Project 3 - TMA4300 Computer Intensive Statistical Methods

Jostein Aastebøl Aanes, Frederick Nilsen

29.04.2022

Resampling residuals of AR(2) parameters

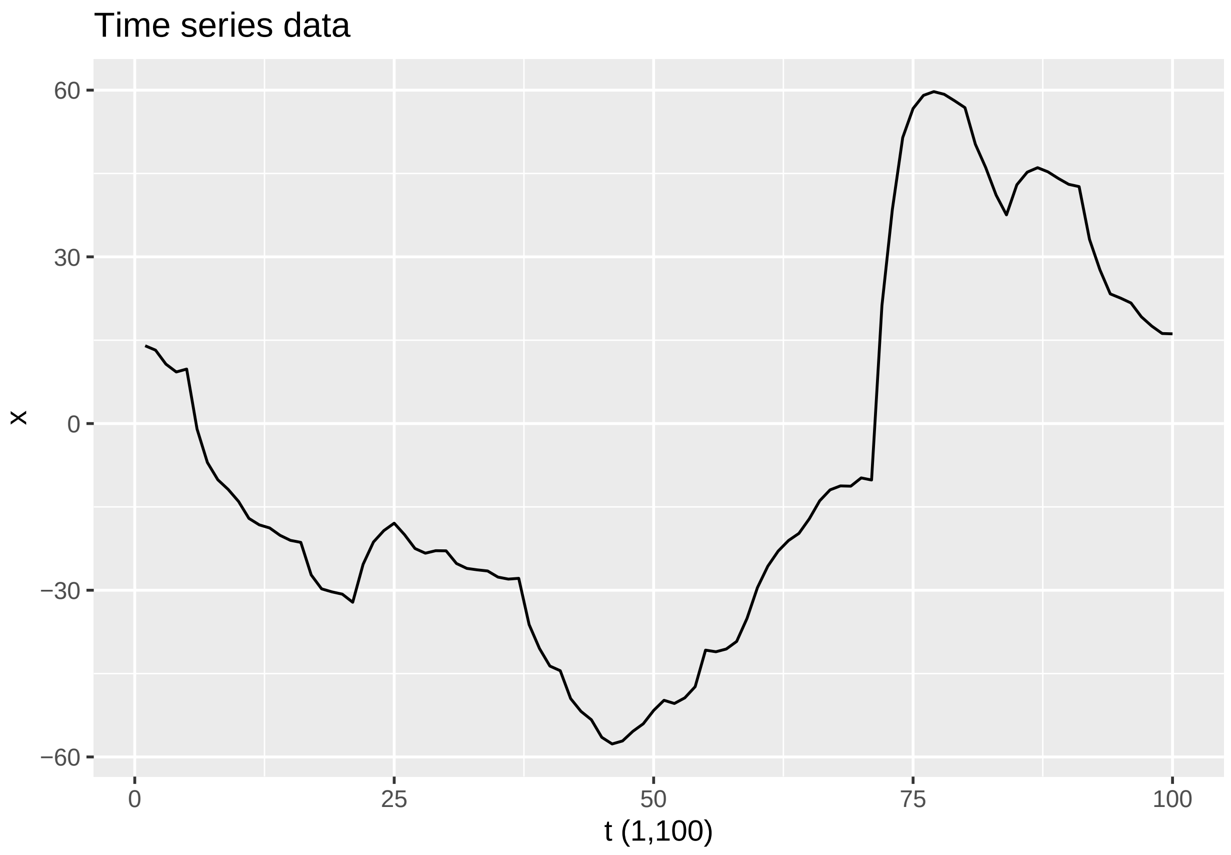

Figure 1: Plot of time series data over each time step, $t \in [0, 100]$.

For this problem, we will consider a non-Gaussian time series shown in Figure 1, and compare two different parameter estimators. We will consider an autoregressive model of order 2, $AR(2)$, defined by $x_t = \beta_1 x_{t-1} + \beta_2 x_{t-2} + e_t$ where e_t are iid random variables with mean zero and constant variance. We will compare two different estimators for $\boldsymbol{\beta} = (\beta_1, \beta_2)$. The first estimator is the least squares estimator, denoted as $\widehat{\boldsymbol{\beta}}_{LS}$, and is defined by minimizing the following loss function with respect to $\boldsymbol{\beta}$

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

The second estimator is the least absolute sum of residuals estimator, denoted $\widehat{\boldsymbol{\beta}}_{LA}$, and is defined by minimizing the following loss function with respect to $\boldsymbol{\beta}$

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

Performance of model estimators with bootstrapping

We consider the least squares and least absolute residual minimizers, $\widehat{\boldsymbol{\beta}}_{LS}$ and $\widehat{\boldsymbol{\beta}}_{LA}$ respectively. In particular, the residual resampling bootstrap method can be used to determine an estimate of the estimators' bias and variances.

This is done by first fitting the data to an AR(2) model of T time steps. Thereafter, we compute estimated residuals $\hat{e}_t = x_t - \hat{\beta}_1 x_{t-1} - \hat{\beta}_2 x_{t-2}, t = 3, \dots, T$, denoting $\bar{e}$ as the mean so that the zero-mean residual is $\hat{\varepsilon}_t = \hat{e}_t - \bar{e}$.

We sample from the zero-mean residuals (with replacement) iteratively B times, randomly select a subsequence of observed values, x_1^*, x_2^* and fit to a AR(2) model again to obtain $\widehat{\boldsymbol{\beta}^*_{LS}}$ and $\widehat{\boldsymbol{\beta}^*_{LA}}$. For sufficiently large B, the variance of all $\widehat{\boldsymbol{\beta}^*_{..}}$ gives an estimate of the variance of $\widehat{\boldsymbol{\beta}_{..}}$.

The code for the bootstrapping is implemented below.

```r
bootstrap_AR2 <- function(method, B){
  allowed.methods = c("LS", "LA")
  stopifnot(method %in% allowed.methods)

  # Fit to AR(2)
  betas.list = ARp.beta.est(data3A$x, p=2)
  beta.vec =  get(method, betas.list)
  init.res = ARp.resid(data3A$x, beta.vec)

  n = length(init.res) # 98
  results = array(data=NA, dim=c(B, 2))

  for(b in 1:B){
    # Sample eps.1*, ..., eps.n* at random with replacement
    res.samples = sample(init.res, n, replace=T)

    # Choose x1 (and x2) at random
    x0 = sample.int(100 - 1, size=1)
    x = ARp.filter(data3A$x[x0 : (x0+1)], beta.vec, res.samples)

    # Add bootstrapped betas
    results[b, ] = get(method, ARp.beta.est(x, p=2))

  }
  return(results)
}

boot.result.LS = bootstrap_AR2("LS", 1500)
boot.result.LA = bootstrap_AR2("LA", 1500)
```

From the resulting array, the estimated variance for both $\widehat{\boldsymbol{\beta}}_{LS}$ and $\widehat{\boldsymbol{\beta}}_{LA}$ are available by the empirical variance of bootstrap samples. The estimated bias can be computed by using the plug-in principle, which means the estimated bias is computed by taking the difference between bootstrap mean and initial estimate.

```r
beta.var.LS = c(var(boot.result.LS[, 1]), var(boot.result.LS[, 2]))
beta.var.LA = c(var(boot.result.LA[, 1]), var(boot.result.LA[, 2]))

beta.bias.LS = colMeans(boot.result.LS) - ARp.beta.est(data3A$x, p=2)$LS
beta.bias.LA = colMeans(boot.result.LA) - ARp.beta.est(data3A$x, p=2)$LA
```

We see that $\text{Var}(\widehat{\boldsymbol{\beta}}_{LS}) = [0.0054, 0.0053]^T$, $\text{Var}(\widehat{\boldsymbol{\beta}}_{LA}) = [4\times10^{-4}, 4\times10^{-4}]^T$, $\widehat{\text{Bias}}_B(\widehat{\boldsymbol{\beta}}_{LS}) = [-0.0167, 0.0106]^T$, and $\widehat{\text{Bias}}_B(\widehat{\boldsymbol{\beta}}_{LA}) = [-0.0018, 0.0013]^T$. We see that $\widehat{\boldsymbol{\beta}}_{LS}$ har larger bias and considerably larger variance than $\widehat{\boldsymbol{\beta}}_{LA}$. Hence, $\widehat{\boldsymbol{\beta}}_{LS}$ is not optimal in this problem. It is true that $\widehat{\boldsymbol{\beta}}_{LS}$ is optimal for a Gaussian $AR(p)$ process, but as stated earlier, the time series is non-Gaussian. Therefore $\widehat{\boldsymbol{\beta}}_{LS}$ is not optimal for this problem.

Prediction interval for forecasted time series

We want to get a 95% prediction interval for x_{101}. This is done by using the bootstrapped time series to get an estimate of the residual distribution and use this to simulate x_{101} based on the observed data.

According to the formula for an AR(2) process, x_{101} should be on the form $\beta_1 x_{100} + \beta_2 x_{99} + \varepsilon_{101}$. The points x_{100} and x_{99} are known. Furthermore, β_1 and β_2 are decided by *sampling with replacement* from the bootstrapped time series, and ϵ_{101} is sampled from the corresponding estimated residual distribution. Thus, the variability of our prediction reflects our lack of knowledge about the parameters and the residual distribution.

```r
get_next_x_preds <- function(method, result, B=1500){
  # Method to get B=1500 predictions of x_101
  # method: c("LA", "LS")
  # result: resulting list array from bootstrap_AR2
  #      B: number of bootstraps, defaults to 1500
  # return: B=1500 long list of samples of x_101

  stopifnot(method %in% c("LS", "LA"))

  # Get residuals
  init.res = ARp.resid(data3A$x, get(method, ARp.beta.est(data3A$x, p=2)))

  #Sample epsilon101
  epsilon101 <- sample(init.res, size=B, replace=T)

  # beta_1 * x100 + beta_2 * x99 + epsilon
  return(list(predicted_x101 = result[sample(B, replace=T), 1]
              * data3A$x[100] +
                result[sample(B, replace=T), 2] * data3A$x[99] +
                epsilon101, epsilon101 = epsilon101)
  )
}

result_pred.LS <- get_next_x_preds("LS", boot.result.LS)
result_pred.LA <- get_next_x_preds("LA", boot.result.LA)

x101.LS <- result_pred.LS$predicted_x101
x101.LA <- result_pred.LA$predicted_x101

epsilon.LS <- result_pred.LS$epsilon101
epsilon.LA <- result_pred.LA$epsilon101
```

```
x101.LS.predint = quantile(x101.LS, c(0.025, 0.975))
x101.LA.predint = quantile(x101.LA, c(0.025, 0.975))
```

From the code above, we get a prediction interval for LS method between $[8.01, 22.651]$ and for LA in the interval $[7.54, 21.224]$. The LA method seem to be a little smaller in (numerical) value. Although the lengths of the prediction intervals are relatively similar, the interval for LA is a bit smaller, indicating a more precise prediction than the latter. Histograms of the predicted values for x_{101} can be seen in Figure 2. In the plots we see that there does exist some predictions of x_{101} for both estimators around the value of 50. These values are obviously outside the prediction interval. This means that there are some abnormally large estimated residuals compared to the other estimates.

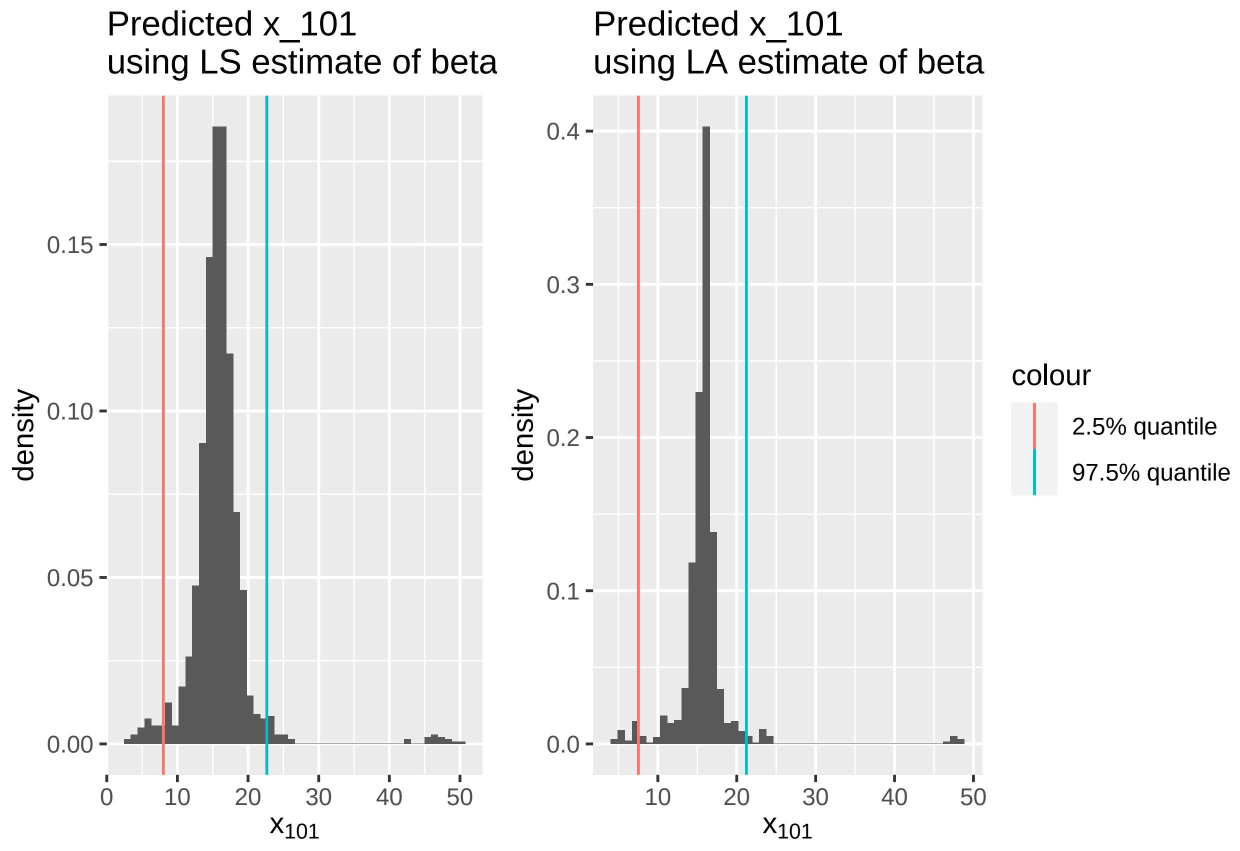

Figure 2: Histograms of predicted x101 using LS- and LA-estimator

Permutation test

For this problem, we wish to inspect the measures of amount of bilirubin in the blood of three men. The file `bilirubin.txt` contains 29 measurements, 11 from person 1, 10 from person 2, and 8 from person 3. Let, Y_{ij} be measurment number j for person i. we will a fit a regression model of the form

$$\ln(Y_{ij}) = \beta_i + \epsilon_{ij},$$

where it is assumed that $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$ i.i.d.

From this, we wish to test whether or not $\beta_1 = \beta_2 = \beta_3$. We say that our null-hypothesis $H_0 : \beta_1 = \beta_2 = \beta_3$ and our alternative hypothesis is $H_A : (\beta_1 \neq \beta_2) \vee (\beta_1 \neq \beta_3) \vee (\beta_2 \neq \beta_3)$. We will be using a significance level of 0.05.

```r
#Read in file
bilirubin <- read.table("bilirubin.txt",header=T)
```

Fitting the linear model

To begin, we create boxplots of log of meas for each of the three persons, to see if the mean appears similar.

```r
# Boxplot of log(meas) for each of the three people
ggplot(bilirubin, aes(x = pers, y = log(meas))) + geom_boxplot()
```

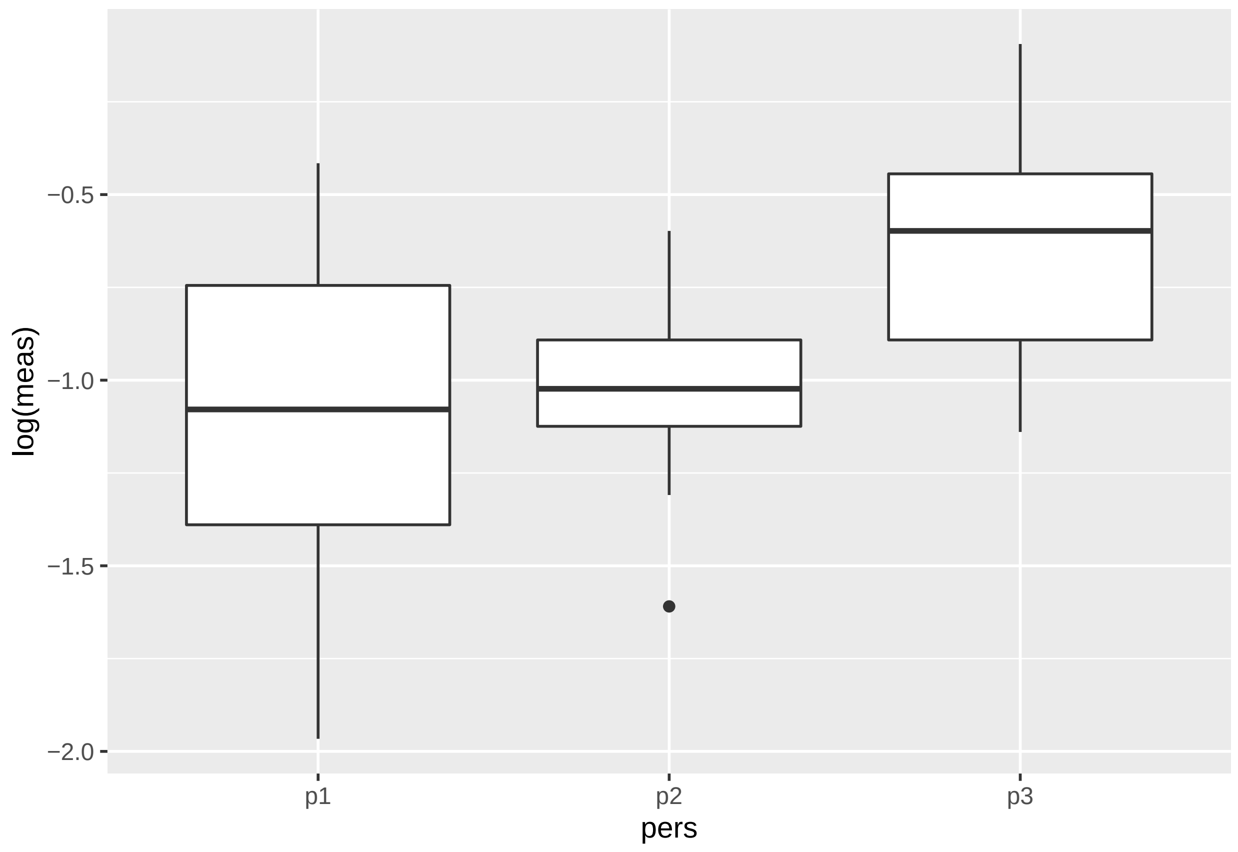

Figure 3: Box plot of log of meas for the three people in the study.

From Figure 3 it is quite clear that the mean of log of meas for person 1 and 2 is quite similar. But the mean of log of meas for person 3 is outside of the 75th quantile for both person 1 and 2, and thus we can expect this to be different. Therefore, we expect the result of our hypothesis test to reflect this, and to tell us to reject H_0.

We now fit the regression model and examine the summary output.

```r
#Fit a linear regression model of form log(Y_ij) = beta_i+epsilon_ij
linearRegressionModel <- lm(log(meas) ~ pers, data = bilirubin)

#Now we want to find our test statistic through summary.lm function
summarylm <- summary.lm(linearRegressionModel)
fval <- summarylm$fstatistic
```

The summary of the regression model is shown in Table 1. The intercept is the value of β_1, `persp2` captures

Table 1: R-summary of linear model fit

	Model 1
(Intercept)	-1.09^{***}
	(0.12)
persp2	0.06
	(0.17)
persp3	0.46^{*}
	(0.18)
R^2	0.22
Adj. R^2	0.16
Num. obs.	29
F statistic	3.67

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

the difference between the intercept and β_2, and similarly, `persp3` captures the difference between the intercept and β_3.

From Table 1, we get as expected that `persp2` is quite small as β_2 is close to the intercept. In addition we also get a relatively large `persp3` compared to `persp2` which is expected from Figure 3, where we see that the mean of log(meas) is larger for person three than person two. When looking at the value of our F-statistic it is 3.67 with corresponding p-value 0.039, which is smaller than 0.05, and thus we reject our null-hypothesis.

Implementing the permutation test

Below a function to carry out a permutation test on the `bilirubin.txt` dataset is implemented. The permutation test generates a permutation of the data between the different individuals. Then, a linear model of the same form is fitted to the permutated dataset. Under the assumption of H_0, the original F-statistic 3.67, should not be extreme compared to the permutated F-statistics, since $\beta_1 = \beta_2 = \beta_3$.

```r
permutation_Test <- function(df, original_F_stat = fval["value"], n){
  # Method for developing permutation test
  #
  # df               : dataframe consisting of response values and labels
  # original_F_stat : the F-statistic calculated originally
  # n                : number of permutations to be made
  # Returns          : List with p-value for F-statistic and all F-statistics

  test_stats = rep(0, n) # Object to store the test-stats for each permutation
  df_copy = data.frame(df)

  for (i in 1:n) {
    # Random shuffling
    df_copy$pers <- sample(df_copy$pers)

    # Create lm-model as the original model
    df_copy.lm <- lm(log(meas) ~ pers, data = df_copy)

    # Calculate F-stat
    test_stats[i] = summary.lm(df_copy.lm)$fstatistic["value"]

  }
  # Count number of times permutation F-stat> original F-stat and divide by n
  p_value <- sum(ifelse(rep(original_F_stat, n) < test_stats, 1, 0))/n
  return(list(p_value = p_value, F_values = test_stats))
}
```

With the function `permutation_Test()`, we can generate 999 permutations and test whether or not $\beta_1 = \beta_2 = \beta_3$. Under the null-hypothesis it should not matter if labels are switched, and if so, we should get that our original value for the F-statistic should not be extreme compared to the ones calculated after the permutations.

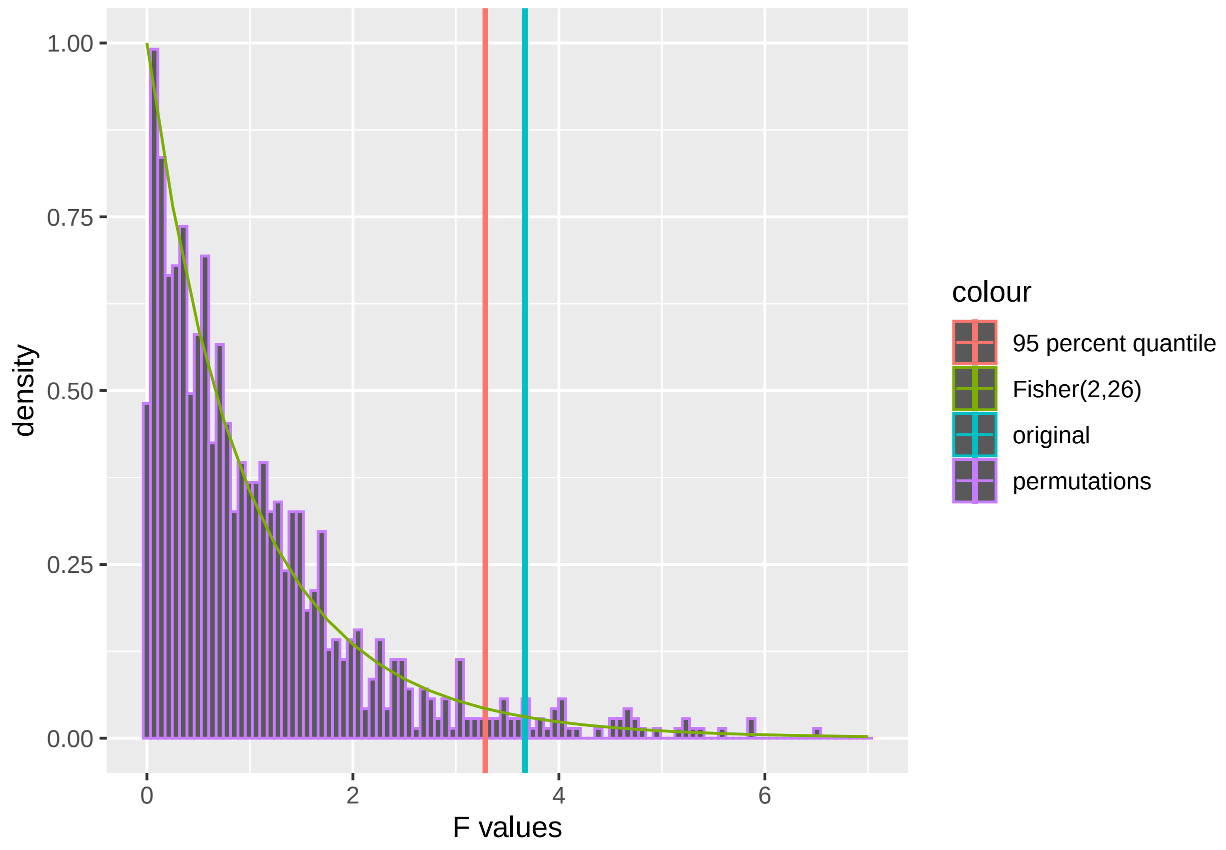

Figure 4: Histogram of F-statistics from all permutations, accompanied with a 95 percent prediction interval.

In Figure 4 the test statistics calculated from the permutated samples are plotted as histograms, and the density plot of the Fisher$_{2,26}$ distribution is plotted as a line. From Figure 4 we clearly see that the f-statistics calculated from the permutations, closely follows the Fisher distribution, as expected. In Figure 4 we also see that the original f-statistic is outside the 95%-quantile of the permutations test values. This is shown when we estimate the p-value as 0.035035 and this is clearly below 0.05. Therefore it is only reasonable to reject the null-hypothesis.

The EM algorithm and bootstrapping

This part of the report wants to explore the expectation maximization (EM) algorithm. Here, we consider the variables $\{x_i\}, i = 1, \ldots, n \quad \sim \text{Exp}(\lambda_0)$ and $\{y_i\}, i = 1, \ldots, n \quad \sim \text{Exp}(\lambda_1)$ which are all independent. We do not directly observe $\{x_i\}, \{y_i\},\ i = 1, \ldots, n$, instead we observe the relations $z_i = \max(x_i, y_i)$ and $u_i = I(x_i \geq y_i)$ for $i = 1, \ldots, n$, where $I(\cdot)$ is the indicator function for a condition $(\cdot)$.

Expected log likelihood for complete data

We would like to get a maximum likelihood estimate for $(\lambda_0, \lambda_1)^T$. Firstly, an expression for the joint (log) likelihood for the complete data $(x_i, y_i),\ i = 1, \ldots, n$ is required. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\theta = (\lambda_0, \lambda_1)$. Thus,

$$\ln(\mathcal{L}(\theta; \boldsymbol{x}, \boldsymbol{y})) = l(\theta; \boldsymbol{x}, \boldsymbol{y}) = \ln(f(\boldsymbol{x}, \boldsymbol{y}|\lambda_0, \lambda_1))$$

Let f_x and f_y be the pdf of $\{x_i\}$ and $\{y_i\}$. Due to independence, we get

$$\ln(f(\boldsymbol{x}, \boldsymbol{y}|\lambda_0, \lambda_1)) = \ln\left(\prod_{i=1}^n f_x(x_i|\lambda_0) f_y(y_i|\lambda_1)\right) = \ln\left(\prod_{i=1}^n \lambda_0\lambda_1 e^{-\lambda_0 x_i} e^{-\lambda_1 y_i}\right)$$

$$= n(\ln(\lambda_0) + \ln(\lambda_1)) - \lambda_0 \sum_{i=1}^n x_i - \lambda_1 \sum_{i=1}^n y_i.$$

Now, we wish to use this to find out what $E[\ln(f(\boldsymbol{x}, \boldsymbol{y}|\lambda_0, \lambda_1))|\boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}]$ is equal to. We plug in the expression for $\ln(f(\boldsymbol{x}, \boldsymbol{y}|\lambda_0, \lambda_1))$

$$E[\ln(f(\boldsymbol{x}, \boldsymbol{y}|\lambda_0, \lambda_1))|\boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}] = E\left[n(\ln(\lambda_0) + \ln(\lambda_1)) - \lambda_0 \sum_{i=1}^n x_i - \lambda_1 \sum_{i=1}^n y_i \middle| \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right]$$

Since n, λ_0, λ_1 are fixed, the expression becomes

$$n(\ln(\lambda_0) + \ln(\lambda_1)) - \lambda_0 \sum_{i=1}^n E[x_i|\boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}] - \lambda_1 \sum_{i=1}^n E[y_i|\boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}].$$

Now we wish to find an expression for $E[x_i|\boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}]$. Firstly, due to independence, x_i is independent of z_j, u_j for $j \neq i$. Therefore $E[x_i|\boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}] = E[x_i|z_i, u_i, \lambda_0^{(t)}, \lambda_1^{(t)}]$. In addition, x_i does not depend on λ_1 and therefore it does not depend on the value of $\lambda_1^{(t)}$ either, i.e. $E[x_i|z_i, u_i, \lambda_0^{(t)}]$. We have two different scenarios to consider:

1) $x_i \geq y_i \implies z_i = x_i,\ u_i = 1,$
2) $x_i < y_i \implies z_i = y_i,\ u_i = 0.$

This in turn means that we can say

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

Lets start with case 1) finding $E[x_i|z_i, u_i = 1, \lambda_0^{(t)}]$. Since in this case $z_i = x_i$ we have $E[x_i|z_i = x_i, u_i = 1, \lambda_0^{(t)}] = z_i$. Now, case 2) finding $E[x_i|z_i = y_i, u_i = 0, \lambda_0^{(t)}]$. We have that

$$E[x_i|z_i = y_i, u_i = 0, \lambda_0^{(t)}] = \int_0^\infty x_i f(x_i|z_i = y_i, u_i = 0, \lambda_0^{(t)}) dx_i.$$

Now

$$f(x_i|z_i = y_i, u_i = 0, \lambda_0^{(t)}) = \begin{cases} 0,\ x_i \geq z_i \\ \frac{\lambda_0^{(t)} e^{-\lambda_0^{(t)} x_i}}{1 - e^{-\lambda_0^{(t)} z_i}},\ x_i < z_i \end{cases}$$

Hence

$$\int_0^\infty x_i f(x_i|z_i = y_i, u_i = 0, \lambda_0^{(t)}) dx_i = \int_0^{z_i} x_i \frac{\lambda_0^{(t)} e^{-\lambda_0^{(t)} x_i}}{1 - e^{-\lambda_0^{(t)} z_i}} dx_i = \frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)} z_i} - 1}$$

Thus, we get that

$$E[x_i|z_i, u_i, \lambda_0^{(t)}] = u_i z_i + (1 - u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)} z_i} - 1}\right).$$

Following a similar argument for $E[y_i|z_i,u_i,\lambda_1^{(t)}]$, we get that

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

Therefore, the expression for

$$E[\ln(f(\boldsymbol{x},\boldsymbol{y}|\lambda_0,\lambda_1))|\boldsymbol{z},\boldsymbol{u},\lambda_0^{(t)},\lambda_1^{(t)}] =$$

$$n(\ln(\lambda_0)+\ln(\lambda_1)) - \lambda_0\sum_{i=0}^{n}\left[u_iz_i + (1-u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)}z_i}-1}\right)\right] - \lambda_1\sum_{i=1}^{n}\left[(1-u_i)z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{e^{\lambda_1^{(t)}z_i}-1}\right)\right].$$

We define $Q(\boldsymbol{\lambda}) = E[\ln(f(\boldsymbol{x},\boldsymbol{y}|\lambda_0,\lambda_1))|\boldsymbol{z},\boldsymbol{u},\lambda_0^{(t)},\lambda_1^{(t)}]$ for the sake of notation.

Maximum likelihood estimates using the EM algorithm

For the next step in the EM-algorithm, we need to define a recursion in $(\lambda_0^{(t)},\lambda_1^{(t)})^T$ for finding maximum likelihood estimates for $(\lambda_0,\lambda_1)^T$. This is done by finding the maximum of the aforementioned expression with respect to λ_0 and λ_1. The gradient with respect to λ_0 and λ_1 becomes

$$\nabla_{(\lambda_0,\lambda_1)}\left(E[x_i|\boldsymbol{z},\boldsymbol{u},\lambda_0^{(t)},\lambda_1^{(t)}]\right) = \begin{pmatrix} \frac{n}{\lambda_0} - \sum_{i=1}^{n}\left[u_iz_i + (1-u_i)\left(1/\lambda_0^{(t)} - \frac{z_i}{\exp(\lambda_0^{(t)}z_i)-1}\right)\right] \\ \frac{n}{\lambda_1} - \sum_{i=1}^{n}\left[(1-u_i)z_i + u_i\left(1/\lambda_1^{(t)} - \frac{z_i}{\exp(\lambda_1^{(t)}z_i)-1}\right)\right]\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix},$$

and solving for $(\lambda_0,\lambda_1)^T =: (\lambda_0^{(t+1)},\lambda_1^{(t+1)})^T$ to obtain the recursion formula, we get

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

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

The algorithm is implemented below.

```r
Estep <- function(lambda, lambda.prev, ui, zi){
  # E-step for EM-algorithm
  # Computes expected value of log-likelihood function
  #
  # lambda     : (lambda_0, lambda_1) for current iteration
  # lambda.prev: (lambda_0, lambda_1) for previous iteration
  # ui         : Observed data u (size 200)
  # zi         : Observed data z (size 200)
  # Returns    : E[xi | ...] expected value given observed data
  n = length(zi)
  return(
    n * (log(lambda[1]) + log(lambda[2]))
    - lambda[1] * sum(ui*zi + (1-ui)*(1/lambda.prev[1] - zi/(exp(lambda.prev[1]*zi)-1)))
    - lambda[2] * sum((1-ui)*zi + ui*(1/lambda.prev[2] - zi/(exp(lambda.prev[2]*zi)-1)))
  )

}
EMstep <- function(lambda.prev, ui, zi){
  # EM-step / Iterative step to get next lambda values using
  # recursion formula above
  # lambda.prev : (lambda_0, lambda_1) for current iteration
  # ui          : Observed data u (size 200)
  # zi          : Observed data z (size 200)
  # Returns     : (lambda_0, lambda_t) for iteration t+1
  n = length(zi)
  return(c(
    n/(sum( # = lambda_0^(t+1)
      ui*zi+(1-ui)*(1/lambda.prev[1]-zi/(exp(lambda.prev[1]*zi)-1))
    )),
    n/(sum( # = lambda_1^(t+1)
      (1-ui)*zi+ui*(1/lambda.prev[2]-zi/(exp(lambda.prev[2]*zi)-1))
    ))
  ))
}


EM <- function(tol, u, z){
  # EM algorithm
  #
  # tol     : Tolerance, determines stopping criterion
  # ui      : Observed data u (size 200)
  # zi      : Observed data z (size 200)
  # Returns : list of expected loglik over and norm difference for each iter,
  #           as well as predicted (lambda_0, lambda_1)

  # Initialize containers
  convergence = FALSE
  lambdas = c(1,1) # Initial value for lambdas
  expected.loglik = c()
  while(! convergence){
    lambdas.new = EMstep(lambdas, u, z) # Compute new (lambda_0, lambda_1)

    # Compute metrics to compare convergence (expected loglik, norm difference)
    expected.loglik = c(expected.loglik, Estep(lambdas.new, lambdas, u, z))

    # Determine if algorithm has converged
    convergence = if(length(expected.loglik) < 2) FALSE else (
      abs(diff(tail(expected.loglik,2))) < tol)
    lambdas = lambdas.new
  }
  return(list(loglik=expected.loglik, lambdas=lambdas))
}

EM.results = EM(1e-8, u, z)
lambda.hat = EM.results$lambdas
```

The algorithm yields $\widehat{\boldsymbol{\lambda}} = (\widehat{\lambda}_0, \widehat{\lambda}_1)^T = (3.465735, 9.3532149)^T$ after 23 iterations. The stopping criterion used in the algorithm is that $|Q(\boldsymbol{\lambda}^{(t)}) - Q(\boldsymbol{\lambda}^{(t-1)})| <$ `tol`, where `tol` is chosen as 10^{-8}. Figure 5 shows how $Q(\boldsymbol{\lambda})$ changes over iterations, highlighting a rapid convergence. The pattern from the plot to the left in the same figure indicates that the tolerance chosen may have been too strict, as most of the iterations stabilize around the same value.

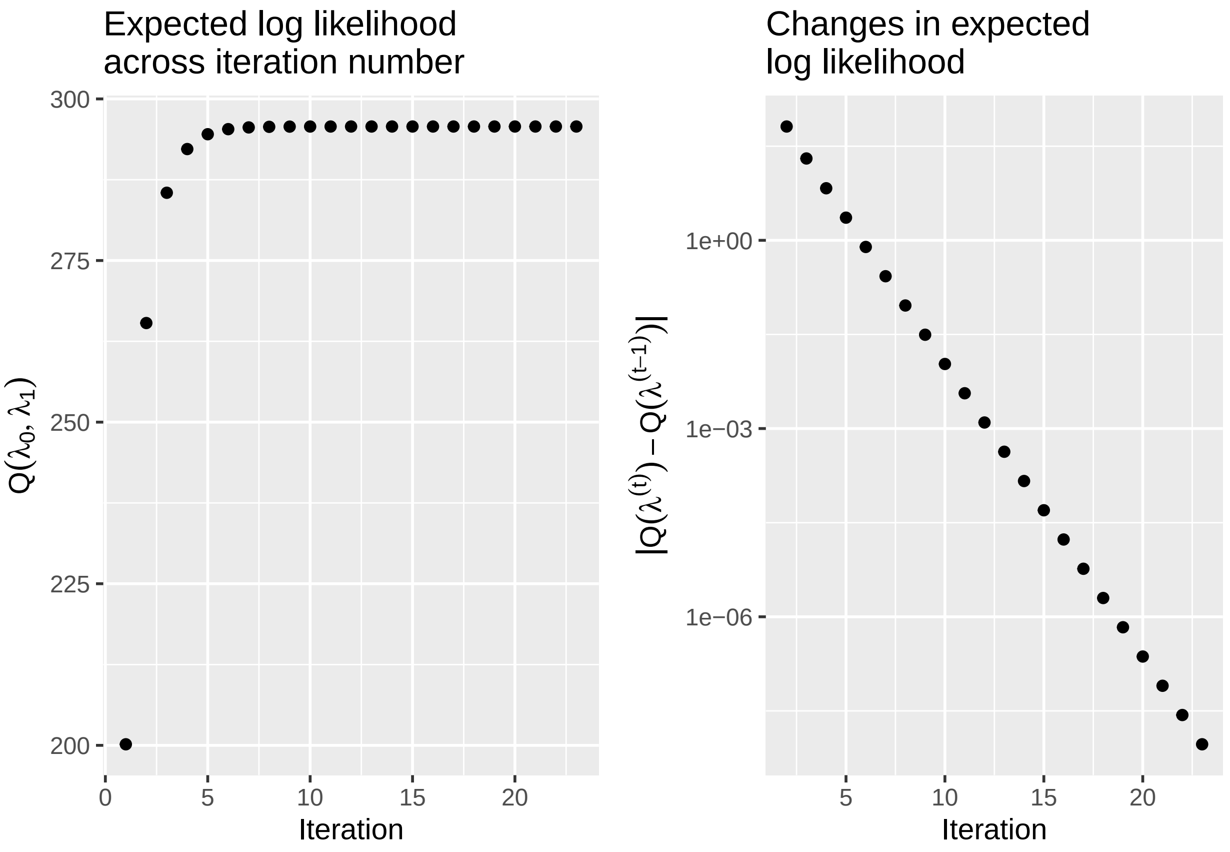

Figure 5: Two plots highlighting rate of convergence for the EM algorithm. The plot to the right shows how each new iteration adjusts the log likelihood by smaller and smaller values. The plot to the left shows how the expected value of the log likelihood of the complete data quickly stabilizes around iteration 5.

Bootstrapping for the EM algorithm

A significant drawback of the EM-algorithm is that it does not provide an estimate of uncertainty. To remedy this, we can use parametric bootstrap to get an estimate of the error. This can be implemented as follows

```
## Psudeocode for bootstrap ##
Compute initial (lambda_0, lambda_1) from EM-algorithm
For i = 1, ..., B do:
  Draw x_1, ..., x_n ~ Exp(lambda_0)
  Draw y_1, ..., y_n ~ Exp(lambda_1)
  For each pairwise (x, y) sampled do:
    z* = max(x, y)
    u* = 1 when x >= y, otherwise 0
  end For
  Fit new (lambda_0, lambda_1) using EM-algorithm with bootstrapped z*, u*
end For
```

Collecting all B new pairs of $(\widehat{\lambda}_0^*, \widehat{\lambda}_1^*)^T$, one can use its empirical statistical properties to get an estimate for the properties of the original $(\widehat{\lambda}_0, \widehat{\lambda}_1)^T$. The bootstrap algorithm and computations of these measures are implemented below.

```r
bootstrap.EM <- function(B, lambda, datadim=200, tol.EM=1e-8){
  # Bootstrap for EM Algorithm
  #
  # B       : Number of bootstraps
  # lambda  : Fitted (lambda_0, lambda_1) using EM
  # datadim : Number of data points, defaults to 200
  # tol.EM  : tolerance to be used in EM bootstrapping, defaults to 1e-8
  # Returns : Bx2-length array containing bootstrapped (lambda_0*, lambda_1*)
  n = datadim
  bootstrap.res = array(dim=c(B,2))
  for(b in 1:B){
    x.star = rexp(n, lambda[1]) # x_1*, ..., x_n*
    y.star = rexp(n, lambda[2]) # y_1*, ..., y_n*
    u.star = ifelse(x.star >= y.star, 1, 0) # Indicator function
    z.star = pmax(x.star, y.star) # max(x_j, y_j) \forall j=1, ..., n
    bootstrap.res[b, ] = EM(tol.EM, u.star, z.star)$lambdas
  }
  return(bootstrap.res)
}

bootstrap.estimates <- function(result, lambda){
  # Compute bootstrap metrics using result from function above
  #
  # res     : Bx2 array of bootstrap (lambda0, lambda1)
  # lambda  : 2-length list of initial (lambda0, lambda1) to compare
  # Returns : list of metrics (std.dev, bias and correlation)

  estimates = list()
  estimates$stddev = c( sd(result[, 1]), sd(result[, 2]))
  estimates$bias = c(mean(result[, 1]) - lambda[1], mean(result[, 2]) - lambda[2])
  estimates$correlation = cor(result[, 1], result[, 2])
  return(estimates)
}
```

The results are shown Table 2, using $B = 1500$.

Table 2: Estimated standard deviation, bias and correlation for lambdas using bootstrapping.

	Standard deviation	Bias	Correlation
$\widehat{\lambda}_0$	0.2553156	0.0148333	-0.0432358
$\widehat{\lambda}_1$	0.8548547	0.0817101	-0.0432358

The bias seem to be about 0.86% and 0.51% of the estimated value, which is small. The standard deviation is also of magnitude 10^{-1} for both $\widehat{\lambda}_i$. In addition, they do not seem to be correlated since the correlation is of order 10^{-2}. Considering that the bias is so low, there is not incentive enough to use the bias-corrected estimates for $\boldsymbol{\lambda}$, unless further computations require extremely precise estimates. Using the bias-corrected estimates will also increase the variance which is already considerably high compared to the bias.

Analytical maximum likelihood estimators and direct optimization

We now want to find an analytical formula for $f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1)$ in pursuit of finding an analytical maximum likelihood estimator for λ_0 and λ_1. Firstly it is worth noting the following:

1) For $Z_i = z_i$ we have $(x_i = z_i \cap y_i \leq z_i) \cup (x_i \leq z_i \cap y_i = z_i)$.
2) For $U_i = 1$ we have $x_i \geq y_i$.
3) For $U_i = 0$ we have $x_i < y_i$.

Therefore

$$f_{Z_i,U_i}(z_i, u_i = 1|\lambda_0, \lambda_1) = P(x_i = z_i, y_i \leq z_i|\lambda_0, \lambda_1).$$

Since x_i and y_i independent, this becomes

$$f_{Z_i,U_i}(z_i, u_i = 1|\lambda_0, \lambda_1) = f_{x_i}(x_i = z_i|\lambda_0)F_{y_i}(z_i|\lambda_1) = \lambda_0 e^{-\lambda_0 z_i}(1 - e^{-\lambda_1 z_i}).$$

Furthermore

$$\begin{aligned} f_{Z_i,U_i}(z_i, u_i = 0|\lambda_0, \lambda_1) &= P(x_i < z_i, y_i = z_i|\lambda_0, \lambda_1) \\ &= f_{y_i}(z_i|\lambda_1)F_{x_i}(z_i|\lambda_0) = \lambda_1 e^{-\lambda_1 z_i}(1 - e^{-\lambda_0 z_i}). \end{aligned}$$

Hence,

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

This can be written compactly as $f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1) = u_i\lambda_0 e^{-\lambda_0 z_i}(1 - e^{-\lambda_1 z_i}) + (1 - u_i)\lambda_1 e^{-\lambda_1 z_i}(1 - e^{-\lambda_0 z_i})$. Now, we write out the expression for the log-likelihood

$$\begin{aligned} l(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u}) &= \ln\left(\prod_{i=1}^{n} \left[u_i\lambda_0 e^{-\lambda_0 z_i}(1 - e^{-\lambda_1 z_i}) + (1 - u_i)\lambda_1 e^{-\lambda_1 z_i}(1 - e^{-\lambda_0 z_i})\right]\right) \\ &= \ln\left(\prod_{u_i=1} \lambda_0 e^{-\lambda_0 z_i}(1 - e^{\lambda_1 z_i}) \prod_{u_i=0} \lambda_1 e^{-\lambda_1 z_i}(1 - e^{\lambda_0 z_i})\right) \\ &= \sum_{u_i=1} \left(\ln(\lambda_0) - \lambda_0 z_i + \ln(1 - e^{-\lambda_1 z_i})\right) + \sum_{u_i=0} \ln(\lambda_1) - \lambda_1 z_i + \ln(1 - e^{-\lambda_0} z_i). \end{aligned}$$

Now we will find the gradient of $l(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u})$ with respect to λ_0 and λ_1.

$$\frac{\partial}{\partial\lambda_0} l(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u}) = \sum_{u_i=1} \left(\frac{1}{\lambda_0} - z_i\right) + \sum_{u_i=0} \left(\frac{z_i e^{-\lambda_0 z_i}}{1 - e^{-\lambda_0 z_i}}\right)$$

$$\frac{\partial}{\partial\lambda_1} l(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u}) = \sum_{u_i=1} \left(\frac{z_i e^{-\lambda_1 z_i}}{1 - e^{-\lambda_1 z_i}}\right) + \sum_{u_i=0} \left(\frac{1}{\lambda_1} - z_i\right)$$

Although the expression is possible to simplify analytically, we maximize the log likelihood numerically using the `optim` function. This function finds the mimimum, so the implementation solves for minimum of $-l_{Z,U}$ (i.e. maximum of $l_{Z,U}$), shown below.

```r
loglik.zu <- function(lambda, z, u){
  # Negative of log-likelihood as discussed above
  return( -1 *(
  sum(log(lambda[1]*exp(-lambda[1]*z[u==1])*(1-exp(-lambda[2]*z[u==1])))) +
  sum(log(lambda[2]*exp(-lambda[2]*z[u==0])*(1-exp(-lambda[1]*z[u==0]))))
  ))
}

lambda.hat.MLE = optim(c(1,1), loglik.zu, z=z, u=u)
```

The values for the maximum likelihood estimates become $\widehat{\boldsymbol{\lambda}}_{MLE} = (\widehat{\lambda}_{0,MLE}, \widehat{\lambda}_{1,MLE})^T = (3.46589, 9.3511034)^T$. We see that these estimates closely resemble those from the EM algorithm, with a difference of $(1.5501365 \times 10^{-4}, 0.0021115)$.

The direct optimization can be more effective than the iterative nature of the EM algorithm. In addition, it is possible to obtain performance measures like variance of the MLE without bootstrapping. A clear downside of using direct optimization, is that it intrinsically is not a general method, meaning that it must be adapted to each problem, and often require calculations before it can be implemented.