NTNU
Kunnskap for en bedre verden

TMA4300 - Computer Intensive Statistical Methods

Project 3

Author:
Gard Gravdal & Marcus Hjørund

Date

Table of Contents

List of Figures

List of Tables

1 Introduction

2 Problem 1: Bootstrapping a GLM

In this problem we have the independent random variables $Y_1, \dots, Y_n$, where each $Y_i \sim \text{bin}(m_i, p_i)$ and where $\ln \frac{p_i}{1-p_i} = \beta_0 + \beta_1 x_i$ for $i = 1, \dots, n$. m_i is the number of trials, y_i is the number of successes, and p_i is the probability of success. This is a generalized linear model which can be fitted by maximum likelihood in R. We use the `glm()`-function, with argument `formula = cbind(y,m-y) x` and `family = binomial` to produce a maximum likelihood estimate (MLE) $\hat{\boldsymbol{\beta}}$ of $\boldsymbol{\beta}$. We know that $\text{Var}\hat{\boldsymbol{\beta}}$ in general is asymptotically equal to the inverse of the Fisher information matrix $F(\boldsymbol{\beta})$, and an approximate estimate of $\text{Var}\hat{\boldsymbol{\beta}}$ is given as $(F(\hat{\boldsymbol{\beta}}))^{-1}$. This is extracted in R using the `vcov()`-function. The data we use in this problem can be loaded using the following command in R, `load(file=url("https://www.math.ntnu.no/emner/TMA4300/2024v/data.Rdata"))`, in the data frame `data`.

2.1 a)

In this problem we will write a function that takes in the fitted model, and simulates $B = 10000$ bootstrap samples by resampling the observation triplets (m_i, y_i, x_i) with replacements. We then refit the above model to each bootstrap sample to obtain bootstrap replicates $\hat{\boldsymbol{\beta}}$, $b = 1, \dots, B$ of $\hat{\boldsymbol{\beta}}$.

```r
load(file=url("https://www.math.ntnu.no/emner/TMA4300/2024v/data.Rdata"))
n <- length(data$y)
mod <- glm(cbind(y, m-y)~x, family = binomial, data = data)
#Writing a bootstrap function
beta_star <- function(mod){
  return(coef(mod))
}
bootstrap <- function(model, B, n){
  beta_star <- t(replicate(B, {
  resample <- sample(1:n,replace = TRUE)
  beta_star(glm(cbind(y,m-y)~x, family = binomial, data = model$data[resample,]))}))
  return(beta_star)
}
bs_replicates <- bootstrap(mod, 10000,n)
library(matrixStats)
bs_estimate <- colMeans(bs_replicates)
beta_hat <- coef(mod)
```

In the code, `bs_estimate` is the mean of the bootstrap samples and is given as `bs_estimate` $=$
ok $(-0.7273831, 0.107467)^T$ for the intercept and slope. `beta_hat` is the MLE from the `glm()` function,
and is given as `beta_hat` $= (-0.6781067, 0.1009905)^T$ for the intercept and slope.

2.2 b)

Having calculated the bootstrap estimates, we want to see how the estimated covariance matrix from bootstrapping compares to the covariance matrix we obtain from the MLE. We use the `vcov()`-function for the MLE $\hat{\boldsymbol{\beta}}$ and the `cov()`-function for the bootstrap estimates $\hat{\boldsymbol{\beta}}^{b*}$.

Estimated covariance matrices				
	Bootstrap estimate		glm()estimate	
	Intercept	Slope	Intercept	Slope
Intercept	0.59329856	-0.07928622	0.19640741	-0.28022675
Slope	-0.07928622	0.01129421	-0.02802267	0.005049491

ok

Table 1: Estimated covariance matrices for $\boldsymbol{\beta}$

From Table 1, we see the estimated covariance matrices, using the MLE estimator and using bootstrap estimation. We see that the MLE has lower variance and covariance than the bootstrap estimates, almost by a factor of three.

2.3 c)

In this section we will estimate the bias of the MLEs of the intercept and slope parameters. In order to estimate the bias of the intercept and slope parameters we use the fact that an unbiased estimator of the expectation of the estimator $\mathrm{E}[\hat{\boldsymbol{\beta}}]$ is given by $\widehat{\mathrm{E}[\hat{\boldsymbol{\beta}}]} = \frac{1}{B}\sum_{b=1}^{B} \hat{\boldsymbol{\beta}}^{b*}$, for $b = 1, \dots, B$, with $\hat{\boldsymbol{\beta}}^{b*}$ is the b'th boostrap replicate of $\boldsymbol{\beta} = [\beta_0, \beta_1]^T$. Thus we can estimate the bias, defined by

$$\mathrm{Bias}[\hat{\boldsymbol{\beta}}] = \mathrm{E}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta}$$

as

$$\widehat{\mathrm{Bias}[\hat{\boldsymbol{\beta}}]} = \widehat{\mathrm{E}[\hat{\boldsymbol{\beta}}]} - \hat{\boldsymbol{\beta}}$$

where $\hat{\boldsymbol{\beta}}$ is the MLE, and $\widehat{\mathrm{E}[\hat{\boldsymbol{\beta}}]}$ is the bootstrap estimate. One can then produce bias-corrected estimates as

$$\hat{\boldsymbol{\beta}}_C = \hat{\boldsymbol{\beta}} - \widehat{\mathrm{Bias}[\hat{\boldsymbol{\beta}}]}.$$

We do this in R as follows:

```
bias_hat <- bs_estimate - beta_hat
```

Using the above procedure, we estimate the bias of the coefficients as $\widehat{\mathrm{Bias}[\beta_0]} \approx -0.049276408$ and $\widehat{\mathrm{Bias}[\beta_1]} \approx 0.006476799$.

In order to test whether the bias is significant, we use the Wald-test, with the hypothesis H_0 : $\mathrm{Bias}[\hat{\boldsymbol{\beta}}] = 0$ against H_1 : $\mathrm{Bias}[\hat{\boldsymbol{\beta}}] \neq 0$, using the estimated covariance matrix from `vcov()`. This can be done in R in the following way:

```
library(aod)
MLE_covariance <- vcov(mod)
wald_result <- wald.test(Sigma = MLE_covariance, b = beta_hat-bs_estimate,Terms = 1:2)
```

which yields a χ^2_2 variable. The Wald test returns $\chi^2 = 0.0013$ and $P(> \chi^2) = 0.99$. Based on the test we do not reject H_0 and we can conclude that the bias is not significant. Thus, adding a bias correction term will increase variance of our estimator, while not providing significant information.

Why do you use a Wald test? to test both parameters at the same time?
I'm not so sure but I think one of the assumptions of the Wald test is joint normality of all the parameters, and I'm not sure whether we can make that assumption here, due to the small n.
If you use a one sample t-test, the bias turns out to be significant.

2.4 d)

In this section we will use the percentile method to compute an approximate 95% confidence interval for each model parameter from the bootstrap replicates. We will compare this to the confidence intervals obtained based on the profile likelihoods of the parameters. In R we do this using the `confint()` function. The implementation is given in the code below.

```
library(matrixStats)
confint_intercept_bs <- colQuantiles(bs_replicates, probs = c(0.025, 0.975))[1,]
confint_slope_bs <- colQuantiles(bs_replicates, probs = c(0.025, 0.975))[2,]
confint_intercept_glm <- confint(mod)[1,]
confint_slope_glm <- confint(mod)[2,]
```

The results can be seen in Figure 1. We see that both the confidence intervals cover the predictions made by each other, but that the bootstrap confidence intervals are larger.

	2.5%	97.5%
Intercept bootstrap	-2.0161396	0.8307121
Intercept MLE	-1.5718941	0.1779204
Slope bootstrap	-0.1030725	0.3002614
Slope MLE	-0.0367474	0.2434345

ok

Figure 1: Bootstrap vs. MLE approximate confidence intervals

2.5 e)

In this section we will do the same procedures as previously, except now we use parametric bootstrapping. In parametric bootstrapping we assume we know the underlying probability distribution $F(\boldsymbol{x};\boldsymbol{\theta})$, but we have to estimate the parameter of interest $\boldsymbol{\theta}$ and simulate realizations $(x_1,\dots,x_n) \sim F(\boldsymbol{x};\hat{\boldsymbol{\theta}})$. We estimate $\boldsymbol{\theta}$ by for example MLE, giving $\hat{\boldsymbol{\theta}}$. For each iteration of the bootstrap algorithm, we then compute $\hat{\boldsymbol{\theta}}^{b*}$ for $b = 1,\dots,B$ by same estimation, in this case the MLE. In our case we have that $Y_i \sim \text{bin}(m_i, p_i)$ and $\ln\frac{p_i}{1-p_i} = \beta_0 + \beta_1 x_i$ for $i = 1,\dots,x_n$. We want to simulate Y_i, and we thus estimate $\boldsymbol{p}$ as $\hat{\boldsymbol{p}} = \frac{\boldsymbol{y}}{\boldsymbol{m}}$, which is the MLE. We then generate $\boldsymbol{y}_1^{1*},\dots,\boldsymbol{y}_n^{B*} \sim \text{bin}(m_i,\hat{p}_i)$ for $i = 1,\dots,n$. We then proceed as before using the `glm()` function to estimate the parameters β_0 and β_1. The R code can be seen below:

```
#parametric bootstrap
#data£y/data£m estimate for p
#y number of successes
#m number of trials
p <- data$y/data$m
data$p <- p
bootstrap_param <- function(mod, B,n) {
  data <- mod$data
  beta_star <- t(replicate(B, {
    y_star <- rbinom(n,data$m, p)
    data$y <- y_star
    beta_star(glm(cbind(y,(m-y))~x, family = binomial, data = data))
  }))
}
bs_replicates_param <- bootstrap_param(mod, 10000, n)
bs_estimate_param <- colMeans(bs_replicates_param)
bs_covariance_param <- cov(bs_replicates_param)
bs_estimate_param
beta_hat
bias_hat_param <- bs_estimate_param - beta_hat
bias_hat_param
wald.test(Sigma = MLE_covariance, b = beta_hat-bs_estimate_param, Terms = 1:2)
confint_intercept_param <- colQuantiles(bs_replicates_param, probs = c(0.025, 0.975))[1,]
confint_slope_param <- colQuantiles(bs_replicates_param, probs = c(0.025, 0.975))[2,]
```

Here we have bs_estimate_param $= (-0.6928781, 0.1034373)^T$ for the intercept and slope parameters, as the estimate for the expected value of the MLE. The estimate for $\text{Var}\hat{\boldsymbol{\beta}}$ is given in Table

Estimated covariance matrices				
	Param. bootstrap estimate		`glm()`estimate	
	Intercept	Slope	Intercept	Slope
Intercept	0.15902419	-0.023461606	0.19640741	-0.28022675
Slope	-0.023461606	0.004528715	-0.02802267	0.005049491

Table 2: Estimated covariance matrices for $\boldsymbol{\beta}$ with parametric bootstrapping

We note that the estimated covariance matrix using parametric bootstrapping shows less variance and covariance than using non-parametric bootstrapping. We get even smaller variances and covariances than using MLE.

We also perform the same test as before, the Wald test, to check whether there is any significant bias. The Wald test yields $\chi^2 = 0.0012$ and $P(> \chi^2) = 1$, and we again reject H_0. Even though we have less variance using parametric bootstrapping, adding a bias correction term will increase variance of our estimator, and not add any significant information.

	2.5%	97.5%
Intercept par. bootstrap	-1.5048451	0.0533744
Intercept MLE	-1.5718941	0.1779204
Slope par. bootstrap	-0.0210751	0.2398372
Slope MLE	-0.0367474	0.2434345

ok

Figure 2: Parametric bootstrap vs. MLE approximate confidence intervals

The 95% approximate confidence intervals from parametric bootstrapping and the MLE can be seen in Figure 2. We note that the confidence intervals are much more similar for parametric bootstrapping than for non-parametric bootstrapping.

And why is this the case?
And what would you advise, to use P or NP bootstrap, in this case?

3 Problem 2: Bootstrap confidence intervals

In this section we will be working with problems related to bootstraping confidence intervals. We suppose we have an iid sample $X_1, X_2, ..., X_n$ from the exponential distribution with scale parameter β. In other words, we have the probability density function

$$f(x;\beta) = \frac{1}{\beta}e^{-x/\beta}, \quad x \geq 0 \tag{1}$$

with expectation $\mathrm{E}[X_i] = \beta$, variance $\mathrm{Var}[X_i] = \beta^2$, for $i = 1, 2, ..., n$ and moment generating function

$$M_X(t) = \frac{1}{1-\beta t} \tag{2}$$

3.1 a)

In this section we will show that pivotal quantity $Q = \frac{2}{\beta}\sum_{i=1}^{n} X_i$ is chi-square with $2n$ degrees of freedom, and then use this fact to derive an exact $(1-\alpha)$ confidence interval for β.

We start by defining the variable $Y_i := \frac{2}{\beta}X_i \sim \mathrm{Exp}(\frac{\beta}{2\beta}) = \mathrm{Exp}(\frac{1}{2})$, where we have used closure under scaling by a positive factor for the exponential function, i.e. $X \sim \mathrm{Exp}(\frac{1}{\beta}) \Rightarrow \frac{1}{k}X \sim \mathrm{Exp}(\frac{k}{\beta})$, for constant $k > 0$. We show that Q is χ^2_{2n}-distributed by the uniqueness property of the moment generating function. The moment generating function of Q is given as

$$\begin{aligned} M_Q(t) = \mathrm{E}[e^{tQ}] &= E\left[\exp\left(2t\sum_{i=1}^{n}\frac{X_i}{\beta}\right)\right] \\ &= \mathrm{E}\left[\prod_{i=1}^{n}\exp\left(\frac{2tX_i}{\beta}\right)\right] \overset{iid}{=} \prod_{i=1}^{n}\mathrm{E}\left[\exp\left(t\frac{2X_i}{\beta}\right)\right] \\ &= \prod_{i=1}^{n}\mathrm{E}\left[\exp\left(tY_i\right)\right] = \prod_{i=1}^{n} M_{Y_i}(t) = \prod_{i=1}^{n}\frac{1}{1-2t} \\ &= \left(\frac{1}{1-2t}\right)^{n} \end{aligned}$$

The moment generating function for the chi-squared distribution with ν degrees of freedom is given as

$$M_{\chi^2_\nu}(t) = \left(\frac{1}{1-2t}\right)^{\frac{\nu}{2}} \tag{3}$$

We recognize the moment generating function of Q as the moment generating function of χ^2_ν with $\nu = 2n$ degrees of freedom. Thus, by the uniqueness property of the moment generating function, $Q \sim \chi^2_{2n}$. Using this result, we find that

$$\begin{aligned} P\left(\chi^2_{\alpha/2,2n} \le Q \le \chi^2_{1-\alpha/2,2n}\right) &= 1-\alpha \\ \Rightarrow P\left(\frac{2\sum_{i=1}^{n} X_i}{\chi^2_{1-\alpha/2,2n}} \le \beta \le \frac{2\sum_{i=1}^{n} X_i}{\chi^2_{\alpha/2,2n}}\right) &= 1-\alpha \\ \Leftrightarrow P\left(\frac{2n\bar{X}}{\chi^2_{1-\alpha/2,2n}} \le \beta \le \frac{2n\bar{X}}{\chi^2_{\alpha/2,2n}}\right) &= 1-\alpha \end{aligned}$$

ok

where $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$. This expression gives an exact $(1-\alpha)$ confidence interval for β.

3.2 b)

In this section, we explore an alternative approach by considering parametric bootstrapping to construct a bootstrap confidence interval for the parameter β. Here, we employ the percentile method to construct the confidence interval using the empirical $\alpha/2$ and $1-\alpha/2$ quantiles of the distribution of bootstrap replicates $\hat{\beta}^*$ derived from the maximum likelihood estimator $\hat{\beta}$.

The percentile method is a simple method for drawing inference about a parameter using bootstrap simulations to construct a confidence interval. It amounts to reading percentiles off the histogram of the bootstrap replicates. In this section, we will find the exact distribution of the bootstrap replicate $\hat{\beta}^*$ when it is based on samples from $F(x;\hat{\beta})$. This will allow us to find analytic formulas for the resulting confidence limits as functions of $\hat{\beta}$ for a given sample size n.

First, we must find the expression for $\hat{\beta}$, i.e. the MLE of β. The likelihood function $L(x;\beta)$ of the exponential distribution is given as

$$L(x;\beta) = \prod_{i=1}^{n} f(x;\beta) = \prod_{i=1}^{n}\frac{1}{\beta}e^{-x/\beta}$$

Further, the log-likelihood function $l(x;\beta)$ is given as

$$l(x;\beta) = \log L(x;\beta) = -n\log\beta - \frac{n\bar{x}}{\beta}$$

We differentiate with respect to β and equate with zero:

$$\frac{\partial}{\partial\beta}l(x;\beta) = 0 \Rightarrow -\frac{n}{\beta} + \frac{n\bar{x}}{\beta^2} = 0 \Rightarrow \hat{\beta} = \bar{x}$$

Having found the expression for $\hat{\beta}$, we find that $\hat{\beta}^*$ are based on bootstrap samples from

$$F(x;\hat{\beta}) = 1 - e^{-x/\hat{\beta}}$$

In other words, we resample $X_1, X_2, ..., X_n$ iid from $\text{Exp}\left(\frac{1}{\hat{\beta}}\right)$, and the bootstrap replicates are then given by

$$\hat{\beta}^* = \frac{1}{n}\sum_{i=1}^{n} X_i = \bar{X}, \quad X_i \sim \text{Exp}\left(\frac{1}{\hat{\beta}}\right)$$

To find the exact distribution of the bootstrap replicates $\hat{\beta}^*$, we make use of the moment generating function given by

$$\begin{aligned} M_{\hat{\beta}^*}(t) &= \text{E}\left[e^{t\hat{\beta}^*}\right] = \text{E}\left[\exp\left(t\frac{1}{n}\sum_{i=1}^{n} X_i\right)\right] \\ &= \text{E}\left[\prod_{i=1}^{n}\exp\left(t\frac{1}{n}X_i\right)\right] \overset{iid}{=} \prod_{i=1}^{n}\text{E}\left[\exp\left(t\frac{1}{n}X_i\right)\right] \end{aligned}$$

Defining the variable $s := \frac{1}{n}t$, and using the fact that $X_i \sim \text{Exp}\left(\frac{1}{\hat{\beta}}\right)$ for $i = 1, ..., n$, we find that

$$\begin{aligned} M_{\hat{\beta}^*}(t) &= \prod_{i=1}^{n}\text{E}\left[\exp\left(sX_i\right)\right] \\ &= \left(\frac{1}{1-\hat{\beta}s}\right)^n = \left(\frac{1}{1-\frac{\hat{\beta}}{n}t}\right)^n \end{aligned}$$

Further, if we define the variable $k := \frac{1}{2}\frac{\hat{\beta}}{n}t$, we find that

$$M_{\hat{\beta}^*}(t) = \left(\frac{1}{1-2k}\right)^n$$

which we again recognize as the moment generating function of χ^2_ν with $\nu = 2n$ degrees of freedom. We note that

$$M_{\hat{\beta}^*}(t) = \text{E}\left[\exp\left(\frac{2n}{\hat{\beta}}\hat{\beta}^* k\right)\right] = M_{\frac{2n}{\hat{\beta}}\hat{\beta}^*}(k)$$

which implies that

$$\frac{2n}{\hat{\beta}}\hat{\beta}^* \sim \chi^2_{2n} \Rightarrow \hat{\beta}^* \sim \frac{\hat{\beta}}{2n}\chi^2_{2n} \Leftrightarrow \hat{\beta}^* \sim \Gamma(n, \frac{n}{\hat{\beta}})$$

ok

where we in the final step use that if $X \sim \chi^2_\nu$ and constant $c > 0$, then $cX \sim \Gamma(\nu/2, 2c)$. Using the fact that $\hat{\beta}^* \sim \frac{\hat{\beta}}{2n}\chi^2_{2n}$, the $(1-\alpha)$ confidence interval for $\hat{\beta}^*$ is given as

$$\hat{\beta}^*_{CI} = \frac{\hat{\beta}}{2n}\left[\chi^2_{\alpha/2,2n}, \chi^2_{1-\alpha/2,2n}\right] := \left[\hat{\beta}^*_L, \hat{\beta}^*_U\right] \tag{4}$$

ok

3.3 c)

In this section we want to find the exact coverage of the parametric bootstrap percentile interval found in the previous section, in terms of the CDF and quantile function of the chi-squared distribution. In other words, we are looking for the probability

$$P\left(\hat{\beta}^*_L \leq \beta \leq \hat{\beta}^*_U\right)$$

We begin by noticing that since $2\sum_{i=1}^{n} X_i/\beta \sim \chi^2_{2n}$, then $\beta \sim 2n\hat{\beta}/\chi^2_{2n}$. Thus, we find that

$$\begin{aligned}
&P\left(\hat{\beta}^*_L \leq \beta \leq \hat{\beta}^*_U\right) = \\
&P\left(\frac{\hat{\beta}}{2n}\chi^2_{\alpha/2,2n} \leq 2n\hat{\beta}/\chi^2_{2n} \leq \frac{\hat{\beta}}{2n}\chi^2_{1-\alpha/2,2n}\right) \\
\Rightarrow &P\left(\frac{1}{4n^2}\chi^2_{\alpha/2,2n} \leq \frac{1}{\chi^2_{2n}} \leq \frac{1}{4n^2}\chi^2_{1-\alpha/2,2n}\right) \\
\Rightarrow &P\left(\frac{4n^2}{\chi^2_{1-\alpha/2,2n}} \leq \chi^2_{2n} \leq \frac{4n^2}{\chi^2_{\alpha/2,2n}}\right)
\end{aligned}$$

ok

We note that this probability should ideally be approximately equal to the nominal confidence level. Or, in other words, it should be approximately equal to $(1-\alpha)$. Next, we compute the exact coverage for $n = 5, 10, 20, 50, 100$ and for $\alpha = 0.05$. We implement this in the code section below.

```
probability <- function(n,alpha = 0.05){
  # Returns P(beta_hatstarL <= beta <= beta_hatstar_H)
  lower <- (4*n^2)/qchisq(1-alpha/2,2*n)
  upper <- (4*n^2)/qchisq(alpha/2, 2*n)
  return(diff(pchisq(c(lower,upper),2*n)))
}
# Calculate probability for n = 5,10,20,50,100:
probability(5)
probability(10)
probability(20)
probability(50)
probability(100)
```

The result can be seen in Table 3.

Percentile method					
n	5	10	20	50	100
$P\left(\hat{\beta}^*_L \leq \beta \leq \hat{\beta}^*_U\right)$	0.8982827	0.9227943	0.9360563	0.9443414	0.9471573

ok

Table 3: Exact coverage of the parametric bootstrap percentile interval for different n and $\alpha = 0.05$

From the table, we can see the general trend that as n increases, the exact coverage for the percentile method goes to $(1-\alpha) = 0.95$.

3.4 d)

In this section, we explore another alternative approach by implementing the *Accelerated Bias-Corrected Percentile Method* (BC$_a$). We will implement an R-function that takes the observed sample $x_1, ..., x_n$ and α as input and computes a two-sided $(1-\alpha)$ BC$_a$-confidence interval for β based on parametric bootstrapping.

As the name suggests, the accelerated bias-corrected percentile method is an algorithm that usually offers a substantial improvement over the regular percentile method, as it corrects for bias in this method. The algorithm for the accelerated bias-corrected percentile method that we implement is given in Algorithm 1. In the alogrith, we use notation $\boldsymbol{x}_{(-i)} = (x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_n)^T$. Further, z_α denotes the quantile for the standard normal distribution. In the general algorithm for the accelerated bias-corrected percentile method, the expression for the constant b is given as

$$b = \Phi^{-1}\left(\hat{F}^*(\hat{\theta})\right)$$

Algorithm 1 Accelerated Bias-Corrected Percentile Method

```
Input $\boldsymbol{x} = (x_1, x_2, ..., x_n)^T, x_i \overset{iid}{\sim} \text{Exp}\left(\frac{1}{\beta}\right)$ and $\alpha$
$\hat{\beta} = \text{mean}(\boldsymbol{x})$
$b = \Phi^{-1}\left(\Gamma(n, n/\hat{\beta})\right)$
For $i = 1, ..., n$:
  $\theta_i = \text{mean}(\boldsymbol{x}_{(-i)})$
end for
$\Psi_i = \bar{\theta} - \theta_i$
$a = \frac{1}{6}\sum_{i=1}^{n}\Psi_i^3 \Big/ \left(\sum_{i=1}^{n}\Psi_i^2\right)^{3/2}$
$\kappa_1 = \Phi\left(b + \frac{b+z_{\alpha/2}}{1-a(b+z_{\alpha/2})}\right)$
$\kappa_2 = \Phi\left(b + \frac{b+z_{1-\alpha/2}}{1-a(b+z_{1-\alpha/2})}\right)$
$\text{CI}_{BC_a} = \left(\frac{\hat{\beta}}{2n}\chi^2_{\kappa_1,2n}, \frac{\hat{\beta}}{2n}\chi^2_{\kappa_2,2n}\right)$
return $\text{CI}_{BC_a}$
```

very good that you wrote the procedure as an algorithm

In our algorithm, we have exploited the fact that we know the analytical distribution of the bootstrap replicates $\hat{\beta}^*$. Using this, we do not need to carry out simulations to find quantiles. The algorithm is implemented in the code below. Note that it is tested in the next section.

```r
set.seed(123)
BC_a <- function(x, alpha){
  # Accelerated bias-corrected percentile method
  # Note: using same notation as Givens & Hoeting
  beta_hat <- mean(x)
  n <- length(x)
  b = qnorm(pgamma(beta_hat, shape = n, rate = n/beta_hat))

  theta_i <- c()
  for (i in 1:n){
    theta_i = c(theta_i, mean(x[-i]))
  }
  psi <- mean(theta_i) - theta_i
  a <- (1/6)*sum(psi^3) / ((sum(psi^2))^(3/2))

  kappa1 <- pnorm(b + (b + qnorm(alpha/2)) / (1 - a*(b + qnorm(alpha/2))))
  kappa2 <- pnorm(b + (b + qnorm(1 - alpha/2)) / (1 - a*(b + qnorm(1 - alpha/2))))
  CI_BCa <- c(beta_hat*qchisq(kappa1,2*n)/(2*n), beta_hat*qchisq(kappa2,2*n)/(2*n))
  return(CI_BCa)
  }
```

3.5 e)

In this section we will assume a true value of $\beta = 1$, then estimate the coverage of the BC$_a$-interval in the previous section for $\alpha = 0.05$ by simulating 10000 random samples, each of size $n = 10$, then checking if the BC$_a$-intervals contain $\beta = 1$. We will compare the estimated coverage with the coverage of the regular percentile method implemented previously, as well as the exact $(1 - \alpha) = 0.95$ confidence interval. Finally, we will compare the three confidence intervals in the case of $n = 10$ and $\alpha = 0.05$.

In the next code section, we calculate the estimated coverage of the BC$_a$-interval for $n = 10$ by simulating 10000 random samples and compare it to the regular percentile method for $n = 10$.

```
beta_in_CI_BCa <- function(N = 10000, n = 10, beta = 1, alpha = 0.05){
  # Check if 1 in CI_BCa N times, for x sim Exp(n, 1/beta)
  # Models probability of beta in CI_BCa -> coverage CI_BCa
  count <- 0
  for (i in 1:N){
    x <- rexp(n, 1/beta)
    CI_BCa <- BC_a(x, alpha)
    # 1 in CI_BCa
    bool <- CI_BCa[1] < 1 & 1 < CI_BCa[2]
    if (bool){
      count <- count + 1
    }
  }
  return(count/N)
}
set.seed(123)
cat("Probability beta in CI for n = 10 \n:")
cat("CI_BCa: \n",beta_in_CI_BCa(n = 10))
cat("CI percentile method: \n", probability(n = 10))
```

We find that the estimate for the coverage of the BC$_a$-interval is given as 0.947 with this setup, while the coverage of the percentile method is given as 0.9227943. As expected, the accelerated bias-corrected percentile method performs better in the sense that it is closer to the exact coverage $(1-\alpha) = 0.95$. Further analysis of the two implementations can be seen in Figure 3. This figure is made with the code implemented below.

ok

```
# Vector of probabilities for percentile method
vec <- c(probability(5),probability(10),probability(20),probability(50),probability(100))
x_vec <- c(5,10,20,50,100)
plot(x_vec, vec, col = "red", ylab = "Probability", xlab = "n", main = "Probabilities of being in
points(10, beta_in_CI_BCa(n = 10), col = "blue", pch = 16)
legend(65, 0.91, legend=c("BCa", "Percentile method"),
       col=c("blue", "red"), pch = 16)
```

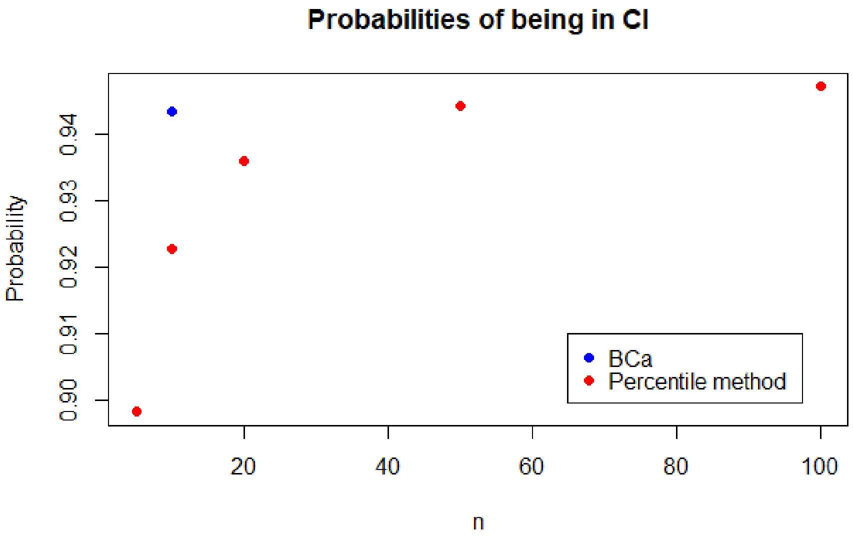


Figure 3: Comparison of coverages for BC$_a$ and Percentile method

From the figure, we can see that the estimated coverage of the BC$_a$-interval performs much better compared to the coverage for the interval of the regular percentile method. The coverage of the regular percentile method for $n = 50$ is close to the estimated coverage of the BC$_a$ for $n = 10$. Even though the accelerated bias-corrected percentile method outperforms the regular percentile method, both methods approximate exactly $(1 - \alpha)$ when n increases.

Next, we will compare the confidence interval off BC$_a$, the confidence interval of the regular percentile method, as well as exact $(1 - \alpha)$ confidence interval for β, with input $x_1, x_2, ..., x_{10}$, $x_i \sim \text{Exp}\left(\frac{1}{\beta}\right)$ with $\beta = 1$ and $\alpha = 0.05$. The intervals are calculated in the code below.

```
set.seed(1234)
x <- rexp(10, 1)
CI_BCa <- BC_a(x, 0.05)
beta_hat <- sum(x)/length(x)
CI_percentile <- beta_hat/(2*length(x)) * c(qchisq(0.05/2, 2*length(x)), qchisq(1-0.05/2, 2*lengt
beta_L <- (2*sum(x))/qchisq(1-0.05/2, 2*length(x))
beta_U <- (2*sum(x))/qchisq(0.05/2, 2*length(x))
CI_exact <- c(beta_L, beta_U)
cat("The confidence interval of BCa is given as: \n", CI_BCa, "\n")
cat("The confidence interval of percentile method is given as: \n", CI_percentile, "\n")
cat("The exact (1-alpha) CI for beta is: \n", CI_exact, "\n")
```

With these inputs, the BC$_a$-interval is given as $[0.4265276, 1.482772]$. The interval for the regular percentile method is given as $[0.3644566, 1.29847]$. Finally, the exact $(1 - \alpha)$ confidence interval of β is given as $[0.4448484, 1.584887]$. As expected, the BC$_a$-interval is closer to the exact interval compared to the interval for the regular percentile method.

very nice

4 Problem 3: The EM-algorithm and bootstrapping

good that you introduce the problem

In this problem we let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be independent random variables from the exponential distribution, where x_i has intensity parameter λ_0 and y_i has intensity parameter λ_1, for $i = 1, ..., n$. We assume that we do not observe $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ directly, but we observe

$$z_i = \max(x_i, y_i) \text{ for } i = 1, \ldots, n \tag{5}$$

and

$$u_i = \mathbb{I}(x_i \geq y_i) \text{ for } i = 1, \ldots, n, \tag{6}$$

where $\mathbb{I}(A) = 1$ if A is true and 0 otherwise. For each $i = 1, \ldots, n$, we thus observe the largest value of x_i and y_i and we know whether the observed value is x_i or y_i. Based on the observed (z_i, u_i), $i = 1, \ldots, n$, we will use the expectation maximization (EM) algorithm to find the maximum likelihood estimates for (λ_0, λ_1). The EM algorithm is given in Algorithm 2.

4.1 a)

In this section we obtain the so-called E-step of the EM-algorithm. We will start by obtaining an expression for the joint log likelihood of $(\boldsymbol{x}, \boldsymbol{y})$. The joint distribution, for each i, knowing that x_i and y_i are independent, is given by

$$f(x_i, y_i | \lambda_0, \lambda_1) = \lambda_0 \lambda_1 \exp\left\{-\lambda_0 x_i - \lambda_1 y_i\right\}.$$

The expression for the full likelihood is thus given by

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

Using the natural logarithm, we then obtain

$$\ln f(\boldsymbol{x},\boldsymbol{y}|\lambda_0,\lambda_1) = n(\ln\lambda_0+\ln\lambda_1) - \sum_{i=1}^{n}(\lambda_0 x_i + \lambda_1 y_i) \tag{7}$$

To estimate the model parameters, we use the EM algorithm given in Algorithm 2. In the algorithm, we have defined the E-step and the M-step. We start by finding the E-step. We have

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

ok

We continue by finding an expression for $\mathrm{E}[x_i|z_i,u_i,\lambda_0^{(t)}]$. We can partition $\boldsymbol{x}$ into two parts, $x_i = z_i$ and $x_i \leq z_i$. For the first part we have $\mathrm{E}[x_i|z_i,u_i,\lambda_0^{(t)}] = u_i z_i$. For the second part, we have to do some calculations. When $x_i < z_i$, we have

should you maybe use notation that corresponds to the density function? P(x_i) is technically =0

$$\mathrm{E}[x_i|x_i\leq z_i,u_i,\lambda_0^{(t)}] = (1-u_i)\cdot\int_0^{z_i} x_i P(x_i|x_i\leq z_i,\lambda_0^{(t)})dx_i$$

Now we need a way to express the probability $P(x_i|x_i\leq z_i,\lambda_0^{(t)})$. This can be done using Bayes theorem, from which we find

$$\begin{aligned}
P(x_i|x_i\leq z_i,\lambda_0^{(t)}) &= \frac{\overset{=1}{P(x_i\leq z_i|x_i,\lambda_0^{(t)})}\cdot P(x_i|\lambda_0^{(t)})}{P(x_i\leq z_i|\lambda_0^{(t)})} = \frac{P(x_i|\lambda_0^{(t)})}{P(x_i\leq z_i|\lambda_0^{(t)})} \\
&= \frac{\lambda_0^{(t)}\exp\{-\lambda_0^{(t)}x_i\}}{1-\exp\{-\lambda_0^{(t)}z_i\}}
\end{aligned}$$

where we have used that $P(x_i|\lambda_0^{(t)})$ is given from an exponential distribution with intensity parameter $\lambda_0^{(t)}$ and $P(x_i\leq z_i|\lambda_0^{(t)}) = F_{x_i}(z_i)$, which is the cumulative distribution function of x_i. Now, all that is left is determining the integral in

$$\begin{aligned}
&\int_0^{z_i} x_i P(x_i|x_i\leq z_i|\lambda_0^{(t)})dx_i = \int_0^{z_i} x_i\frac{\lambda_0^{(t)}\exp\{-\lambda_0^{(t)}x_i\}}{1-\exp\{-\lambda_0^{(t)}z_i\}}dx_i \\
&= \frac{\lambda_0^{(t)}}{1-\exp\{-\lambda_0^{(t)}z_i\}}\int_0^{z_i} x_i\exp\{-\lambda_0^{(t)}x_i\}dx_i \\
&\overset{i.b.p}{=} \frac{\lambda_0^{(t)}}{1-\exp\{-\lambda_0^{(t)}z_i\}}\left(\frac{1}{\lambda_0^{(t)}}\left(\exp\{-\lambda_0^{(t)}z_i\}\left(z_i+\frac{1}{\lambda_0^{(t)}}\right)+\frac{1}{\lambda_0^{(t)}}\right)\right) \\
&= \frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp\{\lambda_0^{(t)}z_i\}-1}
\end{aligned}$$

Putting it all together, we have

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

The same calculation can be done for $\mathrm{E}[y_i|z_i,u_i,\lambda_1^{(t)}]$, obtaining

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

Now, bringing all the pieces together, we obtain an expression for the expectation of the log-likelihood function:

$$\begin{aligned}\mathrm{E}[\ln f(\boldsymbol{x},\boldsymbol{y}|\lambda_0,\lambda_1)|\boldsymbol{z},\boldsymbol{u},\lambda_0^{(t)},\lambda_1^{(t)}] &= n(\ln\lambda_0+\ln\lambda_1)\\ &\quad-\lambda_0\sum_{i=1}^{n}\left(u_i z_i+(1-u_i)\left(\frac{1}{\lambda_0^{(t)}}-\frac{z_i}{\exp\{\lambda_0^{(t)}z_i\}-1}\right)\right)\\ &\quad-\lambda_1\sum_{i=1}^{n}\left((1-u_i)z_i+u_i\left(\frac{1}{\lambda_1^{(t)}}-\frac{z_i}{\exp\{\lambda_1^{(t)}z_i\}-1}\right)\right) \qquad (8)\end{aligned}$$

ok

Algorithm 2 Expectation Maximization (EM) algorithm

Aim: Maximize "difficult" likelihood $f_X(x|\theta)$ of X where $X = M(Y)$ for Y the complete data
Algorithm: Let

$$\begin{aligned}Q(\theta|\theta^{(t)}) &= \mathrm{E}[\ln f_Y(Y|\theta)|X=x,\theta=\theta^{(t)}]\\ &= \int_{\{y:M(y)=x\}} \ln f_Y(y|\theta) f_{Y|X}(y|x,\theta^{(t)})dy\end{aligned}$$

nice

E-step: Find $Q(\theta|\theta^{(t)})$
M-step: Set $\theta^{(t+1)} = \arg\max_{\theta} Q(\theta|\theta^{(t)})$

4.2 b)

In the previous section we found the E-step of the EM-algorithm. In other words, we found that

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

In this section, we will derive the M-step, and then we implement the EM-algorithm. First, we find analytical estimates for the MLE. We find that

$$\frac{\mathrm{d}}{\mathrm{d}\lambda_0}\mathrm{E}[\ln f(\boldsymbol{x},\boldsymbol{y}|\lambda_0,\lambda_1)|\boldsymbol{z},\boldsymbol{u},\lambda_0^{(t)},\lambda_1^{(t)}] = \frac{n}{\lambda_0}-\sum_{i=1}^{n}\left(u_i z_i+(1-u_i)\left(\frac{1}{\lambda_0^{(t)}}-\frac{z_i}{\exp\{\lambda_0^{(t)}z_i\}-1}\right)\right)=0$$

and

$$\frac{\mathrm{d}}{\mathrm{d}\lambda_1}\mathrm{E}[\ln f(\boldsymbol{x},\boldsymbol{y}|\lambda_0,\lambda_1)|\boldsymbol{z},\boldsymbol{u},\lambda_0^{(t)},\lambda_1^{(t)}] = \frac{n}{\lambda_1}-\sum_{i=1}^{n}\left(u_i z_i+(1-u_i)\left(\frac{1}{\lambda_0^{(t)}}-\frac{z_i}{\exp\{\lambda_0^{(t)}z_i\}-1}\right)\right)=0$$

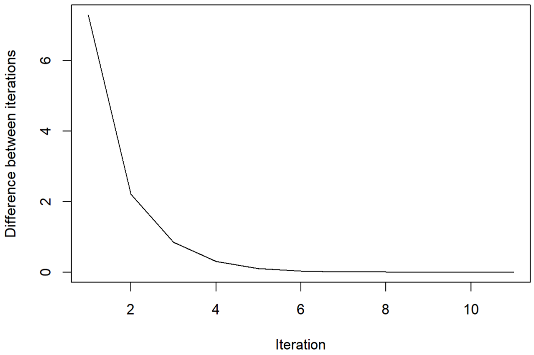
such that the MLE is given as

$$\hat{\lambda}_0 = \frac{n}{\sum_{i=1}^{n}\left(u_i z_i+(1-u_i)\left(\frac{1}{\lambda_0^{(t)}}-\frac{z_i}{\exp\{\lambda_0^{(t)}z_i\}-1}\right)\right)}, \qquad \hat{\lambda}_1 = \frac{n}{\sum_{i=1}^{n}\left(u_i z_i+(1-u_i)\left(\frac{1}{\lambda_0^{(t)}}-\frac{z_i}{\exp\{\lambda_0^{(t)}z_i\}-1}\right)\right)}$$

ok

With the E-step and the M-step completed, we implement the EM-algorithm. The implementation is done in the code below.

```
u <- scan(file="https://www.math.ntnu.no/emner/TMA4300/2024v/u.txt")
z <- scan(file="https://www.math.ntnu.no/emner/TMA4300/2024v/z.txt")
n <- length(u)
tol <- 1e-10
lambda_0_init <- 1
lambda_1_init <- 1
```



ok

Figure 4: Convergence of the EM algorithm

```
E_step <- function(lambda,n,u,z, lambda_0_temp, lambda_1_temp){
  return ((-1)*(n*(log(lambda[1]*lambda[2]))-
            lambda[1]*sum(u*z+(1-u)*(1/lambda_0_temp-z/(exp(lambda_0_temp*z)-1))) -
            lambda[2]*sum((1-u)*z+u*(1/lambda_1_temp-z/(exp(lambda_1_temp*z)-1)))))
}
EM_algorithm <- function(n, u, z, lambda_0_temp = 1, lambda_1_temp = 1,step, tolerance = tol){
  difference <- 1e6
  convergence <- c(difference)
  lambda_temp <- c(lambda_0_temp,lambda_1_temp)
  expectation <- E_step(c(lambda_0_temp,lambda_1_temp),n,u,z, lambda_0_temp, lambda_1_temp)
  while (difference >tol){

    lambda_0 <- n/(sum(u*z+(1-u)*(1/lambda_temp[1]-
    z/(exp(lambda_temp[1]*z)-1))))
    lambda_1 <- n/(sum((1-u)*z+u*(1/lambda_temp[2]-
    z/(exp(lambda_temp[2]*z)-1))))
    lambda <- c(lambda_0,lambda_1)
    expectation_temp <- E_step(lambda,n,u,z, lambda_temp[1], lambda_temp[2])
    difference <- abs(expectation-expectation_temp)
    expectation <- expectation_temp
    convergence <- c(convergence, difference)
    lambda_temp <- lambda
  }
  return(list(lambda = lambda, conv = convergence))
}
result <- EM_algorithm(n,u,z)
result$conv
result$lambda
```

We have decided to stop the algorithm when the difference between two iterations of the algorithm is smaller than some predefined tolerance. From `result$lambda` we obtain the estimates, which are given as

$$\hat{\boldsymbol{\lambda}}_{EM} = \begin{pmatrix}\hat{\lambda}_0\\ \hat{\lambda}_1\end{pmatrix}_{EM} = \begin{pmatrix}3.465735\\ 9.353215\end{pmatrix}_{EM}$$

ok

The convergence, plotting `difference` for each iteration, can be seen in Figure 4. It can be shown that the EM algorithm is guaranteed to find a local minimum or saddle point.

4.3 c)

In this section we use bootstrapping in order to find standard deviations and biases for $\hat{\lambda}_0$ and $\hat{\lambda}_1$, and to estimate $\mathrm{Corr}[\hat{\lambda}_0, \hat{\lambda}_1]$. We use non-parametric bootstrapping, resampling from $\boldsymbol{u}$ and $\boldsymbol{z}$, and generate bootstrap replicates $\hat{\boldsymbol{\lambda}}^{b*}$, for $b = 1, \dots, B$ using the EM-algorithm. Pseudocode for non-parametric bootstrapping is given in Algorithm 3. Next, we implement the algorithm in the following code block.

Algorithm 3 Non-parametric boostrapping

```
Suppose we have iid sample $\boldsymbol{x} = x_1, \dots, x_n \sim F$ where $F$ is an unknown cumulative distribution function. Estimate $F$ by $\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(x_i < x)$.
Algorithm:
Generate B boostrap samples $x^{1*}, \dots, x^{B*}$ where $x^{b*} = (x_1^{b*}, \dots, x_n^{B*}$ and $x_i^{b*} \overset{iid}{\sim} \hat{F}$ for $i = 1, \dots, n$ and $b = 1, \dots, B$.
Compute corresponding boostrap replicates of plug-in estimator $\hat{\theta}$ of $\theta$,
    $\hat{\theta}^{b*} = t(\hat{F})$
Estimate e.g. $\mathrm{E}_{\hat{F}}[\hat{\theta}]$ and $\mathrm{SD}_{\hat{F}}[\hat{\theta}]$ or in general $\mathrm{E}[h(\hat{\theta})]$, i.e.
    $\widehat{\mathrm{E}[\hat{\theta}]} = \frac{1}{B}\sum_{b=1}^{B} \hat{\theta}^{b*}$
and
    $\widehat{\mathrm{SD}[\hat{\theta}]} = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}^{b*} - \widehat{\mathrm{E}[\hat{\theta}]})}$
```

```r
bootstrap_EM_fun <- function(B,n){
  lambda_star <- t(replicate(B,{
    resample <- sample(1:n, replace = TRUE)
    EM_algorithm(length(u[resample]),u[resample],z[resample])$lambda
  }))
}
lambda_sim <- bootstrap_EM_fun(10000,n)
bs_lambda <- colMeans(lambda_sim)
bs_lambda_sd <- colSds(lambda_sim)
bias_lambda_0_hat <-bs_lambda[1] -lambda_0_hat
bias_lambda_1_hat <- bs_lambda[2] -lambda_1_hat
bias_lambda_0_hat
bias_lambda_1_hat
cor(lambda0_sim,lambda1_sim)
bs_lambda_sd
```

The estimated biases of the EM-estimate are given in Table 4.

$\widehat{\mathrm{Bias}[\hat{\lambda}_0]}$	$\widehat{\mathrm{Bias}[\hat{\lambda}_1]}$
0.01695085	0.08861417

ok

Table 4: Estimated biases for $\hat{\lambda}_0$ and $\hat{\lambda}_1$

The estimated standard deviations of the estimators are given in Table 5.

$\widehat{\text{SD}[\hat{\lambda}_0]}$	$\widehat{\text{SD}[\hat{\lambda}_1]}$
0.2469660	0.0.7957022

ok

Table 5: Estimated standard deviations for $\hat{\lambda}_0$ and $\hat{\lambda}_1$

The estimated correlation is given as

$$\text{Corr}[\hat{\lambda}_0, \hat{\lambda}_1] = -0.00641908.$$

We see that we have a relatively small bias compared to the estimated coefficients from the EM algorithm. Looking at the standard deviation, we see that the bias corrected estimates will introduce variance to the estimates, which when compared to the bias, leads us to think that we would prefer to use the MLE instead of the bias corrected estimates. The estimated correlation is very small, which is consistent with the fact that $\boldsymbol{x}$ and $\boldsymbol{y}$ are assumed to be independent. ok

4.4 d)

In this section we will determine whether it is possible to find an analytical formula for $f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1)$. We will examine the statistical properties of u_i and z_i. We have

$$u_i = \mathbb{I}(x_i, y_i) = \begin{cases} 1 & \text{for } x_i \geq y_i \\ 0 & \text{else} \end{cases} \sim \text{bernoulli}(P(x_i \geq y_i|\lambda_0, \lambda_1))$$

and

$$z_i = \max(x_i, y_i) \sim \begin{cases} \text{Exp}(\lambda_0) & \text{for } x_i > y_i \\ \text{Exp}(\lambda_1) & \text{for } y_i > x_i \end{cases}$$

Thus, we can make an expression for the joint distribution of z_i, u_i, using the law of total probability and the statistical property $P(A \cap B) = P(A|B)P(B)$:

$$\begin{aligned} P(Z_i = z_i, U_i = u_i|\lambda_0, \lambda_1) &= u_i P(Z_i = z_i, X_i \geq Y_i|\lambda_0, \lambda_1) + (1 - u_i)P(Z_i = z_i, Y_i \geq X_i|\lambda_0, \lambda_1) \\ &= u_i P(Z_i = z_i|X_i \geq Y_i, \lambda_0, \lambda_1)P(X_i \geq Y_i|\lambda_0, \lambda_1) \\ &+ (1 - u_i)P(Z_i = z_i|Y_i \geq X_i, \lambda_0, \lambda_1)P(Y_i \geq X_i|\lambda_0, \lambda_1) \end{aligned}$$

Now, we find that $P(Z_i = z_i|X_i \geq Y_i, \lambda_0, \lambda_1) = P(Z_i = x_i|\lambda_0, \lambda_1)$ and $P(X_i \geq Y_i|\lambda_0, \lambda_1) = F_{Y_i}(X_i = x_i)$ where F is the cdf. The same is true for when $Y_i \geq X_i$. Using this, obtain

$$\begin{aligned} P(Z_i = z_i, U_i = u_i|\lambda_0, \lambda_1) = u_i(\lambda_0 \exp\{-\lambda_0 z_i\})(1 - \exp\{-\lambda_1 z_i\}) \\ + (1 - u_i)(\lambda_1 \exp\{-\lambda_1 z_i\})(1 - \exp\{-\lambda_0 z_i\}) \end{aligned}$$

ok

Having an expression for $P(Z_i = z_i, U_i = u_i|\lambda_0, \lambda_1)$, we need to determine the log-likelihood, $\ln f(\boldsymbol{z}, \boldsymbol{u}|\lambda_0, \lambda_1)$ in order to estimate the parameters λ_0 and λ_1. We might run into some issues, since $\ln(\prod_{i=1}^n u_i) = \sum_{i=1}^n ln(u_i)$, and $\ln 0$ is undefined. Therefore, we need to consider the cases separately, assuming we do not evaluate the first half in the occurences where $u_i = 0$, and similarly we do not evaluate the second half when $u_i = 1$. We can then proceed to obtain an expression for the log-likelihood. First for the case $u_i = 1$, we obtain

$$\ln(f(\boldsymbol{z}, \boldsymbol{u}|\lambda_0, \lambda_1)) = \ln\left[\lambda_0^{n_1} \exp\{-\lambda_0 \sum_{i=1}^n u_i z_i\} \prod_{i=1}^n u_i(1 - \exp\{-\lambda_1 z_i\})\right]$$

For the case $u_i = 0$, we find that

$$\ln(f(\boldsymbol{z}, \boldsymbol{u}|\lambda_0, \lambda_1)) = \ln\left[\lambda_1^{n_0} \exp\{-\lambda_1 \sum_{i=1}^n (1 - u_i)z_i\} \prod_{i=1}^n (1 - u_i)(1 - \exp\{-\lambda_0 z_i\})\right]$$

where $n_1 = \sum_{i=1}^{n} u_i$ and $n_0 = n - n_1$. Putting it together, we obtain

$$\ln(f(\boldsymbol{z},\boldsymbol{u}|\lambda_0,\lambda_1) = n_1 \ln\lambda_0 + n_0 \ln\lambda_1 - \sum_{i=1}^{n} u_i(\lambda_0 z_i + \ln(1-\exp\{-\lambda_1 z_i\})) \\ - \sum_{i=1}^{n}(1-u_i)(\lambda_1 z_i + \ln(1-\exp\{-\lambda_0 z_i\})) \tag{9}$$

We find that we cannot solve this equation for λ_0 nor for λ_1 by differentiating and finding its maximum analytically. For λ_1, this can be seen by the fact that

$$\frac{\mathrm{d}}{\mathrm{d}\lambda_1}(\ln(1-\exp\{-\lambda_1 z_i\})) = \frac{z_i}{\exp\{\lambda_0 z_i\}-1} = 0$$

has no closed for solution, and thus we need to solve the optimization problem numerically. Similarly can be shown for λ_0. The optimization problem is solved numerically in the following code, using the R-function `optim()`.

ok

```
log_lik <- function(param,u,z){
  n_1 <- sum(u)
  n_0 <- sum(1-u)
  return((-1)*(n_1*log(param[1])+n_0*log(param[2])-
                sum(u*(param[1]*z-log(1-exp(-param[2]*z))))-
                sum((1-u)*(param[2]*z-log(1-exp(-param[1]*z))))))
}
param <- c(10,10)
result <- optim(param, fn = log_lik, u = u, z = z)
result$par
```

From the implementation, we found that

$$\hat{\boldsymbol{\lambda}} = \begin{pmatrix} \hat{\lambda}_0 \\ \hat{\lambda}_1 \end{pmatrix} = \begin{pmatrix} 3.465991 \\ 9.353262 \end{pmatrix}$$

ok

This result is almost identical to the previous results obtained. Optimizing from the likelihood directly is much faster than using the EM-algorithm, since the EM-algorithm generally converges slower than MLE. The EM-algorithm is only guaranteed to find local optima or saddle points, while the MLE. The likelihood function we found is convex, such that it has a unique maximum.