Q1:

The following distribution has the density function:

$$f(x;\theta) = \frac{1-\cos(x-\theta)}{2\pi}, \quad 0 \le x \le 2\pi, \quad -\pi < \theta < \pi$$

For an observed random sample $x_1, x_2, \ldots, x_n$ from this distribution, the log likelihood is seen to be:

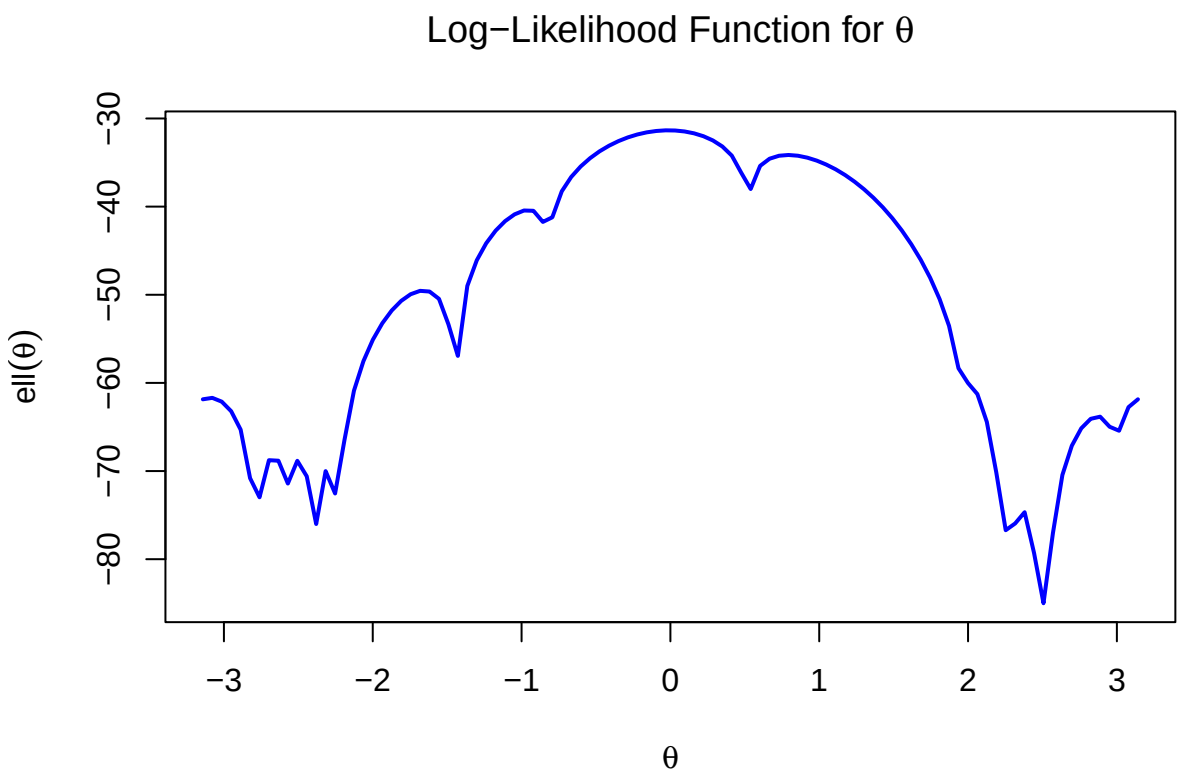
$$\ell(\theta) = -n \log 2\pi + \sum_{i=1}^{n} \log\{1-\cos(x_i-\theta)\}$$

Suppose that $(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96, 2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2.99, 2.54, 0.52, 2.50)$ is an observed random sample from the above distribution.

(a)

Plot the log likelihood $\ell(\theta)$ in the range $-\pi < \theta < \pi$.

```r
# Given observed sample data
x <- c(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96,
       2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2.99, 2.54, 0.52, 2.50)

# Define the log-likelihood function
log_likelihood <- function(theta, x) {
  n <- length(x)
  -n * log(2 * pi) + sum(log(1 - cos(x - theta)))
}

# Generate values of theta over the range -pi to pi
theta_values <- seq(-pi, pi, length.out = 100)
log_lik_values <- sapply(theta_values, log_likelihood, x = x)

# Plot the log-likelihood function
plot(theta_values, log_lik_values, type = "l", lwd = 2, col = "blue",
     main = expression("Log-Likelihood Function for " * theta),
     xlab = expression(theta), ylab = expression(ell(theta)))
```



(b)

Use the R function `optimize()` to find the maximum likelihood estimate of θ.

```r
# Define the log-likelihood function
log_likelihood <- function(theta, x) {
  n <- length(x)
  -n * log(2 * pi) + sum(log(1 - cos(x - theta)))
}

# Find the maximum likelihood estimate of theta using optimize
mle_result <- optimize(log_likelihood, interval = c(-pi, pi), x = x, maximum = TRUE)

# Display the MLE of theta and the corresponding log-likelihood value
theta_mle <- mle_result$maximum
log_lik_mle <- mle_result$objective

cat("The maximum likelihood estimate of theta is:", theta_mle, "\n")
```

```
## The maximum likelihood estimate of theta is: -0.0119724
```

```r
cat("The maximum log-likelihood value is:", log_lik_mle, "\n")
```

```
## The maximum log-likelihood value is: -31.34291
```

(c)

Use function `newton()` in the class handout to find the maximum likelihood estimate of θ by solving $\ell'(\theta) = 0$ using the starting value of $\theta_{(0)} = 0$.

```
# Given observed sample data
x <- c(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96,
       2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2.99, 2.54, 0.52, 2.50)

# Define the first derivative of the log-likelihood function (ell')
log_likelihood_prime <- function(theta) {
  -sum(sin(x - theta) / (1 - cos(x - theta)))
}

# Define the second derivative of the log-likelihood function (ell'')
log_likelihood_double_prime <- function(theta) {
  -sum(1 / (1 - cos(x - theta)))
}

# Initial value for theta
theta_start <- 0
# Tolerance level
tolerance <- 1e-6

# Provided newton function from the image
newton <- function(fun, derf, x0, eps) {
  iter <- 0
  repeat {
    iter <- iter + 1
    x1 <- x0 - fun(x0) / derf(x0)
    if (abs(x0 - x1) < eps || abs(fun(x1)) < 1e-10)
      break
    x0 <- x1
    cat("****** Iter. No:", iter, " Current Iterate =", x1, "\n", fill = TRUE)
  }
  return(x1)
}

# Use the newton function to find the MLE of theta
theta_mle <- newton(log_likelihood_prime, log_likelihood_double_prime, theta_start, tolerance)
```

```
## ****** Iter. No: 1  Current Iterate = -0.01191193
##
## ****** Iter. No: 2  Current Iterate = -0.011972
```

```
cat("The maximum likelihood estimate of theta using Newton's method is:", theta_mle, "\n")
```

```
## The maximum likelihood estimate of theta using Newton's method is: -0.011972
```

(d)

What happens if you use $\theta_{(0)} = -2.0$ and -2.7, respectively, as starting values? Explain why.

```r
# Test with starting value theta_start = -2.0
cat("Starting with theta_start = -2.0\n")
```

```
## Starting with theta_start = -2.0
```

```r
theta_mle_2_0 <- newton(log_likelihood_prime, log_likelihood_double_prime, -2.0, tolerance)
```

```
## ****** Iter. No: 1  Current Iterate = -1.756154
##
## ****** Iter. No: 2  Current Iterate = -1.641367
##
## ****** Iter. No: 3  Current Iterate = -1.657301
##
## ****** Iter. No: 4  Current Iterate = -1.65828
##
## ****** Iter. No: 5  Current Iterate = -1.658283
```

```r
cat("The MLE of theta with theta_start = -2.0 is:", theta_mle_2_0, "\n\n")
```

```
## The MLE of theta with theta_start = -2.0 is: -1.658283
```

```r
# Test with starting value theta_start = -2.7
cat("Starting with theta_start = -2.7\n")
```

```
## Starting with theta_start = -2.7
```

```r
theta_mle_2_7 <- newton(log_likelihood_prime, log_likelihood_double_prime, -2.7, tolerance)
```

```
## ****** Iter. No: 1  Current Iterate = -2.674114
##
## ****** Iter. No: 2  Current Iterate = -2.666794
##
## ****** Iter. No: 3  Current Iterate = -2.6667
```

```r
cat("The MLE of theta with theta_start = -2.7 is:", theta_mle_2_7, "\n")
```

```
## The MLE of theta with theta_start = -2.7 is: -2.6667
```

Convergence: If these points are near enough to the true maximum or within a region where Newton's method successfully iterates toward the maximum.

Divergence or Slow Convergence: If these starting points are far from a peak or in a flat region, Newton's method might fail to converge or take many iterations to do so.

Turn in the plot, any functions you write, function calls, and the results.

P.S. To help you out with the necessary derivatives, I derived them below, but you need to check them out!

$$\frac{\partial \ell}{\partial \theta} = -\sum_{i=1}^{n} \frac{\sin(x_i - \theta)}{1 - \cos(x_i - \theta)}$$

$$\frac{\partial^2 \ell}{\partial \theta^2} = -\sum_{i=1}^{n} \frac{1}{1 - \cos(x_i - \theta)}$$

Q2:

Regression to the mean. Consider the following very simple genetic model in which a population consists of equal numbers of two sexes: male and female. At each generation, men and women are paired at random, and each pair produces exactly two offspring, one male and one female. We are interested in the distribution of height from one generation to the next. Supposing that the height of both children is just the average of the heights of their parents, how will the distribution of height change across generations?

(a)

Represent the heights of the current generation as a dataframe with two variables, M and F for the two sexes. Randomly generate a population at generation 1, as for males with $X_1, X_2, \ldots, X_{100} \sim N(125, 25^2)$, and for females $X_1, X_2, \ldots, X_{100} \sim N(125, 15^2)$.

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

# Generate heights for males and females in generation 1
male_heights <- rnorm(100, mean = 125, sd = 25)
female_heights <- rnorm(100, mean = 125, sd = 15)

# Create a dataframe to represent the current generation
generation1 <- data.frame(M = male_heights, F = female_heights)

# Display the first few rows of the dataframe
head(generation1)
```

```
##          M        F
## 1 110.9881 114.3439
## 2 119.2456 128.8533
## 3 163.9677 121.2996
## 4 126.7627 119.7869
## 5 128.2322 110.7257
## 6 167.8766 124.3246
```

(b)

Take the dataframe from (a) and randomly permute the ordering of men. Men and women are then paired according to rows, and heights for the next generation are calculated by taking the mean of each row. The function should return a dataframe with the same structure, giving the heights of the next generation. You

will need to use the `sample(x, size = n)` function to return a random sample of size n from the vector x. You will also need to use the `apply()` function.

```
# Function to generate the next generation's heights
generate_next_generation <- function(df) {
  # Randomly permute the order of male heights
  permuted_males <- sample(df$M, size = nrow(df))

  # Create a new dataframe with permuted males and original females
  next_generation <- data.frame(M = permuted_males, F = df$F)

  # Calculate the mean height of each pair (male, female) for the next generation
  next_generation$Mean_Height <- apply(next_generation, 1, function(row) mean(row))

  # Keep only the Mean_Height column and rename it to M and F to match the structure of df
  next_generation <- data.frame(M = next_generation$Mean_Height, F = next_generation$Mean_Height)

  return(next_generation)
}

# Example usage
next_gen <- generate_next_generation(generation1)
head(next_gen)
```

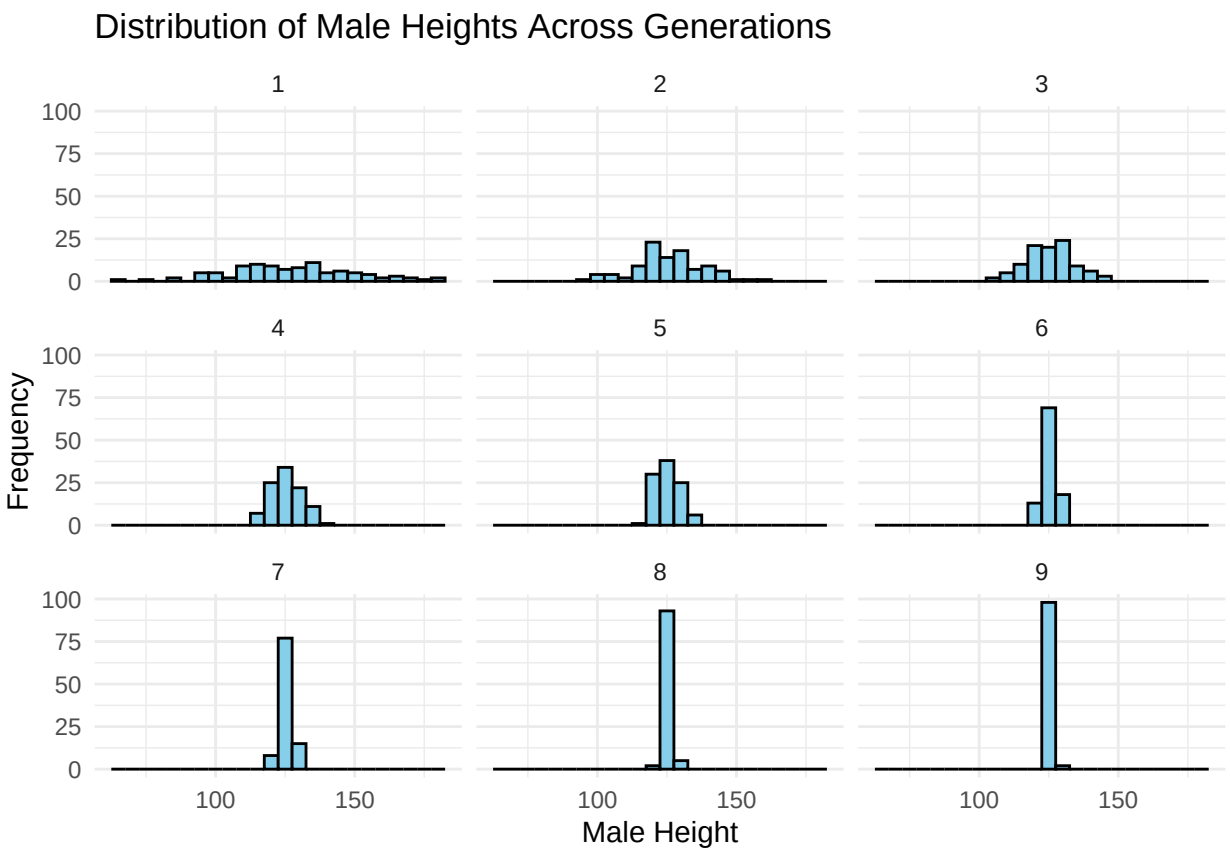
```
##          M        F
## 1 106.8469 106.8469
## 2 132.5293 132.5293
## 3 114.5642 114.5642
## 4 127.1389 127.1389
## 5 114.0383 114.0383
## 6 116.8135 116.8135
```

(c)

Use the above function to generate nine generations, then use `ggplot2` to facet histograms to plot the distribution of male heights in each generation. This is called regression to the mean. *(Hint: Instead of using `facet_grid`, you will need to use `facet_wrap(, nrow = 3)`, in order to create 3 × 3 histograms.)*

```
# Load ggplot2
library(ggplot2)

# Function to generate multiple generations
generate_multiple_generations <- function(df, num_generations) {
  generations <- list()
  generations[[1]] <- df$M  # Store the male heights of the first generation

  # Loop to generate successive generations
  for (i in 2:num_generations) {
    df <- generate_next_generation(df)
    generations[[i]] <- df$M  # Store the male heights of each generation
  }

  # Convert the list to a data frame for plotting
```

```r
  generation_data <- data.frame(
    Generation = rep(1:num_generations, each = nrow(df)),
    Male_Height = unlist(generations)
  )

  return(generation_data)
}

# Generate data for nine generations
num_generations <- 9
generation_data <- generate_multiple_generations(generation1, num_generations)

# Plot the male heights for each generation using ggplot2
ggplot(generation_data, aes(x = Male_Height)) +
  geom_histogram(binwidth = 5, color = "black", fill = "skyblue") +
  facet_wrap(~ Generation, nrow = 3) +
  labs(
    title = "Distribution of Male Heights Across Generations",
    x = "Male Height",
    y = "Frequency"
  ) +
  theme_minimal()
```



Q3:

Clustering Output Parsing

The `mmclustering` program available at http://math.univ-lille1.fr/~wicker/softwares.html (but *note*: you do not need to download or run any such program) provides a partitioning of observations into different groups. However, the output is provided in a form which makes it difficult to easily do further analysis in R. In this exercise, we will write a function that will take the output (from a given file) and write out the classification for each observation as a vector.

The files provided in `Iris1.out` and `Iris2.out` contain results from grouping the iris dataset into a certain number of categories. The file has the following lines:

- The first line in the file contains the number of clusters, indicated by the phrase **Number of clusters** followed by a space, a colon `:`, a space, and then an integer (indicating the number of clusters). For example:

  `Number of clusters : 4`

- The second line is a blank.
- The third line contains the id of the first cluster (always designated by 0) and starts with **Cluster** followed by a space, then `0`, followed by a space, semi-colon `;`, the word **size** followed by an equality sign `=` and an integer which denotes the size of the cluster. For example:

  `Cluster 0; size=37`

- The next number of lines (which should match the cluster size given in the third line) contain the observation indices that belong to that group. This group of indices ends with a blank line.
- The following lines repeat the same format for the second cluster (indicated by **Cluster 1**), and this process continues until all cluster memberships are listed.

(a)

The objective here is to write a function which will read one of the files above and provide a vector of length equal to the sum of the cluster sizes, containing the group indicators of the observations in the total population.

To do this, we can use the `readLines` function to read the file line-by-line (each line will be read in as a character string). Then, we can use multiple string-matching methods to parse the first line to obtain the number of clusters. The second line is a blank line. The first line after each blank line contains the size of the cluster (with memberships following in the next lines). Again, we will use string-splitting and matching techniques (e.g., `strsplit`) to obtain the cluster size. The next lines are converted from character strings to integers. Write the above function.

```
# Improved function with debugging output to parse the clustering output file
parse_cluster_file <- function(filename) {
  # Read the file line-by-line
  lines <- readLines(filename)

  # Check if the first line contains the number of clusters
  if (grepl("Number of clusters", lines[1])) {
    num_clusters <- as.numeric(strsplit(lines[1], ": ")[[1]][2])
    cat("Number of clusters:", num_clusters, "\n")
  } else {
    stop("The file format is incorrect: 'Number of clusters' not found in the first line.")
```

```r
  }

  # Initialize an empty vector for cluster assignments
  cluster_vector <- numeric()

  # Start reading from the third line (after the blank line)
  line_index <- 3

  # Loop through each cluster ID
  for (cluster_id in 0:(num_clusters - 1)) {
    # Debug output to inspect the line being processed
    cat("Processing line", line_index, ":", lines[line_index], "\n")

    # Check if the current line contains cluster size information
    if (grepl("Cluster", lines[line_index])) {
      # Try extracting cluster size using a more flexible method
      cluster_info <- unlist(strsplit(lines[line_index], "[;= ]"))
      cluster_size <- as.numeric(cluster_info[which(cluster_info == "size") + 1])

      # Ensure cluster_size was parsed correctly
      if (is.na(cluster_size)) {
        stop("Failed to parse cluster size on line ", line_index, ": ", lines[line_index])
      }

      # Attempt to extract observation indices for the current cluster
      observation_lines <- lines[(line_index + 1):(line_index + cluster_size)]
      observation_indices <- as.numeric(observation_lines)

      # Check for any NAs in observation indices and handle them
      if (any(is.na(observation_indices))) {
        warning("NAs introduced by coercion when parsing observation indices.")
        observation_indices <- observation_indices[!is.na(observation_indices)]
      }

      # Assign cluster ID to these indices in the cluster vector
      cluster_vector[observation_indices] <- cluster_id

      # Move to the next cluster's line (skip the blank line after the current cluster)
      line_index <- line_index + cluster_size + 2
    } else {
      stop("Expected cluster information on line ", line_index, " but not found.")
    }
  }

  return(cluster_vector)
}

# Example usage
# cluster_vector_1 <- parse_cluster_file("Iris1.out")
# print(cluster_vector_1)
```

(b)

Cross-tabulate the results of a call to the above function on the file `Iris1.out` and the file `Iris2.out`.

```
# Parse the clustering results from each file
# Apply the function to Iris1.out
cluster_vector_1 <- parse_cluster_file("Iris1.out")
```

```
## Number of clusters: 3
## Processing line 3 : Cluster 0 ; size=37
## Processing line 42 : Cluster 1 ; size=50
## Processing line 94 : Cluster 2 ; size=63
```

```
print(cluster_vector_1)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0 0
## [112] 0 2 0 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0 0
## [149] 2
```

```
cluster_vector_2 <- parse_cluster_file("Iris2.out")
```

```
## Number of clusters: 2
## Processing line 3 : Cluster 0 ; size=100
## Processing line 105 : Cluster 1 ; size=50
```

```
print(cluster_vector_2)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [149] 0
```

```
# Cross-tabulate the two cluster assignments
cluster_table <- table(cluster_vector_1, cluster_vector_2)

# Display the cross-tabulation table
print(cluster_table)
```

```
##                 cluster_vector_2
## cluster_vector_1  0  1
##                0 37  0
##                1  0 49
##                2 63  0
```

Q4:

Multivariate Normal Sampling and Standardization

Let $X \sim N_3(\mu, \Sigma)$, where $\mu = \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)'$ and Σ is a 3×3 diagonal matrix with diagonal elements 1 and off-diagonal elements ρ. Consider standardizing X to get $Y = \frac{X}{\|X\|}$.

(Estimating parameters in the general scenario, for large p, and factorial structure of Σ where we specify $\Sigma = \Lambda\Lambda' + \Psi$, with the basics first introduced in Stat 5010, was part of the Fall 2020 dissertation of Fan Dai from Iowa State University.)

(a)

Write a function in R which generates a sample of Y of size n as outlined above.

```r
# Function to generate a sample of Y
generate_Y_sample <- function(n, rho) {
  # Define mean vector and covariance matrix
  mu <- rep(1 / sqrt(3), 3)
  Sigma <- matrix(rho, nrow = 3, ncol = 3)
  diag(Sigma) <- 1  # Set diagonal elements to 1

  # Generate multivariate normal samples for X
  library(MASS)
  X <- mvrnorm(n, mu = mu, Sigma = Sigma)

  # Standardize X to get Y
  Y <- t(apply(X, 1, function(x) x / sqrt(sum(x^2))))

  return(Y)
}

# Example usage
set.seed(123)
Y_sample <- generate_Y_sample(n = 100, rho = 0.5)
head(Y_sample)
```

```
##            [,1]       [,2]        [,3]
## [1,]  0.9500807  0.3114513  0.01856557
## [2,]  0.9353646  0.1584436  0.31620998
## [3,] -0.6890529 -0.4023832 -0.60273858
## [4,]  0.8258153  0.5318643  0.18748200
## [5,]  0.2117427  0.9716430  0.10523732
## [6,] -0.7494310 -0.4525435 -0.48327801
```

(b)

For different $\rho \in \{-0.5, -0.25, 0, 0.25, 0.5\}$, display the observations in three dimensions. You may use $n = 100$ as the sample size in each case. Comment.

```r
# Load necessary libraries
library(MASS)
library(scatterplot3d)

# Function to generate and plot Y for different rho values
plot_Y_for_rho <- function(n, rho_values) {
  par(mfrow = c(3, 2))  # Arrange plots in a 3x2 grid
  for (rho in rho_values) {
    # Generate the sample of Y
    Y <- generate_Y_sample(n, rho)

    # Plot in 3D
    scatterplot3d(Y[,1], Y[,2], Y[,3], main = paste("rho =", rho),
                  xlab = "Y1", ylab = "Y2", zlab = "Y3")
  }
}

# Define rho values and sample size
rho_values <- c(-0.5, -0.25, 0, 0.25, 0.5)
n <- 100

# Generate and plot samples for each rho
plot_Y_for_rho(n, rho_values)
```