Math for Data Science: Problem Set 3

Name: Giorgio Coppola, Group 4: Sai Prusni Bandela, Luca-Verona Vellage, Giorgio Coppola

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.)

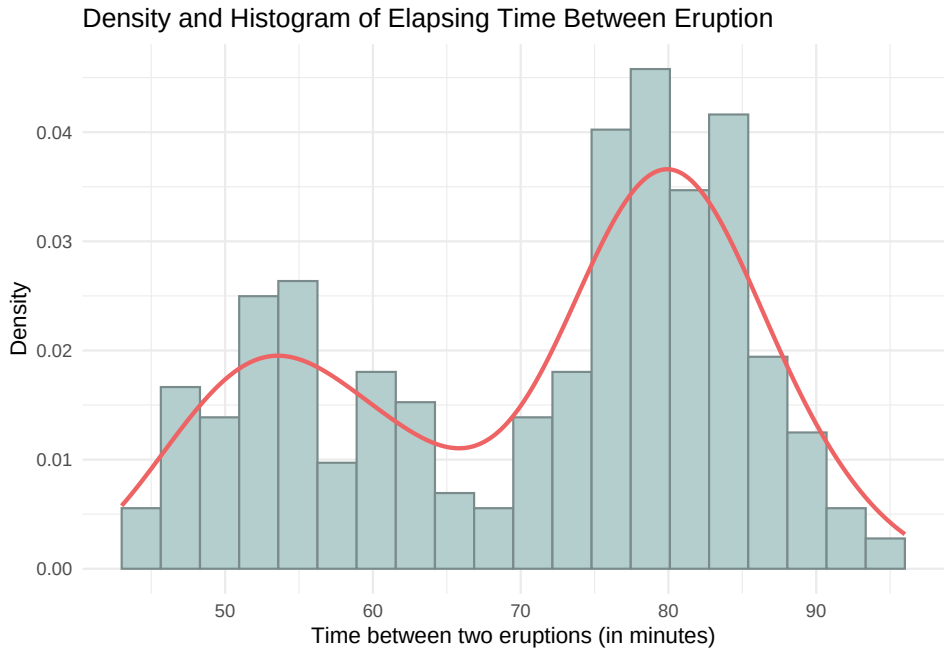
Instructions: Please submit one solution set per person and include your name and your group members' names at the top. This time, please write your solutions within this Rmd file, under the relevant question. Please submit the knitted output as a pdf. Make sure to show all code you used to arrive at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

```
data(faithful)

waiting  = as.matrix(faithful[, 2, drop = FALSE])
eruption  = as.matrix(faithful[, 1, drop = FALSE])

waiting_plot <- ggplot(as.data.frame(waiting), aes(x = waiting)) +
  geom_histogram(aes(y = after_stat(density)),
                 breaks = seq(min(waiting), max(waiting), length.out = 21),
                 col = "lightcyan4", fill = "lightcyan3") +
  geom_density(col = "indianred2", linewidth = 1
               ) +
  labs(x = "Time between two eruptions (in minutes)", y = "Density") +
  theme_minimal() +
  ggtitle("Density and Histogram of Elapsing Time Between Eruption")
```



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.

```r
# prepare the data for the function
X <- waiting
N <- length(X)

# define log.lik helper function
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))))
}

# em function
em_mixture <- function(starting_values, X, tol = .0001, maxits = 100) {

  # initialize convergence to false and iteration number to 0
  converged <- FALSE
  iter <- 0
  N <- length(X)

  # initialize starting values
  pi_1 <- starting_values$pi_1
  pi_2 <- starting_values$pi_2
  mu_1 <- starting_values$mu_1
```

```r
mu_2 <- starting_values$mu_2
var_1 <- starting_values$var_1
var_2 <- starting_values$var_2


# to save vectors
log_likelihoods <- numeric(maxits)
saved_mu_1 <- numeric(maxits)
saved_mu_2 <- numeric(maxits)
saved_pi_1 <- numeric(maxits)
saved_pi_2 <- numeric(maxits)
saved_var_1 <- numeric(maxits)
saved_var_2 <- numeric(maxits)
ll_changes <- numeric(maxits - 1)

while ((!converged) & (iter < maxits)) {

  # 1. evaluate the log likelihood at the initial 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
  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
  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 at the new 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)

  # store new parameters
  log_likelihoods[iter + 1] <- ll.new
  saved_mu_1[iter + 1] <- mu_1
  saved_mu_2[iter + 1] <- mu_2
```

```
  saved_pi_1[iter + 1] <- pi_1
  saved_pi_2[iter + 1] <- pi_2
  saved_var_1[iter + 1] <- var_1
  saved_var_2[iter + 1] <- var_2

  if (iter > 0) {
    ll_changes[iter] <- abs(ll - ll.new) }

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

  # next iteration
  iter <- iter + 1

  # message to keep track of progress
  cat(paste0("Running iteration ", iter,
             ". Log likelihood changed by ", round(abs(ll - ll.new), 4), "\n"))
}

# save the parameter values at the last iteration
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)

 params_df <- data.frame(
               pi_1 = pi_1,
               pi_2 = pi_2,
               mu_1 = mu_1,
               mu_2 = mu_2,
               var_1 = var_1,
               var_2 = var_2) |>
  pivot_longer(
  cols = everything(),
  names_to = c(".value", "group"),
  names_pattern = "(.*)_(.)"
) |>
   as.data.frame()

gammas_df_full <- data.frame(
  waiting_1 = round(gamma_1, 5),
  waiting_2 = round(gamma_2, 5)
)

gammas_df_trimmed <- head(gammas_df_full)


return(list(
  params = params,
```

```
    params_df = params_df,
    log_likelihoods = log_likelihoods[1:iter],
    saved_mu_1 = saved_mu_1[1:iter],
    saved_mu_2 = saved_mu_2[1:iter],
    saved_pi_1 = saved_pi_1[1:iter],
    saved_pi_2 = saved_pi_2[1:iter],
    saved_var_1 = saved_var_1[1:iter],
    saved_var_1 = saved_var_1[1:iter],
    gammas_df_trimmed = gammas_df_trimmed,
    gammas_df_full = gammas_df_full
))
}
```

Output:

```
starting_values_1 <- list(pi_1 = .5, pi_2 = .5, mu_1 = 55, mu_2 = 80, var_1 = 10, var_2 = 10)
em_waiting <- em_mixture(starting_values = starting_values_1, X = waiting)
```

```
## Running iteration 1. Log likelihood changed by 160.5535
## Running iteration 2. Log likelihood changed by 0.2274
## Running iteration 3. Log likelihood changed by 0.0266
## Running iteration 4. Log likelihood changed by 0.0102
## Running iteration 5. Log likelihood changed by 0.0044
## Running iteration 6. Log likelihood changed by 0.0019
## Running iteration 7. Log likelihood changed by 0.0008
## Running iteration 8. Log likelihood changed by 0.0004
## Running iteration 9. Log likelihood changed by 0.0002
## Running iteration 10. Log likelihood changed by 0.0001
```

```
em_waiting$params_df
```

```
##   group        pi       mu      var
## 1     1 0.3610227 54.61941 34.51696
## 2     2 0.6389773 80.09395 34.39662
```

```
em_waiting$gammas_df_trimmed
```

```
##   waiting waiting.1
## 1 0.00011   0.99989
## 2 0.99991   0.00009
## 3 0.00420   0.99580
## 4 0.96774   0.03226
## 5 0.00000   1.00000
## 6 0.99981   0.00019
```

```
em_waiting$saved_mu_1
```

```
##  [1] 54.75426 54.73896 54.70046 54.67148 54.65203 54.63925 54.63087 54.62538
##  [9] 54.62177 54.61941
```

```
em_waiting$saved_mu_2
```

```
##  [1] 80.28271 80.18138 80.14574 80.12646 80.11432 80.10638 80.10115 80.09771
##  [9] 80.09544 80.09395
```

```
em_waiting$log_likelihoods
```

```
##  [1] -1034.274 -1034.046 -1034.020 -1034.009 -1034.005 -1034.003 -1034.002
##  [8] -1034.002 -1034.002 -1034.002
```

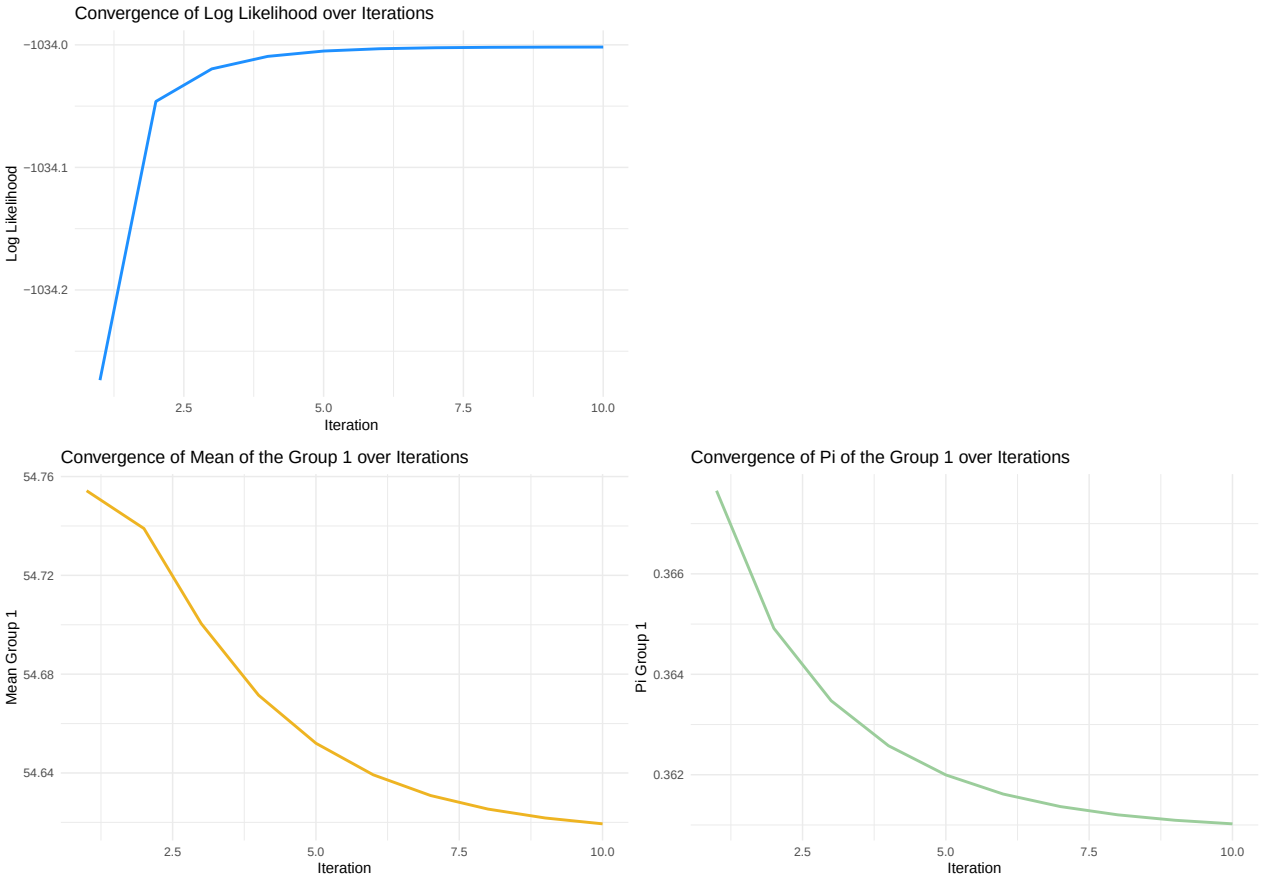
c. (4 points) Generate a plot with the log likelihoods you saved on the y-axis and the iterations of the algorithm on the x-axis. Do the same thing with one or two parameters. Briefly describe and explain what you see.

```
em_waiting_loglik <- em_waiting$log_likelihoods

# plotting log likelihood convergence
em_waiting_loglik_df <-  tibble(
  obs = 1:length(em_waiting_loglik),
  em_log_likelihoods = (em_waiting_loglik)
 )

em_waiting_loglik_plot <- ggplot(em_waiting_loglik_df, aes(x = obs, y = em_log_likelihoods)) +
  geom_line(col = "dodgerblue1", linewidth = 1) +
    labs(x = "Iteration", y = "Log Likelihood", title = "Convergence of Log Likelihood over Iterations")
  theme_minimal()
```

```
em_waiting_mu_1<- em_waiting$saved_mu_1

# plotting log likelihood convergence
em_waiting_mu_1_df <-  tibble(
  obs = 1:length(em_waiting_mu_1),
  em_log_likelihoods = (em_waiting_mu_1)
 )

em_waiting_mu_1_plot <- ggplot(em_waiting_mu_1_df, aes(x = obs, y = em_waiting_mu_1)) +
  geom_line(col = "goldenrod2", linewidth = 1) +
    labs(x = "Iteration", y = "Mean Group 1", title = "Convergence of Mean of the Group 1 over Iteration")
  theme_minimal()
```

```
em_waiting_pi_1 <- em_waiting$saved_pi_1

# plotting log likelihood convergence
```

```r
em_waiting_pi_1_df <-  tibble(
  obs = 1:length(em_waiting_pi_1),
  em_log_likelihoods = (em_waiting_pi_1)
 )

em_waiting_pi_1_plot <- ggplot(em_waiting_pi_1_df, aes(x = obs, y = em_waiting_pi_1)) +
  geom_line(col = "darkseagreen3", linewidth = 1) +
    labs(x = "Iteration", y = "Pi Group 1", title = "Convergence of Pi of the Group 1 over Iterations") +
  theme_minimal()
```



Answer:

Convergence of Log Likelihood over Iterations. The plot helps to answer to the question "how likely is to observe each particular waiting time at each iteration given the adjustment of the parameters and the model as a whole?". Indeed, the convergence of the log likelihood is relative to the whole model with all its parameters, measuring how well the whole model (with its current parameters at each iteration) explains the observed data: as the parameters adjust, the log likelihood increases monotonically. During the first iteration, the log likelihood dramatically increases the precision of the parameters. This improvement effect diminishes progressively in the following iterations (a sort of decreasing marginal returns of improvement from the next iteration). This means that the algorithm was very efficient in the parameters estimation.

Convergence of Mean of the Group 1 over Iterations. The mean of the group 1 converges to around 54.62 at the 10th iteration, decreasing monotonically at each iteration. In this case, the adjustment is more progressive. Indeed, as in the case of the log likelihood convergence, each iteration progressively diminish its adjustment impact on the mean value. However, in this case, the first iteration does not have a dramatic impact to the adjustment as happened before.

Convergence of Pi of the Group 1 over Iterations. Remember that pi is the mixture coefficient of a Gaussian Mixture Model, indicating the overall probability of being in one of the two clusters (groups). More precisely, it is the proportion of each component in the overall GMM. Indeed, pi is a value relative to the entire data set, and not the probability for each data point of the GMM is generated by one of the component of mixture. This latter indeed is the gamma (responsibility that each component has for each distribution in the GMM). From this plot we can see that the pi is adjusted in a similar way compared to the other two parameters.

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.

```
# perform em
eruption  = as.matrix(faithful[, 1, drop = FALSE])
starting_values_erupt <- list(pi_1 = .5, pi_2 = .5, mu_1 = 2, mu_2 = 4, var_1 = 1, var_2 = 1)
em_erupt <- em_mixture(starting_values = starting_values_erupt, X = eruption)
```

```
## Running iteration 1. Log likelihood changed by 59.2056
## Running iteration 2. Log likelihood changed by 61.1015
## Running iteration 3. Log likelihood changed by 28.8845
## Running iteration 4. Log likelihood changed by 4.8385
## Running iteration 5. Log likelihood changed by 0.8025
## Running iteration 6. Log likelihood changed by 0.2967
## Running iteration 7. Log likelihood changed by 0.1382
## Running iteration 8. Log likelihood changed by 0.0637
## Running iteration 9. Log likelihood changed by 0.0274
## Running iteration 10. Log likelihood changed by 0.011
## Running iteration 11. Log likelihood changed by 0.0042
## Running iteration 12. Log likelihood changed by 0.0016
## Running iteration 13. Log likelihood changed by 0.0006
## Running iteration 14. Log likelihood changed by 0.0002
## Running iteration 15. Log likelihood changed by 0.0001
```

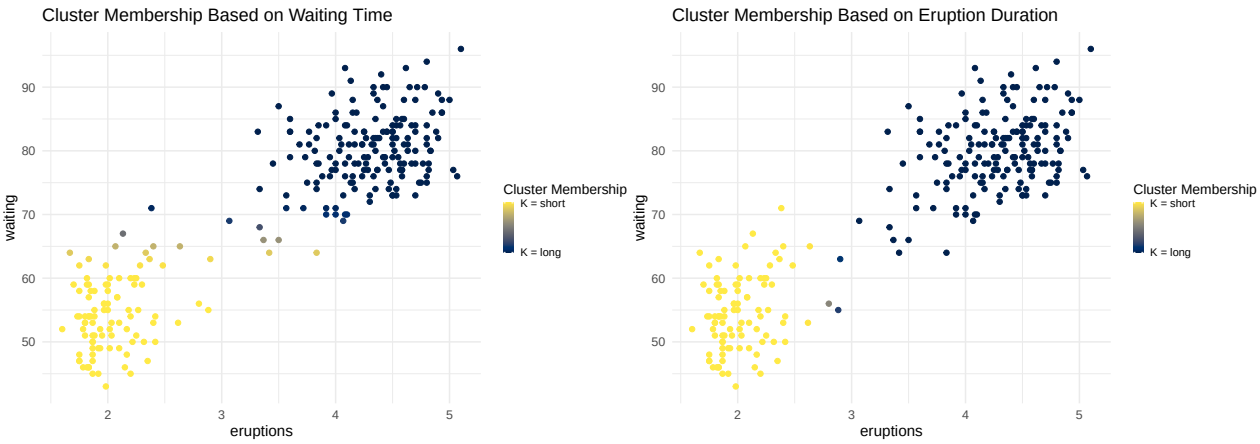
```
# get cluster membership for em_erupt and em_waiting
em_erupt_gammas <- em_erupt$gammas_df_full |>
  rename(k_short_er = eruptions,
         k_long_er = eruptions.1)

em_waiting_gammas <- em_waiting$gammas_df_full |>
  rename(k_short_wt = waiting,
         k_long_wt = waiting.1)

# compute correlations
cor_k1 <- cor(em_erupt_gammas$k_short_er, em_waiting_gammas$k_short_wt)
cor_k2 <- cor(em_erupt_gammas$k_long_er, em_waiting_gammas$k_long_wt)
```

```r
# print
print(c(cor_k1, cor_k2))
```

```
## [1] 0.9554499 0.9554499
```

The correlation (`r`) between the gammas for belonging to the "short" cluster (k = 1) based on the `eruption` variable and those based on the `waiting` variable is highly positive. It is evident that the `r` for the two gammas from the two variables for being in k = 1 is the same as the `r` between the gammas for belonging to the "long" cluster (k = 2).

```r
# prepare data for plot
faithful_plot_data <- cbind(faithful, em_erupt_gammas, em_waiting_gammas)

# plot cluster membership
# plot waiting time clustering
plor_corr_wt <- ggplot(faithful_plot_data, aes(x = eruptions, y = waiting)) +
  geom_point(aes(color = k_short_wt)) +
  scale_color_viridis(option = "cividis", discrete = F,
                      name = "Cluster Membership",
                      breaks = c(0, 1),
                      labels = c("K = long", "K = short")) +
  theme_minimal() +
  ggtitle("Cluster Membership Based on Waiting Time") +
   theme(legend.title = element_text(size = 10),
         legend.text = element_text(size = 8),
         legend.key.size = unit(0.5, "lines"))

# plot eruptions clustering
plor_corr_er <- ggplot(faithful_plot_data, aes(x = eruptions, y = waiting)) +
  geom_point(aes(color = k_short_er)) +
  scale_color_viridis(option = "cividis", discrete = F,
                      name = "Cluster Membership",
                      breaks = c(0, 1),
                      labels = c("K = long", "K = short")) +
  theme_minimal() +
  ggtitle("Cluster Membership Based on Eruption Duration") +
   theme(legend.title = element_text(size = 10),
         legend.text = element_text(size = 8),
         legend.key.size = unit(0.5, "lines"))
```



Answer: The color differentiation is horizontal when based on waiting time (indeed based on values on the y-axis), and vertical when based on eruption duration (indeed based on values on the x-axes). We can notice that the two clustering methods give more or less the same result. Indeed, there is a correlation of approximately 0.95 between being classified on the basis of `eruption` duration and being classified on the basis of `waiting`. This high positive correlation indicates a strong linear relationship between the two variables in their influence on the clustering outcomes. We could use K-means clustering to classify more precisely the data we have, based on the available parameters. Intuitively, one can think that if the correlation was perfect (r = 1), the two clustering outcome, based on eruption or based on waiting time, would be the same.

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.

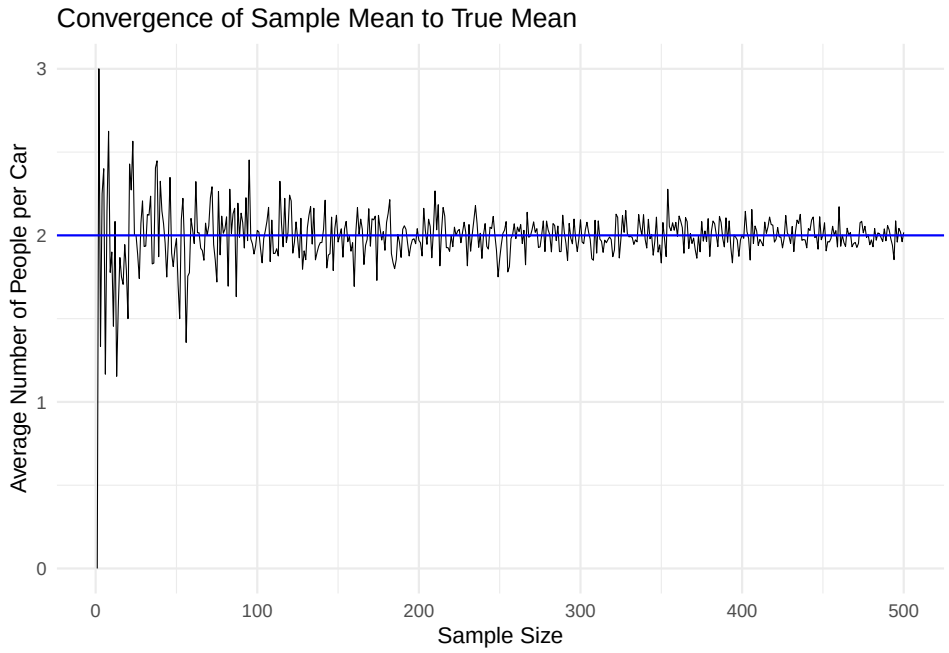
Collecting data on people per car is costly and you'd love to minimize how many data points you have to collect. However, you're also familiar with the Law of Large Numbers and know that the sample mean converges to the true mean as the sample size n grows large.

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 lambda and sample_means vector
lambda <- 2
sample_means <- numeric(500)

# Loop to generate samples and compute means
for (n in 1:500) {
  sample <- rpois(n, lambda)
  sample_means[n] <- mean(sample)
}
```

[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.

```r
# Data frame for plotting
samples <- tibble(sample_size = 1:500, mean = sample_means)

# Plotting
plot_samples <- ggplot(samples, aes(x = sample_size, y = mean)) +
  geom_line(linewidth = 0.25)  +
  geom_hline(yintercept = lambda, color = "blue", linewidth = 0.5) +
  theme_minimal() +
  labs(title = "Convergence of Sample Mean to True Mean",
       x = "Sample Size",
       y = "Average Number of People per Car")
```



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.

Answer: The Central Limit Theorem states that for a sufficiently large sample size, the distribution of sample means will approximate a normal distribution. This holds true regardless of the shape of the population distribution, provided the sample size is large enough. By the Law of Large Numbers, we know that as n grows large, the sample mean X-bar converges to the true mean . For a Poisson distribution, the standardized sample mean converges in distribution to the standard Normal already at a n = 30.

To calculate the average number of people per car in a sample of n = 100 cars, let's first define the parameters:

- Mean: $\lambda = 2$ people per car.
- Standard Deviation: For a Poisson distribution, the standard deviation is equal to the square root of lambda, namely $\sigma = \sqrt{2}$ Remember that the standard deviation is relative to the individual observations.
- Standard Error: The standard error of the sample mean is $\sigma/\sqrt{n}$, therefore $\sqrt{2}/\sqrt{100} = \sqrt{2}/10$.

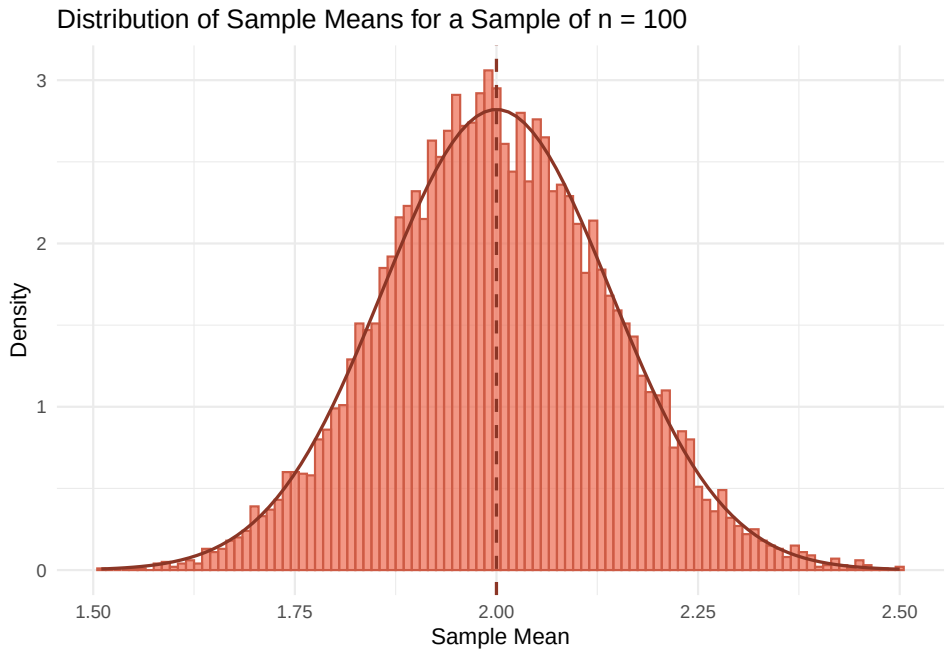
The approximate distribution of the average number of people per car in a sample of 100 cars is a normal distribution with a mean of 2 and a standard error of $\sqrt{2}/10$.

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.

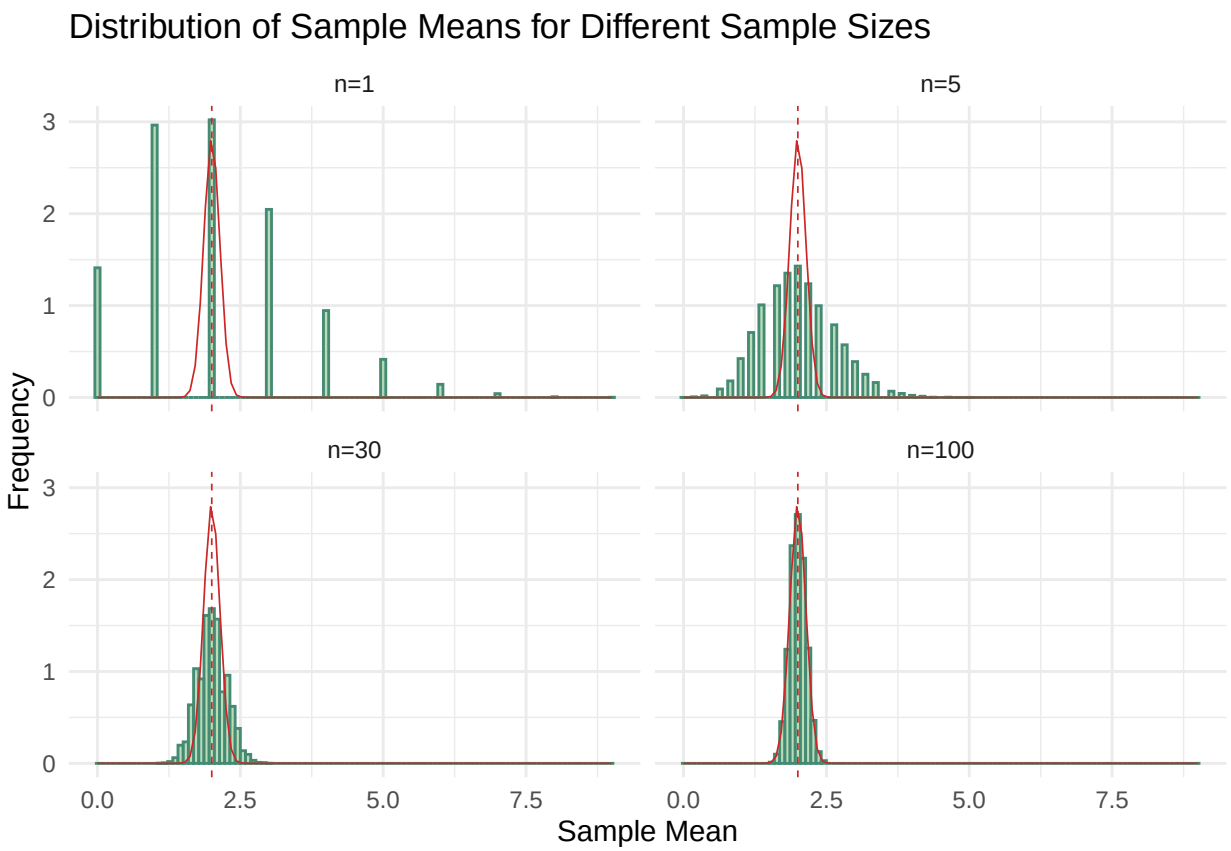
```r
set.seed(11)

# Function to generate replicates of sample means
generate_replicates <- function(n, num_replicates) {
  replicate(num_replicates, mean(rpois(n, lambda)))
}

# Generate replicates for different sample sizes
replicates_n1 <- generate_replicates(1, 10000)
replicates_n5 <- generate_replicates(5, 10000)
replicates_n30 <- generate_replicates(30, 10000)
replicates_n100 <- generate_replicates(100, 10000)

replicates_n100_df <- as.data.frame(replicates_n100)

# Plotting means distribution of the sample n = 100
n_100_plot <- ggplot(replicates_n100_df, aes(x = replicates_n100)) +
  geom_histogram(aes(y = ..density..), bins = 100, fill = 'coral2', color = "coral3",alpha = 0.7) +
  stat_function(fun = dnorm, args = list(mean = 2, sd = (sqrt(2 / 100))), color = "tomato4", linewidth =
  geom_vline(xintercept = 2, color = "tomato4", linetype = "dashed", linewidth = 0.75) +
  xlab("Sample Mean") +
  ylab("Density") +
  ggtitle("Distribution of Sample Means for a Sample of n = 100")+
  theme_minimal()

# Combine data for plotting
replicates <- data.frame(
  sample_size = factor(rep(c('n=1', 'n=5', 'n=30', 'n=100'), each=10000)),
  mean = c(replicates_n1, replicates_n5, replicates_n30, replicates_n100)
)

# Reorder the factor levels
replicates$sample_size <- factor(replicates$sample_size, levels = c('n=1', 'n=5', 'n=30', 'n=100'))

# Plotting facets
replicates_plot <- ggplot(replicates, aes(x = mean)) +
  geom_histogram(aes(y = ..density..), bins = 100, fill = 'darkseagreen3', color = "aquamarine4",alpha =
  stat_function(fun = dnorm, args = list(mean = 2, sd = (sqrt(2 / 100))), color = "firebrick3", linewid
  geom_vline(xintercept = 2, color = "firebrick3", linetype = "dashed", linewidth = 0.3) +
  facet_wrap(~sample_size, scales = 'fixed') +
  theme_minimal() +
  labs(title = "Distribution of Sample Means for Different Sample Sizes",
       x = "Sample Mean",
       y = "Frequency")
```

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

Distribution of Sample Means for a Sample of n = 100:



Distribution of Sample Means for Different Sample Sizes



Answer:

Already before plotting the histograms, one would expect that the one for n = 100 would show a pretty

stable mean, therefore more or less similar in each iteration, and quite close to the mean value (with a small standard deviation). If we see large differences in the means at sample of n = 100), it would mean that the Poisson does not converge to one specific normal distribution of the data.

Now observing the histograms, we are convinced that the Normal approximation you found in the previous question is good enough. The histograms visually demonstrate the Central Limit Theorem in action. As the sample size increases, the distribution of the sample means becomes progressively more normal, regardless of the non-normal shape of the underlying population distribution .The histogram for the sample size of 1 shows a highly discrete distribution that reflects the underlying Poisson distribution. Notice that, focusing on the Poisson distribution (sample with n = 1), the iteration shows that probabilistically we would encounter approximately 1300 self-driving cars with zero passenger on board (even if we are in San Francisco, this is quite unrealistic indeed). The histogram for n = 30 shows a much smoother distribution, and while it might not be perfectly normal, it's clear that the distribution of the sample mean is starting to exhibit the bell-shaped curve that is characteristic of a normal distribution. The histogram for the largest sample size shows a stable, approximately normal distribution around the true mean ($\lambda = 2$). The shape is symmetric, and the variance is consistent with the expected behavior given the properties of the Poisson distribution and the sample size.

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%?

```r
# Given values
lambda <- 2
n <- 100
x_bar <- 1.7

# Standard deviation for a Poisson distribution is the square root of lambda
sigma <- sqrt(lambda)

# Calculate the standard error of the sample mean
standard_error <- sigma / sqrt(n)

# Calculate the z-score for a sample mean of 1.7
z_score <- (x_bar - lambda) / standard_error

# Calculate the probability of getting a sample mean of 1.7 or less
probability <- pnorm(z_score)

# To keep the probability of a Type I error below 1%, find the z-score that corresponds to 99% of the n
z_critical <- qnorm(0.01)

# Calculate the required sample size
# The sample size formula is derived from the standard error formula rearranged for n
required_sample_size <- (sigma / (lambda - x_bar) * z_critical) ^ 2

# Output the probability and required sample size (rounded up to the nearest whole number)
list(probability = probability, required_sample_size = ceiling(required_sample_size))
```

```
## $probability
## [1] 0.01694743
```

```
##
## $required_sample_size
## [1] 121
```

Answer:

With a true population mean of 2 for the number of people per car, the standard deviation of the sample mean can be calculated by dividing the standard deviation of the Poisson distribution, which is the square root of , by the square root of the sample size (n). Consequently, to determine the likelihood of obtaining a sample mean of 1.7 or less, we compute the z-score. This z-score is then used to find the corresponding probability with a standard normal distribution.

The probability of erroneously deducing that the mean number of people per car during rush hour is below 1.7, when it is actually 2, is found to be approximately 1.69%. To decrease this probability to less than 1% – thereby reduce the risk of a Type I error – it is necessary to increase the sample size. Our calculations suggest that a minimum sample size of 121 cars is required to achieve this more stringent error threshold. This adjustment ensures a narrower distribution of the sample mean, aligning with regulatory standards and statistical precision.