TMA4300: Exercise 3

Øyvind Klåpbakken
Jakob Gerhard Martinussen

Spring 2019

Problem A

In this problem we will analyze the data `data3A$x` provided in the file `probAdata.R`.

```
source(here::here("reports", "report_3", "data", "probAdata.R"))
```

The data is assumed to be distributed according to the AR(2) model defined 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 zero mean and constant variance, with no assumption on the exact distribution.This makes $\boldsymbol{x}$ a non-Gaussian time-series. Our sequence is of length $T = 100$.

We will use two methods in order to estimate $\boldsymbol{\beta} := [\beta_1, \beta_2]$:

$$\hat{\boldsymbol{\beta}}_{LS} = \underset{\beta_1,\beta_2}{\operatorname{argmin}}\, Q_{LS}(\boldsymbol{x}), \qquad Q_{LS}(\boldsymbol{x}) := \sum_{t=3}^{T} \left(x_t - \beta_1 x_{t-1} - \beta_2 x_{t-2}\right)^2.$$

$$\hat{\boldsymbol{\beta}}_{LA} = \underset{\beta_1,\beta_2}{\operatorname{argmin}}\, Q_{LA}(\boldsymbol{x}), \qquad Q_{LA}(\boldsymbol{x}) := \sum_{t=3}^{T} \left|x_t - \beta_1 x_{t-1} - \beta_2 x_{t-2}\right|.$$

With other words, optimizing the sum of squared residuals (LS) or the absolute residuals (LA), respectively. The minimizers can be calculated with the `R`-function `ARp.beta.est` provided in the file `probAhelp.R`.

```
source(here::here("reports", "report_3", "data", "probAhelp.R"))
```

We start off by using the supplied function to estimate $\boldsymbol{\beta}_{LS}$ and $\boldsymbol{\beta}_{LA}$.

```
x ⟵ data3A$x
beta ⟵ ARp.beta.est(x, 2)
```

The innovations $\hat{e}_t$ can be estimated by

$$\hat{e}_t = x_t - \hat{\beta}_1 x_{t-1} - \hat{\beta}_2 x_{t-2},\ t = 3, \ldots, T$$

The innovations are estimated using either $\hat{\boldsymbol{\beta}}_{LS}$ or $\hat{\boldsymbol{\beta}}_{LA}$. We will denote the different estimated innovations by $\hat{e}_t^{LS}$ and $\hat{e}_t^{LA}$.

The estimated innovations $\hat{\epsilon}_t$ are then centered by calculating

$$\hat{\epsilon}_t = \hat{e}_t - \bar{e}$$

This is implemented by the function `estimate_pseudo_innovations`.

```
estimate_pseudo_innovations ⟵ function(beta, x) {
  L ⟵ length(x)
  xs ⟵ list(
    xt = x[seq(3, L)],
    xt1 = x[seq(2, L - 1)],
    xt2 = x[seq(1, L - 2)]
  )

  i ⟵ xs %>%
    pmap_dbl(function(xt, xt1, xt2)
      xt - beta[1] * xt1 - beta[2] * xt2)

  pi ⟵ i %>%
    subtract(mean(i))
  return(pi)
}

ls_pi ⟵ estimate_pseudo_innovations(beta$LS, x)
la_pi ⟵ estimate_pseudo_innovations(beta$LA, x)
```

The residuals are plotted in Figure 1. We observe that the residuals are evenly distributed around 0, except for one residual which is much larger than all others.

```
residuals_df ⟵ map2_dfr(
  list(ls_pi, la_pi), list("ls", "la"),
  ~tibble(residual = .x,
          index = seq(1, length(.x)),
          type = .y)
)

residuals_df %>%
  ggplot(aes(x=index, y=residual)) +
  geom_point(bins=50) +
  facet_wrap(~type)
```

We proceed by obtaining $B = 1500$ bootstrap samples of the data sequence for each of the estimators $\hat{\boldsymbol{\beta}}_{LS}$ and $\hat{\boldsymbol{\beta}}_{LA}$. Each bootstrap sample is obtained by first picking a random consecutive sequence of length 2 from the original data sequence $\boldsymbol{x}$ and calculating the remaining $T-2$ data points by using the relation

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

ϵ is chosen randomly from either the set $\{\hat{\epsilon}_3^{LS}, \ldots, \hat{\epsilon}_T^{LS}\}$ or $\{\hat{\epsilon}_3^{LA}, \ldots, \hat{\epsilon}_T^{LA}\}$, depending on which of the two estimators $\hat{\boldsymbol{\beta}}_{LS}$ and $\hat{\boldsymbol{\beta}}_{LA}$ we're using to obtain the sample.

```
calculate_xt ⟵ function(beta, xt1, xt2, pi) {
  beta[1] * xt1 + beta[2] * xt2 + pi
}
```

For each of the B resampled data sequences we estimate $\boldsymbol{\beta}_{LS}$ and $\boldsymbol{\beta}_{LA}$.

```
resample_x ⟵ function(pseudo_innovations, beta, x) {
  L ⟵ length(x)
  rs_pi ⟵ sample(pseudo_innovations,
    L - 2,
    replace = TRUE
  )
```

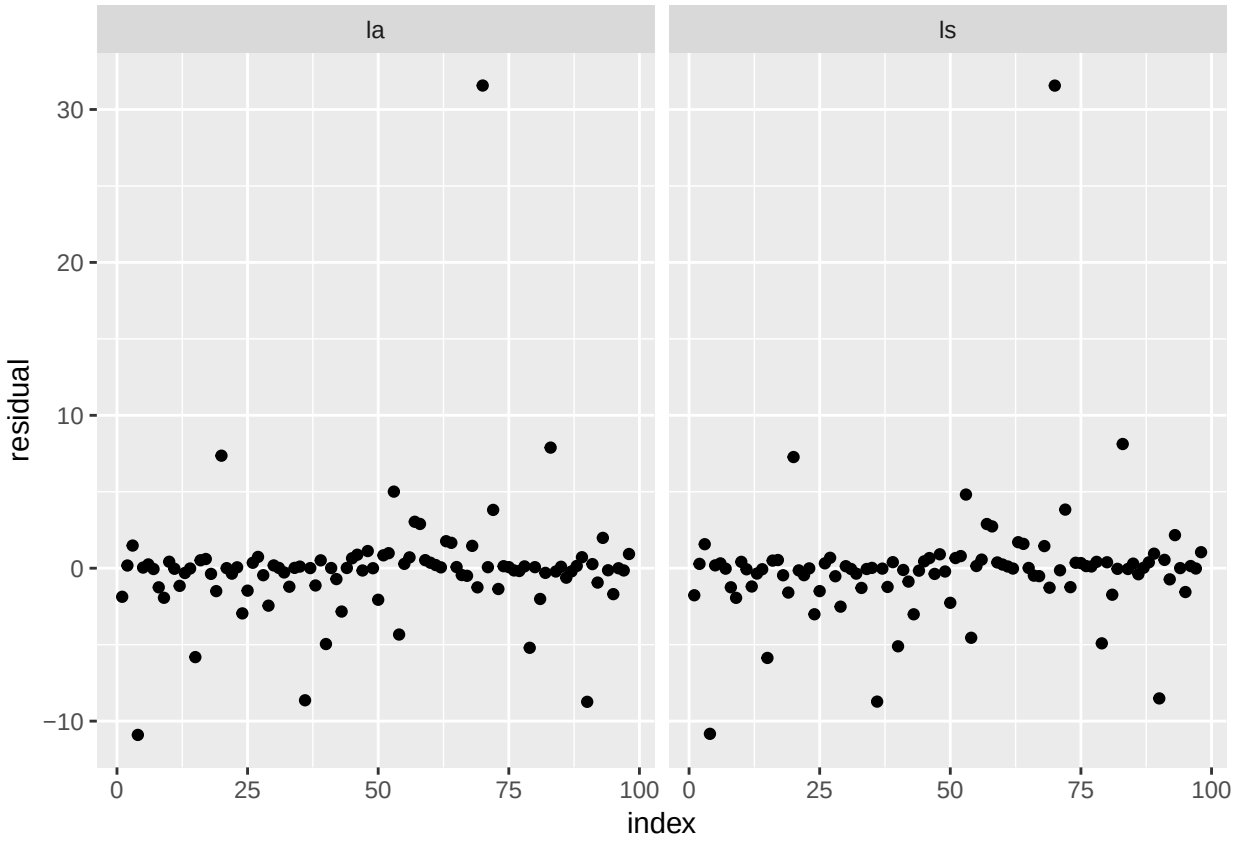

Figure 1: Estimated centered residuals using the two estimators.

```
  init_index ← sample(seq(1, L - 1), 1)
  init_x ← x[init_index:(init_index + 1)]
  xt2 ← init_x[1]
  xt1 ← init_x[2]
  resampled_x ← init_x
  for (i in 1:(L - 2)) {
    xt ← calculate_xt(beta, xt1, xt2, rs_pi[i])
    resampled_x ← c(resampled_x, xt)
    xt2 ← resampled_x[i + 1]
    xt1 ← resampled_x[i + 2]
  }

  return(resampled_x)
}

B ← 1500
resampled_xs_ls ← B %>% rerun(resample_x(ls_pi, beta$LS, x))
resampled_xs_la ← B %>% rerun(resample_x(la_pi, beta$LA, x))

calculate_beta_ls ← as_mapper(~ ARp.beta.est(.x, 2)$LS)
calculate_beta_la ← as_mapper(~ ARp.beta.est(.x, 2)$LA)

bootstrapped_beta_ls ← resampled_xs_ls %>%
  map(calculate_beta_ls) %>%
  map(purrr::set_names, nm = c("beta1", "beta2")) %>%
  transpose() %>%
  as_tibble() %>%
  unnest()

bootstrapped_beta_la ← resampled_xs_la %>%
  map(calculate_beta_la) %>%
  map(purrr::set_names, nm = c("beta1", "beta2")) %>%
  transpose() %>%
  as_tibble() %>%
  unnest()
```

A histogram showing the obtained values are shown in Figure 2 and 3.

```
bootstrapped_beta_ls %>%
  gather() %>%
  ggplot(aes(x = value)) +
  geom_histogram(bins = 100) +
  facet_wrap(~key, scales = "free_x")
```

```
bootstrapped_beta_la %>%
  gather() %>%
  ggplot(aes(x = value)) +
  geom_histogram(bins = 100) +
  facet_wrap(~key, scales = "free_x")
```

The *bias* of our estimator is defined as

$$\text{bias}_F\left(\hat{\beta}, \beta\right) = E_F\left[\hat{\beta}\right] - \beta,$$

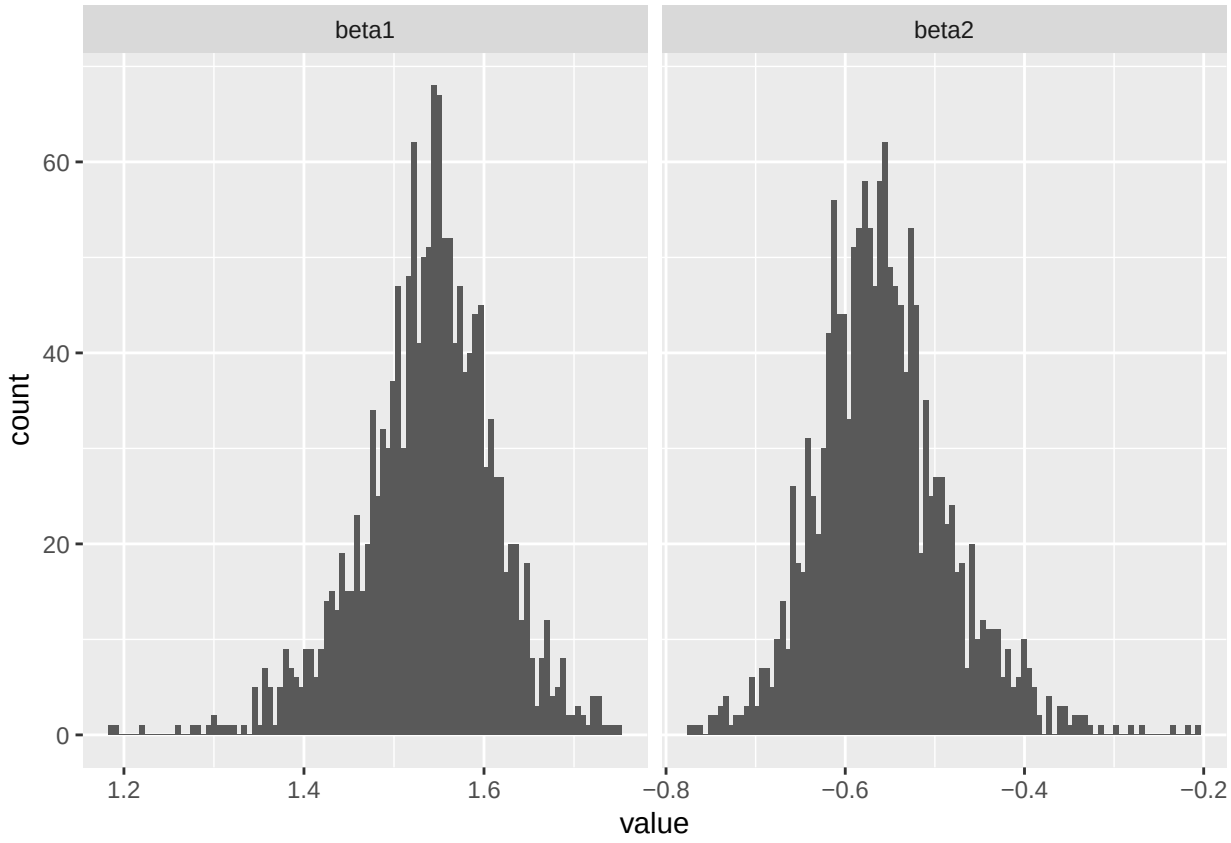

Figure 2: Histogram of $\hat{\beta}_{LS}$ obtained from bootstrap samples of the data sequence.

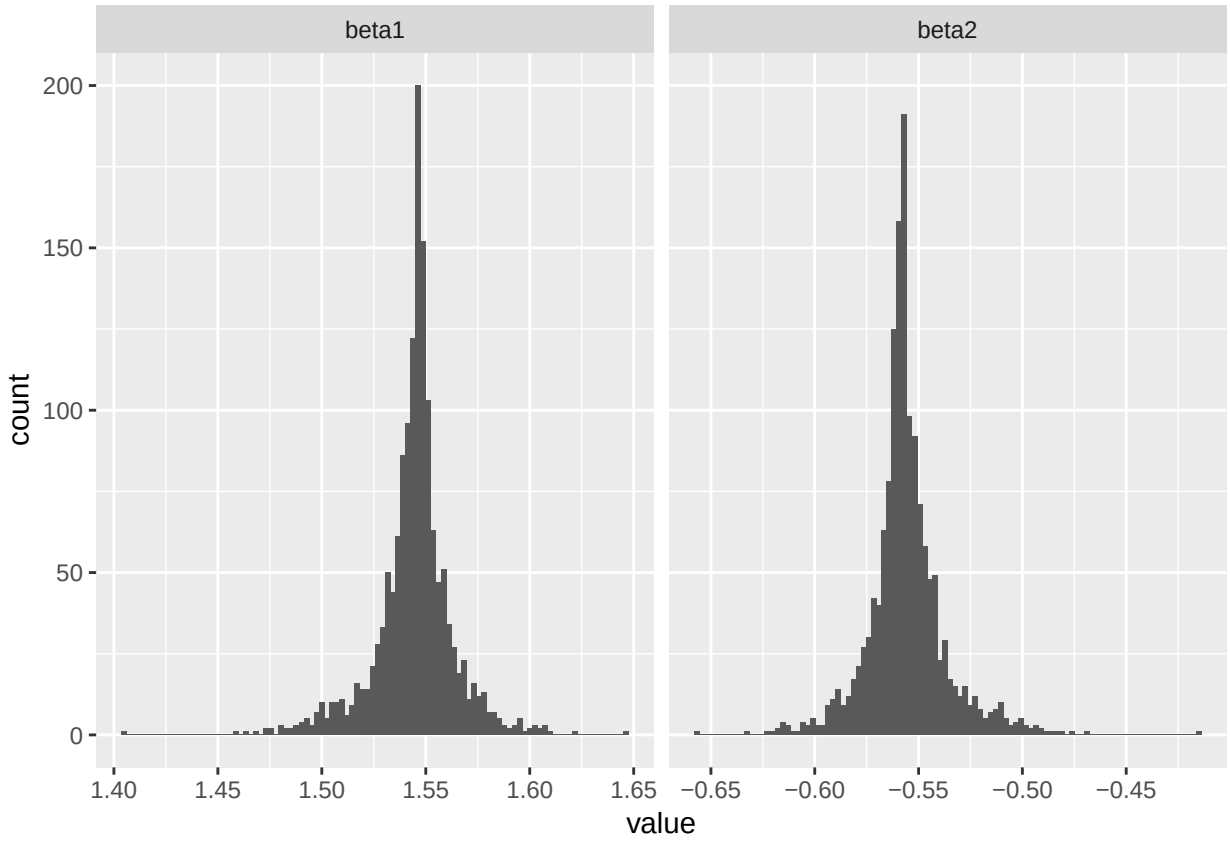

Figure 3: Histogram of $\hat{\beta}_{LA}$ obtained from bootstrap samples of the data sequence.

where F is the true distribution of the time series. Since we do not know the parametrization of the underlying model, this must be estimated with our bootstrap replicates $\hat{\beta}^\star$

$$\widehat{\text{bias}}_B = \hat{\beta}^\star(\cdot) - \hat{\beta},$$

where we have defined

$$\hat{\beta}^\star(\cdot) := \frac{1}{B}\sum_{b=1}^{B} \hat{\beta}^\star(b).$$

The estimated bias for the two components of the two different estimators, $\hat{\boldsymbol{\beta}}_{LS}$ and $\hat{\boldsymbol{\beta}}_{LA}$, can now be calculated.

```
la_bias ← list(
  beta1_bias = bootstrapped_beta_la %>%
    pull(beta1) %>%
    mean() %>%
    subtract(beta$LA[[1]]),
  beta2_bias = bootstrapped_beta_la %>%
    pull(beta2) %>%
    mean() %>%
    subtract(beta$LA[[2]])
)

ls_bias ← list(
  beta1_bias = bootstrapped_beta_ls %>%
    pull(beta1) %>%
    mean() %>%
    subtract(beta$LS[[1]]),
  beta2_bias = bootstrapped_beta_ls %>%
    pull(beta2) %>%
    mean() %>%
    subtract(beta$LS[[2]])
)

ls_bias %>%
  as_tibble() %>%
  knitr::kable(
    caption = "\\label{tab:ls_bias} Estimated bias of $\\hat{\\vec{\\beta}}_{LS}$",
    col.names = c("Bias of $\\hat{\\beta}_1^{LS}$", "Bias $\\hat{\\beta}_2^{LS}$")
    )
```

Table 1: Estimated bias of $\hat{\boldsymbol{\beta}}_{LS}$

Bias of $\hat{\beta}_1^{LS}$	Bias $\hat{\beta}_2^{LS}$
-0.0150943	0.0105704

```
la_bias %>%
  as_tibble() %>%
  knitr::kable(
    caption = "\\label{tab:la_bias}  Estimated bias of $\\hat{\\vec{\\beta}}_{LA}$",
```

```
    col.names = c("Bias of $\\hat{\\beta}_1^{LA}$", "Bias of $\\hat{\\beta}_2^{LA}$")
  )
```

Table 2: Estimated bias of $\hat{\boldsymbol{\beta}}_{LA}$

Bias of $\hat{\beta}_1^{LA}$	Bias of $\hat{\beta}_2^{LA}$
-0.0021238	0.0015458

The bias estimation results are shown in Table 1 and 2.

The variance can be estimated by

$$\widehat{\text{Var}}_B = \frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\beta}^\star(b) - \hat{\beta}^\star(\cdot)\right)^2,$$

which we also implement in the following code:

```
bootstrapped_beta_ls %>%
  select(beta1, beta2) %>%
  summarize(
    beta1_var = var(beta1),
    beta2_var = var(beta2)
    ) %>%
  knitr::kable(
    caption="\\label{tab:ls_var} Estimated variance of $\\hat{\\vec{\\beta}}_{LS}$",
    col.names = c("Variance of $\\hat{\\beta}_1^{LS}$", "Variance $\\hat{\\beta}_2^{LS}$")
    )
```

Table 3: Estimated variance of $\hat{\boldsymbol{\beta}}_{LS}$

Variance of $\hat{\beta}_1^{LS}$	Variance $\hat{\beta}_2^{LS}$
0.005473	0.0054098

```
bootstrapped_beta_la %>%
  select(beta1, beta2) %>%
  summarize(
    beta1_var = var(beta1),
    beta2_var = var(beta2)
    ) %>%
  knitr::kable(
    caption= "\\label{tab:la_var} Estimated variance of $\\hat{\\vec{\\beta}}_{LA}$",
    col.names = c("Variance of $\\hat{\\beta}_1^{LA}$", "Variance $\\hat{\\beta}_2^{LA}$")
    )
```

Table 4: Estimated variance of $\hat{\boldsymbol{\beta}}_{LA}$

Variance of $\hat{\beta}_1^{LA}$	Variance $\hat{\beta}_2^{LA}$
0.0003558	0.0003558

The results can be seen in Table 3 and 4.

The estimated variance for $\hat{\beta}_{LS}$ is a lot higher than for $\hat{\beta}_{LA}$. In addition, the bias seems to be very similar for the two estimators. It's therefore clear that the LS estimator is not optimal for the non-Gaussian time-series we're considering in this problem, seeing as the LA estimator performs better.

A2

In order to obtain a prediction interval for x_{101} we create a prediction for each of the bootstrap samples by sampling a random centered estimated innovation from the set $\{\hat{\epsilon}_3, \ldots, \hat{\epsilon}_T\}$. For a specific bootstrap replicate $\hat{\boldsymbol{\beta}}^\star$ we obtain the prediction by using the relation

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

Again, ϵ denotes a random sample from the corresponding set of centered estimated innovations.

```
sample_x ← function(beta1, beta2, pi, x) {
  return(beta2 * x[99] + beta1 * x[100] + sample(pi, 1))
}

ls_sample_x ← partial(sample_x, pi=ls_pi, x = x)
x_samples ← map2_dbl(bootstrapped_beta_ls$beta1, bootstrapped_beta_ls$beta2, ls_sample_x)

x_df ← enframe(x, name="index") %>%
  bind_rows(
    tibble(
      index=rep(101, length(x_samples)),
      value = x_samples
    )
  )
```

After obtaining the predictions we calculate the 2.5th and 97.5th percentiles in order to obtain a 95% confidence interval.

```
x_summary_df ← x_df %>%
  group_by(index) %>%
  summarise(
    mean = mean(value),
    lq = quantile(value, 0.025),
    uq = quantile(value, 0.975)
  )

x_summary_df %>%
  filter(index == 101) %>%
  select(lq, uq) %>%
  knitr::kable(
    caption = "\\label{tab:conf_int_ls} Confidence interval for $x_{101}$ using $\\hat{\\vec{\\beta}}^{\\star}_{LS
    col.names = c("2.5th percentile", "97.5 percentile")
  )
```

Table 5: Confidence interval for x_{101} using $\hat{\boldsymbol{\beta}}^\star_{LS}$

2.5th percentile	97.5 percentile
7.42839	23.12017

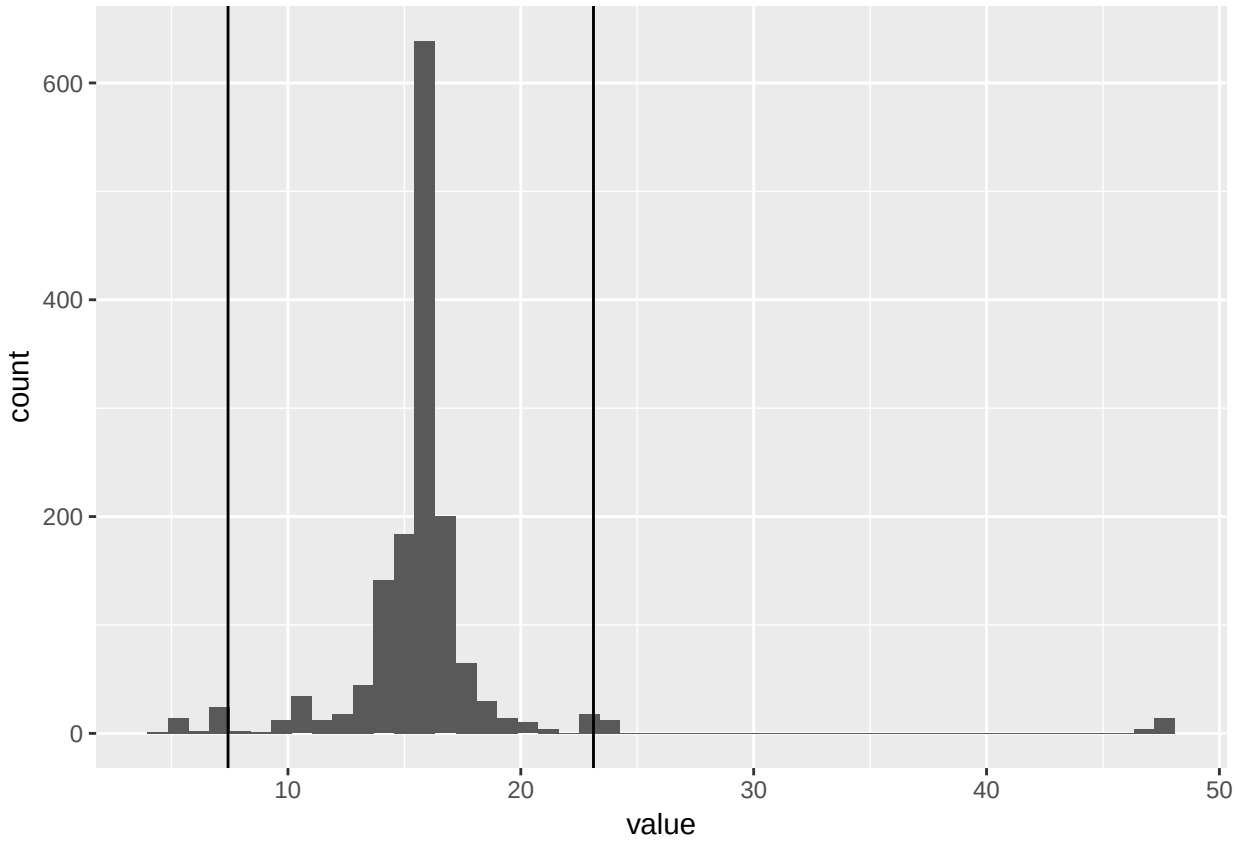

Figure 4: Predicted values of x_{101} using $\hat{\beta}_{LS}$. The black lines show the confidence interval.

The resulting confidence intervals are shown in Table 5 and 6.

```
enframe(x_samples, name="index") %>%
  ggplot() +
  geom_histogram(aes(x=value), bins=50) +
  geom_vline(aes(xintercept = x_summary_df %>%
                   filter(index == 101) %>%
                   pull(lq))) +
  geom_vline(aes(xintercept = x_summary_df %>%
                   filter(index == 101) %>%
                   pull(uq)))
```

The histogram showing the distribution of the predictions can be seen in Figure 4 and 5.

```
la_sample_x ← partial(sample_x, pi=la_pi, x = x)

x_samples ← map2_dbl(bootstrapped_beta_la$beta1, bootstrapped_beta_la$beta2, ls_sample_x)

x_df ← enframe(x, name="index") %>%
  bind_rows(
    tibble(
      index=rep(101, length(x_samples)),
      value = x_samples
```

```
    )
  )

x_summary_df ⟵ x_df %>%
  group_by(index) %>%
  summarise(
    mean = mean(value),
    lq = quantile(value, 0.025),
    uq = quantile(value, 0.975)
  )

x_summary_df %>%
  filter(index == 101) %>%
  select(lq, uq) %>%
  knitr::kable(
    caption = "\\label{tab:conf_int_la} Confidence interval for $x_{101}$ using $\\hat{\\vec{\\beta}}^{\\star}_{LA
    col.names = c("2.5th percentile", "97.5 percentile")
  )
```

Table 6: Confidence interval for x_{101} using $\hat{\beta}^{\star}_{LA}$

2.5th percentile	97.5 percentile
7.217102	23.63717

```
enframe(x_samples, name="index") %>%
  ggplot() +
  geom_histogram(aes(x=value), bins=50) +
  geom_vline(aes(xintercept = x_summary_df %>%
                   filter(index == 101) %>%
                   pull(lq))) +
  geom_vline(aes(xintercept = x_summary_df %>%
                   filter(index == 101) %>%
                   pull(uq)))
```

The value of x_{100} is 16.141801, and the predictions based on both estimators suggests that x_{101} most likely will increase or decrease by less than ~ 8. It's worth noting that the large residual observed in Figure 1 affects the predicted values by contributing to extreme outliers such as the one seen to the right in Figure 5 and 4.

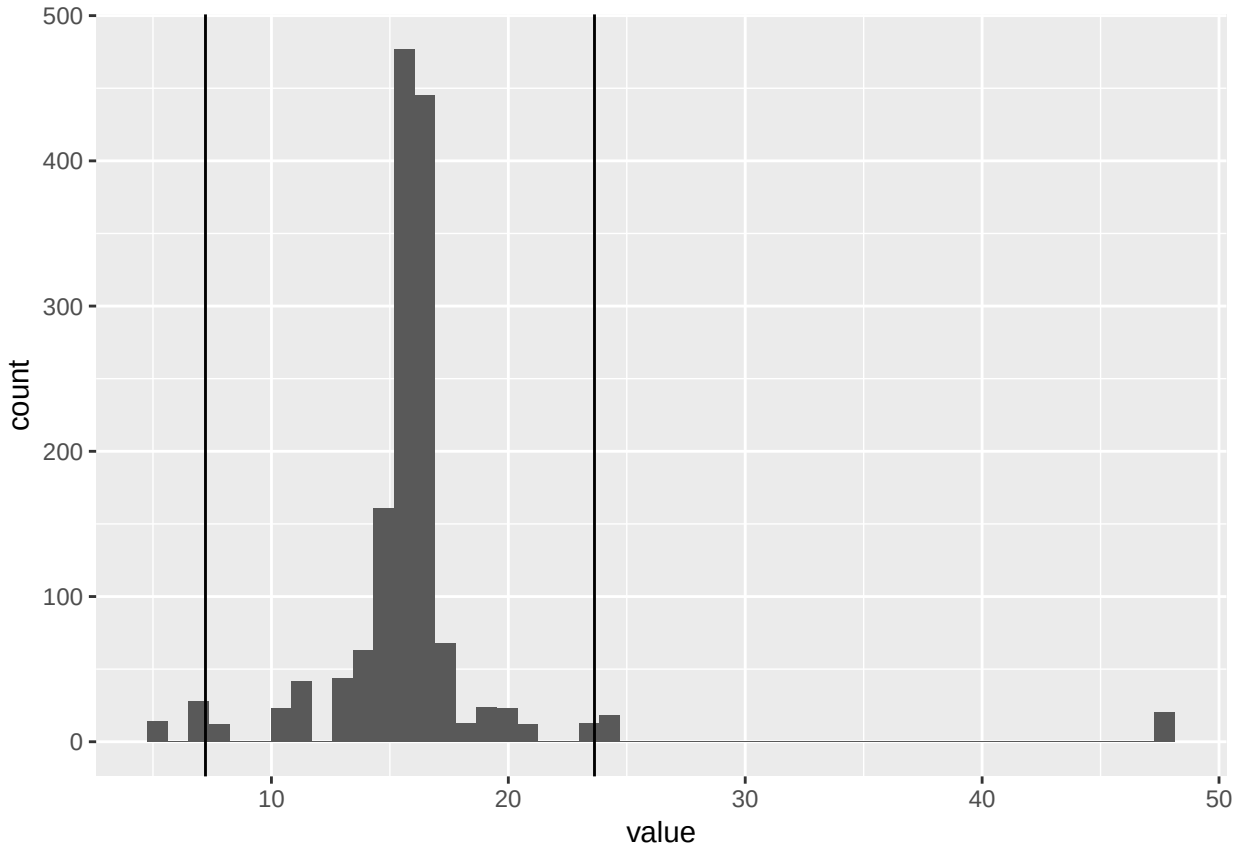

Figure 5: Predicted values of x_{101} using $\hat{\beta}_{LA}$. The black lines show the confidence interval.

Problem B

The concentration of bilirubin (mg/dL) in blood samples from three young men was taken by Jørgensen in 1993.

Individual	Concentration (mg/dL)										
p1	0.14	0.20	0.23	0.27	0.27	0.34	0.41	0.41	0.55	0.61	0.66
p2	0.20	0.27	0.32	0.34	0.34	0.38	0.41	0.41	0.48	0.55	
p3	0.32	0.41	0.41	0.55	0.55	0.62	0.71	0.91			

Table 7: Blood concentration of bilirubin from three young, male individuals.

The resulting data is shown in table 7, and provided in the file `bilirubin.txt`.

```
bilirubin ← here::here("reports", "report_3", "data", "bilirubin.txt") %>%
  read.table(header=T) %>%
  as_tibble()
```

B1

We start off by creating boxplot in order to explore how logarithm of the concentration is distributed across the three different individuals.

```
bilirubin %>%
  mutate(log_meas = log(meas)) %>%
  ggplot(aes(x=pers, y=log_meas)) +
  geom_boxplot()
```

The boxplot is shown in Figure 6. The plots suggest that there might be a difference between the persons, seeing as the variance is considerably lower for `p2` than it is for the other individuals. We can also observe that the mean is considerably higher for `p3` than it is for the other individuals.

Assume the following distribution of the bilirubin concentration Y of individual i and sample j:

$$\log Y_{ij} = \beta_i + \epsilon_{ij}, \text{with } i = 1, 2, 3 \text{ and } j = 1, ..., n,$$

where $n_1 = 11$, $n_2 = 10$, and $n_3 = 8$, and $\epsilon_{ij} \overset{iid}{\sim} \mathcal{N}(0, \sigma^2)$. We proceed by fitting this linear model to the data, estimating the coefficients $\beta_i,\ i = 1, 2, 3$ associated with each of the individuals.

```
linear_model ← lm(log(meas)~pers, data=bilirubin)
summary(linear_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 *
## —
```

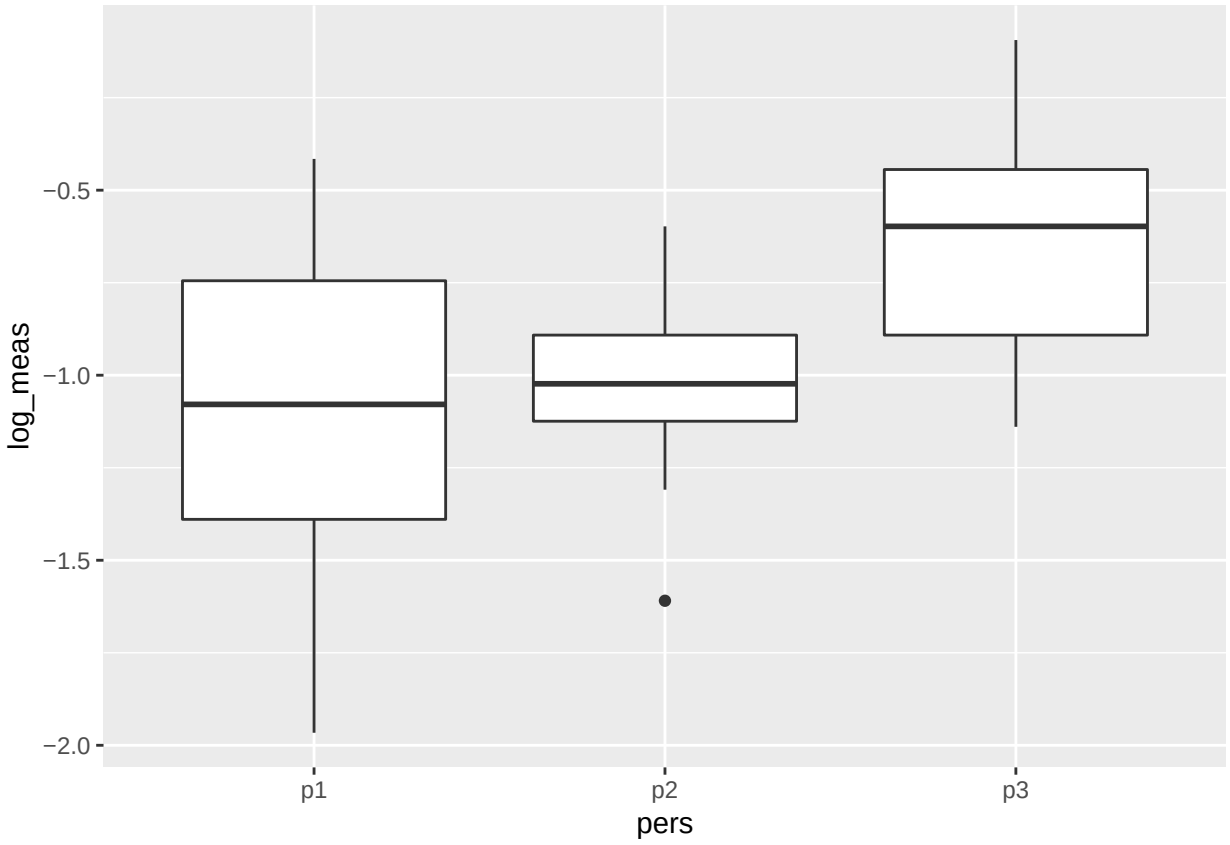

Figure 6: Boxplot for each of the different groups

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## 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
```

The resulting linear model uses *dummy variable encoding* with estimated regression coefficients

$$\hat{\beta}_0 \approx -1.094,$$
$$\hat{\beta}_{\texttt{p2}} \approx 0.064,$$
$$\hat{\beta}_{\texttt{p3}} \approx 0.465.$$

The model therefore predicts

$$\mathrm{E}\left[Y_{ij} \mid i = 1\right] < \mathrm{E}\left[Y_{ij} \mid i = 2\right] < \mathrm{E}\left[Y_{ij} \mid i = 3\right].$$

So the linear model does in fact distinguish between the individuals to some degree, but is this a good enough reason to conclude that $\beta_i \neq \beta_j$ given $i \neq j$?

In order to investigate this, we want to test whether this model performs significantly better than the null-model, i.e. the model that only uses the mean of the data. The F-test statistic can be used in order to test the hypothesis:

$$\mathrm{H}_0 : \beta_1 = \beta_2 = \beta_3$$
$$\mathrm{H}_1 : \beta_i \neq \beta_j, \text{ for some } i \neq j$$

This test is now performed, and the resulting F-test statistic is saved to `Fval`:

```
Fstat ← linear_model %>%
  glance() %>%
  select(statistic, p.value)

Fval ← Fstat$statistic
Fstat %>% kable(format = "pandoc", caption = "\\label{tab:Fstat}Result of F-test.", col.names = c("F-statistic", "
```

Table 8: Result of F-test.

F-statistic	P-value
3.669775	0.0394618

The null-hypothesis must be rejected at a significance level of $\alpha = 0.05$, seeing as the p-value is less than 0.05.

B2

A function that randomly assigns the data to the three different groups, fits a linear model to the data and returns the F-statistics is implemented `permTest()` below.

```
permTest ← function(df){
  perm_df ← tibble(
    meas = df %>% pull(meas) %>%sample(),
    pers = df %>% pull(pers)
  )
  lm(log(meas)~pers, data=perm_df) %>%
```

```
    glance() %>%
    pull(statistic)
}
```

B3

We now apply `permTest()` in order to generate 999 samples of the F-statistic:

```
permTest_df <- tibble(
    run = seq(1, 999),
    F_stat = 999 %>% rerun(permTest(bilirubin))
  ) %>%
  unnest()
```

The null-hypothesis for the permutation test is now

$$H_0 : \text{All data are from the same distribution.}$$

The F-statistic resulting from fitting a linear model to the data is suitable for use with the permutation test. This is because the F-statistic is able to capture some of the information about the difference between the three groups. The permutation test is performed below.

```
perc_95 <- permTest_df %>%
  pull(F_stat) %>%
  quantile(probs = c(0.95))

permTest_df %>%
  ggplot(aes(x=F_stat)) +
  geom_histogram(bins=50) +
  geom_vline(aes(xintercept=perc_95), color="red") +
  geom_vline(aes(xintercept=Fval), color="blue") +
  annotate("text", x = 2, y= 150, color="red", label="95th percentile") +
  annotate("text", x = 5.5, y= 150, color="blue", label="Original F-statistic")
```

```
p_value <- permTest_df %>%
  pull(F_stat) %>%
  map_lgl(~.x >= Fval) %>%
  mean()

p_value %>% knitr::kable(
    caption="\\label{tab:pval} The $p$-value of the permutation test.",
    col.names="P-value",
  ) %>%
  kable_styling(latex_options=c("hold_position"))
```

Table 9: The p-value of the permutation test.

P-value
0.04004

The resulting F-statistics are shown in Figure 7.

The associated p-value can be seen in table 9 and can be seen to be below 0.05. We therefore reject the null-hypothesis at significance level of $\alpha = 0.05$, which agrees with the result obtained from the F-test conducted earlier.

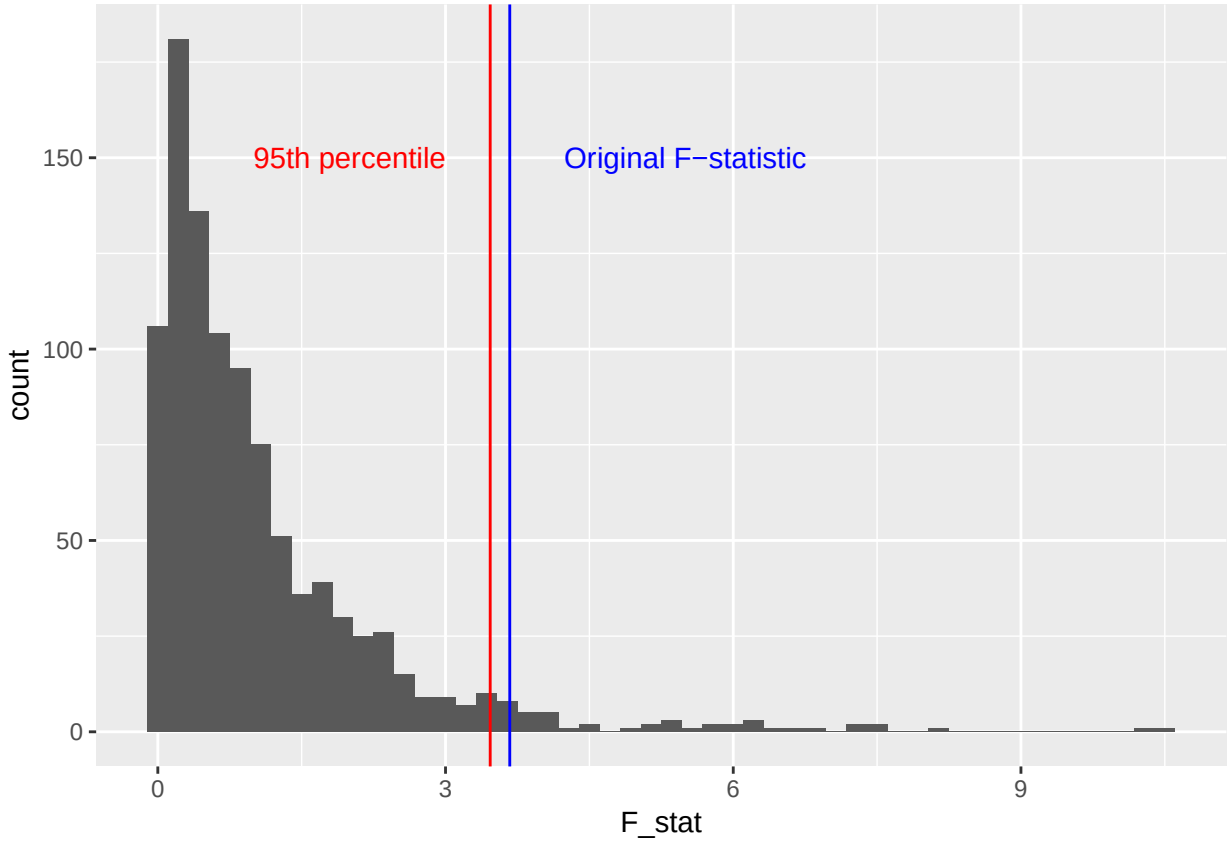

Figure 7: Histogram showing the F-statistic obtained from the different permutations. The red line is the 95th percentile while the blue represents the value of the F-statistic computed for the original data.

The permutation test is a more robust test, as it requires no model assumptions. It is not dependent on the residuals ϵ_{ij} being normally distributed for instance. This strengthens our belief in the distributions being different, since the conclusion is made with less assumptions.

Problem C: The EM-algorithm and bootstrapping

Description

Let $\boldsymbol{x} := [x_1, ..., x_n]$ and $\boldsymbol{y} := [y_1, ..., y_n]$ be two collections of independent random variables from two independent exponential distributions

$$x_1, ..., x_n \sim \exp(\lambda_0)$$
$$y_1, ..., y_n \sim \exp(\lambda_1).$$

Denote the probability density functions of these distributions as $f_x(x|\lambda_0)$ and $f_y(y|\lambda_1)$ respectively. Now assume that we do not observe $\boldsymbol{x}$ and $\boldsymbol{y}$ directly, but rather $\boldsymbol{z} := [z_1, ..., z_n]$ and $\boldsymbol{u} := [u_1, ..., u_n]$, which are constructed as

$$z_i = \max(x_i, y_i)$$
$$u_i = I(x_i \leq y_i)$$

for $i = 1, ..., n$ and where I is the indicator function.

Complete data log likelihood function

The likelihood function for the complete data $(\boldsymbol{x}, \boldsymbol{y})$ is

$$\begin{aligned}
&\mathcal{L}(\lambda_0, \lambda_1; \boldsymbol{x}, \boldsymbol{y}) \\
&= \prod_{i=1}^{n} f_x(x_i|\lambda_0) \cdot \prod_{j=1}^{n} f_y(y_j|\lambda_1) \\
&= \prod_{i=1}^{n} \lambda_0 e^{-\lambda_0 x_i} \cdot \prod_{j=1}^{n} \lambda_1 e^{-\lambda_1 y_i} \\
&= (\lambda_0 \lambda_1)^n \cdot \exp\left\{-\lambda_0 \sum_{i=1}^{n} x_i\right\} \cdot \exp\left\{-\lambda_1 \sum_{j=1}^{n} y_j\right\}
\end{aligned}$$

Thus the log likelihood becomes

$$\begin{aligned}
\ell(\lambda_0, \lambda_1; \boldsymbol{x}, \boldsymbol{y}) &:= \ln \mathcal{L}(\lambda_0, \lambda_1; \boldsymbol{x}, \boldsymbol{y}) \\
&= n(\ln \lambda_0 + \ln \lambda_1) - \lambda_0 \sum_{i=1}^{n} x_i - \lambda_1 \sum_{j=1}^{n} y_j. \qquad (1)
\end{aligned}$$

We can now calculate the expected value of the log likelihood given $\boldsymbol{z}$, $\boldsymbol{u}$, $\lambda_0^{(t)}$, $\lambda_1^{(t)}$. First we calculate the conditional expectation of the first term of (1) consisting of constants:

$$\mathrm{E}\left[n(\ln \lambda_0 + \ln \lambda_1) \mid \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] = n(\ln \lambda_0 + \ln \lambda_1). \qquad (2)$$

For the conditional expectation of x_i, which is not directly observed, notice that

$$\begin{aligned}
\mathrm{E}\left[x_i \mid \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] &= \begin{cases} z_i & \text{if } u_i = 1 \\ \mathrm{E}\left[x_i \mid x_i < z_i, \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] & \text{otherwise.} \end{cases} \\
&= u_i z_i + (1 - u_i)\mathrm{E}\left[x_i \mid x_i < z_i, \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right]
\end{aligned}$$

Here we have rewritten the cases by using u_i. For the $u_i = 0$ case we can calculate the expected value by integration:

$$\begin{aligned}
&\mathrm{E}\left[x_i \;\middle|\; x_i < z_i, \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] \\
&= \frac{\int_{x=0}^{x=z_i} x f_x(x \mid \lambda_0^{(t)}) \mathrm{d}x}{\int_{x=0}^{x=z_i} f_x(x \mid \lambda_0^{(t)}) \mathrm{d}x} \\
&= \frac{\int_{x=0}^{x=z_i} \lambda_0 x e^{-\lambda_0 x} \mathrm{d}x}{\int_{x=0}^{x=z_i} \lambda_0 e^{-\lambda_0 x} \mathrm{d}x} \\
&= \frac{1 - (\lambda_0^{(t)} z_i + 1) e^{-\lambda_0^{(t)} z_i}}{\lambda_0^{(t)}} \cdot \frac{1}{1 - e^{-\lambda_0^{(t)} z_i}} \\
&= \frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)} z_i} - 1}
\end{aligned}$$

Using u_i we can now write the conditional expectation of the second term in (1) as

$$E\left[\lambda_0 \sum_{i=1}^{n} x_i \;\middle|\; \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] = \lambda_0 \sum_{i=1}^{n} \left[u_i z_i + (1 - u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)} z_i} - 1}\right)\right] \tag{3}$$

Using the symmetry of the problem we can find the conditional expectation of the third term in (1) as well

$$E\left[\lambda_1 \sum_{j=1}^{n} y_j \;\middle|\; \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] = \lambda_1 \sum_{j=1}^{n} \left[(1 - u_i) z_i + u_i \left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{e^{\lambda_1^{(t)} z_i} - 1}\right)\right] \tag{4}$$

We can now combine equations (2), (3), and (4) in order to get the conditional expectation of the log likelihood

$$\begin{aligned}
Q(\lambda_0, \lambda_1) := E\left[\ell(\lambda_0, \lambda_1; \boldsymbol{x}, \boldsymbol{y}) \;\middle|\; \boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] &= n(\ln \lambda_0 + \ln \lambda_1) \\
&\quad - \lambda_0 \sum_{i=1}^{n} \left[u_i z_i + (1 - u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)} z_i} - 1}\right)\right] \\
&\quad - \lambda_1 \sum_{j=1}^{n} \left[(1 - u_i) z_i + u_i \left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{e^{\lambda_1^{(t)} z_i} - 1}\right)\right] \\
&:= n(\ln \lambda_0 + \ln \lambda_1) - \lambda_0 S_0 - \lambda_1 S_1
\end{aligned}$$

Here we have denoted this conditional expectation of the log likelihood as $Q(\lambda_0, \lambda_1)$, and the first and second sum in the equation as S_0 and S_1, respectively. This will simplify the notation going forward.

EM algorithm

In order to maximize $Q(\lambda_0, \lambda_1)$ with respect to λ_0 and λ_1 we must solve

$$\begin{aligned}
\nabla Q(\lambda_0, \lambda_1) &= [0, 0], \\
\nabla &:= \left[\hat{\mathbf{i}} \frac{\partial}{\partial \lambda_0}, \; \hat{\mathbf{j}} \frac{\partial}{\partial \lambda_1}\right].
\end{aligned}$$

This equation can be explicitly solved for $\boldsymbol{\lambda} := [\lambda_0, \lambda_1]$ as

$$\begin{aligned} \frac{n}{\lambda_0} = S_0 &\implies \lambda_0 = \frac{n}{S_0}, \\ \frac{n}{\lambda_1} = S_1 &\implies \lambda_1 = \frac{n}{S_1}. \end{aligned} \tag{5}$$

We start by importing the provided data set containing $\boldsymbol{z}$ and $\boldsymbol{u}$:

```
library(tidyverse)
data ← tibble(
  u = scan("data/u.txt", double()) %>% as.logical(),
  z = scan("data/z.txt", double())
)
n ← length(data$z)
```

We can now implement the EM algorithm, using equation (5) in the M-step. We will use

$$||\boldsymbol{\lambda}^{(t)} - \boldsymbol{\lambda}^{(t-1)}||_1 = \max\left(|\lambda_0^{(t)} - \lambda_0^{(t-1)}|,\ |\lambda_1^{(t)} - \lambda_1^{(t-1)}|\right) < \texttt{TOL}$$

as the stopping criterion.

```
expectation_maximization ← function (data, lambda = c(1, 1), tol = 1e-4) {
  # Problem parameters
  z ← data$z
  u ← data$u
  n ← length(u)

  # Data structure used to save each lambda result for every iteration
  lambdas ← lambda

  # Stopping criterion initialized at infinity
  delta ← Inf
  while (delta > tol) {
    # Maximum likelihood estimates
    sum_0 ← sum(z[u]) + sum(1 / lambda[1] - z[!u] / expm1(lambda[1] * z[!u]))
    sum_1 ← sum(z[!u]) + sum(1 / lambda[2] - z[u] / expm1(lambda[2] * z[u]))
    new_lambda ← c(n / sum_0, n / sum_1)

    # Calculate convergence criterion
    delta ← max(abs(lambda - new_lambda))

    # Save results before next iteration
    lambda ← new_lambda
    lambdas ← c(lambdas, lambda)
  }

  lambdas ← tibble(
    iteration = 0:(length(lambdas) / 2 - 1),
    lambda_0 = lambdas[c(TRUE, FALSE)],
    lambda_1 = lambdas[c(FALSE, TRUE)]
  )

  final_lambda ← tibble(
    parameter = c("lambda_0", "lambda_1"),
```

```
    value = lambda
  )
  return(list(lambdas = lambdas, final_lambda = final_lambda))
}
```

We can now invoke the EM-algorithm with `TOL` $= 10^{-4}$ and $\boldsymbol{\lambda}^{(t)} = [1, 1]$, using the provided data:

```
orig_result ← expectation_maximization(data = data)
print(orig_result$final_lambda$value)
```

```
## [1] 3.465735 9.353185
```

The EM-estimator for the rate parameters is $\hat{\boldsymbol{\lambda}}_{\text{EM}} \approx [3.466, 9.353]$. We can plot the convergence of the EM-algorithm:

```
library(scales)
orig_result$lambdas %>%
  gather(lambda, value, c("lambda_0", "lambda_1")) %>%
  ggplot(aes(x = iteration)) +
  geom_line(aes(y = value, col = lambda)) +
  geom_hline(
    data = orig_result$final_lambda,
    aes(yintercept = value, col = parameter, linetype = "Converged value"),
    alpha = 0.5
  ) +
  geom_point(
    data = tibble(x = 0, y = 1),
    aes(x = x, y = y, shape = "Initial value")
  ) +
  scale_x_continuous(breaks=0:dim(orig_result$lambdas)[1]) +
  scale_linetype_manual(name = "", values = c(2, 2)) +
  ylim(0, NA)
```

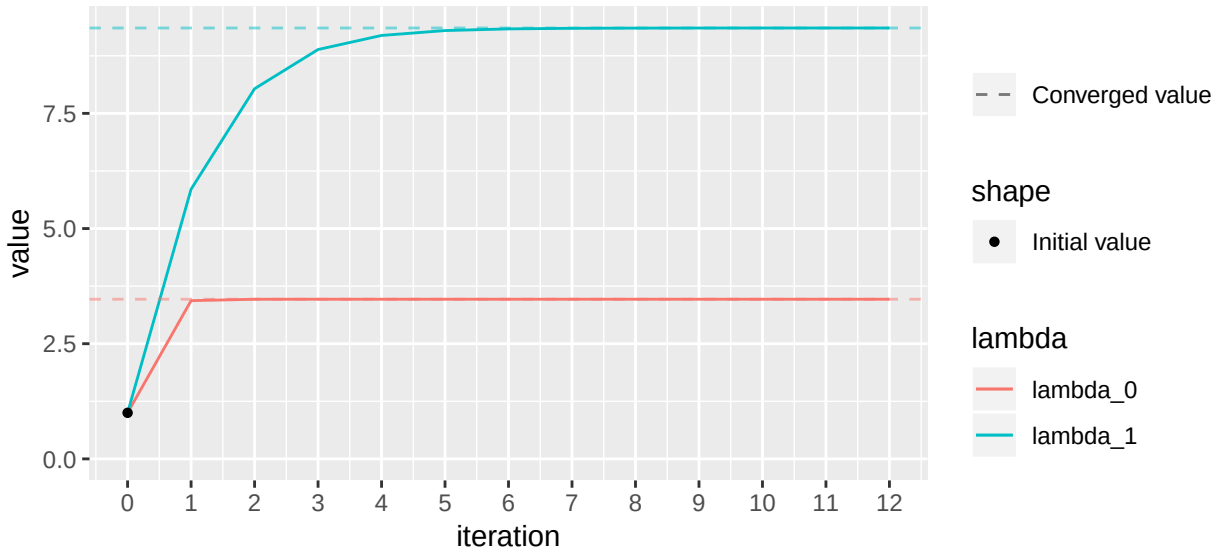

Figure 8: Convergence of the EM-algorithm. λ_0 is shown in red, while λ_1 in shown in blue. The converged values are plotted as horizontal, dashed lines with their respective colors. The initial value for the EM-algorithm is iteration 0 and marked with a black dot.

As we can see in Figure 8 the EM-algorithm converges quite rapidly.

Bootstrapping

We now want to apply bootstrapping in order to find estimates for the standard deviation, bias, and correlation of $\hat{\boldsymbol{\lambda}}_{\mathbf{0}}$ and $\hat{\boldsymbol{\lambda}}_{\mathbf{1}}$. The following pseudocode describes the actual implementation:

- Assume that we have an initial data set $\boldsymbol{z}$ and $\boldsymbol{u}$, defined as before.
- Construct the pairwise observation set $\Omega = \{\omega_1, \omega_2, ..., \omega_n\} := \{(z_1, u_1), (z_2, u_2), ..., (z_n, u_n)\}$. These observation pairs are now independet and identically distributed. Denote this distribution as F.
- Construct the empirical distribution $\hat{F}$ which assigns mass $\frac{1}{n}$ to each observation pair in Ω. This distribution is supposed to approximate F.
- Draw B *bootstrap samples* from $\hat{F}$. Each bootstrap sample is of size n, and is denoted $\Omega^{(b)} = \{\omega_1^{(b)}, ..., \omega_n^{(b)}\} = \{(z_1^{(b)}, u_1^{(b)}), ..., (z_n^{(b)}, u_n^{(b)})\}$ for $b = 1, ..., B$. In practice, this means sampling n elements from Ω *with* replacement.
- For $b = 1, 2, ..., B$:
  - Estimate $\lambda_0^{(b)}$ and $\lambda_1^{(b)}$ using the EM-algorithm as derived above, but this time use the bootstrap replicate $\Omega^{(b)}$ instead of the original data set Ω. These are called bootstrap replicates of λ_0 and λ_1, respectively.

The important part here is that we draw observation *pairs*, as z_i and u_j are dependent for $i = j$ and independent for $i \neq j$. It is this algorithm we now implement, using $B = 2000$:

```
library(rsample)  # For bootstraps() function
library(magrittr)  # For assignment pipe
set.seed(0)
B ← 2000
bs_data ← bootstraps(data, times = B)
```

```
bs_results ⟵ tibble(lambda_0 = double(), lambda_1 = double())

for (bootstrap in bs_data$splits) {
  expectation ⟵ expectation_maximization(data = analysis(bootstrap))
  lambda ⟵ expectation$final_lambda$value
  bs_results %<>% add_row(lambda_0 = lambda[1], lambda_1 = lambda[2])
}
```

The idea is now that the bootstrap replicates $[(\hat{\lambda_0}^{(1)}, \hat{\lambda_1}^{(1)}), ..., (\hat{\lambda_0}^{(B)}, \hat{\lambda_1}^{(B)})]$ can be considered to be approximate samples of the distribution of $\boldsymbol{\lambda}_{\text{EM}}$. We can therefore estimate the standard deviation, bias, and correlation of $\boldsymbol{\lambda}_{\text{EM}}$ from the bootstrap replicates in order to make inferences from the original EM-result.

```
library(corrr)
library(kableExtra)
bs_stats ⟵ list(
  cor = cor(bs_results$lambda_0, bs_results$lambda_1),
  std = bs_results %>% map(sd) %>% unlist(),
  mean = bs_results %>% colMeans()
)
bs_stats$bias ⟵ bs_stats$mean - orig_result$final_lambda$value
bs_stats[-1] %>%
  as_tibble() %>%
  t() %>%
  kable(
    escape = FALSE,
    col.names = c("$\\hat{\\lambda_0}$", "$\\hat{\\lambda_1}$"),
    caption = "\\label{tab:em_results}Estimated EM-estimator statistics from bootstrap results.",
    format = "pandoc"
  )
```

Table 10: Estimated EM-estimator statistics from bootstrap results.

	$\hat{\lambda_0}$	$\hat{\lambda_1}$
std	0.2507487	0.8082275
mean	3.4796042	9.4527979
bias	0.0138693	0.0996133

We can see from table 10 that the bias is relatively small. We also have $\hat{\text{SD}}(\hat{\lambda_0}) < \hat{\text{SD}}(\hat{\lambda_1})$, which may be explained by the fact that $\hat{\lambda_0} < \hat{\lambda_1}$ in combination with the fact that the variance and rate of an exponential distribution are equal. Additionally:

```
print(bs_stats$cor)
```

```
## [1] 0.001474608
```

We have $\hat{\text{Corr}}(\hat{\lambda_0}, \hat{\lambda_1}) \approx 0.0015$ which can be considered very close to 0. This makes sense, as they should be independent.

These results can be visualized in a histogram:

```
binwidth ⟵ 0.07
bs_results %>%
  gather(lambda, value, c("lambda_0", "lambda_1")) %>%
  ggplot(aes(x = value, fill = lambda)) +
```

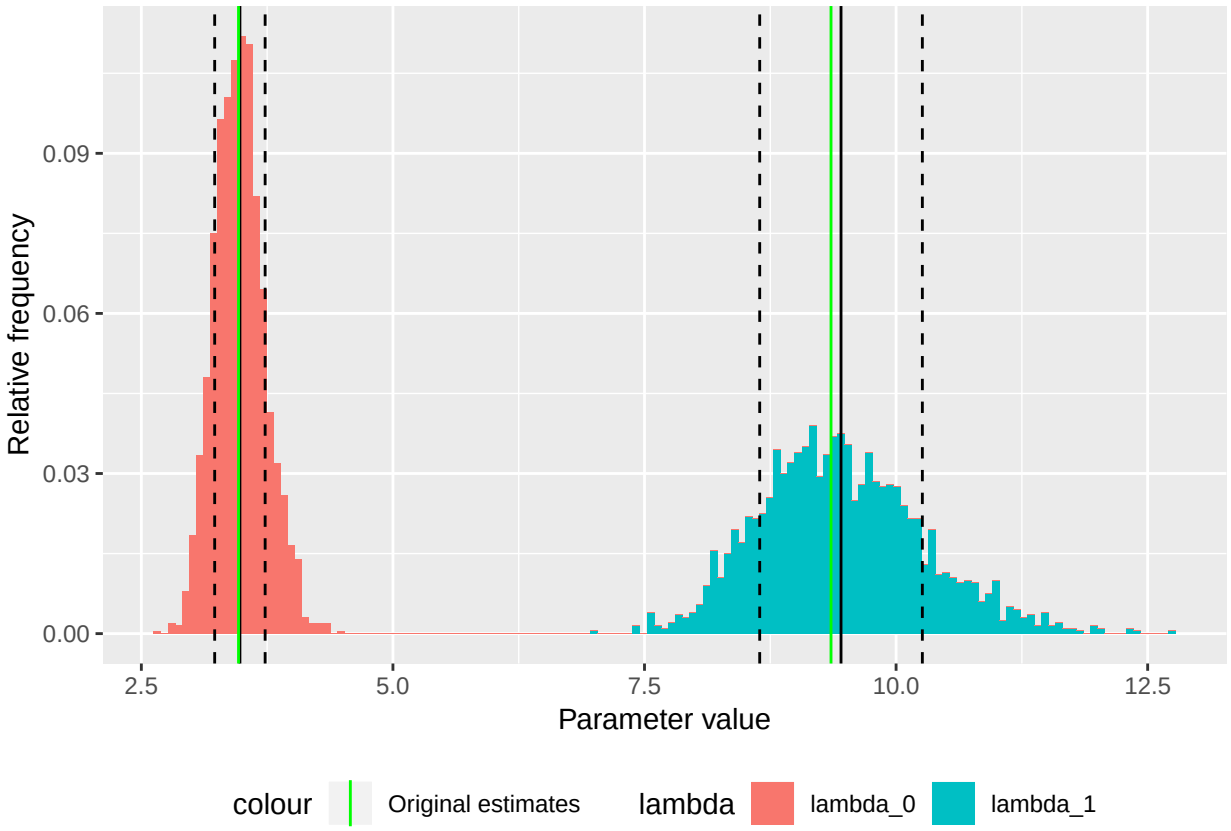

Figure 9: Histogram of the bootstrap replicates of $\hat{\lambda_0}$ (red) and $\hat{\lambda_1}$ (blue). The mean of the bootstrap replicates are show as solid, black, vertical lines. Mean $\pm$ Std of the bootstrap replicates are shown in dashed, black, vertical lines. Finally, the original EM-estimators are shown as green vertical lines.

```
  geom_histogram(aes(y = binwidth * ..density..), binwidth = binwidth) +
  geom_vline(xintercept = bs_stats$mean) +
  geom_vline(xintercept = bs_stats$mean + bs_stats$std, linetype = "dashed") +
  geom_vline(xintercept = bs_stats$mean - bs_stats$std, linetype = "dashed") +
  geom_vline(
    data = orig_result$final_lambda,
    aes(xintercept = value, color = "Original estimates")
  ) +
  scale_color_manual(values = c("green")) +
  ylab("Relative frequency") +
  xlab("Parameter value") +
  theme(legend.position = "bottom")
```

Figure 9 shows the greater estimated variance and bias in $\hat{\lambda_1}$ compared to $\hat{\lambda_0}$. It is possible to construct a *bias corrected* estimate, $\hat{\lambda_1}_c$, for λ_1 on the form

$$\hat{\lambda_1}_c := \hat{\lambda_1} - \widehat{\text{Bias}}(\hat{\lambda}),$$

and likewise for λ_0. This may indeed decrease the bias of our estimators, but will not necessarily yield an overall better estimator as

$$\text{Var}(\hat{\lambda_i}_c) \geq \text{Var}(\hat{\lambda_i}), \quad i = 1, 2$$

due to the estimator consisting of additional terms. It is therefore preferable to *not* apply bias correction if the bias is small. As we consider our bias to be within acceptable bounds, we will not opt for bias correction.

Analytical formula for $f_{Z_i,U_i}(z_i, u_i \mid \lambda_0, \lambda_1)$

We now want to find an analytical expression for $f_{Z_i,U_i}(z_i, u_i \mid \lambda_0, \lambda_1)$. Start by finding $F_Z(z_i \mid u_i = 1)$:

$$\begin{aligned}
F_Z(z_i \mid u_i = 1) &= P(X_i \leq z_i, Y_i \leq X_i) \\
&= \int_0^{z_i} \int_0^{x} f_X(x \mid \lambda_0) f_Y(y \mid \lambda_1) \,\mathrm{d}y \,\mathrm{d}x \\
&= \int_0^{z_i} \lambda_0 e^{-\lambda_0 x} \int_0^{x} \lambda_1 e^{-\lambda_1 y} \,\mathrm{d}y \,\mathrm{d}x \\
&= \int_0^{z_i} \lambda_0 e^{-\lambda_0 x} \left(1 - e^{-\lambda_1 x}\right) \,\mathrm{d}x \\
&= \frac{\lambda_0}{\lambda_0 + \lambda_1} \left(e^{-(\lambda_0+\lambda_1) z_i} - 1\right) - e^{-\lambda_0 z_i} + 1
\end{aligned}$$

We can now find $f_Z(z_i \mid u_i = 1)$ by differentiating:

$$\begin{aligned}
f_Z(z_i \mid u_i = 1) &= \frac{\mathrm{d}F_Z(z_i \mid u_i = 1)}{\mathrm{d}z_i} \\
&= -\lambda_0 e^{-(\lambda_0+\lambda_1) z_i} + \lambda_0 e^{-\lambda_0 z_i} \\
&= \lambda_0 e^{-\lambda_0 z_i} \left(1 - e^{-\lambda_1 z_i}\right)
\end{aligned}$$

Again, by the symmetry of the problem, we can find $f_Z(z_i \mid u_i = 0)$:

$$f_Z(z_i \mid u_i = 1) = \lambda_1 e^{-\lambda_1 z_i} \left(1 - e^{-\lambda_0 z_i}\right)$$

The joint distribution for $\boldsymbol{z}$ and $\boldsymbol{u}$ and inversely the likelihood of λ_0 and λ_1, depending on what you consider to be the variables, can now be found:

$$\begin{aligned}
f_{Z,U}(\boldsymbol{z}, \boldsymbol{u} \mid \lambda_0, \lambda_1) &= \prod_{i:u_i=0} f_Z(z_i \mid u_i = 0) \prod_{i:u_i=1} f_Z(z_i \mid u_i = 1) \\
&= \prod_{i=1}^{n} \left(u_i \lambda_0 e^{-\lambda_0 z_i} \left(1 - e^{-\lambda_1 z_i}\right) + (u_i - 1) \lambda_1 e^{-\lambda_1 z_i} \left(1 - e^{-\lambda_0 z_i}\right)\right) \\
&= \mathcal{L}(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u})
\end{aligned}$$

We can now find the log likelihood:

$$\begin{aligned}
\ell(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u}) &:= \ln \mathcal{L}(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u}) \\
&= \sum_{i:u_i=0} \left(\ln(\lambda_1) - \lambda_1 z_i + \ln\left(1 - e^{-\lambda_0 z_i}\right)\right) + \sum_{i:u_i=1} \left(\ln(\lambda_0) - \lambda_0 z_i + \ln\left(1 - e^{-\lambda_1 z_i}\right)\right) \\
&= n_1 \ln(\lambda_0) + n_0 \ln(\lambda_1) + \sum_{i:u_i=0} \left(\ln\left(1 - e^{-\lambda_0 z_i}\right) - \lambda_1 z_i\right) + \sum_{i:u_i=1} \left(\ln\left(1 - e^{-\lambda_1 z_i}\right) - \lambda_0 z_i\right)
\end{aligned}$$

Where we have defined:

$$n_0 := \sum_{i=1}^{n} (1 - u_i), \qquad n_1 := \sum_{i=1}^{n} u_i$$

In order to find the maximum likelihood estimators $\hat{\lambda_0}$ and $\hat{\lambda_1}$, we derive the *score vector*:

$$\begin{aligned}\boldsymbol{s}(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u}) : &= \begin{bmatrix} \frac{\partial \ell(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u})}{\partial \lambda_0} \\ \frac{\partial \ell(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u})}{\partial \lambda_1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{n_1}{\lambda_0} + \sum_{i:u_i=0} \frac{z_i}{e^{\lambda_0 z_i} - 1} - \sum_{i:u_i=1} z_i \\ \frac{n_0}{\lambda_1} + \sum_{i:u_i=1} \frac{z_i}{e^{\lambda_1 z_i} - 1} - \sum_{i:u_i=0} z_i \end{bmatrix}\end{aligned}$$

We can now solve the optimization problem

$$\hat{\lambda_0}, \hat{\lambda_1} = \underset{\lambda_0, \lambda_1}{\operatorname{argmax}} \, \mathcal{L}(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u}),$$

by solving the dual problem of

$$\boldsymbol{s}(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u}) = \boldsymbol{0},$$

with respect to λ_0 and λ_1.

Now, notice that $s_1 = \frac{\mathrm{d}\ell}{\mathrm{d}\lambda_0}$ is a function of only λ_0, not λ_1, and vice versa for s_2. The Hessian of $\ell(\lambda_0, \lambda_1; \boldsymbol{z}, \boldsymbol{u})$ is therefore diagonal with the following diagonal elements:

$$\begin{aligned}\frac{\partial^2 \ell}{\partial {\lambda_0}^2} &= -\frac{n_1}{{\lambda_0}^2} - \sum_{i:u_i=0} \frac{{z_i}^2 e^{\lambda_0 z_i}}{(e^{\lambda_0 z_i} - 1)^2} < 0 \\ \frac{\partial^2 \ell}{\partial {\lambda_1}^2} &= -\frac{n_0}{{\lambda_1}^2} - \sum_{i:u_i=1} \frac{{z_i}^2 e^{\lambda_1 z_i}}{(e^{\lambda_1 z_i} - 1)^2} < 0\end{aligned}$$

The Hessian is therefore negative definite. In conclusion, there is only one maxima, which is the global maxima. We can therefore be sure that if we solve the optimization problem, we have in fact found maximum likelihood estimators $\hat{\lambda_0}$ and $\hat{\lambda_1}$.

```
score ← function (lambda, data, index) {
  #' Evaluate the log likelihood score function for a given lambda
  #' lambda = double value type for lambda_{0,1}
  #' data = Data frame with `z` and `u` columns
  #' index = Either 0 or 1, indicating if lambda_0 or lambda_1 is of interest

  if (index == 0) {
    indeces ← data$u
  } else {
    indeces ← !data$u
  }

  return(
    sum(indeces) / lambda
    + sum(data$z[!indeces] / expm1(lambda * data$z[!indeces]))
    - sum(data$z[indeces])
  )
}
```

```
lambda_mle ← c(
  lambda_0 = uniroot(f = score, interval = c(1e-3, 100), data = data, index = 0)$root,
  lambda_1 = uniroot(f = score, interval = c(1e-3, 100), data = data, index = 1)$root
)
print(lambda_mle)
```

```
## lambda_0 lambda_1
## 3.465737 9.353232
```

The result of the log likelihood estimation is $\boldsymbol{\lambda}_{\text{MLE}} \approx [3.466, 9.353]$. Comparing this to the result of the EM-algorithm, $\hat{\boldsymbol{\lambda}}_{\text{EM}} \approx [3.466, 9.353]$, they yield the same result to three decimal digits. The exact difference, $\hat{\boldsymbol{\lambda}}_{\text{EM}} - \boldsymbol{\lambda}_{\text{MLE}}$, is

```
orig_result$final_lambda$value - lambda_mle
```

```
##      lambda_0      lambda_1
## -1.821222e-06 -4.715742e-05
```

So both values are equal to at least 5 decimals, which strengthen our belief in the correctness of both implementations.

Some of the advantages of optimizing the likelihood directly compared to the EM algorithm are:

- Only one optimization is necessary, compared to the EM algorithm where you might have to optimize differently parametrized functions iteratively. However, the EM-algorithm is rarely applied when explicit expressions for the minimizers can not be found.
- The Fisher information matrix can be calculated in order to infer the degree of information encoded in the maximum likelihood.
- You optimize the function of interest directly, instead of only approximating it.