STAT243 - Problem Set 8

Cristobal Pais - cpaismz@berkeley.edu

December 1st, 2017

Problem 1: Importance Sampling

The main objective of this problem consists of comparing the usefulness of different distributions when performing an importance sampling estimation depending on the shape of the tail associated with each distribution. As we know from theory, the method needs a distribution with a heavy tail as the sampling density in order to work properly. Therefore, we will test this assumption by comparing the importance sampling when an exponential and Pareto distributions are used as sampling distributions.

We want to estimate $\phi = \mathbf{E}(X)$ and $\phi = \mathbf{E}(X^2)$ with respect to a density f.

First, we will compare both distributions (exponential and Pareto) in order to visualize their tails and conclude based on our knowledge of the distributions and relevant plots which one has a heavier tail and thus, fits better as the sampling distribution for the importance sampling approach. Then, we will empirically compare the results obtained when each distribution plays this fundamental role during the estimation of ϕ.

a) Pareto vs Exponential distributions

In order to determine which tail decay more quickly between both distributions, we simply plot a series of graphs where we can compare the different distribution functions for some combinations of their relevant parameters: (1) λ rate in the case of the exponential and (2) α (location) and β (shape) in the case of the Pareto distribution.

1. First, we define two auxiliary functions that will allow us to create simple plots for both distributions. The fist one will plot the density of an exponential distribution while the second one will address the Pareto distribution. A simple data frame is created with the values obtained from the densities after generating a large x vector with values between 0 to 20.

   Both functions return the plot object.

```
plotExp <- function(Lambda){
  # Create the dataframe with the density to plot
  x <- seq(0, 20, length.out=1000)
  dat <- data.frame(x = x, px = dexp(x, rate = Lambda))

  # Create the plot using visual enhancements
  p <- ggplot(dat, aes(x = x,y = px)) +
        theme(axis.text.x = element_text(angle = 0, hjust = 1)) +
        theme(axis.line = element_line(size = 1, colour = "black"),
              panel.grid.major = element_line(colour = "#d3d3d3"),
              panel.grid.minor = element_blank(),
              panel.border = element_blank(),
              panel.background = element_blank()) +
        theme(legend.position="top") +
        geom_line(colour = "#ff1a1a", alpha = 0.8) +
        labs(x = "x value", y = "density",
             title = paste("exp(Lambda =", Lambda,")"))
```

```
  # Return the plot object
  return(p)
}

plotPareto <- function(Alpha, Beta){
  # Create the DataFrame with the density to plot
  x <- seq(0, 20, length.out=1000)
  dat <- data.frame(x=x, px = dpareto(x, location = Alpha, shape = Beta))

  # Create the plot using visual enhancements
  p <- ggplot(dat, aes(x = x,y = px)) +
        theme(axis.text.x = element_text(angle = 0, hjust = 1)) +
        theme(axis.line = element_line(size = 1, colour = "black"),
              panel.grid.major = element_line(colour = "#d3d3d3"),
              panel.grid.minor = element_blank(),
              panel.border = element_blank(),
              panel.background = element_blank()) +
        theme(legend.position="none") +
        geom_line(colour = "#1a75ff", alpha = 0.8) +
        labs(x = "x value", y = "density",
             title = paste("Pareto (Alpha =",
                           Alpha, ", Beta =", Beta, ")"))

  # Return the plot object
  return(p)
}
```

2. Once the functions are defined, we declare an extra auxiliary function that will allow us to plot multiple plots in only one figure for simplicity of the comparisons. This *multiplot*() function is available online in the *ggplot*2 package website and here we are including it as-is from the website.

```
# Multiple plot function
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
```

```
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}
```

3. Then, four different plots (different λ values) are generated for the exponential distribution by calling the auxiliary functions already defined: *plotExp*() and *multiplot*().

```
# Exponential distribution
# Supress warnings
options(warn = -1)

# Load libraries
library(ggplot2)

# Define the rate and create a DataFrame
Lambda = c(1, 10, 0.5, 0.1)

# Create a simple plot for visualizing the distribution
p1 <- plotExp(Lambda[1])
p2 <- plotExp(Lambda[2])
p3 <- plotExp(Lambda[3])
p4 <- plotExp(Lambda[4])

multiplot(p1, p2, p3, p4, cols=2)
```

4. Similarly, four different plots (different α and β values) are generated for the Pareto distribution by calling the auxiliary functions already defined: *plotPareto*() and *multiplot*().

```
# Supress warnings
options(warn = -1)

# Loading library
suppressMessages(library(EnvStats))

# Pareto distribution
Alpha = c(1, 10, 0.5, 0.1)
Beta = c(1, 2, 5, 10)

# Create a simple plot for visualizing the distribution
p1 <- plotPareto(Alpha[1], Beta[1])
p2 <- plotPareto(Alpha[2], Beta[2])
```

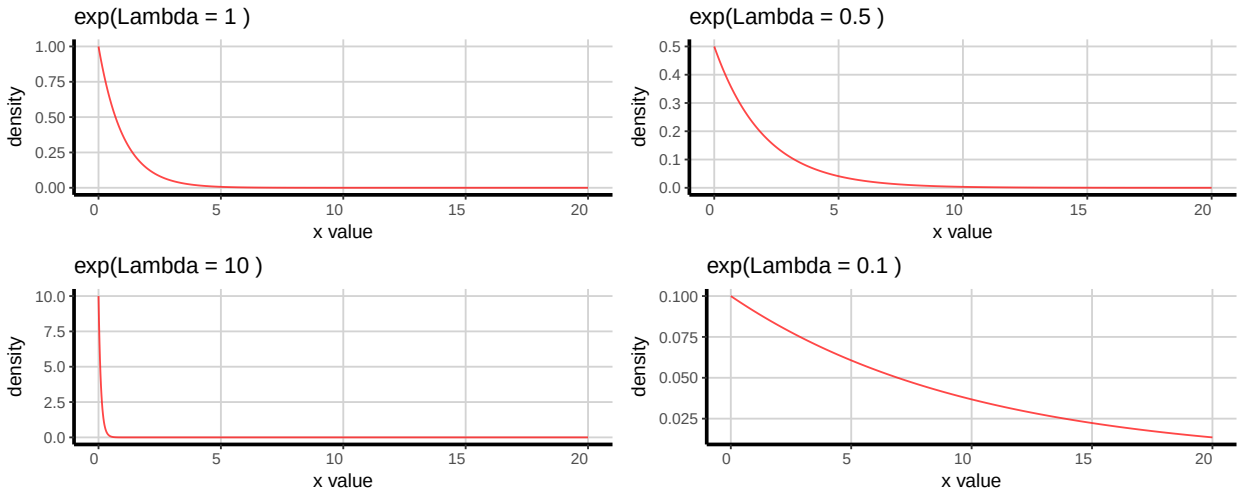

Figure 1: Exponential distribution plots (density)

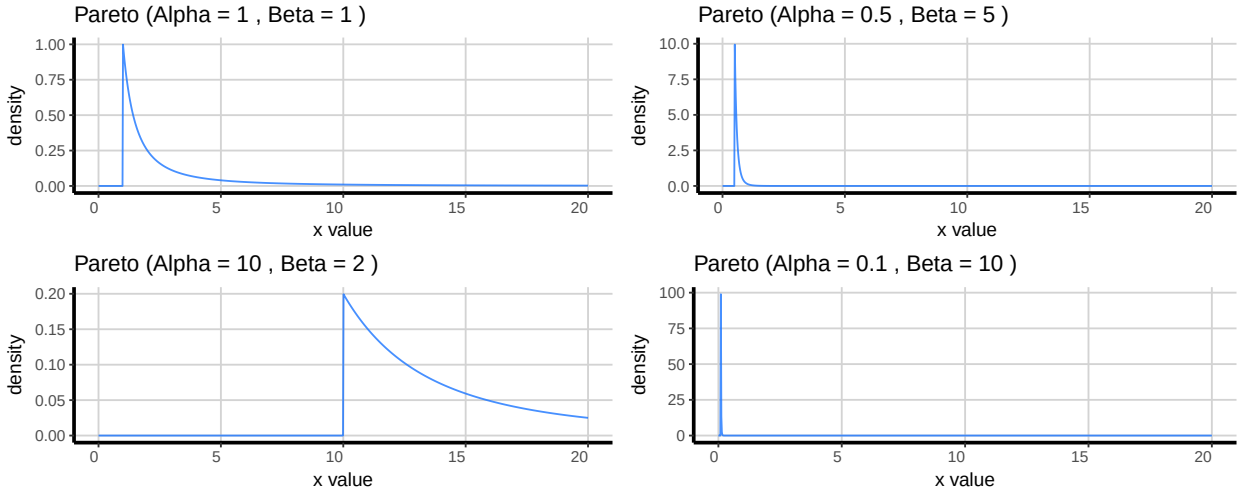

Figure 2: Pareto distribution plots (density)

```
p3 <- plotPareto(Alpha[3], Beta[3])
p4 <- plotPareto(Alpha[4], Beta[4])

multiplot(p1, p2, p3, p4, cols=2)
```

Looking at the plots, we can easily notice that the distribution with the heavier tail is the Pareto distribution as we expected since we know (from our background and classes) that *heavy-tailed distributions are probability distributions whose tails are not exponentially bounded: that is, they have heavier tails than the exponential distribution.* In this case, we know that the Pareto distribution is a subexponential distribution and thus, one of the classic heavy-tailed distributions.

Therefore, we will expect better estimation results using it as the sampling distribution for the importance sampling method. As we know from the lecture notes: *things can be badly behaved if we sample from a density with lighter tails than the density of interest.*

b) IS Estimation: Pareto distribution as sampling density

In this case, we have that the function f is an exponential distribution with $\lambda = 1$, shifted by two units to the right, such that we have:

$$f(x) = \lambda e^{-\lambda(x-2)} = e^{2-x} \tag{1}$$

On the other hand, we will use a Pareto distribution with parameters $\alpha = 2$ and $\beta = 3$ as the g function in the importance sampling method. Hence, we have:

$$g(x) = \frac{\beta \alpha^{\beta}}{x^{\beta+1}} = \frac{24}{x^4} \tag{2}$$

Finally, we will use $m = 10000$ in order to estimate $\phi = \mathbf{E}(X)$ and $\phi = \mathbf{E}(X^2)$, remembering that $\mathbf{VAR}(\hat{\phi}) \propto \mathbf{VAR}(h(X)f(X)/g(X))$. Histograms of $h(X)f(X)/g(X)$ and $f(X)/g(X)$ will be generated in order to identify wheter the variance $\mathbf{VAR}(\hat{\phi})$ is large. Extreme weights will be analyzed in order to check their impact on the estimation of $\hat{\mu}$.

1. First, we can easily define $w(X) = f(X)/g(X) = \dfrac{24e^{2-x}}{x^4}$ and we will drawn $m = 10000$ x_i values from our sampling distribution $g(X) = Pareto(\alpha, \beta) = 24/x^4$ where $x \in]\alpha, +\infty[$. Therefore, we perform the following operations:

```
# Perform the sampling operation
set.seed(0)
m = 10000
Alpha = 2
Beta = 3
x = rpareto(n = m, location = Alpha, shape = Beta)

# Define the f(x) function
fx <- function(x){
  return(exp(2 - x))
}

# Define the g(x) function
gx <- function(x){
  return(24 / (x ** 4))
}

# Determine w(X)
wx = (fx(x)/gx(x))
```

2. Now, since we have the weights $w(X)$, we can easily compute the mean value of the final expression in order to estimate the $\phi = \mathbf{E}(X)$ and $\phi = \mathbf{E}(X^2)$ values.

```
# E(X) estimation
Ex = mean(x * wx)
print(paste0("Estimation of phi = E(x) = ", Ex))
```

```
## [1] "Estimation of phi = E(x) = 3.01011721925379"
```

```
# By real formula
ParetoMean = (Beta * Alpha) / (Beta - 1)
print(paste0("E(x) = ", ParetoMean))
```

```
## [1] "E(x) = 3"
```

```
# E(X^2) estimation
Ex2 = mean((x ** 2) * wx)
print(paste0("Estimation of phi = E(x^2) = ", Ex2))
```

```
## [1] "Estimation of phi = E(x^2) = 10.1144394883712"
```

```
# Variance estimation
# By formula VAR(X) = E[X^2] - E[X]^2
Varx = Ex2 - (Ex ** 2)
print(paste0("Estimation of VAR(phi) = ", Varx))
```

```
## [1] "Estimation of VAR(phi) = 1.05363381472306"
```

```
# By direct implementation
Varx_2 = var(x * wx)
print(paste0("Estimation of VAR(X * w(X)) = ", Varx_2))
```

```
## [1] "Estimation of VAR(X * w(X)) = 2.43006469145961"
```

```
# By real formula
ParetoVar = (Beta * (Alpha ** 2)) / (((Beta - 1) ** 2) * (Beta - 2))
print(paste0("VAR(X) = ", ParetoVar))
```

```
## [1] "VAR(X) = 3"
```

From the previous results, we can notice that the estimation of the mean value is pretty accurate (almost identical to the real mean of the distribution). On the other hand, it is possible to see that we are able to reduce the variance of the sampling distribution thanks to the importance sampling implementation with a heavy tailed distribution such as with the Pareto distribution we are using in this case. Therefore, the main objective of the approach is satisfied.

3. Histograms for $h(X)f(X)/g(X)$ and the weights $w(X) = f(X)/g(X)$ are generated in order to get an idea wheter the $\mathbf{VAR}(\phi)$ is large.

```
# Create histograms
# h(X)f(X)/g(X)
qplot(x*wx, geom="histogram",
      xlab = "h(X)f(X)/g(X)",
      ylab = "frequency",
      fill=I("#ff1a1a"),
      col=I("black"),
```

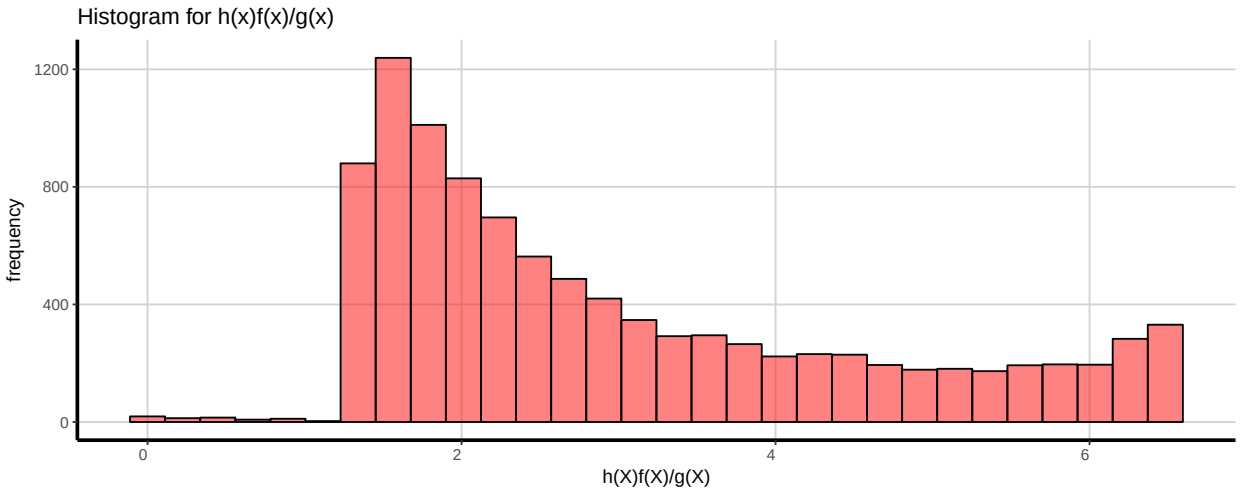

Figure 3: $h(X)w(X)$ histogram using shifted exp() distribution as sampling function

```
      alpha = 0.8) +
    theme(axis.text.x = element_text(angle = 0, hjust = 1)) +
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank()) +
    theme(legend.position="none") +
    labs(title = "Histogram for h(x)f(x)/g(x)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

Looking at the previous histogram we can notice that several values are concentrated around the value of 2 for the function under study. On the other hand, the maximum values are approximately 6.5 units.

```
# Create histograms
# w(X) = f(X)/g(X)
qplot(wx, geom="histogram",
      xlab = "w(X) = f(X)/g(X)",
      ylab = "frequency",
      fill=I("#1a75ff"),
      col=I("black"),
      alpha = 0.8)+
    theme(axis.text.x = element_text(angle = 0, hjust = 1)) +
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank()) +
    theme(legend.position="none") +
    labs(title = "Histogram for w(X) = f(X)/g(X)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

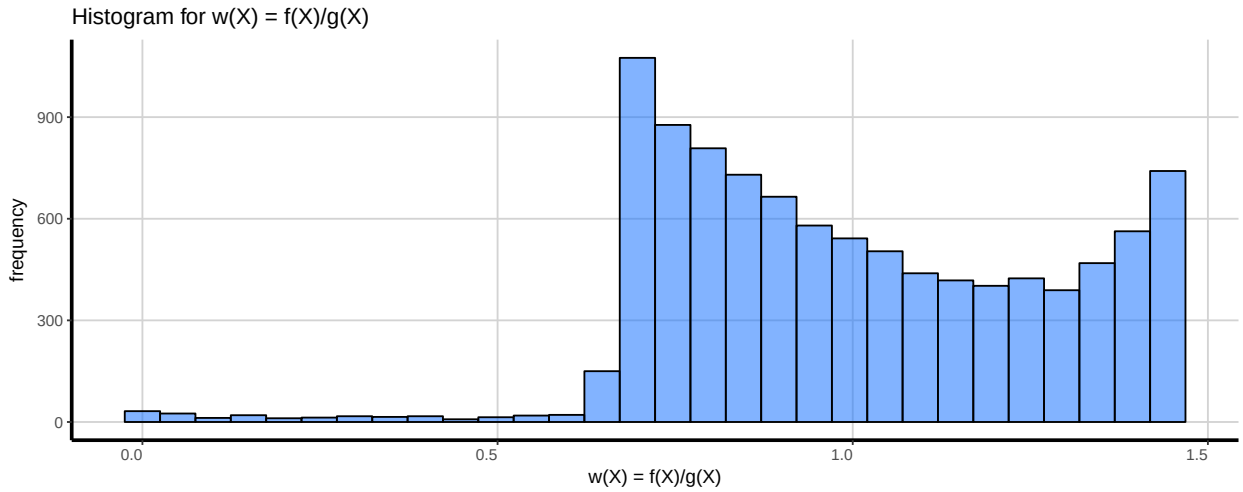

Figure 4: Weights $w(X)$ histogram using Pareto distribution as sampling function

Based on the previous plot containing the values of the weights, we can clearly see that there are not extreme weights that would have a very strong influence on $\hat{\mu}$ since the largest value is about 1.444. Therefore, the estimation of the expected value of X is very accurate.

c) IS Estimation: Exponential distribution as sampling density

Repeating the same procedure as before while changing the role of g and f, we have:

1. First, we can easily define $w(X) = f(X)/g(X) = \dfrac{x^4}{24e^{2-x}}$ and we will drawn $m = 10000$ x_i values from our sampling distribution $g(X) = exp(\lambda)_{shifted} = e^{2-x}$ where $x \in]2, +\infty[$. Therefore, we perform the following operations:

```r
# Perform the sampling operation
set.seed(0)
m = 10000
Lambda = 1
x = rexp(n = m, rate = Lambda) + 2

# Define the f(x) function
fx <- function(x){
  return(24 / (x ** 4))
}

# Define the g(x) function
gx <- function(x){
  return(exp(2 - x))
}

# Determine w(X)
wx = (fx(x)/gx(x))
```

2. Now, since we have the weights $w(X)$, we can easily compute the mean value of the final expression in order to estimate the $\phi = \mathbf{E}(X)$ and $\phi = \mathbf{E}(X^2)$ values.

```r
# E(X) estimation
Ex = mean(x * wx)
print(paste0("Estimation of phi = E(x) = ", Ex))
```

```
## [1] "Estimation of phi = E(x) = 2.96239840429051"
```

```r
# By real formula (solve integral from shifted mean)
ExpMean = 3
print(paste0("E(x) = ", ExpMean))
```

```
## [1] "E(x) = 3"
```

```r
# E(X^2) estimation
Ex2 = mean((x ** 2) * wx)
print(paste0("Estimation of phi = E(x^2) = ", Ex2))
```

```
## [1] "Estimation of phi = E(x^2) = 10.4236483237609"
```

```r
# Variance estimation
# By formula VAR(X) = E[X^2] - E[X]^2
Varx = Ex2 - (Ex ** 2)
print(paste0("Estimation of VAR(phi) = ", Varx))
```

```
## [1] "Estimation of VAR(phi) = 1.64784401801794"
```

```r
# By direct implementation
Varx_2 = var(x * wx)
print(paste0("Estimation of VAR(X * w(X)) = ", Varx_2))
```

```
## [1] "Estimation of VAR(X * w(X)) = 10.8972332392936"
```

```r
# By real formula (solving the integrals)
ExpVar = 1
print(paste0("VAR(X) = ", ExpVar))
```

```
## [1] "VAR(X) = 1"
```

From the previous results we can easily check that the mean value estimation is very accurate but in this case, the variance estimation surpass the true value of the distribution variance due to the fact that the exponential distribution is not as good as the Pareto distribution for applying the importance sampling methodology: it does not satisfy the assumption of being a heavy tail distribution. Therefore, we are not able to reduce the variance of the sampling.

3. Histograms for $h(X)f(X)/g(X)$ and the weights $w(X) = f(X)/g(X)$ are generated in order to get an idea wheter the $\mathbf{VAR}(\phi)$ is large.

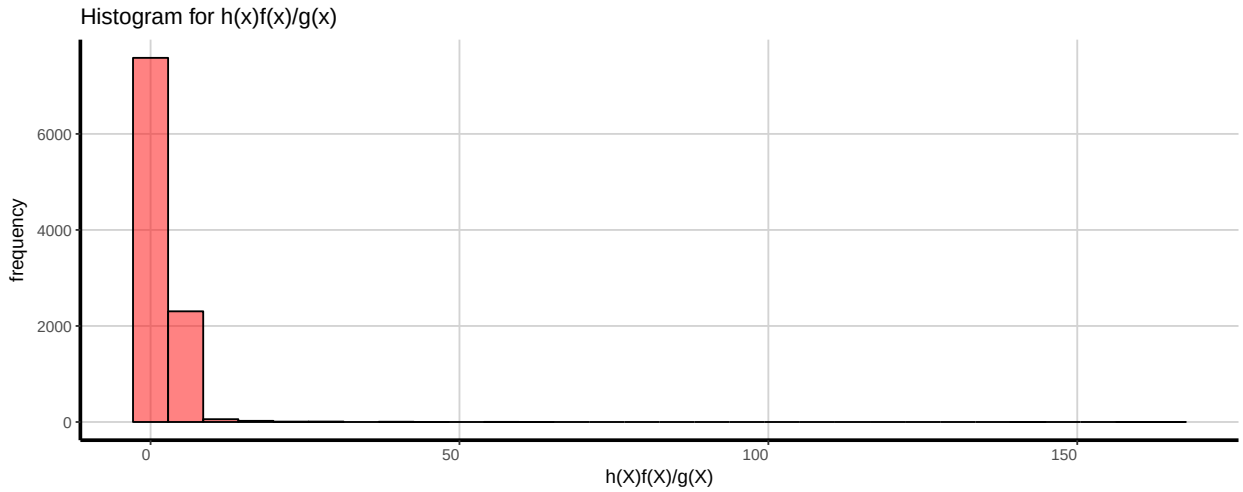

Figure 5: $h(X)w(X)$ histogram using shifted exp() distribution as sampling function

```
# Create histograms
# h(X)f(X)/g(X)
hist1 <- qplot(x*wx, geom="histogram",
      xlab = "h(X)f(X)/g(X)",
      ylab = "frequency",
      fill=I("#ff1a1a"),
      col=I("black"),
      alpha = 0.8) +
    theme(axis.text.x = element_text(angle = 0, hjust = 1)) +
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank()) +
    theme(legend.position="none") +
    labs(title = "Histogram for h(x)f(x)/g(x)")

print(hist1)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

Looking at the previous histogram we can notice that several values are concentrated around the value of $2-3$ for the function under study. On the other hand, the maximum values are approximate 167.157 units.

```
# Create histograms
# w(X) = f(X)/g(X)
qplot(wx, geom="histogram",
      xlab = "w(X) = f(X)/g(X)",
      ylab = "frequency",
      fill=I("#1a75ff"),
      col=I("black"),
      alpha = 0.8,
      xlim = c(0,15))+
    theme(axis.text.x = element_text(angle = 0, hjust = 1)) +
```

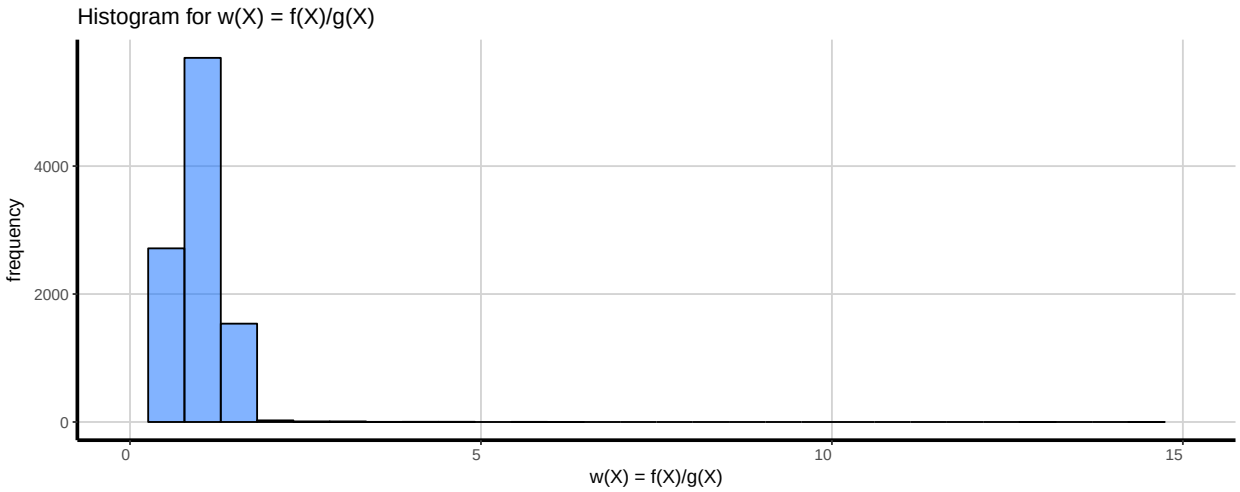

Figure 6: Weights $w(X)$ histogram using shifted exp() distribution as sampling function

```
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank()) +
    theme(legend.position="none") +
    labs(title = "Histogram for w(X) = f(X)/g(X)")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

Based on the previous plot containing the values of the weights, we can clearly see that there are extreme weights that would have a very strong influence on $\hat{\mu}$ since the largest value is about 14.945. Therefore, we can expect that the value of $\hat{\mu}$ will not be significantly impacted by these extreme weights since their frequency is very low in comparison to the rest of the variables, but the variance of the distribution will be significantly impacted by these extreme weights, as we already saw when comparing it to the variance of the original distribution (larger).

Problem 2: Helical valley function

In this problem, we want to study the so called *helical valley* function included in the $ps8.R$ file. The main idea consists of analyzing it behavior using visual tools such as *contour* lines for ploting it in a two-dimensional fashion and then, after understanding the basic patterns of the function, try to optimize it and find its minimum by a series of different functions included in R such as $optim()$, $optimx()$, and $nlm()$.

Finally, a series of five different starting points will be tested in order to check and explore the possibility of reaching multiple local minima.

1. We start by loading the file containing the function under study:

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Load the function
source("ps8.R")

# Testing the function (f)
x = c(1,10,4)
f(x)
```

```
## [1] 8481.129
```

Now, we start analyzing the function.

a) Initial analysis: contours of the function

In order to be able to generate beautiful and explicit contour plots of the function, we define an auxiliary function called $ContourPlot()$ that will fix the value of the third component of the vector X (given as an argument of the function) in order to generate a slice of the three-dimensional function. The main logic of the function is as follows:

i) A grid for the components $X1$ and $X2$ is generated in order to be able to generate a surface. These vectors are manipulated in order to be able to create a DataFrame containing all the components (each row is a vector) needed to pass as arguments of the helical function.

ii) Using this DataFrame, a simple $apply()$ call using the helical function as the main input will allow us to evaluate each row/vector with the function, obtaining a new vector of z values.

iii) This vector is transformed into a matrix (surface format) in order to be able to use the classic contour functions from both the base and *ggplot* packages.

iv) Depending on the flag passed to the *color* argument (boolean), a simple black and white or a colorful contour plot will be generated.

Therefore, the $ContourPlot()$ function is as follows:

```
ContourPlot <- function(X3=0, color=TRUE){
  # Load the relevant function
  source("ps8.R")
```

```
  # Create the grid based on simple vectors
  X1 = seq(-300,300,1)
  X2 = seq(-300,300,1)
  X11 = rep(X1, each=length(X2))
  X22 = rep(X2, length(X1))

  # Fix the third variable value
  X3 = rep(X3,length(X1)*length(X2))

  # Generate a DataFrame
  Dat = data.frame("x1" = X11, "x2" = X22, "x3" = X3)

  # Apply the helical function and transform output to a matrix (surface)
  Z = apply(Dat, 1, function(x) f(x))
  Z = matrix(Z,length(X1),length(X2))

  # Plot the contour
  if (color == FALSE) {
    fc <- contour(X1,X2,Z)
  }

  # Fancy plot
  if (color == TRUE){
    fc <- filled.contour(x = X1,
                         y = X2,
                         z = Z,
                         color.palette = colorRampPalette(c("yellow", "red")),
                         xlab = "X1",
                         ylab = "X2",
                         main = paste0("Helical contour X3 = ", X3[1]),
                         key.title = title(main = "Value", cex.main = 1))
  }

  # Return the plot object
  return(fc)
}
```

Thus, we only need to call the function using different values for the fixed $X3$ component of the vectors in order to obtain a series of different (and beautiful) contour plots:

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
library("ggplot2")

# Call the function for different values of X3
fc0 <- ContourPlot()
```

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
```

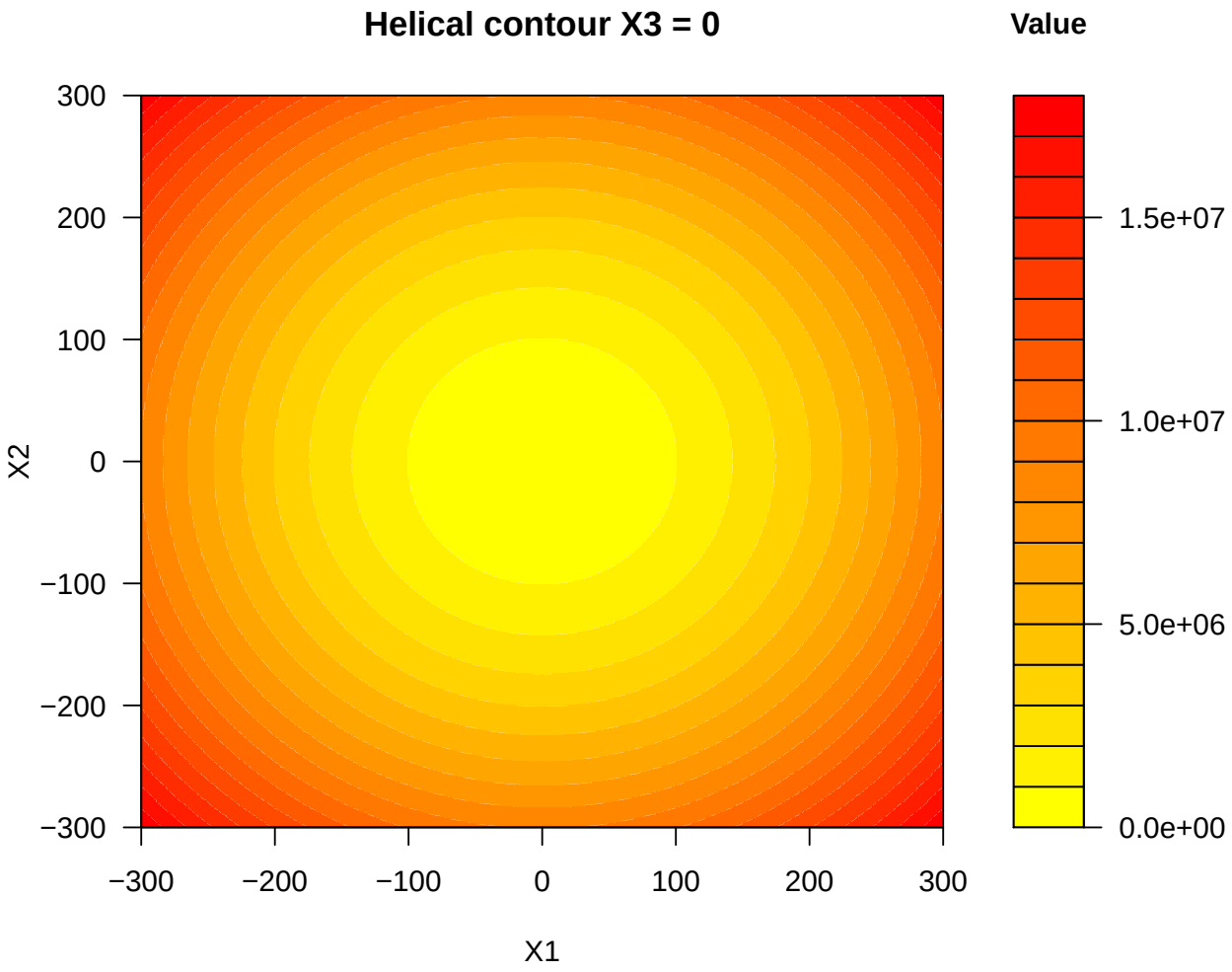

Figure 7: Different contour plots from the Helical function (fixed $X3 = 0$ value)

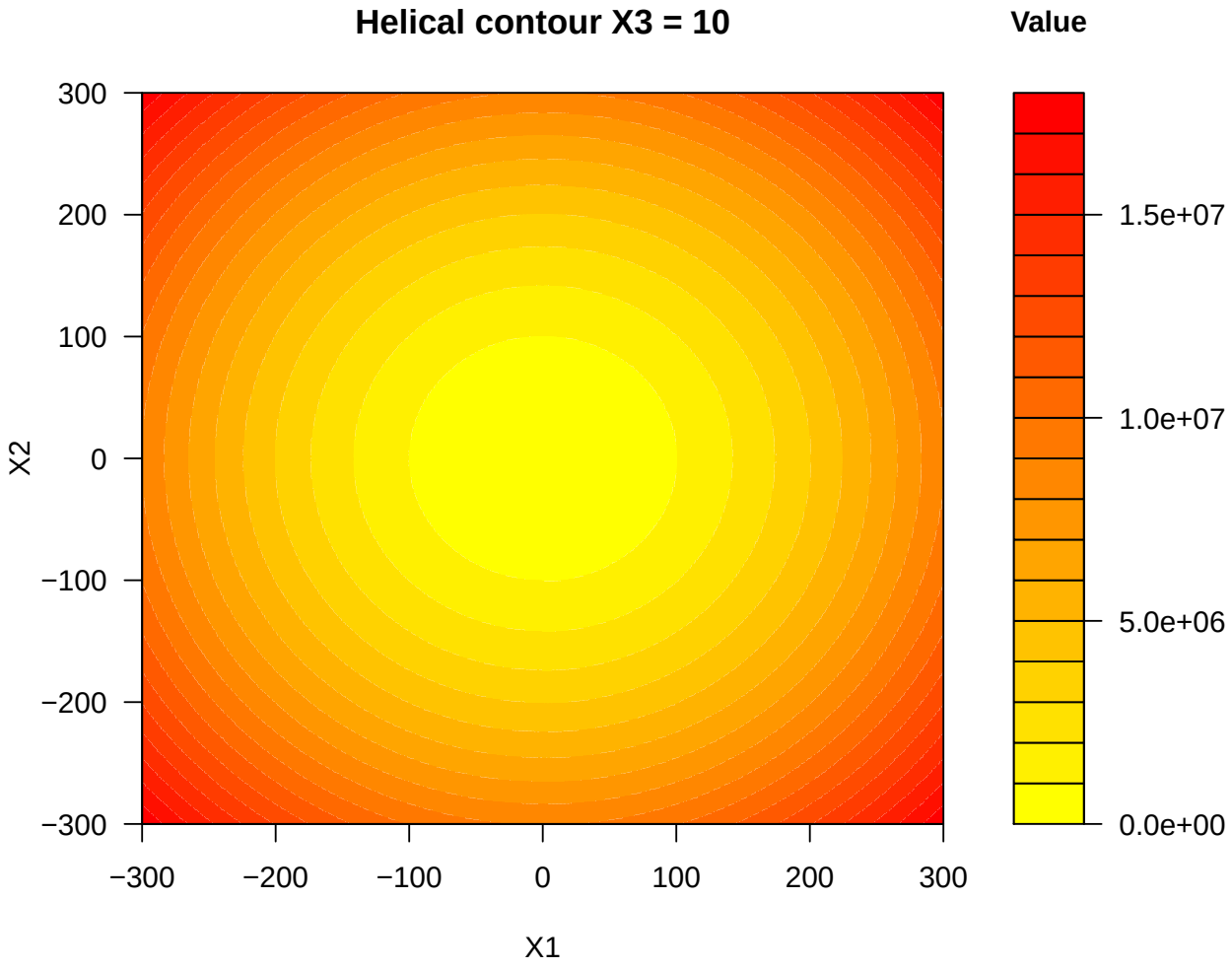

Figure 8: Different contour plots from the Helical function (fixed $X3 = 1$ value)

```
library("ggplot2")

# Generate the contour plot
fc1 <- ContourPlot(10)
```

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
library("ggplot2")

# Generate the contour plot
fc2 <- ContourPlot(100)
```

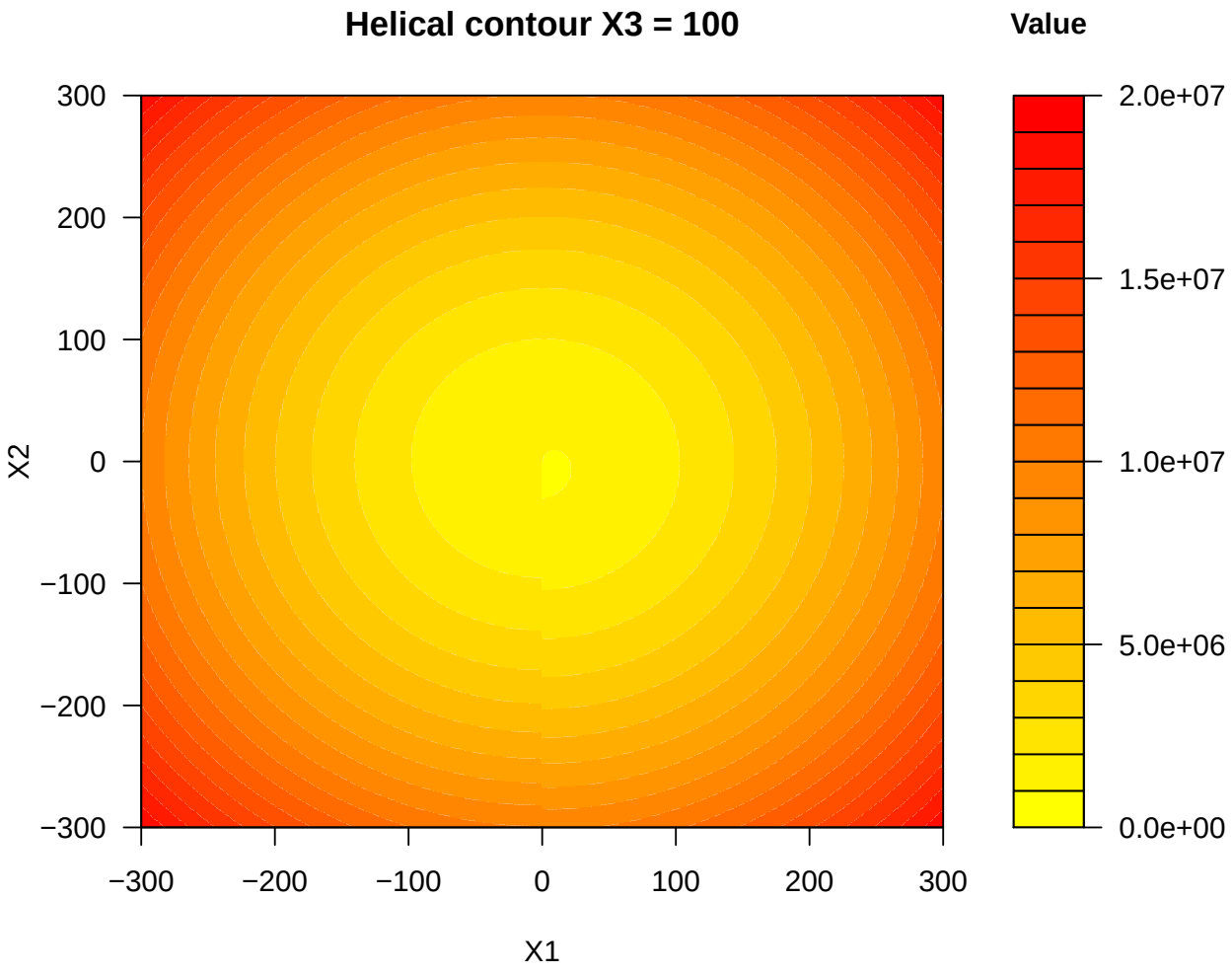

Figure 9: Different contour plots from the Helical function (fixed $X3 = 10$ value)

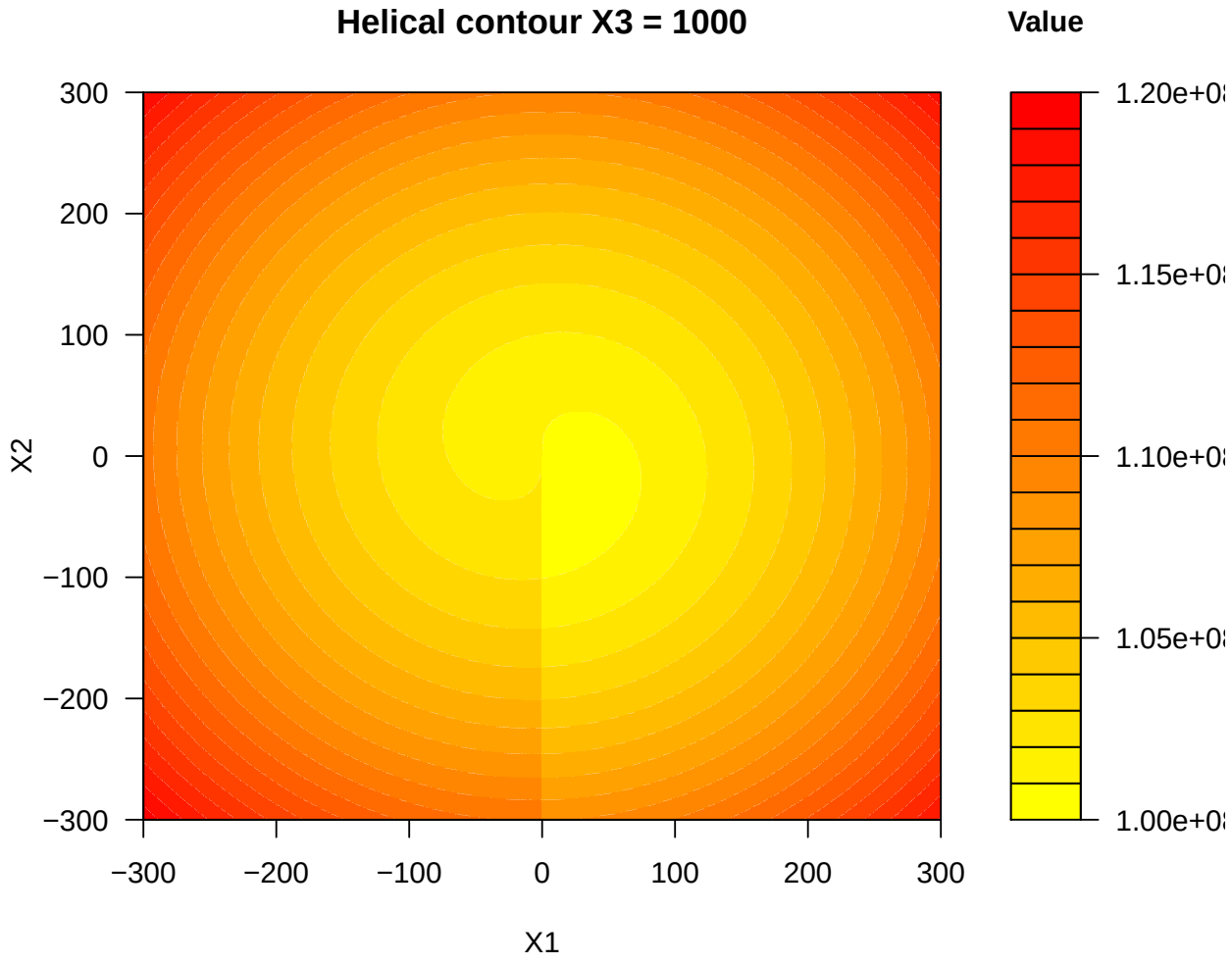

Figure 10: Different contour plots from the Helical function (fixed $X3 = 1e3$ value)

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
library("ggplot2")

# Generate the contour plot
fc3 <- ContourPlot(1000)
```

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
library("ggplot2")

# Generate the contour plot
```

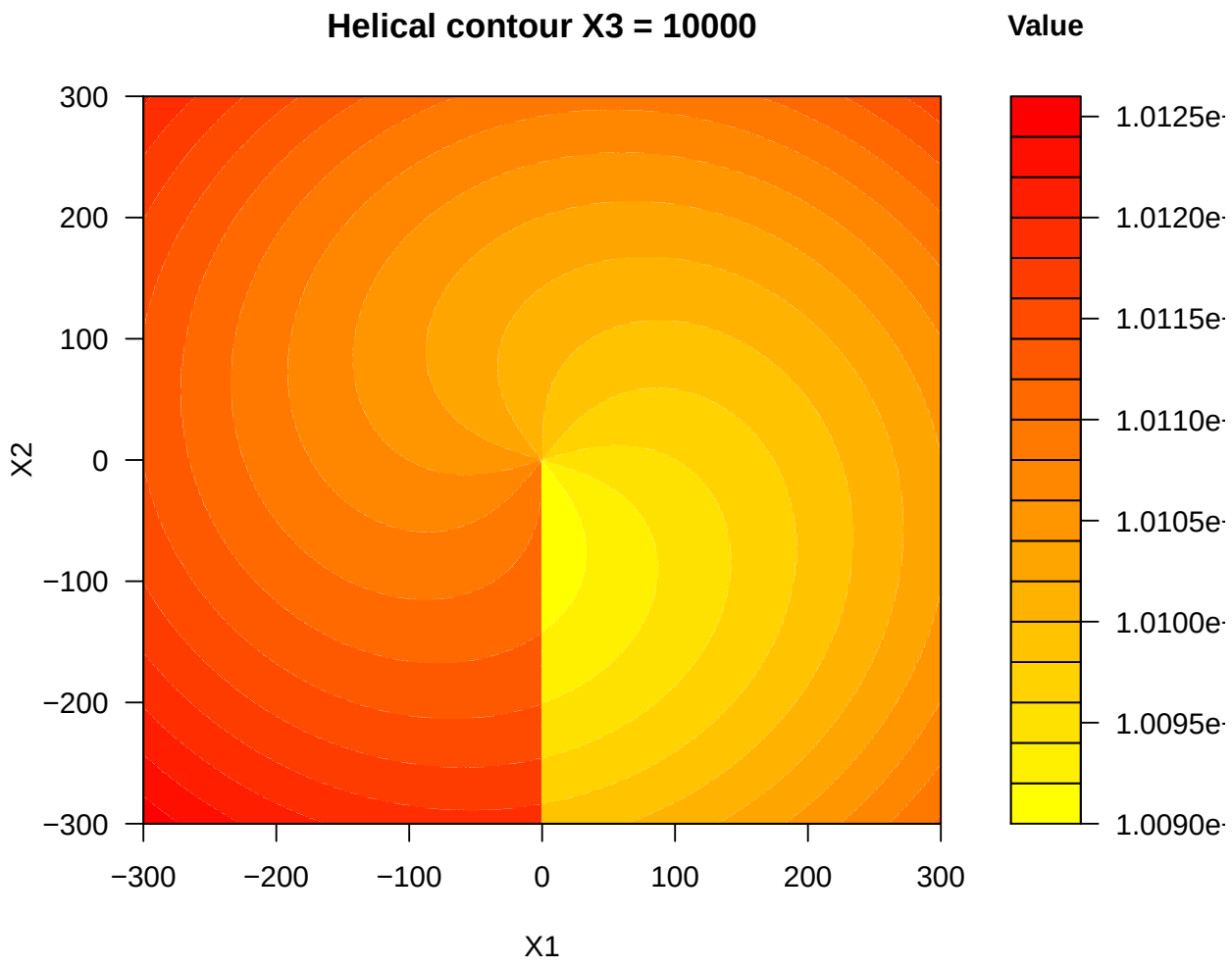

Figure 11: Different contour plots from the Helical function (fixed $X3 = 1e4$ value)

```
fc4 <- ContourPlot(10000)
```

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
library("ggplot2")

# Generate the contour plot
fc5 <- ContourPlot(100000)
```

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
```

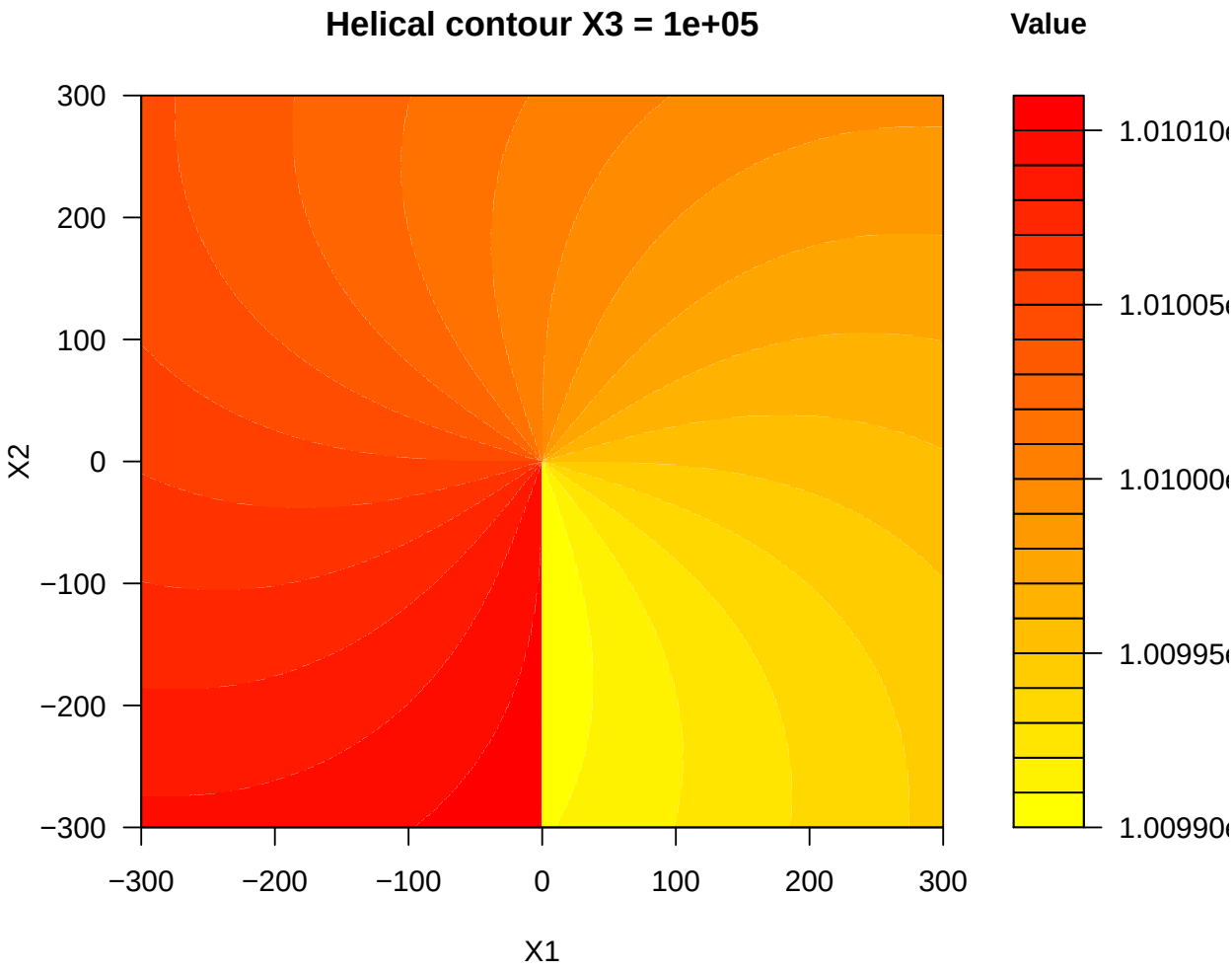

Figure 12: Different contour plots from the Helical function (fixed $X3 = 1e5$ value)

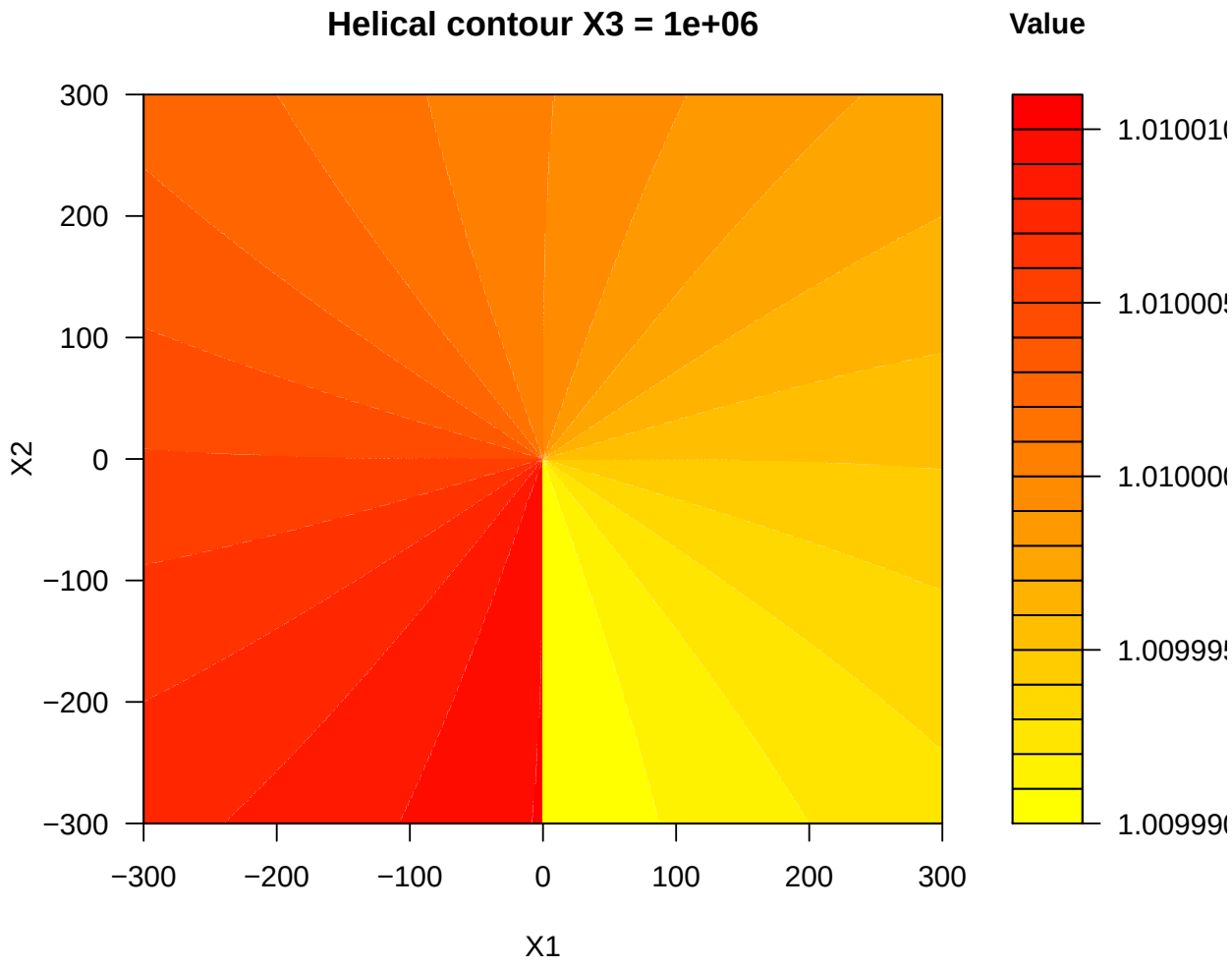

Figure 13: Different contour plots from the Helical function (fixed $X3 = 1e6$ value)

```
library("ggplot2")

# Generate the contour plot
fc6 <- ContourPlot(1000000)
```

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
library("ggplot2")

# Generate the contour plot
fc7 <- ContourPlot(-1000, FALSE)
```

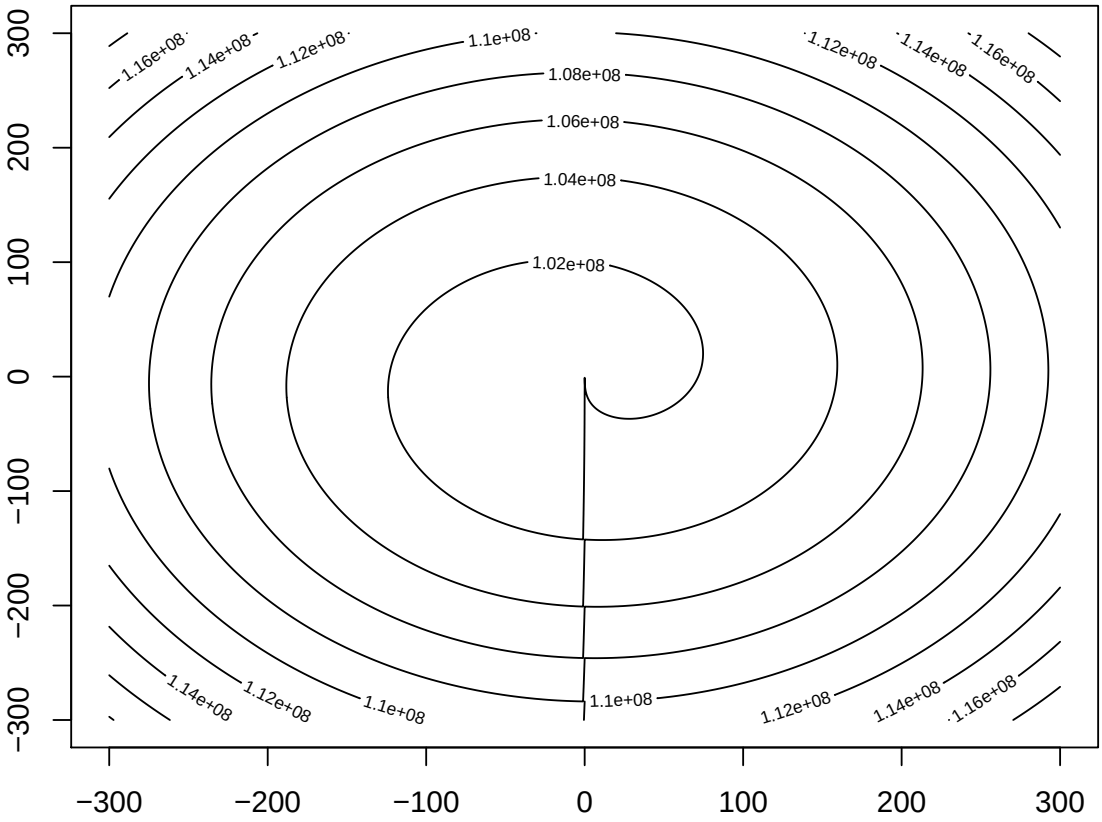

Figure 14: Different contour plots from the Helical function (fixed $X3 = -1e4$ value, no color example)

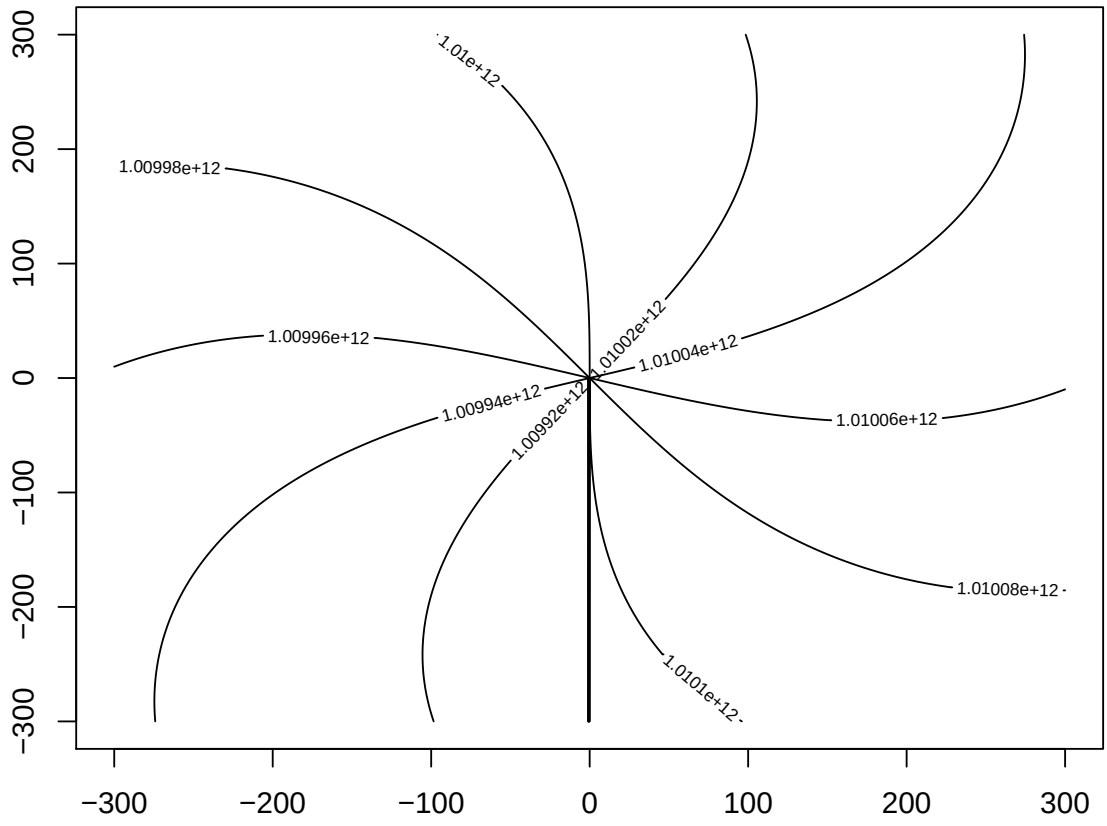

Figure 15: Different contour plots from the Helical function (fixed $X3 = -1e6$ value, no color example)

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Loading library
library("ggplot2")

# Generate the contour plot
fc8 <- ContourPlot(-100000, FALSE)
```

b) Finding the minimum: *optim*(), *optimx*(), and *nlm*() functions

In this section, we would like to study the function in terms of its potential local minima points as well as try to find the global (if exists) minimum value of the helical function. In order to perform this task, we will define a series of auxiliary functions that will act as wrappers of the already studied *optim*(), *optimx*(), and *nlm*() functions in R. The main idea and strategy will be the following:

i) Several starting points X_0 will be tested in order to find the minimum of the function. Thanks to this approach, we will be able to detect potential (multiple) local minima.

ii) After each optimization with the *optim*() and *nlm*() functions, we will take the final solution as a new starting point, checking if the convergence of the algorithms was not enough, solving the minimization problem in an iterative way such that we can clearly identify which points are local minimums (if they exist). This procedure is avoided with the *optimx*() function in order to contrast the iterative results with a one-optimization (classic) approach. Important is to note that the *optimx*() function gives us more information about the type of solution reached by reporting the status of the KKT multipliers.

iii) Different optimization methods such as BFGS and Nelder-Mead will be tested for comparison purposes.

Thus, we proceed as follows:

1. We first define our *OptF*() auxiliary function that will take a starting point x_0 given by the user and will optimize the helical function. Once the new point and new objective value are obtained, we check if the current solution is an optimal point or a local minimum by re-optimizing the function from that point. We continue with this fashion until the solution converges (no improvement is obtained) to a certain value.

```
OptF <- function(X0 = c(0, 0, 0)){
  # Load relevant functions
  source("ps8.R")

  # Initial Objective Value
  InitialObj <- f(X0)

  # List of methods available
  Methods <- c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent")

  # Perform the optimization
  Optimization <- optim(X0, f, method = Methods, control = (maxit = 1e6))

  # Initial Optimization summary
  NewObjValue <- Optimization$value
  NewPoint <- Optimization$par

  # Print results
  print("------- Optimization Results -------")
  print("Initial point:")
  print(X0)
  print(paste0("Minimum value obtained: ", NewObjValue))
  print("Optimal Point obtained:")
  print(NewPoint)
  cat("\n")

  # Update the initial objective function
```

```
  InitialObj = NewObjValue

  # Re-optimization loop: breaks if no improvement is attained
  while (TRUE) {
    # Checking if the algorithm got stuck in a local minimum: re-optimize
    print("------- Re-optimization from new point -------")
    Optimization <- optim(NewPoint, f, method = Methods)
    print("Initial point:")
    print(NewPoint)
    print(paste0("Minimum value obtained: ", Optimization$value))
    print("Optimal Point obtained:")
    print(Optimization$par)

    # Update the solution
    NewObjValue = Optimization$value
    NewPoint = Optimization$par

    # Check if no improvement occurs
    if (NewObjValue >= InitialObj){
      break
    }

    # Else, go to next iteration starting from the new point (update obj function)
    else {
      InitialObj = NewObjValue
      cat("\n")
    }

  }
}
```

2. Thus, we test a series of different starting points x_0 in order to study the behavior of the algorithms when different x_0 values are provided.

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Initial points
X0 = c(0, 0, 0)
X0_2 = c(1, 0, 0)
X0_3 = c(100, 100, 100)
X0_4 = c(1, 0, -1)
X0_5 = c(1, -1, 0)
X0_6 = c(0, 1, 0)
X0_7 = c(0, 0, 1)
X0_8 = c(1, 1, 1)
X0_9 = c(-1, -1, -1)
X0_10 = c(0, -1, 0)
X0_11 = c(1e8,-1e8,100)
InitialP = list(X0, X0_2, X0_3, X0_4, X0_5,
                X0_6, X0_7, X0_8, X0_9, X0_10,
                X0_11)
```

```r
# Call the function
for (x0 in InitialP) {
  OptF(x0)
  cat("\n\n")
}
```

```
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 0 0 0
## [1] "Minimum value obtained: 1.87685068944381e-05"
## [1] "Optimal Point obtained:"
## [1] 0.999978292 0.002730698 0.004284640
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 0.999978292 0.002730698 0.004284640
## [1] "Minimum value obtained: 6.86578740796613e-14"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -4.976556e-08 -7.539281e-08
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -4.976556e-08 -7.539281e-08
## [1] "Minimum value obtained: 4.81105865727058e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.343413e-09 4.077143e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 2.343413e-09 4.077143e-09
## [1] "Minimum value obtained: 4.80126333544469e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -6.202660e-10 -1.477071e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -6.202660e-10 -1.477071e-09
## [1] "Minimum value obtained: 4.80126333544469e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -6.202660e-10 -1.477071e-09
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 1 0 0
## [1] "Minimum value obtained: 0"
## [1] "Optimal Point obtained:"
## [1] 1 0 0
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1 0 0
## [1] "Minimum value obtained: 0"
```

```
## [1] "Optimal Point obtained:"
## [1] 1 0 0
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 100 100 100
## [1] "Minimum value obtained: 0.090781904737348"
## [1] "Optimal Point obtained:"
## [1] 0.9990108 0.1564930 0.2580923
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 0.9990108 0.1564930 0.2580923
## [1] "Minimum value obtained: 2.74527570386929e-10"
## [1] "Optimal Point obtained:"
## [1]  1.000001e+00 -9.272825e-06 -1.410066e-05
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000001e+00 -9.272825e-06 -1.410066e-05
## [1] "Minimum value obtained: 4.83841935092773e-16"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.886505e-09 2.708915e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 2.886505e-09 2.708915e-09
## [1] "Minimum value obtained: 1.14373076117071e-16"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -1.408613e-09 -3.076882e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -1.408613e-09 -3.076882e-09
## [1] "Minimum value obtained: 1.14373076117071e-16"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -1.408613e-09 -3.076882e-09
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1]  1  0 -1
## [1] "Minimum value obtained: 1.37400152502972e-06"
## [1] "Optimal Point obtained:"
## [1] 0.9999588519 0.0005906569 0.0009880445
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 0.9999588519 0.0005906569 0.0009880445
## [1] "Minimum value obtained: 6.17209951149445e-15"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -2.253101e-08 -3.750548e-08
##
```

```
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -2.253101e-08 -3.750548e-08
## [1] "Minimum value obtained: 9.26446920575781e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -7.317122e-10 -1.877517e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -7.317122e-10 -1.877517e-09
## [1] "Minimum value obtained: 9.26446920575781e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -7.317122e-10 -1.877517e-09
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1]  1 -1  0
## [1] "Minimum value obtained: 1.39365418949491e-06"
## [1] "Optimal Point obtained:"
## [1]  1.0000458016 -0.0003983705 -0.0007158905
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.0000458016 -0.0003983705 -0.0007158905
## [1] "Minimum value obtained: 1.90754350968538e-15"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.460889e-08 3.950748e-08
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 2.460889e-08 3.950748e-08
## [1] "Minimum value obtained: 1.02990878037001e-16"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -1.523482e-09 -2.734133e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -1.523482e-09 -2.734133e-09
## [1] "Minimum value obtained: 1.02990878037001e-16"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -1.523482e-09 -2.734133e-09
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 0 1 0
## [1] "Minimum value obtained: 4.63550713746938e-06"
## [1] "Optimal Point obtained:"
## [1]  0.9998882225 -0.0002040018 -0.0001412321
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  0.9998882225 -0.0002040018 -0.0001412321
```

```
## [1] "Minimum value obtained: 1.96159197870355e-14"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 7.619601e-08 1.135280e-07
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 7.619601e-08 1.135280e-07
## [1] "Minimum value obtained: 9.35257875411798e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 1.182485e-10 2.922815e-12
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 1.182485e-10 2.922815e-12
## [1] "Minimum value obtained: 9.35257875411798e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 1.182485e-10 2.922815e-12
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 0 0 1
## [1] "Minimum value obtained: 1.21960665609239e-05"
## [1] "Optimal Point obtained:"
## [1] 0.999711245 0.001178555 0.001921611
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 0.999711245 0.001178555 0.001921611
## [1] "Minimum value obtained: 2.04959623860683e-13"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.351168e-08 2.656368e-08
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 2.351168e-08 2.656368e-08
## [1] "Minimum value obtained: 2.50685830168957e-16"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -1.936495e-09 -1.572035e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -1.936495e-09 -1.572035e-09
## [1] "Minimum value obtained: 1.54147270740746e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00  2.102641e-10 -5.653182e-11
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00  2.102641e-10 -5.653182e-11
## [1] "Minimum value obtained: 1.54147270740746e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00  2.102641e-10 -5.653182e-11
##
```

```
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 1 1 1
## [1] "Minimum value obtained: 1.34309833281301e-07"
## [1] "Optimal Point obtained:"
## [1]  0.9999779414 -0.0001349269 -0.0001927127
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  0.9999779414 -0.0001349269 -0.0001927127
## [1] "Minimum value obtained: 3.2705314538929e-15"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.062078e-08 3.353981e-08
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 2.062078e-08 3.353981e-08
## [1] "Minimum value obtained: 5.340526369598e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.980380e-09 4.692446e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 2.980380e-09 4.692446e-09
## [1] "Minimum value obtained: 5.340526369598e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.980380e-09 4.692446e-09
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] -1 -1 -1
## [1] "Minimum value obtained: 7.57038844468969e-06"
## [1] "Optimal Point obtained:"
## [1] 0.9997731731 0.0005162083 0.0009457322
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 0.9997731731 0.0005162083 0.0009457322
## [1] "Minimum value obtained: 7.05786164128479e-14"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 1.003230e-07 1.486682e-07
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 1.003230e-07 1.486682e-07
## [1] "Minimum value obtained: 6.37402474898431e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.220662e-09 4.202129e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 2.220662e-09 4.202129e-09
```

```
## [1] "Minimum value obtained: 6.37402474898431e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.220662e-09 4.202129e-09
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1]  0 -1  0
## [1] "Minimum value obtained: 1.36812797910892e-06"
## [1] "Optimal Point obtained:"
## [1] 1.0000086475 0.0002886151 0.0003480098
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.0000086475 0.0002886151 0.0003480098
## [1] "Minimum value obtained: 2.82857560029275e-14"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 7.463305e-08 1.226361e-07
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 7.463305e-08 1.226361e-07
## [1] "Minimum value obtained: 9.33744997662316e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -2.722594e-09 -4.308707e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -2.722594e-09 -4.308707e-09
## [1] "Minimum value obtained: 4.51039394149301e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.255781e-09 3.584260e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 2.255781e-09 3.584260e-09
## [1] "Minimum value obtained: 4.43499515909872e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -3.617634e-10 -1.141196e-12
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -3.617634e-10 -1.141196e-12
## [1] "Minimum value obtained: 4.43499515909872e-17"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -3.617634e-10 -1.141196e-12
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1]  1e+08 -1e+08  1e+02
## [1] "Minimum value obtained: 46824169178.9009"
## [1] "Optimal Point obtained:"
## [1]  14847.797  -9694.643 -12342.341
```

```
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  14847.797  -9694.643 -12342.341
## [1] "Minimum value obtained: 477.338316786789"
## [1] "Optimal Point obtained:"
## [1] -1.422405  1.550247  1.804855
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] -1.422405  1.550247  1.804855
## [1] "Minimum value obtained: 4.33326233872697e-06"
## [1] "Optimal Point obtained:"
## [1]  0.9998894863 -0.0003725762 -0.0007526326
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  0.9998894863 -0.0003725762 -0.0007526326
## [1] "Minimum value obtained: 1.20189811706891e-14"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -4.293611e-08 -7.593080e-08
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1]  1.000000e+00 -4.293611e-08 -7.593080e-08
## [1] "Minimum value obtained: 8.9441864470321e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 3.875178e-09 6.821869e-09
##
## [1] "------- Re-optimization from new point -------"
## [1] "Initial point:"
## [1] 1.000000e+00 3.875178e-09 6.821869e-09
## [1] "Minimum value obtained: 8.9441864470321e-17"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 3.875178e-09 6.821869e-09
```

Based on the results, we can clearly see that some points such as $(1, -6.202e-10-1.477e-09)$ reach a very low (close to zero) objective function value without reaching the optimal solution of the optimization problem, and thus, the algorithm got stuck in that (those) point(s) without further improvements in the objective function. However, we can notice that all solutions besides the one associated with the last starting point (very high magnitude of its components) tend to reach points very similar to the $(1, 0, 0)$. The last starting point provided is a particular case, acting as an outlier in terms of the magnitude of its components and the final objective value reached with some methods such as $Nelder - Mead$ is clearly not optimal (diverges), however, with other methods such as BFGS we can see that the solution tends to be the $(1, 0, 0)$ vector.

Thus, we can conclude that the optimization of the objective function is not particularlt very sensitive with respect to the given starting point provided and the method applied, noticing that some points are not good starting points for this particular function if we want to find the global optimum value in a few iterations. On the other hand, we can notice that several starting points reached a value that is not a exactly a global minimum after the first optimization, and hence, a series of iterative re-optimization steps are performed in order to reach a better solution. Comparing the previous results, we can clearly see that the best solution obtained at this point consists of $x^* = (1, 0, 0)$, using $x_0 = (1, 0, 0)$ as a starting point - notice that the algorithm does not improve this value - where the objective function is equal to 0.

3. In order to check and compare the previous results with other optimization methods, we define a

wrapper function $OptX()$ for calling the $optimx()$ function, an improved version of the original $optim()$ function. In this case, we perform a simple one-step call of the optimization methods in order to be able to study the results obtained by just one call of the function instead of using an iterative approach (like in the previous section). Therefore, we construct a DataFrame containing all the results from the optimization methods and we display the most relevant ones: optimal point, the optimal value found, and KKT condition check.

Thus, the simple function is as follows (traditional optimization methods are implemented, we included the full list as a comment):

```
# Function for optimizing from a starting point X0 using optimx function
OptX <- function(X0 = c(0, 0, 0)){
  # Supress warnings
  options(warn = -1)

  # Load relevant functions
  source("ps8.R")

  # Possible methods (selection of the most traditional ones)
  # Other options: "spg", "ucminf", "newuoa", "bobyqa", "nmkb",
  #                "hjkb", "Rcgmin", or "Rvmmin"
  Methods <- c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "nlm", "nlminb")

  # Optimize
  print("----------------- Optimization results -----------------")
  print("Initial point:")
  print(X0)
  R <- data.frame(optimx(X0, f, method = Methods, itnmax=NULL))
  print(R[c(1:4, 9, 10)])
}
```

4. After defining the function, we are ready to load the corresponding package and compare the solutions obtained when using different starting points for the optimization methods.

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Load libraries
library("optimx")

# Initial point
X0 = c(0, 0, 0)
X0_2 = c(1, 0, 0)
X0_3 = c(100, 100, 100)
X0_4 = c(1, 0, -1)
X0_5 = c(1, -1, 0)
X0_6 = c(0, 1, 0)
X0_7 = c(0, 0, 1)
X0_8 = c(1, 1, 1)
X0_9 = c(-1, -1, -1)
X0_10 = c(0, -1, 0)
X0_11 = c(1e8,-1e8,100)
InitialP = list(X0, X0_2, X0_3, X0_4, X0_5,
                X0_6, X0_7, X0_8, X0_9, X0_10,
```

```
               X0_11)

# Call the function
for (x0 in InitialP) {
  OptX(x0)
  cat("\n\n")
}
```

```
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1] 0 0 0
##                    p1          p2         p3        value  kkt1  kkt2
## Nelder-Mead 0.9999783 0.002730698 0.00428464 1.876851e-05 FALSE  TRUE
## BFGS        1.0000000 0.000000000 0.00000000 3.155444e-28  TRUE  TRUE
## CG          1.0000000 0.000000000 0.00000000 2.885014e-15  TRUE  TRUE
## L-BFGS-B    0.0000000 0.000000000 0.00000000 1.000000e+02 FALSE FALSE
## nlm         0.0000000 0.000000000 0.00000000 1.000000e+02 FALSE FALSE
## nlminb      0.0000000 0.000000000 0.00000000 1.000000e+02 FALSE FALSE
##
##
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1] 1 0 0
##             p1 p2 p3        value kkt1 kkt2
## Nelder-Mead  1  0  0 0.000000e+00 TRUE TRUE
## BFGS         1  0  0 4.930381e-28 TRUE TRUE
## CG           1  0  0 0.000000e+00 TRUE TRUE
## L-BFGS-B     1  0  0 0.000000e+00 TRUE TRUE
## nlm          1  0  0 0.000000e+00 TRUE TRUE
## nlminb       1  0  0 0.000000e+00 TRUE TRUE
##
##
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1] 100 100 100
##                    p1            p2            p3        value  kkt1 kkt2
## Nelder-Mead 0.9990108  1.564930e-01  2.580923e-01 9.078190e-02 FALSE TRUE
## BFGS        1.0000000 -6.247291e-11 -1.008344e-10 1.114046e-20  TRUE TRUE
## CG          0.9705744  2.320666e-01  3.726632e-01 1.374352e-01 FALSE TRUE
## L-BFGS-B    1.0000000 -1.297339e-07 -9.506358e-08 1.278534e-12  TRUE TRUE
## nlm         1.0000000  2.380007e-09  5.289625e-09 3.800306e-16  TRUE TRUE
## nlminb      1.0000000 -2.681956e-11 -1.281165e-10 7.668135e-19  TRUE TRUE
##
##
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1]  1  0 -1
##                    p1            p2            p3        value  kkt1 kkt2
## Nelder-Mead 0.9999589  5.906569e-04  9.880445e-04 1.374002e-06 FALSE TRUE
## BFGS        1.0000000  3.960969e-12  6.357065e-12 4.441686e-23  TRUE TRUE
## CG          1.0000137 -8.731341e-04 -1.388521e-03 1.908056e-06 FALSE TRUE
## L-BFGS-B    1.0000000  3.715421e-10  1.006048e-09 2.175844e-17  TRUE TRUE
## nlm         1.0000000 -6.124013e-10 -5.080451e-10 1.369096e-16  TRUE TRUE
```

```
## nlminb      1.0000000  5.666358e-11  9.145532e-11 2.763612e-20  TRUE TRUE
##
##
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1]  1 -1  0
##                    p1            p2            p3        value  kkt1 kkt2
## Nelder-Mead 1.0000458 -3.983705e-04 -7.158905e-04 1.393654e-06 FALSE TRUE
## BFGS        1.0000000 -1.699923e-11 -2.698937e-11 7.303745e-22  TRUE TRUE
## CG          0.9999845 -1.282998e-03 -2.050772e-03 4.183200e-06 FALSE TRUE
## L-BFGS-B    1.0000004 -9.961460e-07 -2.074503e-06 4.620931e-11  TRUE TRUE
## nlm         0.9999995 -8.215770e-05 -1.299604e-04 1.697832e-08  TRUE TRUE
## nlminb      1.0000000  1.937863e-10  1.891778e-10 1.485742e-18  TRUE TRUE
##
##
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1] 0 1 0
##                    p1            p2            p3        value  kkt1 kkt2
## Nelder-Mead 0.9998882 -2.040018e-04 -1.412321e-04 4.635507e-06 FALSE TRUE
## BFGS        1.0000000  2.462619e-08  3.901586e-08 1.568233e-15  TRUE TRUE
## CG          0.9784449  2.185358e-01  3.471343e-01 1.206710e-01 FALSE TRUE
## L-BFGS-B    1.0000008  9.617366e-08 -6.348226e-07 1.329042e-10  TRUE TRUE
## nlm         0.9999995 -8.222338e-05 -1.300666e-04 1.700530e-08  TRUE TRUE
## nlminb      1.0000000 -3.165676e-10 -4.056214e-10 2.693780e-18  TRUE TRUE
##
##
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1] 0 0 1
##                    p1            p2            p3        value  kkt1  kkt2
## Nelder-Mead 0.9997112  1.178555e-03  1.921611e-03 1.219607e-05 FALSE  TRUE
## BFGS        1.0000000  7.677536e-13  4.385274e-12 5.000478e-20  TRUE  TRUE
## CG          1.0000056  1.626330e-02  2.605140e-02 6.735098e-04 FALSE  TRUE
## L-BFGS-B    0.9999997  1.055212e-06  1.785767e-06 1.076400e-11  TRUE  TRUE
## nlm         1.0000000 -2.028385e-08 -5.205089e-08 7.685578e-14  TRUE  TRUE
## nlminb      0.0000000  0.000000e+00  1.000000e+00 2.010000e+02 FALSE FALSE
##
##
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1] 1 1 1
##                    p1            p2            p3        value  kkt1 kkt2
## Nelder-Mead 0.9999779 -1.349269e-04 -1.927127e-04 1.343098e-07 FALSE TRUE
## BFGS        1.0000000  4.657335e-13  7.726236e-13 8.169820e-25  TRUE TRUE
## CG          0.9996429  3.340593e-02  5.313813e-02 2.791890e-03 FALSE TRUE
## L-BFGS-B    1.0000000 -6.396348e-08 -1.117770e-07 2.586340e-14  TRUE TRUE
## nlm         0.9999995 -8.225859e-05 -1.301257e-04 1.702065e-08  TRUE TRUE
## nlminb      1.0000000 -1.308309e-10 -1.780827e-10 1.232535e-19  TRUE TRUE
##
##
## [1] "----------------- Optimization results -----------------"
## [1] "Initial point:"
## [1] -1 -1 -1
```

```
##                    p1            p2            p3        value  kkt1 kkt2
## Nelder-Mead 0.9997732  5.162083e-04  9.457322e-04 7.570388e-06 FALSE TRUE
## BFGS        1.0000000  1.295516e-10  2.032027e-10 4.287143e-20  TRUE TRUE
## CG          0.9810031 -2.079314e-01 -3.317339e-01 1.080007e-01 FALSE TRUE
## L-BFGS-B    1.0000000 -1.951190e-07 -3.081557e-07 1.263369e-13  TRUE TRUE
## nlm         0.9999995 -8.223171e-05 -1.300801e-04 1.700915e-08  TRUE TRUE
## nlminb      1.0000000  7.086449e-11  1.162347e-10 1.566368e-20  TRUE TRUE
##
##
## [1] "------------------ Optimization results ------------------"
## [1] "Initial point:"
## [1]  0 -1  0
##                    p1            p2            p3        value  kkt1 kkt2
## Nelder-Mead 1.0000086  2.886151e-04  3.480098e-04 1.368128e-06 FALSE TRUE
## BFGS        1.0000000 -2.462619e-08 -3.901586e-08 1.568233e-15  TRUE TRUE
## CG          0.9784449 -2.185358e-01 -3.471343e-01 1.206710e-01 FALSE TRUE
## L-BFGS-B    1.0000008 -9.617366e-08  6.348226e-07 1.329042e-10  TRUE TRUE
## nlm         1.0000000 -7.526181e-09 -8.792126e-09 6.221899e-15  TRUE TRUE
## nlminb      1.0000000  3.165635e-10  4.056097e-10 2.692834e-18  TRUE TRUE
##
##
## [1] "------------------ Optimization results ------------------"
## [1] "Initial point:"
## [1]  1e+08 -1e+08  1e+02
##                        p1            p2            p3        value  kkt1
## Nelder-Mead 1.484780e+04 -9.694643e+03 -1.234234e+04 4.682417e+10  TRUE
## BFGS        1.000000e+00  3.637214e-10  5.796933e-10 3.362546e-19  TRUE
## CG          9.989102e-01 -9.786303e-03 -1.532267e-02 2.861285e-04 FALSE
## L-BFGS-B    1.000000e+00  3.849017e-07  6.207479e-07 5.108814e-13  TRUE
## nlm         9.999995e-01 -8.223237e-05 -1.300813e-04 1.700942e-08  TRUE
## nlminb      1.000000e+00 -1.757212e-12 -2.393024e-12 1.291467e-21  TRUE
##             kkt2
## Nelder-Mead TRUE
## BFGS        TRUE
## CG          TRUE
## L-BFGS-B    TRUE
## nlm         TRUE
## nlminb      TRUE
```

Based on the results, we can clearly see that again, we have a great convergence of the solutions toward the vector $(1, 0, 0)$ where the value of the objective function is 0. Looking at the helical function defined in $ps8.R$, we can clearly see that the final objective function consists of a summation of quadratic terms and thus, all of them are ≥ 0. Therefore, it is possible to conclude that the optimal (global minimum) objective value of the function under study is 0, reached at the point $(1, 0, 0)$, indicating us that the tested optimization methods tend to have a great convergence toward the global optimal solution of the function.

5. For completeness, we test the $nlm()$ function using the same starting points as before, in order to check its convergence towards the optimal value of the function. In order to do this, we define a new auxiliary variable $OptNLM()$ as a wrapper of the original function:

```
# Define the auxiliary function
OptNLM <- function(X0 = c(0, 0, 0)){
  # Supress warnings
```

```
  options(warn = -1)

  # Load relevant functions
  source("ps8.R")

  # Perform the optimization
  NLMOpt <- nlm(f, X0)

  # Initial Optimization summary
  NewObjValue <- NLMOpt$minimum
  NewPoint <- NLMOpt$estimate

  # Print results
  print("------- Optimization Results -------")
  print("Initial point:")
  print(X0)
  print(paste0("Minimum value obtained: ", NewObjValue))
  print("Optimal Point obtained:")
  print(NewPoint)
}
```

6. Finally, as before, we call the function in an iterative fashion for solving the optimization problem with different starting points:

```
# Set working directory
setwd("C:/Users/chile/Desktop/Stats243/HW/HW8/")

# Load libraries
library("optimx")

# Initial point
X0 = c(0, 0, 0)
X0_2 = c(1, 0, 0)
X0_3 = c(100, 100, 100)
X0_4 = c(1, 0, -1)
X0_5 = c(1, -1, 0)
X0_6 = c(0, 1, 0)
X0_7 = c(0, 0, 1)
X0_8 = c(1, 1, 1)
X0_9 = c(-1, -1, -1)
X0_10 = c(0, -1, 0)
X0_11 = c(1e8,-1e8,100)
InitialP = list(X0, X0_2, X0_3, X0_4, X0_5,
                X0_6, X0_7, X0_8, X0_9, X0_10,
                X0_11)

# Call the function
for (x0 in InitialP) {
  OptNLM(x0)
  cat("\n\n")
}
```

```
## [1] "------- Optimization Results -------"
```

```
## [1] "Initial point:"
## [1] 0 0 0
## [1] "Minimum value obtained: 100"
## [1] "Optimal Point obtained:"
## [1] 0 0 0
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 1 0 0
## [1] "Minimum value obtained: 0"
## [1] "Optimal Point obtained:"
## [1] 1 0 0
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 100 100 100
## [1] "Minimum value obtained: 3.80030638958706e-16"
## [1] "Optimal Point obtained:"
## [1] 1.000000e+00 2.380007e-09 5.289625e-09
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1]  1  0 -1
## [1] "Minimum value obtained: 1.36909646371849e-16"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -6.124013e-10 -5.080451e-10
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1]  1 -1  0
## [1] "Minimum value obtained: 1.69783168305e-08"
## [1] "Optimal Point obtained:"
## [1]  0.9999994969 -0.0000821577 -0.0001299604
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 0 1 0
## [1] "Minimum value obtained: 1.70052975436292e-08"
## [1] "Optimal Point obtained:"
## [1]  9.999995e-01 -8.222338e-05 -1.300666e-04
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 0 0 1
## [1] "Minimum value obtained: 7.685578317637e-14"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -2.028385e-08 -5.205089e-08
##
```

```
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] 1 1 1
## [1] "Minimum value obtained: 1.70206475290325e-08"
## [1] "Optimal Point obtained:"
## [1]  9.999995e-01 -8.225859e-05 -1.301257e-04
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1] -1 -1 -1
## [1] "Minimum value obtained: 1.70091484420005e-08"
## [1] "Optimal Point obtained:"
## [1]  9.999995e-01 -8.223171e-05 -1.300801e-04
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1]  0 -1  0
## [1] "Minimum value obtained: 6.22189931142631e-15"
## [1] "Optimal Point obtained:"
## [1]  1.000000e+00 -7.526181e-09 -8.792126e-09
##
##
## [1] "------- Optimization Results -------"
## [1] "Initial point:"
## [1]  1e+08 -1e+08  1e+02
## [1] "Minimum value obtained: 1.70094177266386e-08"
## [1] "Optimal Point obtained:"
## [1]  9.999995e-01 -8.223237e-05 -1.300813e-04
```

By looking at the results, we can easily check that the results are consistent and coherent with the ones obtained by our previous implementations, where the optimization methods tend to converge to the optimal point $(1, 0, 0)$ from any of the tested starting points with different convergence rates (with a special exception with the origin, where the $nlm()$ function is not able to improve the original solution). Therefore, we have analyzed and studied the proposed helical function both using visual and optimization approaches.

Problem 3: censored regression problem

In this problem, we want to consider a simple linear censored regression problem with the functional form of:

$$Y_i \to N(\beta_0 + \beta_1 x_i, \sigma) \tag{3}$$
$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i \;\; \epsilon \to N(0, \sigma) \tag{4}$$

We will assume that we have an iid sample, but that for any observation with $Y > \tau$ we know that Y exceeded the threshold and not its actual value. Following the example of the problem statement, we have that c of the n observations will be censored (stochastically) depending on how many of them exceed the fixed value τ.

For the rest of the problem, we define $m = n - c$ with c the number of censored values. Without loss of generality, we sort the values of the original y vector such that the first m components are the uncensored/observed values while the $m+1, ..., n$ components consist of the censored values. Thus, we will denote the censored values by $z = (z_{m+1}, ..., z_n)$ and by $y = (y_1, ..., y_m)$ the observed components.

Therefore, for simplicity, we can define:

$$y_i^* = \begin{cases} y_i & i \leq m \\ z_i & i > m \end{cases} \tag{5}$$

The main objective of this problem is to propose, design, and implement a EM algorithm in order to estimate the three main parameters of the model: $\theta = (\beta_0, \beta_1, \sigma)$, while taking the complete data to be available in addition to the censored observations (actual values).

a) EM algorithm

In order to develop our EM algorithm, we need to follow a series of steps that will allow us to formulate all the needed expressions needed for calculating the complete log-likelihood of the sampled data, the expected log-likelihood using all observations, and find the optimal values (maximization) of the *theta* parameters per iteration of the algorithm.

Step 1: Complete-data log-likelihood

The first step consists of calculating the complete-data log-likelihood. In this case, we do not take into account the fact that the z_i $i \in \{m+1, ..., n\}$ values are censored. First, we calculate the complete-data likelihood as follows:

$$
\begin{aligned}
L^c(\theta/y^*) &= \prod_{i=1}^{n} f(y_i^*) && (6)\\
&= \prod_{i=1}^{m} f(y_i) \prod_{j=m+1}^{n} f(z_j) && (7)\\
&= \prod_{i=1}^{m} \phi\left(\frac{y_i - x_i^{'}\beta}{\sigma}\right)\sigma^{-1} \prod_{j=m+1}^{n} \phi\left(\frac{\tau - x_j^{'}\beta}{\sigma}\right)\sigma^{-1} && (8)\\
&= \sigma^{-n}\prod_{i=1}^{m} \phi\left(\frac{y_i - x_i^{'}\beta}{\sigma}\right) \prod_{j=m+1}^{n} \phi\left(\frac{\tau - x_j^{'}\beta}{\sigma}\right) && (9)\\
&= \sigma^{-n}\prod_{i=i}^{m} \frac{1}{\sqrt{2\pi}} e^{\frac{-(y_i - x_i^{'}\beta)^2}{2\sigma^2}} \prod_{j=m+1}^{n} \frac{1}{\sqrt{2\pi}} e^{\frac{-(z_j - x_j^{'}\beta)^2}{2\sigma^2}} && (10)\\
&= \sigma^{-n}\frac{1}{(2\pi)^{n/2}}\prod_{i=i}^{m} e^{\frac{-(y_i - x_i^{'}\beta)^2}{2\sigma^2}} \prod_{j=m+1}^{n} e^{\frac{-(z_j - x_j^{'}\beta)^2}{2\sigma^2}} && (11)
\end{aligned}
$$

Notice that we are using the standard density function for an $N(0,1)$ by standardizing the original distribution. In addition, we expanded the terms for simplicity when calculating the complete-data log-likelihood function.

Now, we apply the logarithmic function to the previous expression as follows:

$$
\begin{aligned}
\log(L^c(\theta/y^*)) &= \underbrace{-\frac{n}{2}\log(2\pi)}_{C} - n\log(\sigma) - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - x_i^{'}\beta)^2 - \frac{1}{2\sigma^2}\sum_{j=m+1}^{n}(z_j - x_j^{'}\beta)^2 && (12)\\
&= C - n\log(\sigma) - \frac{1}{2\sigma^2}\left(\sum_{i=1}^{m}(y_i - x_i^{'}\beta)^2 + \sum_{j=m+1}^{n}(z_j - x_j^{'}\beta)^2\right) && (13)\\
&= C - n\log(\sigma) - \frac{1}{2\sigma^2}\left(\sum_{i=1}^{n}(y_i^* - x_i^{'}\beta)^2\right) && (14)\\
& && (15)
\end{aligned}
$$

Looking at the previous expression, we can clearly see that is very similar to a classic linear regression model with the main difference that in this case, we have a set of censored values z_j for which we do not know their real values, only that they satisfy $z_j > \tau \; j \in \{m+1, ..., n\}$.

Now, we are ready to go to the next step and starting with the relevant calculations for the EM algorithm.

Step 2: Expected complete-data log-likelihood

In order to use an EM algorithm approach, we need to calculate the expected value of the previous log-likelihood function, given that we have a vector of estimated parameters θ_t for the iteration t of the algorithm and a vector x of observations. Therefore, we want to find:

$$
Q(\theta/\theta_t) = \mathbf{E}[\log(L^c(\theta/y^*))|x, \theta_t] \quad (16)
$$

This is known as the E-step in the EM algorithm. Notice that we are interested in calculating the expectation over the missing data in a closed form. In addition, we have to be extremely careful to separate the values of *theta* (that we want to optimize) from the values of $theta_t$ (already estimated).

In order to calculate this expression, we proceed as follows:

$$
\begin{aligned}
Q(\theta/\theta_t) &= \mathbf{E}[\log(L^c(\theta/y^*))|x,\theta_t] && (17)\\
&= \mathbf{E}\left[C - n\log(\sigma) - \frac{1}{2\sigma^2}\left(\sum_{i=1}^{m}(y_i - x_i^{'}\beta)^2 + \sum_{j=m+1}^{n}(z_j - x_j^{'}\beta)^2\right)|x,\theta_t\right] && (18)\\
&= C - n\log(\sigma) - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - x_i^{'}\beta)^2 - \frac{1}{2\sigma^2}\sum_{j=m+1}^{n}\underbrace{\mathbf{E}\left[(z_j - x_j^{'}\beta)^2|x,\theta_t\right]}_{A} && (19)
\end{aligned}
$$

Now, we can focus on the A expression since we need to calculate the expected value of the censored components of the data, given that we know the x vector, the $theta_t$ vector, and that zi follows a truncated normal distribution by τ.

In order to calculate this expression, we expand its components and we will use the known results and properties from both the expectation and the truncated normal distribution:

$$
\begin{aligned}
\mathbf{E}\left[(z_j - x_j^{'}\beta)^2|x,\theta_t\right] &= \mathbf{E}\left[z_j^2 - 2z_j x_j^{'}\beta + (x_j^{'}\beta)^2|x,\theta_t\right] && (20)\\
&= \underbrace{\mathbf{E}\left[z_j^2|x,\theta_t\right]}_{B} - 2\underbrace{\mathbf{E}\left[z_j|x,\theta_t\right]}_{C}x_j^{'}\beta + (x_j^{'}\beta)^2 && (21)
\end{aligned}
$$

Now, we solve B:

$$
\begin{aligned}
\mathbf{E}\left[z_j^2|x,\theta_t\right] &= \mathbf{E}\left[z_j^2|z_j > \tau, x, \theta_t\right] && (22)\\
&= \mathbf{Var}\left[z_j|z_j > \tau\right] + \left(\mathbf{E}\left[z_j|z_j > \tau\right]\right)^2 && (23)\\
&= \sigma_t^2(1 + \tau_{t,j}^*\rho(\tau_{t,j}^*) - \rho(\tau_{t,j}^*)^2) + (\mu_{t,j} + \sigma_t\rho(\tau_{t,j}^*))^2 && (24)
\end{aligned}
$$

And for C:

$$
\begin{aligned}
\mathbf{E}\left[z_j|x,\theta_t\right] &= \mathbf{E}\left[z_j|z_j > \tau, x, \theta_t\right] && (25)\\
&= \mathbf{E}\left[z_j|z_j > \tau\right] && (26)\\
&= \mu_{t,j} + \sigma_t\rho(\tau_{t,j}^*) && (27)
\end{aligned}
$$

where $\mu_{t,j} = \beta_0^t + \beta_1^t x_j$, $\tau_{t,j}^* = \dfrac{(\tau - \mu_{t,j})}{\sigma_t}$, and $\rho(\tau_{t,j}^*) = \dfrac{\phi(\tau_{t,j}^*)}{\left(1 - \Phi(\tau_{t,j}^*)\right)}$

Having these values and re-arranging the terms by noting the fact that we have an analog expression to the classic quadratic difference term, we can finally obtain the expression for $Q(\theta/\theta_t)$:

$$
Q(\theta/\theta_t) = C - n\log(\sigma) - \frac{1}{2\sigma^2}\left(\sum_{i=1}^{m}(y_i - x_i^{'}\beta)^2 + \sum_{j=m+1}^{n}\left((\mathbf{E}[z_j] - x_j^{'}\beta)^2 + \mathbf{Var}[z_j]\right)\right) \quad (28)
$$

Looking at the previous expression, we can clearly see that we have: (1) the same expressions as for a classic regression for the observed data, (2) a similar expression for the censored data where the values are replaced by the conditional expectation (given that $z_j > \tau$ of the variable z_j (omitted in the expected value and variance of the previous equation for simplicity), and (3) an extra term including the variance of the censored observations.

Thus, we can easily express (as mentioned in the problem statement) the Q function as a regression term plus the variance of the censored elements as follows:

$$Q(\theta/\theta_t) = C - n\log(\sigma) - \frac{1}{2\sigma^2}\left(\sum_{i=1}^{n}(\delta_i - x_i^{'}\beta)^2 + \sum_{j=m+1}^{n}\mathbf{Var}[z_j]\right) \tag{29}$$

where $\delta_i = \begin{cases} y_i & i \leq m \\ \mathbf{E}[z_i] & i > m \end{cases}$

Step 3: Optimal parameters expressions

Finally, we need to calculate the expressions for calculating the optimal parameters $\theta = (\beta_0, \beta_1, \sigma)$ such that the previous function is maximized. Clearly, the optimal $\hat{\beta}$ values can be easily obtained by performing a simple linear regression using adapted vectors $Y = (y_{obs}, y_{cen}) = (y, \mathbf{E}[z])$ and $X = (x_{obs}, x_{cen}) = x$ (assuming w.l.o.g. that the values are sorted). Therefore, we are simply solving a least squares problem in order to obtain both coefficients (check appendix for the specific expressions obtained by direct derivation of the Q function).

On the other hand, we need to explicitly solve the maximization problem for σ by taking the derivative of the Q function with respect to σ (or σ^2 if wanted). Thus, we obtain the following expression:

$$\frac{\partial Q(\theta/\theta_t)}{\partial\sigma} = -\frac{n}{\sigma} + \frac{1}{\sigma^3}\left(\sum_{i=1}^{m}(y_i - x_i^{'}\beta)^2 + \sum_{j=m+1}^{n}(z_j - x_j^{'}\beta)^2 + \sum_{j=m+1}^{n}\mathbf{Var}[z_j]\right) \tag{30}$$

$$= 0 \tag{31}$$

Solving for σ we get:

$$\sigma^{2^*} = \frac{1}{n}\left(\sum_{i=1}^{m}(y_i - x_i^{'}\beta)^2 + \sum_{j=m+1}^{n}(z_j - x_j^{'}\beta)^2 + \sum_{j=m+1}^{n}\mathbf{Var}[z_j]\right) \tag{32}$$

Therefore, at this point, we have all the expressions needed for developing our EM algorithm.

Algorithm

The main logic of the algorithm is very simple: starting from a particular point for the optimization θ_0, a series of iterations where the E and the M steps will be performed, obtaining a new vector θ_t during iteration t such that the value of the Q function is maximized (and thus, our log-likelihood function). The algorithm will iterate until a convergence threshold is satisfied like a difference between consecutive values of the objective function or the estimated parameter values or by a maximum number of possible iterations.

The proposed algorithm and its pseudo-code are the followings:

0) Global parameters are specified: Maximum number of iterations ($MaxIters$), convergence tolerance (ϵ), empty list for the θ_t results, and a convergence boolean flag ($Convergence$) for checking the final status of the algorithm.

1) We start our EM algorithm by generating/separating the data depending if the observation is censored or not. In the case that the data is generated/simulated, we define a certain threshold τ for censoring it.

2) Once the data is separated by $Y = (y, z)$ and $X = (x_{obs}, x_{cen})$, we compute an initial (reasonable) starting point. In order to perform this step, a simple linear regression will be developed with the observed data such that the coefficients β_0, β_1 are initialized with these estimations and also, computing the standard deviation of the observed sample will be the selected value for initializing the σ parameter. Hence, we initialize the algorithm with θ_0 with the values described before.

3) Once the initialization is done, the main loop of the algorithm begins. Using vectorized operations in R, all the relevant parameters needed for the calculations of the expected values, variances, and optimal θ_t parameters with t the current iteration will be computed: $\mu_{t,j}$, $\tau^*_{t,j}$, $\rho(\tau^*_{t,j})$, $\mathbf{E}[z_j]$, $\hat{\beta^{t}}^*$, σ^*_t, and the $Log - likelihood$ function for convergence purposes.

4) Breaking conditions will be checked after every iteration. A convergence checking based on the tolerance parameter will be performed by comparing the value of the objective function between two consecutive iterations. If the absolute change in its value is less than the specified tolerance, the algorithm breaks. In addition, the algorithm breaks if the maximum number of iterations is reached before achieving the convergence (base on the ϵ value) to a particular solution.

5) If no breaking condition is satisfied, return to step 3.

Thus, we can see that the algorithm is very simple and clear, something that will be helpful for developing it as a function in R in section c.

Based on the previous discussion of the main steps of the algorithms, we can explicitly develop the pseudo-code of it as follows:

Therefore, we will implement this algorithm in R in section (c) of the current problem.

b) Starting values: based on observations

As seen in the previous section, the most reasonable values for starting the algorithm for β_0, β_1, and σ are:

i) (β_0, β_1): We can easily compute the coefficients associated with the linear regression over the observed values y_{obs}, x_{obs}. By a simple call of the $lm()$ function in R, we can estimate the $\hat{beta}$ coefficients and use them as natural starting points for our EM optimization algorithm.

ii) σ: Following the logic of the previous section, we can initialize the variable of σ by simply computing the standard deviation of the observed values y_{obs} from the sample.

Therefore, we will use the following starting point:

$$\begin{aligned} \theta_0 &= (\hat{\beta}_0, \hat{\beta}_1, \hat{\sigma}) && (33) \\ &= (0.1440021, 1.928731, 1.683601) && (34) \end{aligned}$$

Algorithm 1 EM algorithm for upper censored regression

procedure STEP 0: INITIALIZATION
  $MaxIters, \epsilon, n, \tau, Convergence, \theta_t, LL_t$ # Global Parameters
procedure STEP 1: DATA GENERATION/IMPORTATION
  if No Data **then**
    $X_{Obs}, Y_{Obs}, X_{Cen}, Y_{Cen} \leftarrow GenerateData(n, \tau)$ # Generate random data
procedure STEP 2: INITIAL STARTING POINT
  $\theta_0 \leftarrow StartingPoint(X_{obs}, Y_{obs})$ # Starting point using observed data.
procedure STEP 3: MAIN LOOP
  $t \leftarrow 1$ # Current iteration
  while TRUE **do**
    $\beta_t \leftarrow ComputeBetas(X, Y, \tau, \theta_{t-1})$ # New β vector
    $\sigma_t \leftarrow ComputeSigma(X, Y, \beta_t)$ # New σ value
    $\theta_t \leftarrow (\beta_t, \sigma_t)$ # Update solutions
    $LL_t \leftarrow CalculateLL(\theta_t, \tau, X, Y)$ # New objective value
    if $SolConverged(\epsilon, LL_t, LL_{t-1})$ **then** # Check if solution converged
      $\theta^* \leftarrow \theta_t$ # Update the best solution
      $LL^* \leftarrow LL(\theta^*, X, Y, \tau)$
      $Convergence \leftarrow TRUE$
      break
    if $t > MaxIters$ **then** # Check max number of iterations
      break
procedure STEP 4: RETURN RESULTS
  if $Convergence == TRUE$ **then** # If convergence is reached, report optimal
    return$(\theta^*, Convergence)$
  else
    return$(\theta_t, Convergence)$ # Else, best solution

c) R function and tests

Based on the pseudo-code and deep explanation of the algorithm included in the previous section, we will perform the following actions for implementing it:

i) A series of auxiliary functions will be created in order to easily perform iterative tasks while trying to minimize the source of error of the calculations along the same code.

ii) Using these functions, a simple adaptation of the linear regression model function $lm()$ will be utilized in order to estimate the optimal β vector. Then, these values will be used to calculate the optimal σ value for the iteration.

iii) Convergence will be checked as well as the number of iterations performed. If any breaking condition is satisfied, break, otherwise, we go to the next iteration.

iv) Following the statement of the problem, the data will be generated based on the $ps8.R$ file, testing two main instances with 20% and 80% of censored observations (associated with $\tau = 4$ and $\tau = 0$ after some preliminary tests with the code).

Thus, we developed the following code:

1. Based on the code provided in the $ps8.R$ file, we create an auxiliary function $GenData()$ for generating the random data from the normal distribution, including a threshold τ for determining whether the observation is censored or not. The generated data is separated by (y, z) (observed and censored) in order to be able to perform all the following operations of the algorithm in a simple way.

```
GenData <- function(n=100, Tau=0){
  # Code from ps8.R file AS-IS
  set.seed(1)
  beta0 <- 1
  beta1 <- 2
  sigma2 <- 6

  x <- runif(n)
  yComplete <- rnorm(n, beta0 + beta1*x, sqrt(sigma2))

  ## parameters chose such that signal in data is moderately strong
  ## estimate divided by std error is ~ 3
  mod <- lm(yComplete ~ x)
  summary(mod)$coef
  # End of original code

  # Censored and observed values: separation
  yCen = yComplete[yComplete > Tau]
  yObs = yComplete[yComplete <= Tau]
  XCen = x[yComplete > Tau]
  XObs = x[yComplete <= Tau]

  # Return all vectors
  return(list(yCen, yObs, XCen, XObs))
}

# Test the function
Data <- GenData(Tau = 4)
```

```r
yCen <- Data[[1]]
yObs <- Data[[2]]
XCen <- Data[[3]]
XObs <- Data[[4]]

# Check lengths
print(length(yCen))
```

```
## [1] 20
```

```r
print(length(yObs))
```

```
## [1] 80
```

2. An auxiliary $StartingP()$ function is defined in order to initialize the $theta_0$ vector following the previous discussion.

```r
# Calculate a good starting point based on the data
StartingP <- function(XObs, yObs){
  # Estimate betas and sigma from non-censored values
  Betas = unname(lm(yObs ~ XObs)$coefficients)
  Sigma = sd(yObs)
  unname(Betas, force = TRUE)

  return(c("Beta0" = Betas[1],
           "Beta1" = Betas[2],
           "Sigma" = Sigma))
}

# Test the function
theta0 <- StartingP(XObs, yObs)
print("Starting parameters:")
```

```
## [1] "Starting parameters:"
```

```r
theta0
```

```
##     Beta0     Beta1     Sigma
## 0.1440021 1.9287315 1.6836006
```

3. We define an auxiliary function for computing the mu values.

```r
# Compute the value of mu per iteration
muT <- function(X, thetas_t){
  # Return the mu value
  return(thetas_t["Beta0"] + thetas_t["Beta1"] * X)
}

# Test the function
Mu_T <- muT(XCen, theta0)
Mu_T
```

```
##  [1] 0.5329924 1.8767545 0.5412717 1.4690847 2.0571226 1.6434852 1.9467975
##  [8] 1.0959107 1.7397830 1.6758775 1.3920074 1.6540691 1.1656889 0.3358456
## [15] 1.1443083 0.9286683 1.0294166 1.7630571 1.8648131 1.0221043
```

4. Following the same logic, we define an auxiliary function for calculating the τ^* values.

```
TauT <- function(theta_t, mu_t, Tau=0){
  # Return the standardized value
  return( (Tau - mu_t) / theta_t["Sigma"] )
}

# Test the function
Tau_T <- TauT(theta0, Mu_T, 4)
Tau_T
```

```
##  [1] 2.059282 1.261134 2.054364 1.503275 1.154001 1.399688 1.219531
##  [8] 1.724928 1.342490 1.380448 1.549057 1.393401 1.683482 2.176380
## [15] 1.696181 1.824264 1.764423 1.328666 1.268226 1.768766
```

5. Same with $\rho(\tau^*)$:

```
rhoT <- function(Tau_t){
  # Calculate the pdf and cdf, return the specific ratio
  pdf = dnorm(x = Tau_t, mean = 0, sd = 1)
  cdf = pnorm(q = Tau_t, mean = 0, sd = 1)
  return(pdf / (1 - cdf))
}

# Test the function
Rho_T <- rhoT(Tau_T)
Rho_T
```

```
##  [1] 2.425825 1.738039 2.421453 1.941463 1.649815 1.853794 1.703641
##  [8] 2.131968 1.805792 1.837614 1.980496 1.848504 2.096063 2.530327
## [15] 2.107051 2.218516 2.166297 1.794235 1.743921 2.170079
```

6. We define the $Yexp()$ function for calculating the expected value of the censored values, needed for optimizing the Q function.

```
Yexp <- function(mu_t, theta_t, rho_t){
  # Compute the expected value of the censored components
  yexp <- mu_t + theta_t["Sigma"] * rho_t
  return(unname(yexp))
}

# Test the function
YExp_T <- Yexp(Mu_T, theta0, Rho_T)
YExp_T
```

```
##  [1] 4.617113 4.802918 4.618032 4.737733 4.834752 4.764534 4.815048
##  [8] 4.685293 4.780015 4.769686 4.726372 4.766211 4.694622 4.595906
## [15] 4.691741 4.663763 4.676595 4.783832 4.800879 4.675650
```

7. A very simple wrapper to the $lm()$ function is defined in order to perform the linear regression with the observed and conditional expected values of the censored components of the data in order to estimate the best β vector for the iteration t.

```
lmWrap <- function(Xobs, Xcen, yobs, yexp){
  # Create the full vectors
  X <- c(Xobs, Xcen)
  Y <- c(yobs, yexp)

  # Calculate the regression model
  model <- lm(Y ~ X)

  # Coefficents
  BetasNew <- model$coefficients
  names(BetasNew) <- c("Beta0", "Beta1")

  # Return coefficients
  return(BetasNew)
}

# Test the function
Betas_New <- lmWrap(XObs, XCen, yObs, YExp_T)
Betas_New
```

```
##     Beta0     Beta1
## 0.4647043 2.6152550
```

8. Direct calculation of the square difference term (including observed and censored components) is performed via the function $ATerm()$.

```
ATerm <- function(xobs, xcen, yobs, mu_t, rho_t, BetasNew, thetas_t){
  # Calculate the observed and censored terms of the expression
  Obs <- (yobs - BetasNew["Beta0"] - BetasNew["Beta1"] * xobs) ^ 2
  Cen <- (Yexp(mu_t, thetas_t, rho_t) - BetasNew["Beta0"] - BetasNew["Beta1"] * xcen) ^ 2

  # Return total summation
  return(sum(Obs) + sum(Cen))
}

# Test the function
AVal_T <- ATerm(XObs, XCen, yObs, Mu_T, Rho_T, Betas_New, theta0)
AVal_T
```

```
## [1] 386.6646
```

9. Similarly, the summation of the variances of the censored components is performed by calling the function $BTerm()$.

```
BTerm <- function(theta_t, rho_t, tau_t){
  # Calculate the expression
  B <- (theta_t["Sigma"] ^ 2) * sum(1 + tau_t * rho_t - rho_t ^ 2)
```

```
  # Return total summation
  return(unname(B))
}

# Test the function
BVal_T <- BTerm(theta0, Rho_T, Tau_T)
BVal_T
```

```
## [1] 8.249481
```

10. The *SigmaOpt*() function will find the optimal σ value for the iteration based on the expressions that we found in the previous sections.

```
SigmaOpt <- function(n, A, B){
  # Return the optimal value (from derivative)
  return(sqrt((A + B) / n ))
}

# Test the function
n <- 100
SigmaNew <- SigmaOpt(n, AVal_T, BVal_T)
SigmaNew
```

```
## [1] 1.987244
```

11. Finally, we define the *LLval*() function for keeping track of the current objective value.

```
LLval <- function(A, B, n, ThetasNew){
  # Calculate the expression by pieces
  Exp1 <- - n * log(ThetasNew["Sigma"])
  Exp2 <- - (1 / (2 * (ThetasNew["Sigma"] ^ 2))) * (A + B)

  # Return final summation
  return(unname(Exp1 + Exp2))
}

# Test the function
ThetasNew <- c(Betas_New, "Sigma" = SigmaNew)
LLVal_T <- LLval(AVal_T, BVal_T, n, ThetasNew)
ThetasNew
```

```
##     Beta0     Beta1     Sigma
## 0.4647043 2.6152550 1.9872444
```

```
LLVal_T
```

```
## [1] -118.6749
```

12. Based on all the previous analysis and defined functions, we have the main EM function. It follows the algorithm already discussed and presented as pseudo-code in the previous sections: (1) Data is generated, (2) Global parameters are defined, and a (3) Reasonable starting point is initialized.

```
EMOpt <- function(Tau, n=100, MaxIters=1e2, Epsilon=1e-3){
  # Generate Data
  Data <- GenData(n, Tau)
  yCen <- Data[[1]]
  yObs <- Data[[2]]
  XCen <- Data[[3]]
  XObs <- Data[[4]]

  # Check % of censored
  print(paste0("Number of censored observations (out of ", n, "): ", length(yCen)))

  # Global Parameters
  # List containing the Theta Vectors and vector with LL values
  Thetas_T <- rep(list(c(0,0,0)), MaxIters)
  LL_T <- rep(0, MaxIters)

  # Initial Theta values: Call the starting point generation function
  theta0 <- StartingP(XObs, yObs)

  # Auxiliary counter for number of iteration
  niter <- 1

  # Convergence flag (True is converged)
  Convergence <- FALSE

  # Pre-loop
  print("Starting point for the EM algorithm, Theta_0:")
  print(theta0)

  # Record initial values inside the list
  Thetas_T[[niter]] <- theta0
```

The main loop of the algorithm: all elements needed are calculated iteratively in order to estimate the optimal β and σ parameters for the current iteration. Values are recorded.

```
  # Main loop (while true until break condition(s) reached)
  while(0==0){
    ## Iteration info
    print(paste0("---------------- Iteration ", niter, " ----------------"))
    print(paste0("Theta_", niter - 1, ":"))
    print(Thetas_T[[niter]])

    ## Step 1: Calculate the Optimal Betas and Sigma
    # Betas: Compute all the expressions and solve an adapted linear regression model
    Mu_T <- muT(XCen, Thetas_T[[niter]])
    Tau_T <- TauT(Thetas_T[[niter]], Mu_T, Tau)
    Rho_T <- rhoT(Tau_T)
    YExp_T <- Yexp(Mu_T, Thetas_T[[niter]], Rho_T)

    Betas_New <- lmWrap(XObs, XCen, yObs, YExp_T)

    # Sigma: Compute all the expressions using Betas_T
    AVal_T <- ATerm(XObs, XCen, yObs, Mu_T, Rho_T, Betas_New, Thetas_T[[niter]])
```

```
  BVal_T <- BTerm(Thetas_T[[niter]], Rho_T, Tau_T)
  Sigma_New <- SigmaOpt(n, AVal_T, BVal_T)

  # Record new values
  NewVals = c(Betas_New["Beta0"],
              Betas_New["Beta1"],
              "Sigma" = Sigma_New)

  # Display new values:
  print(paste0("Optimal Theta_", niter," values obtained:"))
  print(NewVals)
```

The objective function is calculated and the vector containing all the solutions per iteration is updated.

```
  ## Step 2: Compute the LL function value
  LL_T[niter] <- LLval(AVal_T, BVal_T, n, NewVals)
  print("Log-likelihood value:")
  print(LL_T[niter])

  ## Step 3: Update Results
  Thetas_T[[niter + 1]] <- NewVals
```

Break conditions are tested: Convergence and a maximum number of iterations. If none of them is $TRUE$, go to the next iteration and repeat the procedure.

```
  ## Step 4: Check breaking conditions
  # Change of the objective function
  if (niter > 1){
    if (abs(LL_T[niter - 1] - LL_T[niter]) <= Epsilon) {
      print("Convergence of the objective function reached, break")
      Convergence <- TRUE
      break
    }
  }

  # Maximum number of iterations
  if (niter >= MaxIters){
    print("Maximum number of iterations reached, break")
    break
  }

  # If no break, next iteration
  niter <- niter + 1

}
```

After finishing the algorithmic steps, a final summary table is printed out to the console in order to allow the user to check the evolution of the algorithm as well as check all the relevant information of the EM approach. The DataFrame is returned to the user.

```
  # Final Summary
  SummaryDF <- data.frame("Iteration" = seq(1, niter, 1),
                          "Beta0" = unlist(Thetas_T[1:niter])[seq(1, (niter) * 3, 3)],
                          "Beta1" = unlist(Thetas_T[1:niter])[seq(2, (niter) * 3, 3)],
                          "Sigma" = unlist(Thetas_T[1:niter])[seq(3, (niter) * 3, 3)],
                          "LL_Val" = LL_T[1:niter])

  print("---------- Summary Results ----------")
  SummaryDF
  print(paste0("Convergence Status: ", Convergence))
  print(paste0("Number of iterations: ", niter ))

  # Return solution
  return(SummaryDF)
}
```

Therefore, we are ready to test the function using $\tau = 4$ (20% of censored data) and $\tau = 0$ (80% of censored data). **Note:** We added a *verbose* flag to the previous code in order to not print-out all the information per iteration on the console for visualization purposes, just showing the final summary DataFrame in the following tests.

```
Result1 <- EMOpt(Tau=4, n=100, MaxIters=1e2, Epsilon=1e-3, verbose=FALSE)
```

```
## [1] "---------- Summary Results ----------"
## [1] "Convergence Status: TRUE"
## [1] "Number of iterations: 10"
```

```
print(Result1)
```

```
##    Iteration     Beta0    Beta1    Sigma    LL_Val
## 1          1 0.1440021 1.928731 1.683601 -118.6749
## 2          2 0.4647043 2.615255 1.987244 -124.0883
## 3          3 0.4574438 2.763560 2.097788 -125.6935
## 4          4 0.4565279 2.804186 2.131732 -126.1879
## 5          5 0.4563369 2.816478 2.142298 -126.3420
## 6          6 0.4562844 2.820293 2.145601 -126.3902
## 7          7 0.4562685 2.821484 2.146636 -126.4053
## 8          8 0.4562636 2.821857 2.146960 -126.4100
## 9          9 0.4562621 2.821974 2.147062 -126.4115
## 10        10 0.4562616 2.822011 2.147093 -126.4120
```

Based on the results we can see that the algorithm converges after 10 iterations (very fast), obtaining an optimal vector θ_{10}. Now, we solve the optimization problem for the instance with 80% of censored values ($\tau = 0$):

```
Result2 <- EMOpt(Tau=0, n=100, MaxIters=2e2, Epsilon=1e-3, verbose=FALSE)
```

```
## [1] "---------- Summary Results ----------"
## [1] "Convergence Status: TRUE"
## [1] "Number of iterations: 102"
```

```
print(Result2)
```

```
##     Iteration       Beta0     Beta1     Sigma     LL_Val
## 1           1 -1.29215258 0.3748579 0.9847210  -33.58610
## 2           2 -0.25158171 0.8103841 0.8486240  -46.39504
## 3           3 -0.19809920 1.1116684 0.9645925  -56.36676
## 4           4 -0.15275450 1.3229792 1.0657381  -64.17798
## 5           5 -0.11073678 1.4886452 1.1523228  -70.49370
## 6           6 -0.07314555 1.6264668 1.2274477  -75.72439
## 7           7 -0.03980432 1.7446852 1.2933605  -80.13484
## 8           8 -0.01019749 1.8478628 1.3516802  -83.90520
## 9           9  0.01619532 1.9389018 1.4036163  -87.16345
## 10         10  0.03982227 2.0198356 1.4501028  -90.00373
## 11         11  0.06105444 2.0921844 1.4918804  -92.49735
## 12         12  0.08019749 2.1571351 1.5295498  -94.69956
## 13         13  0.09750499 2.2156435 1.5636074  -96.65410
## 14         14  0.11318951 2.2684961 1.5944694  -98.39615
## 15         15  0.12743124 2.3163511 1.6224891  -99.95442
## 16         16  0.14038449 2.3597667 1.6479699 -101.35265
## 17         17  0.15218263 2.3992213 1.6711743 -102.61071
## 18         18  0.16294188 2.4351285 1.6923314 -103.74532
## 19         19  0.17276412 2.4678486 1.7116422 -104.77076
## 20         20  0.18173927 2.4976975 1.7292842 -105.69922
## 21         21  0.18994704 2.5249537 1.7454148 -106.54127
## 22         22  0.19745841 2.5498637 1.7601741 -107.30606
## 23         23  0.20433682 2.5726467 1.7736874 -108.00159
## 24         24  0.21063914 2.5934985 1.7860669 -108.63487
## 25         25  0.21641652 2.6125943 1.7974135 -109.21207
## 26         26  0.22171505 2.6300914 1.8078183 -109.73867
## 27         27  0.22657637 2.6461315 1.8173632 -110.21949
## 28         28  0.23103818 2.6608422 1.8261226 -110.65887
## 29         29  0.23513463 2.6743390 1.8341639 -111.06066
## 30         30  0.23889675 2.6867266 1.8415481 -111.42829
## 31         31  0.24235274 2.6980996 1.8483308 -111.76488
## 32         32  0.24552828 2.7085443 1.8545625 -112.07320
## 33         33  0.24844677 2.7181389 1.8602893 -112.35576
## 34         34  0.25112954 2.7269547 1.8655531 -112.61482
## 35         35  0.25359607 2.7350567 1.8703923 -112.85244
## 36         36  0.25586417 2.7425043 1.8748420 -113.07046
## 37         37  0.25795012 2.7493513 1.8789340 -113.27057
## 38         38  0.25986880 2.7556475 1.8826977 -113.45429
## 39         39  0.26163385 2.7614378 1.8861598 -113.62302
## 40         40  0.26325776 2.7667637 1.8893450 -113.77801
## 41         41  0.26475196 2.7716631 1.8922756 -113.92042
## 42         42  0.26612695 2.7761706 1.8949723 -114.05129
## 43         43  0.26739235 2.7803181 1.8974539 -114.17159
## 44         44  0.26855698 2.7841345 1.8997379 -114.28219
## 45         45  0.26962896 2.7876468 1.9018401 -114.38388
## 46         46  0.27061571 2.7908793 1.9037751 -114.47740
## 47         47  0.27152407 2.7938546 1.9055563 -114.56342
## 48         48  0.27236031 2.7965933 1.9071961 -114.64254
## 49         49  0.27313020 2.7991144 1.9087057 -114.71533
## 50         50  0.27383904 2.8014353 1.9100956 -114.78231
```

```
## 51         51  0.27449170 2.8035721 1.9113753 -114.84393
## 52         52  0.27509265 2.8055394 1.9125536 -114.90065
## 53         53  0.27564602 2.8073508 1.9136386 -114.95284
## 54         54  0.27615558 2.8090186 1.9146377 -115.00088
## 55         55  0.27662482 2.8105543 1.9155577 -115.04510
## 56         56  0.27705694 2.8119685 1.9164049 -115.08580
## 57         57  0.27745488 2.8132708 1.9171851 -115.12328
## 58         58  0.27782137 2.8144700 1.9179037 -115.15777
## 59         59  0.27815890 2.8155744 1.9185654 -115.18953
## 60         60  0.27846975 2.8165915 1.9191748 -115.21877
## 61         61  0.27875604 2.8175282 1.9197361 -115.24570
## 62         62  0.27901973 2.8183909 1.9202531 -115.27049
## 63         63  0.27926259 2.8191854 1.9207292 -115.29332
## 64         64  0.27948627 2.8199172 1.9211677 -115.31434
## 65         65  0.27969230 2.8205912 1.9215716 -115.33370
## 66         66  0.27988206 2.8212119 1.9219437 -115.35153
## 67         67  0.28005685 2.8217837 1.9222863 -115.36794
## 68         68  0.28021784 2.8223103 1.9226019 -115.38306
## 69         69  0.28036613 2.8227954 1.9228927 -115.39699
## 70         70  0.28050273 2.8232421 1.9231604 -115.40981
## 71         71  0.28062854 2.8236537 1.9234071 -115.42163
## 72         72  0.28074444 2.8240327 1.9236343 -115.43250
## 73         73  0.28085119 2.8243819 1.9238436 -115.44252
## 74         74  0.28094952 2.8247035 1.9240363 -115.45175
## 75         75  0.28104010 2.8249997 1.9242139 -115.46025
## 76         76  0.28112353 2.8252726 1.9243775 -115.46808
## 77         77  0.28120039 2.8255240 1.9245281 -115.47529
## 78         78  0.28127119 2.8257555 1.9246669 -115.48193
## 79         79  0.28133640 2.8259688 1.9247948 -115.48805
## 80         80  0.28139647 2.8261653 1.9249125 -115.49369
## 81         81  0.28145181 2.8263462 1.9250210 -115.49888
## 82         82  0.28150278 2.8265130 1.9251209 -115.50366
## 83         83  0.28154974 2.8266665 1.9252130 -115.50806
## 84         84  0.28159299 2.8268080 1.9252978 -115.51212
## 85         85  0.28163284 2.8269383 1.9253759 -115.51586
## 86         86  0.28166954 2.8270583 1.9254478 -115.51930
## 87         87  0.28170335 2.8271689 1.9255141 -115.52247
## 88         88  0.28173450 2.8272707 1.9255752 -115.52539
## 89         89  0.28176319 2.8273646 1.9256314 -115.52808
## 90         90  0.28178962 2.8274510 1.9256832 -115.53056
## 91         91  0.28181397 2.8275306 1.9257310 -115.53284
## 92         92  0.28183639 2.8276040 1.9257749 -115.53495
## 93         93  0.28185705 2.8276715 1.9258154 -115.53689
## 94         94  0.28187609 2.8277338 1.9258527 -115.53867
## 95         95  0.28189362 2.8277911 1.9258871 -115.54031
## 96         96  0.28190977 2.8278439 1.9259188 -115.54183
## 97         97  0.28192465 2.8278926 1.9259479 -115.54322
## 98         98  0.28193835 2.8279374 1.9259748 -115.54451
## 99         99  0.28195098 2.8279787 1.9259995 -115.54569
## 100       100  0.28196261 2.8280167 1.9260223 -115.54678
## 101       101  0.28197332 2.8280517 1.9260433 -115.54779
## 102       102  0.28198319 2.8280840 1.9260627 -115.54871
```

Looking at the results, we can see in this case, that the algorithm did not achieved the desired convergence

within 100 iterations, iterating until it reaches the maximum number of iterations condition and thus, we can conclude that it is not suitable for this particular case. In order to reach the convergence, we needed to increase the maximum number of iterations up to 102. As expected, the convergence of the algorithm is very slow since we are dealing with a very high percentage of non-observable/censored data.

d) Direct optimization: optim() and BFGS

In this case, we will solve the same test instances by a direct optimization of the log-likelihood function without needing to perform an iterative EM approach. In order to perform these operations, we will define the observed log-likelihood function as follows:

$$
\begin{aligned}
L(\theta/y) &= \prod_{i=1}^{m} f(y_i) \prod_{j=m+1}^{n} [1 - F(\tau)] && (35)\\
&= \prod_{i=1}^{m} \phi\left(\frac{y_i - x_i^{'}\beta}{\sigma}\right)\sigma^{-1} \prod_{j=m+1}^{n} [1 - F(\tau)] && (36)\\
&= \prod_{i=1}^{m} \phi\left(\frac{y_i - x_i^{'}\beta}{\sigma}\right)\sigma^{-1} \prod_{j=m+1}^{n} [1 - \Phi\left(\frac{\tau - x_j^{'}\beta}{\sigma}\right)] && (37)
\end{aligned}
$$

From this previous expression, we can easily compute the observed log-likelihood by applying the *log*() function:

$$
\begin{aligned}
\log(L(\theta/y)) &= \sum_{i=1}^{m} \log\left[\phi\left(\frac{y_i - x_i^{'}\beta}{\sigma}\right)\sigma^{-1}\right] + \sum_{j=m+1}^{n} \log\left(1 - \Phi\left(\frac{\tau - x_j^{'}\beta}{\sigma}\right)\right) && (38)\\
&= -m\log(\sigma) - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - x_i^{'}\beta)^2 + \sum_{j=m+1}^{n} \log\left(1 - \Phi\left(\frac{\tau - x_j^{'}\beta}{\sigma}\right)\right) && (39)
\end{aligned}
$$

1. Based on the previous analysis, we can define the *LLObserved_Optim_SigmaLog*() function, taking into account the relevant issue indicated in Piazza by the instructor regarding the treatment of the σ variable using the *log*() function and then recalculating it inside the function for avoiding problems and NaN issues within the optimization approach.

```
LLObserved_Optim_SigmaLog <- function(par, Tau, yObs, XObs, yCen, XCen){
  # Observable term
  Obs1 <- - (1 / (2 * (exp(par[3]) ^ 2)))
  Obs2 <- sum((yObs - par[1] - par[2] * XObs) ^ 2)
  Obs3 <- - length(yObs) * par[3]
  Obs <- Obs1 * Obs2 + Obs3

  # Standardized value
  cdfVal <- (Tau - par[1] - XCen * par[2]) / exp(par[3])

  # Censored term
  Cen <- sum(log(1 - pnorm(q = cdfVal, mean = 0, sd = 1)) )

  # Return the LL value
  return(Obs + Cen)
}
```

2. Therefore, we generate the first test instance with 20% of censored variables and we optimize the function.

```r
# Optimize the function
# Model Data
n <- 100
Tau <- 4

# Generate Data
Data <- GenData(n, Tau)
yCen <- Data[[1]]
yObs <- Data[[2]]
XCen <- Data[[3]]
XObs <- Data[[4]]

# Check % of censored
print(paste0("Number of censored observations (out of ", n, "): ", length(yCen)))
```

```
## [1] "Number of censored observations (out of 100): 20"
```

```r
# Optimize the LL function by using optim()
Results = optim(par=c(0.1440021, 1.928731, 1.683601), LLObserved_Optim_SigmaLog,
                Tau = Tau, yObs = yObs, XObs = XObs,
                yCen=yCen, XCen = XCen,
                method = "BFGS", control=list(fnscale=-1, trace=TRUE, REPORT=1))
```

```
## initial  value 161.622652
## iter   2 value 135.689960
## iter   3 value 125.630687
## iter   4 value 122.891211
## iter   5 value 121.610193
## iter   6 value 121.429625
## iter   7 value 121.415465
## iter   8 value 121.409451
## iter   9 value 121.409341
## iter  10 value 121.409331
## iter  11 value 121.409300
## iter  11 value 121.409299
## iter  11 value 121.409299
## final  value 121.409299
## converged
```

```r
# Obtain the optimal theta values
OptTheta <- Results$par
OptTheta[3] <- exp(OptTheta[3])
names(OptTheta) <- c("Beta0", "Beta1", "Sigma")

# Print summary
print(paste0("Optimal LL Value: ",Results$value))
```

```
## [1] "Optimal LL Value: -121.409299314009"
```

```r
print("Optimal Theta vector:")
```

```
## [1] "Optimal Theta vector:"
```

```r
print(OptTheta)
```

```
##     Beta0     Beta1     Sigma
## 0.4562605 2.8220287 2.1471080
```

```r
print("Number of iterations:")
```

```
## [1] "Number of iterations:"
```

```r
print(Results$counts)
```

```
## function gradient
##       32       11
```

Based on the results, we can see that: (1) the number of iterations is larger (11 gradient calls and 32 function calls vs 10 iterations of the EM algorithm) than in the case of the EM algorithm, and (2) the results obtained are almost identical to the ones achieved by the EM algorithm. Therefore, we are able to conclude that our EM algorithm implementation has been successful and arises as a very useful alternative for maximizing the likelihood of problems with censored data. Now, we proceed to optimize the model with 80% of censored data.

```r
# Optimize the function
# Model Data
n <- 100
Tau <- 0

# Generate Data
Data <- GenData(n, Tau)
yCen <- Data[[1]]
yObs <- Data[[2]]
XCen <- Data[[3]]
XObs <- Data[[4]]

# Check % of censored
print(paste0("Number of censored observations (out of ", n, "): ", length(yCen)))
```

```
## [1] "Number of censored observations (out of 100): 80"
```

```r
# Optimize the LL function by using optim()
Results = optim(par=c( -1.29215258, 0.3748579, 0.9847210), LLObserved_Optim_SigmaLog,
                Tau = Tau, yObs = yObs, XObs = XObs,
                yCen=yCen, XCen = XCen,
                method = "BFGS", control=list(fnscale=-1, trace=TRUE, REPORT=1))
```

```
## initial  value 106.666604
## iter   2 value 83.564058
## iter   3 value 55.146459
## iter   4 value 52.720239
## iter   5 value 51.087512
## iter   6 value 50.385273
```

```
## iter   7 value 50.313680
## iter   8 value 50.278301
## iter   9 value 50.260810
## iter  10 value 50.255357
## iter  11 value 50.238913
## iter  12 value 50.238335
## iter  12 value 50.238334
## iter  12 value 50.238334
## final  value 50.238334
## converged
```

```
# Obtain the optimal theta values
OptTheta <- Results$par
OptTheta[3] <- exp(OptTheta[3])
names(OptTheta) <- c("Beta0", "Beta1", "Sigma")

# Print summary
print(paste0("Optimal LL Value: ",Results$value))
```

```
## [1] "Optimal LL Value: -50.2383342496005"
```

```
print("Optimal Theta vector:")
```

```
## [1] "Optimal Theta vector:"
```

```
print(OptTheta)
```

```
##     Beta0     Beta1     Sigma
## 0.2821218 2.8284998 1.9263184
```

```
print("Number of iterations:")
```

```
## [1] "Number of iterations:"
```

```
print(Results$counts)
```

```
## function gradient
##       36       12
```

In this case, the convergence is faster than in the EM algorithm requiring only 12 gradient iterations and 36 calls to the function, in comparison to the 102 iterations performed by the EM algorithm in order to reach the convergence tolerance threshold. Again, we can see that both solutions are almost identical, telling us that our EM algorithm implementation has been successful.