Math for Data Science: Problem Set 3

Name: GIULIA_MARIA_PETRILLI, Group: DANIEL_BOPPERT, LONNY_CHEN

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.)

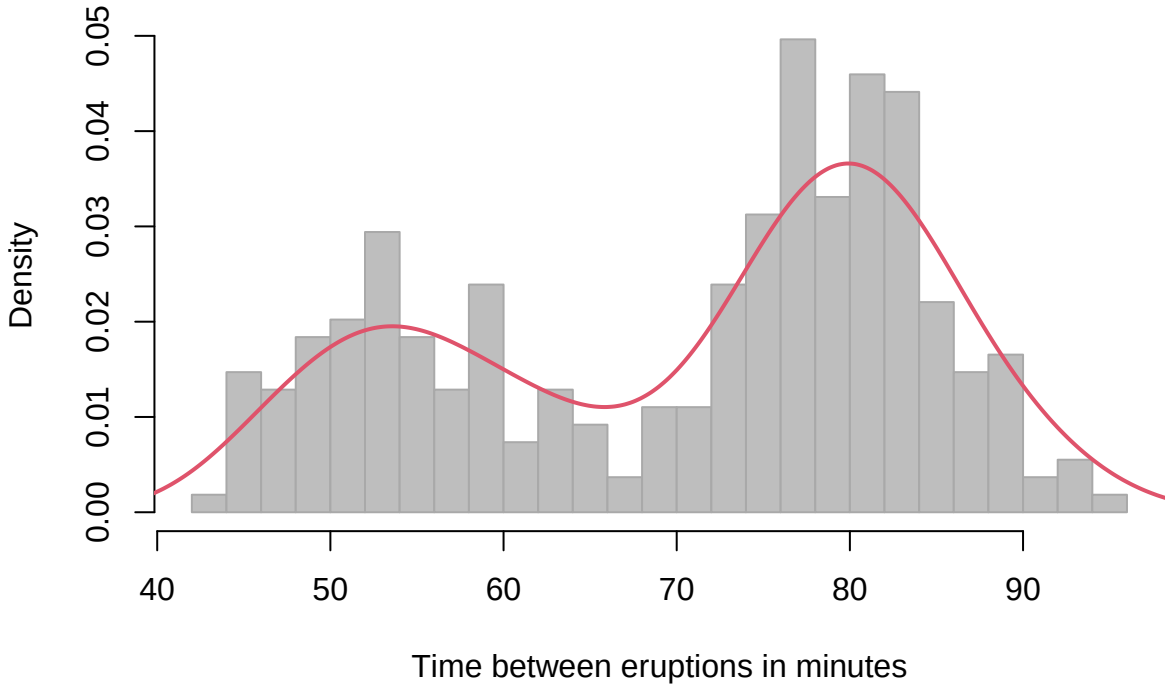
Instructions: Please submit one solution set per person and include your name and your group members' names at the top. This time, please write your solutions within this Rmd file, under the relevant question. Please submit the knitted output as a pdf. Make sure to show all code you used to arrive at 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.

```
waiting = as.matrix(faithful[, 2, drop = FALSE])
h <- hist(waiting, col = "grey", bor = "darkgrey", breaks = 20, prob = TRUE,
          xlab = "Time between eruptions 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.

Prepare the data. Write a function to compute your log likelihood.

```
X <- waiting
N <- length(X)

log.lik <- function(X, mu_1, mu_2, var_1, var_2, pi_1, pi_2) {
  sum(log(pi_1 * dnorm(X, mu_1, sd = sqrt(var_1)) + pi_2 * dnorm(X, mu_2, sd = sqrt(var_2))))
}
```

1. Initialize your parameters. I set these parameters, and I set randomly or taking educated guesses from the data, because the algorithm needs a starting point to improve from.

```
# Initialize your parameters.
# Chosen randomly but approximating from the graph above
pi_1 <- .5
pi_2 <- .5
mu_1 <- 90
```

```
mu_2 <- 100
var_1 <- 40
var_2 <- 40

# evaluate the log likelihood at these parameters
ll <- log.lik(X = X,
              pi_1 = pi_1,
              pi_2 = pi_2,
              mu_1 = mu_1,
              mu_2 = mu_2,
              var_1 = var_1,
              var_2 = var_2)
```

2. E-Step: evaluate the responsibilities using the initial parameter values. Gamma 1 and 2 are the probabilities that the values observed belong to the first or to the second latent classes, governed by the parameters specified above. They are also called responsibilities, as they indicate the probability that a given data point belongs to a particular component (or class) of the model, given the current parameter estimates - "To what extent are the latent classes under the specified parameters *responsible* for the data points' distribution?"

```
gamma_1 <- pi_1 * dnorm(X, mu_1, sd = sqrt(var_1)) /
  (pi_1 * dnorm(X, mu_1, sd = sqrt(var_1)) + pi_2 * dnorm(X, mu_2, sd = sqrt(var_2)))
gamma_2 <- pi_2 * dnorm(X, mu_2, sd = sqrt(var_2)) /
  (pi_1 * dnorm(X, mu_1, sd = sqrt(var_1)) + pi_2 * dnorm(X, mu_2, sd = sqrt(var_2)))
```

3. M-Step: reestimate the parameter values using the responsibilities you just calculated in the E-Step.

```
pi_1 <- sum(gamma_1)/N
pi_2 <- sum(gamma_2)/N
mu_1 <-  sum(X * gamma_1) / sum(gamma_1)
mu_2 <-  sum(X * gamma_2) / sum(gamma_2)
var_1 <- sum((X - mu_1)^2 * gamma_1) / sum(gamma_1)
var_2 <- sum((X - mu_2)^2 * gamma_2) / sum(gamma_2)
```

4. Evaluate the log-likelihood under the new responsibilities and parameter values.

```
ll.new <- log.lik(X = X,
              pi_1 = pi_1,
              pi_2 = pi_2,
              mu_1 = mu_1,
              mu_2 = mu_2,
              var_1 = var_1,
              var_2 = var_2)
```

5. Check convergence.

```
abs(ll - ll.new)
```

```
## [1] 1703.371
```

Finally, We Loop Until Convergence.

```
# Finally, We Loop Until Convergence
em_mixture <- function(starting_values, X, tol = .0001, maxits = 100) {

  # Initialize convergence flag and iteration counter
  converged <- FALSE
  iter <- 0
```

```r
N <- length(X)

# 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

# Initialize list to store log-likelihood at each iteration, as well as two params.
likelihood_vector <- list()
variance_vector <- list()
pi_vector <- list()

while ((!converged) & (iter < maxits)) {
  # Evaluate the log likelihood with current parameter values
  ll <- log.lik(X = X, 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: Calculate responsibilities
  gamma_1 <- pi_1 * dnorm(X, mu_1, sd = sqrt(var_1)) /
    (pi_1 * dnorm(X, mu_1, sd = sqrt(var_1)) + pi_2 * dnorm(X, mu_2, sd = sqrt(var_2)))
  gamma_2 <- 1 - gamma_1

  # M-Step: Update parameter estimates
  pi_1 <- sum(gamma_1) / N
  pi_2 <- sum(gamma_2) / N
  mu_1 <- sum(X * gamma_1) / sum(gamma_1)
  mu_2 <- sum(X * gamma_2) / sum(gamma_2)
  var_1 <- sum(gamma_1 * (X - mu_1)^2) / sum(gamma_1)
  var_2 <- sum(gamma_2 * (X - mu_2)^2) / sum(gamma_2)

  # Evaluate the log likelihood with updated parameter values
  ll.new <- log.lik(X = X, pi_1 = pi_1, pi_2 = pi_2,
                    mu_1 = mu_1, mu_2 = mu_2, var_1 = var_1, var_2 = var_2)

  # Check for convergence
  if (abs(ll - ll.new) < tol) {
    converged <- TRUE
  }

  # Append the new log-likelihood value to the vector
  likelihood_vector[[iter + 1]] <- ll.new
  variance_vector[[iter + 1]] <- var_2
  pi_vector[[iter + 1]] <- pi_2


  # Increment the iteration counter
  iter <- iter + 1
}

# Convert the likelihood list to a vector
```

```
  likelihood_v <- unlist(likelihood_vector)
  #print(likelihood_v)

  variance_v <- unlist(variance_vector)
  #print(variance_v)

  pi_v <- unlist(pi_vector)
  #print(pi_v)

  gammas_df <- data.frame(gamma_1 = gamma_1, gamma_2 = gamma_2)

  # Return the final parameters and likelihood vector
  params <- list(pi_1 = pi_1, pi_2 = pi_2, mu_1 = mu_1, mu_2 = mu_2,
                 var_1 = var_1, var_2 = var_2, likelihood = likelihood_v,
                 variance = variance_v, pi = pi_v, gammas = gammas_df)

  return(params)
}

starting_values_1 <- list(pi_1 = .5, pi_2 = .5, mu_1 = 60, mu_2 = 100, var_1 = 40, var_2 = 40)
em1 <- em_mixture(starting_values = starting_values_1, X = waiting)
```

```
# Making the required elements for each iteration available to the user
likelihoods_final <- em1$likelihood
variance_final <- em1$variance
pi_final <- em1$pi
gammas_final <- em1$gammas

# Changing the name for the columns in gamma
names(gammas_final)[1] <- "gamma_1_waiting"
names(gammas_final)[2] <- "gamma_2_waiting"

# Printing the heads demonstrate that my algorithm successfully accomplishes the task
head(variance_final)
```

```
## [1] 15.01030 17.65421 18.94091 19.86077 20.75567 21.69570
```

```
head(pi_final)
```

```
## [1] 0.3215235 0.3344310 0.3607572 0.3878755 0.4121446 0.4334773
```

```
head(gammas_final)
```

```
##   gamma_1_waiting gamma_2_waiting
## 1    1.055863e-04    9.998944e-01
## 2    9.999110e-01    8.896047e-05
## 3    4.207612e-03    9.957924e-01
## 4    9.677853e-01    3.221469e-02
## 5    1.267985e-06    9.999987e-01
## 6    9.998134e-01    1.866419e-04
```

```
# Parameters are accessible here
parameters <- em1[1:6]
parameters
```

```
## $pi_1
```

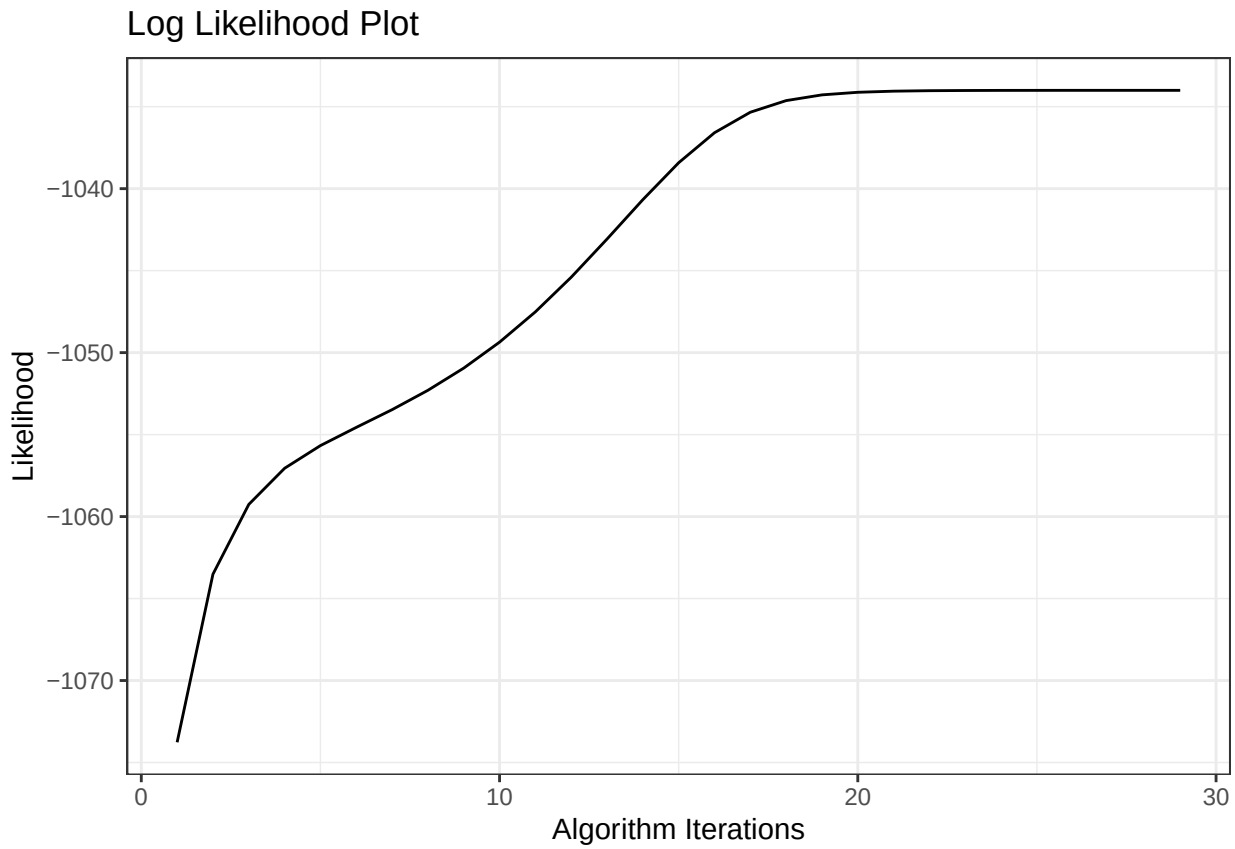
```
## [1] 0.3610383
##
## $pi_2
## [1] 0.6389617
##
## $mu_1
## [1] 54.61993
##
## $mu_2
## [1] 80.09427
##
## $var_1
## [1] 34.5222
##
## $var_2
## [1] 34.39279
```

```
# My algorithm is running 29 iterations
length(likelihoods_final)
```

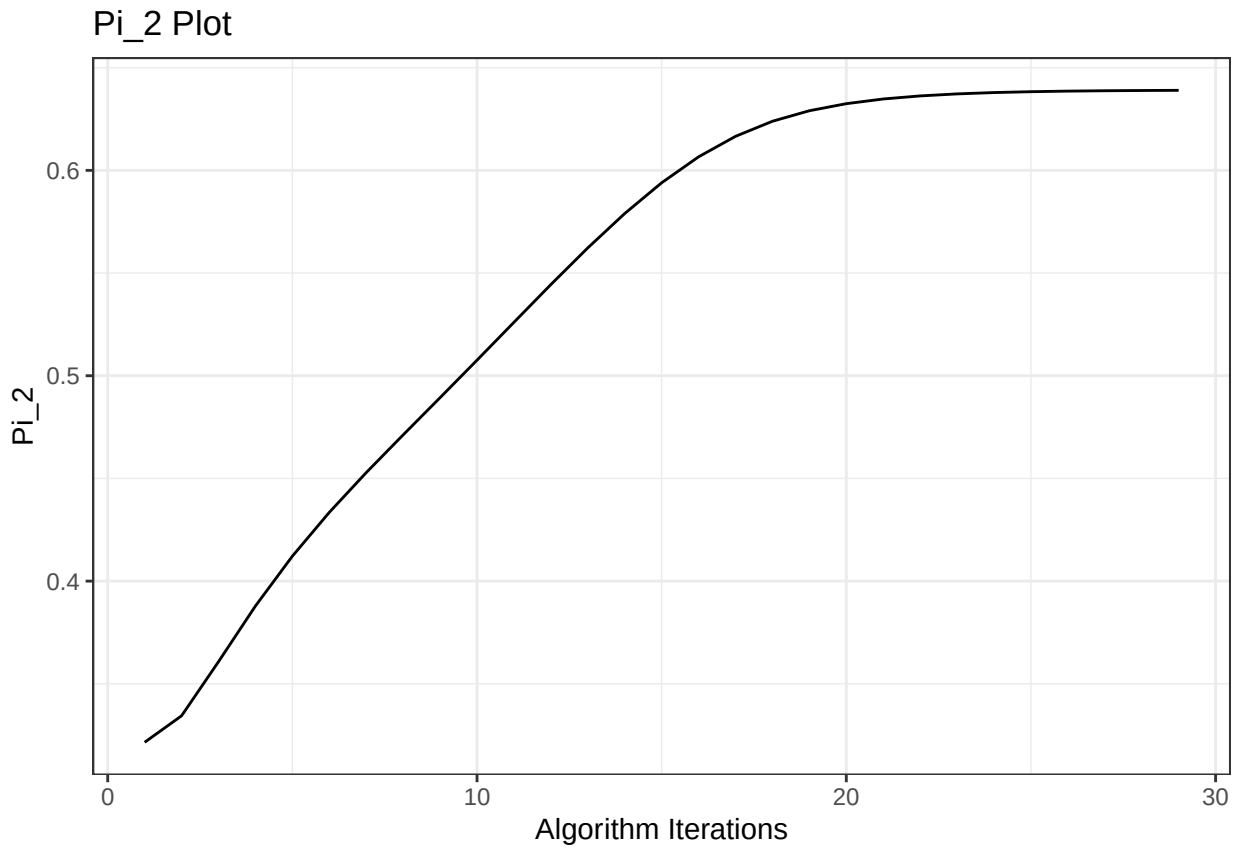
```
## [1] 29
```

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.

```
# My parameters of choiche are log likelyhood and pi for the second distribution

logs_df <- as.data.frame(likelihoods_final) %>%
            mutate(iteration = 1:length(likelihoods_final))

logs_plot <- ggplot(logs_df, aes(x = likelihoods_final, y = iteration)) +
              geom_line(stat = 'identity') +
              labs(y = "Algorithm Iterations", x = "Likelihood", title = "Log Likelihood Plot") +
              coord_flip() +
              theme_bw()

logs_plot
```




```
pi_2_df <- as.data.frame(pi_final) %>%
                mutate(iteration = 1:length(pi_final))

pi_2_plot <- ggplot(pi_2_df, aes(x = pi_final, y = iteration)) +
                  geom_line(stat = 'identity') +
                  labs(y = "Algorithm Iterations", x = "Pi_2", title = "Pi_2 Plot") +
                  coord_flip() +
                  theme_bw()

pi_2_plot
```



- For the logs plot: the likelihood gets closer to the maximum likelihood through the whole curve, with a small dip at around 10 iterations. The adjustments starts being small and then almost non existent at around 20 iterations, which means that a close approximation to the max likelihood has been found.

- For the pi plot: this parameter adjusts gradually but constantly, but only reaches a point where there needs to be little adjustment around iteration 20 too.

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.
- (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.

```
# Comparison of cluster membership probability
eruption <- as.matrix(faithful[, 1 , drop = FALSE])
E_starting_values_1 <- list(pi_1 = .5, pi_2 = .5, mu_1 = 2, mu_2 = 5, var_1 = 1, var_2 = 1)

#starting_values_1 <- list(pi_1 = .5, pi_2 = .5, mu_1 = 52, mu_2 = 79, var_1 = 20, var_2 = 20)
E_em1 <- em_mixture(starting_values = E_starting_values_1, X = eruption)

E_gammas_final <- as.data.frame(E_em1$gammas)

all_gammas_df <- cbind.data.frame(waiting_gamma_1 = gammas_final$gamma_1_waiting,
                                  waiting_gamma_2 = gammas_final$gamma_2_waiting,
```

```
                                  eruption_gamma_1 = E_gammas_final$eruptions,
                                  eruption_gamma_2 = E_gammas_final$eruptions.1)

# Correlation between waiting time and eruption times for short type
gammas_1_cor <- cor(all_gammas_df$waiting_gamma_1, all_gammas_df$eruption_gamma_1)
gammas_1_cor
```

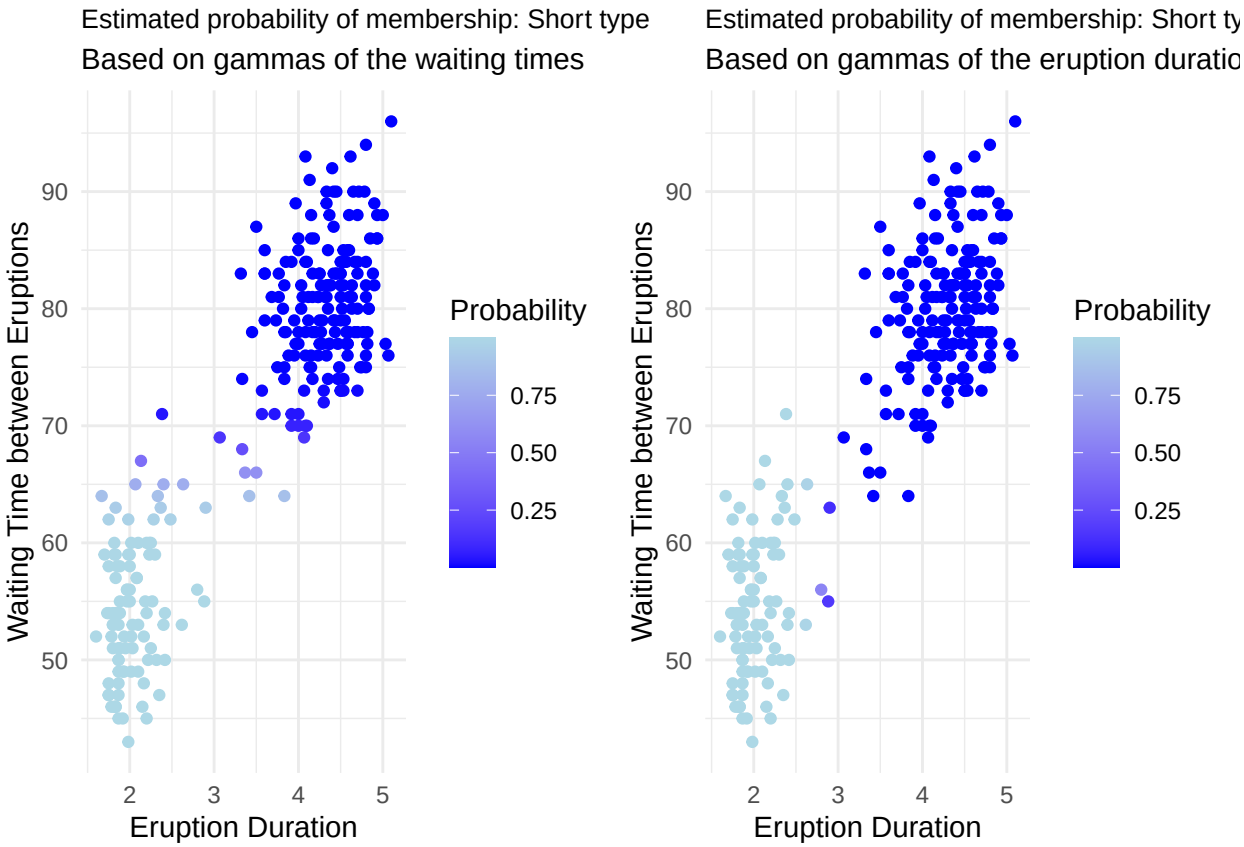
```
## [1] 0.9554664
```

```
# Correlation between waiting time and eruption times for long type
gammas_2_cor <- cor(all_gammas_df$waiting_gamma_2, all_gammas_df$eruption_gamma_2)
gammas_2_cor
```

```
## [1] 0.9554664
```

The correlations are the same because if you add gamma 2 and 1 up you of course get 1. Gamma 1 and gamma 2 carry the same information probability wise; If you know the probability that a eruption is in the short type, you also know the probability that it is in the long type.

```
pl <- ggplot(data = all_gammas_df, aes(x = eruption, y = waiting, color = waiting_gamma_1)) +
  scale_color_gradient(low = "blue", high = "lightblue", name = "Probability") +
  theme_minimal() +
  labs(x = "Eruption Duration",
       y = "Waiting Time between Eruptions",
       title = "Estimated probability of membership: Short type",
       subtitle = "Based on gammas of the waiting times") +
  theme(legend.position = "right",
        plot.title = element_text(size = 10, lineheight = 0.8)) +
  geom_point()


pl1 <- ggplot(data = all_gammas_df, aes(x = eruption, y = waiting, color = eruption_gamma_1)) +
  scale_color_gradient(low = "blue", high = "lightblue", name = "Probability") +
  theme_minimal() +
  labs(x = "Eruption Duration",
       y = "Waiting Time between Eruptions",
       title = "Estimated probability of membership: Short type",
       subtitle = "Based on gammas of the eruption duration ") +
  theme(legend.position = "right",
        plot.title = element_text(size = 10, lineheight = 0.8)) +
  geom_point()

grid.arrange(pl, pl1, ncol = 2)
```



The plot on the right estimates membership based on eruption duration, while the plot on the left estimates membership based on waiting time between eruptions. Both correlations are ~ 0.96, hence it makes sense that the plots are almost exactly the same.

If you estimate the membership based on wait times, the data points are gonna be divided horizontally, which makes sense as the Waiting Time variable is on the Y axis. Similarly, if you estimate the membership based on Eruption duration, the data points are gonna be divided vertically, which makes sense as the Eruption duration variable is on the Y axis.

If a data point is light blue, it means that the algorithm recognizes a high probability that the eruption belongs in the short type membership. Same goes for the dark blue, where the algorithm recognizes a high probability that the observation is created under the short type membership.

There are some points in the middle where the algorithm is not very sure, especially for the plot that is based on the waiting times between eruptions gamma.

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.

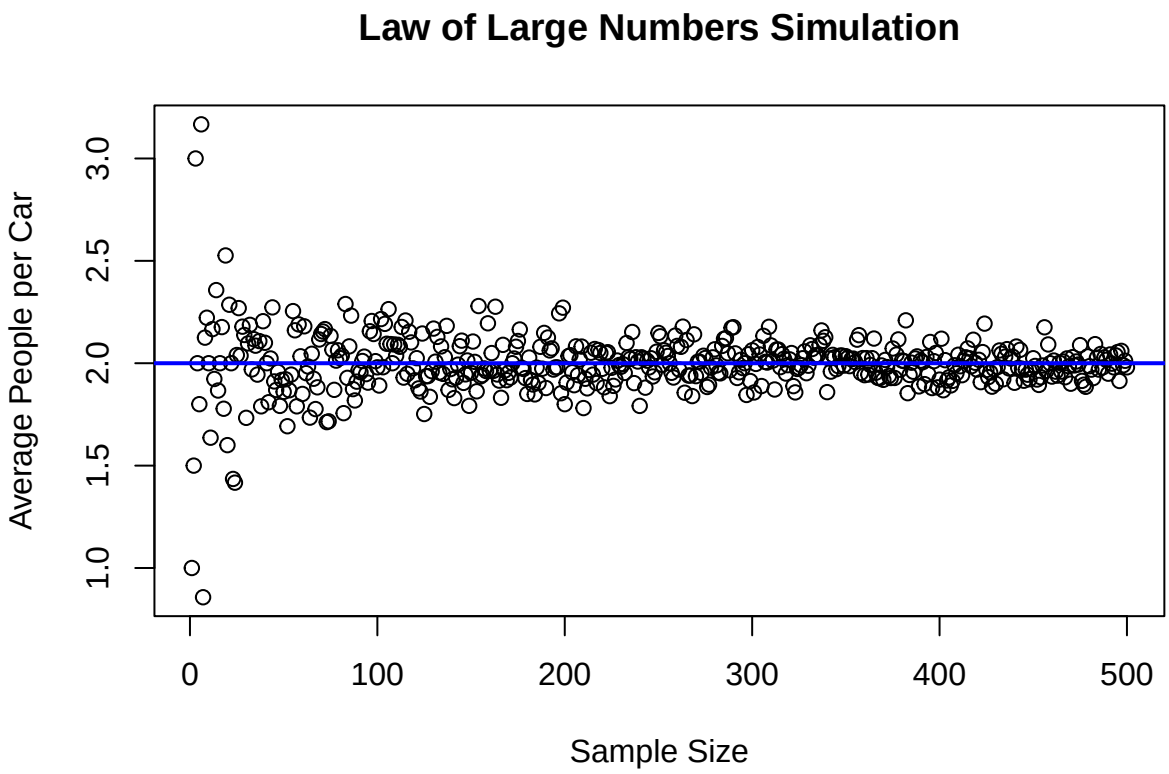
Collecting data on people per car is costly and you'd love to minimize how many data points you have to collect. However, you're also familiar with the Law of Large Numbers and know that the sample mean converges to the true mean as the sample size n grows large.

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.

```
# Set the True mean of people per car
lambda_rate <- 2

set.seed(1) # For reproducibility

sample_sizes <- seq(1, 500)
sample_means <- numeric(500)

# Generate 500 samples, with the first sample having 1 car, the second one 2 cars and so on
# For each sample, generate 'n' random values from the Poisson distribution

for (size in sample_sizes) {
sample <- rpois(size, lambda_rate)
sample_mean <- mean(sample)
sample_means[size] <- sample_mean
}

# Plot
# y = average number of people per car in each sample
# x = sample size

plot(sample_sizes, sample_means, type = "p", main = 'Law of Large Numbers Simulation',
xlab = 'Sample Size', ylab = 'Average People per Car') +
  # Line to signal the true mean
abline(h = 2, col = 'blue', lwd = 2)
```

[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.




```
## integer(0)
```

I can draw some conclusions from the plot above: - The bigger the sample size, the closer the average number of people becomes to our lambda 2. - When the sample sizes are small, the observations are more spread out.

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.

The Central Limit Theorem states that to the distribution of the sample average for almost any process, even non-Normal, is normally distributed provided the process has well defined mean and variance. Of course, the approximation improves as n approaches infinity. The variance for a single value of the random variable X is sigmaˆ2. Hence, the variance for the sample mean of the random variable X is sigmaˆ2/n.

The CLT can be written in approximation form as follows:

$$\bar{X}_n \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right)$$

The sample mean bar X for n observations is approximately normally distributed with a mean mu and a variance for the sample mean sigma squared (or variance for a single value of the random variable) divided by n.

The Poisson distribution has a particularly simple mean, E (X) = lambda , and variance, V (X) = lambda (for clarity purposes, this is as saying sigmaˆ2)

$$\bar{X}_{100} \sim \mathcal{N}\left(\lambda, \frac{\lambda}{100}\right)$$

Our mean and variance will both be 2. Hence:

$$\bar{X}_{100} \sim \mathcal{N}\left(2, \frac{2}{100}\right)$$

Which can be written as follows:

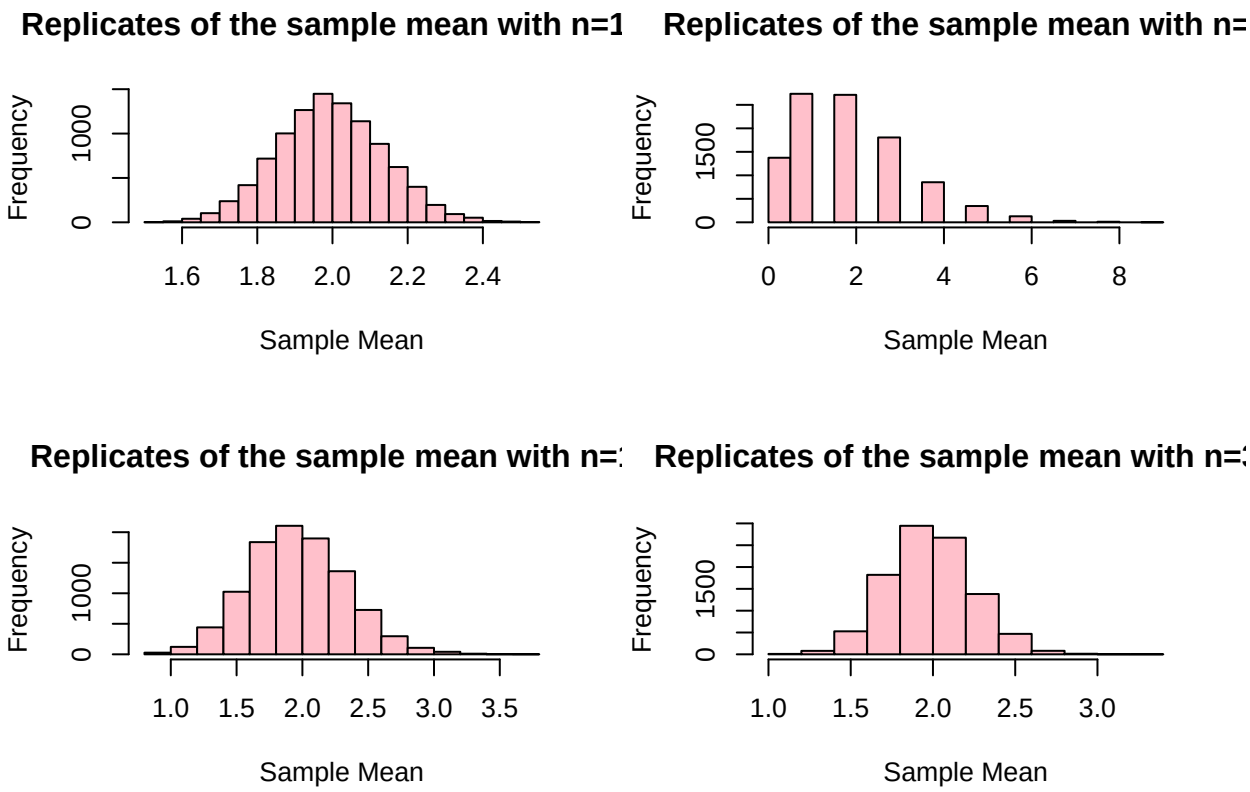
$$\bar{X}_{100} \sim \mathcal{N}\left(2, 0.02\right)$$

The random variable above will look normal once we replicate it many times.

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.

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

v100 <- replicate(10000, mean(rpois(n = 100, 2))) |>
  hist(main = "Replicates of the sample mean with n=100", xlab = "Sample Mean", col = "pink")

v1 <- replicate(10000, mean(rpois(n = 1, 2))) |>
  hist(main = "Replicates of the sample mean with n=1",  xlab = "Sample Mean", col = "pink")

v15 <- replicate(10000, mean(rpois(n = 15, 2))) |>
  hist(main = "Replicates of the sample mean with n=15", xlab = "Sample Mean", col = "pink")

v30 <- replicate(10000, mean(rpois(n = 30, 2))) |>
  hist(main = "Replicates of the sample mean with n=30", xlab = "Sample Mean", col = "pink")
```

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



Here, I am Re sampling 10000 times from the poisson distribution, and changing n as I go.
One mistake one might do in this exercise is to sample from a sample. The correct way to do it, as it reduces the noise in the mean, is to sample from the poisson distribution.

To answer the question: "Are you convinced that the Normal approximation you found in the previous question is good enough?". No, because generating many (i.e 10,000) replicates of the sample mean allows to have a normal distribution. The bigger the number of observations, the more distributed around the mean and following a bell shape the distribution will be. The approximation improves as n approaches infinity, but starts to look normal as n >= 30.

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%?

This question asks : " If it is true that the random variable of people per car follows a poisson distribution with lambda 2, what is the probability that the mean of a sample will be less or equal than 1.7?"

We are looking for Pr(X <= 1.7) By CLT, we know that as the sample goes bigger, the variable will have a normal distribution, but in a real world scenario we also are often constrained with the expenses of enlarging our sample.

Depending on the size of n, the Pr of having a mean of 1.7 will be bigger or smaller. Let's play around with it in code.

```
# Standard Normal CDF
pnorm((1.7 - 2)/(sqrt(2)/sqrt(10)))
```

```
## [1] 0.2511675
```

```
# If n is 10, there is a 25% chance that the mean will be less or equal to 1.7

pnorm((1.7 - 2)/(sqrt(2)/sqrt(100)))
```

```
## [1] 0.01694743
```

```
# If n is 100, there is a 1.6% chance that the sample mean will less or equal to  1.7.
# N is still not large enough

pnorm((1.7 - 2)/(sqrt(2)/sqrt(121)))
```

```
## [1] 0.009812207
```

```
# If n is 121, there is a 0.98% chance that the sample mean will be less or equal to 1.7.
# This is less than 1%!
```

Say I am a researcher and a minister asks me to conduct a study to check if the mean for cars is less or equal to 1.7 people per car. If this is the case, this will trigger a policy response. If I have a small enough sample size, it is more likely that the mean I get from the sample is 1.7. More specifically, if the sample size is 10, there is a 25% chance of it happening, for example. Similarly, it is more likely that the mean is 3, or 4, or other numbers away from the true mean. As a researcher, I will adjust my sample to make sure that the mean is representative of the population mean. As a data scientist, I should ask myself: "How big should my sample be so that a deviation from the sample mean is unlikely?".

If n is 121, there is a 0.98% chance that the sample mean will be less or equal to 1.7. This is less than 1%! So i would make sure my sample has at least 121 cars. The sample size I'd make sure not to go below can also be calculated as follows:

$$P\left(Z \leq x \cdot \frac{(1.7-2)}{\sqrt{2}}\right) = 0.01$$