Computer Intensive Statistical Methods - Exercise 2

Martin Outzen Berild, Sindre Henriksen

March 2019

Problem A

In this exercise we analyze the data in `data3A$x`, which contains a sequence of length $T = 100$ of a non-Gaussian time-series, and compare two different parameter estimators. We consider 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 variable with zero mean 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:

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

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

Let us denote the minimizers by $\hat{\boldsymbol{\beta}}_{LS}$ and $\hat{\boldsymbol{\beta}}_{LA}$ (calculated by `ARp.beta.est`). Further we define the estimated residuals to be $\hat{e}_t = x_t - \hat{\beta}_1 x_{t-1} - \hat{\beta}_2 x_{t-2}$ for $t = 3, \ldots, T$, and let $\bar{e}$ be the mean of these. Finally we define the centered residuals $\hat{\epsilon}_t = \hat{e}_t - \bar{e}$. The centered residuals are obtained using `ARp.resid`.

1. We use the residual resampling bootstrap method to evaluate the relative performance of the two parameter estimators. First we calculate $\hat{\boldsymbol{\beta}}_{LA} \approx$ (1.5466, -0.5575) and $\hat{\boldsymbol{\beta}}_{LS} \approx$ (1.5528, -0.5680) from the provided data set. Next we simulate the time series $B = 2000$ times for each estimator. This is done by randomly choosing to consecutive values x_t and x_{t+1} from the original time series as initial values (x_1^*, x_2^*) and then iteratively finding x_t^*, $t = 3, \ldots, T$ using the parameter estimators and bootstrap samples of the centered residuals. The `ARp.filter` function is used to build the time series, and the rest is done as shown in the code below.

```r
# Load libraries and data
library(ggplot2)
library(tibble)
source("../data/probAhelp.R")
source("../data/probAdata.R")
set.seed(123)

# Least sum of absolute residuals and least squares estimator
beta = ARp.beta.est(data3A$x, 2)
beta_la = beta$LA
beta_ls = beta$LS

# Centered residuals
epsilon_la = ARp.resid(data3A$x, beta_la)
epsilon_ls = ARp.resid(data3A$x, beta_ls)
```

```r
# AR residual resampling method
AR2.rrbootstrap = function(x, beta, e){
  i = sample(1:99, 1)
  x0 = x[i:(i + 1)]
  return(ARp.filter(x0, beta, sample(e, replace=T)))
}

# Bootstrap samples for LA and LS estimators
B = 2000
m = length(data3A$x)
x_hat_la = t(replicate(
  B, AR2.rrbootstrap(data3A$x, beta_la, epsilon_la)))
x_hat_ls = t(replicate(
  B, AR2.rrbootstrap(data3A$x, beta_ls, epsilon_ls)))

# Plot some of the samples and original data
data = tibble(x=data3A$x, t=1:100)
p_x = ggplot(data) + geom_line(aes(t, x, col="Data"))
nums = c(300, 600, 1000)
p_time_series = p_x
for(i in 1:length(nums)){
  p_time_series = p_time_series +
    geom_line(
      data=tibble(x=x_hat_la[nums[i],], t=1:100),
      (aes(t, x, col="LA"))) +
    geom_line(
      data=tibble(x=x_hat_ls[nums[i],], t=1:100),
      (aes(t, x, col="LS"))) +
    labs(col="")
}
```

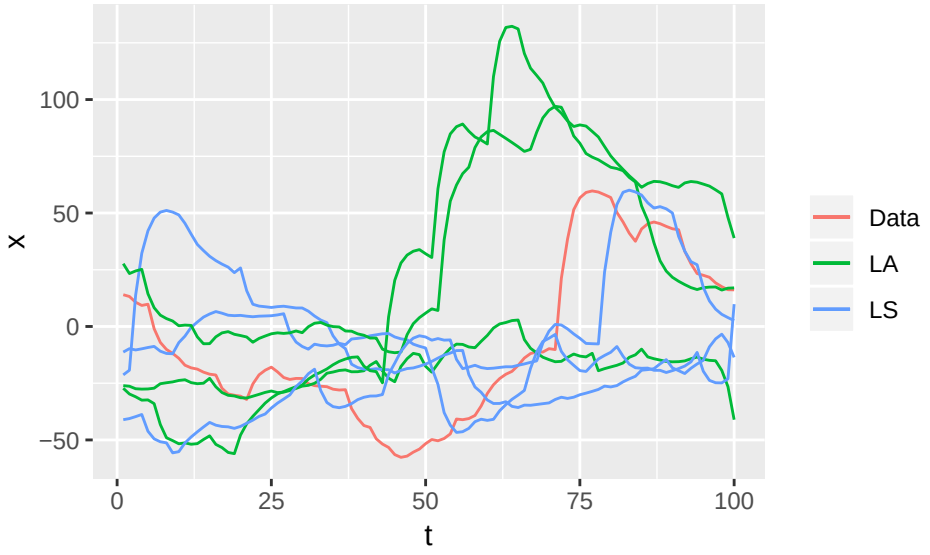

Figure 1: Original time series data and three realizations using parameters estimated by least sum of squared residuals (LS) and parameters estimated by least sum of absolute residuals (LA), and bootstrap samples of the centered residuals.

```
# ggsave("../figures/p1_time_series.pdf", p_time_series,
#        width=5, height=3, units="in")
```

The original time series and three simulated time series using each of the estimators are shown in Figure 1. It is not possible to conclude anything about the quality of the estimators from the plot, but it doesn't give any indications that the estimators are not good.

Now we estimate LA parameters from each of the time series simulated using the LA estimator, and similarly for LS. We calculate the means $\bar{\boldsymbol{\beta}}^*_{LA}$ and $\bar{\boldsymbol{\beta}}^*_{LS}$ and variances of all these and estimated parameters. This gives biases $\bar{\boldsymbol{\beta}}^*_{LA} - \hat{\boldsymbol{\beta}}_{LA} \approx$ (-0.0026, 0.0021), $\bar{\boldsymbol{\beta}}^*_{LS} - \hat{\boldsymbol{\beta}}_{LS} \approx$ (-0.0139, 0.0084), and variances (0.00040, 0.00039) for the LA estimator and (0.00545, 0.00532) for the LS estimator. The biases and variances of the LS estimator are a little larger. However, they are also very small; the biases are $< 1.5\%$ and the variances $< 1\%$ of the original estimator values. Anyways, this is an indication that the LS estimator is not optimal (UMVUE) for this AR process, even though it is for Gaussian AR processes.

2. Next we wish to compute 95% prediction intervals for x_{101} using both the LA and LS estimators. We predict x_{101} using a randomly chosen residual for each of the parameter estimates. I.e. we get 2000 predictions for each of the methods. To get the prediction interval we take the 0.025 and 0.975 quantiles from the predictions.

The resulting intervals are (7.2695, 23.2748) (LA) and (9.4124, 23.2150) (LS).

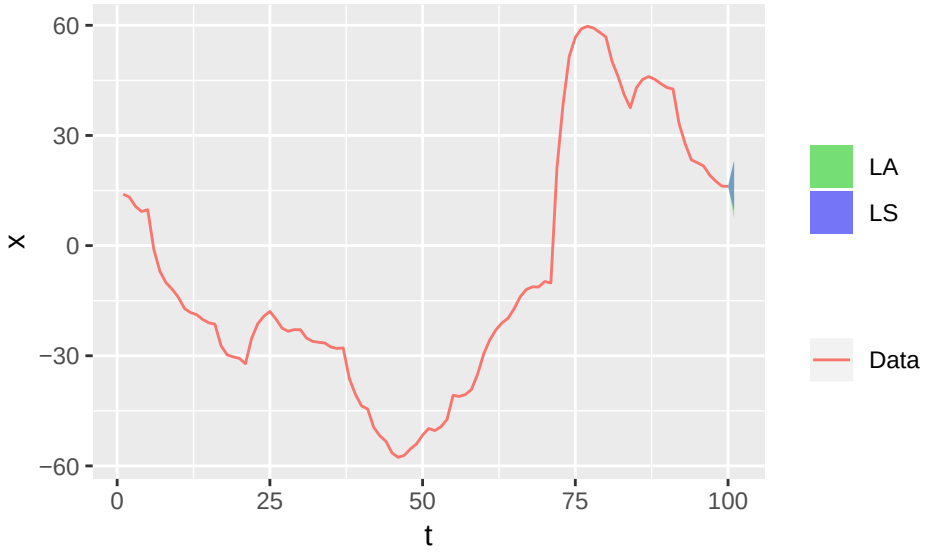

Figure 2: Original time series data and prediction intervals of x_{101} obtained using different LA and LS estimators bootstrap samples of residuals calculated with those estimators.

They are shown in Figure 2. The intervals are so close that they are almost indistinguishable in the plot. This is not surprising, considering the estimated parameters from the methods are very similar, as discussed in 1.

Problem B

1. The data used in this problem is the '*bilirubin.txt*' data set. It contains bilirubin levels in the blood of three different persons. It is imported in the code below.

```
library(tidyverse)
bili.df <-
  as.data.frame(read.table("../data/bilirubin.txt",header=T))
```

A box plot of the *bilirubin* data set is shown in Figure 3. This shows that values from person 'p1' almost have the same mean, but a higher variance, than the values for person 'p2', while the values for person 'p3' are somewhat higher.

```
# boxplot of data
boxplot.bili <-
  ggplot(bili.df, aes(x= pers, y = log(meas),fill=pers)) +
  geom_boxplot() +
  ggtitle("boxplot of bilirubin data")
```

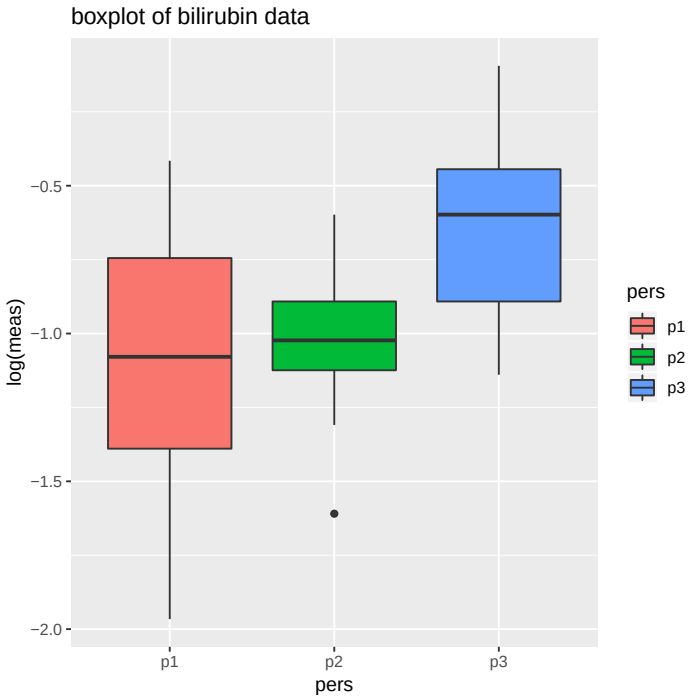

Figure 3: A boxplot of the bilirubin data. On the y-axis the logarithm of the concentration of bilirubin in the blood of three different persons that are grouped on the x-axis.

```
boxplot.bili
```

We perform a linear regression using the function `lm` from **R** to fit the linear model

$$log(\boldsymbol{Y}_{ij}) = \beta_i + \epsilon_{ij};\;\; i = 1, 2, 3,\;\; j = 1, ..., n_i,\;\; \epsilon_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2).$$

We want to do a F-test on the hypothesis $\beta_1 = \beta_2 = \beta_3$, which is done by the hypothesis test

$$\begin{aligned} H_0 :&\quad \beta_1 = \beta_2 = \beta_3 \\ H_1 :&\quad \beta_1 \neq \beta_2 \neq \beta_3. \end{aligned} \tag{1}$$

The code for the linear regression is shown under. We get the F-statistics of the observed data F_{obs} from the `summary.lm` function and we store it in `Fval`. The p-value is $p-\text{value} = 0.039$, so if we choose a significance level of 5% we would reject the null hypothesis. However, this value relies on some assumptions (iid Gaussian residuals), and we wish to make a decision without relying on those.

```
# linear regression of the data
log.y <- lm(log(meas)~pers,data=bili.df)
# getting the summary
sum.log.y <- summary(log.y)
```

```r
# fetching f.statistics
Fval <- as.numeric(sum.log.y$fstatistic[1])
cat("F-statistic:", Fval)
```

```
## F-statistic: 3.669775
```

2. In the `permTest` function defined below a data frame of the *bilirubin* data is the only input. It creates a temporary data frame and then changes the order of the grouping in the `$pers` column, to create a permutation of the data. Then with the same method used in question 1., we calculate and return the F-statistic.

```r
permTest <- function(bili.df){
  # temporary dataframe
  bili.p.df = bili.df
  # permutation on the grouping column
  bili.p.df$pers = bili.df$pers[sample(1:nrow(bili.p.df))]
  # return of summary.lm and fetching the fstat
  return(as.numeric(summary(lm(log(meas)~pers,
                               data = bili.p.df))$fstatistic[1]))
  }
```

3. We are now ready to implement a permutation test using the function created in question 2., with $N = 999$ different permutations. The returned F-statistics $F_i; i \in [1, N]$ can then be used to calculate the p-value of the F-test shown in Equation (1). The p-value is calculated by the equation

$$\text{p-value} = \frac{\sum_{i=1}^{N} I(F_i >= F_{\text{obs}})}{N},$$

where $I(A) = 1$ if A is true and 0 otherwise. A histogram of the F-statistics from the permutations is shown in Figure 4. Here one can see that most of the F-values are to the left of the red line, which is the F-value of the bilirubin data found in question 1. Since most of the points are to the left of the line the p-value is small. Our implementation of the permutation test is shown below, and the p-value is printed.

```r
set.seed(420)
N <- 999
Fvals <- vector()
# running the 999 permutions in permTest
for (i in seq(1,N)){
  Fvals = c(Fvals, permTest(bili.df))
```

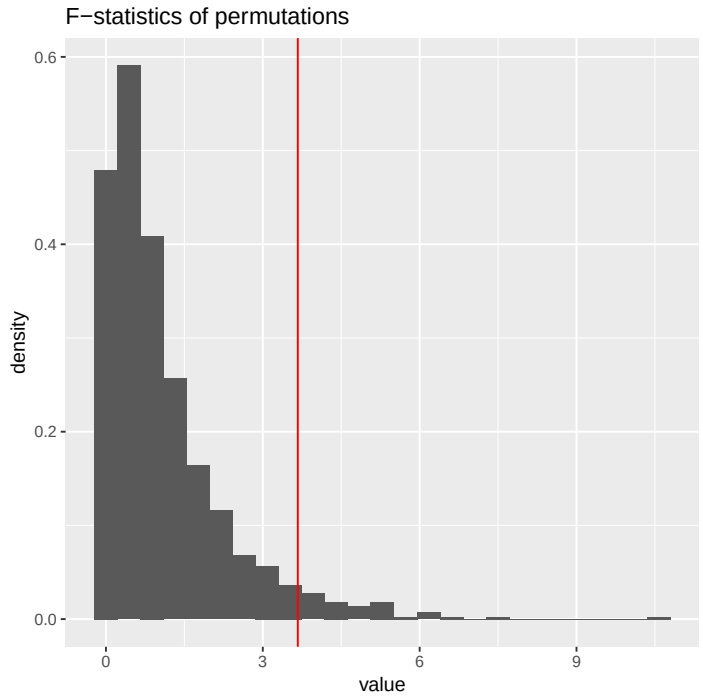

Figure 4: A histogram of the F-statistics from $N = 999$ permutations. The red line is the F-statistic of bilirubin data.

```
}
# dataframe for ggplot histogram
fvals.df <- enframe(Fvals)

# histogram of fstat of permutations
fstat.plot<-ggplot(fvals.df)+
  geom_histogram(aes(x = value, y = ..density..),bins= 25)+
  geom_vline(xintercept = Fval, color = "red")+
  ggtitle("F-statistics of permutations")

fstat.plot

# calculation of p-value
p.F <- mean(Fvals >= Fval)
cat("p-value of F-statistics:", p.F)
```

```
## p-value of F-statistics: 0.04104104
```

The p-value is 0.041. If we chose a significance level of 5% the null hypothesis would be rejected, which indicates that the *bilirubin* levels for the three different persons are from different distributions.

Problem C

Let $\boldsymbol{x} = x_1, \ldots, x_n$ and $\boldsymbol{y} = y_1, \ldots, y_n$ be independent random variables, where the x_i's have an exponential distribution with intensity λ_0 and the y_i's have an exponential distribution with intensity λ_1. Assume we do not observe $\boldsymbol{x}$ and $\boldsymbol{y}$ directly, but that we observe

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

and

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

where $\mathbb{1}(A)$ is 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 maximum likelihood estimates for (λ_0, λ_1).

1. Since z_i and u_i are determined by x_i and y_i, our complete data is simply $\boldsymbol{x}$ and $\boldsymbol{y}$. The likelihood of the complete data is thus

$$f(\boldsymbol{x}, \boldsymbol{y} \,|\, \lambda_0, \lambda_1) = \lambda_0^n \lambda_1^n \exp\left\{ -\lambda_0 \sum_{i=1}^n x_i - \lambda_1 \sum_{i=1}^n y_i \right\},$$

where we have used that the variables are independent and that the PDF of an exponential random variable w with rate λ is $\lambda e^{-\lambda w}$. This gives the log likelihood

$$\ln f(\boldsymbol{x}, \boldsymbol{y} \,|\, \lambda_0, \lambda_1) = n \ln \lambda_0 + n \ln \lambda_1 - \lambda_0 \sum_{i=1}^n x_i - \lambda_1 \sum_{i=1}^n y_i\,.$$

The expectation of the log likelihood is

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

Now

$$\begin{aligned}E[x_i \,|\, z_i, u_i, \lambda_0^{(t)}] &= u_i z_i + (1 - u_i)\mathbf{E}[x_i \,|\, x_i < z_i, \lambda_0^{(t)}] \\ &= u_i z_i + (1 - u_i) \int_0^{z_i} x_i \frac{f(x_i)}{F(z_i)}\,\mathrm{d}x_i \\ &= u_i z_i + (1 - u_i) \frac{1}{1 - e^{-\lambda_0^{(t)} z_i}} \int_0^{z_i} x_i \lambda_0^{(t)} e^{-\lambda_0^{(t)} x_i}\,\mathrm{d}x_i \\ &= u_i z_i + (1 - u_i) \frac{1}{1 - e^{-\lambda_0^{(t)} z_i}} \left[-z_i e^{-\lambda_0^{(t)} z_i} - \frac{e^{-\lambda_0^{(t)} z_i}}{\lambda_0^{(t)}} + \frac{1}{\lambda_0^{(t)}} \right] \\ &= u_i z_i + (1 - u_i) \left[\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)} z_i} - 1} \right]\end{aligned}$$

and, similarly,

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

Thus

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

2. To maximize $Q(\lambda_0, \lambda_1) = \mathbf{E}[\ln f(\boldsymbol{x}, \boldsymbol{y} \,|\, \lambda_0, \lambda_1) \,|\, \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}]$ we need

$$\frac{\partial Q}{\partial \lambda_0} = \frac{n}{\lambda_0} - \sum_{i=1}^{n} E[x_i \,|\, z_i, u_i, \lambda_0^{(t)}] = 0 \quad \Rightarrow \lambda_0 = n \left[\sum_{i=1}^{n} E[x_i \,|\, z_i, u_i, \lambda_0^{(t)}] \right]^{-1},$$

$$\frac{\partial Q}{\partial \lambda_1} = \frac{n}{\lambda_1} - \sum_{i=1}^{n} E[y_i \,|\, z_i, u_i, \lambda_1^{(t)}] = 0 \quad \Rightarrow \lambda_1 = n \left[\sum_{i=1}^{n} E[y_i \,|\, z_i, u_i, \lambda_1^{(t)}] \right]^{-1}.$$

The EM algorithm maximizes $Q(\boldsymbol{\lambda} \,|\, \boldsymbol{\lambda}^{(t)})$, where $\boldsymbol{\lambda} = (\lambda_0, \lambda_1)$ iteratively, setting $\boldsymbol{\lambda}^{(t)} = \text{argmax}_{\boldsymbol{\lambda}} Q(\boldsymbol{\lambda} \,|\, \boldsymbol{\lambda}^{(t-1)})$. This is repeated until some criterion is satisfied. We choose to say that convergence is obtained when the l_2-norm of the difference $\boldsymbol{\lambda}^{(t)} - \boldsymbol{\lambda}^{(t-1)}$ is smaller than some user-defined tolerance. The algorithm is implemented in the following code.

```
# Load libraries and data
library(ggplot2)
library(tibble)
set.seed(123)
data <- data.frame(
  z = read.table("../data/z.txt",header=F, col.names = "z"),
  u = read.table("../data/u.txt",header=F, col.names = "u")
)
```

```
# M-step function (returns argmax of complete likelihood)
argmax_Q = function(z, u, lambda) {
  n = length(z)
  stopifnot(length(u) == n)
  stopifnot(length(lambda) == 2)
```

```r
  lambda_0 = n * sum(
    u * z +
    (1 - u) * (1 / lambda[1] - z / (exp(lambda[1] * z) - 1))
  )^(-1)
  lambda_1 = n * sum(
    (1 - u) * z +
    u * (1 / lambda[2] - z / (exp(lambda[2] * z) - 1))
  )^(-1)

  return(c(lambda_0, lambda_1))
}

# Expecation maximation algorithm
em = function(z, u, lambda_initial, tol) {
  step = Inf
  lambda = lambda_initial
  i = 0
  steps = numeric()
  lambda_steps = list(lambda_initial)
  while(step > tol) {
    i = i + 1
    temp = lambda
    lambda = argmax_Q(z, u, lambda)
    step = sqrt(sum((lambda - temp)^2))
    steps[i] = step
    lambda_steps[[i + 1]] = lambda
  }
  print(paste0("Converged after ", i, " iterations"))
  return(
    list(lambda=lambda, lambda_steps=lambda_steps, steps=steps))
}

# Estimate lambda
lambda_init = c(1, 1)
res = em(data$z, data$u, lambda_init, 1e-2)
lambda_em = res$lambda

# Plot convergence
steps = res$steps
steps_df = tibble(iteration=(1:length(steps)), step=steps)
p_convergence = ggplot(steps_df, aes(x=iteration, y=step)) +
  geom_point() +
  labs(y="l2-norm of step size")
# ggsave("../figures/p3_convergence.pdf", p_convergence,
       # width=4, height=3, units="in")
```

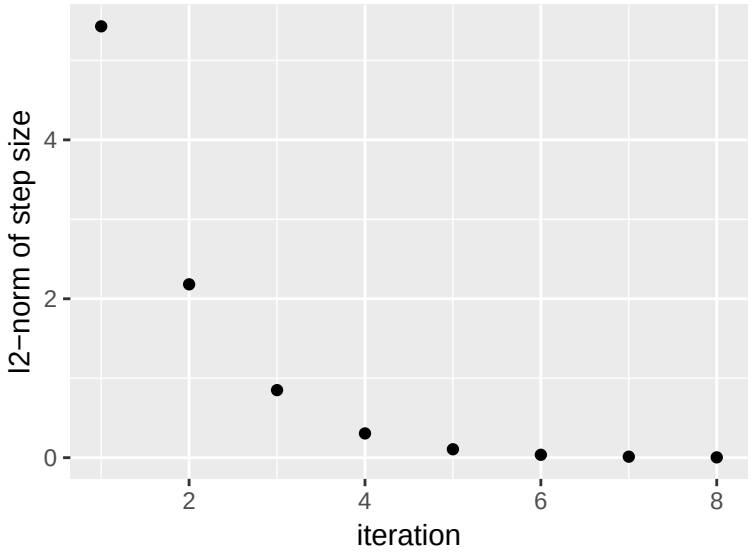

Figure 5: Convergence of EM algorithm for the data in Problem C. The algorithm stops when the l_2-norm of the step-size $\boldsymbol{\lambda}^{(t)} - \boldsymbol{\lambda}^{(t-1)} < 0.01$., i.e. after 8 iterations.

```
lambda_steps = do.call("rbind", res$lambda_steps)
lambda_steps_df = tibble(iteration=0:length(steps),
                         lambda0=lambda_steps[,1],
                         lambda1=lambda_steps[,2])
p_lambda = ggplot(lambda_steps_df) +
  geom_point(aes(x=iteration, y=lambda0, col="lambda0"),
             alpha=0.7) +
  geom_point(aes(x=iteration, y=lambda1, col="lambda1"),
             alpha=0.7) +
  labs(y="Lambda values", col="")
# ggsave("../figures/p3_lambda.pdf", p_lambda,
       # width=5, height=3, units="in")
```

The EM algorithm is run on the data specified in the files `z.txt` and `u.txt`. Setting the tolerance to 0.01, the algorithm converges after 8 iterations, giving the estimate $\boldsymbol{\lambda} = (3.4657, 9.3510)$. The l_2-norms of the step-sizes $\boldsymbol{\lambda}^{(t)} - \boldsymbol{\lambda}^{(t-1)}$ are shown in Figure 5. The values of $\boldsymbol{\lambda}$ after each iteration are shown in Figure 6. We see that λ_0 converges faster than λ_1. The same happens also with different starting values. This might be due to 142 of the 200 data points in `z.txt` being from $\boldsymbol{x}$ (which can be seen from the number of ones in `u.txt`).

3. Similarly to what we did in problem A, we now wish to use bootstrapping to find the standard deviations and the biases of each of $\hat{\lambda}_1$ and $\hat{\lambda}_1$. The algorithm is like follows: Take bootstrap samples of the observation pairs and run the

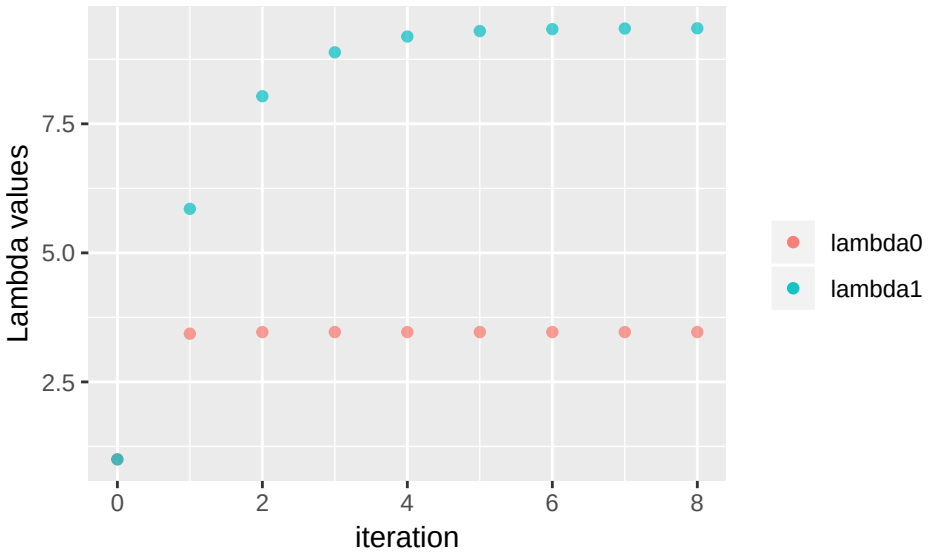

Figure 6: Values of λ_0 and λ_1 after each iteration of the EM algorithm discussed in Problem C.

EM-algorithm on the bootstrap samples, repeat B times. Then calculate the standard deviation of the bootstrap estimators and the bias, i.e. the mean of the bootstrap samples minus the original estimator. We also find the correlation of the estimated parameters.

```
library(gridExtra)
set.seed(56)
n <- length(data$u)
lambda_init = c(1,1)
B <- 1500
theta.hat.boot <- matrix(NA,nrow = B, ncol = 2)
for (i in seq(1,B)){
  index = sample(1:n,n,replace = T)
  theta.hat.boot[i,] = em(
    data$z[index], data$u[index], lambda_init, 1e-2)$lambda
}

lambda.df <- data.frame(
  lambda0 = theta.hat.boot[,1],lambda1 = theta.hat.boot[,2])
load(file="../data/p3.Rdata")

sd.lambda0 <- sd(lambda.df$lambda0)
sd.lambda1 <- sd(lambda.df$lambda1)

bias.lambda0 <- mean(lambda.df$lambda0) - lambda[1]
bias.lambda1 <- mean(lambda.df$lambda1) - lambda[2]
```

```r
cor.lambda <- cor(lambda.df$lambda0, lambda.df$lambda1)

# Calculating the bias correction
lambda0c = lambda[1]- bias.lambda0
lambda1c = lambda[2] - bias.lambda1

hist.lambda0 <- ggplot(lambda.df) +
  geom_histogram(aes(x = lambda0, y = ..density..),bins = 30)+
  geom_vline(xintercept = lambda[1], color = "firebrick",size=1)

hist.lambda1 <- ggplot(lambda.df) +
  geom_histogram(aes(x = lambda1, y = ..density..),bins = 30)+
  geom_vline(xintercept = lambda[2], color = "firebrick",size=1)

hist.lambda <- grid.arrange(hist.lambda0,hist.lambda1)

cat("Standard diviation lambda_0:", sprintf("%.2f",sd.lambda0))
cat("Standard diviation lambda_1:", sprintf("%.2f",sd.lambda1))
cat("Bias of lambda_0:", sprintf("%.2f",bias.lambda0))
cat("Bias of lambda_1:", sprintf("%.2f",bias.lambda1))
cat("Correlation between lambda_0 and lambda_1:",
    sprintf("%.2f",cor.lambda))
cat("Bias corrected lambda_0:", sprintf("%.2f",lambda0c))
cat("Bias corrected lambda_1:", sprintf("%.2f",lambda1c))
```

```
## Standard deviation lambda_0: 0.25
## Standard deviation lambda_1: 0.79
## Bias of lambda_0: 0.02
## Bias of lambda_1: 0.12
## Correlation between lambda_0 and lambda_1: 0.02
## Bias corrected lambda_0: 3.45
## Bias corrected lambda_1: 9.23
```

Figure 7 contains a histogram of the bootstrap estimators. The standard deviations are 0.25 and 0.79 and the biases are 0.02 and 0.12, for $\hat{\lambda}_0$ and $\hat{\lambda}_1$, respectively. The correlation is 0.02. The biases are not very large; $\approx 1\%$ of the size of the original estimators. Thus we are tempted not to use the bias corrected estimators, since they have a larger variance than the original estimators. We could use just one bias corrected estimator and the original estimator for the other component since they seem to be almost uncorrelated. But the biases are both similarly small, so in this case they are probably better left uncorrected.

4. We now wish to find an analytical formula for $f_{Z_i,U_i}(z_i, u_i \mid \lambda_0, \lambda_1)$. Throughout the following derivation, we assumes λ_0 and λ_1 are known without stating

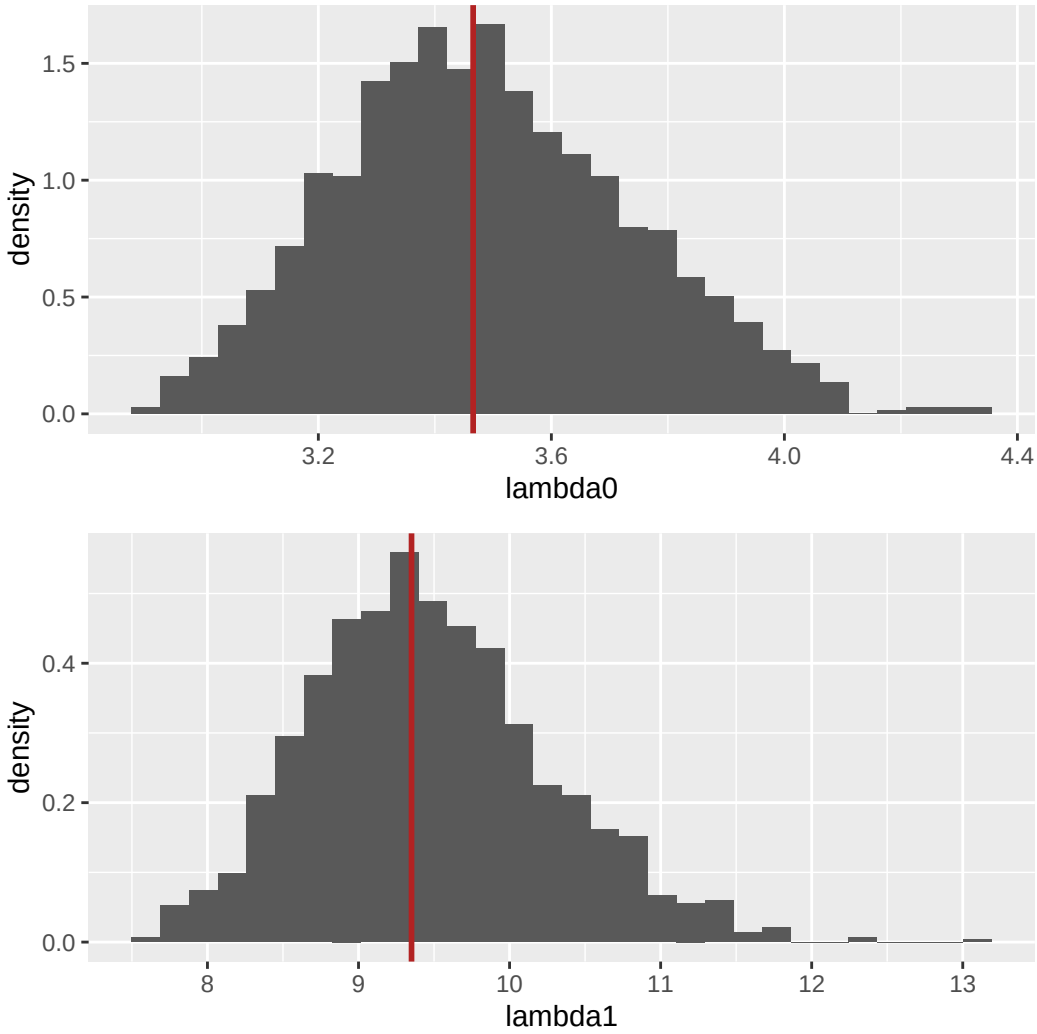

Figure 7: Histogram of frequency of bootstrap estimators of (λ_0, λ_1), found as explained in Problem C3.

it explicitly. We start by finding

$$\mathrm{p}(Z_i \le z_i, u_i) = u_i \mathrm{p}(X_i \le z_i, X_i > Y_i) + (1-u_i)\mathrm{p}(Y_i \le z_i, Y_i > X_i)\,.$$

Now

$$\begin{aligned}\mathrm{p}(X_i \le z_i, X_i > Y_i) &= \int_0^{z_i}\int_y^{z_i} f_Y(y) f_X(x)\,\mathrm{d}x\,\mathrm{d}y \\ &= 1 - e^{-\lambda_0 z_i} + \frac{\lambda_0}{\lambda_0+\lambda_1}(1 - e^{-(\lambda_0+\lambda_1)z_i})\,.\end{aligned}$$

Similarly,

$$\mathrm{p}(Y_i \le z_i, Y_i > X_i) = 1 - e^{-\lambda_1 z_i} + \frac{\lambda_1}{\lambda_0+\lambda_1}(1 - e^{-(\lambda_0+\lambda_1)z_i})\,.$$

Next we find

$$\begin{aligned}f_{Z_i,U_i}(z_i,u_i) &= \tfrac{\partial}{\partial z}\mathrm{p}(Z_i \le z_i, u_i) \\ &= u_i\left[\lambda_0 e^{-\lambda_0 z_i}(1-e^{-\lambda_1 z_i})\right] + (1-u_i)\left[\lambda_1 e^{-\lambda_1 z_i}(1-e^{-\lambda_0 z_i})\right] \\ &= u_i g(z_i) + (1-u_i)h(z_i)\,,\end{aligned}$$

where

$$\begin{aligned}g(z_i) &= \lambda_0 e^{-\lambda_0 z_i}(1-e^{-\lambda_1 z_i}) \\ h(z_i) &= \lambda_1 e^{-\lambda_1 z_i}(1-e^{-\lambda_0 z_i})\,.\end{aligned}$$

Next we wish to find the maximum likelihood estimate of λ_0 and λ_1. First we need the likelihood

$$f_{\boldsymbol{Z},\boldsymbol{U}}(\boldsymbol{z},\boldsymbol{u}) = \prod_{i\,:\,u_i=1} g(z_i) \prod_{i\,:\,u_i=0} h(z_i)\,.$$

The log likelihood is

$$\begin{aligned}\log f_{\boldsymbol{Z},\boldsymbol{U}}(\boldsymbol{Z},\boldsymbol{U}) &= \sum_{i\,:\,u_i=1}\log g(z_i) + \sum_{i\,:\,u_i=0}\log h(z_i) \\ &= \sum_{i=1}^{n} u_i \log g(z_i) + (1-u_i)\log h(z_i)\,.\end{aligned}$$

Now

$$\begin{aligned}\log g(z_i) &= \log\lambda_0 - \lambda_0 z_i + \log(1-e^{-\lambda_1 z_i})\,, \quad \text{and} \\ \log h(z_i) &= \log\lambda_1 - \lambda_1 z_i + \log(1-e^{-\lambda_0 z_i})\,,\end{aligned}$$

so

$$\begin{aligned}&\log f_{\boldsymbol{Z},\boldsymbol{U}}(\boldsymbol{Z},\boldsymbol{U}) = k\log\lambda_0 + (n-k)\log\lambda_1 \\ &+ \sum_{i=1}^{n}\left\{u_i\left[\log(1-e^{-\lambda_1 z_i}) - \lambda_0 z_i\right] + (1-u_i)\left[\log(1-e^{-\lambda_0 z_i}) - \lambda_1 z_i\right]\right\}\,, \quad (2)\end{aligned}$$

where $k = \sum_{i=1}^{n} u_i$. To find the maximum we take the derivative with respect to λ_0 and λ_1;

$$\begin{aligned} \frac{\partial}{\partial\lambda_0} \log f &= \frac{k}{\lambda_0} + \sum_{i=1}^{n} \left[(1-u_i) \frac{z_i}{e^{\lambda_0 z_i} - 1} - u_i z_i \right], \quad \text{and} \\ \frac{\partial}{\partial\lambda_1} \log f &= \frac{n-k}{\lambda_1} + \sum_{i=1}^{n} \left[u_i \frac{z_i}{e^{\lambda_1 z_i} - 1} - (1-u_i) z_i \right], \end{aligned} \tag{3}$$

and we let this equal 0. Even without the sum this equation is not easy to solve analytically, and the sum seems to make it impossible. So we find the MLE numerically. This can be done by finding λ_0 and λ_1 maximizing the log-likelihood (2) directly, or by finding the roots of (3). We do the latter, since it is somewhat simpler to implement in this case.

To be sure that we find a global maximum and not a local one, we prove that the Hessian is negative definite. First we notice

$$\frac{\partial^2}{\partial\lambda_0\partial\lambda_1} \log f = \frac{\partial^2}{\partial\lambda_1\partial\lambda_0} \log f = 0\,.$$

So the off-diagonal elements of the Hessian are 0. Thus the diagonal elements are the eigenvalues, and if they are negative the Hessian is negative definite. We get

$$\begin{aligned} \frac{\partial^2}{\partial\lambda_0^2} \log f &= -\frac{k}{\lambda_0^2} - \sum_{i=1}^{n} (1-u_i) \frac{z_i^2 e^{\lambda_0 z_i}}{(e^{\lambda_0 z_i - 1})^2} < 0\,, \quad \text{and} \\ \frac{\partial^{2\,2}}{\partial\lambda_1^2} \log f &= -\frac{n-k}{\lambda_1^2} - \sum_{i=1}^{n} u_i \frac{z_i^2 e^{\lambda_1 z_i}}{(e^{\lambda_1 z_i - 1})^2} < 0\,. \end{aligned}$$

So we have proven that if we find a maximum it is the global maximum. The maximum likelihood estimators are found numerically in the following code.

```
# Derivatitve of ln f(z,u) as a function of lambda
# (lambda0/lambda1)
dlnf_dlambda = function(lambda, u, z) {
  return(
    sum(u) / lambda + sum(
      (1 - u) * z / (exp(lambda * z) - 1) - u * z
    )
  )
}

# Find the roots
uni0 = uniroot(
  function(x){dlnf_dlambda(x, data$u, data$z)}, c(0.001, 100))
uni1 = uniroot(
  function(x){dlnf_dlambda(x, (1-data$u), data$z)}, c(0.001, 100))
lambda_mle = c(uni0$root, uni1$root)
```

The result is $\boldsymbol{\lambda}_{MLE} = (3.4657, 9.3532)$. This is very close to the result from the expectation maximization algorithm $\boldsymbol{\lambda}_{EM} = (3.4657, 9.3510)$.

One advantage of maximizing the likelihood directly is that we are sure that we actually maximize the right thing; we maximize the likelihood and not the complete likelihood as in EM, where we also have to estimate values for $\boldsymbol{x}$ and $\boldsymbol{y}$. Another advantage is that we only have to maximize once, while with EM we have to maximize once for each iteration. Which in general can be computationally expensive. Furthermore, now that we have the Hessian, we can use it to find the amount of information in the data (if we can find the expectation of the second order partial derivatives).

In some cases, however, the numerical calculations to solve the equation(s) corresponding to setting (3) to 0 are computationally expensive, and less robust than the EM algorithm. And in other cases it might not even be possible to find the likelihood for the observed data; integrating out the hidden variables might not be possible analytically.