Lab6

Cui Qingxuan, Nisal Amashan

2025-03-04

Contents

1 Collaborations

Nisal Amashan: Responsible for the question 1.

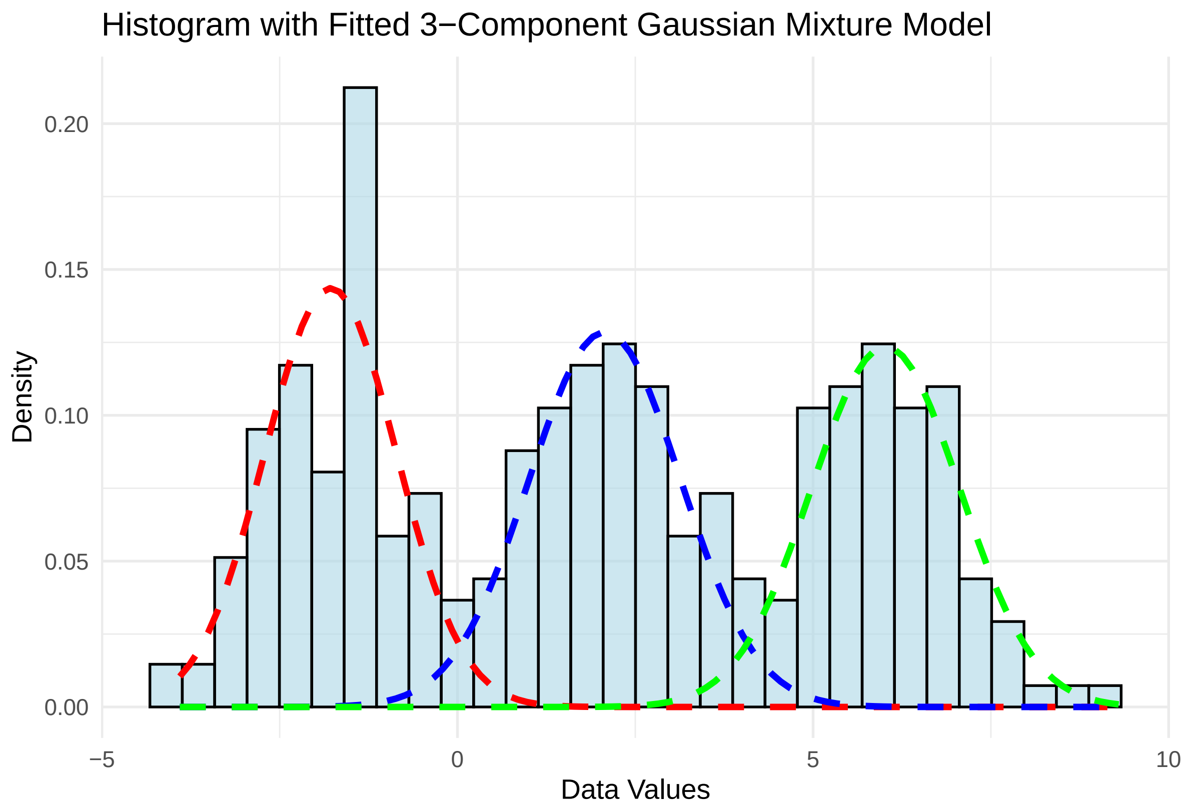
Cui Qingxuan: Responsible for the question 2.

2 Question 1

2.1 a) Extending the EM Algorithm for a Three-Component Normal Mixture Model

```r
library(ggplot2)

emalg_a <- function(dat, eps=1e-8){
  n      <- length(dat)
  gamma  <- matrix(NA, n, 3)


  p1 <- p2 <- p3 <- 1/3
  sigma1 <- sd(dat) * 2/3
  sigma2 <- sigma1
  sigma3 <- sigma1
  mu1 <- mean(dat) - sigma1
  mu2 <- mean(dat)
  mu3 <- mean(dat) + sigma1
  pv <- c(p1, p2, p3, mu1, mu2, mu3, sigma1, sigma2, sigma3)

  cc <- eps + 100  # Initialize convergence criterion
  while (cc > eps){
    pv1 <- pv  # Save previous parameter vector

    ### E step ###
    for (j in 1:n){
      pi1 <- p1 * dnorm(dat[j], mean=mu1, sd=sigma1)
      pi2 <- p2 * dnorm(dat[j], mean=mu2, sd=sigma2)
      pi3 <- p3 * dnorm(dat[j], mean=mu3, sd=sigma3)
      total <- pi1 + pi2 + pi3
      gamma[j, 1] <- pi1 / total
      gamma[j, 2] <- pi2 / total
      gamma[j, 3] <- pi3 / total
    }

    ### M step ###
    p1 <- mean(gamma[,1])
    p2 <- mean(gamma[,2])
    p3 <- mean(gamma[,3])

    mu1 <- sum(gamma[,1] * dat) / sum(gamma[,1])
    mu2 <- sum(gamma[,2] * dat) / sum(gamma[,2])
    mu3 <- sum(gamma[,3] * dat) / sum(gamma[,3])

    sigma1 <- sqrt(sum(gamma[,1] * (dat - mu1)^2) / sum(gamma[,1]))
    sigma2 <- sqrt(sum(gamma[,2] * (dat - mu2)^2) / sum(gamma[,2]))
    sigma3 <- sqrt(sum(gamma[,3] * (dat - mu3)^2) / sum(gamma[,3]))

    pv <- c(p1, p2, p3, mu1, mu2, mu3, sigma1, sigma2, sigma3)
    cc <- sum((pv - pv1)^2)  # Check convergence
  }

  return(pv)
}

set.seed(12345)
data <- c(rnorm(100, mean=-2, sd=0.8),
          rnorm(100, mean=2, sd=1),
          rnorm(100, mean=6, sd=1.2))

# Run the EM algorithm
result <- emalg_a(data)
p1_hat <- result[1]
p2_hat <- result[2]
p3_hat <- result[3]
mu1_hat <- result[4]
mu2_hat <- result[5]
mu3_hat <- result[6]
sigma1_hat <- result[7]
sigma2_hat <- result[8]
sigma3_hat <- result[9]


cat("Estimated Parameters:\n")
cat(sprintf("p1 = %.3f, p2 = %.3f, p3 = %.3f\n", p1_hat, p2_hat, p3_hat))
cat(sprintf("mu1 = %.3f, mu2 = %.3f, mu3 = %.3f\n", mu1_hat, mu2_hat, mu3_hat))
cat(sprintf("sigma1 = %.3f, sigma2 = %.3f, sigma3 = %.3f\n", sigma1_hat, sigma2_hat, sigma3_hat))


ggplot(data.frame(x=data), aes(x)) +
  geom_histogram(aes(y=..density..), bins=30, fill="lightblue", color="black", alpha=0.6) +

  # Add estimated normal distributions with proper scaling
  stat_function(fun = function(x) p1_hat * dnorm(x, mean = mu1_hat, sd = sigma1_hat),
                color = "red", lwd = 1.2, linetype="dashed") +
  stat_function(fun = function(x) p2_hat * dnorm(x, mean = mu2_hat, sd = sigma2_hat),
                color = "blue", lwd = 1.2, linetype="dashed") +
  stat_function(fun = function(x) p3_hat * dnorm(x, mean = mu3_hat, sd = sigma3_hat),
                color = "green", lwd = 1.2, linetype="dashed") +

  ggtitle("Histogram with Fitted 3-Component Gaussian Mixture Model") +
  xlab("Data Values") +
  ylab("Density") +
  theme_minimal()
```

```
## Estimated Parameters:

## p1 = 0.333, p2 = 0.343, p3 = 0.323

## mu1 = -1.783, mu2 = 2.072, mu3 = 6.019

## sigma1 = 0.926, sigma2 = 1.066, sigma3 = 1.043
```



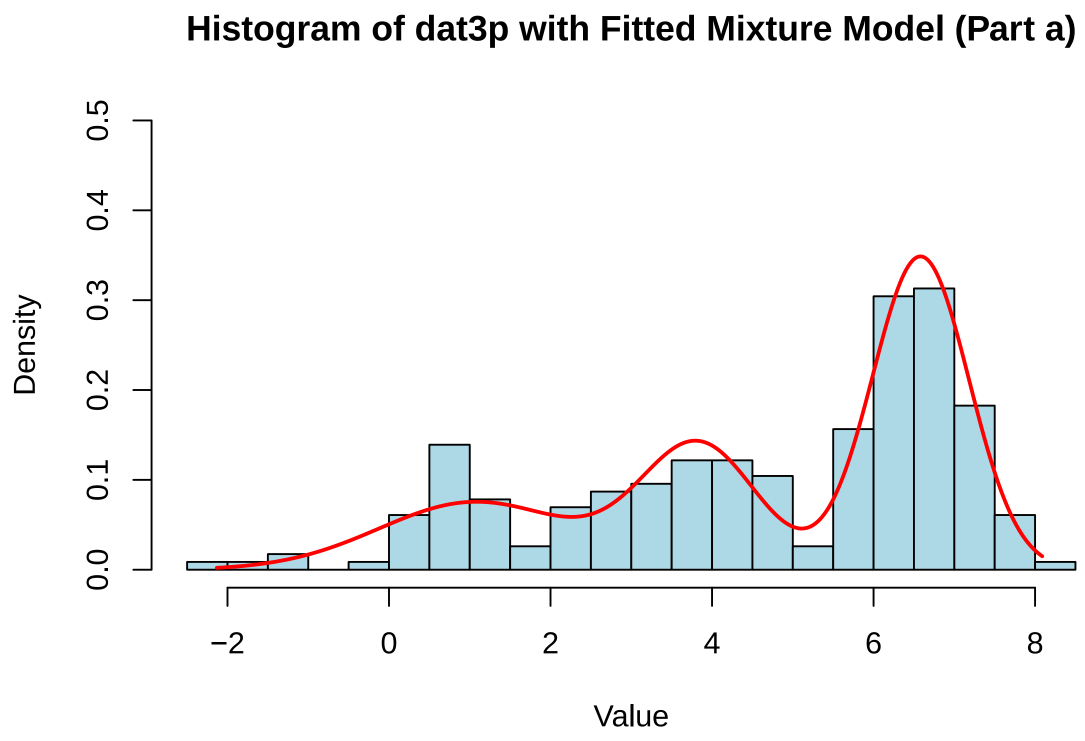
2.2 b) Modification of Stopping Criterion

To make the stopping criterion is invariant to data scaling, I replaced the **sum of squared differences** between parameter vectors with the **maximum relative change**. Specifically, I computed the relative difference for each parameter as `abs((pv - pv1) / pv1)` and used the maximum value as the convergence criterion

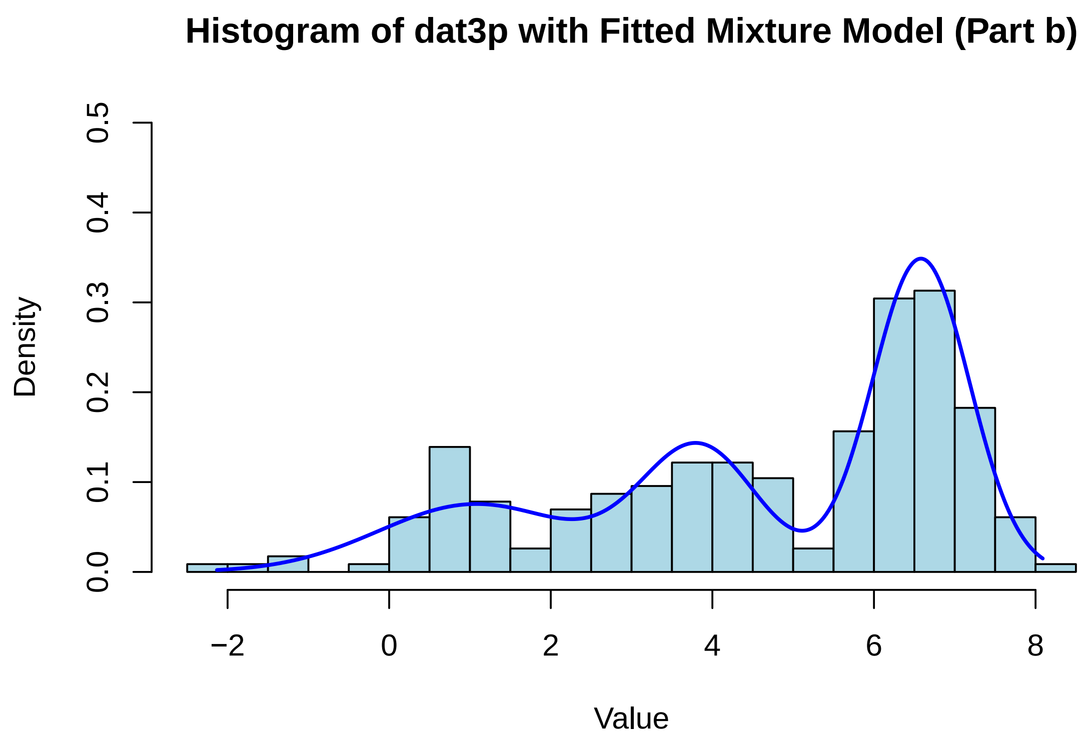
This makes the algorithm stops based on relative changes in parameters, making it independent of data scaling.

2.3 c) Fitted Mixture Models

2.3.1 Fitted Mixture Model (Part a)



2.3.2 Fitted Mixture Model (Part b)



For both versions (Part a and Part b), I obtained similar graphs. The histograms of the `dat3p` data were overlaid with the fitted density curves from the three-component normal mixture model. The fitted curves matched the data distribution well, showing that the EM algorithm worked effectively in estimating the parameters for both versions.

2.3.3 Estimated Model Parameters

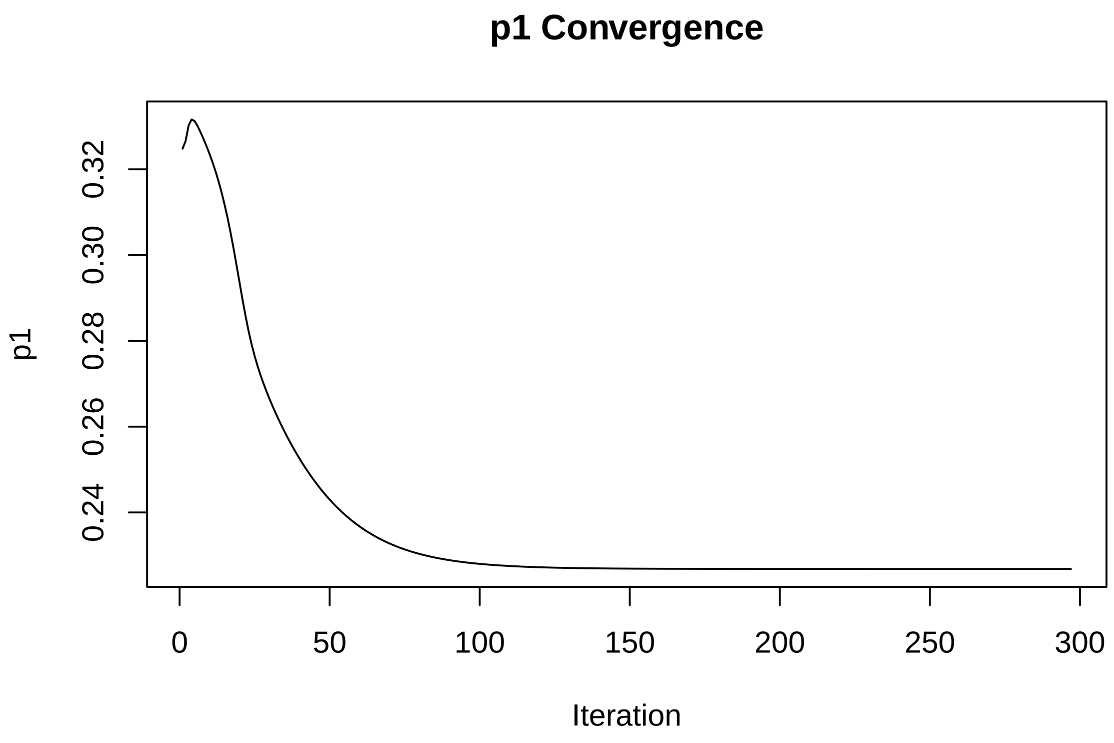
2.3.3.1 With part a stopping criteria :

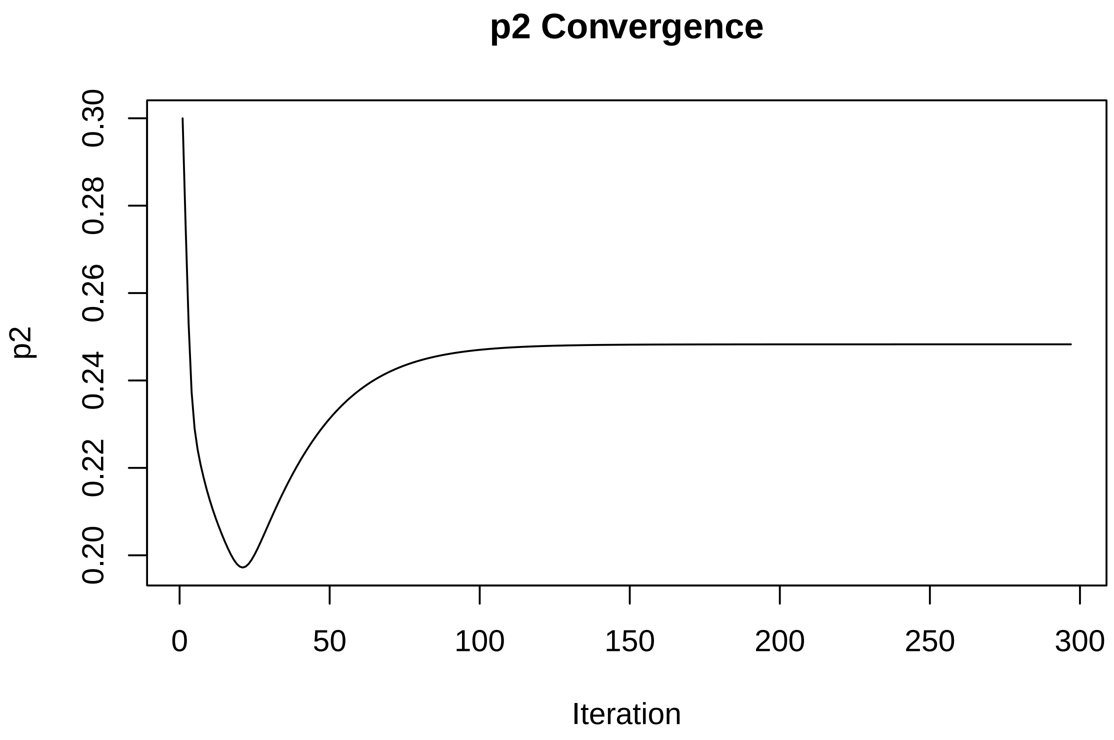
- **Mixing Proportions**:
  - `p1 = 0.227`
  - `p2 = 0.248`
  - `p3 = 0.525`
- **Means**:
  - `mu1 = 1.074`
  - `mu2 = 3.832`
  - `mu3 = 6.582`
- **Standard Deviations**:
  - `sigma1 = 1.199`
  - `sigma2 = 0.717`
  - `sigma3 = 0.601`

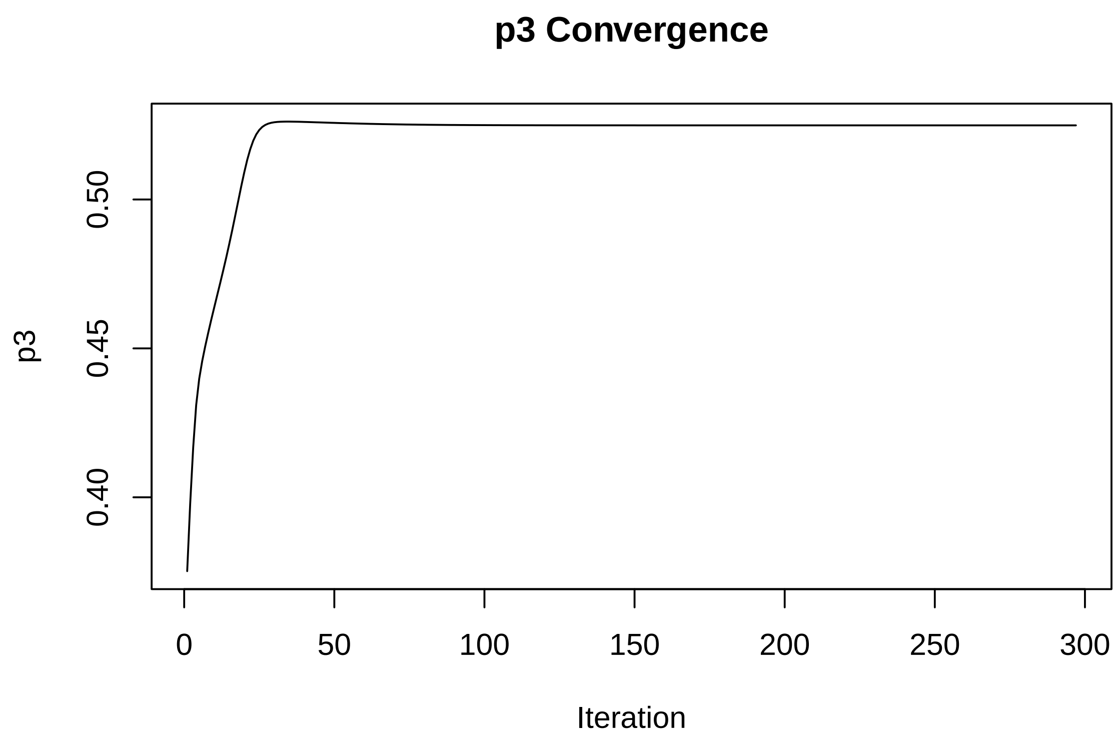
2.3.3.2 With part b stopping criteria :

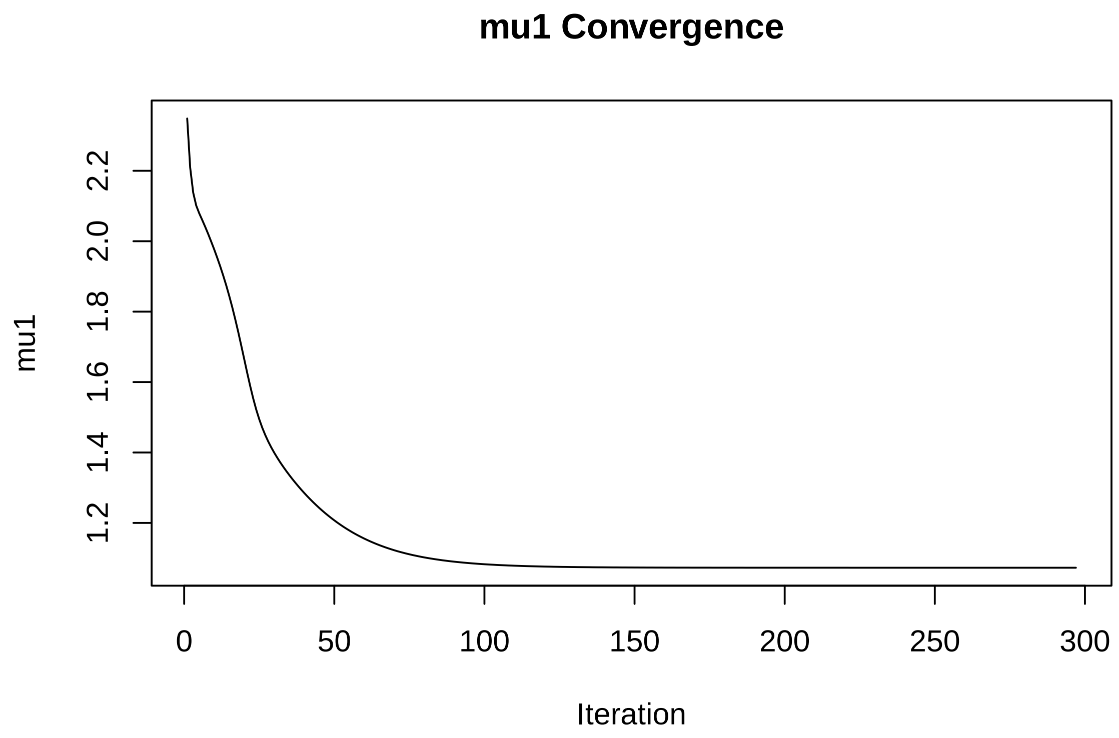
- **Mixing Proportions**:
  - `p1 = 0.227`
  - `p2 = 0.248`
  - `p3 = 0.525`
- **Means**:
  - `mu1 = 1.073`
  - `mu2 = 3.832`
  - `mu3 = 6.582`
- **Standard Deviations**:
  - `sigma1 = 1.198`
  - `sigma2 = 0.717`
  - `sigma3 = 0.601`

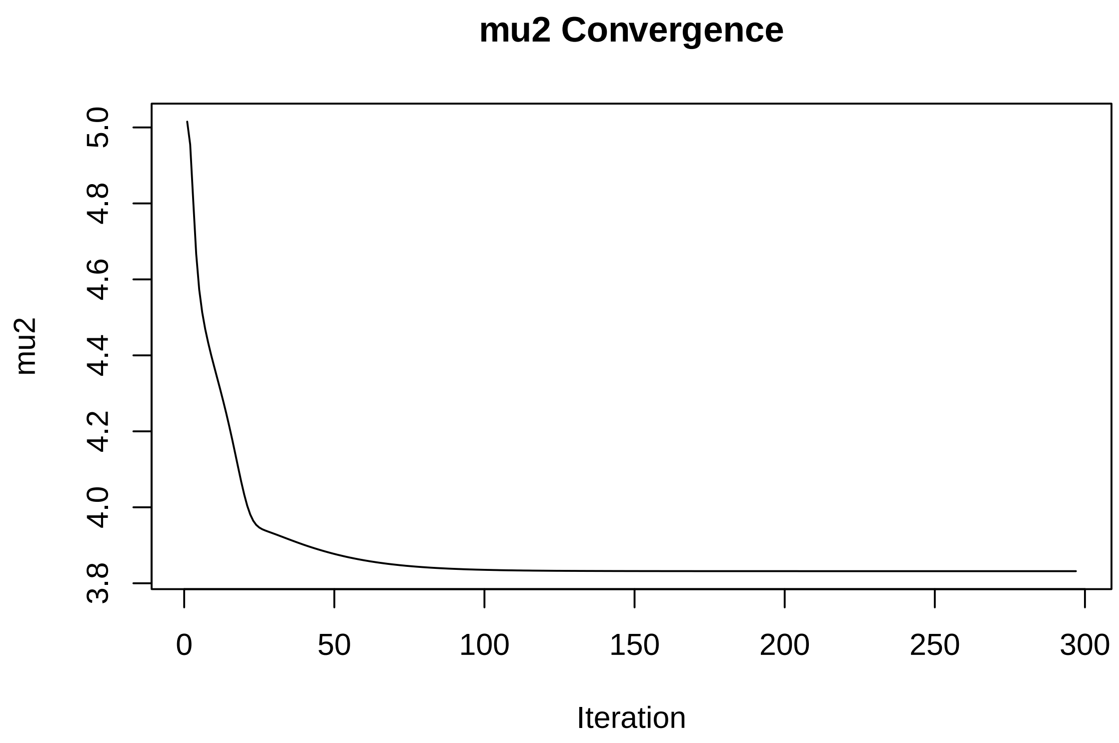
2.4 d) Convergence of Parameter Estimates Over Iterations

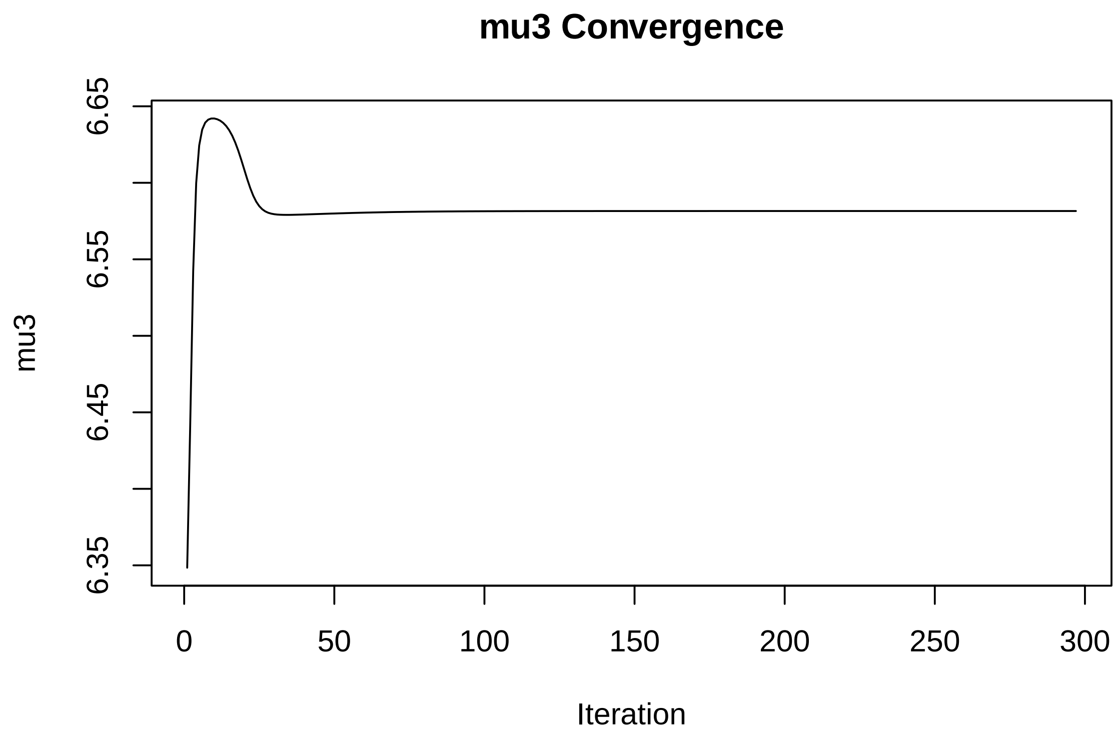


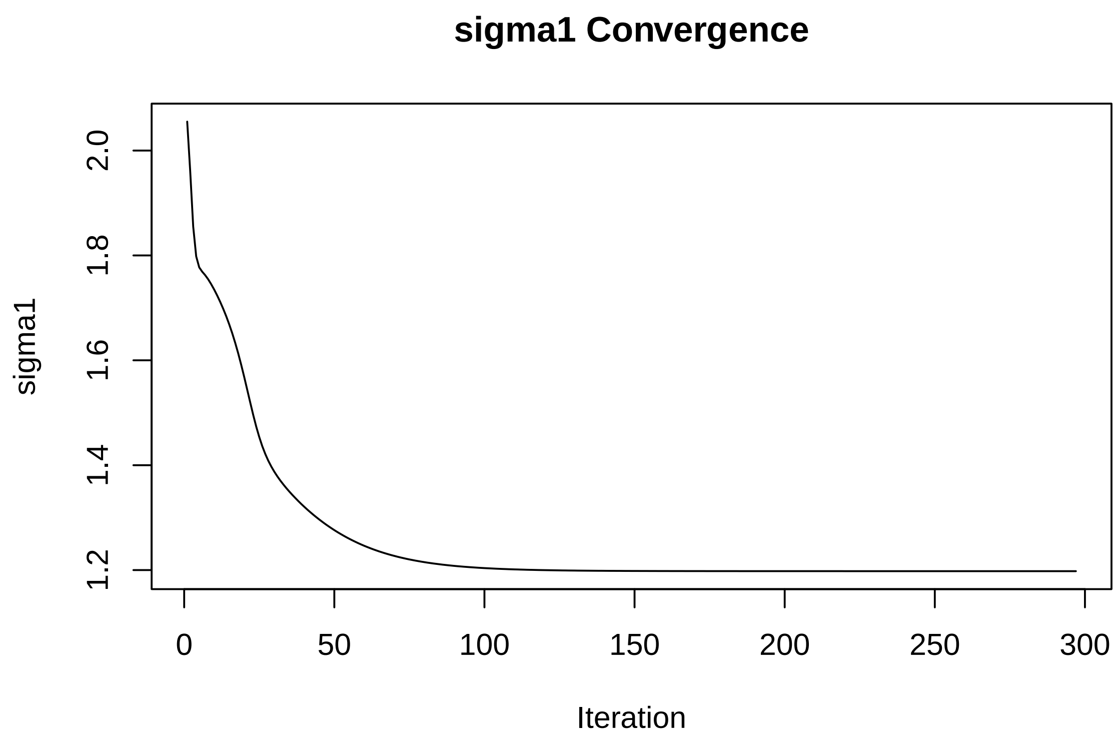


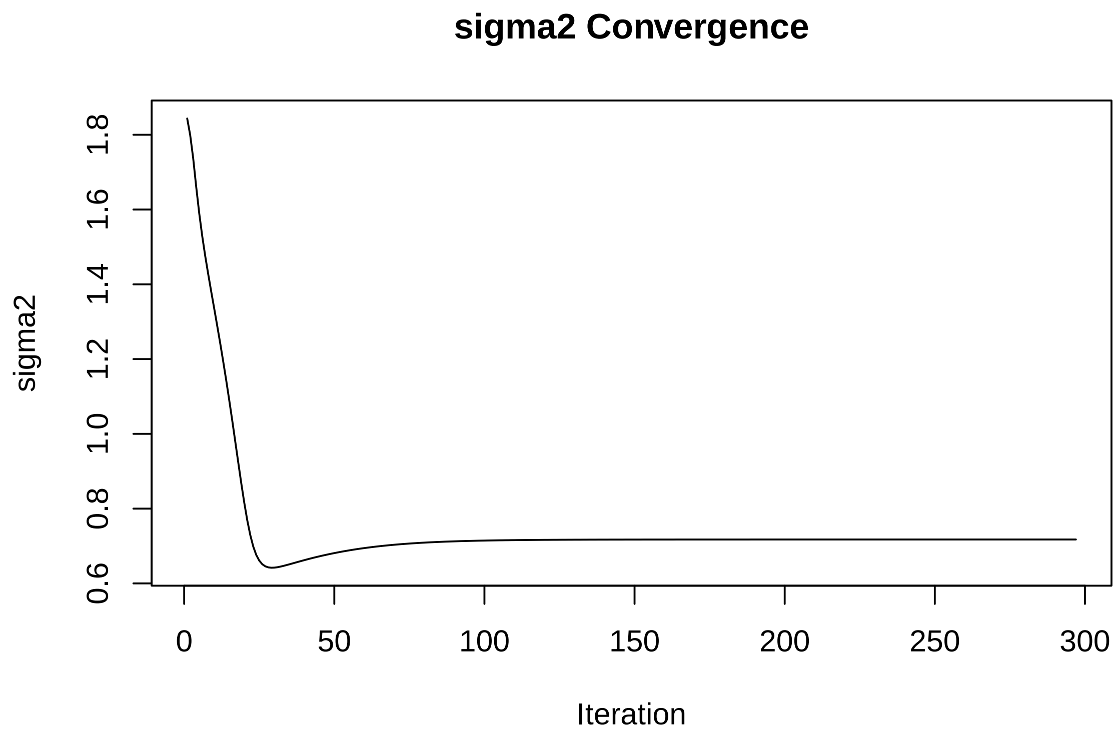

p3 Convergence
p3
0.40
0.45
0.50
0
50
100
150
200
250
300
Iteration

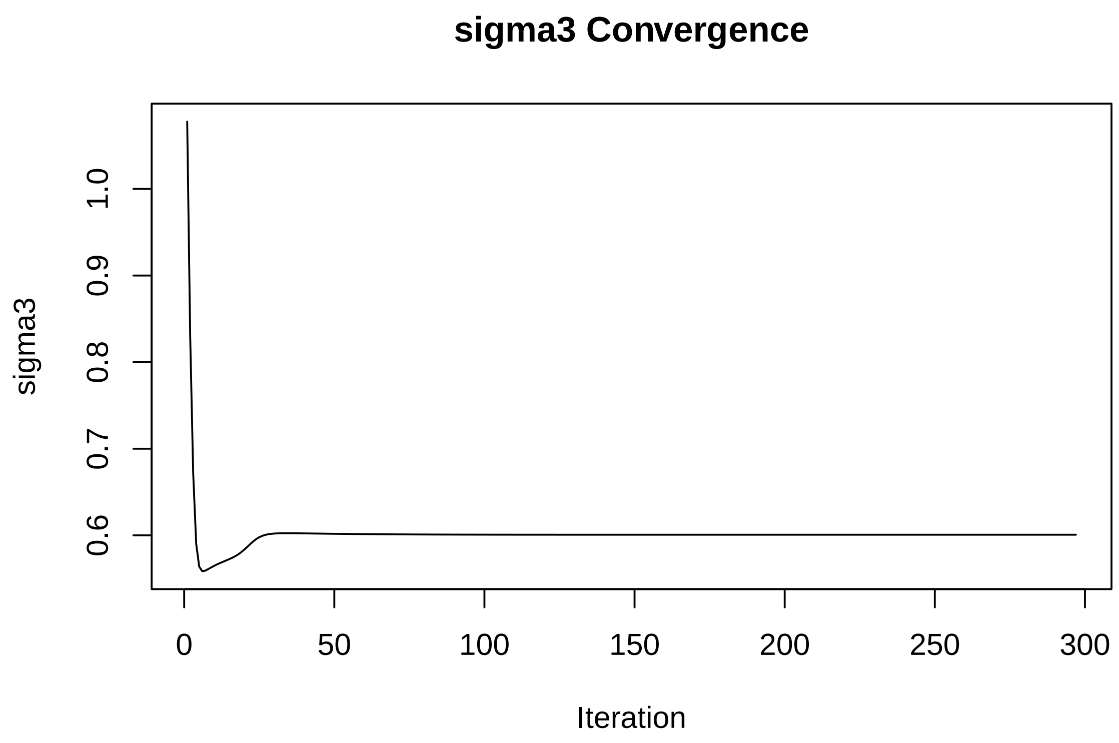

mu1 Convergence
mu1
1.2
1.4
1.6
1.8
2.0
2.2
0
50
100
150
200
250
300
Iteration


mu2 Convergence
mu2
5.0
4.8
4.6
4.4
4.2
4.0
3.8
0
50
100
150
200
250
300
Iteration


mu3 Convergence
mu3
6.65
6.55
6.45
6.35
0
50
100
150
200
250
300
Iteration


sigma1 Convergence
sigma1
2.0
1.8
1.6
1.4
1.2
0
50
100
150
200
250
300
Iteration


sigma2 Convergence
sigma2
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0
50
100
150
200
250
300
Iteration

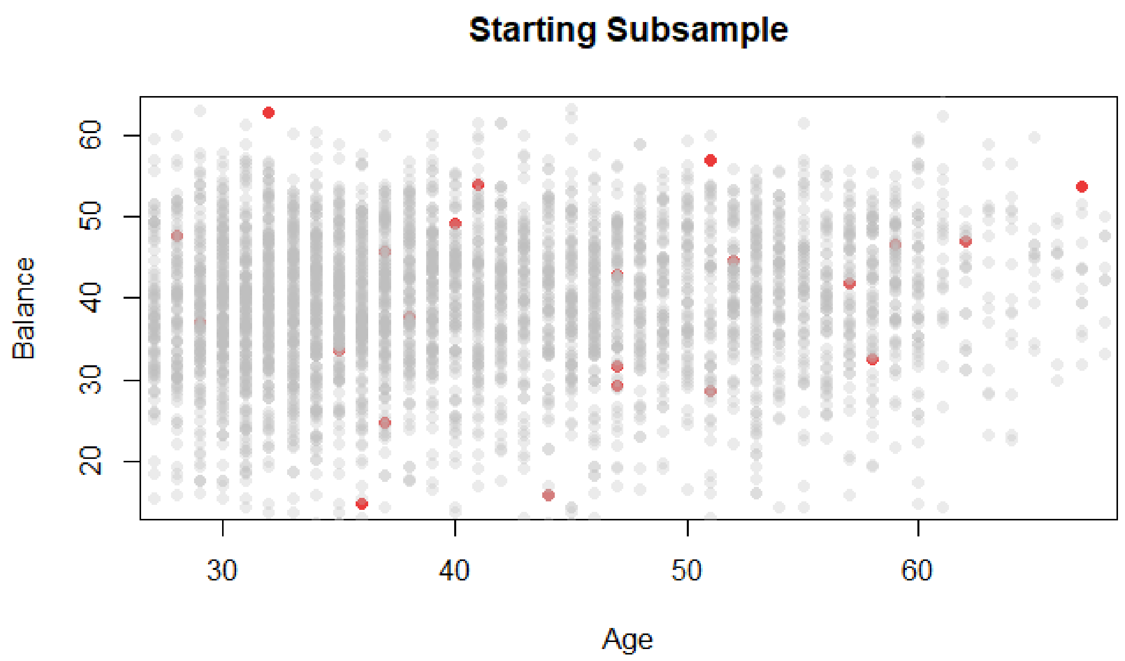


1. **Mixing Proportions**:
   - `p1` stabilizes around **0.23** after approximately 100 iterations.
   - `p2` stabilizes around **0.25** after approximately 100 iterations.
   - `p3` stabilizes around **0.53** after approximately 40 iterations.
2. **Means**:
   - `mu1` stabilizes around **1.1** after approximately 100 iterations.
   - `mu2` stabilizes around **3.85** after approximately 100 iterations.
   - `mu3` stabilizes around **6.555** after approximately 40 iterations.
3. **Standard Deviations**:
   - `sigma1` stabilizes around **1.2** after approximately 100 iterations.
   - `sigma2` stabilizes around **0.7** after approximately 100 iterations.
   - `sigma3` stabilizes around **0.6** after approximately 25 iterations.

3 Question 2

3.1 Plot the Random Selected Data

A total of 22 data points were selected based on a uniform distribution. In the plot below, the selected points are highlighted in red, while the remaining data points from the bank dataset are shown in grey with transparency to provide context.



3.2 Implement Annealing Algorithm

After 50 annealing stages, the algorithm achieved a simulated minimum value of 16,625.77.

The parameters used in the simulated annealing process are as follows:

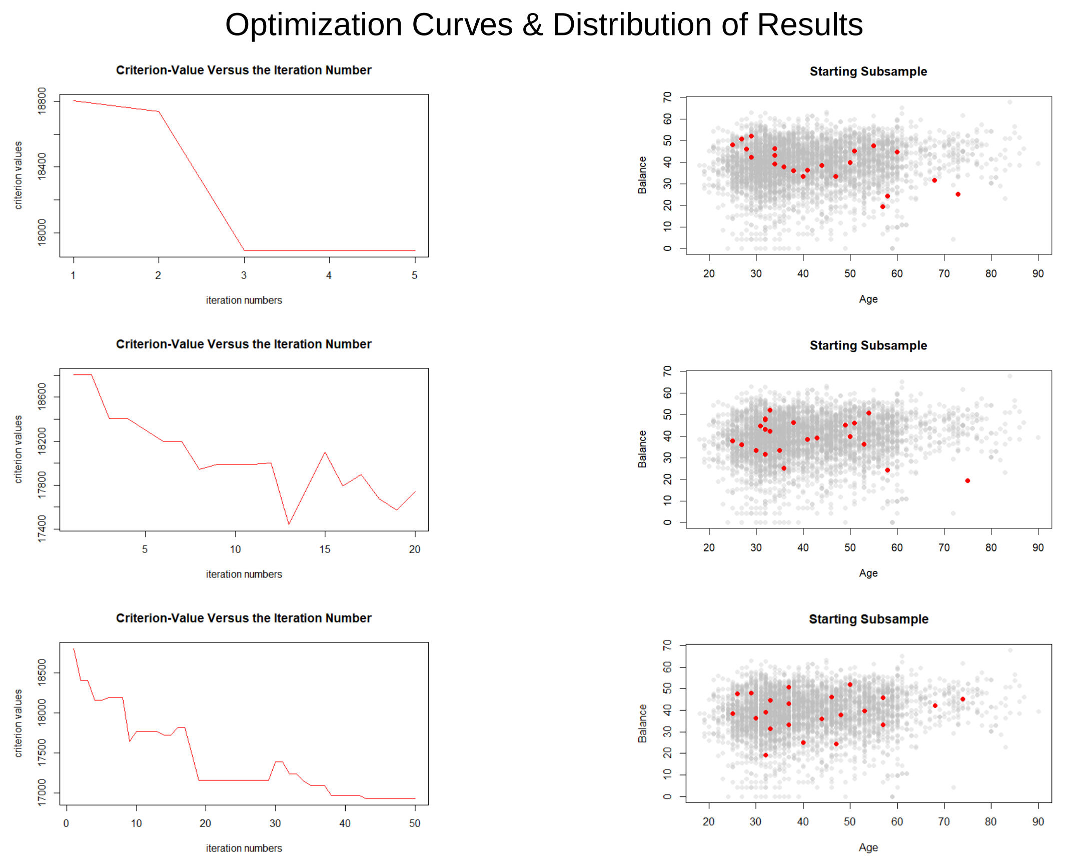
```
simulated_annealing(target_function=crit,
                    x_index=start_index,
                    temperature=80,
                    stages=50,
                    iteration_for_each_stage=10,
                    alpha=0.9,
                    beta=1.1,
                    data=bankdata,
                    annealing_method='linear')
```

3.3 Compare Different Combinations

To evaluate the impact of different parameter settings, we tested three different configurations, as summarized in the table below. Among these, result 3 yielded the best optimization performance.

	result1	result2	result3
iteration	5	20	50
starting_temperature	20	50	80
anneaing_method	exp	log	linear
optimal_criterion	17864.32	17536.20	16625.77

Additionally, we visualize the results using the plots below. The left column presents the optimization curves, which show the criterion value as a function of iteration number, while the right column displays the corresponding selected 22 points for each configuration.



4 Appendix

4.1 Question 1

```r
# Part a

library(ggplot2)


emalg_a <- function(dat, eps = 1e-8) {
  n <- length(dat)
  gamma <- matrix(NA, n, 3)

  # Initialize parameters
  p1 <- p2 <- p3 <- 1/3
  sigma1 <- sd(dat) * 2/3
  sigma2 <- sigma1
  sigma3 <- sigma1
  mu1 <- mean(dat) - sigma1
  mu2 <- mean(dat)
  mu3 <- mean(dat) + sigma1
  pv <- c(p1, p2, p3, mu1, mu2, mu3, sigma1, sigma2, sigma3)

  cc <- eps + 100  # Initialize convergence criterion
  while (cc > eps) {
    pv1 <- pv  # Save previous parameter vector

    ### E step ###
    for (j in 1:n) {
      pi1 <- p1 * dnorm(dat[j], mean = mu1, sd = sigma1)
      pi2 <- p2 * dnorm(dat[j], mean = mu2, sd = sigma2)
      pi3 <- p3 * dnorm(dat[j], mean = mu3, sd = sigma3)
      total <- pi1 + pi2 + pi3
      gamma[j, 1] <- pi1 / total
      gamma[j, 2] <- pi2 / total
      gamma[j, 3] <- pi3 / total
    }

    ### M step ###
    p1 <- mean(gamma[, 1])
    p2 <- mean(gamma[, 2])
    p3 <- mean(gamma[, 3])

    mu1 <- sum(gamma[, 1] * dat) / sum(gamma[, 1])
    mu2 <- sum(gamma[, 2] * dat) / sum(gamma[, 2])
    mu3 <- sum(gamma[, 3] * dat) / sum(gamma[, 3])

    sigma1 <- sqrt(sum(gamma[, 1] * (dat - mu1)^2) / sum(gamma[, 1]))
    sigma2 <- sqrt(sum(gamma[, 2] * (dat - mu2)^2) / sum(gamma[, 2]))
    sigma3 <- sqrt(sum(gamma[, 3] * (dat - mu3)^2) / sum(gamma[, 3]))

    pv <- c(p1, p2, p3, mu1, mu2, mu3, sigma1, sigma2, sigma3)
    cc <- sum((pv - pv1)^2)  # Check convergence
  }

  return(pv)
}

# Generate synthetic data
set.seed(12345)
data <- c(rnorm(100, mean = -2, sd = 0.8),
          rnorm(100, mean = 2, sd = 1),
          rnorm(100, mean = 6, sd = 1.2))

# Run the EM algorithm
result <- emalg_a(data)
p1_hat <- result[1]
p2_hat <- result[2]
p3_hat <- result[3]
mu1_hat <- result[4]
mu2_hat <- result[5]
mu3_hat <- result[6]
```

```r
sigma1_hat <- result[7]
sigma2_hat <- result[8]
sigma3_hat <- result[9]

# Print estimated parameters
cat("Estimated Parameters:\n")
cat(sprintf("p1 = %.3f, p2 = %.3f, p3 = %.3f\n", p1_hat, p2_hat, p3_hat))
cat(sprintf("mu1 = %.3f, mu2 = %.3f, mu3 = %.3f\n", mu1_hat, mu2_hat, mu3_hat))
cat(sprintf("sigma1 = %.3f, sigma2 = %.3f, sigma3 = %.3f\n", sigma1_hat, sigma2_hat, sigma3_hat))

# Plot histogram with fitted mixture model
ggplot(data.frame(x = data), aes(x)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "lightblue", color = "black", alpha = 0.6) +
  stat_function(fun = function(x) p1_hat * dnorm(x, mean = mu1_hat, sd = sigma1_hat),
                color = "red", lwd = 1.2, linetype = "dashed") +
  stat_function(fun = function(x) p2_hat * dnorm(x, mean = mu2_hat, sd = sigma2_hat),
                color = "blue", lwd = 1.2, linetype = "dashed") +
  stat_function(fun = function(x) p3_hat * dnorm(x, mean = mu3_hat, sd = sigma3_hat),
                color = "green", lwd = 1.2, linetype = "dashed") +
  ggtitle("Histogram with Fitted 3-Component Gaussian Mixture Model") +
  xlab("Data Values") +
  ylab("Density") +
  theme_minimal()

# part b

# EM Algorithm with Modified Stopping Criterion
emalg_b <- function(dat, eps = 1e-8) {
  n <- length(dat)
  gamma <- matrix(NA, n, 3)

  # Initialize parameters
  p1 <- p2 <- p3 <- 1/3
  sigma1 <- sd(dat) * 2/3
  sigma2 <- sigma1
  sigma3 <- sigma1
  mu1 <- mean(dat) - sigma1
  mu2 <- mean(dat)
  mu3 <- mean(dat) + sigma1
  pv <- c(p1, p2, p3, mu1, mu2, mu3, sigma1, sigma2, sigma3)

  cc <- eps + 100  # Initialize convergence criterion
  while (cc > eps) {
    pv1 <- pv  # Save previous parameter vector

    ### E step ###
    for (j in 1:n) {
      pi1 <- p1 * dnorm(dat[j], mean = mu1, sd = sigma1)
      pi2 <- p2 * dnorm(dat[j], mean = mu2, sd = sigma2)
      pi3 <- p3 * dnorm(dat[j], mean = mu3, sd = sigma3)
      total <- pi1 + pi2 + pi3
      gamma[j, 1] <- pi1 / total
      gamma[j, 2] <- pi2 / total
      gamma[j, 3] <- pi3 / total
    }

    ### M step ###
    p1 <- mean(gamma[, 1])
    p2 <- mean(gamma[, 2])
    p3 <- mean(gamma[, 3])

    mu1 <- sum(gamma[, 1] * dat) / sum(gamma[, 1])
    mu2 <- sum(gamma[, 2] * dat) / sum(gamma[, 2])
    mu3 <- sum(gamma[, 3] * dat) / sum(gamma[, 3])

    sigma1 <- sqrt(sum(gamma[, 1] * (dat - mu1)^2) / sum(gamma[, 1]))
    sigma2 <- sqrt(sum(gamma[, 2] * (dat - mu2)^2) / sum(gamma[, 2]))
    sigma3 <- sqrt(sum(gamma[, 3] * (dat - mu3)^2) / sum(gamma[, 3]))

    pv <- c(p1, p2, p3, mu1, mu2, mu3, sigma1, sigma2, sigma3)
    cc <- max(abs((pv - pv1) / pv1))  # Relative change
  }

  return(pv)
}


# Part c

# Load data
load("threepops.Rdata")
data <- dat3p

# Run the EM algorithm (Part a)
estimates_a <- emalg_a(data)

# Run the EM algorithm (Part b)
estimates_b <- emalg_b(data)

# Plot histogram with fitted mixture model (Part a)
hist(data, breaks = 30, col = "lightblue", probability = TRUE,
     main = "Histogram of dat3p with Fitted Mixture Model (Part a)",
     xlab = "Value", ylim = c(0, 0.5))

# Define the range for the fitted density curve
x_range <- seq(min(data), max(data), length.out = 1000)

# Compute the fitted density of the mixture model for part a
fitted_density_a <- estimates_a[1] * dnorm(x_range, mean = estimates_a[4], sd = estimates_a[7]) +
  estimates_a[2] * dnorm(x_range, mean = estimates_a[5], sd = estimates_a[8]) +
  estimates_a[3] * dnorm(x_range, mean = estimates_a[6], sd = estimates_a[9])

# Overlay the fitted density curve for part a
lines(x_range, fitted_density_a, col = "red", lwd = 2)

# Plot histogram with fitted mixture model (Part b)
```

```r
hist(data, breaks = 30, col = "lightblue", probability = TRUE,
     main = "Histogram of dat3p with Fitted Mixture Model (Part b)",
     xlab = "Value", ylim = c(0, 0.5))

# Compute the fitted density of the mixture model for part b
fitted_density_b <- estimates_b[1] * dnorm(x_range, mean = estimates_b[4], sd = estimates_b[7]) +
  estimates_b[2] * dnorm(x_range, mean = estimates_b[5], sd = estimates_b[8]) +
  estimates_b[3] * dnorm(x_range, mean = estimates_b[6], sd = estimates_b[9])

# Overlay the fitted density curve for part b
lines(x_range, fitted_density_b, col = "blue", lwd = 2)


# Part d

# EM Algorithm with Parameter Tracking
emalg_track <- function(dat, eps = 1e-8) {
  n <- length(dat)
  gamma <- matrix(NA, n, 3)

  # Initialize parameters
  p1 <- p2 <- p3 <- 1/3
  sigma1 <- sigma2 <- sigma3 <- sd(dat) * 2/3
  mu1 <- mean(dat) - sigma1
  mu2 <- mean(dat)
  mu3 <- mean(dat) + sigma1
  pv <- c(p1, p2, p3, mu1, mu2, mu3, sigma1, sigma2, sigma3)

  # Initialize storage for parameter estimates at each iteration
  param_history <- list(
    p1 = numeric(),
    p2 = numeric(),
    p3 = numeric(),
    mu1 = numeric(),
    mu2 = numeric(),
    mu3 = numeric(),
    sigma1 = numeric(),
    sigma2 = numeric(),
    sigma3 = numeric()
  )

  cc <- eps + 100  # Initialize convergence criterion
  iter <- 0  # Initialize iteration counter

  while (cc > eps) {
    iter <- iter + 1
    pv1 <- pv

    ### E step ###
    for (j in 1:n) {
      pi1 <- p1 * dnorm(dat[j], mean = mu1, sd = sigma1)
      pi2 <- p2 * dnorm(dat[j], mean = mu2, sd = sigma2)
      pi3 <- p3 * dnorm(dat[j], mean = mu3, sd = sigma3)
      total <- pi1 + pi2 + pi3
      gamma[j, 1] <- pi1 / total
      gamma[j, 2] <- pi2 / total
      gamma[j, 3] <- pi3 / total
    }

    ### M step ###
    p1 <- mean(gamma[, 1])
    p2 <- mean(gamma[, 2])
    p3 <- mean(gamma[, 3])

    mu1 <- sum(gamma[, 1] * dat) / sum(gamma[, 1])
    mu2 <- sum(gamma[, 2] * dat) / sum(gamma[, 2])
    mu3 <- sum(gamma[, 3] * dat) / sum(gamma[, 3])

    sigma1 <- sqrt(sum(gamma[, 1] * (dat - mu1)^2) / sum(gamma[, 1]))
    sigma2 <- sqrt(sum(gamma[, 2] * (dat - mu2)^2) / sum(gamma[, 2]))
    sigma3 <- sqrt(sum(gamma[, 3] * (dat - mu3)^2) / sum(gamma[, 3]))

    pv <- c(p1, p2, p3, mu1, mu2, mu3, sigma1, sigma2, sigma3)

    # Store parameter estimates for this iteration
    param_history$p1[iter] <- p1
    param_history$p2[iter] <- p2
    param_history$p3[iter] <- p3
    param_history$mu1[iter] <- mu1
    param_history$mu2[iter] <- mu2
    param_history$mu3[iter] <- mu3
    param_history$sigma1[iter] <- sigma1
    param_history$sigma2[iter] <- sigma2
    param_history$sigma3[iter] <- sigma3

    cc <- max(abs((pv - pv1) / pv1))
  }

  return(param_history)
}

# Run the EM algorithm with parameter tracking
param_history <- emalg_track(data)

# Plot convergence of parameters (one after the other)
plot(param_history$p1, type = "l", xlab = "Iteration", ylab = "p1",
     main = "p1 Convergence", ylim = range(param_history$p1), xlim = c(1, length(param_history$p1)))

plot(param_history$p2, type = "l", xlab = "Iteration", ylab = "p2",
     main = "p2 Convergence", ylim = range(param_history$p2), xlim = c(1, length(param_history$p2)))

plot(param_history$p3, type = "l", xlab = "Iteration", ylab = "p3",
     main = "p3 Convergence", ylim = range(param_history$p3), xlim = c(1, length(param_history$p3)))

plot(param_history$mu1, type = "l", xlab = "Iteration", ylab = "mu1",
     main = "mu1 Convergence", ylim = range(param_history$mu1), xlim = c(1, length(param_history$mu1)))
```

```r
plot(param_history$mu2, type = "l", xlab = "Iteration", ylab = "mu2",
     main = "mu2 Convergence", ylim = range(param_history$mu2), xlim = c(1, length(param_history$mu2)))

plot(param_history$mu3, type = "l", xlab = "Iteration", ylab = "mu3",
     main = "mu3 Convergence", ylim = range(param_history$mu3), xlim = c(1, length(param_history$mu3)))

plot(param_history$sigma1, type = "l", xlab = "Iteration", ylab = "sigma1",
     main = "sigma1 Convergence", ylim = range(param_history$sigma1), xlim = c(1, length(param_history$

plot(param_history$sigma2, type = "l", xlab = "Iteration", ylab = "sigma2",
     main = "sigma2 Convergence", ylim = range(param_history$sigma2), xlim = c(1, length(param_history$

plot(param_history$sigma3, type = "l", xlab = "Iteration", ylab = "sigma3",
     main = "sigma3 Convergence", ylim = range(param_history$sigma3), xlim = c(1, length(param_history$
```

4.2 Question 2

```r
##############################################################
### Computational statistics, Linköping University, VT2025 ###
### Criterion function Lab 6, Q2 (space filling design)    ###
### Frank Miller                                           ###
###                                                        ###
### We are using in this lab partial data from the         ###
### original bankdata available at                         ###
### https://archive.ics.uci.edu/ml/datasets/Bank+Marketing ###
### See also the publication:                              ###
### Sérgio Moro, P. Cortez, P. Rita (2014). A data-driven  ###
### approach to predict the success of bank telemarketing. ###
### Decision Support Systems.                              ###
##############################################################

# you need to save the following dataset at the right place and/or add/set the path where it is located
load("bankdata.Rdata")

nclients <- dim(bankdata)[1]  # number of individuals in the dataset, here 4364

# criterion function: sum of minimal distance to an element in the subset
# dat is the full dataset (here: bankdata), subs is the set of ids for the subset selected
# subs should be a vector of elements in 1, ..., 4364; for this question, it should be of length 22
# example call: crit(bankdata, 1:22), selecting the first 22 individuals
# result of this function is the criterion to be minimized
crit <- function(dat, subs){
  s <- length(subs)
  dist <- matrix(rep(0, nclients*s), ncol=s)
  for (i in 1:s){
    dist[, i] <- sqrt((dat[,1]-dat[subs[i],1])^2+(dat[,2]-dat[subs[i],2])^2)
  }
  sum(apply(dist, 1, min))
}

# it is good to identify the individuals in the full set by their id (1, ..., 4364),
# then we can sample from this set for the starting subset:
fullset <- 1:nclients

# Plot the selected 22 points
set.seed(12345)
start_index = floor(runif(22, min=1, max=length(bankdata[,1])))
start_sub = bankdata[start_index, ]
plot(start_sub[, 1], start_sub[, 2], type="p", col="red", xlab="Age", ylab="Balance", main="Starting Su
points(bankdata[, 1], bankdata[, 2], col=adjustcolor("grey", alpha.f=0.3), pch=16)


# Simulated Annealing Algorithm
annealing = function(target, x_index, tem, iter, m, alpha, beta, data, annealing='exp'){
  # iter: j, stage numbers
  # t: starting temperature
  # target: target function, here is crit
  # proposal: proposal distribution, to generate the x_star
  # x_index: index of x0
  # m: number of iterations per stages
  # bankdata
  # alpha: the factor to update temperature
  # beta: the factore to update m
  # data: bankdata
  x_cri = c()
  for(j in c(1:iter)){
    x_cri = append(x_cri, crit(bankdata, x_index))
    for(t in c(1:m)){
      # sample a candidate xt from proposal distribution
      # we may need to generate 22 data points
      # shuffle
      candidate_index = sample(x_index)
      # exchange the first 5 index
      candidate_index[1:10] = sample(1:nrow(data), 10)

      # compute criteria function
      h = exp((crit(bankdata, x_index) - crit(bankdata, candidate_index)) / tem)

      # update xt+1
      p = min(h, 1)
      # x_index = ifelse(runif(1) < p, candidate_index, x_index)
      if(runif(1) < p) x_index = candidate_index
      # set t<-t+1 and next round to t==m
    }
  # update temperature t and m
    if(annealing == 'exp'){
      tem = alpha * tem
    }
    else if(annealing == 'log'){
      tem = tem / (1+log(alpha))
    }
    else if(annealing=='linear'){
      tem = tem - alpha
    }
    m = beta * m
  }
  return(list(x_cri, x_index))
}


result1=annealing(target=crit,
                  x_index=start_index,
                  tem=20,
                  iter=5,
                  m=10,
                  alpha=0.9,
                  beta=1.1,
                  data=bankdata,
                  annealing='exp')

result2=annealing(target=crit,
                  x_index=start_index,
                  tem=50,
                  iter=20,
                  m=10,
                  alpha=0.9,
                  beta=1.1,
                  data=bankdata,
                  annealing='log')

result3=annealing(target=crit,
                  x_index=start_index,
                  tem=80,
                  iter=50,
                  m=10,
                  alpha=0.9,
                  beta=1.1,
                  data=bankdata,
                  annealing='linear')


plot(c(1:5), result1[[1]], type='l', col='red', xlab='iteration numbers', ylab='criterion values', main=
plot(bankdata[, 1], bankdata[, 2], type="p", col=adjustcolor("grey", alpha.f=0.3), xlab="Age", ylab="Bal
points(bankdata[result2[[2]], 1], bankdata[result3[[2]], 2], col="red", pch=16)

plot(bankdata[, 1], bankdata[, 2], type="p", col=adjustcolor("grey", alpha.f=0.3), xlab="Age", ylab="Bal
points(bankdata[result1[[2]], 1], bankdata[result3[[2]], 2], col="red", pch=16)
plot(c(1:20), result2[[1]], type='l', col='red', xlab='iteration numbers', ylab='criterion values', mai

plot(bankdata[, 1], bankdata[, 2], type="p", col=adjustcolor("grey", alpha.f=0.3), xlab="Age", ylab="Bal
points(bankdata[result3[[2]], 1], bankdata[result3[[2]], 2], col="red", pch=16)
plot(c(1:50), result3[[1]], type='l', col='red', xlab='iteration numbers', ylab='criterion values', mai
```