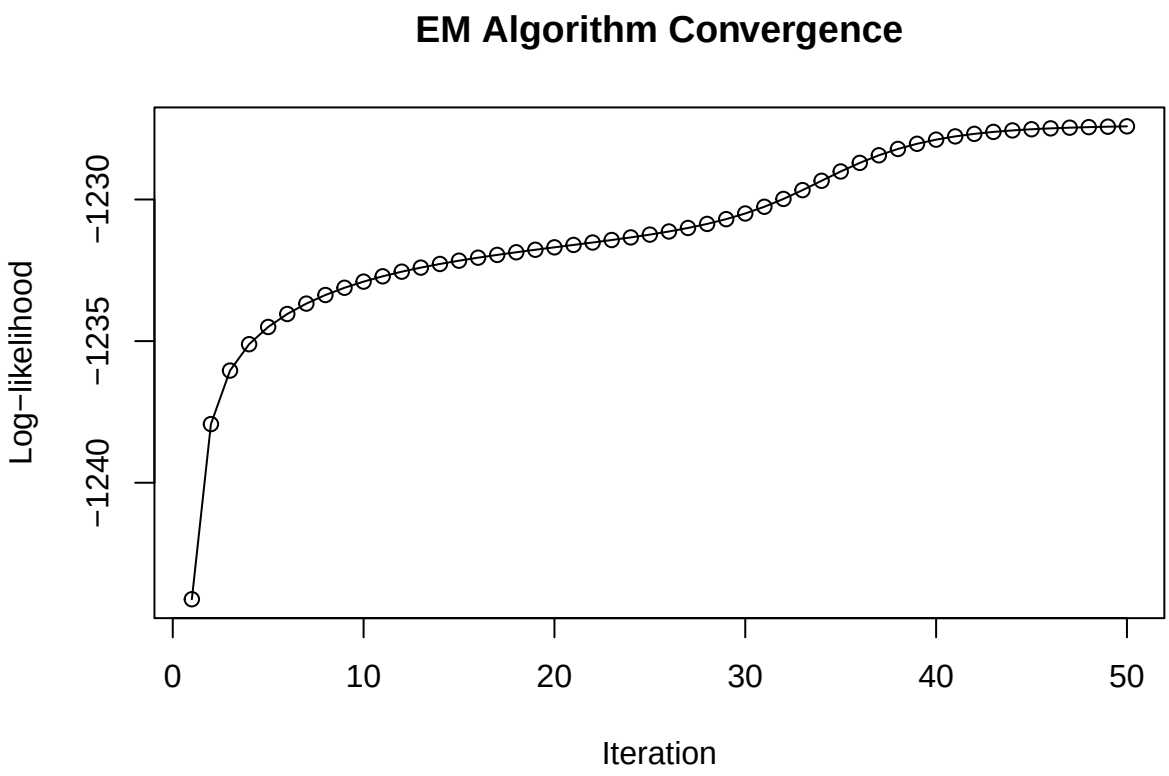
```
X=data
K=3
# Run EM algorithm with initial parameters
model_2d <- em_gaussian_mixture(X, K, max_iter = 50)
```

Looking at the plots we produce, it shows the distinction between the classes and means pretty closely to the real case.

```r
# Plot results
plot(X, col = predict_cluster(model_2d, X), pch = 19)
points(model_2d$mu, col = 1:model_2d$K, pch = 8, cex = 2)
```

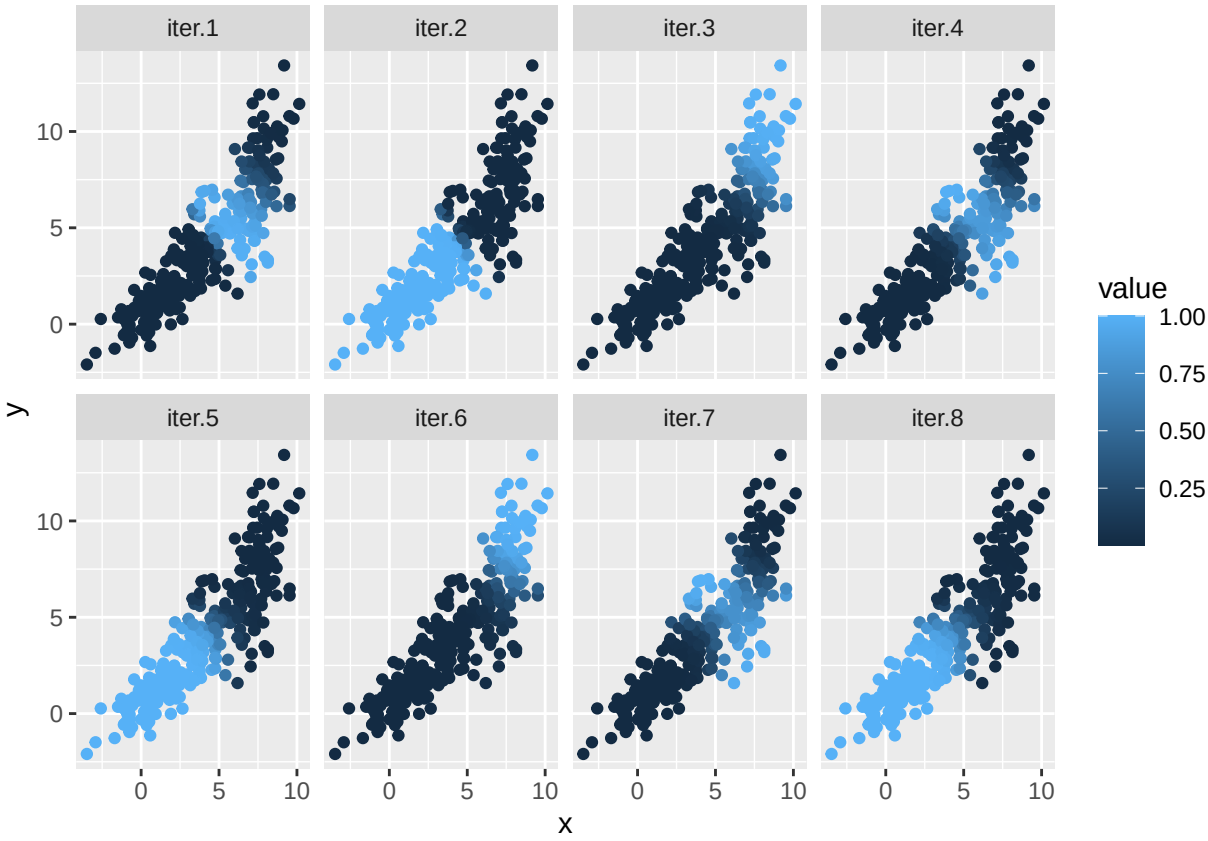



```r
plot_convergence(model_2d)
```



Here we can see a bit more variation before convergence, which again lines up with what we expect from the convergence plot above.

```
to.plot = data.frame(x = data[,1],
                     y = data[,2],
                     iter = model_2d$Alpha[,1:8])

to.plot = melt(to.plot, id.vars = c("x", "y"))

# model_2d$Alpha
ggplot(to.plot)+aes(x,y, color = value)+
  geom_point()+facet_wrap(~variable, nrow = 2)
```



c) 3 dimensions

Solution: Lastly, we try working with 3 dimensions.

```
library(rgl)
Mu1 = c(1,1,1)
Mu2 = c(5,5,5)
Sigma1 = matrix(c(2, 1, 1, 1, 2, 1,1,1,2), 3,3)
Sigma2 = matrix(c(2, 2, 1, 2, 2, 2,1,2,2), 3,3)

pi = 1/3
n = 300

set.seed(100)
data = matrix(0, n, 3)
z = rep(0,n)
for (i in 1:n){
  z[i] = rbinom(1,1,pi)
  if (z[i] ==1){
    data[i,] = rmvnorm(1, Mu1,Sigma1)
  }else{
    data[i,] = rmvnorm(1, Mu2,Sigma2)
  }
}
```

```
to.plot = data.frame(x = data[,1],
                     y = data[,2],
                     z = data[,2],
                     class =  as.factor(z))

mycolors <- c('royalblue1', 'darkorange')
color <- mycolors[ as.numeric(to.plot$class) ]
plot3d(to.plot, col = color)
```

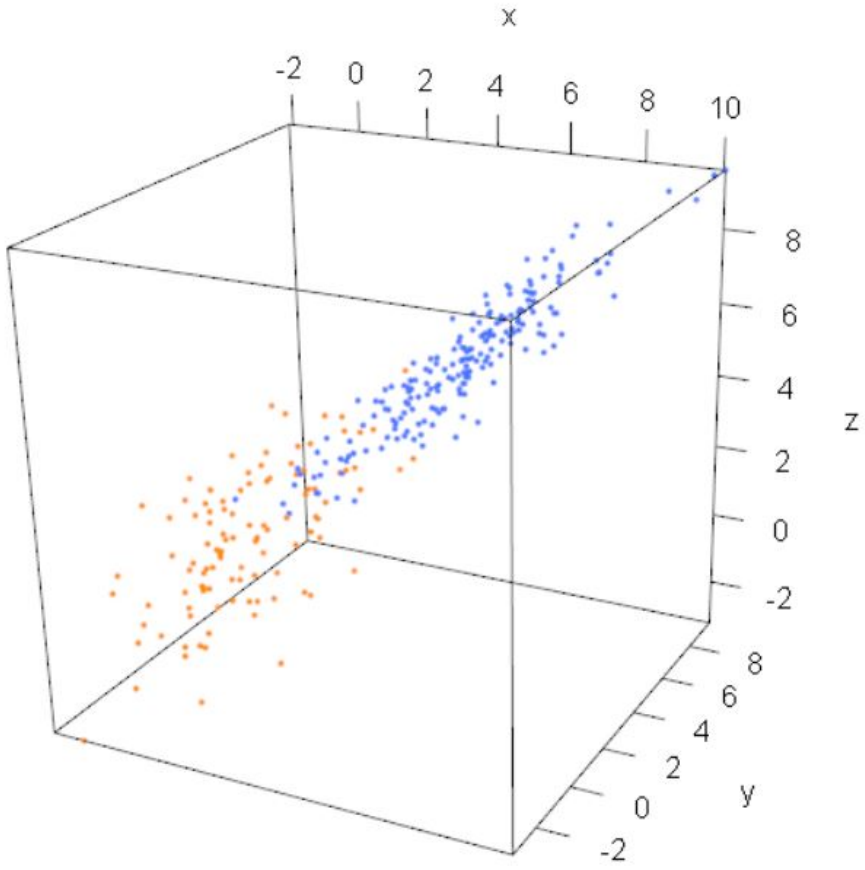


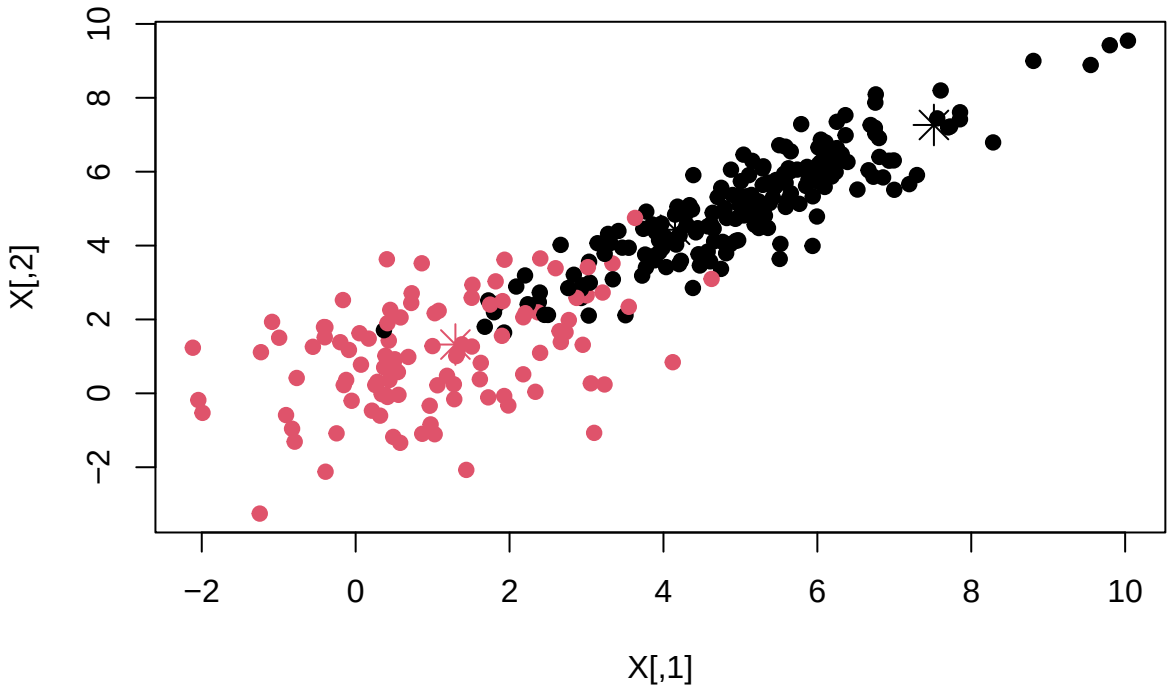
Figure 1: expected output plot

We can see a pretty clear divide between the two groups, so now we run the function for them.

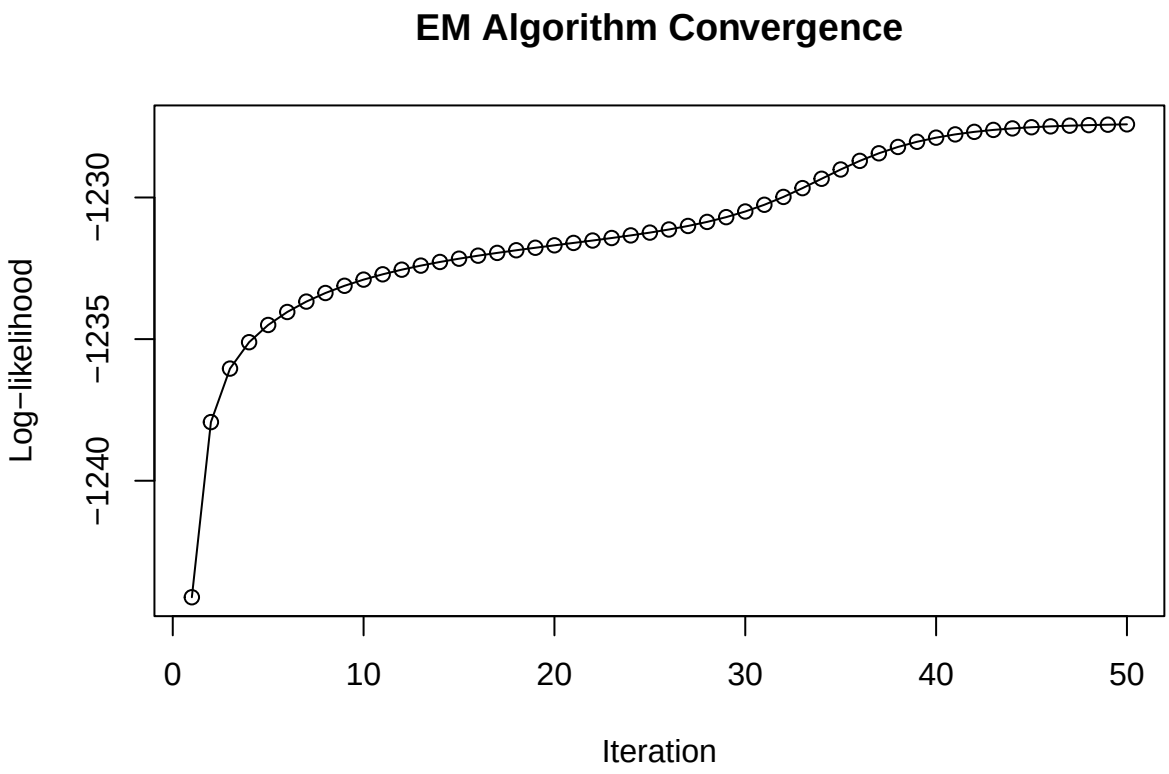
```
X=data
K=2
# Run EM algorithm with initial parameters
```

```r
model_3d <- em_gaussian_mixture(X, K, max_iter = 50)
```

```r
# Plot results
plot(X, col = predict_cluster(model_3d, X), pch = 19)
points(model_2d$mu, col = 1:model_3d$K, pch = 8, cex = 2)
```




```r
plot3d(X, col = predict_cluster(model_3d, X), pch = 19)
plot_convergence(model_2d)
```



Our output looks pretty similar to what we initially had. Again, the convergence struggles a little but eventually stabilizes.

This wiggling is again reflected in the iterations taking a long while before clearly delineating between the classes.

```
to.plot = data.frame(x = data[,1],
                     y = data[,2],
                     z = data[,3],
                     iter = model_2d$Alpha[,1:8])

to.plot = melt(to.plot, id.vars = c("x", "y", "z"))

# model_2d$Alpha
ggplot(to.plot)+aes(x,y,z, color = value)+
  geom_point()+facet_wrap(~variable, nrow = 2)
```

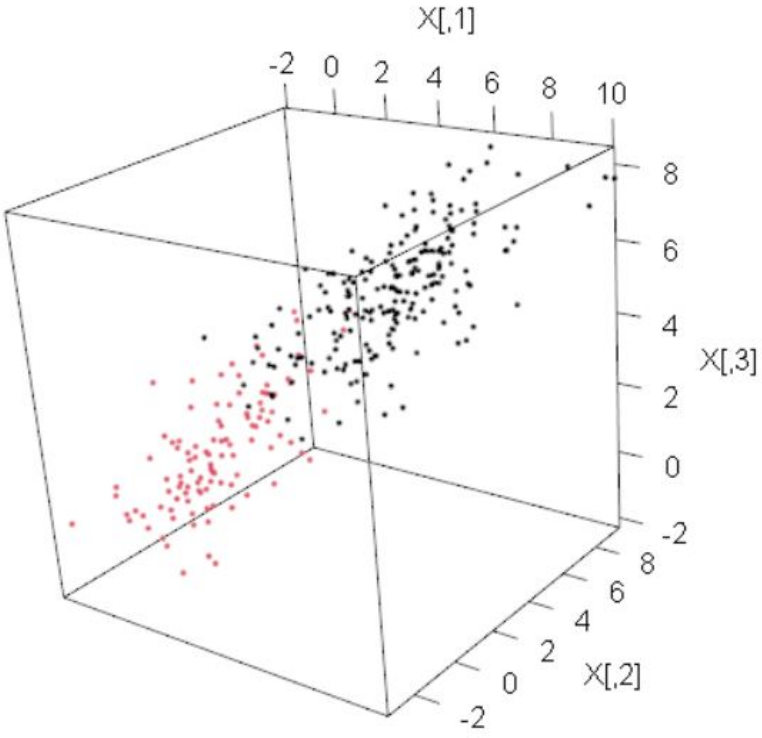
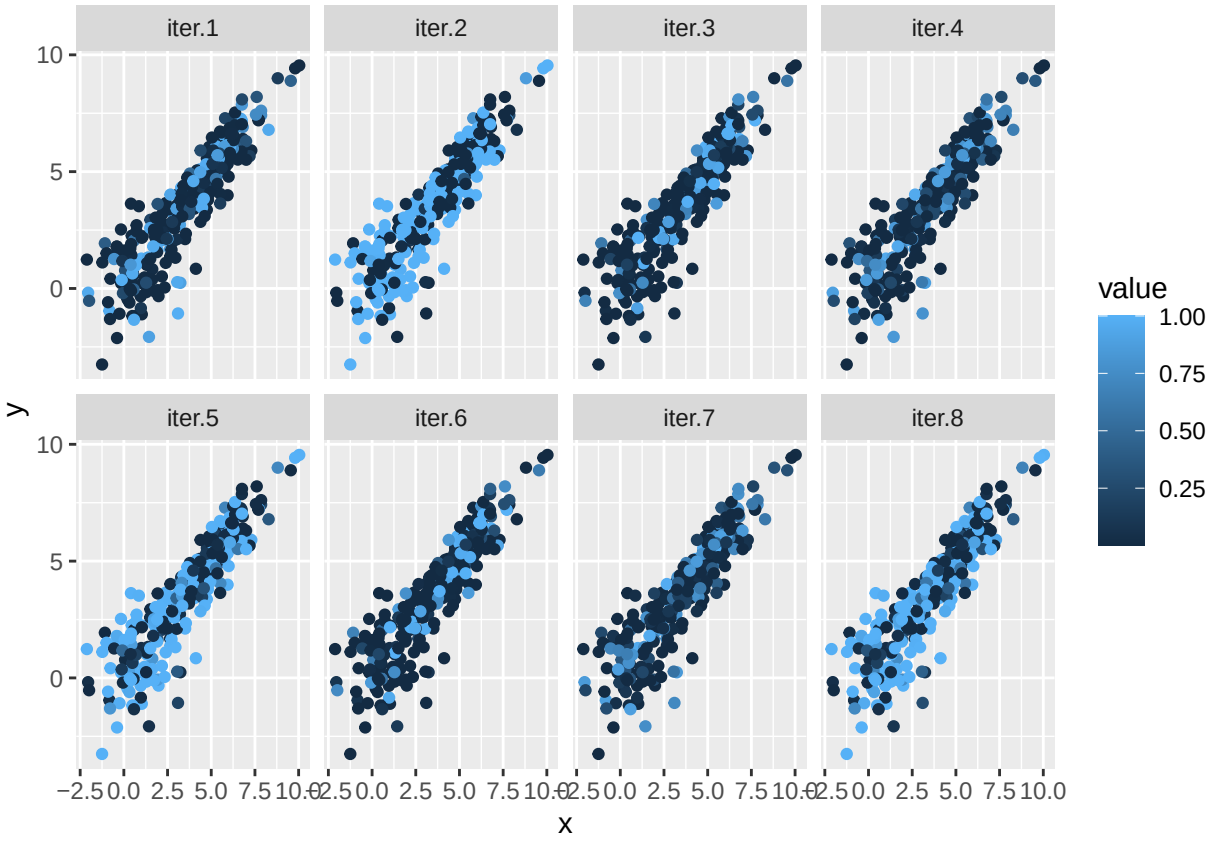


Figure 2: expected 3d output plot



Point 3

Once you know the algorithm works, apply it to segment three images where you think there are different classes and show the result.

Image 1

```
library(OpenImageR)
```

```
## Warning: package 'OpenImageR' was built under R version 4.4.2
```

```
img1 = readImage("1.jpg")
imageShow(img1)
```

```
img1.vect = apply(img1, 3, as.vector )
dim(img1.vect)
```

```
## [1] 129960      3
```

In this first case, we take the blues to try to distinguish the foreground and background.

```
X=as.matrix(img1.vect[,3])
length(X)
```

```
## [1] 129960
```

```
K=2
# model_image_1 <- em_gaussian_mixture(X, K, max_iter = 10)
model_image_1= readRDS("model1.rds") #we load it instead to save time
```

```
prob = model_image_1$gamma[,1]
classification = (prob<0.5)+0
segmented.image = replicate(3,classification, simplify = T)
dim(segmented.image) = dim(img1)
imageShow(segmented.image)
```

Image 2

```
img2 = readImage("2.jpg")
imageShow(img2)
```

```
img2.vect = apply(img2, 3, as.vector )
```

Here, we use the red and blue vectors to separate the back and foreground as well as lighter and darker regions.

```
X=as.matrix(img2.vect[,c(1,3)])

K=3
# model_image_2 <- em_gaussian_mixture(X, K, max_iter = 10)
model_image_2= readRDS("model2.rds") #loading presaved
```

```
prob_red = model_image_2$gamma[,1]
prob_blu = model_image_2$gamma[,2]

classification = (prob_red<0.5 | prob_blu<0.5)+0 #changes to 0 if neither is bigger than 1
classification[classification==1] = (prob_red[classification==1]>prob_blu[classification==1])+0#returns
red = classification

classification = (prob_red<0.5 | prob_blu<0.5)+0 #changes to 0 if neither is bigger than 1
classification[classification==1] = (prob_red[classification==1]<prob_blu[classification==1])#returns w
blu = classification

segmented.image = cbind(red,rep(0,length(blu)),blu)
dim(segmented.image) = dim(img2)
imageShow(segmented.image)
```

Image 3

```
img3 = readImage("3.jpg")
imageShow(img3)
```

```
img3.vect = apply(img3, 3, as.vector )
```

Lastly, we again use the blue vector to distinguish between the object and the background.

```
X=as.matrix(img3.vect[,3])

K=2
# model_image_3 <- em_gaussian_mixture(X, K, max_iter = 10)
model_image_3= readRDS("model3.rds")
```

```
prob = model_image_3$gamma[,1]
classification = (prob<0.5)+0
segmented.image = replicate(3,classification, simplify = T)
dim(segmented.image) = dim(img3)
imageShow(segmented.image)
```