Math for Data Science: Problem Set 3

Name: Ray Hossain, Group: Finn Krueger, Milton Mier

2023-10-11

Due Date: Monday, November 20 by the end of the day. (The Moodle submission link will become inactive at midnight of November 21.)

Instructions: Please submit one solution set per person and include your name and your group members' names at the top. This time, please write your solutions within this Rmd file, under the relevant question. Please submit the knitted output as a pdf. Make sure to show all code you used to arrive at the answer. However, please provide a brief, clear answer to every question rather than making us infer it from your output, and please avoid printing unnecessary output.

1. Revisiting Old Faithful

(20 points) Let's continue with our Old Faithful example from Lab 7. Remember that we had two pieces of information about the geyser: eruption duration, which we worked with, and also the wait time until next eruption.

a. (2 points) Load the data. Plot a histogram with a density curve for time until next eruption, as we did for eruption duration in the lab.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

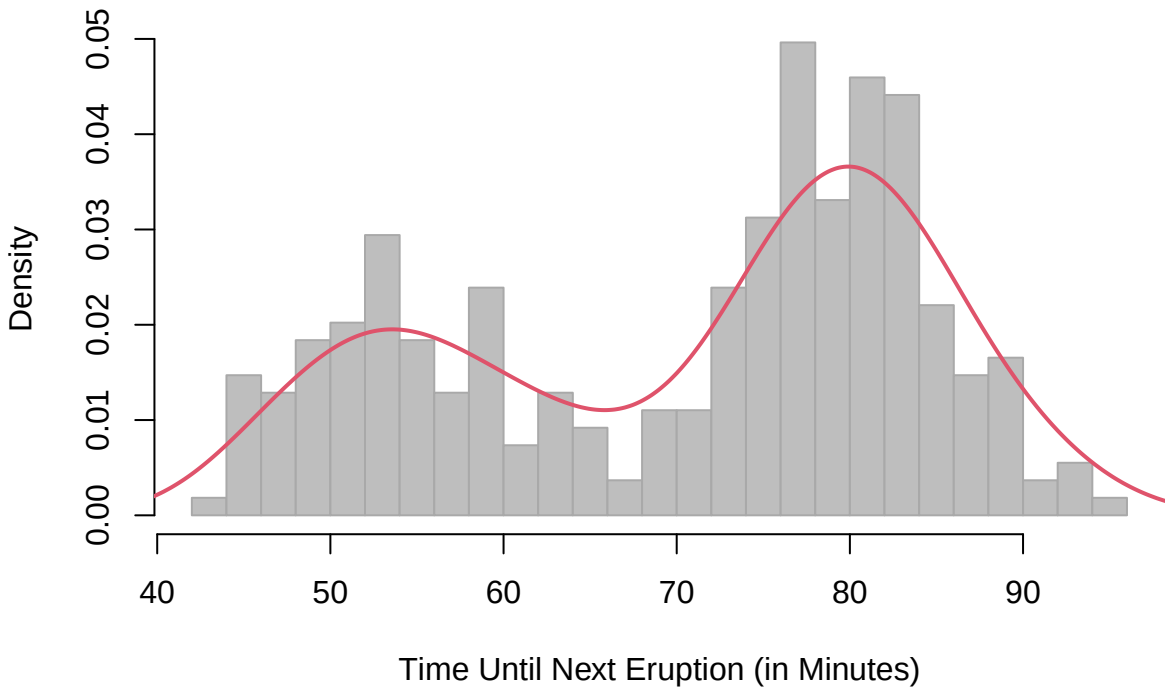
```
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
data(faithful)
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

```
waiting  = as.matrix(faithful[, 2, drop = FALSE])
h <- hist(waiting, col = "grey", bor = "darkgrey", breaks = 24, prob = TRUE,
          xlab = "Time Until Next Eruption (in Minutes)", main="")
lines(density(waiting), col = 2, lwd = 2)
```



b. (6 points) Adapt the EM algorithm code from class to do the following:

- (2 points) In every iteration, please save your log likelihood at the end of the update. You should end up with a vector of log likelihoods that is as long as the number of iterations your algorithm ran. Make this output available to the user, just like the estimated parameters.
- (2 points) Similarly, please save one or two of your updated parameters in every iteration. It doesn't matter which one – you can choose.

- (2 points) Also save the gammas into a data frame. We only need the gammas from the final iteration; no need to save each one.
- Run your EM algorithm for wait times to next eruption. Don't print all the output – just enough to demonstrate that your algorithm successfully accomplishes all of the above.

```
#--- WRITE A FUNCTION TO GET LOG LIKELIHOOD WHICH WILL BE PLACED INSIDE THE EM FUNCTION
log_lik <- function(Y, mu_1, mu_2, var_1, var_2, pi_1, pi_2) {
  sum(log(pi_1 * dnorm(Y, mu_1, sd = sqrt(var_1)) + pi_2 * dnorm(Y, mu_2, sd = sqrt(var_2))))
}


## Define the optimization function

em_mixture <- function(starting_values, Y, tol = .0001, maxits = 100) {
  converged <- FALSE
  iter <- 0
  N <- length(Y)
  df <- tibble()

  # initialize starting values
  pi_1 <- starting_values$pi_1
  pi_2 <- starting_values$pi_2
  mu_1 <- starting_values$mu_1
  mu_2 <- starting_values$mu_2
  var_1 <- starting_values$var_1
  var_2 <- starting_values$var_2

  # Loop until convergence
  while ((!converged) & (iter < maxits)) {
    # Evaluate the log likelihood at the initial parameters
    ll <- log_lik(Y = Y,
                  pi_1 = pi_1,
                  pi_2 = pi_2,
                  mu_1 = mu_1,
                  mu_2 = mu_2,
                  var_1 = var_1,
                  var_2 = var_2)
    # E-Step
    gamma_1 <- pi_1 * dnorm(Y, mu_1, sd = sqrt(var_1)) /
    (pi_1 * dnorm(Y, mu_1, sd = sqrt(var_1)) + pi_2 * dnorm(Y, mu_2, sd = sqrt(var_2)))
    gamma_2 <- pi_2 * dnorm(Y, mu_2, sd = sqrt(var_2)) /
    (pi_1 * dnorm(Y, mu_1, sd = sqrt(var_1)) + pi_2 * dnorm(Y, mu_2, sd = sqrt(var_2)))

    # Store parameters
    df <- rbind(df,
                tibble(
                  iter = iter,
                  ll = ll,
                  pi_1 = pi_1,
                  pi_2 = pi_2,
                  mu_1 = mu_1,
                  mu_2 = mu_2,
                  var_1 = var_1,
```

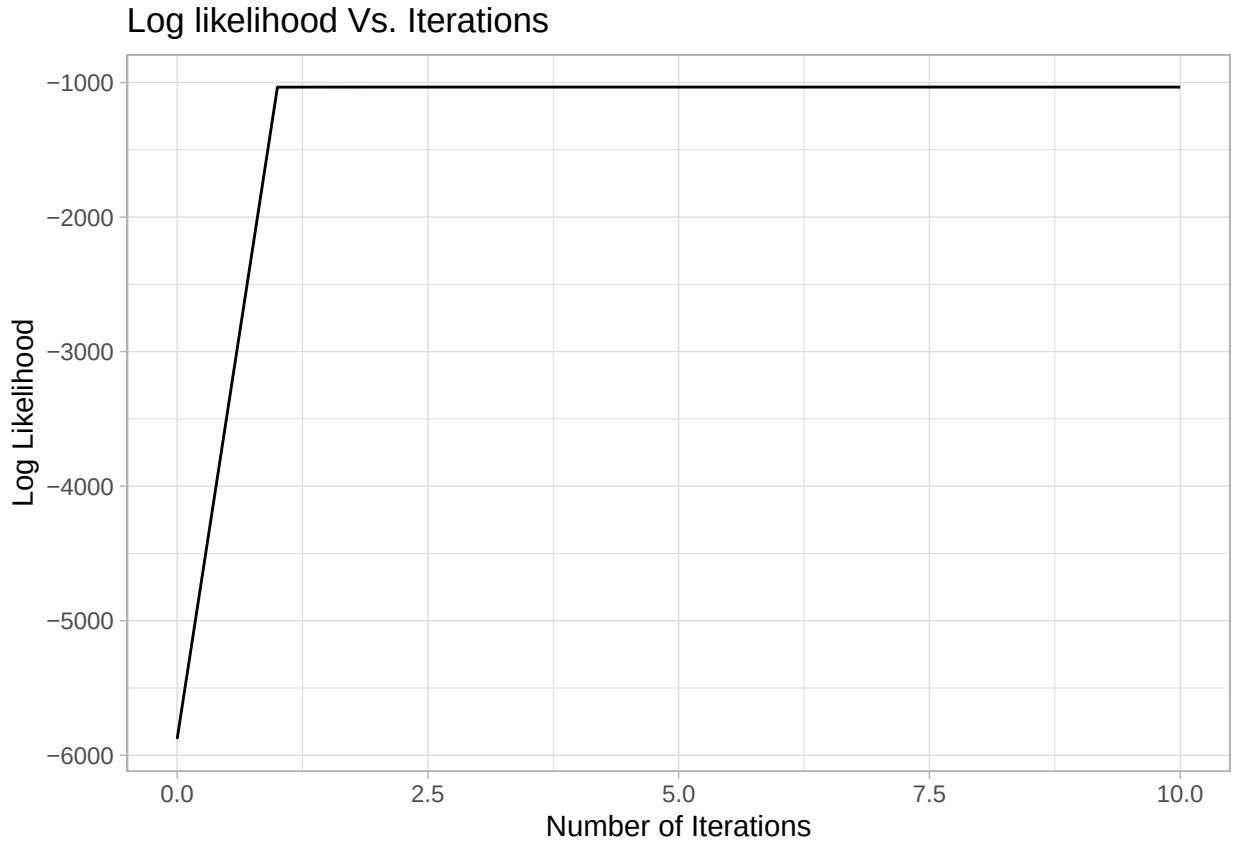
```
                  var_2 = var_2,
                  gamma_1 = list(gamma_1),
                  gamma_2 = list(gamma_2))
                )
    # M-Step
    pi_1 <- sum(gamma_1)/N
    pi_2 <- sum(gamma_2)/N
    mu_1 <-  sum(Y * gamma_1) / sum(gamma_1)
    mu_2 <-  sum(Y * gamma_2) / sum(gamma_2)
    var_1 <- sum((Y - mu_1)^2 * gamma_1) / sum(gamma_1)
    var_2 <- sum((Y - mu_2)^2 * gamma_2) / sum(gamma_2)

    # Evaluate the log likelihood at the new parameter values
    ll.new <- log_lik(Y = Y,
                      pi_1 = pi_1,
                      pi_2 = pi_2,
                      mu_1 = mu_1,
                      mu_2 = mu_2,
                      var_1 = var_1,
                      var_2 = var_2)
    # Once the difference between likelihood in the last iterations, change convergence to finish the l


    if(abs(ll - ll.new) < tol) {
      converged <- TRUE
      }
    # Onto the next iteration
    iter <- iter + 1
    # Give yourself a message to keep track of progress
    cat(paste0("Running iteration ", iter,
               ". Log likelihood changed by ", round(abs(ll - ll.new), 4), "\n"))
    }
  # Return the dataframe with the iterations information
  return(df)
  }


## Define starting values
starting_values <- list(pi_1 = .5, pi_2 = .5, mu_1 = 50, mu_2 = 80, var_1 = 1, var_2 = 1)


## Save the information ir the object em
em <- em_mixture(starting_values = starting_values, Y = waiting)
```

```
## Running iteration 1. Log likelihood changed by 4845.0201
## Running iteration 2. Log likelihood changed by 0.2642
## Running iteration 3. Log likelihood changed by 0.068
## Running iteration 4. Log likelihood changed by 0.0301
## Running iteration 5. Log likelihood changed by 0.0134
## Running iteration 6. Log likelihood changed by 0.0059
## Running iteration 7. Log likelihood changed by 0.0026
## Running iteration 8. Log likelihood changed by 0.0011
## Running iteration 9. Log likelihood changed by 5e-04
```

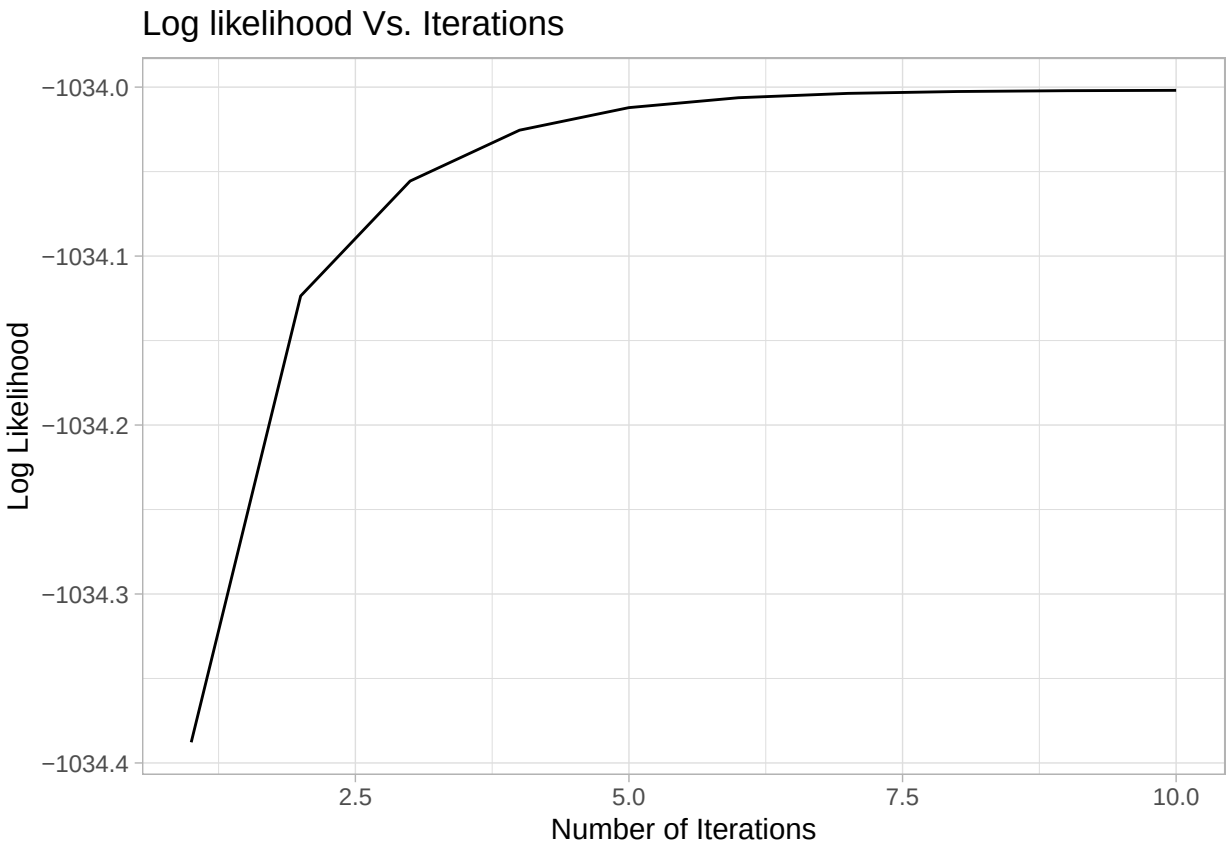
```
## Running iteration 10. Log likelihood changed by 2e-04
## Running iteration 11. Log likelihood changed by 1e-04
```

c. (4 points) Generate a plot with the log likelihoods you saved on the y-axis and the iterations of the algorithm on the x-axis. Do the same thing with one or two parameters. Briefly describe and explain what you see.

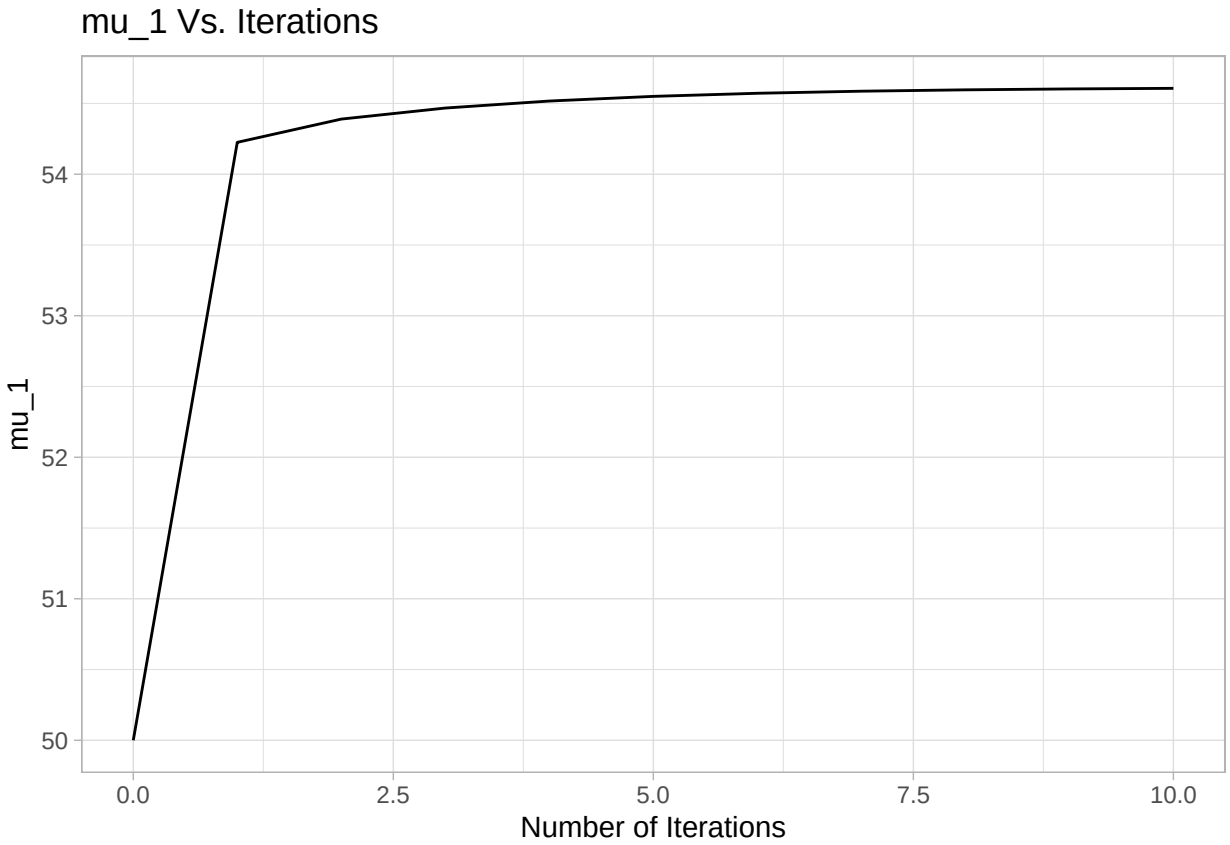
```
### Plot loglikelihood vs iterarions

ggplot(em, aes(x = iter, y = ll)) +
  geom_line() +
  labs(title = "Log likelihood Vs. Iterations",
       x = "Number of Iterations",
       y = "Log Likelihood") +
  theme_light()
```



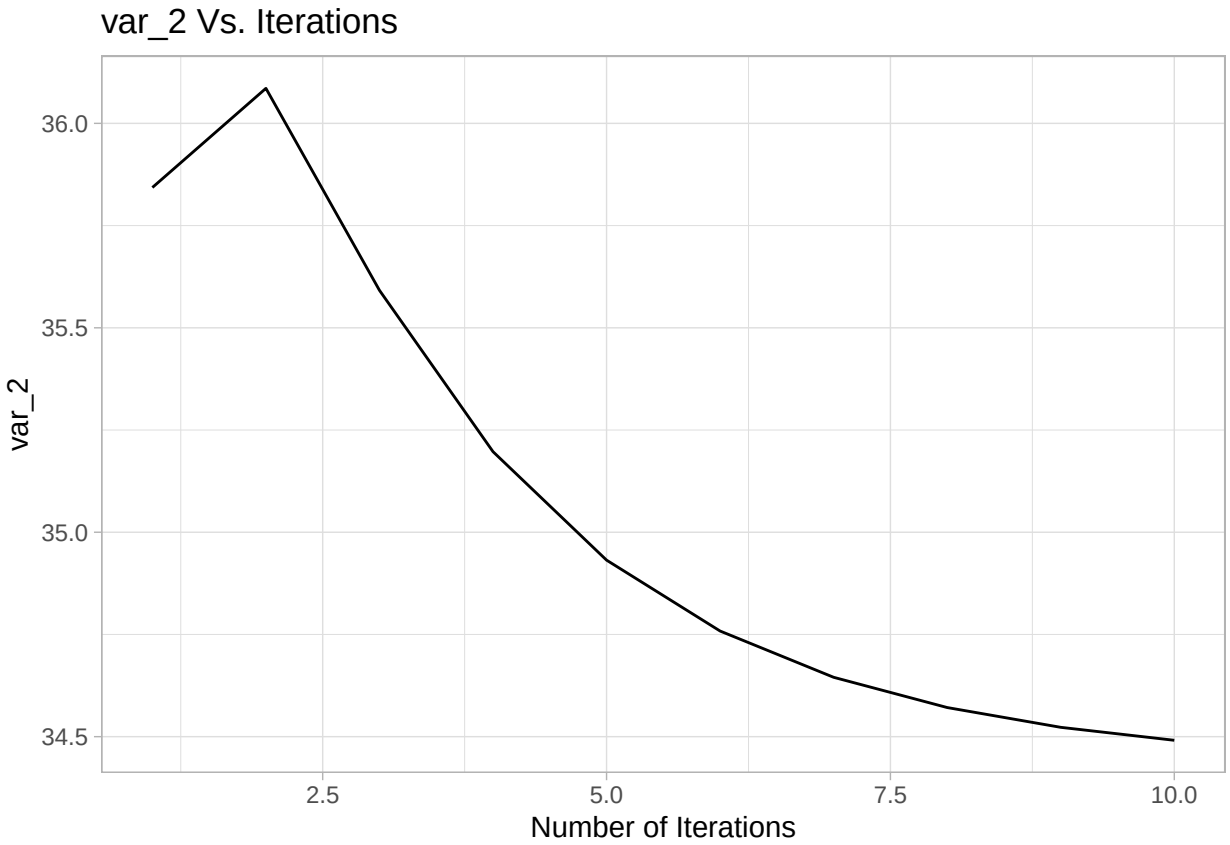

```
### Plot loglikelihood vs iterarions without the first row
em_1 <- em[-1, ]
ggplot(em_1, aes(x = iter, y = ll)) +
  geom_line() +
  labs(title = "Log likelihood Vs. Iterations",
       x = "Number of Iterations",
       y = "Log Likelihood") +
  theme_light()
```




```
### Plot mu_1 vs iterarions
ggplot(em, aes(x = iter, y = mu_1)) +
  geom_line() +
  labs(title = "mu_1 Vs. Iterations",
       x = "Number of Iterations",
       y = "mu_1") +
  theme_light()
```




```
### Plot var_2 vs iterarions without the first row
ggplot(em_1, aes(x = iter, y = var_2)) +
  geom_line() +
  labs(title = "var_2 Vs. Iterations",
       x = "Number of Iterations",
       y = "var_2") +
  theme_light()
```



It looks like when you look at the data with all the iterations (first graph), the log likelihoods spike very quickly and then plateaus. However, when you zoom in a bit further by removing the first iteration (second graph), you can see the gradual change from a typical log chart. This also shows that the values converge to a specific log likelihood. This trend can be seen with the other variables too such as the mu and variance which appears to converge as well.

d. (8 points) We now want to see whether the data points were classified similarly when using wait times vs. eruption duration.

- (4 points) Run your EM algorithm again for eruption duration. Compute the correlations of the relevant cluster membership probabilities and comment on what you see.

```
### Run EM algorithm for eruptions
eruptions = as.matrix(faithful[, 1, drop = FALSE])
starting_values_e <- list(pi_1 = .5, pi_2 = .5, mu_1 = 2, mu_2 = 5, var_1 = 1, var_2 = 1)
em_e <- em_mixture(starting_values = starting_values_e, Y = eruptions)
```

```
## Running iteration 1. Log likelihood changed by 135.6595
## Running iteration 2. Log likelihood changed by 18.6911
## Running iteration 3. Log likelihood changed by 13.8253
## Running iteration 4. Log likelihood changed by 15.4981
## Running iteration 5. Log likelihood changed by 8.028
## Running iteration 6. Log likelihood changed by 1.2646
## Running iteration 7. Log likelihood changed by 0.3859
## Running iteration 8. Log likelihood changed by 0.1745
```

```
## Running iteration 9. Log likelihood changed by 0.0815
## Running iteration 10. Log likelihood changed by 0.0359
## Running iteration 11. Log likelihood changed by 0.0147
## Running iteration 12. Log likelihood changed by 0.0057
## Running iteration 13. Log likelihood changed by 0.0021
## Running iteration 14. Log likelihood changed by 8e-04
## Running iteration 15. Log likelihood changed by 3e-04
## Running iteration 16. Log likelihood changed by 1e-04
```

```
### Include gamma 1 and gamma 2 for waiting ans eruptions in the data frame
gamma_1_w <- unlist(em[11,9])
gamma_1_e <- unlist(em_e[16,9])
gamma_2_w <- unlist(em[11,10])
gamma_2_e <- unlist(em_e[16,10])
faithful2 <- cbind(faithful, gamma_1_w, gamma_2_w, gamma_1_e, gamma_2_e)

### Correlation
cor(gamma_1_w, gamma_1_e)
```

```
## [1] 0.9555738
```

```
cor(gamma_2_w, gamma_2_e)
```

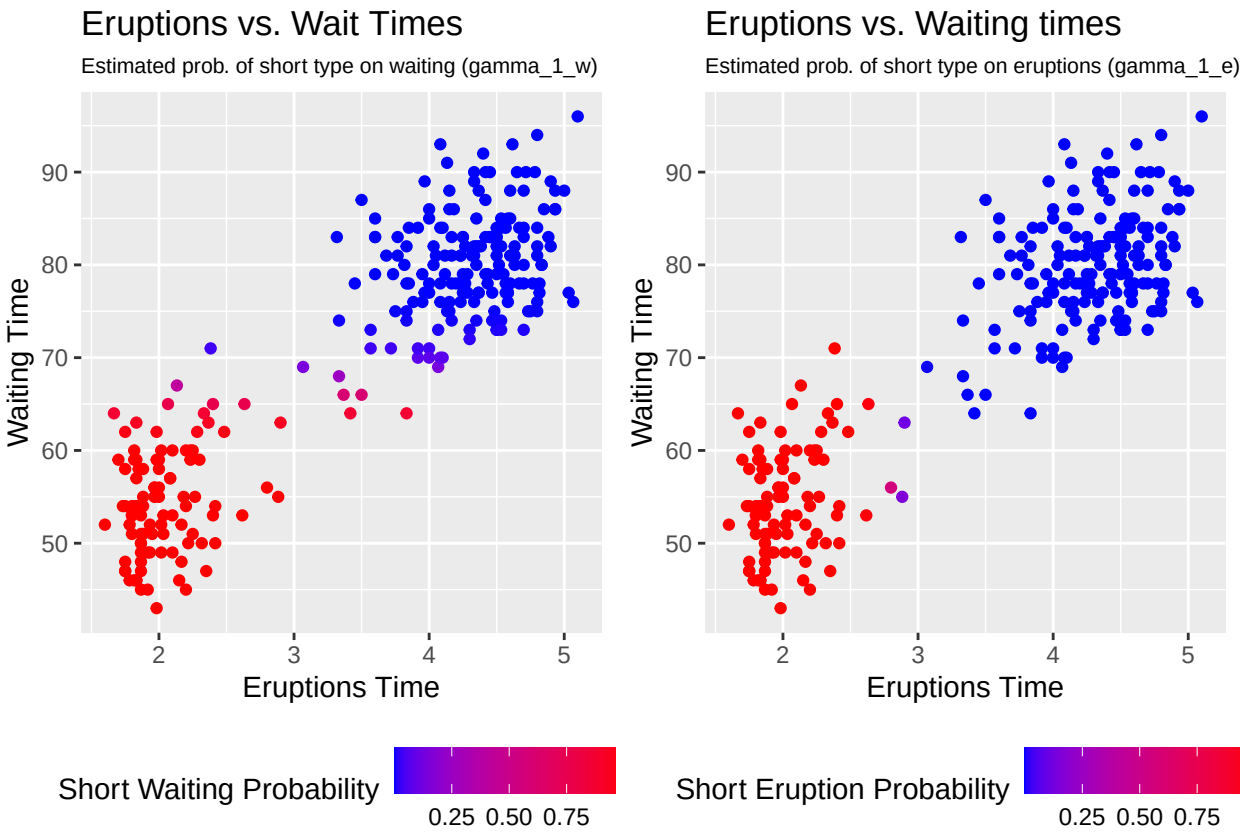
```
## [1] 0.9555738
```

The correlation between the estimated probabilities of being short type on waiting (gamma_1_w) and eruptions (gamma_1_e) is high (i.e. greater than 95%). This suggests a correlation between the two variables and it logically makes sense. The longer eruptions need more time "to get ready", hence a longer wait time versus a short type eruption which has a shorter wait time. This can be seen with an equal correlation between longer wait time (gamma_2_w) and longer eruption (gamma_2_e). This makes sense because correlation between the estimated probabilities of being long type (gamma_2) on waiting and eruptions is equal because those are defined as the complement of the short type ones (gamma_1).

- (4 points) Now, let's make two plots. In both plots, put eruption duration on the x-axis and wait times on the y-axis. In the first plot, color the points by their estimated probability of membership in the first cluster (short type) based on the wait times, and in the second plot color the points by their estimated probability of membership in the first cluster based on eruption duration. Put the plots side by side and make sure they are comparable to one another in axes, point colors, etc. Comment on what you see.

```
### Plot Eruptions vs Waiting with color the points by their estimated probability of membership in the
plot_w <- ggplot(faithful2,
                 aes(x = eruptions,
                     y = waiting,
                     color = gamma_1_w)) +
  geom_point() +
  labs(title = "Eruptions vs. Wait Times",
       subtitle ="Estimated prob. of short type on waiting (gamma_1_w)",
       x = "Eruptions Time",
       y = "Waiting Time",
       color = "Short Waiting Probability") +
  scale_color_gradient(low = "blue", high = "red")+
```

```
  theme(legend.position = "bottom",
        plot.subtitle = element_text(size = 8))

plot_e <- ggplot(faithful2,
                 aes(x = eruptions,
                     y = waiting,
                     color = gamma_1_e)) +
  geom_point() +
  labs(title = "Eruptions vs. Waiting times",
       subtitle ="Estimated prob. of short type on eruptions (gamma_1_e)",
       x = "Eruptions Time",
       y = "Waiting Time",
       color = "Short Eruption Probability"
       ) +
  scale_color_gradient(low = "blue", high = "red")+
  theme(legend.position = "bottom",
        plot.subtitle = element_text(size = 8))

### Arrange plots
gridExtra::grid.arrange(arrangeGrob(plot_w, plot_e, widths = c(1, 1)), ncol = 1)
```



The plots are very similar and the information is consistent with the correlation finding from before. There are two clusters on the graphs. The probability that the eruption is a short type and has a short waiting time are in red. Whereas, the longer eruptions and longer waiting times are clustered near each other (in blue). The blue represents a low probability that the eruption and the waiting times are of the long type.

2. The Law of Large Numbers and the Central Limit Theorem

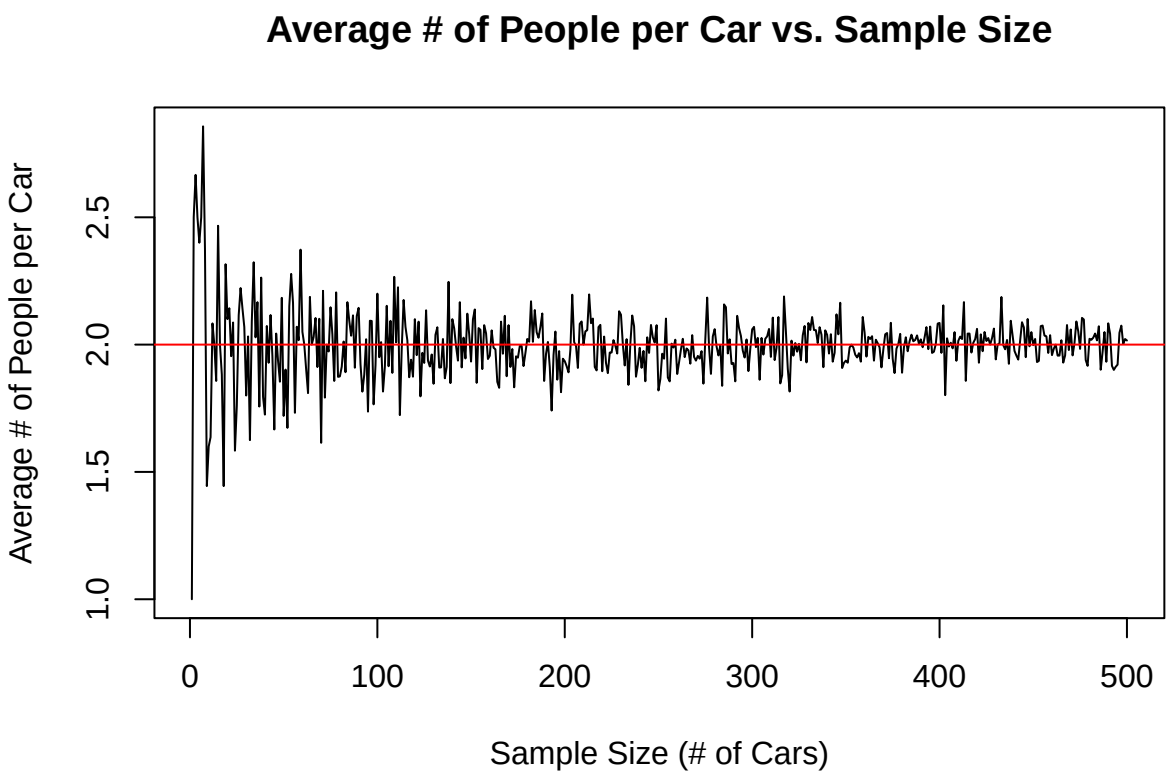
(20 points) You are an urban planner interested in finding out how many people enter and leave the city using personal vehicles every day. (You're not interested in the number of *cars*; you're interested in the number of *people* who use cars to get to work.) To do this, you decide to collect data from a few different points around the city on how many people there are per car. You already have reliable satellite data on the number of cars that come into the city, so if you get a good estimate of people per car you'll be in good shape.

Collecting data on people per car is costly and you'd love to minimize how many data points you have to collect. However, you're also familiar with the Law of Large Numbers and know that the sample mean converges to the true mean as the sample size n grows large.

a. (5 points) Let's illustrate this with a small simulation. Suppose the number of people in a car is distributed Poisson with a rate of $\lambda = 2$ people per car.[1] Construct 500 samples from this distribution, with the first sample having $n = 1$ cars, the second $n = 2$ cars, and so on. Compute the average number of people per car in each sample. Plot this on the y-axis against the sample size on the x-axis and run a horizontal blue line through the true mean. Comment on what you see.

```r
set.seed(123)

# Define the rate of the Poisson distribution
lambda <- 2

# Define the number of samples
n_samples <- 500

# Define the number of cars for each sample
n_cars <- 1:n_samples

# Generate the samples
samples <- lapply(n_cars, function(n) rpois(n, lambda))

# Compute the average number of people per car for each sample
avg_people_per_car <- sapply(samples, mean)

# Plot the results
plot(n_cars, avg_people_per_car, type = "l",
     xlab = "Sample Size (# of Cars)",
     ylab = "Average # of People per Car",
     main = "Average # of People per Car vs. Sample Size")
abline(h = lambda, col = "red")
```

[1] I should have mentioned that you're an urban planner in San Francisco, where it's rare but possible to have 0 people in a car.



From the graph above, there was a large variation in means at first. However, as the number of cars increased the line seems to converge at the true mean. This demonstrates the law of large numbers as the sample size increases the data approaches the true mean which in this case is 2.

b. (4 points) You collect data on 100 cars and compute the average number of people per car in this sample. Use the Central Limit Theorem to write down the approximate distribution of this quantity.

Given: X = number of people per car, $X \sim Poisson(\lambda = 2)$ The σ^2 in a poisson distribution is λ

$\sigma^2 * \sigma^2 = 2$

$\sigma^2 = 2$

According to the Central Limit Theorem, when n is big enough, the sample mean has this distribution:

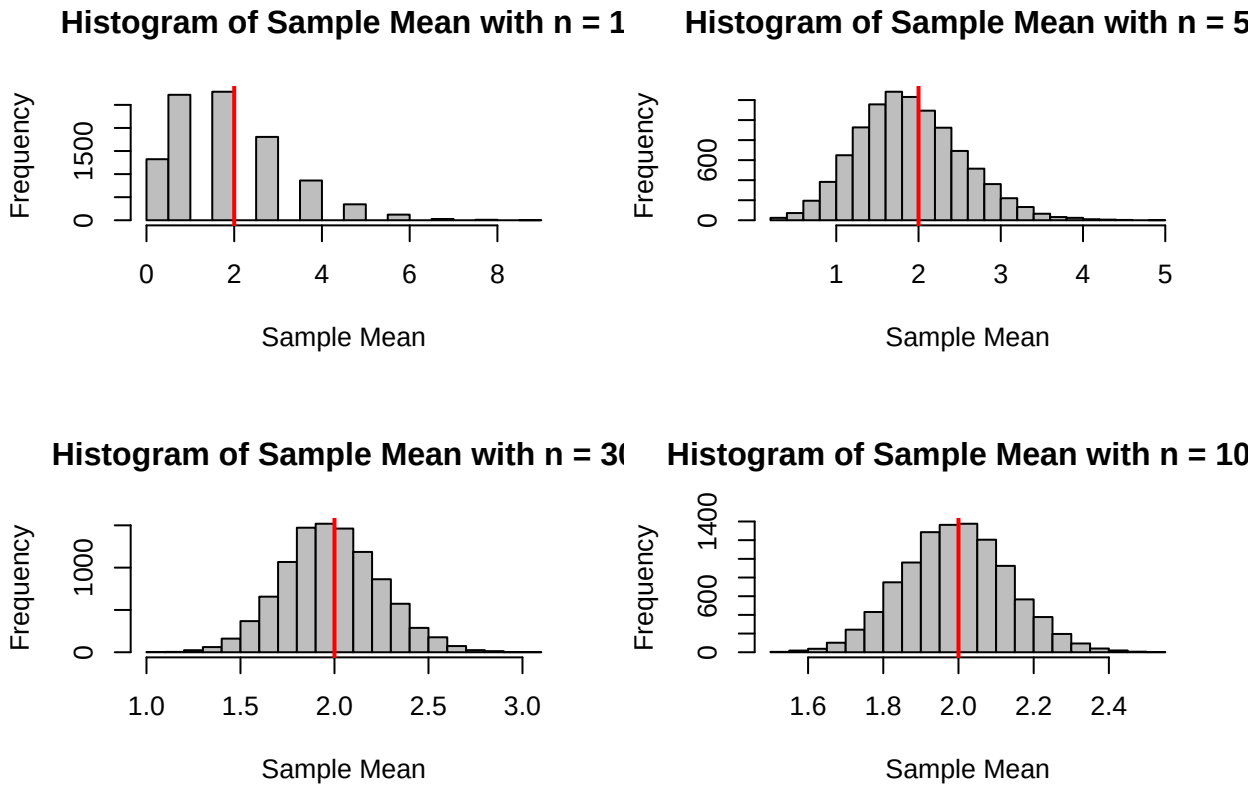
$\sigma^2 \approx N(\frac{\sigma^2}{n})$

$\sigma^2 \approx N(2, \frac{2}{100})$

c. (6 points) Let's examine this distribution more closely. Generate 10,000 replicates of the sample mean with $n = 100$ and plot a histogram.[2] Are you convinced that the Normal approximation you found in the previous question is good enough? Compare this to $n = 1$, $n = 5$, and $n = 30$, generating a histogram for each. (We're aiming to recreate the second row of Figure 10.5 from Slide 12 of Lecture 7.) Comment on what you observe.

[2]Try using the `replicate` function rather than a loop, as this will speed things up considerably.

```r
# Set the seed for reproducibility
set.seed(123)

# Define the sample sizes
n <- c(1, 5, 30, 100)

# Generate the histograms
par(mfrow = c(2, 2))
for (i in 1:length(n)) {
  # Generate the replicates
  replicates <- replicate(10000, mean(rpois(n[i], lambda)))

  # Plot the histogram
  hist(replicates, breaks = 20, col = "gray",
       main = paste0("Histogram of Sample Mean with n = ", n[i]),
       xlab = "Sample Mean")

  # Add a vertical line at the true mean
  abline(v = lambda, col = "red", lwd = 2)
}
```



When there are less cars, the distribution is skewed. As the number of cars increase, data points get filled up (so at n = 5 there are no gaps). As the number of cars increase, the distribution becomes more like a normal distribution. For example, $n = 5$ is more skewed than $n = 30$ which is more skewed than $n = 100$ (the bulge approaches the mean with more cars added). In fact, ten thousand replicates of $n = 100$ looks very much like a normal distribution where most of the values are centered around the mean.

d. (5 points) Suppose the city government will enact measures to regulate the number of people allowed per car during rush hour if they think the mean is below 1.7 people per car. Using the Normal approximation from part (b) above, find the probability that you get a mean of 1.7 **or less** in your sample of 100, even though the true mean is 2. (Please give the theoretical answer, not a simulation. You can use `R` as a calculator.) What should you do to ensure that this probability stays below 1%?

```
true_mean <- 2
sigma <- sqrt(2)
n_cars <- 100
x_bar <- 1.7

z_score <- (x_bar - true_mean) / (sigma / sqrt(n_cars))

prob <- pnorm(z_score)

cat("The probability of getting a 1.7 or less", prob, "\n")
```

```
## The probability of getting a 1.7 or less 0.01694743
```

```
#Z-score formula solve for a new value, find the score when prob is 0.01

req_xbar <- ( qnorm(0.01) * (sigma / sqrt(n_cars)) ) + true_mean

cat("To ensure the probability stays below 1%, you need to make sure the mean number of people per car
```

```
## To ensure the probability stays below 1%, you need to make sure the mean number of people per car is
```

The mean of 1.7 has a greater than 1% chance with 100 cars. To ensure the probability is below 1%, the mean needs to be less than 1.7. For the city government, they will need to pass regulations to reduce the number of people per car during rush hour or remove incentives for having too many people per car. Alternatively, they can increase the number of cars which would make sample mean closer to the true mean which would also decrease the probability of getting a value of 1.7. Something like this in real life would be if you add more lanes so that there are more cars "to sample". However, this may exacerbate rush hour traffic problems.