Math for Data Science: Problem Set 3

Name: Ray Hossain, Group: GROUP_MEMBERS_NAMES

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

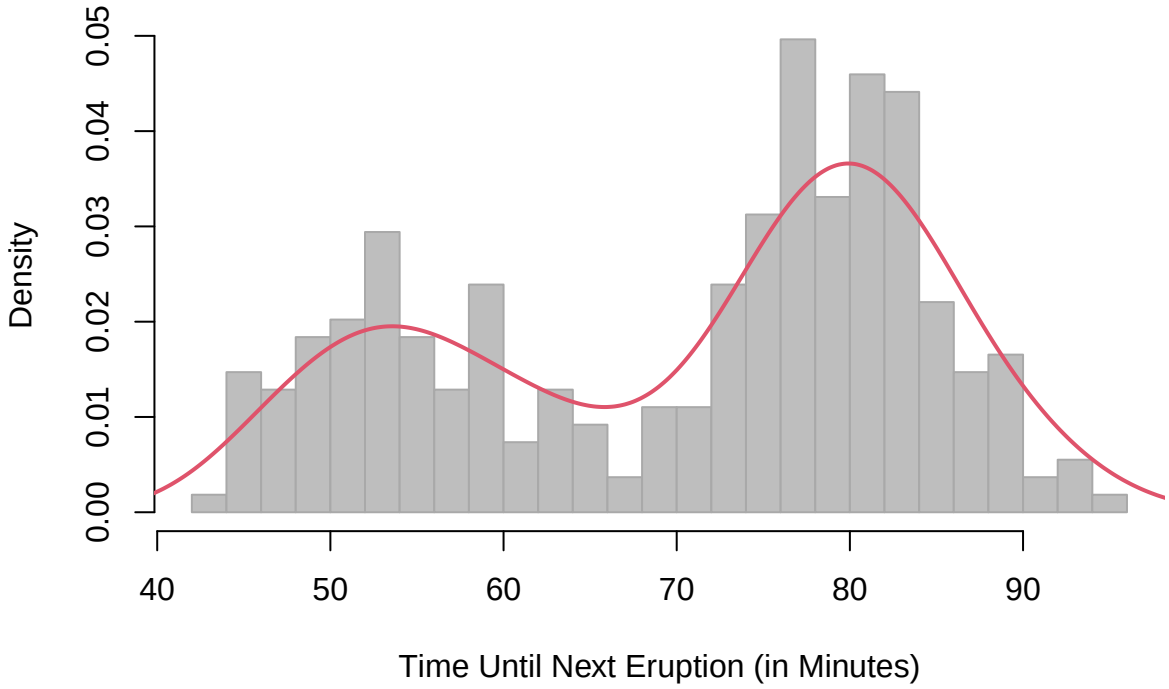
(20 points) Let's continue with our Old Faithful example from Lab 7. Remember that we had two pieces of information about the geyser: eruption duration, which we worked with, and also the wait time until next eruption.

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)
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(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))))
}

##--- CREATE A FUNCTION TO GET EM
em_mixture <- function(starting_values, X, tol = .0001, maxits = 100) {

  # initialize convergence to false and iteration number to 0
  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

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)

  # 5. Check convergence
  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"))
  }

  # When finished, save the parameter values
  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)
  return(params)
}

waiting_values_1 <- list(pi_1 = .5, pi_2 = .5, mu_1 = 52, mu_2 = 80, var_1 = 10, var_2 = 10)

em_waiting <- em_mixture(starting_values = waiting_values_1, X = waiting)
```

```
## Running iteration 1. Log likelihood changed by 194.3501
## Running iteration 2. Log likelihood changed by 0.1398
## Running iteration 3. Log likelihood changed by 0.0038
## Running iteration 4. Log likelihood changed by 7e-04
## Running iteration 5. Log likelihood changed by 3e-04
## Running iteration 6. Log likelihood changed by 1e-04
## Running iteration 7. Log likelihood changed by 1e-04
```

```
em_waiting
```

```
## $pi_1
## [1] 0.3607605
##
## $pi_2
## [1] 0.6392395
##
## $mu_1
## [1] 54.61068
##
## $mu_2
## [1] 80.08842
##
## $var_1
## [1] 34.42934
##
## $var_2
## [1] 34.46138
```

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.

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.

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

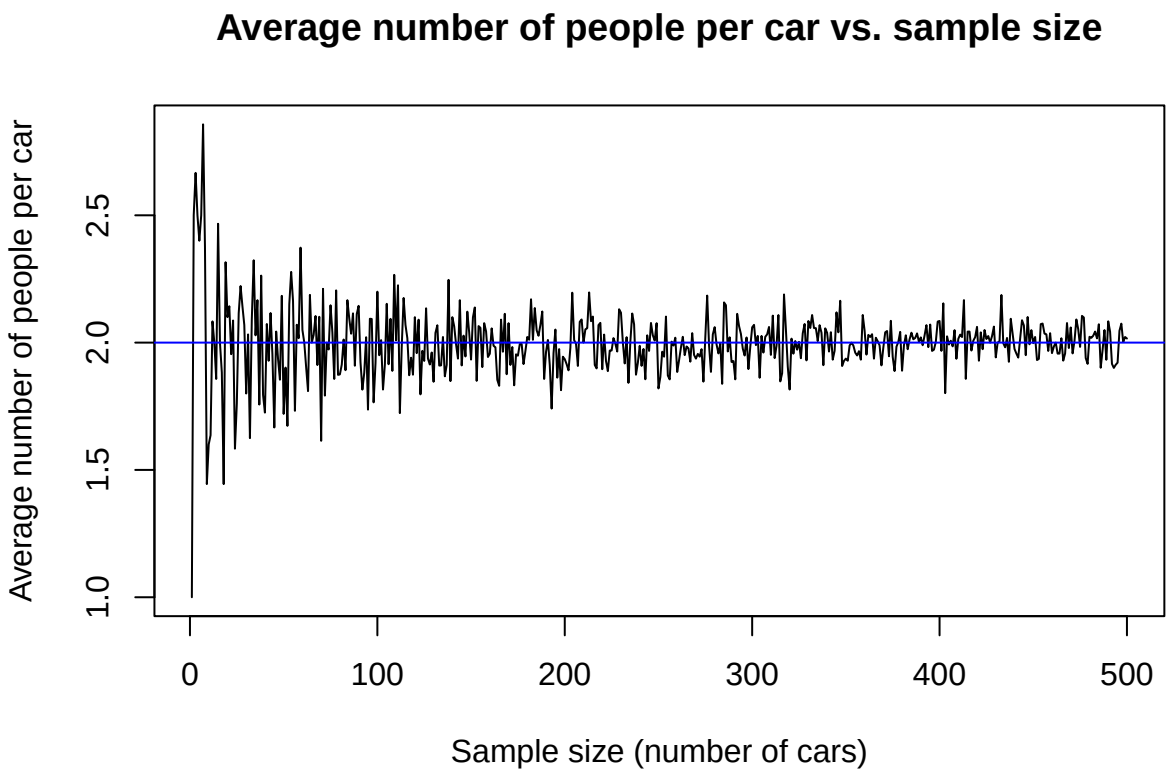
(20 points) You are an urban planner interested in finding out how many people enter and leave the city using personal vehicles every day. (You're not interested in the number of *cars*; you're interested in the number of *people* who use cars to get to work.) To do this, you decide to collect data from 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 (number of cars)", ylab = "Average numb
abline(h = lambda, col = "blue")
```

[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.



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.

```
set.seed(123)

hundred_mean <- mean(rpois(100, 2))

hundred_sd <- sd(rpois(100, 2))

approx_sd <- hundred_sd / sqrt(length(rpois(100, 2)))

cat("Sample mean:", hundred_mean, "\n")
```

```
## Sample mean: 2.02
```

```
cat("Sample standard deviation:", hundred_sd, "\n")
```

```
## Sample standard deviation: 1.309021
```

```
cat("CLT standard deviation:", approx_sd, "\n")
```

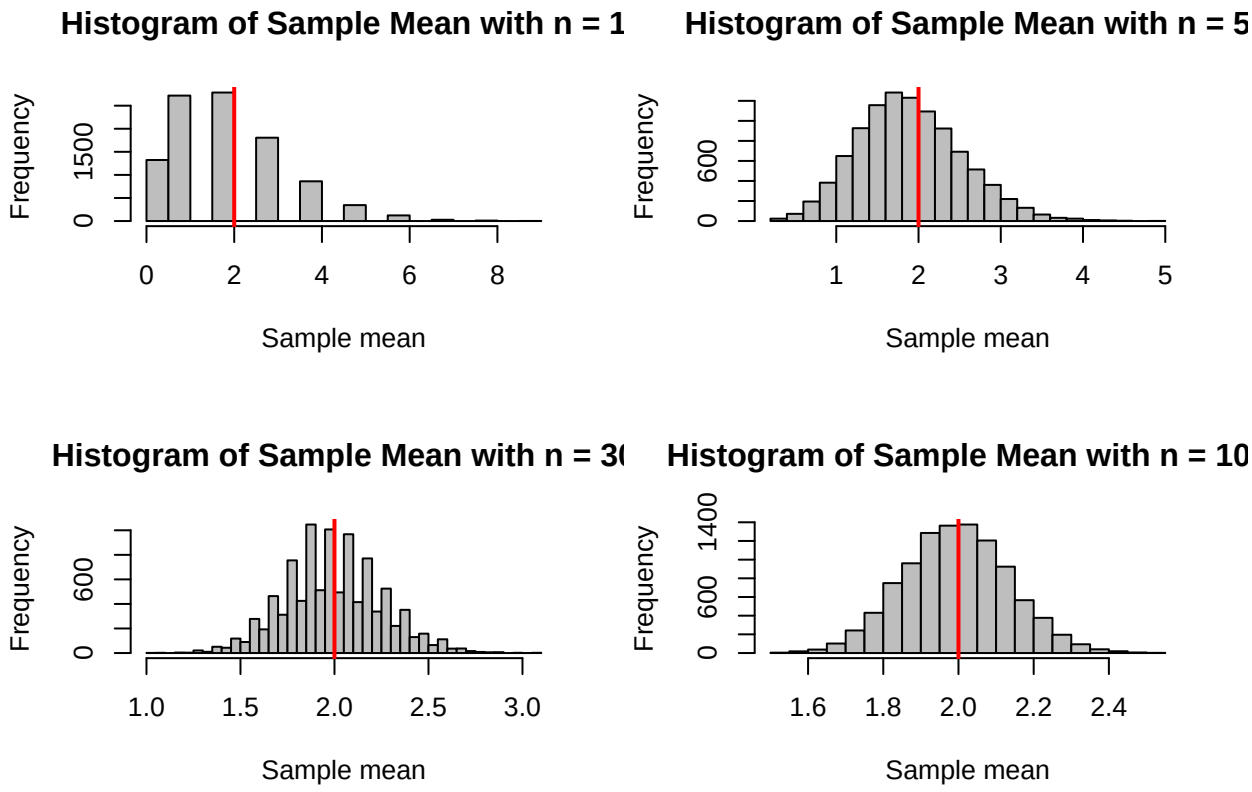
```
## CLT standard deviation: 0.1309021
```

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

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

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 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 = 30, col = "gray", main = paste0("Histogram of Sample Mean with n = ", n[i]),

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



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%?

```
z_score <- (1.7 - 2) / approx_sd

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 0.0109589
```

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

x_hat <- ( qnorm(0.01) * approx_sd ) + 2

cat("To ensure the probability stays below 1%, you need to make sure the mean number of cars is", x_hat
```

```
## To ensure the probability stays below 1%, you need to make sure the mean number of cars is 1.695476
```