Compulsory exercise 3: Group 5

TMA4300 Computational Statistics V2022

Markus Johnsen Aase, Nora Ræhnebæk Aasen

28 april, 2022

Contents

1 Problem A: Comparing AR(2) parameter estimators using resampling of residuals

```
source("probAhelp.R")
source("probAdata.R")
```

The data above contains a sequence of length $T = 100$ of a non-Gaussian timeseries. We will use the data to compare two different parameter estimators.

We have an AR(2) model which is specified by the relation

$$x_t = \beta_1 x_{t-1} + \beta_2 x_{t-2} + e_t$$

where e_t are iid random variables with mean zero and constant variance.

The least sum of squared residuals (LS) and least sum of absolute residuals (LA) are obtained by minimizing the following loss functions with respect to β:

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

$$Q_{LA}(\mathbf{x}) = \sum_{t=3}^{T}|x_t - \beta_1 x_{t-1} + \beta_2 x_{t-2}|$$

We denote the minimizers as $\hat{\beta}_{LS}$ and $\hat{\beta}_{LA}$ and the estimated residuals are defined to be $\hat{e}_t = x_t - \hat{\beta}_1 x_{t-1} - \hat{\beta}_2 x_{t-2}$ for $t = 3, \ldots, T$. Where $\bar{e}$ is the mean of these. We re-center $\hat{e}_t$ to have mean zero by defining: $\hat{\epsilon}_t = \hat{e}_t - \bar{e}$

1.1 1)

```
x = data3A$x                #The data we're investegating
n = length(x)               #Length of the data, T = 100
beta = ARp.beta.est(x, 2)   #Using implemented/downloaded function for beta, gives us beta_1 and beta_2

obs_LA_e = ARp.resid(x, beta$LA)   #Calculate the observed residual (e) for Beta_LA (downloaded functio
obs_LS_e = ARp.resid(x, beta$LS)   #Calculate the observed residual (e) for Beta_LS (downloaded functio


set.seed(1234)          #Set seed for reproducible result.
B = 1500                #Number of Bootstrap samples.

beta_LA = beta[[2]]     #Beta_1 and Beta_2 for LA
beta_LS = beta[[1]]     #Beta_1 and Beta_2 for LS

#creating matrix for data storage (bootstrapped samples) later on.
boot_beta_LA = matrix(data = NA, nrow = length(beta_LA), ncol = B, dimnames = NULL)
boot_beta_LS = matrix(data = NA, nrow = length(beta_LS), ncol = B, dimnames = NULL)

for (i in 1:B) {

 u = sample(1:99, 1)     #Finding a random x_1 for initializing.

 #Vector of the to consecutive x_1 and x_2.
 x0 = c(x[u], x[u+1])


 #Creating bootstrap samples using the sample function, form the observed residual, for LS and LA.
 resample_residual.LA = sample(obs_LA_e, n, TRUE)
 resample_residual.LS = sample(obs_LS_e, n, TRUE)



 x_boot_LS = ARp.filter(x0, beta_LS, resample_residual.LS)[3:(n+2)]   #Using implemented function to gen
 x_boot_LA = ARp.filter(x0, beta_LA, resample_residual.LA)[3:(n+2)]   #Using implemented function to gen

 #Estimate the betas, using the downloaded function.
 beta_1 = ARp.beta.est(x_boot_LS, 2)
 beta_2 = ARp.beta.est(x_boot_LA, 2)
 #Save data in existing matrices.
 boot_beta_LS[,i] = beta_1[[1]] #Saving the LS beta's.
 boot_beta_LA[,i] = beta_2[[2]] #Saving the LA beta's.

}

#Code for calculating variance of LS and LA
beta_LS_var = c(var(boot_beta_LS[1,]), var(boot_beta_LS[2,]))
beta_LA_var = c(var(boot_beta_LA[1,]), var(boot_beta_LA[2,]))

#Code for calculating bias of LS and LA
beta_LS_mean = c(mean(boot_beta_LS[1,]), mean(boot_beta_LS[2,]))  #Calculating mean first.
beta_LA_mean = c(mean(boot_beta_LA[1,]), mean(boot_beta_LA[2,]))  #Calculating mean first.
beta_LS_bias = beta_LS_mean - beta_LS   #Calculating the bias.
beta_LA_bias = beta_LA_mean - beta_LA   #Calculating the bias.


# Organize the results in a matrix
M = matrix(c(round(beta_LS_var,5), round(beta_LS_bias,5),round(beta_LA_var,5), round(beta_LA_bias,5)), 

colnames(M) = c('var_B1','var_B2','bias_B1', 'bias_B2')
rownames(M) = c('LS','LA')

M
```

```
##     var_B1  var_B2  bias_B1 bias_B2
## LS 0.00559 0.00547 -0.01705 0.01088
## LA 0.00039 0.00038 -0.00187 0.00138
```

We know that the Least Sum of Squared Residuals (LS) estimator is optimal for Gaussian AR(p) processes. However, from our printed results above we can see that Least Sum of Absolute (LA) residuals has both lower variance and bias than LS. This can definitely make sense, because we were informed that the data is a non-Gaussian time-series. Hence, for this dataset LA was better then LS.

1.2 2)

In this task we're going to compute a 95% PI (Prediction Interval) for the 101th x, specifically x_{101}. This will be based on both estimators, LS and LA. So we use the bootstrapped time series and parameter estimates we obtained from part A-1 to estimate the corresponding residual distribution and then use this to simulate a value x_{101} for the observed time series. Here we will find x_{101} using $x_t = \beta_1 x_{t-1} + \beta_2 x_{t-2} + e_t$.

```
# set seed and number of boostrap samples
B2 = 2000
set.seed(1234)

# make empty vectors for storing the predictions
x_101_LA = rep(0, B2)
x_101_LS = rep(0, B2)

# make a bootstrap sample of the observed errors
res_LA = sample(obs_LA_e, B2, TRUE)
res_LS = sample(obs_LS_e, B2, TRUE)
```

```
for (i in (1:B2)){
  a = sample(1:B,4, replace = T) # sample 4 random values: these values will be
                                 # used to extract random betas from the
                                 # bootstrap samples in A1.

  beta_LA = c(boot_beta_LA[1,a[1]],boot_beta_LA[2,a[2]])
  beta_LS = c(boot_beta_LS[1,a[3]],boot_beta_LS[2,a[4]])

  # predict x_101 using our samples betas and errors and the AR(2) model
  x_101_LA[i] = beta_LA[1]*x[99]+beta_LA[2]*x[100]+res_LS
  x_101_LS[i] = beta_LS[1]*x[99]+beta_LS[2]*x[100]+res_LS
}

# Calculate quantiles as desired.
quantile_LA = quantile(x_101_LA, c(0.025, 0.975))
quantile_LS = quantile(x_101_LS, c(0.025, 0.975))

# Organize the results in a matrix
N = matrix(c(round(quantile_LS,5),round(quantile_LA,5)), nrow = 2, byrow = T)

rownames(N) = c('LS','LA')
colnames(N) = c('Lower', 'Upper')

N
```

```
##       Lower    Upper
## LS 12.51031 19.44620
## LA 14.96025 17.01246
```

In the printed result above we see the values of the 95% prediction intervals for x_{101} for both LS and LA-methods. There are two reasons why the PI's are so large: uncertainty in the model and uncertainty in the parameter estimates. Both prediction intervals are based on the same model, and therefore they have the same model uncertainty. Therefore the PI using $\hat{\beta}_{LA}$ is smaller since this estimator has a lower variance than $\hat{\beta}_{LS}$.

2 Problem B: Permutation test

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

# investigate the dataset
head(bilirubin)
```

```
##   meas pers
## 1 0.14   p1
## 2 0.20   p1
## 3 0.23   p1
## 4 0.27   p1
## 5 0.27   p1
## 6 0.34   p1
```

```r
tail(bilirubin)
```

```
##    meas pers
## 24 0.41   p3
## 25 0.55   p3
## 26 0.55   p3
## 27 0.62   p3
## 28 0.71   p3
## 29 0.91   p3
```

```r
#Creating vectors of the different measurments of concentrations
individ1_values = c(bilirubin$meas)[1:11]
individ2_values = c(bilirubin$meas)[12:21]
individ3_values = c(bilirubin$meas)[22:29]
```

2.1 1)

We use `lm` to fit the regression model $\log Y_{ij} = \beta_i + \epsilon_{ij}$ where $i = 1, 2, 3$ and $j = 1, \ldots, n_i$. Where the number of observations for the first, second and third individual is $n_1 = 11, n_2 = 10$ and $n_3 = 8$, respectively. And ϵ_{ij} is iid $N(0, \sigma^2)$.

```r
# creating a boxplot of our data
log_bili = list(p1 = log(individ1_values), p2 = log(individ2_values), p3 = log(individ3_values))

boxplot(log_bili,
        names=c("Individ 1","Individ 2","Individ 3"),
        ylab = "log-values",
        main = NULL)
```

```r
#Creating the log-regression model
reg_model <- lm(log(meas) ~ pers, data = bilirubin)

#Inspecting the summary of the regression model
summary.lm(reg_model)
```

```
##
## Call:
## lm(formula = log(meas) ~ pers, data = bilirubin)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.87215 -0.26246  0.03131  0.20236  0.67844
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.09396    0.11747  -9.312 9.15e-10 ***
## persp2       0.06412    0.17023   0.377   0.7095
## persp3       0.46482    0.18104   2.568   0.0163 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

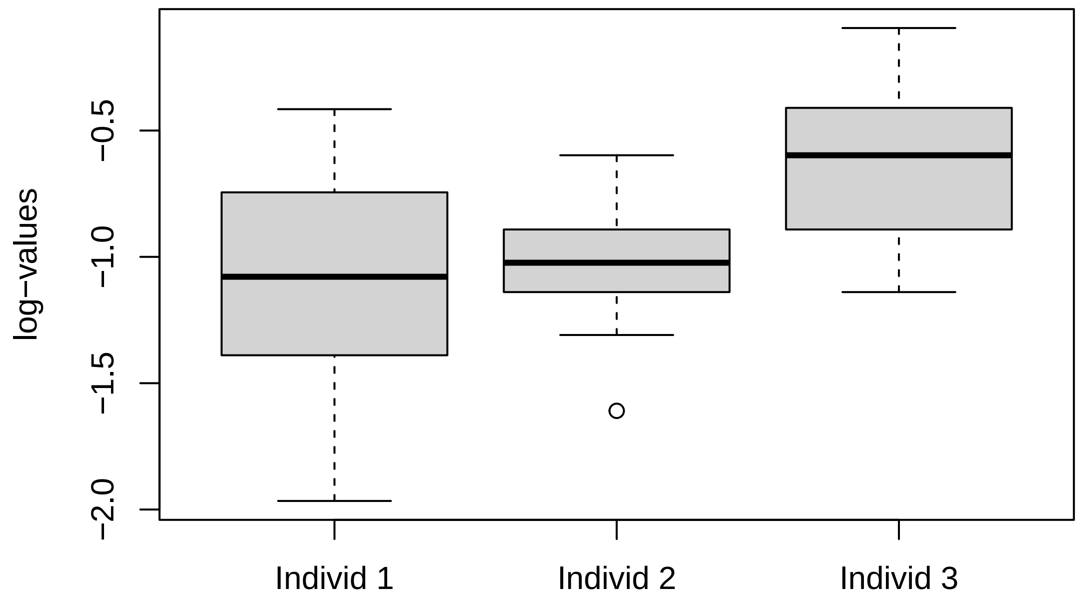

Figure 1: The logarithm of concentration for each individual.

```
## Residual standard error: 0.3896 on 26 degrees of freedom
## Multiple R-squared:  0.2201, Adjusted R-squared:  0.1602
## F-statistic:  3.67 on 2 and 26 DF,  p-value: 0.03946
```

```
Fval = summary.lm(reg_model)$fstatistic[1] #F-statistic value from summary.lm
```

In Figure 1 we see a box-plot that compares the concentration (on log-scale) of each individuals. There seem to be some different between each individual. We then use the F-test on the following hypothesis:

$$H_0 : \beta_1 = \beta_2 = \beta_3 \text{ vs } H_1 : \text{Not all equal}$$

We have a F-statstic with value 3.6698.The p-value from the test is less than 0.05, and we therefore reject the null-hypothesis that all intercepts are equal.

2.2 2)

Here we write a function that permutes our data and returns the F-value and p-value of the same test as before. The idea here is to mix the data to see if there truly is difference between the defined groups in our dataset.

```
permtest <- function(data){

  data$meas = sample(data$meas) # shuffle the measurements

  reg_model <- lm(log(meas) ~ pers, data = data) # fit the same model as before

  A = anova(reg_model,test = "F") # perform an F-test on our model

  # return F-value and p-value
  return(list(F_value = A$`F value`[1],
              p_value = A$`Pr(>F)`[1]))
}
```

2.3 3)

We now wish to shuffle our data multiple times and store each F-value for comparison with the originally computed F-value from problem 2.1.

```
# calculate the F-value for 999 different permutations
F_vec = numeric(999)
for (i in (1:999)){
  F_vec[i] = permtest(bilirubin)$F_value
}

# calculate the p-value
p_val = sum(F_vec > Fval)/999

# visualize the p-value
hist(F_vec, col="lightblue", xlab = "F-values",nclass = 50, main=NULL)
abline(v = Fval, col="orange", lwd=3, lty=1)
```

When permuting the data we got a p-value of 0.032, which is statistically significant. Hence, once more we reject the null-hypothesis that all betas are similar. The result is visualized in figure 2.

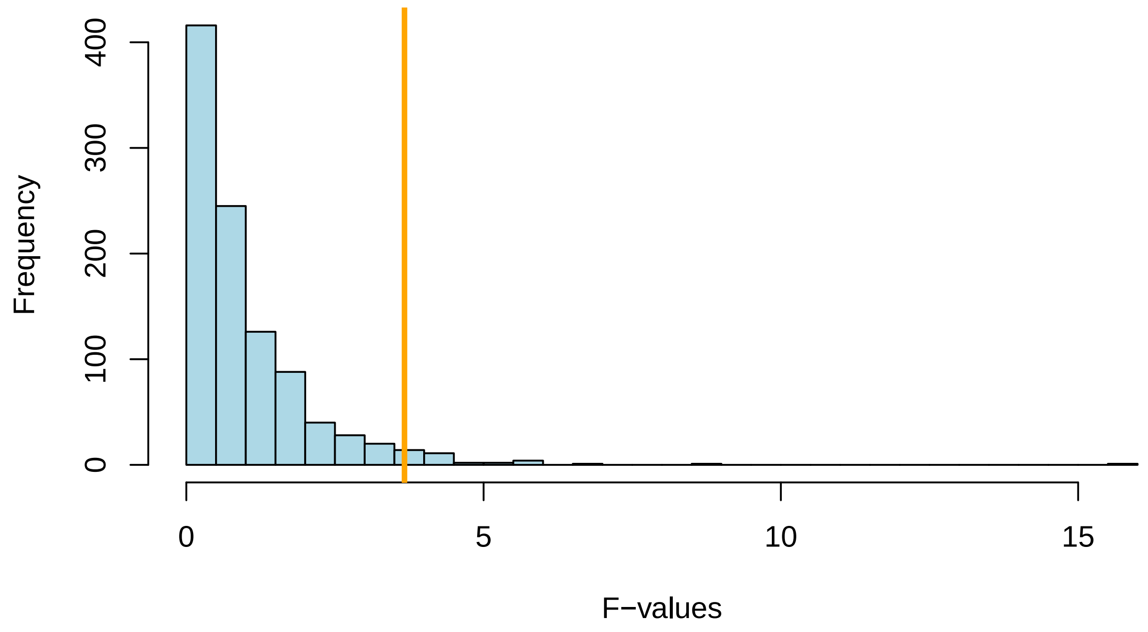

Figure 2: A histogram showing the F-values calculated from our permutations, and our original F-value marked in orange.

3 Problem C: The EM-algorithm and bootstrapping

In this task we let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be independent random variables, where the x_i´s and the y_i´s both have an exponential distribution with intensity λ_0 and λ_1, respectively. We assume that we don´t observe the x_i´s and y_i´s directly, but rather we observe

$$z_i = \max(x_i, y_i) \text{ for } i = 1, \ldots, n$$

and

$$u_i = I(x_i \geq y_i) \text{ for } i = 1, \ldots, n$$

Where $I(A) = 1$ if A is true and 0 otherwise. Based on the observed $(z_i, u_i),\ i = 1, \ldots, n$ we will use the EM algorithm to find the ML estimates for (λ_0, λ_1).

3.1 1)

We have that $x_1, \ldots, x_n \sim Exp(\lambda_0)$ and $y_1, \ldots, y_n \sim Exp(\lambda_1)$. The probability density function of $Exp(\lambda_0)$ is

$$f_{\mathbf{x}} = \lambda_0 \cdot \exp(-\lambda_0 \cdot \mathbf{x})$$

if $x \geq 0$ and 0 otherwise.

This means that the likelihood functions of x_i and y_i is

$$L(\lambda_0; x_1, \ldots, x_n) = \lambda_0^n \cdot \exp(-\lambda_0 \cdot \sum x_j) L(\lambda_1; y_1, \ldots, y_n) = \lambda_1^n \cdot \exp(-\lambda_1 \cdot \sum y_j)$$

but for the complete data $(x_i, y_i), i = 1, \ldots, n$ this becomes a bit different.

We want to find the log-likelihood of the complete data, and this we can derive from the full conditional distribution of the joint distribution of the complete data. Note: We're assuming that the x_i's and y_i's are independent!

First step is finding $f(x_i, y_i | \lambda_0, \lambda_1)$:

$$\begin{aligned} f(x_i, y_i | \lambda_0, \lambda_1) &= f(x_i | \lambda_0) \cdot f(y_i | \lambda_1) \\ &= \lambda_0 \lambda_1 \cdot \exp(-(\lambda_0 x_i + \lambda_1 y_i)) \end{aligned}$$

$$\implies f(\mathbf{x}, \mathbf{y} | \lambda_0, \lambda_1) = \prod_{i=1}^{n} f(x_i, y_i | \lambda_0, \lambda_1) = (\lambda_0 \cdot \lambda_1)^n \cdot \exp\left(-\lambda_0 \sum_i x_i - \lambda_1 \sum_i y_i\right)$$

$$\implies \ln f(\mathbf{x_i}, \mathbf{y_i} | \lambda_0, \lambda_1) = n \cdot (\ln\, \lambda_0 + \ln\, \lambda_1) - \lambda_0 \sum_i x_i - \lambda_1 \sum_i y_i$$

We have now found the log-likelihood function for the complete data $(x_i, y_i), i = 1, \ldots, n$.

$$\begin{aligned} E\left[\ln\, f(\mathbf{x_i}, \mathbf{y_i} | \lambda_0, \lambda_1) | \mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] &= n \cdot (\ln\, \lambda_0 + \ln\, \lambda_1) \\ &- \lambda_0 \sum_i E[x_i | z_i, u_i, \lambda_0^{(t)}] \\ &- \lambda_1 \sum_i E[y_i | z_i, u_i, \lambda_1^{(t)}]. \end{aligned}$$

Now we have to show that the expectations come out as desired.

We look at $E[x_i | z_i, u_i, \lambda_0^{(t)}]$ first:

$$E[x_i | z_i, u_i, \lambda_0^{(t)}] = u_i z_i + (1 - u_i) \int_0^{z_i} x_i \cdot f(x_i | \ldots) dx.$$

We now look at $E[y_i|z_i, u_i, \lambda_1^{(t)}]$:

$$E[y_i|z_i, u_i, \lambda_1^{(t)}] = (1-u_i)z_i + u_i \int_0^{z_i} y_i \cdot f(y_i|\ldots)dy.$$

This shows that we need to find the full conditional distributions for both x_i and y_i to calculate these expected values:

$$f\left(x_i|z_i, u_i, \lambda_0^{(t)}\right) = \begin{cases} z_i, & u_i = 1 \\ \frac{\lambda_0 \cdot \exp\left(-\lambda_0^{(t)} x_i\right)}{1-\exp\left(-\lambda_0^{(t)} z_i\right)}, & u_i = 0 \end{cases}$$

$$\begin{aligned} \implies E[x_i|z_i, u_i, \lambda_0^{(t)}] &= u_i z_i + (1-u_i)\int_0^{z_i} x_i \cdot \frac{\lambda_0 \cdot \exp\left(-\lambda_0^{(t)} x_i\right)}{1-\exp\left(-\lambda_0^{(t)} z_i\right)} dx \\ &= u_i z_i + (1-u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp(\lambda_0^t z_i) - 1}\right). \end{aligned}$$

This is the desired expected value from the task description.

Now we find the full conditional for y_i and check what the expected value is.

$$f\left(y_i|z_i, u_i, \lambda_1^{(t)}\right) = \begin{cases} z_i, & u_i = 0 \\ \frac{\lambda_1 \cdot \exp\left(-\lambda_1^{(t)} y_i\right)}{1-\exp\left(-\lambda_1^{(t)} z_i\right)}, & u_i = 1 \end{cases}$$

$$\begin{aligned} \implies E[y_i|z_i, u_i, \lambda_1^{(t)}] &= (1-u_i)z_i + u_i\int_0^{z_i} y_i \cdot \frac{\lambda_1 \cdot \exp\left(-\lambda_1^{(t)} y_i\right)}{1-\exp\left(-\lambda_1^{(t)} z_i\right)} dy_i \\ &= (1-u_i)z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp(\lambda_1^{(t)} z_i) - 1}\right). \end{aligned}$$

This is also the desired expectation and we've shown that

$$\begin{aligned} E\left[\ln f(\mathbf{x_i}, \mathbf{y_i}|\lambda_0, \lambda_1)|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] &= n \cdot (\ln \lambda_0 + \ln \lambda_1) \\ &\quad - \lambda_0 \sum_i u_i z_i + (1-u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp(\lambda_0^t z_i) - 1}\right) \\ &\quad - \lambda_1 \sum_i (1-u_i) z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp(\lambda_1^{(t)} z_i) - 1}\right). \end{aligned}$$

3.2 2)

Now we're going to find a recursion in $(\lambda_0^{(t)}, \lambda_1^{(t)})$ to find the MLE's of (λ_0, λ_1), and then we implement the recursion and find the MLE's when we use the specified data from the 'AdditionalFiles'.

In the EM algorithm we need to maximize a function with respect to the intensity of interest (λ_0 or λ_1). This function is often called $Q(\cdot)$, which is the expected value of the log-likelihood of the complete data $(x_i, y_i), i = 1, \ldots, n$.

$$
\begin{aligned}
Q(\cdot) = n \cdot (\ln \lambda_0 + \ln \lambda_1) \\
&- \lambda_0 \sum_i u_i z_i + (1-u_i) \left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp(\lambda_0^t z_i) - 1} \right) \\
&- \lambda_1 \sum_i (1-u_i) z_i + u_i \left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp(\lambda_1^{(t)} z_i) - 1} \right)
\end{aligned}
$$

Next we wish to use a recursive algorithm to find the MLEs of λ_0 and λ_1. First we calculate

$$\frac{\partial Q(\cdot)}{\partial \lambda_0} = 0, \quad \frac{\partial Q(\cdot)}{\partial \lambda_1} = 0,$$

and then we get the following recursive algorithms for our two parmeters:

$$\lambda_0^{(t+1)} = \frac{n}{\sum_i u_i z_i + (1-u_i) \left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp(\lambda_0^t z_i) - 1} \right)} \lambda_1^{(t+1)} = \frac{n}{\sum_i (1-u_i) z_i + u_i \left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp(\lambda_1^{(t)} z_i) - 1} \right)}$$

```r
# read data
u <- read.table("u.txt", header=T)
z <- read.table("z.txt", header=T)

# make a function that runs the EM algorithm
EM_alg <- function(u,z,l0 = 10,l1 = 10,its = 50){
  n = length(u)               # find n

  lambda_0 = l0               # find the starting points of our iteration
  lambda_1 = l1

  # empty vectors for storing
  lambda_0_list = numeric(its)
  lambda_1_list = numeric(its)

  for (i in (1:its)){
    # implement the iterative algorithm we found analytically
    lambda_0 = n/(sum(u*z + (1-u)*(1/lambda_0 - z/(exp(lambda_0*z)-1))))
    lambda_1 = n/(sum((1-u)*z + u*(1/lambda_1 - z/(exp(lambda_1*z)-1))))

    # store each value in the vectors
    lambda_0_list[i] = lambda_0
    lambda_1_list[i] = lambda_1
  }

return(list(lambda_0 = lambda_0_list,
            lambda_1 = lambda_1_list))
}

k = 15          # number of iterations

estimates = EM_alg(u$X1,z$X0.2452096,10,10,k)

# find rate of convergence (lambda_t+1 - lambda_t)
conv_0 = rev(diff(rev(estimates$lambda_0)))
conv_1 = rev(diff(rev(estimates$lambda_1)))
```

```r
# find the converged value
MLE_0 = round(estimates$lambda_0[k],5)
MLE_1 = round(estimates$lambda_1[k],5)

# plot of convergence
df = as.data.frame(list(x = c(1:(k-1)), y0 = conv_0, y1 = conv_1))
ggplot(data = df) +
  geom_line(aes(x,y0), color = 'darkgreen') +
  geom_line(aes(x,y1), color = 'orange') +
  ylab(expression(lambda^(t+1)-lambda^(t))) +
  xlab("iterations") +
  labs(caption = "green = lambda_0, orange = lambda_1")
```

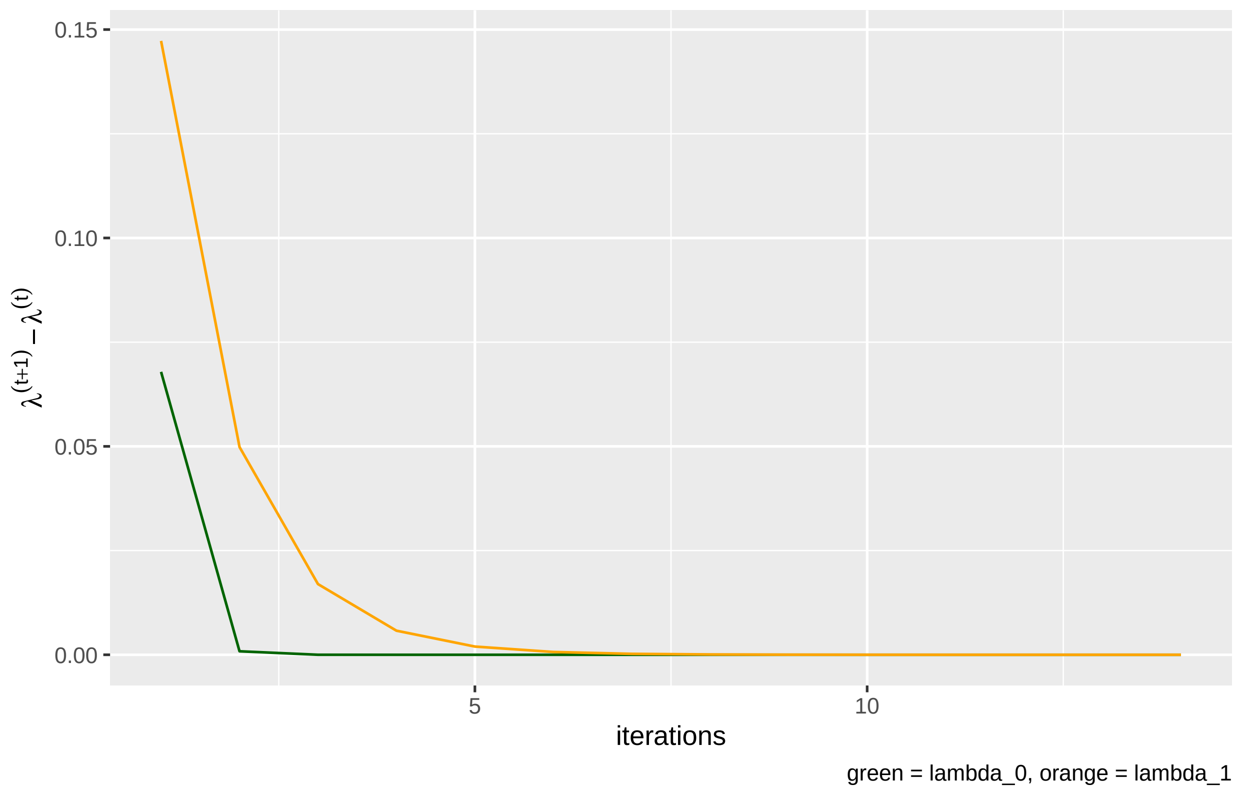

Figure 3: Plot that shows rate of convergence in the EM algorithm for the two parameters.

From figure 3 we see that the two parameters converge quickly to their estimated values. The MLEs we got from the EM algorithm was 3.46309 for $\hat{\lambda}_0$ and 9.33488 for $\hat{\lambda}_1$.

3.3 3)

We now wish to use bootstrapping to estimate the standard deviations and bias of $\hat{\lambda}_0$ and $\hat{\lambda}_1$, and to estimate $\text{Corr}[\hat{\lambda}_0, \hat{\lambda}_1]$. The pseudo-code for our algorithm is as follows:

```r
# This is the pseudo-code for our bootstrap algorithm
```

```
B = number of bootstrap samples
k = length of data

for 1,...,B:
  extract k pairs (u_i,z_i) with replacement

  use the EM algorithm on these pairs to get MLE for lambda_0 and lambda_1

  store the converged estimates in vectors
```

Below we implement our algorithm, and calculate our measures of interest.

```
# set seed and decide number of bootstrap samples
set.seed(1234)
B = 10000

# pair up the u and z data in one dataframe
uz_df = data.frame(u,z)

# find the length of our data
n = length(uz_df$X1)

its = 10

# emptu vectors for storing
lambda_0_vec = numeric(B)
lambda_1_vec = numeric(B)


for (i in (1:B)){
  rows = sample(1:n, n, replace = T)   # bootstrap which rows we will use

  uz_boot_df = uz_df[rows,]            # extract these rows from our data frame

  # use EM-algorithm on our boostrapped df
  EM_est = EM_alg(uz_boot_df$X1,uz_boot_df$X0.2452096, its = its)

  # store the converged value
  lambda_0_vec[i] = EM_est$lambda_0[its]
  lambda_1_vec[i] = EM_est$lambda_1[its]


}

# estimate std.dev
lambda_0_sd = sd(lambda_0_vec)
lambda_1_sd = sd(lambda_1_vec)

# estimate bias
lambda_0_mean = mean(lambda_0_vec)
lambda_1_mean = mean(lambda_1_vec)

lambda_0_bias = lambda_0_mean - estimates$lambda_0[15]
```

```r
lambda_1_bias = lambda_1_mean - estimates$lambda_1[15]

# estimate correlation
cor_01 = cor(lambda_0_vec, lambda_1_vec)

# store results in a matrix
P = matrix(c(round(lambda_0_mean,4), round(lambda_1_mean,4),
             round(lambda_0_sd,4),round(lambda_1_sd,4),
             round(lambda_0_bias,4),round(lambda_1_bias,4)), byrow = F, nrow = 2)

colnames(P) = c('mean','sd','bias')
rownames(P) = c('lambda_0', 'lambda_1')

P
```

```
##            mean     sd   bias
## lambda_0 3.4800 0.2488 0.0169
## lambda_1 9.4259 0.8057 0.0911
```

```r
par(mfrow=c(1,2))
# visualize the p-value
hist(lambda_0_vec, col="lightblue", xlab = expression(lambda[0]),main = NULL )
abline(v = lambda_0_mean, col="red", lwd=3, lty=1)
hist(lambda_1_vec, col="lightblue", xlab = expression(lambda[1]), main = NULL)
abline(v = lambda_1_mean, col="red", lwd=3, lty=1)
```

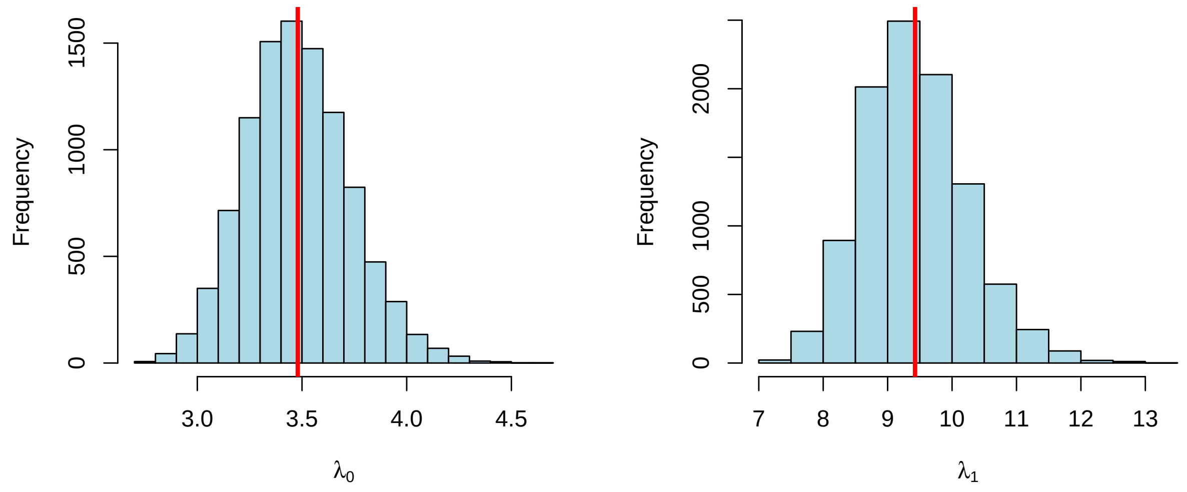

Figure 4: Histograms of the different lambda-values calculated from the bootstrap samples

The bootstrapped samples and the estimated mean is shown in figure 4. As we can see from both the table and the histograms, the standard deviation is significantly higher for $\hat{\lambda}_1$. The correlation is -0.0029. As we can see, there is no indication of a significant correlation between our two parameters, which makes sense as the x_i's and y_i's were assumed to be independent.

In our case we see that the bias is not that high for our maximum likelihood estimates. Therefore, we would not correct for the bias, as there would be a trade-off resulting in higher variance, which we deem not preferable in our case.

3.4 4)

We now want to find an analytical formula for the probability density function $f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1)$.

In problem 3.1 we've already found $f\left(x_i|z_i, u_i, \lambda_0^{(t)}\right)$ and $f\left(y_i|z_i, u_i, \lambda_1^{(t)}\right)$, for u_i, $i = 0, 1$.

We first set up a double-integral to find the CDF of our pdf of interest, and then we can use the fundamental theorem of calculus to find $f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1)$.

We start with the case when $u_i = 1 \implies z_i = \max(x_i, y_i) = x_i$:

$$F(z_i|u_i = 1, \lambda_0, \lambda_1) = \int_0^{z_i=x_i} \int_0^{z_i=x_i} f(x_i|\lambda_0) \cdot f(y_i|\lambda_1) dy_i dx_i$$

We remember that both pdf's are exponentially distributed. Hence

$$F(z_i|u_i = 1, \lambda_0, \lambda_1) = \int_0^{z_i=x_i} \int_0^{z_i=x_i} \lambda_0 \exp(-\lambda_0 x_i) \cdot \lambda_1 \exp(-\lambda_1 y_i) dy_i dx_i$$

$$F(z_i|u_i = 1, \lambda_0, \lambda_1) = \int_0^{z_i=x_i} \lambda_0 \exp(-\lambda_0 \cdot x_i) \cdot (-\exp(-\lambda_1 \cdot x_i) + 1) dx_i$$

$$\implies f(z_i|u_i = 1, \lambda_0, \lambda_1) = \lambda_0 \exp(-\lambda_0 \cdot z_i) \cdot (1 - \exp(-\lambda_1 \cdot z_i))$$

We now do the case where $u_i = 0 \implies z_i = \max(x_i, y_i) = y_i$ which yields a similar result:

$$\implies f(z_i|u_i = 0, \lambda_0, \lambda_1) = \lambda_1 \exp(-\lambda_1 \cdot z_i) \cdot (1 - \exp(-\lambda_0 \cdot z_i))$$

We now have an expression for $f(z_i|u_i = 1, \lambda_0, \lambda_1)$ and $f(z_i|u_i = 0, \lambda_0, \lambda_1)$. Combining these two gives:

$$f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1) = \begin{cases} \lambda_0 \exp(-\lambda_0 \cdot z_i) \cdot (1 - \exp(-\lambda_1 \cdot z_i), & u_i = 1 \\ \lambda_1 \exp(-\lambda_1 \cdot z_i) \cdot (1 - \exp(-\lambda_0 \cdot z_i)), & u_i = 0 \end{cases}$$

which is equivalent to

$$f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1) = \lambda_0 \exp(-\lambda_0 z_i)(1 - \exp(-\lambda_1 z_i) I(u_i = 1) + \lambda_1 \exp(-\lambda_1 z_i)(1 - \exp(-\lambda_0 z_i)) I(u_i = 0)$$

We have now found the pdf of interest and continue to find the log-likelihood function to find the maximum likelihood estimators $\hat{\lambda}_0$ and $\hat{\lambda}_1$.

$$\begin{aligned} \ln f(\mathbf{z}, \mathbf{u}|\lambda_0, \lambda_1) &= \ln\left(\prod_{i=1}^n f(z_i, u_i|\lambda_0, \lambda_1)\right) \\ &= \sum_{i=1}^n (\ln f(z_i, u_i|\lambda_0, \lambda_1)) \\ &= \sum_{i=1}^n \ln(\lambda_0 \exp(-\lambda_0 z_i)(1 - \exp(-\lambda_1 z_i) I(u_i = 1) + \lambda_1 \exp(-\lambda_1 z_i)(1 - \exp(-\lambda_0 z_i)) I(u_i = 0)) \\ &= \sum_{i:u_i=1} \ln(\lambda_0) - \lambda_0 z_i + \ln(1 - \exp(-\lambda_1 z_i)) + \sum_{i:u_i=0} \ln(\lambda_1) - \lambda_1 z_i + \ln(1 - \exp(-\lambda_0 z_i)) \end{aligned}$$

To find the MLE we find the derivative and set equal to zero. Denote by k_0 and k_1 the number of u_i's equal to 0 and 1, respectively.

$$\frac{\partial}{\partial\lambda_0}\ln f = \frac{k_1}{\lambda_0} - \sum_{i:u_i=1} z_i + \sum_{i:u_i=0} \frac{z_i\exp(-\lambda_0 z_i)}{1-\exp(-\lambda_0 z_i)}$$

$$\frac{\partial}{\partial\lambda_1}\ln f = \frac{k_0}{\lambda_1} - \sum_{i:u_i=0} z_i + \sum_{i:u_i=1} \frac{z_i\exp(-\lambda_1 z_i)}{1-\exp(-\lambda_1 z_i)}$$

We do not wish to solve this analytically, although it is possible, so instead we use `optimize` to find numerical solutions for the MLE.

```r
# split the data into cases where u=0 and u=1
u0z_df = uz_df$X0.2452096[which(uz_df$X1 == 0)]
u1z_df = uz_df$X0.2452096[which(uz_df$X1 == 1)]
k0 = length(u0z_df)
k1 = length(u1z_df)

# make two functions, one to optimize lambda_0 and lambda_1
l1 = 5 # since the derivative of lambda_0 doesn't depend on lambda_1, we
       # choose can choose any value at random for the function that will optimize lambda_0
f_0 <- function(l0){
  return(k1*log(l0)
         - l0*sum(u1z_df)
         + sum(log(1-exp(-l1*u1z_df)))
         + k0*log(l1)
         - l1*sum(u0z_df)
         + sum(log(1-exp(-l0*u0z_df))))
}

l0 = 5 # choose any random value
f_1 <- function(l1){
  return(k1*log(l0)
         - l0*sum(u1z_df)
         + sum(log(1-exp(-l1*u1z_df)))
         + k0*log(l1)
         - l1*sum(u0z_df)
         + sum(log(1-exp(-l0*u0z_df))))
}

# find the maximum of our log-likelihood functions
l0_MLE = round(optimize(f_0, interval = c(1e-16,100), maximum = T)$maximum,5)
l1_MLE = round(optimize(f_1, interval = c(1e-16,100), maximum = T)$maximum,5)
```

The results from obtimizing the likelihood is 3.46308 for $\hat{\lambda}_0$ and 9.33486 for $\hat{\lambda}_1$. These results are similar to what we got from the EM-algorithm, where we got 3.46309 for $\hat{\lambda}_0$ and 9.33488 for $\hat{\lambda}_1$. The EM-algorithm iteratively searches for the MLE, but it might converge to a local maximum. When we optimize the likelihood directly we always find the global maximum, and hence it is preferable if it is not too computationally hard.