Problem Set 8

Natalia Sarabia Vásquez

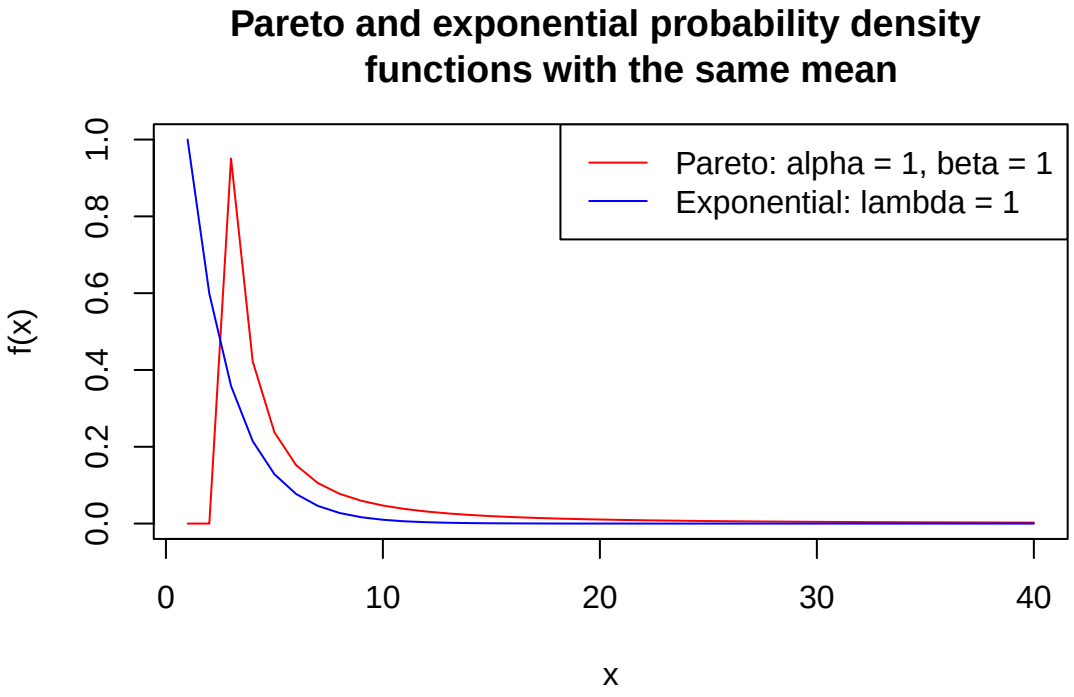
December 03, 2021

1 Problem 1

a) Does the tail of a Pareto decays more quickly or more slowly than that of an exponential distribution?

Pareto is a heavy tailed distribution (its moment generating function does not exist for $t > 0$, which is the definition of heavy tailed) bounded below by the probability density function of an exponential distribution. Therefore, Pareto decays slower than the exponential distribution.

```
alpha <- 1
beta <- 1
lambda <- 1

x <- dpareto(seq(0,20,length=40),a=beta,b=alpha)
y <- dexp(seq(0,20,length=40),rate=lambda)

plot(x, type = "l", col = "red", ylim=c(0,1), main = "Pareto and exponential probability density
    functions with the same mean", ylab = "f(x)", xlab = "x")
lines(y, type = "l", col = "blue")
legend(x = "topright", legend = c("Pareto: alpha = 1, beta = 1", "Exponential: lambda = 1"),
       col=c("red","blue"), lty=c(1,1))
```



b) Suppose f is an exponential density with parameter value equal to 1 ($\lambda = 1$), shifted by two to the right, so that $f(x) = 0$ for $x < 2$. Pretend that you can't sample from f and use importance sampling where our sampling density, g, is a Pareto distribution with $\alpha = 2$ and $\beta = 3$. Use $m = 10000$ to estimate $\mathbb{E}(X)$ and $\mathbb{E}(X^2)$ and compare to the known expectations for the shifted exponential.

Recall the following properties of the exponential distribution:

1. If $X \sim exp(\lambda)$, the probability density function is given by: $f_X(x) = \lambda e^{-\lambda x}$,
2. Its mean is given by: $\mathbb{E}(X) = \frac{1}{\lambda}$ and
3. Its variance is given by: $Var(X) = \frac{1}{\lambda^2}$.

Some useful results. If I shift the exponential distribution 2 units to the right, I have the following results (the last two are because of the linearity of the expectation):

1. Density function: $f_{X+2}(x) = \lambda e^{-\lambda(x-2)}$,
2. First moment (Mean): $\mathbb{E}(X + 2) = \mathbb{E}(X) + 2 = \frac{1}{\lambda} + 2$,
3. and second moment is: $\mathbb{E}((X + 2)^2) = \mathbb{E}(X^2 + 2X + 4) = \mathbb{E}(X^2) + 2\mathbb{E}(X) + 4$.

As I am given that $\lambda = 1$ and $\mathbb{E}(X^2) = Var(X) + \mathbb{E}(X) = \frac{1}{\lambda^2} + \frac{1}{\lambda}$, I can substitute and get that:

1. $\mathbb{E}(X + 2) = 1 + 2 = 3$ and
2. $\mathbb{E}((X + 2)^2) = \mathbb{E}(X^2) + 4\mathbb{E}(X) + 4 = 2 + 4 + 4 = 10$

Also recall that importance sampling allows us to estimate: $\phi = \mathbb{E}_f(h(X)) = \int h(x)\frac{f(x)}{g(x)}g(x)dx$ using the following expression $\hat{\phi} = \frac{1}{m}\sum_{i=1}^{m} h(x_i)\frac{f(x_i)}{g(x_i)}$

Under the particular set up of the problem, I want to calculate the first moment (expectation) and second moment of a shifted exponential f with parameter $\lambda = 1$ using a Pareto density g of parameters $(2, 3)$. X are sampled from a Pareto distribution, $h(X) = X$ is equal to the identity function, $f(X) = 1 \times e^{-1\times(X-2)}$ and $g(X)$ is the probability density function of the Pareto with the given parameters. Then, I compute:

$$\hat{\phi} = \mathbb{E}(X) = \frac{1}{m}\sum_{i=1}^{m} x_i \frac{f(x_i)}{g(x_i)} \tag{1}$$

and

$$\hat{\phi} = \mathbb{E}(X^2) = \frac{1}{m}\sum_{i=1}^{m} x_i^2 \frac{f(x_i)}{g(x_i)} \tag{2}$$

```
# Parameters given in the problem:
m <-10000
alpha <- 2
beta <- 3

# Generate X with the given specifications
X <- rpareto(m, beta, alpha)
# Calculate g(X)
g <- dpareto(X, beta, alpha)
# Calculate f(X)
```

```r
f <- 1 * exp(-1 * (X - 2))

# Using equation (1):
first_moment <- mean((X * f) / g)
first_moment
```

```
## [1] 3.037474
```

```r
# I know that a shifted exponential with parameter 1, has a first moment (mean) of:
3
```

```
## [1] 3
```

```r
# Using equation (2):
second_moment <- mean((X^2 * f) / g)
second_moment
```

```
## [1] 10.21355
```

```r
# I know that a shifted exponential with parameter 1, has a first moment (mean) of:
10
```

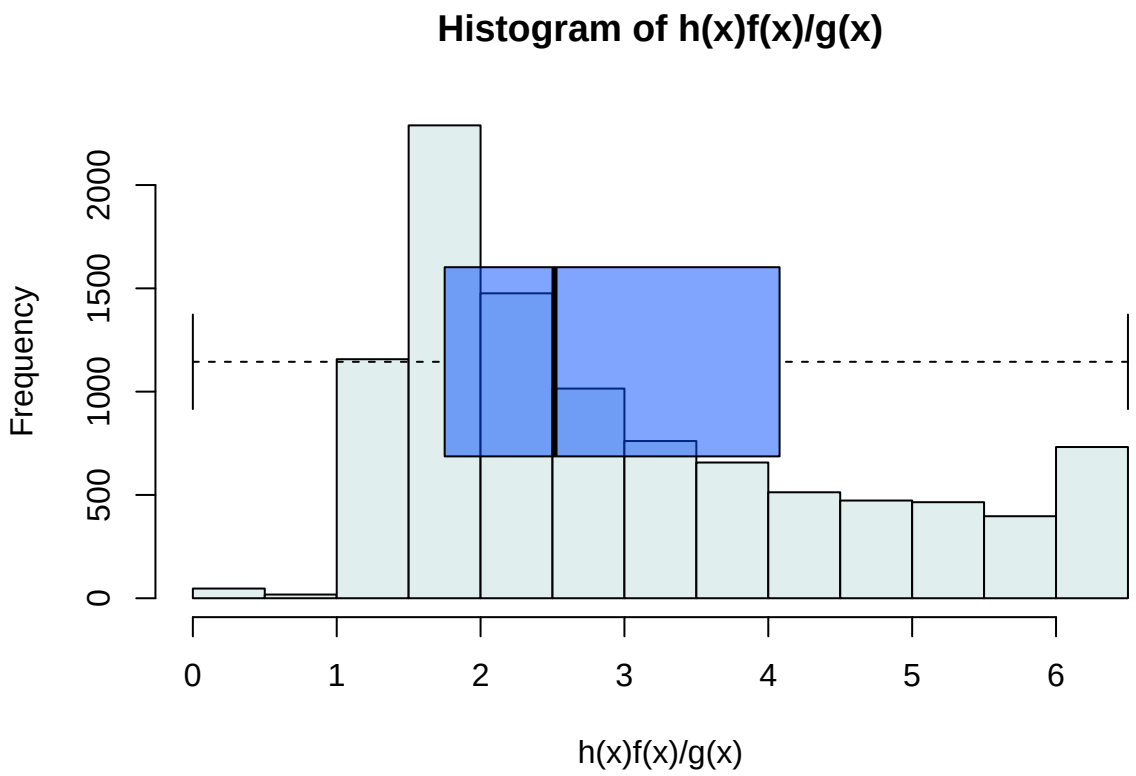
```
## [1] 10
```

In this case, I am getting that the parameters of my sampled distributions are the ones I am expecting.

Recall that $Var(\hat{\phi}) \propto Var(\frac{h(X)f(X)}{g(X)})$. Create histograms of $\frac{h(x)f(x)}{g(x)}$ and of the weights $\frac{f(X)}{g(X)}$ to get an idea for whether $Var(\hat{\phi})$ is large.

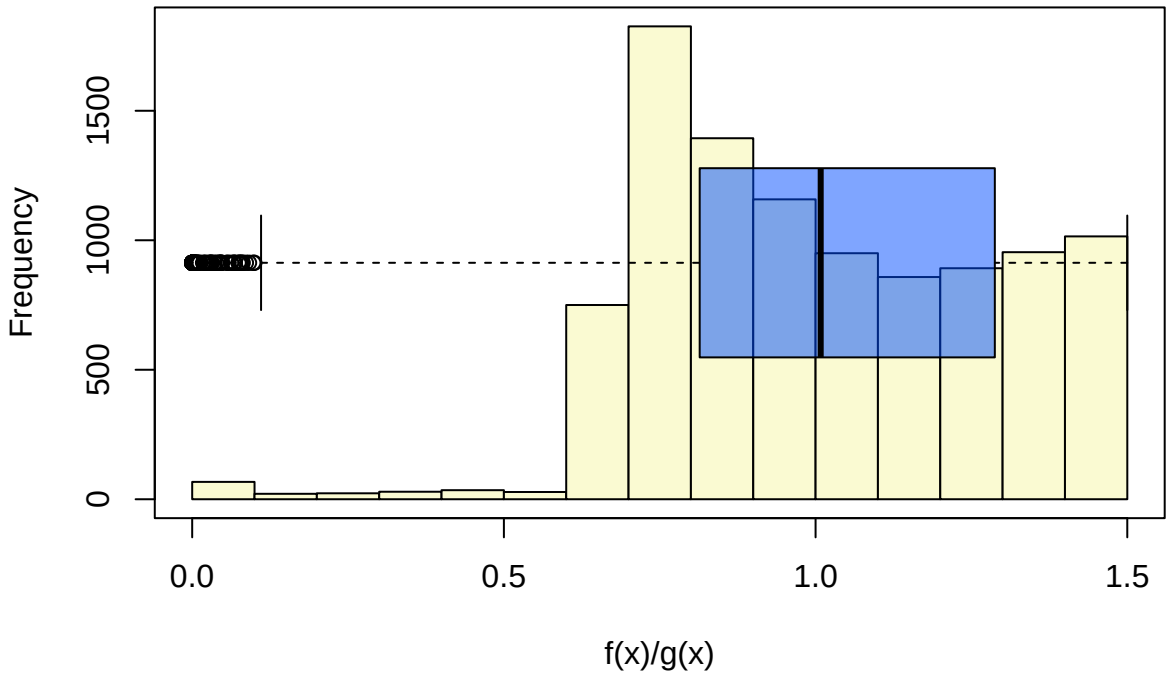
Histogram of $\frac{h(x)f(x)}{g(x)}$:

```r
hist((X * f) / g, main = "Histogram of h(x)f(x)/g(x)", xlab = "h(x)f(x)/g(x)",
     ylab = "Frequency",col="azure2")

par(new = TRUE)

# Box plot
boxplot((X * f) / g, horizontal = TRUE, axes = FALSE,
        col = rgb(0, 0.3, 1, alpha = 0.5))
```



Histogram of $\frac{f(x)}{g(x)}$:

```
hist(f / g, main = "Histogram of the weigths, f(x)/g(x)", xlab = "f(x)/g(x)",
     ylab = "Frequency",col="lightgoldenrodyellow")

par(new = TRUE)

# Box plot
boxplot(f / g, horizontal = TRUE, axes = FALSE,
        col = rgb(0, 0.3, 1, alpha = 0.5))
box()
```



In this case, $g(x)$ has heavier tails. After seeing the graphs, I can see that the weights $f(x_i)/g(x_i)$ are large when $h(x_i)$ is small. Therefore I won't have excessively influential points.

c) Now suppose f is the Pareto distribution described above and pretend you can't sample from f and use importance sampling where our sampling density, g, is the exponential described above. Respond to the same questions as for part b), comparing to the known values of the Pareto.

Recall the following properties of the pareto distribution:

1. If $X \sim pareto(\alpha, \beta)$, the probability density function is given by: $f_X(x) = \frac{\beta\alpha^\beta}{x^{\beta+1}}$ for $\alpha < x < \infty$, $\alpha > 0$ and $\beta > 0$
2. Its mean is given by: $\mathbb{E}(X) = \frac{\beta\alpha}{\beta-1}$ for $\beta \leq 1$,
3. Its variance is given by: $Var(X) = \frac{\beta\alpha^2}{(\beta-1)^2(\beta-2)}$ and
4. Its second moment is given by:

$$\begin{aligned}\mathbb{E}(X^2) &= Var(X) + \mathbb{E}(X) \\ &= \frac{\beta\alpha}{\beta-1} + \frac{\beta\alpha^2}{(\beta-1)^2(\beta-2)} \\ &= \frac{\beta\alpha(\beta-1)(\beta-2) + \beta\alpha^2}{(\beta-1)^2(\beta-2)} \\ &= \frac{\beta\alpha((\beta-1)(\beta-2)+\alpha)}{(\beta-1)^2(\beta-2)}\end{aligned}$$

For the exponential distribution, I have that:

$f_{X+2}(x) = \lambda e^{-\lambda(x-2)}$
$y = \lambda e^{-\lambda(x-2)}$

And the quantile function is then given by:

$q_X(u) = \frac{-ln(\frac{u}{\lambda})}{\lambda} + 2$

```r
# Parameters given in the problem:
m <-10000
lambda <- 1

# Generate X with the given specifications
U <- runif(m, 0, 1)
X <- ((-log(U/lambda))/lambda) + 2

# Calculate g(X)
g <- lambda * exp(-lambda*(X-2))
# Calculate f(X)
f <- dpareto(X, beta, alpha)

# Using equation (1):
first_moment <- mean((X * f) / g)
first_moment
```

```
## [1] 2.94428
```

```r
# I know that a pareto with parameters alpha and beta, has a first moment (mean) of:
beta * alpha / (beta - 1)
```

```
## [1] 3
```

```r
beta * alpha^2 / (((beta-1)^2) * (beta-2))
```

```
## [1] 3
```

```r
# Using equation (2):
second_moment <- mean((X^2 * f) / g)
second_moment
```

```
## [1] 10.29922
```

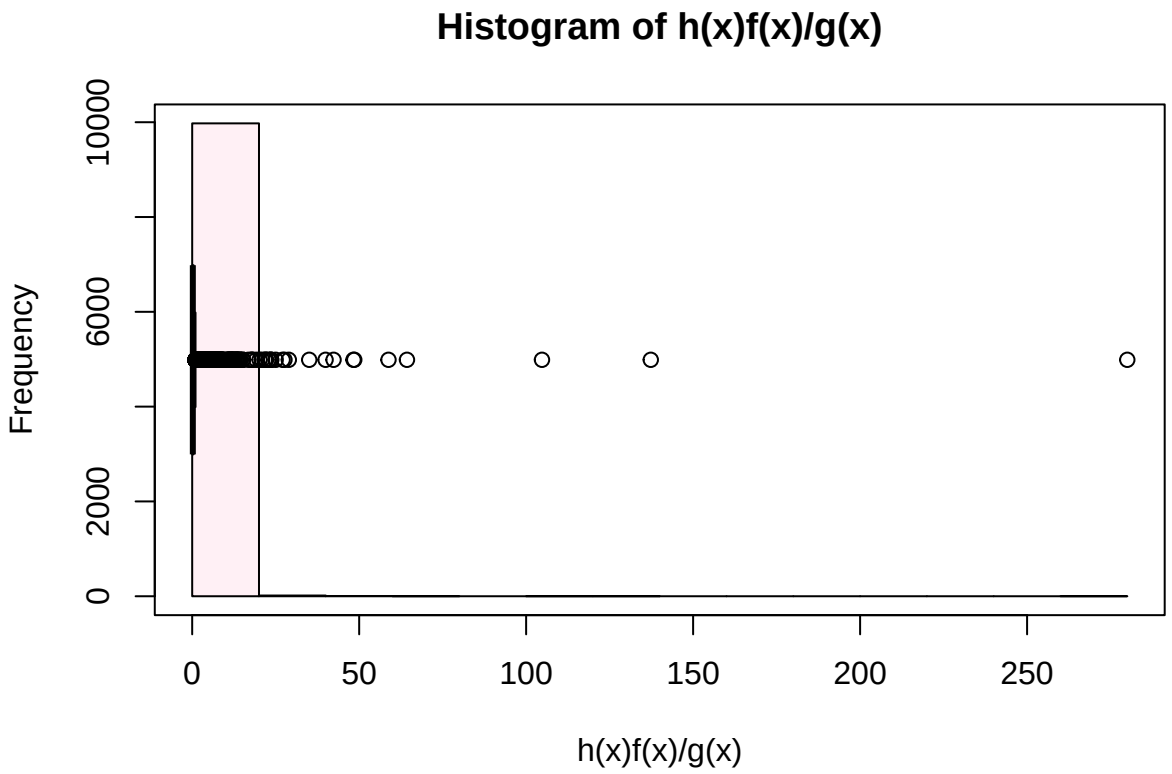
```r
# I know that the second moment of a pareto is:
var <- (beta * alpha^2)/((beta-1)^2*(beta-2))
mean <- (beta * alpha)/(beta-1)
second_moment <- var + mean^2
second_moment
```

```
## [1] 12
```

In this case, I am able to get the theoretical value for the mean but not for the second moment.

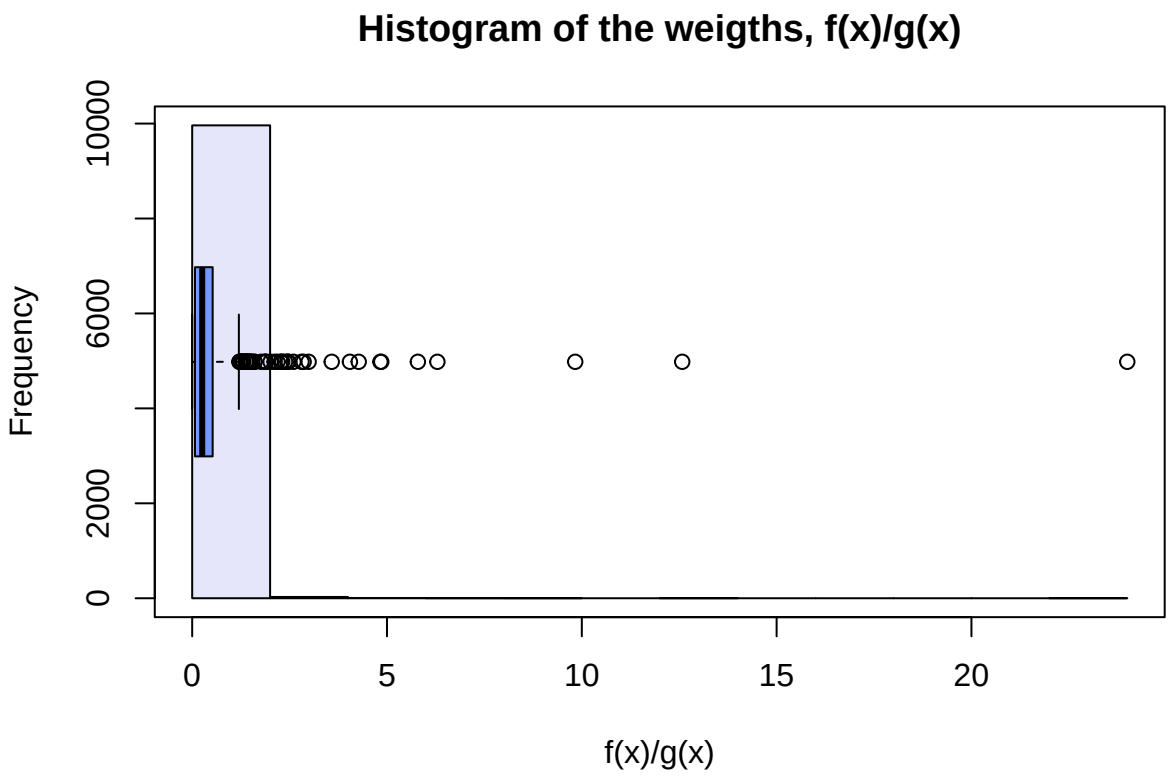
Histogram of $\frac{h(x)f(x)}{g(x)}$:

```r
hist((X * f) / g, main = "Histogram of h(x)f(x)/g(x)", xlab = "h(x)f(x)/g(x)",
     ylab = "Frequency",col="lavenderblush")
par(new = TRUE)

# Box plot
boxplot((X * f) / g, horizontal = TRUE, axes = FALSE,
        col = rgb(0, 0.3, 1, alpha = 0.5))
box()
```



Histogram of $\frac{f(x)}{g(x)}$:

```
hist(f / g, main = "Histogram of the weigths, f(x)/g(x)", xlab = "f(x)/g(x)",
     ylab = "Frequency",col="lavender")

par(new = TRUE)

# Box plot
boxplot(f / g, horizontal = TRUE, axes = FALSE,
        col = rgb(0, 0.3, 1, alpha = 0.5))
box()
```



In this case, $g(x)$ has lighter tails. After seeing the graphs, I notice that I have excessively influential points (see outliers in the boxplot). Although the densities have the same support, I am sampling from a density with lighter tails than the density of interest, and then I am having issues with the estimation of the moments.

2 Problem 2

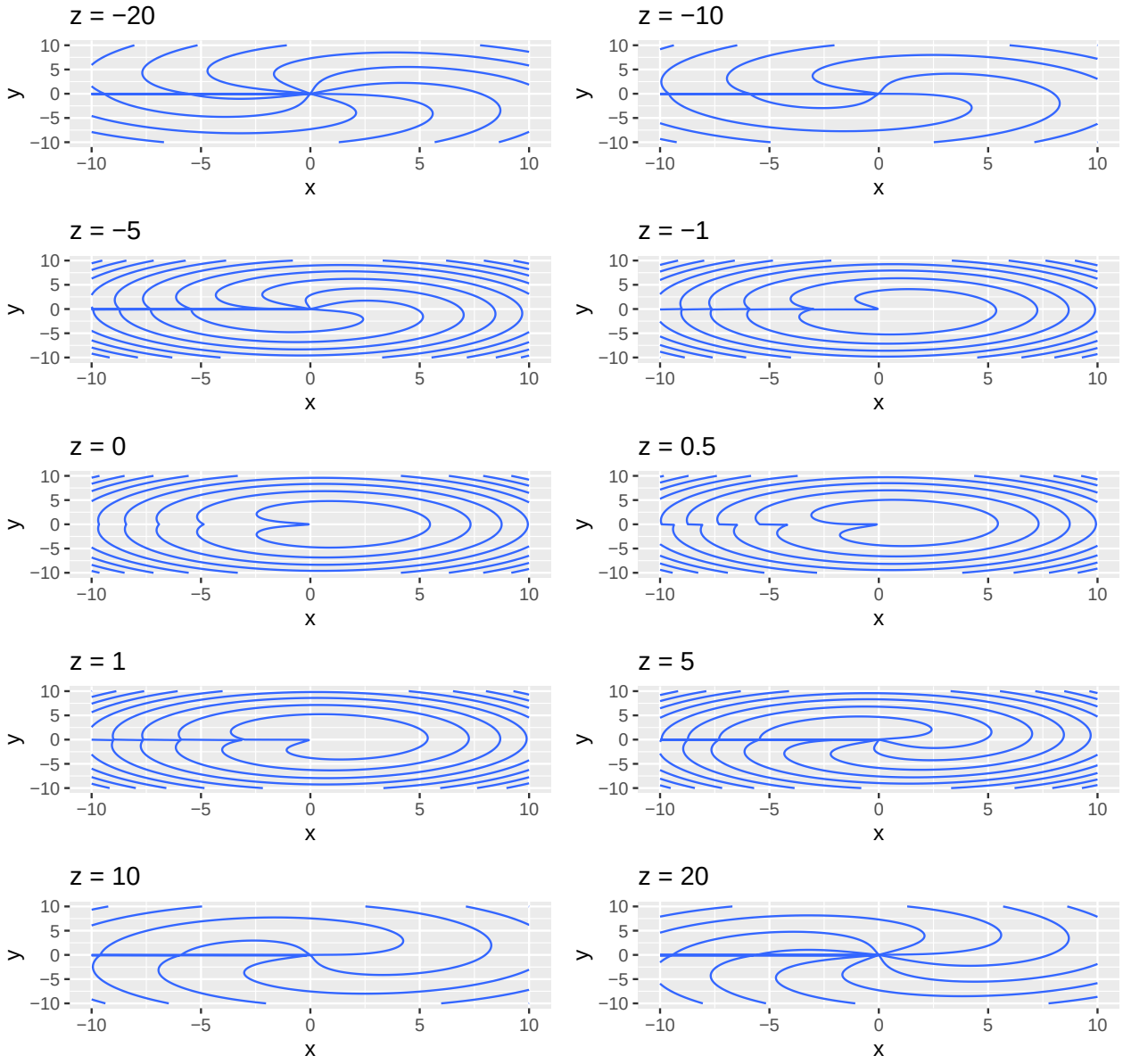
Plot slices of the function to get a sense for how it behaves (i.e., for a constant value of one of the inputs, plot as a 2-d function of the other two):

```
# Define theta function:
theta <- function(x1,x2) atan2(x2, x1)/(2*pi)

# Define helical function:
helical <- function(x) {
  f1 <- 10*(x[3] - 10*theta(x[1],x[2]))
  f2 <- 10*(sqrt(x[1]^2 + x[2]^2) - 1)
  f3 <- x[3]
  return(f1^2 + f2^2 + f3^2)
}

# Set two grids: one for each of the non-fixed variables:
grid1 <-  seq(-10, 10, 0.1)
grid2 <-  seq(-10, 10, 0.1)
```

2.1 z fixed

```
# Generate a dataframe with every possible combination of x and y:
data.fit <-  expand.grid(x = grid1, y = grid2)
```

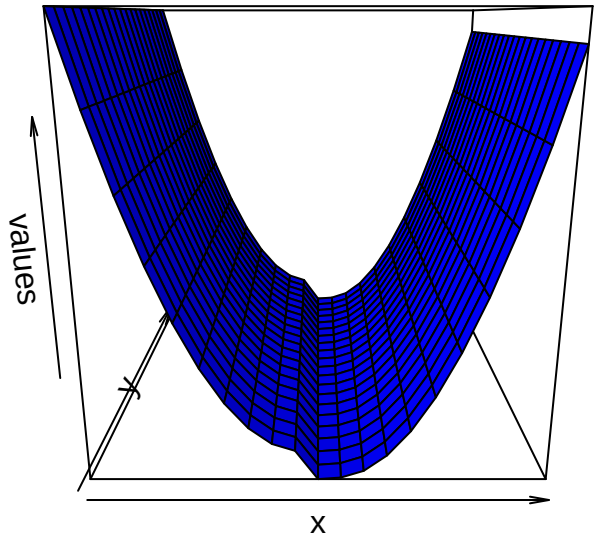
```r
# Auxiliar loop to try different fixed values against the two variables:
all_plots <- list(10)
k = 1

for(i in c(-20, -10, -5, -1, 0, 0.5, 1, 5, 10, 20)){
  # Evaluate helical:
  values <- apply(cbind(data.fit,i), 1, helical)
  # Construct the plot:
  plot <- cbind(data.fit,values) %>%
  ggplot(aes(x = x, y = y, z = values)) +
    stat_contour() +
    ggtitle(paste0("z = ", i))
  # Save the plots in a list:
  all_plots[[k]] <- plot

  k <- k + 1
}

# Save plots' grid:
p <- plot_grid(plotlist = all_plots, nrow = 6, ncol = 2)
# Add title:
title <- ggdraw() + draw_label("Function helical for z fixed", fontface='bold')
# Plot everything in a "gridstyle":
plot_grid(title, p, ncol=1, rel_heights=c(0.1, 1))
```



3D plot fixing z:

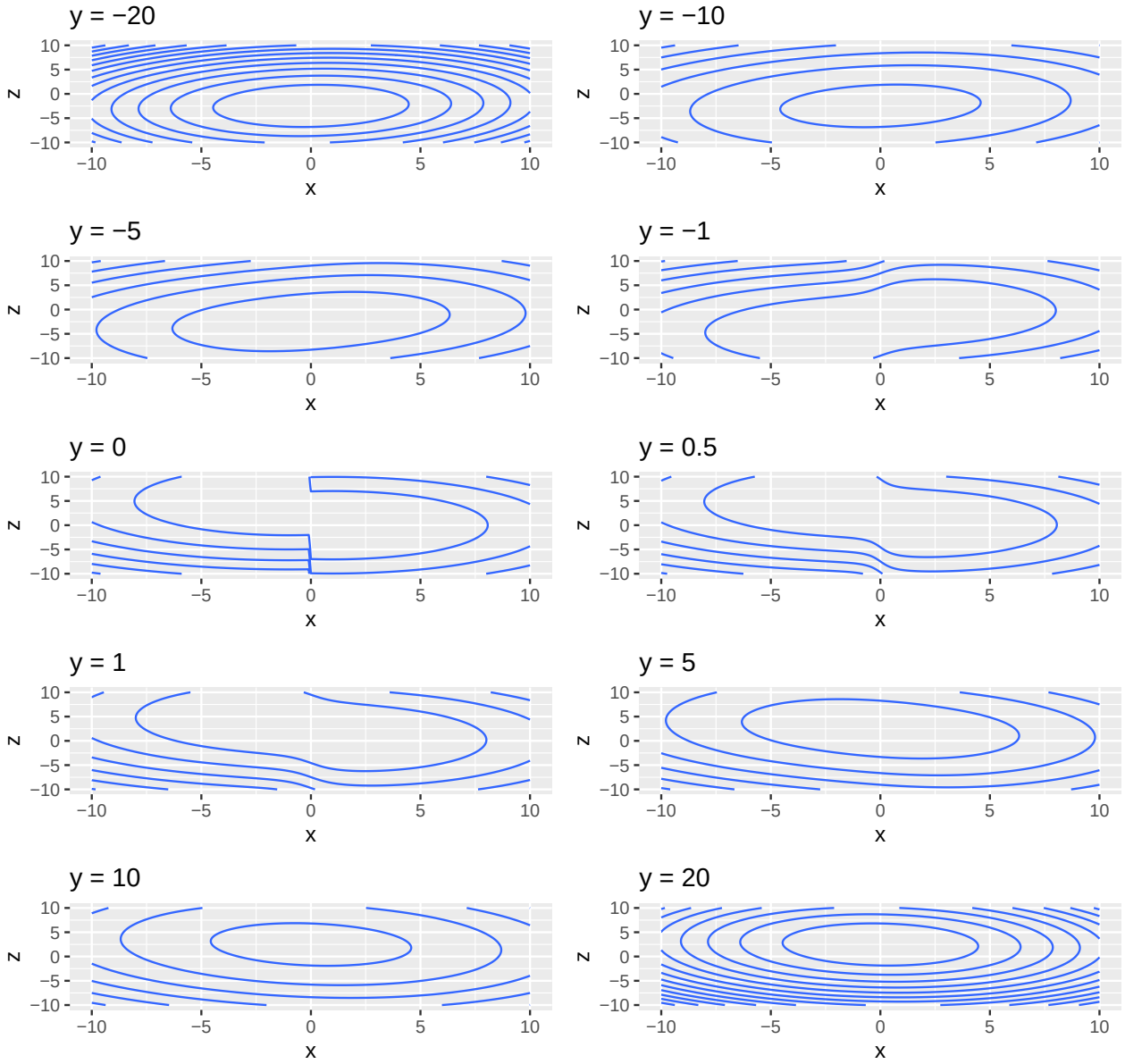
```
helical_modified <- function(y,z) {
  z = 0
  f1 <- 10*(z - 10*theta(x,y))
  f2 <- 10*(sqrt(x^2 + y^2) - 1)
  f3 <- z
  return(f1^2 + f2^2 + f3^2)
}

n <- 10
x <- seq(-n, n, 1)
y <- seq(-n, n, 1)
```

```
values <- outer(x, y, helical_modified)

oldpar <- par(bg = "white")
persp(x,y,values,col="blue",shade=0.2)
```



2.2 y fixed

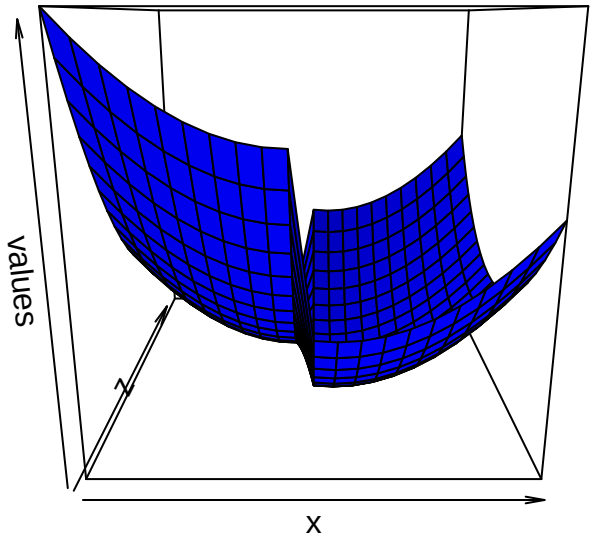
```
# Generate a dataframe with every possible combination of x and z:
data.fit <-  expand.grid(x = grid1, z = grid2)

# Auxiliar loop to try different fixed values against the two variables:
all_plots <- list(10)
k = 1
for(i in c(-20, -10, -5, -1, 0, 0.5, 1, 5, 10, 20)){

  values <- apply(cbind(data.fit$x,i,data.fit$z), 1, helical)

  plot <- cbind(data.fit,values) %>%
  ggplot(aes(x = x, y = z, z = values)) +
    stat_contour() +
    ggtitle(paste0("y = ", i))

  all_plots[[k]] <- plot

  k <- k + 1
}

p <- plot_grid(plotlist = all_plots, nrow = 6, ncol = 2)

title <- ggdraw() + draw_label("Function helical for y fixed", fontface='bold')
plot_grid(title, p, ncol=1, rel_heights=c(0.1, 1))
```



3D plot fixing y:

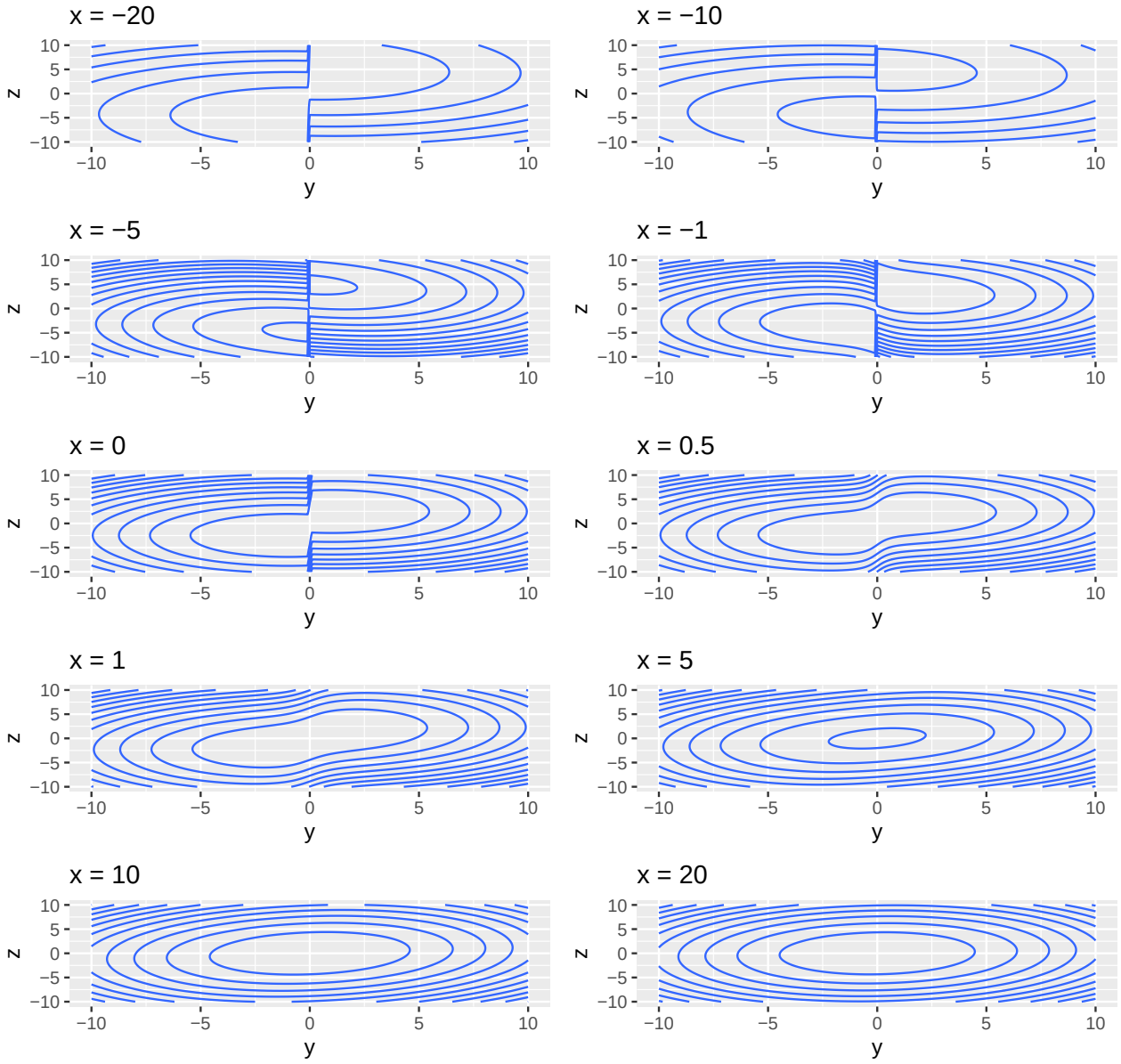
```
helical_modified <- function(x,z) {
  y = 0
  f1 <- 10*(z - 10*theta(x,y))
  f2 <- 10*(sqrt(x^2 + y^2) - 1)
  f3 <- z
  return(f1^2 + f2^2 + f3^2)
}

n <- 10
x <- seq(-n, n, 1)
z <- seq(-n, n, 1)
```

```
values <- outer(x, z, helical_modified)

oldpar <- par(bg = "white")
persp(x,z,values,col="blue",shade=0.2)
```



2.3 x fixed

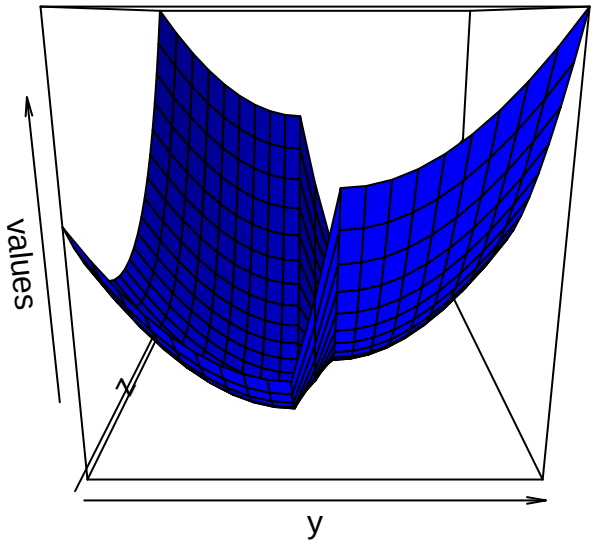
```
# Generate a dataframe with every possible combination of y and z:
data.fit <-  expand.grid(y = grid1, z = grid2)

# Auxiliar loop to try different fixed values against the two variables:
all_plots <- list(10)
k = 1
for(i in c(-20, -10, -5, -1, 0, 0.5, 1, 5, 10, 20)){

  values <- apply(cbind(i, data.fit$y, data.fit$z), 1, helical)

  plot <- cbind(data.fit,values) %>%
  ggplot(aes(x = y, y = z, z = values)) +
    stat_contour() +
    ggtitle(paste0("x = ", i))

  all_plots[[k]] <- plot

  k <- k + 1
}

p <- plot_grid(plotlist = all_plots, nrow = 6, ncol = 2)

title <- ggdraw() + draw_label("Function helical for x fixed", fontface='bold')
plot_grid(title, p, ncol=1, rel_heights=c(0.1, 1))
```



3D plot fixing x:

```
helical_modified <- function(y,z) {
  x = 0
  f1 <- 10*(z - 10*theta(x,y))
  f2 <- 10*(sqrt(x^2 + y^2) - 1)
  f3 <- z
  return(f1^2 + f2^2 + f3^2)
}

n <- 10
y <- seq(-n, n, 1)
z <- seq(-n, n, 1)
```

```
values <- outer(y, z, helical_modified)

oldpar <- par(bg = "white")
persp(y,z,values,col="blue",shade=0.2)
```



Now try out optim() and nlm() for finding the minimum of this function (or use optimx()). Explore the possibility of multiple local minima by using different starting points.

I will try different initial values for both functions and then summarize everything in a table. I was expecting to have the same results but it turns out that I always get the same results using nlm() but this is not the case for optim():

```
# Define different values for x, y and z to evaluate each function:
results_optim <- as.data.frame(
  matrix(c(1,1,1,
           10000,10000,10000,
           runif(1,-1000,1000),runif(1,-1000,1000),runif(1,-1000,1000),
           runif(1,-1000,0),runif(1,2,1000),1,
           -100000000000,0,pi),byrow = TRUE,ncol=3,nrow=5))

results_nlm <- as.data.frame(
  matrix(c(1,1,1,
           10000,10000,10000,
           runif(1,-1000,1000),runif(1,-1000,1000),runif(1,-1000,1000),
           runif(1,-1000,0),runif(1,2,1000),1,
           -100000000000,0,pi),byrow = TRUE,ncol=3,nrow=5))

# Apply the helical function to each combination I created for x, y and z:
summary_optim <- apply(results_optim,1,optim,helical,method="Nelder-Mead")
summary_nlm <- apply(results_nlm,1,function(x) nlm(helical,x))

# Obtain the minimized points:
for(i in 1:5){
  results_optim[i,c(4:6)] <- as.vector(summary_optim[[i]]$par)
  results_optim[i,7] <- as.vector(summary_optim[[i]]$value)
  results_nlm[i,c(4:6)] <- as.vector(summary_nlm[[i]]$estimate)
  results_nlm[i,7] <- as.vector(summary_nlm[[i]]$minimum)
```

```
}

names(results_nlm) <- c("x","y","z","par1","par2","par3","minimum value of helical")
names(results_optim) <- c("x","y","z","par1","par2","par3","minimum value of helical")
```

Results obtained with function optim():

```
# Using stargazer, set the information in tables:
stargazer(results_optim, summary=FALSE, rownames=FALSE, table.placement = "H",
          header = FALSE,font.size = "scriptsize")
```

Table 1:

x	y	z	par1	par2	par3	minimum value of helical
1	1	1	1.000	-0.0001	-0.0002	0.00000
10,000	10,000	10,000	2.610	-0.383	-0.692	290.072
239.881	563.391	893.990	-0.356	0.935	3.139	10.202
-807.389	114.069	1	1.032	-0.087	-0.233	1.175
-100,000,000,000	0	3.142	-6,444,540.000	6,467,258.000	-8,000,006.000	14,799,767,113,585,220

Results obtained with function nlm():

```
# Using stargazer, set the information in tables:
stargazer(results_nlm, summary=FALSE, rownames=FALSE, table.placement = "H",header = FALSE)
```

Table 2:

x	y	z	par1	par2	par3	minimum value of helical
1	1	1	1.000	-0.0001	-0.0001	0.00000
10,000	10,000	10,000	1.000	-0.0001	-0.0001	0.00000
993.438	785.410	-95.100	1.000	-0.0001	-0.0001	0.00000
-635.165	389.492	1	1	0	0	0
-100,000,000,000	0	3.142	1	0	0	0

It seems that nlm() provides more stable solutions. It does not matter where I start, I always obtain the same values for the parameters, but this is not the case with optim(). Also, be aware that sometimes convergence can depend on the starting point if a functions is not unimodal rather than on the algorithm.

3 Problem 3

3.1 (a) **Design an algorithm to estimate the three parameters, $\theta = (\beta_0, \beta_1, \sigma^2)$. taking the actual data to be available, plus the actual values for the censored observations:**

Consider a censored regression problem. Assume a simple linear regression model $Y_i \sim N(\beta_0 + \beta_1 x, \sigma^2)$.

Suppose I have an iid sample, but for any observation with $Y > \tau$, all we are told is that Y exceeds τ and not its actual value.

- In each sample, n_c of the n observations will be censored, depending on how many exceed τ.

a) Design an algorithm to estimate the three parameters

$$\theta = (\beta_0, \beta_1, \sigma^2)$$

Let $W = (Y, Z)$, where Y_i are uncensored dependent variable (from $i = 1$ to n_c) and Z_i are censored dependent variables (from $i = n_{c+1}$ to n):

$$L(\theta; W) = \prod_{i=1}^{n} f(\theta; w_i) \qquad Y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$$

To optimize the log likelihood, we will have troubles because of the censored data, as it involves this:

$$f(w; \theta) = \int f(y, z; \theta)\, dz$$

However, the EM-algorithm provides a recipe for this case:

Let θ^t be the current value of θ. Then, define:

$$Q(\theta | \theta^t) = \mathbb{E}(\log L(\theta | W) | Y; \theta^t)$$

E-step

And taking the expectation of the log-likelihood:

$$\mathbb{E}\left(\log \left\{ \prod_{i=1}^{n} \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) e^{-\frac{1}{2\sigma^2}\left[(y_i - \beta_0 - \beta_1 x_i)^2 + (z_i - \beta_0 - \beta_1 x_i)^2\right]} \right\} \middle| Y; \theta^t; z_i > \tau \right)$$

$$= \mathbb{E}\left(\log \left\{ \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n_c}(y_i - \beta_0 - \beta_1 x_i)^2} e^{-\frac{1}{2\sigma^2}\sum_{i=n_c+1}^{n}(z_i - \beta_0 - \beta_1 x_i)^2} \right\} \middle| Y; \theta^t; z_i > \tau \right)$$

$$= \mathbb{E}\left(-n\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n_c}(y_i-\beta_0-\beta_1 x_i)^2 - \frac{1}{2\sigma^2}\sum_{i=n_c+1}^{n}(z_i-\beta_0-\beta_1 x_i)^2 \,\middle|\, y_i\theta^t_i, z>\tau\right)$$

Remember, the only random part is z_i, so:

$$= -n\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n_c}(y_i-\beta_0-\beta_1 x_i)^2 - \frac{1}{2\sigma^2}\sum_{i=n_c+1}^{n}\mathbb{E}\left[(z_i-\beta_0-\beta_1 x_i)^2 \,|\, y_i\theta^t_i, z>\tau\right] \quad \cdots (1)$$

Working in the last term:

$$\sum_{i=n_c+1}^{n}\mathbb{E}\left((z_i-\beta_0-\beta_1 x_i)^2 \,|\, y_i\theta^t_i, z>\tau\right)$$

$$= \sum_{i=n_c+1}^{n}\mathbb{E}(z_i^2 \,|\, y_i\theta^t_i, z>\tau) - 2\sum_{i=n_c+1}^{n}\mathbb{E}(z_i \,|\, y_i\theta^t_i, z>\tau)(\beta_0+\beta_1 x_i)$$

$$+ \sum_{i=n_c+1}^{n}(\beta_0+\beta_1 x_i)^2 \quad \cdots (2)$$

Recalling that $\mathbb{E}(X^2) = \mathrm{Var}(X) + \mathbb{E}(X)$

In my case:

$k_{1i} = \mathbb{E}(z_i \,|\, y_i\theta^t_i, z>\tau)$ → This is the mean of a truncated normal with expectation $\beta_0+\beta_1 x_i$, var $= \sigma^2$

$k_{2i} + k_{1i}^2 = \mathrm{Var}(z_i \,|\, y_i\theta^t_i, z>\tau) + \mathbb{E}(z_i \,|\, y_i\theta^t_i, z>\tau) =$

$\mathbb{E}(z_i^2 \,|\, y_i\theta^t_i, z>\tau)$ ↳ This is the second moment of the same truncated normal.

Then:

Rewriting (2):

$$= \sum_{i=n_c+1}^{n} k_{2i} + k_{1i}^2 - 2\sum_{i=n_c+1}^{n} k_{1i}(\beta_0+\beta_1 x_i) + \sum_{i=n_c+1}^{n}(\beta_0+\beta_1 x_i)^2$$

We have something of the form $\Sigma(\bullet)^2 + \Sigma(\bullet\bullet)$

$$= \sum_{i=n_c+1}^{n}\left(k_{1i} - (\beta_0+\beta_1 x_i)\right)^2 + \sum_{i=n_c+1}^{n} k_{2i}$$

Substituting in (1):

$$= -n\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{nc}\left(y_i - \beta_0 - \beta_1 x_i\right)^2 - \frac{1}{2\sigma^2}\sum_{i=nc+1}^{n}\left(k_{1i} - (\beta_0 + \beta_1 x_i)\right)^2$$

$$+ \sum_{i=nc+1}^{n} k_{2i}\left(\frac{1}{2\sigma^2}\right)$$

Define y^* as $\begin{pmatrix} y_1 \\ \vdots \\ y_{nc} \\ k_{1,nc+1} \\ \vdots \\ k_{1n} \end{pmatrix}$ { from $i = 1$ to nc → uncensored
{ from $i = nc+1$ to n → censored

(In professor's notation, m_L)

So I can complete the sum to run from $i=1$ to n by just redefining the y vector:

$$= -n\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i^* - (\beta_0 + \beta_1 x_i)^2\right) - \frac{1}{2\sigma^2}\sum_{i=nc+1}^{n} k_{2i}$$

To optimize this function, with respect to β_0, β_1 and σ^2 I should derive the latter expresion with respect to β_0, β_1 and σ^2, equal to zero and solve for each of them.

It turns out that this is just a problem of simple linear regression of X and y^*. Therefore, I can use the `lm` function in R to obtain my estimates for β_0 and β_1.

In the case of σ^2, I should do:

$$\frac{dQ}{d\sigma} = -n\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i^* - (\beta_0 + \beta_1 x_i)\right)^2 - \frac{1}{2\sigma^2}\sum_{i=nc+1}^{n} k_{2i}$$

$$-n\frac{\left(\frac{1}{2}\right)(2\pi\sigma^2)^{-1/2}(2\pi 2\sigma)}{\sqrt{2\pi\sigma^2}} = -n\left(\frac{1}{2}\right)\left(\frac{1}{2\pi\sigma^2}\right)(2\pi 2\sigma) = -\frac{n}{\sigma}$$

$$= -\frac{n}{\sigma} + \frac{1}{\sigma^3}\left\{\sum_{i=1}^{n}(y_i^* - (\beta_0 + \beta_1 x_i))^2 + \sum_{i=n_c+1}^{n} k_{zi}\right\}$$

Equaling to zero and solving:

$$\frac{n}{\sigma} = \frac{1}{\sigma^3}\left\{\sum_{i=1}^{n}(y_i^* - (\beta_0 + \beta_1 x_i))^2 + \sum_{i=n_c+1}^{n} k_{zi}\right\}$$

$$\Rightarrow \quad \sigma^2 = \frac{\left\{\sum_{i=1}^{n}(y_i^* - (\beta_0 + \beta_1 x_i))^2 + \sum_{i=n_c+1}^{n} k_{zi}\right\}}{n}$$

M-Step

I will maximize the simple linear regression model I just found plus the σ^2 term.

As we have seen in several previous classes, the β_0 and β_1 estimates have closed expressions (since $\sum k_{zi}$ will be constant, and then, the derivative of that is zero and I'm left with the usual regression).

There are closed forms for $\hat{\beta_0}$ and $\hat{\beta_1}$. But if I want to be elegant, I can actually use the lm function to obtain them (because that's how it actually get the estimates).

Algorithm:

After all this math, I can write the algorithm:

1. Determine initial values for $\beta_0^t, \beta_1^t, \sigma$.
2. Compute $\tau^*, \rho(\tau^*)$, $E(w|w>\tau) = z_1$, $Var(w|w>\tau) = z^2$ (how I name it in my code)
3. Calculate Y^* and fit
   $lm(Y^* \sim X, data)$
   get $\hat{\beta}_0^{t+1}, \hat{\beta}_1^{t+1}, \hat{\sigma}^{2\,t+1}$
4. Recalculate $\tau^*, \beta(\tau^*), z_1, z_2$ } Repeat until convergence.
5. Step 3

Your estimator for σ^2 should involve a ratio where the numerator involves the usual sum of squares for the non-censored data plus an additional term that you should interpret statistically. The additional term is the

variance of the truncated normal. This term will play a significant role to the estimation of the variance of the censored model, it will account by the fact that I have n_c censored data which variance could be greater than the observed in the censored data.

You should be able to analytically maximize the expected log likelihood. In 201B, we learned how to get the MLE (that in this specific case, correspond to the OLS method). I will add them as a proof of how the estimators can be obtained maximizing the loglikelihood function. This is the general case, therefore it works for particular cases as mine. Note that the extra term in my loglikelihood does not affect the maximum because it does not depends on β_0^t nor β_1^t:

1. Consider the normal linear regression model

$$Y_i \overset{\text{indep}}{\sim} N(\beta_0 + \beta_1 x_i, \sigma^2) \qquad i = 1, \ldots, n$$

find the MLEs for β_0, β_1 and σ^2.

We write the likelihood function $\mathcal{L}(\beta_0, \beta_1, \sigma^2)$ as:

$$\mathcal{L}(\beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{n} f_{y_i}(\beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{n} \left(\frac{1}{\sigma\sqrt{\pi 2}}\right) e^{-\frac{(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^2}}$$

$$= \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n e^{-\sum_{i=1}^{n} \frac{(y_i - \beta_0 - \beta_1 x_i)^2}{a\sigma^2}}$$

then $\ell(\beta_0, \beta_1, \sigma^2) = \ln\left[\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n e^{-\sum_{i=1}^{n} \frac{(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^2}}\right]$

$$= -n\left(\ln(\sigma\sqrt{2\pi})\right) - \sum_{i=1}^{n} \frac{(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^2}$$

For β_0:

$$\frac{\partial \ell(\beta_0, \beta_1, \sigma^2)}{\partial \beta_0} = \left(-\frac{1}{2\sigma^2}\right)(2)\left(\Sigma(y_i - \beta_0 - \beta_1 x_i)\right)(-1) = \frac{1}{\sigma^2}\left(\Sigma(y_i - \beta_0 - \beta_1 x_i)\right)$$

For β_1:

$$\frac{\partial \ell(\beta_0, \beta_1, \sigma^2)}{\partial \beta_1} = \left(-\frac{1}{2\sigma^2}\right)(2)\left(\Sigma(y_i - \beta_0 - \beta_1 x_i)\right)(-x_i) = \frac{1}{\sigma^2}\left(\Sigma(y_i - \beta_0 - \beta_1 x_i)x_i\right)$$

Setting both derivatives to zero, in order to find the estimates that maximize $\ell(\beta_0, \beta_1, \sigma^2)$ we have:

$$\frac{\partial \ell}{\partial \beta_0} = 0 \Rightarrow \Sigma y_i - n\beta_0 - \beta_1 \Sigma x_i = 0 \Rightarrow \beta_0 = \frac{\Sigma y_i}{n} - \frac{\beta_1 \Sigma x_i}{n} = \bar{y} - \bar{x}\beta_1$$

$$\frac{\partial \ell}{\partial \beta_1} = 0 \Rightarrow \Sigma x_i y_i - \beta_0 \Sigma x_i - \beta_1 \Sigma x_i^2 = 0 \Rightarrow$$ if we substitute the value of β_0, we have:

$$\sum x_i y_i - (\bar{y} - \bar{x}\beta_1)\sum x_i - \beta_1 \sum x_i^2 = 0$$

$$\sum x_i y_i - \sum x_i \bar{y} + \beta_1 \sum x_i \bar{x} - \beta_1 \sum x_i^2 = 0$$

$$\beta_1 = \frac{\sum x_i \bar{y} - \sum x_i y_i}{\sum x_i \bar{x} - \sum x_i^2}$$

In this particular case, it will suffice with just checking that the second derivatives with respect to β_0, β_1 and σ^2 are negative, to show that we have found a maximum

$$\frac{\partial^2 l(\beta_0, \beta_1, \sigma^2)}{\partial \beta_0^2} = \frac{1}{\sigma^2}(-\beta_0) < 0$$

$$\frac{\partial^2 l(\beta_0, \beta_1, \sigma^2)}{\partial \beta_1^2} = \frac{1}{\sigma^2}(-\sum x_i^2) < 0$$

To find the MLE estimate for σ^2 :

$$\frac{\partial l(\beta_0, \beta_1, \sigma^2)}{\partial \sigma^2} = -\frac{\sqrt{2\pi}\, n}{\sigma\sqrt{2\pi}} - (-2)\frac{\sum (y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^3}$$

$$= -\frac{n}{\sigma} + \frac{\sum (y_i - \beta_0 - \beta_1 x_i)^2}{\sigma^3}$$

$$\frac{n}{\sigma} = \frac{\sum (y_i - \beta_0 - \beta_1 x_i)^2}{\sigma^2} \Rightarrow \sigma^2 = \frac{\sum (y_i - \beta_0 - \beta_1 x_i)^2}{n}$$

Verifying that the function has a maximum in this point.

$$\frac{\partial^2 l(\beta_0, \beta_1, \sigma^2)}{\partial \sigma^2} \Rightarrow = \frac{(-n)(-1)}{\sigma^2} + \frac{\sum (y_i - \beta_0 - \beta_1 x_i)^2 (-3)}{\sigma^4}$$

$$= \frac{n}{\sigma^2} - 3\frac{\sum (y_i - \beta_0 - \beta_1 x_i)^2}{\sigma^4}$$

If we remember that $\sum (y_i - \beta_0 - \beta_1 x_i)^2$ is an approx of how large the errors are, we have $\sum (y_i - \beta_0 - \beta_1 x_i)^2 \simeq \sigma^2 n$

$$= \frac{n}{\sigma^2} - \frac{3n\sigma^2}{\sigma^4} \simeq -\frac{2n}{\sigma^2} < 0$$

Finally:

$$\hat{\beta_0} = \bar{y} - \bar{x}\hat{\beta_1}$$

$$\hat{\beta_1} = \frac{\Sigma x_i \bar{y} - \Sigma x_i y_i}{\Sigma x_i \bar{x} - \Sigma x_i^2}$$

$$\hat{\sigma}^2 = \frac{\Sigma (y_i - \hat{\beta_0} - \hat{\beta_1} x_i)^2}{n}$$

3.2 (b) Provide reasonable starting values for β_0, β_1 and σ^2:

I propose as my initial values for β_0 and β_1 the coefficients of the linear regression of the values that are not censored. For σ^2, I propose the sum of the residuals of the mentioned linear regression:

```
set.seed(1)

# Sample size:
n <- 100

# Parameters of the simulated data:
beta0 <- 1
beta1 <- 2
sigma2 <- 6
tau <- 4

# Simulate the data:
x <- runif(n)

# Generate y:
y <- rnorm(n, beta0 + beta1 * x, sqrt(sigma2))
tau1 <- quantile(y, probs = c(.80))
tau2 <- quantile(y, probs = c(.20))

# Separate censored data from uncensored data:
censored <- as.data.frame(cbind(x,y)) %>%
  filter(y > tau)

uncensored <- as.data.frame(cbind(x,y)) %>%
  filter(y <= tau)

# Propose initial solutions:
# Fit the model for the uncensored data
model <- lm(y ~ x, data = uncensored)

# Retrieve the parameters of the fit and save them:
b0_t <- model$coefficients[1]
b1_t <- model$coefficients[2]
sigma2_t <- (sum(model$residuals^2))/nrow(uncensored)
```

3.3 (c) Write an R function to estimate the three parameters:

```
EM_algorithm <- function(b0t, b1t, sigma2_t, tau){
      set.seed(1)

      # Sample size:
      n <- 100

      # Parameters of the simulated data:
      beta0 <- 1
      beta1 <- 2
      sigma2 <- 6

      # Simulate the data:
```

```r
x <- runif(n)

# Generate y:
y <- rnorm(n, beta0 + beta1 * x, sqrt(sigma2))

# Separate censored data from uncensored data:
censored <- as.data.frame(cbind(x,y)) %>%
  filter(y > tau)

uncensored <- as.data.frame(cbind(x,y)) %>%
  filter(y <= tau)

# Compute values for the calculation of z1 and z2:
mu <- b0_t + b1_t * censored$x
tau_star <- (tau - mu)/sqrt(sigma2_t)
rho_tau_star <- (dnorm(tau_star)/(1-pnorm(tau_star)))

z1 <- mu + sqrt(sigma2_t) * rho_tau_star
z2 <- sigma2_t * (1 + (tau_star * rho_tau_star) - (rho_tau_star)^2)

# Define tolerance:
tol <- 0.00000000000001

# Create an indicator for the number of iteration:
iteration = 1

# Initialize differences
check <- 1000

# Define x:
x <- c(uncensored$x,censored$x)

# Define the condition when the algorithm should finish the iterations:
while( (check > tol) ){
  # Define y_star as a vector with the censored and uncensored y's:
  y <- c(uncensored$y,z1)
  data <- as.data.frame(cbind(x,y))

  # Fit a simple linear regression. This will help us to obtain the values
  # for beta0 and beta1. Once we have their updated values, we can replace
  # them in the expression for sigma2:
  fit <- lm(y ~ x, data = data)

  # Keep the coefficients of the fitted regression:
  b0_tplus1 <- summary(fit)$coef[1]
  b1_tplus1 <- summary(fit)$coef[2]

  # Estimate sigma:
  sigma2_tplus1 <- ( sum( (y - b0_tplus1 - b1_tplus1 * x) ^ 2) +
                       ( sum(z2) ) ) / length(x)

  # Check the difference between the values of beta0_t and beta0_plus1:
  check <- abs(b0_t - b0_tplus1)
```

```
        # Assign the "news" b0_t, b1_t and sigma2_t:
        b0_t <- b0_tplus1; b1_t <- b1_tplus1; sigma2_t <- sigma2_tplus1

        # Update the value of mu, tau_star and rho(tau_star):
        mu <- b0_t + b1_t * censored$x
        tau_star <- (tau - mu) / sqrt(sigma2_t)
        rho_tau_star <- (dnorm(tau_star) / (1 - pnorm(tau_star)))

        # Update the value for z1 and z2:
        z1 <- mu + sqrt(sigma2_t) * rho_tau_star
        z2 <- sigma2_t * (1 + (tau_star * rho_tau_star) - (rho_tau_star) ^ 2)

        # Add one to the count of iterations:
        iteration <- iteration + 1
      }

      return(c(b0_tplus1, b1_tplus1, sigma2_tplus1, iteration - 1 ))
}
```

3.3.1 (c.1) Test your function using data simulated based on the code in ps8.R with a modest proportion of exceedances expected, say 20%:

τ_1 and τ_2 were obtained using the quantile function over y, getting the cutoffs at 80% and 20%, respectively:

```
EM_results <- EM_algorithm(b0t, b1t, sigma2_t, tau1)
cat(" tau_1:", tau1,"\n",
    "b0 estimate:", EM_results[1],"\n",
    "b1 estimate:", EM_results[2],"\n",
    "sigma2 estimate:", EM_results[3],"\n",
    "Number of iterations:", EM_results[4],"\n")
```

```
##  tau_1: 4.006751
##  b0 estimate: 0.4566128
##  b1 estimate: 2.824108
##  sigma2 estimate: 4.618876
##  Number of iterations: 25
```

3.3.2 (c.2) Test your function using data simulated based on the code in ps8.R with a modest proportion of exceedances expected, say 80%:

```
EM_results <- EM_algorithm(b0t, b1t, sigma2_t, tau2)
cat(" tau_2:", tau2,"\n",
    "b0 estimate:", EM_results[1],"\n",
    "b1 estimate:", EM_results[2],"\n",
    "sigma2 estimate:", EM_results[3],"\n",
    "Number of iterations:", EM_results[4],"\n")
```

```
##  tau_2: 0.02991534
##  b0 estimate: 0.3126394
##  b1 estimate: 2.87922
##  sigma2 estimate: 3.841964
##  Number of iterations: 345
```

3.4 (d) Estimate the parameters (and standard errors) for your test cases using optim() with the BFGS option in R:

The idea here is:
1. Write the log-likelihood function
2. Use the optim() function and the option "BFGS"
3. Compute the se of the estimators

The log-likelihood function is:

$$l(\theta) = -(n-n_c)log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2}\sum_{i=n_c+1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2 + \sum_{i=1}^{n_c} log(\mathbb{P}(Y_i > \tau))$$

```
set.seed(1)

# Sample size:
n <- 100

# Parameters of the simulated data:
beta0 <- 1
beta1 <- 2
sigma2 <- 6
tau <- 4

# Simulate the data:
x <- runif(n)

# Generate y:
y <- rnorm(n, beta0 + beta1*x, sqrt(sigma2))

# Separate censored data from uncensored data:
censored <- as.data.frame(cbind(x,y)) %>%
  filter(y > tau)

uncensored <- as.data.frame(cbind(x,y)) %>%
  filter(y <= tau)

# Define negative log likelihood function:
negLoglikelihood <- function(x){
  # Parameters
  beta0 <- x[1]
  beta1 <- x[2]
  sigma22 <- exp(x[3])

  # log-likelihood function
   ll <- - length(uncensored$x) * log(sqrt(2*pi*sigma22)) -
     (1/(2*sigma22)) * sum((uncensored$y-beta0-beta1*uncensored$x)^2) +
     sum(pnorm(tau, beta0+beta1*censored$x,sqrt(sigma22),log = TRUE, lower.tail = FALSE))

   return(-ll)
}

# Propose initial solutions:
# Fit the model for the uncensored data
model <- lm(y ~ x, data = uncensored)
```

```r
# Retrieve the parameters of the fit and save them:
b0_initial <- model$coefficients[1]
b1_initial <- model$coefficients[2]
sigma2_initial <- (sum(model$residuals^2))/nrow(uncensored)

# Create a vector with the proposed initial values:
initial <- c(b0_initial,b1_initial,log(sigma2_initial))

# Optimize the negative loglikelihood, optim() minimizes by default, so we need the -ll:
optim_results <- optim(initial, negLoglikelihood , method = "BFGS", hessian = TRUE)

# Compute the Fisher information matrix:
fisher_info <- solve(optim_results $hessian)

# Compute the standard errors of our estimates:
se <- diag( sqrt( diag(fisher_info)) )
```

```r
cat(" b0 estimate:", optim_results$par[1],"\n",
    "b0 se:", se[1,1],"\n",
    "b1 estimate:", optim_results$par[2],"\n",
    "b1 se:", se[2,2],"\n",
    "sigma2 estimate:", optim_results$par[3],"\n",
    "sigma2 se:", se[3,3],"\n",
    "Number of iterations:", optim_results$counts[1],"\n")
```

```
##  b0 estimate: 0.45659
##  b0 se: 0.4768636
##  b1 estimate: 2.820797
##  b1 se: 0.8301784
##  sigma2 estimate: 1.528526
##  sigma2 se: 0.1660406
##  Number of iterations: 22
```

Comments:
The number of iterations of the EM algorithm (25) is greater than the number of iterations of the optim() function (22). I obtained the "same" result, up to two decimals under the EM algorithm and the function optim() using my defined tolerance for the EM algorithm.