TMA4300; Exercise 3

Martinius Singdahlsen, Ola Rasmussen, Johan Lagardére

Contents

Problem A:

Introduction

We will examine to different parameter estimations for a non Gaussian time series of length $T = 100$ fitted with the AR(2) model. The AR(2) model is given by,

$$x_t = \beta_1 x_{t-1} + \beta_2 x_{t-2} + e_t, \tag{A.1}$$

where e_t are all iid with constant variance and zero mean. There are multiple methods of obtaining the parameters in an AR(n) model. We will focus on the least sum of squares (denoted LS) and least sum of absolute value (denoted LA), and compare them to each other. They are given by,

$$Q_{LS}(\mathbf{x}) = \sum_{t=3}^{T} (x_t - \beta_1 x_{t-1} - \beta_2 x_{t-2})^2, \tag{A.2}$$

$$Q_{LA} = \sum_{t=3}^{T} |x_t - \beta_1 x_{t-1} - \beta_2 x_{t-2}|. \tag{A.3}$$

We seek to minimize this function with respect to β_1 and β_2 to obtain their estimate. The first six values in the time series can be seen below:

```r
source("Files/probAhelp.R")
source("Files/probAdata.R")
head(data3A$x)
```

```
## [1] 13.9967400 13.2107777 10.6915149  9.2789323  9.8034937 -0.9814602
```

1.

First we will preform a fit of the time series with the two methods, least sum of squares and least sum of absolute value.

```r
## LS model first (then repeat for LA) We start with the
## fit that we will bootstrap the residuals from
LS_b <- ARp.beta.est(data3A$x, 2)$LS
LA_b <- ARp.beta.est(data3A$x, 2)$LA
LS_em0 <- ARp.resid(data3A$x, LS_b)
LA_em0 <- ARp.resid(data3A$x, LA_b)
```

This gives us the estimates of the parameters β_1 and β_2,

$$\begin{aligned}\text{LS: } \hat{\beta}_1 &= 1.553\\ \hat{\beta}_2 &= -0.568,\\ \text{LA: } \hat{\beta}_1 &= 1.547\\ \hat{\beta}_2 &= -0.558.\end{aligned}$$

To evaluate how good the two parameter estimators are we will perform residual resampling bootstrap. This works by picking two consecutive x_i, x_{i+1} at random from the time series, and predicting 98 new data points,

$$x_i, x_{i+1}, x_2^*, x_3^*, ..., x_{99}^*, x_{100}^*.$$

The data points are predicted by the AR(2) model where the estimated $\hat{\beta}_1$ and $\hat{\beta}_2$ will be used, and the error e_t will be bootstrapped by the standardized error, $\hat{\epsilon}_t$. With this bootstrapped time series we can again use the corresponding method of obtaining our $\hat{\beta}_i$ to obtain the bootstrapped $\hat{\beta}_i^*$.

With the parameters estimated in the bootstrap ($\hat{\beta_1}^*$ and $\hat{\beta}_2^*$) we can calculate the bootstrap estimate of bias given by,

$$\begin{aligned}bias_{\hat{F}}(\beta_i) &= E_{\hat{F}}[s(X^*)] - t(\hat{F})\\ &= \frac{1}{B}\sum_{j=1}^{B}[\hat{\beta}_{ij}^*] - \hat{\beta}_i, \text{ for } i = 1, 2.\end{aligned} \tag{A.4}$$

Here $\hat{\beta}_{ij}$ is the j'th bootstrapped estimate. We will also calculate the variance of the bootstrapped parameters. This will be used to evaluate the performance of the estimators LA and LS.

```r
set.seed(98)
matrix_of_beta_LS <- matrix(c(NA, NA), nrow = 1)
matrix_of_beta_LA <- matrix(c(NA, NA), nrow = 1)
res_matrix_LS <- matrix(data = NA, nrow = 1500, ncol = 98)
res_matrix_LA <- matrix(data = NA, nrow = 1500, ncol = 98)
B <- 1500
for (i in c(1:1500)) {
    random_start <- sample(99, 1)  #random index for start
    # find random two starts
    bootstrap_start <- sapply(c(random_start, random_start +
        1), function(x) {
        data3A$x[x]
    })
    # random index
    random_sample <- sample(98, 100, replace = T)
    # now sample random e
    bootstrap_e <- sapply(random_sample, function(x) {
        LS_em0[x]
    })
    # generate random sequence with the residuals, beta,
    # and random start
    botsrapped_sequence <- ARp.filter(bootstrap_start, LS_b,
        bootstrap_e)[3:102]
    # fit the regression
    beta_boot <- ARp.beta.est(botsrapped_sequence, 2)$LS
    res_matrix_LS[i, ] <- ARp.resid(botsrapped_sequence, beta_boot)
    matrix_of_beta_LS <- rbind(matrix_of_beta_LS, beta_boot)
    # Repeat for LA
    random_start_LA <- sample(99, 1)
    bootstrap_start_LA <- sapply(c(random_start_LA, random_start_LA +
        1), function(x) {
        data3A$x[x]
    })
    random_sample_LA <- sample(98, 100, replace = T)
    bootstrap_e_LA <- sapply(random_sample_LA, function(x) {
        LA_em0[x]
    })
    botsrapped_sequence_LA <- ARp.filter(bootstrap_start_LA,
        LA_b, bootstrap_e_LA)[3:102]
    beta_boot_LA <- ARp.beta.est(botsrapped_sequence_LA, 2)$LA
    res_matrix_LA[i, ] <- ARp.resid(botsrapped_sequence_LA, beta_boot_LA)
    matrix_of_beta_LA <- rbind(matrix_of_beta_LA, beta_boot_LA)
}
matrix_of_beta_LS <- matrix_of_beta_LS[-c(1), ]
```

```r
matrix_of_beta_LA <- matrix_of_beta_LA[-c(1), ]

b1_LS_bias <- mean(matrix_of_beta_LS[, 1]) - LS_b[1]
b2_LS_bias <- mean(matrix_of_beta_LS[, 2]) - LS_b[2]
b1_LS_var <- var(matrix_of_beta_LS[, 1])
b2_LS_var <- var(matrix_of_beta_LS[, 2])
b1_LA_bias <- mean(matrix_of_beta_LA[, 1]) - LA_b[1]
b2_LA_bias <- mean(matrix_of_beta_LA[, 2]) - LA_b[2]
b1_LA_var <- var(matrix_of_beta_LA[, 1])
b2_LA_var <- var(matrix_of_beta_LA[, 2])
```

We end up with the following values for the variance and the bootstrap estimate of the bias for the LA and LS:

```r
cat("Beta 1 variance,", paste(round(b1_LS_var, 10), ",", sep = ""),
    "and bias,", paste(round(b1_LS_bias, 10), ",", sep = ""),
    " using LS.", "\nBeta 1 variance,", paste(round(b1_LA_var,
        10), ",", sep = ""), "and bias,", paste(round(b1_LA_bias,
        10), ",", sep = " "), "using LA.", "\nBeta 2 variance,",
    paste(round(b2_LS_var, 10), ",", sep = ""), "and bias, ",
    paste(round(b2_LS_bias, 10), ",", sep = ""), "using LS.",
    "\nBeta 2 variance,", paste(round(b2_LA_var, 10), ",", sep = ""),
    "and bias, ", paste(round(b2_LA_bias, 10), ",", sep = ""),
    "using LA.")
```

```
## Beta 1 variance, 0.0058185469, and bias, -0.0152248122,  using LS.
## Beta 1 variance, 0.0004262247, and bias, -0.002931104 , using LA.
## Beta 2 variance, 0.0056872531, and bias,  0.0095715666, using LS.
## Beta 2 variance, 0.0004195207, and bias,  0.0023977122, using LA.
```

We can see that both the variance and the bias is greater for both parameters when the LS estimator is applied. Thus the LA estimator outperforms the LS estimator and is optimal for this data. This can be explained by the fact that the time series is non Gaussian and that the LS estimator is optimal for Gaussian time series. However it does not mean that the LS estimator outperforms the LA estimator on non Gaussian time series, as this is quite a large class of time series.

2.

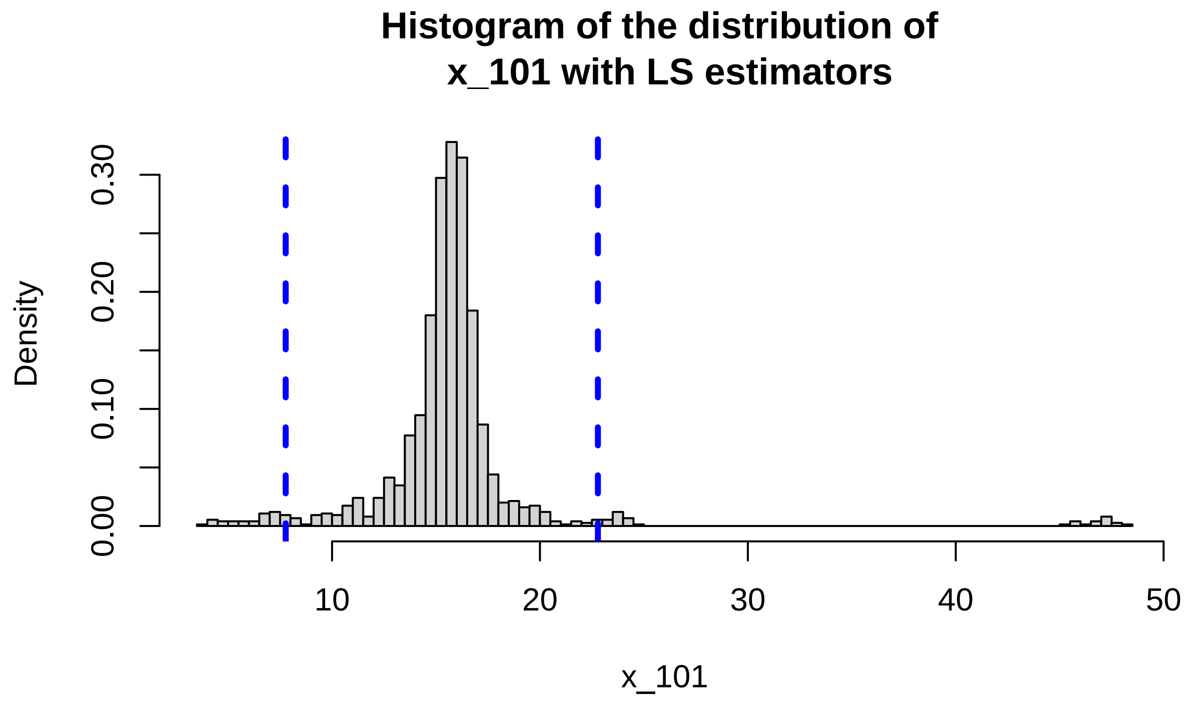
Now we want to compute a prediction interval for x_{101} based on both estimators. To do this we will use the boot strapped generated $\hat{\beta}_i^*$ from problem 1A. Since the bootstrapped $\hat{\beta}_i^*$ represents the distribution of β_i we will sample from this list of $\hat{\beta}_i^*$. We will also bootstrap all residuals ϵ_t calculated from all the boot strapped time series. Thus we can predict x_{101} with the uncertainty of β_0, β_1, and e_t. The histograms of x_{101} using both methods can be seen in Figure 1.

```
set.seed(98)
x <- data3A$x
n <- length(data3A$x)
# sample error from original sequence
error_index_LS_1 <- sample(1500, 1500, replace = T)
error_index_LS_2 <- sample(98, 1500, replace = T)
error_index_LA_1 <- sample(1500, 1500, replace = T)
error_index_LA_2 <- sample(98, 1500, replace = T)
error_boot_LS <- rep(0, 1500)
error_boot_LA <- rep(0, 1500)
for (j in c(1:1500)) {
    error_boot_LS[j] <- res_matrix_LS[error_index_LS_1[j], error_index_LS_2[j]]
    error_boot_LA[j] <- res_matrix_LA[error_index_LA_1[j], error_index_LA_2[j]]
}
# Bootstrap sample from distribution of beta
index_sample_LS <- sample(1500, 1500, replace = T)
index_sample_LA <- sample(1500, 1500, replace = T)
# Do not know if beta_1 is independent of beta_2 thus we
# sample from same bootstrap
beta1_boot_LS <- sapply(index_sample_LS, function(x) {
    matrix_of_beta_LS[x, 1]
})
beta2_boot_LS <- sapply(index_sample_LS, function(x) {
    matrix_of_beta_LS[x, 2]
})
beta1_boot_LA <- sapply(index_sample_LA, function(x) {
    matrix_of_beta_LA[x, 1]
})
beta2_boot_LA <- sapply(index_sample_LA, function(x) {
    matrix_of_beta_LA[x, 2]
})
x_101_LS <- rep(0, B)
x_101_LA <- rep(0, B)
for (i in c(1:B)) {
    x_101_LS[i] = beta1_boot_LS[i] * x[n] + beta2_boot_LS[i] *
        x[n - 1] + error_boot_LS[i]
    x_101_LA[i] = beta1_boot_LA[i] * x[n] + beta2_boot_LA[i] *
        x[n - 1] + error_boot_LA[i]
}
# calculate quantiles
q_LS = quantile(x_101_LS, c(0.025, 0.975))
q_LA = quantile(x_101_LA, c(0.025, 0.975))
# print results
cat("Quantiles LS", q_LS, "\nQuantiles LA", q_LA)
```

```
## Quantiles LS 7.769529 22.78734
## Quantiles LA 7.197511 22.90848
```

```
df_LS = data.frame(x = x_101_LS)
df_LA = data.frame(x = x_101_LA)
par(mfrow = c(2, 1))
hist(df_LS$x, freq = F, breaks = 100, main = "Histogram of the distribution of \n x_101
    xlab = "x_101")
abline(v = q_LS, lwd = 3, lty = 2, col = "blue")
hist(df_LA$x, freq = F, breaks = 100, main = "Histogram of the distribution of \n x_101
    xlab = "x_101")
abline(v = q_LA, lwd = 3, lty = 2, col = "blue")
```

Finally we have as a result a slightly bigger limits in the prediction interval for LA than for LS but the distribution for both method of our x_{101} remains almost the same so that there is no method outperforming the other.



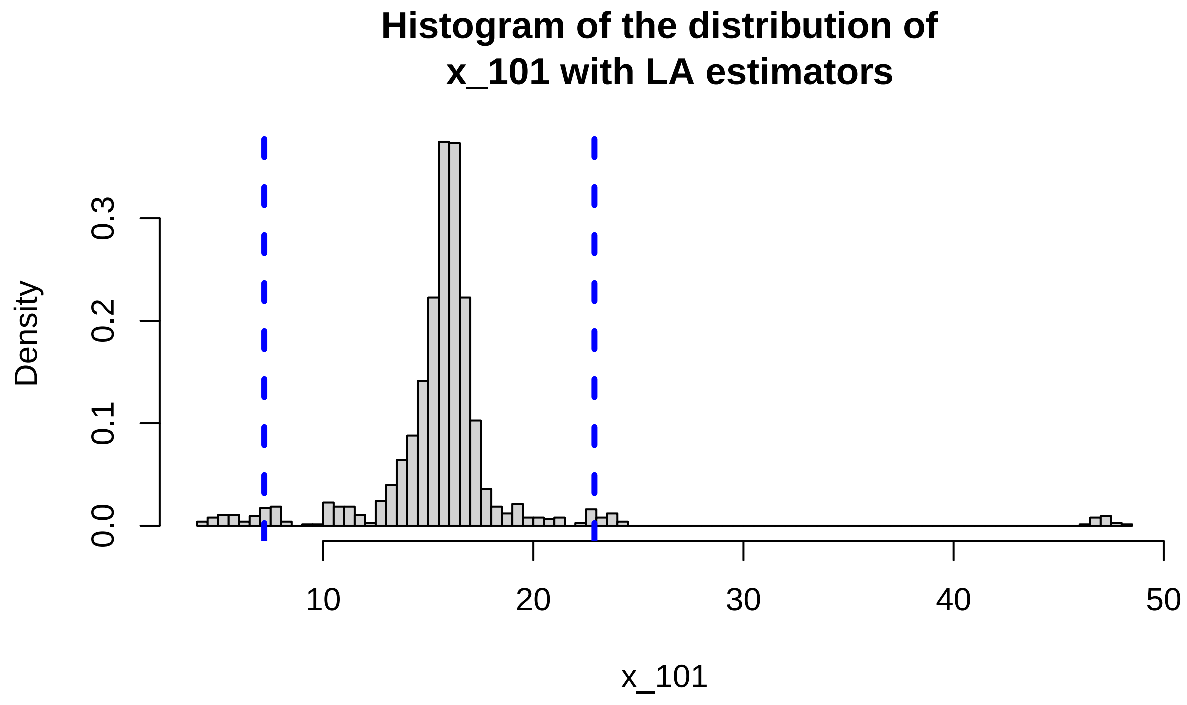


Figure 1: Here we see the histograms of our x101. The dotted blue lines are the 95 percent confidence intervals.

Problem B:

Introduction

We will in this problem use the F-statistic to perform a permutation test of the bilirubin data. The data contain measurements of the concentration of bilirubin (mg/dL) in blood samples taken from three young men. The data can be seen in the table below.

```r
bilirubin <- read.table("Files/bilirubin.txt", header = T)
```

Table 1: Concentration (mg/dL)

Individual 1:	0.14	0.20	0.23	0.27	0.27	0.34	0.41	0.41	0.55	0.61	0.66
Individual 2:	0.20	0.27	0.32	0.34	0.34	0.38	0.41	0.41	0.48	0.55	
Individual 3:	0.32	0.41	0.41	0.55	0.55	0.62	0.71	0.91			

1.

In this part we will use a boxplot to inspect the logarithm of the concentrations for each individual, i.e. the model,

$$log(Y_{ij}) = \beta_i + \epsilon_{ij}, \quad \text{with } i = 1,2,3 \text{ and } j = 1,\dots,n_i \tag{B.1}$$

where $n_1 = 11$, $n_2 = 10$ and $n_3 = 8$, and $\epsilon_{ij} \overset{iid}{\sim} \mathcal{N}(0,\sigma^2)$. This boxplot can be seen in Figure 2. We will then test the hypothesis that $\beta_1 = \beta_2 = \beta_3$ using the F-Test.

```r
boxplot(log(meas) ~ pers, bilirubin, col = c("red", "green",
    "blue"), main = "", xlab = "", names = c("Individual 1",
    "Individual 2", "Individual 3"), ylab = "log-values")
```

```r
mod <- lm(log(meas) ~ pers, bilirubin)
mod_summary = summary(mod)
Fval <- mod_summary$fstatistic[1]
cat("F-statistic:", Fval, "with", mod_summary$fstatistic[2],
    "and", mod_summary$fstatistic[3], "degrees of freedom.")
```

```
## F-statistic: 3.669775 with 2 and 26 degrees of freedom.
```

Our F-statistic is 3.6697751, which is larger than the critical value of 3.3690164, so the hypothesis is rejected. We can therefore say with some certainty that the individuals in the data are not equal.

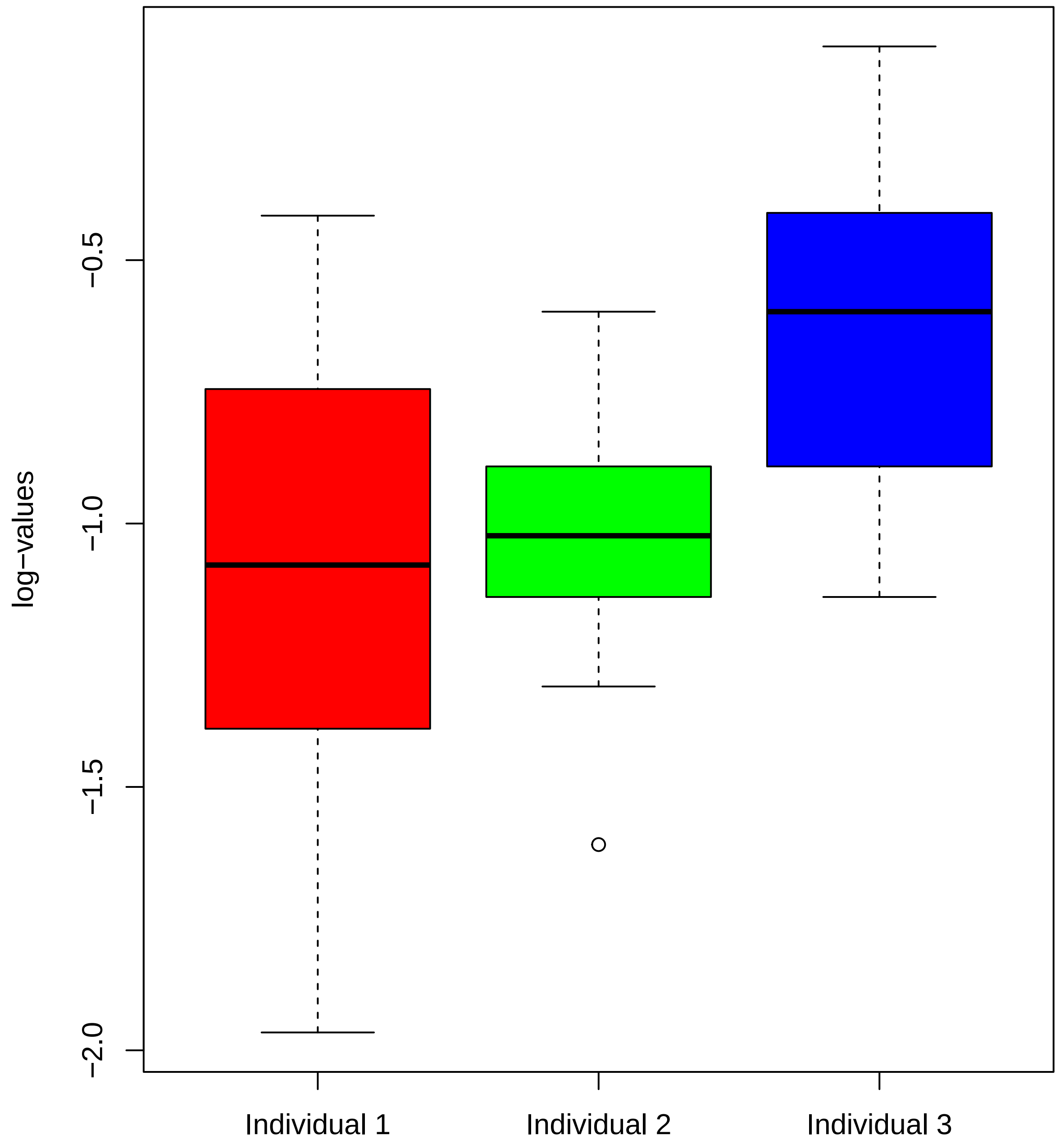


Figure 2: Here we have plotted the logarithmic value of the concentrations for each individual. From this we can already see, before performing the F-test, that the three individuals does not seem to be equal, but we cannot be certain.

2.

Here we will make a function called **permTest** which generates a permutation of the data, fits the model given in Eq. 2.1, and returns the value of the F-statistic for testing $\beta_1 = \beta_2 = \beta_3$.

```r
permTest <- function(data, seed) {
    set.seed(seed)
    # Permutate the data
    data$meas <- sample(data$meas)
    # Fit model
    mod <- lm(log(meas) ~ pers, data)
    # Finding F-statistic
    F_statistic <- summary(mod)$fstatistic[1]
    return(F_statistic)
}
```

3.

Do end this problem we will generate 999 samples using the function **permTest** and then find the p-value for **Fval** using these samples.

```r
n <- 999
F_statistic <- numeric(n)
# Running 999 iterations
for (i in 1:n) {
    F_statistic[i] <- permTest(bilirubin, 97 + i)
}
# Calculating p-value
p_value = sum(F_statistic > Fval)/n
cat("The p-value we get for our F-statistics is", p_value)
```

```
## The p-value we get for our F-statistics is 0.02902903
```

We get from the permutation test that our p-value is equal to 0.029029. This is sufficiently small so we can again reject the hypothesis that all of the individuals are equal. A histogram of our samples are shown in Figure 3.

```r
hist(F_statistic, xlab = "F-statistics", breaks = 100, main = NULL,
    freq = F, xlim = c(0, 6), ylim = c(0, 1), col = "darkgrey")
lines(seq(0, 6, 0.01), df(seq(0, 6, 0.01), df1 = mod_summary$fstatistic[2],
    df2 = mod_summary$fstatistic[3]), lwd = 5, col = "blue")
abline(v = Fval, lwd = 5, col = "red")
abline(v = quantile(F_statistic, probs = c(0.95)), lwd = 3, lty = 2,
    col = "red")
```

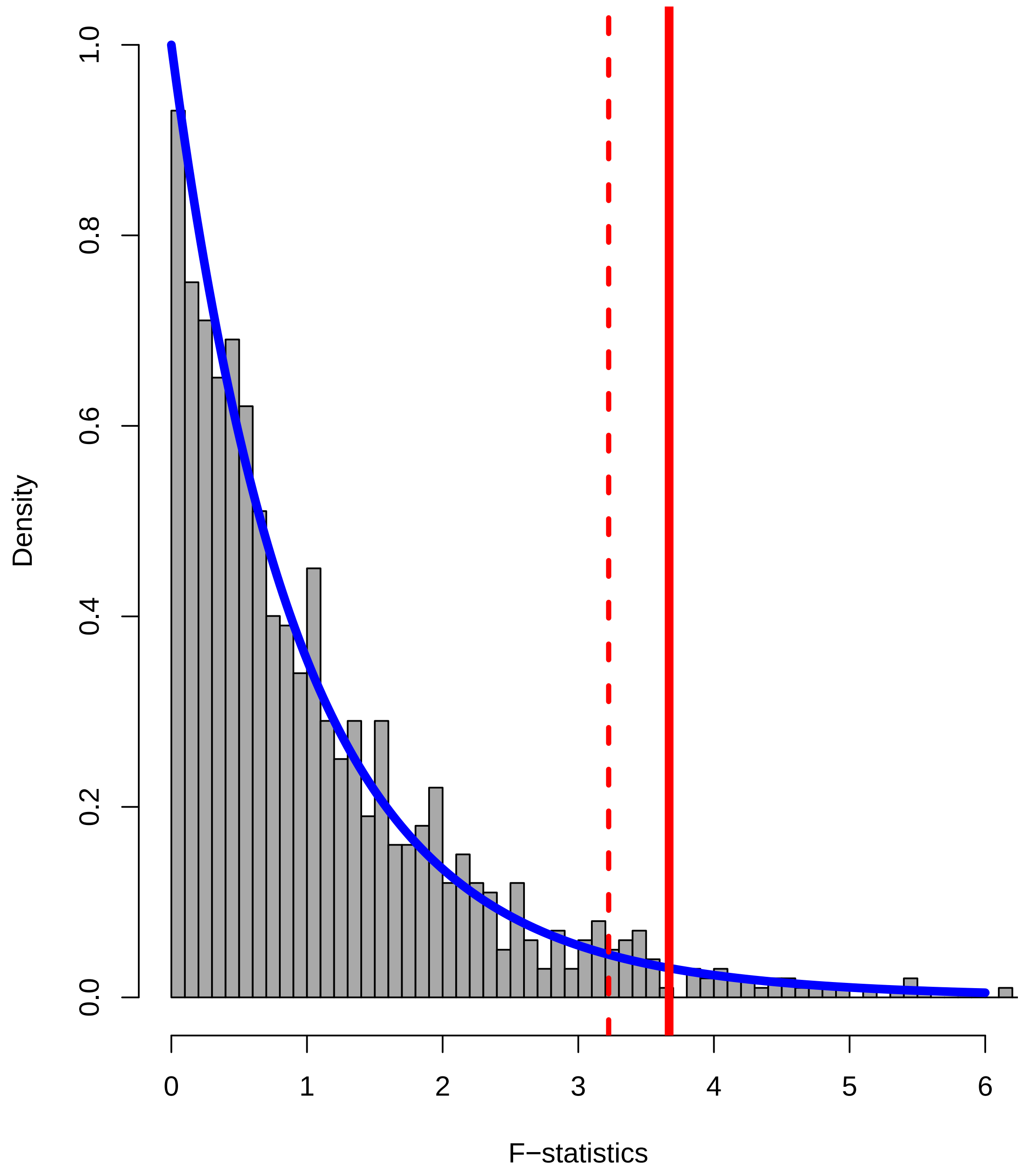


Figure 3: Histogram of our permutated F-statistics. Here we see that our samples follow the Fisher distribution with 2 and 26 degrees of freedom, the blue line. We also see that our F-statistic, the red line, is outside the 95 percent quantile, the red dotted line.

Problem C:

Introduction

In this final problem we want to use the EM algorithm to find the maximum likelihood estimate for (λ_0, λ_1). We let $x_1, \dots, x_n$ and $y_1, \dots, y_n$ be independent random variables, where the $x_i \overset{iid}{\sim} \text{Exp}(\lambda_0)$ and $y_i \overset{iid}{\sim} \text{Exp}(\lambda_1)$. We assume that we do not observe $x_1, \dots, x_n$ and $y_1, \dots, y_n$ directly, but that we observe

$$z_i = \max\{x_i, y_i\}, \tag{C.1}$$

and

$$u_i = \mathbf{I}(x_i \geq y_i), \tag{C.2}$$

for $i = 1, \dots, n$.

1.

We want to find the log likelihood, $f(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1)$, for the complete data (x_i, y_i), $i = 1, \dots, n$, and then use this to show that

$$\begin{aligned} E\Big[ln\Big(f(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1)\Big)|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\Big] &= n\Big(ln(\lambda_0) + ln(\lambda_1)\Big) \\ &- \lambda_0 \sum_{i=1}^{n}\left[u_i z_i + (1 - u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp\{\lambda_0^{(t)} z_i\} - 1}\right)\right] \\ &- \lambda_1 \sum_{i=1}^{n}\left[(1 - u_i) z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp\{\lambda_1^{(t)} z_i\} - 1}\right)\right]. \end{aligned}$$

We first derive the likelihood

$$f(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1) = \prod_{i=1}^{n} f(x_i, y_i|\lambda_0, \lambda_1) = \prod_{i=1}^{n} f(x_i|\lambda_0) f(y_i|\lambda_1) = \prod_{i=1}^{n} \lambda_0 \lambda_1 e^{-\lambda_0 x_i} e^{-\lambda_1 y_i},$$

and then we take the log of this to find the log likelihood

$$\begin{aligned} ln\Big[f(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1)\Big] &= ln\left[\prod_{i=1}^{n} f(x_i, y_i|\lambda_0, \lambda_1)\right] = \sum_{i=1}^{n}\Big[ln(\lambda_0) + ln(\lambda_1) - \lambda_0 x_i - \lambda_1 y_1\Big] \\ &= n\Big[ln(\lambda_0) + ln(\lambda_1)\Big] - \lambda_0 \sum_{i=1}^{n} x_i - \lambda_1 \sum_{i=1}^{n} y_i. \end{aligned}$$

Using the log likelihood we then find

$$E\Big[ln\Big(f(\mathbf{x},\mathbf{y}|\lambda_0,\lambda_1)\Big)|\mathbf{z},\mathbf{u},\lambda_0^{(t)},\lambda_1^{(t)}\Big] = n\Big(ln(\lambda_0)+ln(\lambda_1)\Big) - \lambda_0\sum_{i=1}^{n}E\Big[x_i|z_i,u_i,\lambda_0^{(t)}\Big] - \lambda_1\sum_{i=1}^{n}E\Big[y_i|z_i,u_i,\lambda_1^{(t)}\Big]. \tag{C.3}$$

For $E\Big[x_i|z_i,u_i,\lambda_0^{(t)}\Big]$, we have that either $u_i=1$ or $u_i=0$. We can then divide this expression into two parts, and when $u_i=1$, then $z_i=x_i$, and when $u_i=0$, then $z_i=y_i$. So we get

$$E\Big[x_i|z_i,u_i,\lambda_0^{(t)}\Big] = u_iE\Big[x_i|z_i=x_i,u_i=1,\lambda_0^{(t)}\Big] + (1-u_i)E\Big[x_i|z_i=y_i,u_i=0,\lambda_0^{(t)}\Big],$$

where

$$E\Big[x_i|z_i=x_i,u_i=1,\lambda_0^{(t)}\Big] = z_i,$$

$$\begin{aligned} E\Big[x_i|z_i=y_i,u_i=0,\lambda_0^{(t)}\Big] &= \int_0^{z_i} x_i f(x_i|z_i=y_i,u_i=0,\lambda_0^{(t)}) \\ &= \int_0^{z_i} x_i\frac{\lambda_0^{(t)}\exp\{-\lambda_0^{(t)}x_i\}}{1-\exp\{-\lambda_0^{(t)}z_i\}}dx_i \\ &= \frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp\{\lambda_0^{(t)}z_i\}-1}. \end{aligned}$$

We therefore get the expression

$$E\Big[x_i|z_i,u_i,\lambda_0^{(t)}\Big] = u_iz_i + (1-u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp\{\lambda_0^{(t)}z_i\}-1}\right). \tag{C.4}$$

Similarly for $E\Big[y_i|z_i,u_i,\lambda_1^{(t)}\Big]$ we get

$$E\Big[y_i|z_i,u_i,\lambda_1^{(t)}\Big] = (1-u_i)z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp\{\lambda_1^{(t)}z_i\}-1}\right). \tag{C.5}$$

Plugging Eq. C.4 and C.5 into Eq. C.3 we get the desired expression.

2.

Now we will use the EM algorithm to find the maximum likelihood estimate for (λ_0, λ_1). To do this we first need to find the recursion in $(\lambda_0^{(t)}, \lambda_1^{(t)})$. To find this recursion we take the derivative of the expression we showed in Problem C:1, here called $Q(\cdot)$, to find an expression that maximizes each value. These recursion can be seen in Eq. C.6 and C.7.

$$\frac{\partial Q(\cdot)}{\partial \lambda_0} = \frac{n}{\lambda_0} - \sum_{i=1}^{n}\left[u_i z_i + (1-u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp\{\lambda_0^{(t)} z_i\} - 1}\right)\right] = 0,$$
$$\frac{\partial Q(\cdot)}{\partial \lambda_1} = \frac{n}{\lambda_1} - \sum_{i=1}^{n}\left[(1-u_i) z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp\{\lambda_1^{(t)} z_i\} - 1}\right)\right] = 0.$$

We then get

$$\lambda_0^{(t+1)} = \frac{n}{\sum_{i=1}^{n}\left[u_i z_i + (1-u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp\{\lambda_0^{(t)} z_i\}-1}\right)\right]}, \tag{C.6}$$

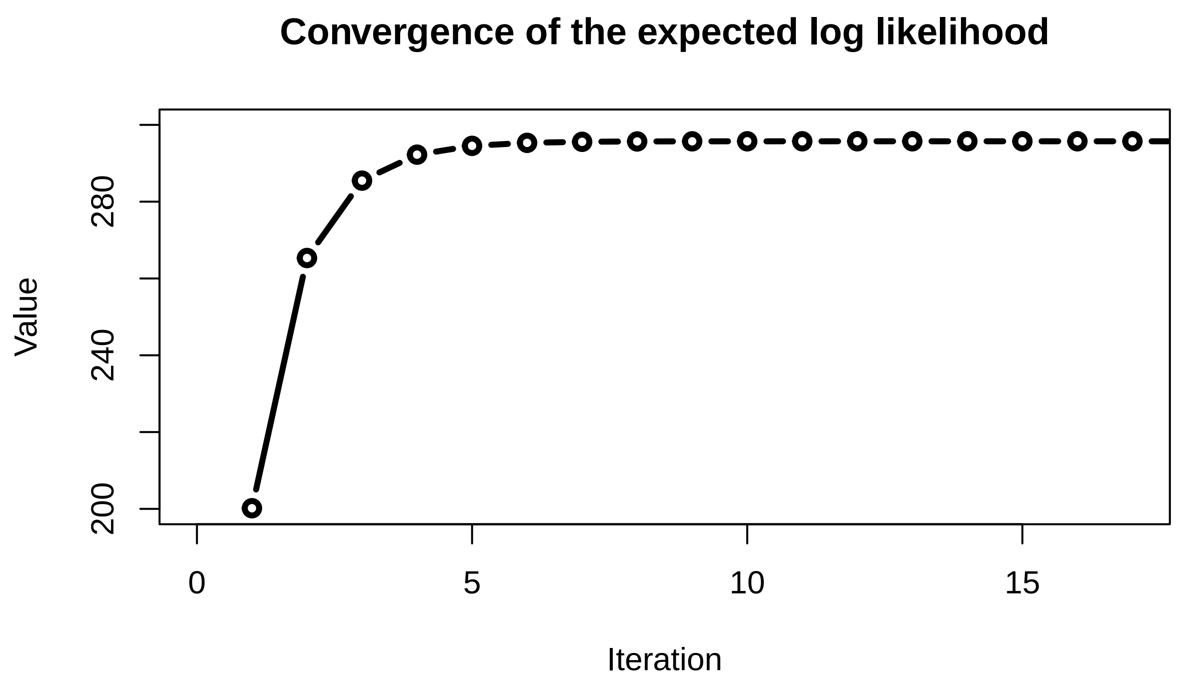
$$\lambda_1^{(t+1)} = \frac{n}{\sum_{i=1}^{n}\left[(1-u_i) z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp\{\lambda_1^{(t)} z_i\}-1}\right)\right]}. \tag{C.7}$$

```r
z <- read.table("Files/z.txt")
u <- read.table("Files/u.txt")
E_step <- function(lambda, lambdat, z, u) {
    # Eq. C.3:
    n <- max(nrow(z), length(z))
    first <- n * (log(lambda[1]) + log(lambda[2]))
    second <- lambda[1] * sum(u * z + (1 - u) * (1/lambdat[1] -
        z/(exp(lambdat[1] * z) - 1)))
    third <- lambda[2] * sum((1 - u) * z + u * (1/lambdat[2] -
        z/(exp(lambdat[2] * z) - 1)))
    return(first - second - third)
}
M_step <- function(lambdat, z, u) {
    n <- max(nrow(z), length(z))
    # Eq. C.6:
    lambda0new <- n/(sum(u * z + (1 - u) * (1/lambdat[1] - z/(exp(lambdat[1] *
        z) - 1))))
    # Eq. C.7:
    lambda1new <- n/(sum((1 - u) * z + u * (1/lambdat[2] - z/(exp(lambdat[2] *
        z) - 1))))
    return(c(lambda0new, lambda1new))
}
EMalg <- function(z, u, lambda0 = 1, lambda1 = 1, tol = 1e-10) {
    conv <- F
    lambdat <- c(lambda0, lambda1)
    lambdaall <- matrix(lambdat, ncol = 2)
    e.log.lik <- c()
    while (!conv) {
        lambda <- M_step(lambdat, z, u)
        e.log.lik <- c(e.log.lik, E_step(lambda, lambdat, z,
            u))
        if (length(e.log.lik) >= 2) {
            if (abs(diff(tail(e.log.lik, 2))) < tol) {
                conv <- T
            }
        }
        lambdat <- lambda
        lambdaall <- rbind(lambdaall, as.numeric(lambdat))
    }
    return(list(eloglikelihoods = e.log.lik, lambdas = lambdat,
        lambdaall = lambdaall))
}
EM <- EMalg(z, u)
par(mfrow = c(2, 1))
plot(1:length(EM$eloglikelihoods), EM$eloglikelihoods, xlim = c(0,
    17), ylim = c(200, 300), main = "Convergence of the expected log likelihood",
    xlab = "Iteration", ylab = "Value", type = "b", lwd = 3)
plot(0:length(EM$eloglikelihoods), EM$lambdaall[, 1], ylim = c(0,
    max(EM$lambdaall)), main = "Convergence of lambda", xlab = "Iteration",
    ylab = "Value", type = "b", lwd = 3, col = "blue")
lines(0:length(EM$eloglikelihoods), EM$lambdaall[, 2], type = "b",
    lwd = 3, col = "red")
legend("bottomright", c("lambda0", "lambda1"), col = c("blue",
    "red"), lwd = 3, inset = 0.05)
```

```r
cat("The final value of lambda0 is", tail(EM$lambdaall[, 1],
    1), "\nThe final value of lambda1 is", tail(EM$lambdaall[,
    2], 1))
```

```
## The final value of lambda0 is 3.465735
## The final value of lambda1 is 9.353215
```

From our EM algorithm we get that $\widehat{\boldsymbol{\lambda}} = (\widehat{\lambda}_0, \widehat{\lambda}_1)^T = (3.465735, 9.3532149)^T$. The algorithm was stopped when the difference between the old expected log likelihood and the new was smaller than 1e-5. The convergence of our algorithm can be seen in Figure 4.



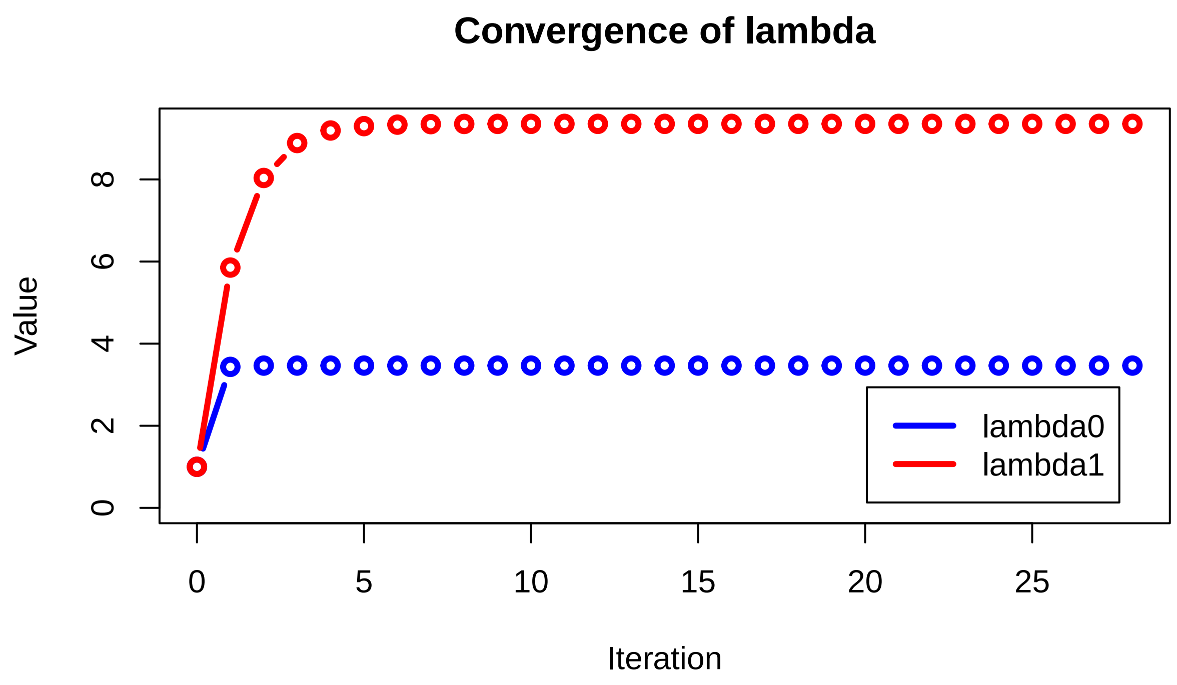


Figure 4: In the first plot we can see the convergence of the expected log likelihood. The second plot shows the convergence of our lambdas. The blue line is lambda0 and the red line is lambda1. We see also here the convergence of each value. We can in these two plots see that the algorithm stabilizes after about 5 iterations.

3.

To find the standard deviation and the biases of $\widehat{\boldsymbol{\lambda}}$ we will use bootstrapping. We will also find $\text{Corr}[\widehat{\lambda}_0, \widehat{\lambda}_1]$. The pseudocode for the bootstrapping algorithm can be seen below.

```
# Pseudocode:
# Start
Compute lambda0 and lambda1 from EM-algorithm
for b in 1:B:
  Draw x_1, ..., x_n from exp. distr. with intensity lambda0
  Draw y_1, ..., y_n from exp. distr. with intensity lambda1
  Set zz from Eq. C.1 and uu from Eq. C.2
  Use EM-algorithm with these new bootstrapped zz and uu to find new lambdas
# End
```

```r
set.seed(98)
B <- 10000
n <- nrow(z)
lambda <- EM$lambdas
lambdastorage <- data.frame(matrix(NA, ncol = 2, nrow = B))
for (b in 1:B) {
    x <- rexp(n, lambda[1])
    y <- rexp(n, lambda[2])
    zz <- pmax(x, y)
    uu <- ifelse(x >= y, 1, 0)
    lambdastorage[b, ] <- EMalg(zz, uu)$lambdas
}
```

Table 2: Standard Deviation, Bias, and Correlation of lambda0 and lambda1

	Standard Deviation:	Bias:	Correlation:
lambda0	0.2459392	0.0179099	-0.0132397
lambda1	0.8335450	0.1045277	-0.0132397

In the table above we see that the bias is small for both λ_0 and λ_1. If we use the bias corrected estimates, the estimates will not change that much, but the variance will increase. This means that is it best to use the uncorrected estimates of the estimators.

4.

However it is possible to find an analytically formula for the likelihood of our observations. For two observations z_i and u_i this can be expressed as,

$$f(z_i, u_i|\theta) \text{ where, } \theta = (\lambda_0, \lambda_1). \tag{C.8}$$

To continue the derivation, consider the random variable u_i. This can only take two values zero and one. Thus it its convenient to find the two PDFs corresponding to $u_i = 0$ and $u_i = 1$. Consider first the case when $u_i = 1$, which implies $X \geq Y$. The corresponding cumulative distribution will then become,

$$\begin{aligned} F(z_i, u_i = 1|\theta) &= P(Y \leq X, X \leq z_i|\theta) \\ &= \lambda_0\lambda_1 \int_0^{z_i} \int_0^x e^{-\lambda_0 * x} e^{-\lambda_1 * y} dy dx \\ &= \lambda_0\lambda_1 \int_0^{z_i} e^{-\lambda_0 * x} [\frac{-1}{\lambda_1}(e^{-\lambda_1 * x} - 1)] dx \\ &= -\lambda_0 \int_0^{z_i} e^{-(\lambda_0+\lambda_1)x} - e^{-\lambda_0} dx \\ &= \frac{\lambda_0}{\lambda_0+\lambda_1} e^{-(\lambda_0+\lambda_1)z_i} - \frac{\lambda_0}{\lambda_0+\lambda_1} - e^{-\lambda_0 z_i} + 1 \\ &= \frac{\lambda_0}{\lambda_0+\lambda_1}(e^{-(\lambda_0+\lambda_1)z_i} - 1) - (e^{-\lambda_0 z_i} - 1). \end{aligned} \tag{C.9}$$

For the calculation of $F(z_i, u_i = 0|\theta)$, we can use symmetry and see that because our boundary of the integral becomes, $\{(x, y) : 0 \leq y \leq z_i, 0 \leq y \leq x\}$. This gives the same result however λ_0 and λ_1 is interchanged:

$$\begin{aligned} F(z_i, u_i = 0|\theta) &= P(X \leq Y, Y \leq z_i|\theta) \\ &= \lambda_0\lambda_1 \int_0^{z_i} \int_0^y e^{-\lambda_0 * x} e^{-\lambda_1 * y} dx dy \\ &= \frac{\lambda_1}{\lambda_0+\lambda_1}(e^{-(\lambda_0+\lambda_1)z_i} - 1) - (e^{-\lambda_1 z_i} - 1). \end{aligned} \tag{C.10}$$

Finding the derivative of the cumulative distribution's we then obtain,

$$\begin{aligned} f(z_i, u_i = j|\theta) =& \frac{dF(z_i, u_i = j|\theta)}{dz_i} \\ =& \lambda_{1-j} e^{-\lambda_{1-j} z_i} - \lambda_{1-j} e^{-(\lambda_0+\lambda_1)z_i}, \end{aligned} \tag{C.11}$$

which is the probability density function of $f(z_i, u_i|\theta)$ corresponding to u_i evaluated at j. Note that it can be expressed as a fractioned probability density function,

$$f(z_i, u_i|\theta) = \begin{cases} \lambda_1 e^{-\lambda_1 z_i} - \lambda_1 e^{-(\lambda_0+\lambda_1)z_i}, \text{ for } u_i = 0 \\ \lambda_0 e^{-\lambda_0 z_i} - \lambda_0 e^{-(\lambda_0+\lambda_1)z_i}, \text{ for } u_i = 1. \end{cases} \tag{C.12}$$

So it is possible to find an analytically expression of $f(z_i, u_i|\theta)$.It is not possible to find analytically formulas for the maximum likelihood estimators. This becomes apparent when we try to derive them. Consider the log likelihood function,

$$\begin{aligned}
L(f) =& \Pi_{i=1}^{n} f(z_i, u_i|\theta)\\
l(f) =& \sum_{i=1}^{n} ln(I(u_i = 0)[\lambda_1 e^{-\lambda_1 z_i} - \lambda_1 e^{-(\lambda_0+\lambda_1) z_i}] + I(u_i = 1)[\lambda_0 e^{-\lambda_0 z_i} - \lambda_0 e^{-(\lambda_0+\lambda_1) z_i}]).\\
=& \sum_{u_i=0} ln(\lambda_1 e^{-\lambda_1 z_i} - \lambda_1 e^{-(\lambda_0+\lambda_1) z_i}) + \sum_{u_i=1} ln(\lambda_0 e^{-\lambda_0 z_i} - \lambda_0 e^{-(\lambda_0+\lambda_1) z_i})\\
=& \sum_{u_i=0} ln(\lambda_1) - \lambda_1 z_i + ln(1 - e^{-\lambda_0 z_i}) + \sum_{u_i=1} ln(\lambda_0) - \lambda_0 z_i + ln(1 - e^{-\lambda_1 z_i}).
\end{aligned}$$

If we take the derivative with regards to λ_1 and λ_2 we obtain,

$$\begin{aligned}
\frac{dl(f)}{d\lambda_1} &= \sum_{u_i=0} [\frac{1}{\lambda_1} - z_i)] + \sum_{u_i=1} [\frac{\lambda_1 e^{-\lambda_1 z_1}}{1 - e^{-\lambda_1 z_i}}]\\
\frac{dl(f)}{d\lambda_0} &= \sum_{u_i=1} [\frac{1}{\lambda_0} - z_i] + \sum_{u_i=0} [\frac{\lambda_0 e^{-\lambda_0 z_1}}{1 - e^{-\lambda_0 z_i}}]
\end{aligned}$$

It is clear that it is not possible to find an analytically solution to these two equations set to zero. Thus we will find it numerically using the log likelihood function.

```r
u_data <- read.delim("Files/u.txt")
z_data <- read.delim("Files/z.txt")
# Something went a bit wrong with the data So i manually
# added the data point lost
u <- u_data$X1
u <- c(1, u)
z <- z_data$X0.2452096
z <- c(0.2452096, z)
log_likelihod_f <- function(par) {
    return(-1 * (sum(log((u < (1/2)) * ((par[2]) * exp(-par[2] *
        z) - (par[2]) * exp(-(par[1] + par[2]) * z)) + (u > (1/2)) *
        ((par[1]) * exp(-par[1] * z) - (par[1]) * exp(-(par[1] +
            par[2]) * z))))))
}
optim(par = c(2, 2), log_likelihod_f, method = "L-BFGS-B")$par
```

```
## [1] 3.465735 9.353215
```

This gives the solution of $\lambda_0 = 3.466$ and $\lambda_1 = 9.353$. We can see that the values we get by numerically minimizing the log likelihood function with respect to λ_0 and λ_1 gives approximately the same result as with the EM-algorithm.

The benefits of directly calculating our parameters through the log likelihood function when it is possible is computational time. We are guaranteed that the EM-algorithm will converge however it does acquire more computational power. Further more the EM-algorithm can converge to local optima which is not possible when we have an analytically expression for the estimator of our parameters.