Exercise 3

Thomas Torkildsen and Sivert Laukli

4/21/2022

This project consists of solving three different problems, problem A, B and C.

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

In this problem the time series will be approximated using two different parameter estimators for an **AR(2)** model. An **AR(2)** model is defined as below

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

where e_t are an iid random variable with zero mean and constant variance. The to different parameter estimators, which will be used to estimate β_1 and β_2, are the least sum of squared residuals (LS) and least sum of absolute residuals (LA). Which are found by minimizing the following loss functions (Q_{LS} and Q_{LA}) with respect to $\boldsymbol{\beta}$:

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

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

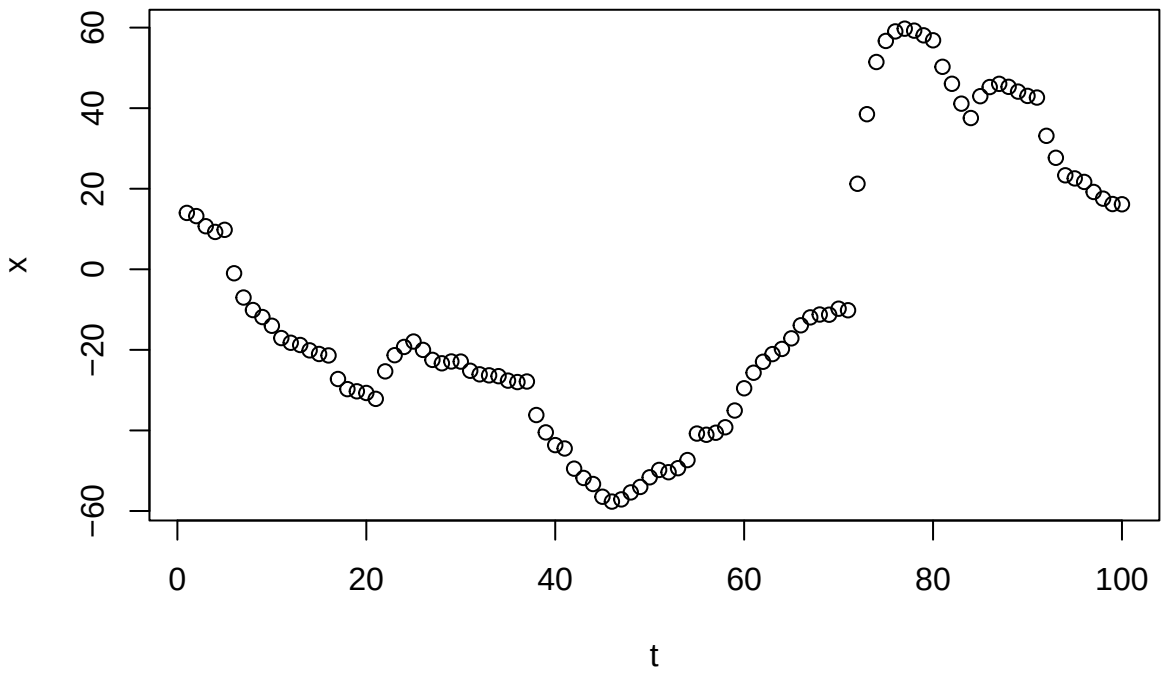
The β-values found from least sum of squared residuals will be denoted as $\hat{\boldsymbol{\beta}}_{LS}$ and from least sum of absolute residuals as $\hat{\boldsymbol{\beta}}_{LA}$.

In this task the pre-programmed files `probAhelp.R` and `probAdata.R` are is used. These files contains the time series used and some functions needed to solve the problem.

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

Below the time series is visualized and it is a clear correlation between the parameters.

```
x = data3A$x
plot(x,xlab = "t")
```



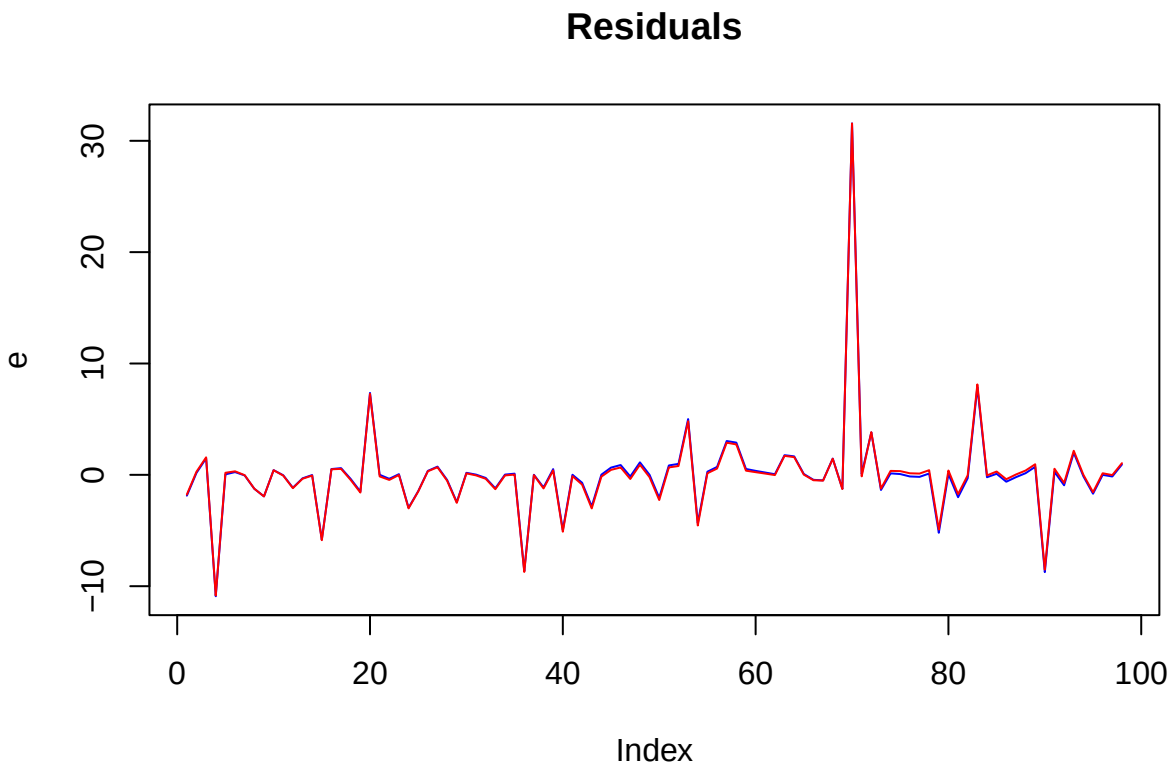
To find $\hat{\boldsymbol{\beta}}_{LS}$ and $\hat{\boldsymbol{\beta}}_{LA}$ the predefined function `ARp.beta.est` is used.

```
# Estimating the beta-values with LS and LA
betas = ARp.beta.est(x, p = 2)
betaLS = betas$LS
betaLA = betas$LA
betas
```

```
## $LS
## [1]  1.5528106 -0.5680178
##
## $LA
## [1]  1.5466096 -0.5575002
```

Now denote the estimated residuals as $\hat{e}_t = x_t - \hat{\beta}_1 x_{t-1} - \hat{\beta}_2 x_{t-2}$ for $t = 3, ..., T$ and let $\overline{e}$ be the mean of these. $\hat{e}_t$ is centered by defining $\hat{\epsilon}_t = \hat{e}_t - \overline{e}$. The results for $\hat{\epsilon}_t$ obtained by LS and LA and are calculated with the predefined function `ARp.resid`.

```
# Finding the estimated residuals
epsLA = ARp.resid(x, betaLA)
epsLS = ARp.resid(x, betaLS)
plot(epsLA, col = "blue", type = "l", main = "Residuals", ylab = "e")
lines(epsLS, col = "red")
```

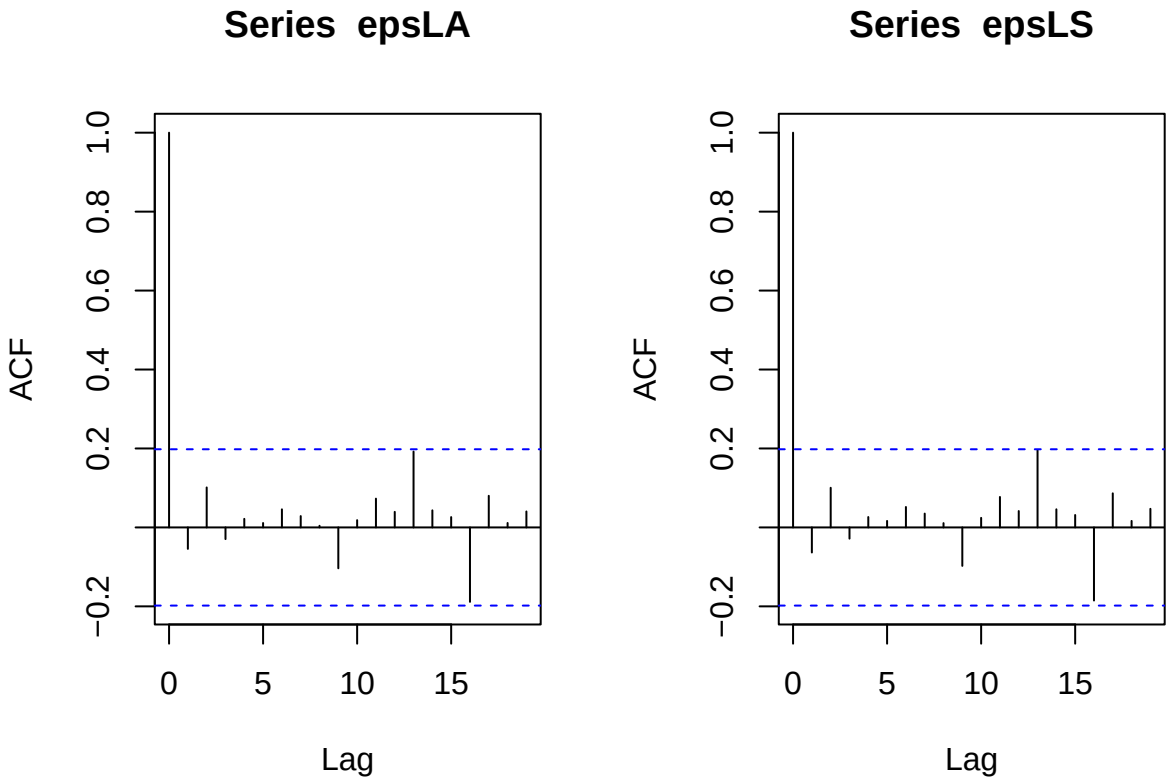


As seen above, the residuals are almost identical. More analysis is done in the next section.

1)

Here the residual resampling bootstrap method is used to evaluate the relative performance of the two parameter estimators. In order to perform the the residual resampling bootstrap method the estimated residuals need to be iid. Given that there are no significant correlations in the autocorrelation plots below both the plots will be assumed iid from this point on.

```
par(mfrow = c(1,2))
acf(epsLA)
acf(epsLS)
```



Residual resampling bootstrap is done by first drawing $T = 100$ random samples with replacement from the previously estimated residuals, defined as $\boldsymbol{\epsilon}^*$.The second step is generating pseudo data by

$$x_t^* = \hat{\beta}_1 x_{t-1} + \hat{\beta}_2 x_{t-2}^* + \hat{\epsilon}_t^*$$

Where $t = 3, ..., T$ and the initial values x_1 and x_2 are chosen as a random sub-sequence of $\mathbf{x}$. The second step is done by the function `ARp.filter`, which takes initial values, $\boldsymbol{\beta}$ and a vector of residuals as input and return a fitted sequence. Finally, the new time series´ corresponding β-values ($\hat{\boldsymbol{\beta}}_i^*$) are found using `ARp.beta.est`. This process is repeated $B = 1500$ times.

```
#install.packages("matrixStats")
library(matrixStats)

Te = 100
B = 1500

boot_resLA <- boot_resLS <- matrix(nrow = Te, ncol = B)
boot_AR_LA <- boot_AR_LS <- matrix(nrow = Te, ncol = B)
boot_betaLA <- boot_betaLS <- matrix(nrow = 2, ncol = B)

for (i in 1:B) {
  boot_resLA[,i]= sample(epsLA, size=Te, replace=TRUE)
  boot_resLS[,i] = sample(epsLS, size=Te, replace=TRUE)

  ind = sample(1:(Te-1), size = 2, replace = TRUE)
  boot_AR_LA[,i] = ARp.filter(x[ind[1]:(ind[1]+1)],betaLA, boot_resLA[,i])[3:(Te+2)]
  boot_AR_LS[,i] = ARp.filter(x[ind[2]:(ind[2]+1)],betaLS, boot_resLS[,i])[3:(Te+2)]

  boot_betaLA[,i] = ARp.beta.est(boot_AR_LA[,i], p = 2)$LA
  boot_betaLS[,i] = ARp.beta.est(boot_AR_LS[,i], p = 2)$LS
```

```
}
```

After all bootstrap estimates of $\boldsymbol{\beta}$ are calculated. Assuming that

```
betas = data.frame(LA = betaLA, mean_LA_boot = rowMeans(boot_betaLA),
                       var_LA_boot = rowVars(boot_betaLA), LS = betaLS,
                       mean_LS_boot =rowMeans(boot_betaLS),
                       var_LS_boot = rowVars(boot_betaLS))
row.names(betas) = c("beta_1", "beta_2")
betas
```

```
##                LA mean_LA_boot  var_LA_boot         LS mean_LS_boot var_LS_boot
## beta_1  1.5466096    1.5441397 0.0004063528  1.5528106    1.5406524 0.005104957
## beta_2 -0.5575002   -0.5555756 0.0004028458 -0.5680178   -0.5618016 0.004912395
```

The bootstrap estimate of the bias is defined as

$$\sum_{i=1}^{B} \frac{\hat{\boldsymbol{\beta}}_i^* - \hat{\boldsymbol{\beta}}}{B} = \overline{\boldsymbol{\beta}}^* - \hat{\boldsymbol{\beta}}$$

The bias and variance of the β-values from the two different methods are displayed in the table below.

```
bias_var = data.frame(biasLS = betas$mean_LS_boot - betas$LS, biasLA = betas$mean_LA_boot - betas$LA, va
                      var_LA_boot = betas$var_LA_boot )
row.names(bias_var) = c("beta_1", "beta_2")
bias_var
```

```
##              biasLS       biasLA var_LS_boot  var_LA_boot
## beta_1 -0.012158179 -0.002469871 0.005104957 0.0004063528
## beta_2  0.006216267  0.001924567 0.004912395 0.0004028458
```

The bias from the least sum of absolute residuals (LA) gives a result with less bias and less variance, then the least sum of squared residuals (LS). This is true for both β_1 and β_2. Thus, the LA estimator is a better method for fitting an AR(2) process to the given time series. Since a LS estimator is optimal a Gaussian AR(p) process, the times series data `data3A$x` simply cannot follow a Gaussian AR(2) process.

2)

In this section the bootstrapped time series and parameter estimates obtained in the previous section will be used to estimate the corresponding residual distribution and in turn use this to simulate a prediction of x_{101} for the observed time series. In order to take all the variance of the bootstrapped predictions of x_{101} into account. One sample of the bootstrapped prediction is defined as follows

$$x_{101}^* = \hat{\beta}_1^{**} x_{100} + \hat{\beta}_2^{**} x_{99} + \hat{\epsilon}_{101}^*$$

Where $\hat{\epsilon}_{101}^*$ is a randomly drawn residual calculated earlier, $\hat{\beta}_1^{**}$ and $\hat{\beta}_2^{**}$ are randomly from the earlier $\hat{\beta}^{**}$ samples. This yields the following distribution of predictions.

```
# function that takes 2 matrices of betas, 2 vectors of residuals and the time series x as input
# and returns a list with 2 vectors of predictions of x101
pred_x101 <- function(betasLA, betasLS, epsilonLA, epsilonLS, x) {
  x101_LA = c() # empty vector
  x101_LS = c() # empty vector
  for (i in 1:length(epsilonLA)){
    pred_LA = betasLA[1,i]*x[100] + betasLA[2,i]*x[99] + epsilonLA[i] #one LA prediction
    pred_LS = betasLS[1,i]*x[100] + betasLS[2,i]*x[99] + epsilonLS[i] #one LS prediction
    x101_LA = c(x101_LA, pred_LA) #insert prediction in vector
    x101_LS = c(x101_LS, pred_LS) #insert prediction in vector
```

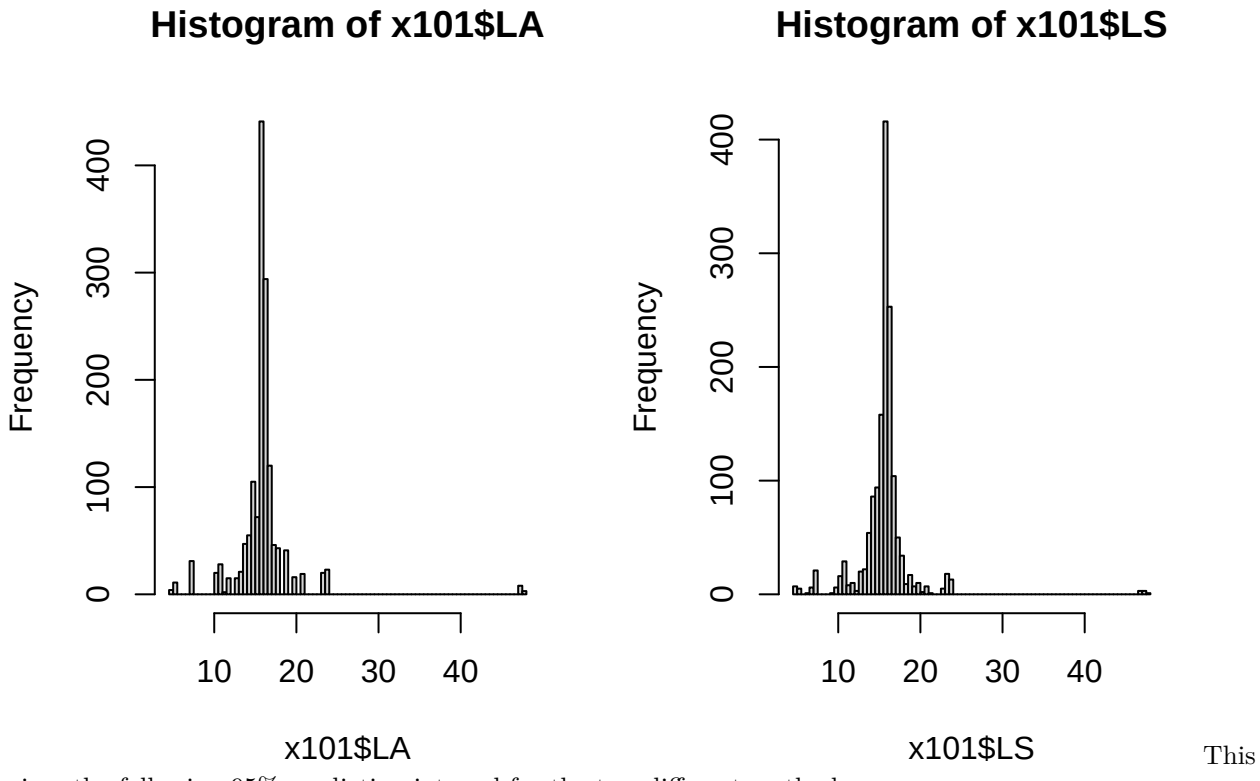
```
  }
  return(list(LA = x101_LA, LS = x101_LS ))
}
```

```
B = 1500 #number of boostrapped samples
set.seed(101)
epsLA_boot = sample(epsLA, size = B, replace = TRUE) #sampling residuals
epsLS_boot = sample(epsLA, size = B, replace = TRUE) #sampling residuals

ind_boot = sample(1:length(boot_betaLA[1,]), size = B, replace = TRUE) #bootstrap samples for betas
boot_betaLA2 = boot_betaLA[,ind_boot] #creating new bootstrapped data
boot_betaLS2 = boot_betaLS[,ind_boot] #creating new bootstrapped data

#predicting x101 for both methods
x101 = pred_x101(boot_betaLA2, boot_betaLS2, epsLA_boot, epsLS_boot, x)

par(mfrow = c(1,2))
hist(x101$LA, n = 100)
hist(x101$LS, n = 100)
```



This gives the following 95% prediction interval for the two different methods

```
quant_LA = quantile(x101$LA, c(0.025, 0.975))
quant_LS = quantile(x101$LS, c(0.025, 0.975))

data.frame("Quantile LA" = quant_LA,"Quantile LS" = quant_LS)
```

```
##       Quantile.LA Quantile.LS
## 2.5%     7.271576    7.405027
## 97.5%   23.297743   23.049796
```

Note that the 95% prediction interval for x_{101} is smaller for LS then for LA, this contradicts the result

obtained in the previous task.

C

2

In this section the data `z` and `u` will be used and is imported below.

```
#read u and z from file
u = read.delim("additionalFiles/u.txt")[[1]]
z = read.delim("additionalFiles/z.txt")[[1]]
```

Now we will use the EM algorithm to find a recursion in $(\lambda_0^{(t)}, \lambda_1^{(t)})$ for finding the maximum likelihood estimates for (λ_0, λ_1). when assuming that Q has a single maximum, it is attained when.

$$\frac{\partial Q}{\partial \lambda_0} = 0 \quad \text{and} \quad \frac{\partial Q}{\partial \lambda_1} = 0$$

Note that $\lambda_0^{(t)}$ and $\lambda_1^{(t)}$ are assumed constant. solving the above for λ_0 and λ_1 thus yields.

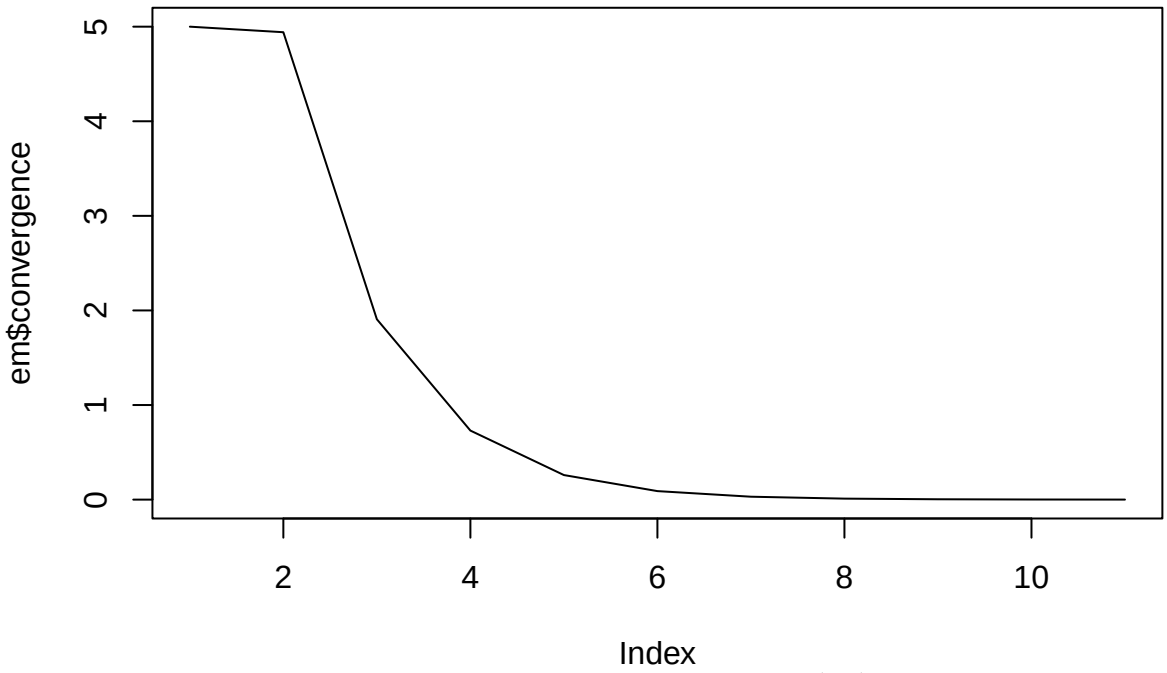
$$\lambda_0 = \frac{n}{\sum_{i=1}^{n}\left[(1-u_i)z_i + u_i\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp\left\{\lambda_0^{(t)} z_i\right\}-1}\right)\right]} \lambda_1 = \frac{n}{\sum_{i=1}^{n}\left[u_i z_i + (1-u_i)\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp\left\{\lambda_1^{(t)} z_i\right\}-1}\right)\right]}$$

Since the expectation is maximized by the lambdas above given $\lambda_0^{(t)}$ and $\lambda_1^{(t)}$, then we set $(\lambda_0^{(t+1)}, \lambda_1^{(t+1)}) = (\lambda_0, \lambda_1)$. After the new lambdas are found for the new step $(t+1)$. This process is repeated until convergence. convergence is defined as $\|(\lambda_0^{(t+1)}, \lambda_1^{(t+1)}) - (\lambda_0^{(t)}, \lambda_1^{(t)})\|_2$. The algorithm is implemented below.

```
EM <- function(u, z, tol) {
  n = length(z) #length of data
  # initial guess for lambda
  lambda0 = c(1)
  lambda1 = c(2)
  diff = c(5) #initial condition for while loop
  count = 1 # counter
  while (diff[count] > tol) {
    lam0 = n/sum(u*z + (1 - u)*(1/lambda0[count] - z/(exp(lambda0[count]*z) - 1))) #new lambda0
    lam1 = n/sum((1 - u)*z + u*(1/lambda1[count] - z/(exp(lambda1[count] * z) - 1))) #new lambda1

    d = sqrt((lambda0[count]-lam0)^2 + (lambda1[count] - lam1)^2) #updatring while condition
    diff = c(diff, d)
    lambda0 = c(lambda0, lam0)
    lambda1 = c(lambda1, lam1)
    count = count + 1
  }
  ret = list(convergence = diff, lambda0 = lambda0, lambda1 = lambda1, iterations = count)
  return(ret)
}
tol = 0.001
em = EM(u, z, tol)
#PLOT DETTE

plot(em$convergence, type = "l")
```



The convergence of the algorithm is visualized above. The final values of $(\hat{\lambda}_0, \hat{\lambda}_1)$ are.

```
data.frame(lambda0 = em$lambda0[em$iterations],lambda1 = em$lambda1[em$iterations])
```

```
##    lambda0 lambda1
## 1 3.463089 9.33466
```

3

Now we will use bootstrapping to estimate the standard deviations and biases of $(\hat{\lambda}_0, \hat{\lambda}_1)$ and to estimate $\text{Corr}[\hat{\lambda}_0, \hat{\lambda}_1]$. The boostrapping is done by resampling `z` and `u`, then fitting the EM algorithm to the new data, resulting in a new estimate of λ_0 and λ_1. This process is repeated B times. $\hat{\lambda}^*_{0,i}$ and $\hat{\lambda}^*_{1,i}$ are defined as the i´th bootstrapped estimate.

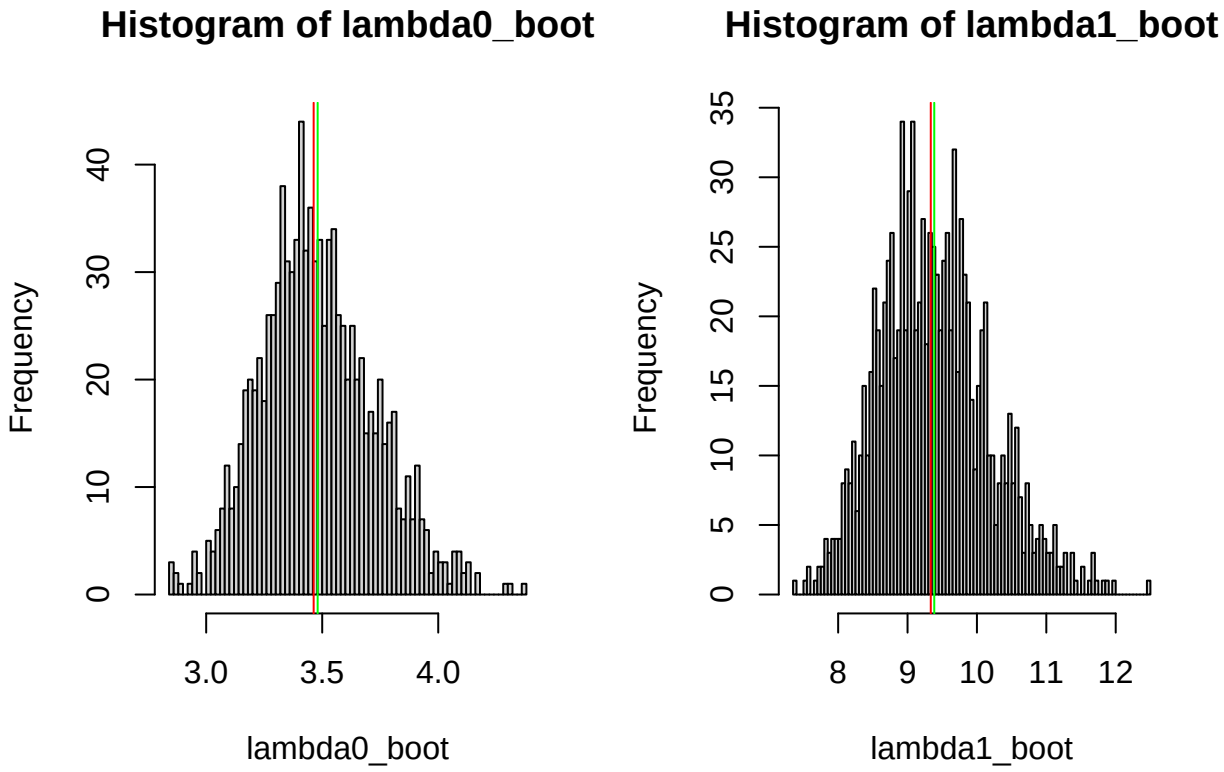
Psudocode for the bootstrap algorithm:

For i in 1:B do 1. sample n with replacement from the pair (z, u), where n is the length of (z, u) 2. Use the EM algorithm from the previous task to calculate $\hat{\lambda}^*_{0,i}$ and $\hat{\lambda}^*_{1,i}$ 3. Store the lambda-estimates from step 2.

The algorithm is implemented below.

```
B = 1000
n = length(z)
set.seed(1)
lambda0_boot = lambda1_boot = rep(0, B)
for (i in 1:B) {
  ind = sample(1:n, size = n, replace = TRUE)
  u_boot = u[ind]
  z_boot = z[ind]
  em_temp = EM(u_boot, z_boot, tol)
  lambda0_boot[i] = em_temp$lambda0[em_temp$iterations]
  lambda1_boot[i] = em_temp$lambda1[em_temp$iterations]
}
#histograms of the bootstrap estiamtes
par(mfrow = c(1,2))
hist(lambda0_boot, n = 100)
```

```
abline(v = c(mean(lambda0_boot), em$lambda0[em$iterations]), col = c("green","red"))
hist(lambda1_boot, n = 100)
abline(v = c(mean(lambda1_boot), em$lambda1[em$iterations]), col = c("green","red"))
```



In the histogram above the mean of the bootstrapped lambda estimates is drawn in green and the initial estimate in red. Clearly the bias is small for both values. The bias, standard deviation and correlation estimate is calculated and displayed below.

```
data.frame(correlation = cor(lambda0_boot, lambda1_boot), standard_dev_lambda0 = sd(lambda0_boot),
           standard_dev_lambda1 = sd(lambda1_boot),
           bias_lambda0 = mean(lambda0_boot) - em$lambda0[em$iterations],
           bias_lambda1 = mean(lambda1_boot) - em$lambda1[em$iterations])
```

```
##   correlation standard_dev_lambda0 standard_dev_lambda1 bias_lambda0
## 1  -0.0217066           0.2444434           0.7945142   0.01727289
##   bias_lambda1
## 1   0.05026008
```

The estimated correlation between λ_0 and λ_1 is small, practically zero. This is an indication of independence between $x_1, ..., x_n, y_1, ..., y_n$, which is the same as the initial assumption and again is an indication that the implemented algorithm is correct. The standard deviation is small, so that the estimated parameters likely are appropriate. Even tho the bias is small for both $\hat{\lambda}_0$ and $\hat{\lambda}_1$ we want to find the true values of λ_0 and λ_1. Thus, the biased corrected estimate is the prefer method.