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:

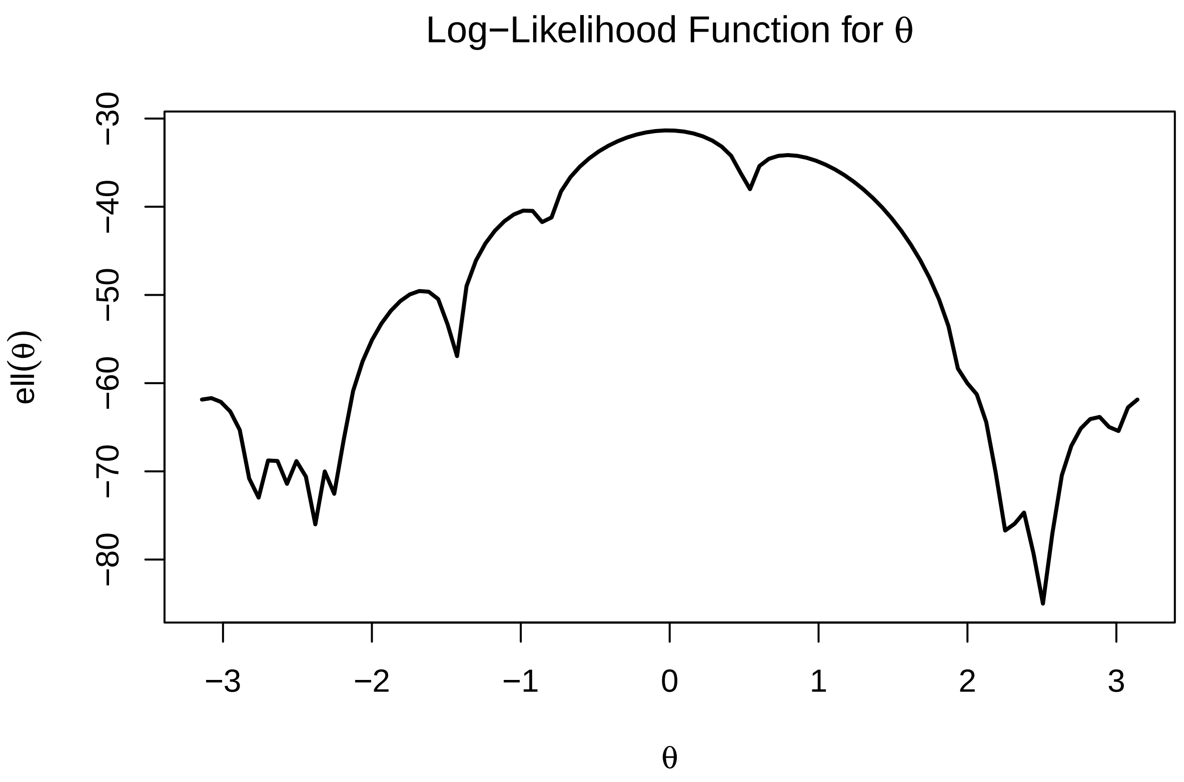
$$\ell(\theta) = -n\log 2\pi + \sum_{i=1}^{n} \log\{1-\cos(x_i-\theta)\}$$

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
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)

logLike <- function(theta, x) {
  n <- length(x)
  -n * log(2 * pi) + sum(log(1 - cos(x - theta)))
}

thetaVal <- seq(from = -pi,
                to = pi,
                length.out = 100)

logLikeVal <- sapply(X = thetaVal,
                     FUN = logLike,
                     x = x)

plot(x = thetaVal,
     y = logLikeVal,
     type = "l",
     lwd = 2,
     col = "black",
     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.

```
mleEst <- optimize(f = logLike,
                   interval = c(-pi, pi),
                   x = x,
                   maximum = TRUE)

theta_mle <- mleEst$maximum
log_lik_mle <- mleEst$objective

# adding messages like the newton function
# pretty nifty, yeah?
cat(paste0("The maximum likelihood estimate of theta is:", theta_mle, "\n"))
```

```
## The maximum likelihood estimate of theta is:-0.0119724034926742
```

```
cat(paste0("The maximum log-likelihood value is:", log_lik_mle, "\n"))
```

```
## The maximum log-likelihood value is:-31.3429125904982
```

(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 $\theta_{(0)} = 0$.

```r
logLikePrime <- function(theta) {
  -sum(sin(x - theta) / (1 - cos(x - theta)))
}

logLikeDP <- function(theta) {
  -sum(1 / (1 - cos(x - theta)))
}

init <- 0
tolerance <- 1e-6

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
    # me-ow, again
    cat("****** Iter. No:", iter, " Current Iterate =", x1, "\n", fill = TRUE)
  }
  return(x1)
}

theta_mle <- newton(fun = logLikePrime,
                    derf = logLikeDP,
                    x0 = init,
                    eps = tolerance)
```

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

```r
cat(paste0("The maximum likelihood estimate is: ", theta_mle))
```

```
## The maximum likelihood estimate is: -0.0119720022874401
```

(d)

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

```r
# x0 = -2.0
theta_mle_2_0 <- newton(fun = logLikePrime,
                        derf = logLikeDP,
                        x0 = -2.0,
                        eps = 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
```

```
# Cheeky lil line break, eh?
cat(paste0("The MLE of theta with initial value -2.0 is: ", theta_mle_2_0, "\n"))
```

```
## The MLE of theta with initial value -2.0 is: -1.65828322990256
```

```
# x0= -2.7
theta_mle_2_7 <- newton(fun = logLikePrime,
                        derf = logLikeDP,
                        x0 = -2.7,
                        eps = tolerance)
```

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

```
cat(paste0("The MLE of theta with initial value -2.7 is:", theta_mle_2_7))
```

```
## The MLE of theta with initial value -2.7 is:-2.66669992610095
```

In both instances we achieve convergence to a local maximum. This happens because the method used doesn't look over a large enough range of values, i.e. it finds a sufficient area for searching finds the "peaks" of those areas, not always arriving at the global maximum. This is because the `newton` function as-written, but also somewhat generally, encounters this issue, as we are able to satisfy convergence criteria without guanrenteeing we found the global maximum over the entire range of values being considered. (I think this is an illustration of why we sometimes add random noise to optimization procedures, as a way to "escape" local max or mins.)

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(43)

maleHeights <- rnorm(n = 100,
                     mean = 125,
                     sd = 25)
femaleHeights <- rnorm(n = 100,
                       mean = 125,
                       sd = 15)

gen1 <- data.frame(M = maleHeights,
                   F = femaleHeights)
```

(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.

```r
genNextGen <- function(df) {
  permM <- sample(df$M, size = nrow(df))

  nextGen <- data.frame(M = permM, F = df$F)
  nextGen$Mean_Height <- apply(X = nextGen,
                               MARGIN = 1,
                                FUN = function(row) {mean(row)}
                                )
  # update
  nextGen <- data.frame(M = nextGen$Mean_Height, F = nextGen$Mean_Height)
  nextGen
}

# start with gen1
nextGen <- genNextGen(df = gen1)
head(nextGen)
```

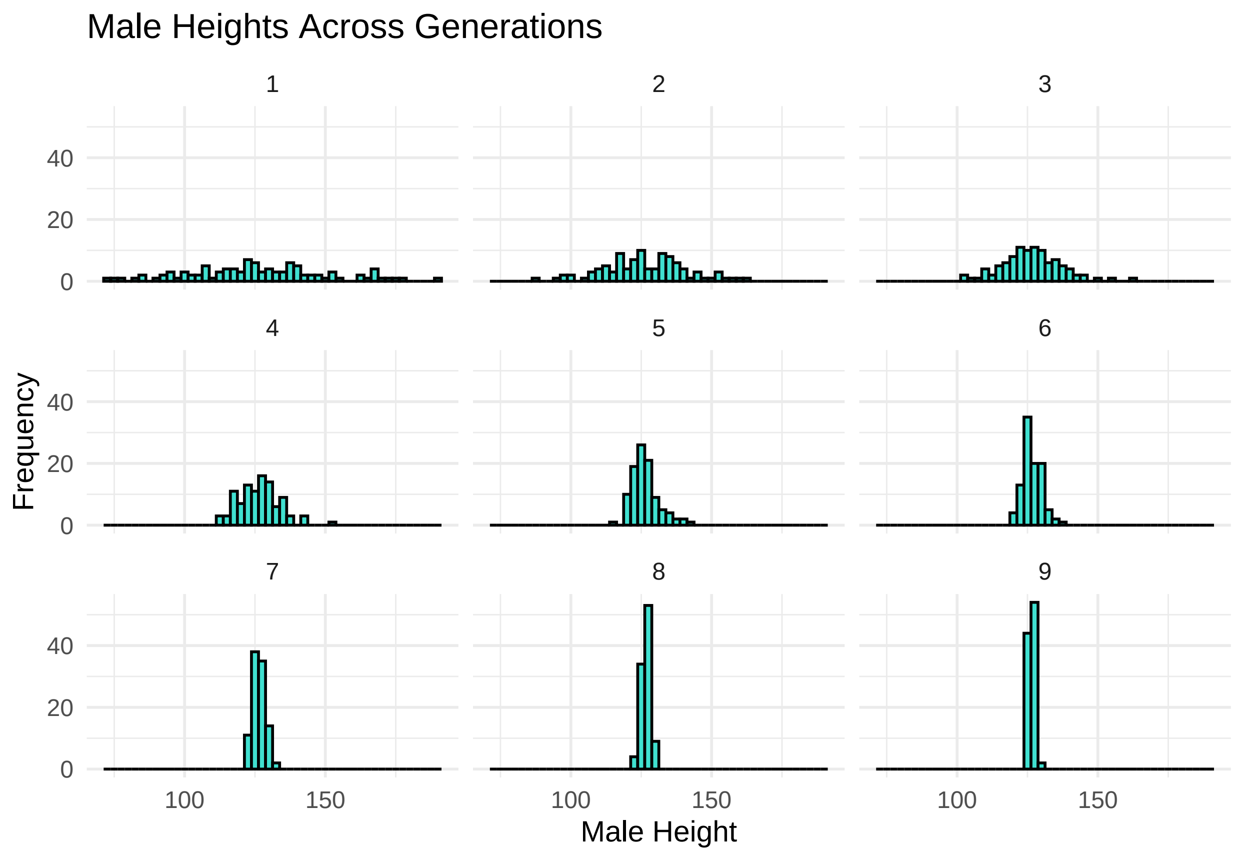
```
##          M        F
## 1 139.5480 139.5480
## 2 126.8168 126.8168
## 3 128.4576 128.4576
## 4 120.1116 120.1116
## 5 142.6801 142.6801
## 6 128.3991 128.3991
```

(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 $\times$ 3 histograms.)*

```
# me-ow
library(purrr)

genMGen <- function(df, nGen) {
  generations <- map(1:nGen, function(i) {
    if (i > 1) {
      # update
      df <<- genNextGen(df)
    }
    df$M
  })

  genDat <- data.frame(
    Generation = rep(1:nGen, each = nrow(df)),
    Male_Height = unlist(generations)
  )

  genDat
}

# iterate until we reach 9th generation
# FUN FACT WE ARE ON THE 9TH GENERATION OF POKEMON
# COINCIDENCE!?!?
nGenData <- genMGen(df = gen1, nGen = 9)
```

```r
library(ggplot2)

ggplot(data = nGenData,
       aes(x = Male_Height)) +
  # you wouldn't believe how to spell turquoise, would you?
  # I got it wrong like twice before checking with the ol' Google
  geom_histogram(binwidth = 2.5, color = "black", fill = "turquoise") +
  facet_wrap(~ Generation, nrow = 3) +
  labs(
    title = "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.

Notes:

- Iris1: 37, 50, 63
- Iris2: 100, 50
- Want rows, columns to add up to above dimensions

```r
# for loop
# "BAD" version
parseClusterFileLoop <- function(file) {
  allLines <- readLines(file)
  groups <- cumsum(allLines == "")
  splitVec <- split(allLines[allLines != ""], groups[allLines != ""])
  clusterAssignment <- list()

  for (i in seq_along(splitVec)) {
    elements <- splitVec[[i]]
    elements <- elements[!grepl("[A-Za-z]", elements)]
    elements <- as.numeric(elements)

    if (length(elements) == 0) next
```

```r
      clusterAssignment[[i]] <- data.frame(
      observation = elements,
      cluster = i - 1
    )
  }

  result <- do.call(rbind, clusterAssignment)
  result <- result[order(result$observation), "cluster"]
  return(result)
}


# non for loop
# requires global assignment
parseClusterFileGlobal <- function(file) {
  allLines <- readLines(file)
  groups <- cumsum(allLines == "")
  splitVec <- split(allLines[allLines != ""], groups[allLines != ""])

  # GLOBAL
  clusterAssignment <<- rep(NA, length(allLines))

  # tbh this used to be a crazy for loop operation (who would've guessed?)
  # but then I saw the "monstrocity" Craig made
  # and this is loosely adapted off that
  lapply(seq_along(splitVec), function(i) {
    elements <- splitVec[[i]]
    elements <- elements[!grepl("[A-Za-z]", elements)]
    elements <- as.numeric(elements)

    # and yes, this breaks when I don't do a global assignment
    clusterAssignment[elements] <<- i - 1
  })

  as.vector(na.omit(clusterAssignment))
}

# non for loop
# no global assignment
parseClusterFile <- function(file) {
  allLines <- readLines(file)
  groups <- cumsum(allLines == "")
  splitVec <- split(allLines[allLines != ""], groups[allLines != ""])

  clusterAssignments <- unlist(
    mapply(function(group, index) {
      elements <- as.numeric(group[!grepl("[A-Za-z]", group)])
      setNames(rep(index - 1, length(elements)), elements)
    },
    splitVec, seq_along(splitVec))
  )

  clusterAssignments <- clusterAssignments[order(as.numeric(names(clusterAssignments)))]
  return(as.vector(clusterAssignments))
}
```

An Alternative Method?

Issues with alternative:

- Pre-initializes/loads in the data, then processes it
- Assumes fixed positions (are Iris 1 and Iris 2 structured exactly the same?)

```r
iris1<- readLines("Iris1.out")
iris2<- readLines("Iris2.out")

# Ethan's version
# weirdFn<- function(iris_i) {
#   num_clusters<- as.numeric(sub(pattern = "Number of clusters : *",
#   replacement = "", x = iris_i[1]))
#   blanks<- which(nchar(iris_i)==0)
#   cluster_sizes<- as.numeric(sub(pattern = "Cluster . ; size=*",
#   replacement = "", x = iris_i[(blanks[1:num_clusters]+1)]))
#   group_indicators<-c()
#
#   for (i in 1:(length(blanks)-1)) {
#     x<- rep(i-1,cluster_sizes[i])
#     group_indicators<- c(group_indicators,x)
#   }
#   return(group_indicators)
# }

weirdFn <- function(iris_i) {
  num_clusters <- as.numeric(sub(
    pattern = "Number of clusters : *",
    replacement = "",
    x = iris_i[1]
  ))

  blanks <- which(nchar(x = iris_i) == 0)
  cluster_sizes <- as.numeric(sub(
    pattern = "Cluster . ; size=*",
    replacement = "",
    x = iris_i[(blanks[1:num_clusters] + 1)]
  ))

  group_indicators <- unlist(
    mapply(
      FUN = function(index, size) rep(x = index - 1, length.out = size),
      seq_along(cluster_sizes),
      cluster_sizes
    )
  )

  return(group_indicators)
}

## weird, this gives slightly different answers than my "main" method
# iris1Out <- weirdFn(iris1)
# iris2Out <- weirdFn(iris2)
#
# table(iris1Out, iris2Out)
```

A Message for Gautham

```
string_list <- c(
  "Gautham", "I", "beg", "of", "you,",
  "have", "mercy", "upon", "me", "for", "committing", "the", "grave", "sin", "of", "writing", "a", "for
  "forgive", "me", "my", "sins", "of", "not", "believing", "us", "to", "be", "rational,",
  "and", "forgive", "me", "most", "of", "all", "for", "not", "having", "yet", "learned", "how", "to", "
)

for (word in string_list) {
  print(word)
}
```

(b)

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

```
Iris1 <- parseClusterFile("Iris1.out")
Iris1
```

```
##   [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 1
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 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 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

```
Iris2 <- parseClusterFile("Iris2.out")
Iris2
```

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

```
irisTable <- table(Iris1, Iris2)
irisTable
```

```
##      Iris2
## Iris1  1  2
##     1 37  0
##     2 50  0
##     3 13 50
```

The above method using `parseClusterFile` corresponds to the output given by "Ethan's method". This is such a headache. Whew!

Alternate Methods

This alternative methods provides slightly different answers. I feel not great about that, but for the purposes of grading I'm just including these for comparison. I stand by the above method and its output in terms of which one I "choose" as an answer.

```
# global assignment method
Iris1 <- parseClusterFileGlobal("Iris1.out")
Iris1
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1 1
## [112] 1 3 1 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 1 1 1 1 3 1 1 1 3 1 1 1 3 1 1
## [149] 3
```

```
Iris2 <- parseClusterFileGlobal("Iris2.out")
Iris2
```

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

```
irisTable <- table(Iris1, Iris2)
irisTable
```

```
##      Iris2
## Iris1  1  2
##     1 37  0
##     2  0 49
##     3 63  0
```

```
# for loop method
Iris1 <- parseClusterFileLoop("Iris1.out")
Iris1
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1
## [112] 1 1 3 1 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 1 1 1 1 3 1 1 1 3 1 1 1 3 1
## [149] 1 3
```

```
Iris2 <- parseClusterFileLoop("Iris2.out")
Iris2
```

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

```
irisTable <- table(Iris1, Iris2)
irisTable
```

```
##      Iris2
## Iris1  1  2
##     1 37  0
##     2  0 50
##     3 63  0
```

```
# Ethan's method
iris1Out <- weirdFn(iris1)
iris2Out <- weirdFn(iris2)

iris1Out
```

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

```
iris2Out
```

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

```
table(iris1Out, iris2Out)
```

```
##         iris2Out
## iris1Out  0  1
##        0 37  0
##        1 50  0
##        2 13 50
```

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.)

Gautham Commentary:

My understanding of Q4.a) is that:

- using just X doesn't answer the question
- to get one sample of Y, you would need one sample of X (a multivariate normal sample, which can be any 3D value), and normalize it so it ends up on the surface of the unit sphere.
- For n samples of Y, we can probably write an apply with a custom function, use the rowNorm/sweep code given below, or equivalents

Q4.b) then asks how the sampled data looks visually. How does the covariance matrix affect distribution on the sphere? Using just X, you don't get data on a sphere, so 4.b) becomes harder to answer.

(a)

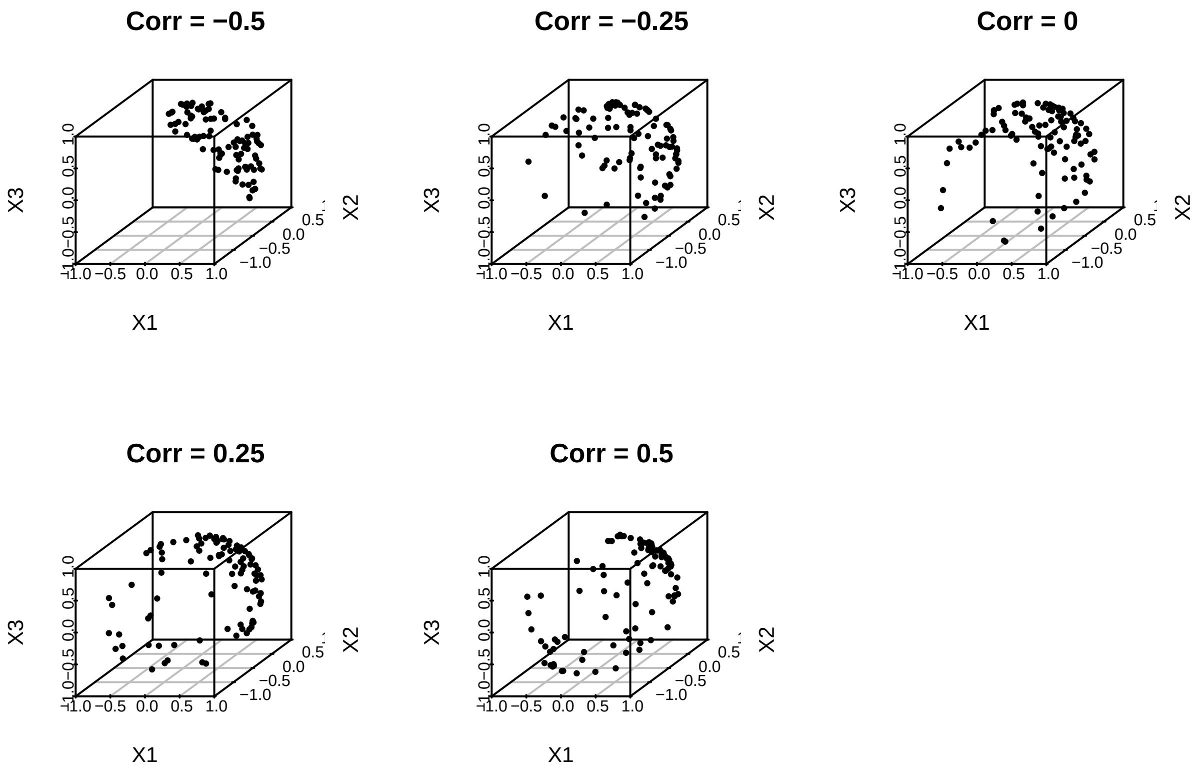
Write a function in R which generates a sample of Y of size n as outlined above.

```r
library(MASS)
mu <- c(1 / sqrt(3), 1 / sqrt(3), 1 / sqrt(3))
generate_Y <- function(n, mu, Sigma) {
  X <- MASS::mvrnorm(n, mu = mu, Sigma = Sigma)

  # still unsure if this normalizing/standardizing is right
  row_norms <- sqrt(rowSums(X^2))
  Y <- sweep(X, MARGIN = 1, STATS = row_norms, FUN = "/")
  Y
}
```

(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
# library(rgl)
# library(knitr)
# library(rmarkdown)
# mainly use this scatterplot3d
# b/c the rgl stuff wouldn't play well with knitting together
library(scatterplot3d)

rho_values <- c(-0.5, -0.25, 0, 0.25, 0.5)
n <- 100

par(mfrow = c(2, 3))
set.seed(43)

# for loop BAD, very BAD
# no more for loops
# for loop BAD
# for (rho in rho_values) {
#   Sigma <- matrix(rho, nrow = 3, ncol = 3)
#   diag(Sigma) <- 1
#   Y <- generate_Y(n, mu, Sigma)
#     scatterplot3d::scatterplot3d(x =Y[, 1],
#                                y = Y[, 2],
#                                z = Y[, 3],
#                                color = "black",
#                                pch = 16,
#                                main = paste("Corr =", rho),
#                                xlab = "X1",
#                                ylab = "X2",
#                                zlab = "X3")
# }

plot_Rho <- function(rho) {
  Sigma <- matrix(rho, nrow = 3, ncol = 3)
  diag(Sigma) <- 1
  Y <- generate_Y(n, mu, Sigma)
  scatterplot3d::scatterplot3d(x = Y[, 1],
                               y = Y[, 2],
                               z = Y[, 3],
                               color = "black",
                               pch = 16,
                               main = paste("Corr =", rho),
                               xlab = "X1",
                               ylab = "X2",
                               zlab = "X3")
}

par(mfrow = c(2, 3))

# need this or you'll get like 10 extra pages of nonsense of my R console smh
invisible(lapply(X = rho_values,
                 FUN = plot_Rho))
```



Smaller (or Zero) correlation (Rho): There's more dispersion (across variables/ across dimensions) over the sphere when Rho has a smaller magnitude (-.25, 0, and .25).

Positive correlation (larger Rho): Positive Rho values mean positive correlation (across variables/ across dimensions), causing observations to cluster more tightly together (lower dispersion, at times appearing more like a diagonal line across the surface of the sphere).

Negative correlation (negative Rho): Negative Rho values mean negative correlation (across variables/ across dimensions). This pushes the observations toward opposite quadrants (quadrants of a sphere? anyways, I mean we see things more like the pairs (+x, -y, +z) or (-x, +y, -z)).

Zero correlation (Rho): When Rho is zero then the variables are uncorrelated, resulting in observations that spread more uniformly on the sphere, without strong clustering in any particular direction.

Alternative, incorrect (I believe) code given below, for reference. Thank you again, Gautham, for clarifying your understanding of this question.

```r
# REFERENCE, NOT MY ANSWER
# no sphere, no beers, amiright
library(MASS)
library(scatterplot3d)

# alternative method
sampleYs<- function(n,rho) {
  Sigma<- matrix(c(1,rho,rho,rho,1,rho,rho,rho,1),nrow=3,ncol=3)
  X<-mvrnorm(n,mu,Sigma)
  Y<- X/norm(X)
  return(Y)
}

# rho, rho, rho your boat
rho_values <- c(-0.5, -0.25, 0, 0.25, 0.5)
par(mfrow = c(2, 3))

invisible(lapply(X = rho_values,
                 FUN = function(rho) {
  Y <- sampleYs(100, rho)
  scatterplot3d::scatterplot3d(x = Y[, 1],
                               y = Y[, 2],
                               z= Y[, 3],
                               color = "black",
                               pch = 16,
                               main = paste("Corr =", rho),
                               xlab = "X1",
                               ylab = "X2",
                               zlab = "X3")
}))
```