TMA4300 - Project 3

Jakob Heide and Celine Olsson

2024-04-26

Part 1: Bootstrapping a GLM

Let $Y_1, \ldots, Y_n$ be independent random variables, where $Y_i \sim \text{bin}(m_i, p_i)$ where $\ln(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 x_i$, for $i = 1, \ldots, n$. This is a logistic regression model that we fit by maximum likelihood.

```r
load(file=url("https://www.math.ntnu.no/emner/TMA4300/2024v/data.Rdata"))
mod <- glm(cbind(y, m - y) ~ x, family=binomial, data=data)
```

1a)

First we want to fit the same model again, only using bootstrapping. Below is a function which takes in the fitted model `mod`, and simulates $B = 10000$ bootstrap samples by resampling from the data containing (m_i, y_i, x_i) with replacements. Then the bootstrap samples are used to refit the same model as before. This gives the bootstrap replicates $\hat{\beta}^{b*}$ of $\hat{\beta}$, where $b = 1, \ldots, B$. The bootstrap replicates are returned as a $(B \times 2)$-matrix, where $\hat{\beta}_0^{b*}$ is in the first column and $\hat{\beta}_1^{b*}$ is in the second column.

```r
bootstrapGLM <- function(mod){
  B = 10000
  bootstraps = matrix(NA,nrow = B,ncol = length(mod$coefficients))

  for (i in 1:B){
    sampleIndexes = sample(mod$data$x, length(mod$data$x), replace = T)
    newSample = mod$data[sampleIndexes,]
    rownames(newSample) = NULL
    newMod = glm(cbind(y, m - y) ~ x, family = binomial, data = newSample)

    bootstraps[i,] = newMod$coefficients
  }
  return (bootstraps)
}
```

1b)

Now we want to calculate an estimate for the variance of $\hat{\beta}$ using the bootstrap replicates $\hat{\beta}^{b*}$, which we just calculated. This is done by first taking the variance of all $\hat{\beta}_0^{b*}$ values which gives an estimate of $Var(\hat{\beta}_0)$, then the same is done for all $\hat{\beta}_1^{b*}$ values. This gives the following variance estimates

```
bootstrapSamples = bootstrapGLM(mod)
varEst = apply(bootstrapSamples,2,var)
varEst
```

```
## [1] 0.59459870 0.01115203
```

We compare these to the ones we get when using the `vcov` function on the original fitted model, which gives

```
vcov(mod)
```

```
##             (Intercept)            x
## (Intercept)  0.19649741 -0.028022675
## x           -0.02802267  0.005049491
```

The values we got from our bootstrapping does not match the values given from `vcov` very well. The variance is a lot bigger for the bootstrapping.

Problem 1c)

Now the bias of the MLEs of the intercept and slope parameters is to be estimated. To do this we use that the estimated bias can be found by calculating the following

$$\widehat{\text{Bias}(\hat{\beta})} = \widehat{E(\hat{\beta})} - \hat{\beta}$$

The estimate of the expected value is defined as the following

$$\widehat{E(\hat{\beta})} = \frac{1}{B}\sum_{i=1}^{B}\hat{\beta}^{b*}$$

In our case $\hat{\beta}$ is the coefficients given from the original fitted model. The bias estimator is calculated below.

```
biasEstimator = colMeans(bootstrapSamples) - mod$coefficients
biasEstimator
```

```
##  (Intercept)            x
## -0.061386185  0.008234163
```

The bias-corrected estimator, $\hat{\beta}_c$, is calculated by taking

$$\hat{\beta}_c = \hat{\beta} - \widehat{\text{Bias}(\hat{\beta})}$$

The calculated bias-corrected estimator becomes the following

```
correctedEstimator = mod$coefficients - biasEstimator
correctedEstimator
```

```
## (Intercept)           x
## -0.61672050  0.09275636
```

1d)

Next we want to make a 95%-confidence interval for each model parameter, using the bootstrap replicates. This is done using the `quantile` function on all the bootstrap replicates for both $\hat{\beta}_0$ and $\hat{\beta}_1$. The following gives a 95%-confidence interval.

```
betaCI = apply(bootstrapSamples,2,quantile,c(0.025,0.975))
colnames(betaCI) = c("(Intercept)","x")
betaCI
```

```
##       (Intercept)          x
## 2.5%   -2.0360561 -0.1023157
## 97.5%   0.8298432  0.3004280
```

This is then compared to the confidence interval given by the `confint` function on the original fitted model.

```
t(confint(mod))
```

```
##        (Intercept)          x
## 2.5 %   -1.5718941 -0.0367474
## 97.5 %   0.1779204  0.2434345
```

We see that some of the values are close, but that the confidence interval we got from the bootstrapping is not a match. The CI from the bootstrapping is not as narrow.

1e)

Now all of **a)** to **d)** is to be done again, only with parametric bootstrapping instead.

First the R-function for making bootstrap replicates. Here we use that the distribution of $Y_i \sim bin(m_i, p_i)$ is known, and that $\ln(\frac{p_i}{1-p_i}) = \beta_o + \beta_1 x_i$ for $i = 1, \ldots, n$. This can be rewritten to find an expression for p_i:

$$\ln\left(\frac{p_i}{1-p_i}\right) = \boldsymbol{x}^T\boldsymbol{\beta} \quad \Rightarrow \quad p_i = \frac{1}{1+\exp(-\boldsymbol{x}^T\boldsymbol{\beta})} = \text{expit}(\boldsymbol{x}^T\boldsymbol{\beta})$$

Here $\boldsymbol{\beta} = (\beta_0, \beta_1)^T$, $\boldsymbol{x} = (1, x_i)^T$ and the expit-function is the inverse of the logit-function.

```
parametricBootstrap <- function(mod){
  B = 10000
  p = inv.logit(mod$coefficients[1] + mod$coefficients[2]*mod$data$x)
  bootstraps = matrix(NA,nrow = B,ncol = length(mod$coefficients))
  sample = data

  for (i in 1:B){
    newSample = rbinom(10,mod$data$m,p)

    sample$y = newSample

    newMod = glm(cbind(y, m - y) ~ x, family = binomial, data = sample)
    bootstraps[i,] = newMod$coefficients

  }
  return (bootstraps)
}
```

The variance of these replicates are then calculated the same way as in **b)**.

```
bootstrapSamples = parametricBootstrap(mod)
varEst = apply(bootstrapSamples,2,var)
varEst
```

```
## [1] 0.20362153 0.00521503
```

Again we compare them to the values given by `vcov`.

```
vcov(mod)
```

```
##             (Intercept)            x
## (Intercept)  0.19649741 -0.028022675
## x           -0.02802267  0.005049491
```

These seem to be a much better match than when using non-parametric bootstrapping.

Over to the bias estimator.

```
biasEstimator = colMeans(bootstrapSamples) - mod$coefficients
biasEstimator
```

```
##  (Intercept)            x
## -0.014488123  0.002573057
```

The bias estimators are lower than when using non-parametric bootstrapping. The bias-corrected estimator is calculated below.

```
correctedEstimator = mod$coefficients - biasEstimator
correctedEstimator
```

```
## (Intercept)           x
## -0.66361856  0.09841746
```

Lastly the 95% confidence interval is calculated the same way as in **d)**, and gives the following.

```
betaCI = apply(bootstrapSamples,2,quantile,c(0.025,0.975))
colnames(betaCI) = c("(Intercept)","x")
betaCI
```

```
##       (Intercept)           x
## 2.5%   -1.6263745 -0.03643712
## 97.5%   0.1825463  0.24990750
```

The CI using the `confint` function on the fitted model is shown below.

```
t(confint(mod))
```

```
##        (Intercept)          x
## 2.5 %   -1.5718941 -0.0367474
## 97.5 %   0.1779204  0.2434345
```

We see that again the values from the parametric bootstrapping is a much better match than with the non-parametric bootstrapping.

Part 2: Bootstrapping confidence intervals

2a)

Let $X_1, X_2, \dots, X_n$ be i.i.d. samples from an exponential distribution with scale parameter β. The exponential distribution is a special case of the Gamma distribution with shape parameter $\alpha = 1$, that is

$$X_i \sim \exp(1/\beta) \iff X_i \sim \text{Gamma}(1, 1/\beta), \quad i = 1, \dots, n$$

The scaling property of the Gamma distribution gives us

$$X_i \sim \text{Gamma}(1, 1/\beta) \implies \frac{2}{\beta} X_i \sim \text{Gamma}(1, 1/2), \quad i = 1, \dots, n$$

In addition, the Gamma distribution has a summation property, which states that

$$\frac{2}{\beta}\sum_{i=1}^{n} X_i \sim \text{Gamma}(n, 1/2),$$

which holds since X_i are independent and have the same scale parameter. Lastly, we use that the chi-square distribution is a special case of the gamma distribution with rate parameter 1/2, that is,

$$\frac{2}{\beta}\sum_{i=1}^{n} X_i \sim \text{Gamma}(n, 1/2) \iff \frac{2}{\beta}\sum_{i=1}^{n} X_i \sim \chi^2_{2n}$$

We construct a $(1-\alpha)\%$ confidence interval. Let $\mathcal{X} = \frac{2}{\beta}\sum_{i=1}^{n} X_i$, then

$$\begin{aligned} & P(\chi^2_{\alpha/2,2n} \le \mathcal{X} \le \chi^2_{1-\alpha/2,2n}) = 1-\alpha \\ \iff & P\left(\frac{2\sum_i X_i}{\chi^2_{1-\alpha/2,2n}} \le \beta \le \frac{2\sum_i X_i}{\chi^2_{\alpha/2,2n}}\right) = 1-\alpha \\ \iff & P\left(\frac{2n\hat{\beta}}{\chi^2_{1-\alpha/2,2n}} \le \beta \le \frac{2n\hat{\beta}}{\chi^2_{\alpha/2,2n}}\right) = 1-\alpha, \end{aligned}$$

where in the last relation we have inserted the MLE estimate of β, namely $\hat{\beta} = \frac{1}{n}\sum_{i=1}^{n} X_i$.

2b)

Now we want to use parametric bootstrapping instead, and construct a bootstrapping confidence interval for β using the percentile method. Define $\hat{\beta}^*$ to be the bootstrap replicates of $\hat{\beta}$, where $\hat{\beta}$ is the MLE of β.

From earlier the distribution of $2\sum_{i=1}^{n} X_i/\beta$ is known. When using bootstrapping, we now consider X_i^* for $i = 1, \dots, n$, to be the bootstrapping samples. The bootstrapping samples X_i^* is then exponentially distributed with scale parameter $\hat{\beta}$. When estimating the mean we get

$$\frac{1}{n}\sum_{i=1}^{n} X_i^* = \hat{\beta}^*$$

This give the following distribution when using bootstrapping

$$2\sum_{i=1}^{n} \frac{X_i}{\beta} \sim \chi^2_{2n} \quad \overset{\text{bootstrap}}{\Rightarrow} \quad \frac{2n\hat{\beta}^*}{\hat{\beta}} \sim \chi^2_{2n}$$

Using this gives the distribution of $\hat{\beta}^*$, which is

$$\hat{\beta}^* \sim \frac{\hat{\beta}}{2n}\chi^2_{2n} \sim \text{Gamma}(n, n/\hat{\beta}).$$

We know that

$$P\left(\chi^2_{\alpha/2,2n} \leq \frac{2n\hat{\beta}^*}{\hat{\beta}} \leq \chi^2_{1-\alpha/2,2n}\right) = 1-\alpha$$

Rewriting gives the confidence interval of $\hat{\beta}^*$:

$$P\left(\frac{\hat{\beta}}{2n}\chi^2_{\alpha/2,2n} \leq \hat{\beta}^* \leq \frac{\hat{\beta}}{2n}\chi^2_{1-\alpha/2,2n}\right) = 1-\alpha$$

2c)

Now we want to find an expression for the exact coverage of the parametric bootstrap percentile interval from the previous task.

Define the upper and lower limit of the confidence interval of $\hat{\beta}^*$ to be

$$\hat{\beta}^*_L = \frac{\hat{\beta}}{2n}\chi^2_{\alpha/2,2n}$$
$$\hat{\beta}^*_U = \frac{\hat{\beta}}{2n}\chi^2_{1-\alpha/2,2n}$$

The exact coverage is the probability that $\beta \in [\hat{\beta}^*_L, \hat{\beta}^*_U]$. From a), we know that

$$\mathcal{X} = \frac{2n\hat{\beta}}{\beta} \sim \chi^2_{2n} \Rightarrow \beta = \frac{2n\hat{\beta}}{\mathcal{X}}$$

The probability becomes

$$P\left(\hat{\beta}^*_L \leq \beta \leq \hat{\beta}^*_U\right) = P\left(\hat{\beta}^*_L \leq \frac{2n\hat{\beta}}{\mathcal{X}} \leq \hat{\beta}^*_U\right)$$
$$= P\left(\frac{2n\hat{\beta}}{\hat{\beta}^*_U} \leq \mathcal{X} \leq \frac{2n\hat{\beta}}{\hat{\beta}^*_L}\right) = P\left(\frac{4n^2}{\chi^2_{1-\alpha/2,2n}} \leq \mathcal{X} \leq \frac{4n^2}{\chi^2_{\alpha/2,2n}}\right)$$
$$= F\left(\frac{4n^2}{\chi^2_{\alpha/2,2n}}\right) - F\left(\frac{4n^2}{\chi^2_{1-\alpha/2,2n}}\right),$$

where $F(\cdot)$ is the cumulative distribution function of the chi-square distribution with $2n$ degrees of freedom.

We compute the exact coverage for $n = 5, 10, 20, 50, 100$ and $\alpha = 0.05$.

```r
coverage <- function(n, alpha=0.05){
  alpha <- alpha/2
  chi_vals <- qchisq(c(1-alpha,alpha),2*n)

  beta_lims <- 4*n^2 /  chi_vals

  return(beta_lims)
}
```

```
for (i in c(5,10,20,50,100)) {
  lims <- coverage(i)

  print(pchisq(lims[2], 2*i) - pchisq(lims[1], 2*i))
}
```

```
## [1] 0.8982827
## [1] 0.9227943
## [1] 0.9360563
## [1] 0.9443414
## [1] 0.9471573
```

We see that with increasing n, the probability approaches $(1 - \alpha) = 0.95$.

2d)

We implement the accelerated bias-corrected percentile method (BC_a) which computes a two-sided $(1 - \alpha)$ confidence interval. In the implementation, we exploit the fact that we know the distribution of $\hat{\beta}^*$, so we do not need to simulate or do any bootstrapping.

```
BCconf <- function(x,alpha){
  n = length(x)
  betaHat = mean(x)

  b = qnorm(pgamma(betaHat,shape = n,rate = n/betaHat))

  theta_i = c()
  for (i in 1:n){
    theta_i = c(theta_i,mean(x[-i]))
  }
  theta_not = mean(theta_i)
  phi = theta_not - theta_i

  a = (1/6)*sum(phi^3) / ((sum(phi^2))^(3/2))

  beta1 = pnorm(b + (b+qnorm(alpha/2))/(1-a*(b + qnorm(alpha/2))))
  beta2 = pnorm(b + (b+qnorm(1-alpha/2))/(1-a*(b + qnorm(1-alpha/2))))

  return (c(betaHat*qchisq(beta1,2*n)/(2*n),betaHat*qchisq(beta2,2*n)/(2*n)))
}
```

2e)

We assume $\beta = 1$, and simulate 10000 random samples from the exponential distribution of size $n = 10$. For each sample, we calculate the 95% BC_a confidence interval, and check if the interval contains $\beta = 1$. This way we can estimate the coverage of the BC_a method.

```
#2e)

beta = 1
```

```r
alpha = 0.05

coverage = 0
set.seed(4)
for (i in 1:10000){
  sample = rexp(10,rate = 1/beta)
  CI = BCconf(sample,alpha)
  if (CI[1] <= beta && CI[2] >= beta){
    coverage = coverage + 1
  }
}
coverage = coverage/10000 #Probability!!!
print(coverage)
```

```
## [1] 0.9462
```

We see that the BC_a intervals contain the value $\beta = 1$ for a total of 94.62% of the samples. In 2c), we saw that the probability of the parametric bootstrap confidence interval containing the true value of β grew towards 0.95 with increasing n, and for $n = 100$ the probability was 0.947, surpassing the coverage of the BC_a interval (which had a sample size of 10). However, for $n = 10$, the BC_a outperforms the parametric bootstrap confidence interval.

Part 3: The EM algorithm and bootstrapping

3a)

Let $\boldsymbol{X} = (X_1, X_2, \dots, X_n)^T$ and $\boldsymbol{Y} = (Y_1, Y_2, \dots, Y_n)^T$ be independent random variables, with $X_i \sim \exp(\lambda_0)$, $Y_i \sim \exp(\lambda_1)$. The complete data likelihood is

$$
\begin{aligned}
f(\boldsymbol{x}, \boldsymbol{y}|\lambda_0, \lambda_1) = \prod_{i=1}^n f(x_i, y_i|\lambda_0, \lambda_1) &= \prod_{i=1}^n f(x_i|\lambda_0) f(y_i|\lambda_1) \\
&= \prod_{i=1}^n \lambda_0\lambda_1 \exp(-\lambda_0 x_i)\exp(-\lambda_1 y_i) \\
= (\lambda_0\lambda_1)^n \exp\left(-\sum_{i=1}^n (\lambda_0 x_i + \lambda_1 y_i)\right), &\quad x_i, y_i, \lambda_0, \lambda_1 > 0.
\end{aligned}
$$

The log-likelihood is

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

However, we do not observe $\boldsymbol{X}$ and $\boldsymbol{Y}$ directly. Instead, we observe

$$
z_i = \max(x_i, y_i), \quad i = 1, \dots, n,
$$
$$
u_i = I(x_i \geq y_i) = \begin{cases} 1, & x_i \geq y_i \\ 0, & \text{else} \end{cases}
$$

We will attempt to find the maximum likelihood estimates of the parameters λ_0, λ_1 using the expectation-maximization algorithm. We therefore condition the log-likelihood on the values of $\boldsymbol{z} = (z_1, \dots, z_n)^T$ and

$\boldsymbol{u} = (u_1, \ldots, u_n)^T$, in addition to the parameters $\lambda_0^{(t)}$ and $\lambda_1^{(t)}$, which contain the tentative values of the parameters in the algorithm. We take the expectation and find

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

For the term $E[x_i|z_i, u_i, \lambda_0^{(t)}]$, we find

$$\begin{aligned} E[x_i|z_i, u_i, \lambda_0^{(t)}] &= u_i z_i + (1 - u_i)E[x_i|x_i < z_i, \lambda_0^{(t)}] \\ &= u_i z_i + (1 - u_i)\int_{-\infty}^{\infty} x_i P(x_i|x_i < z_i, \lambda_0^{(t)})dx_i \end{aligned}$$

Using Bayes' rule, the probability term inside the integral is

$$\begin{aligned} P(x_i|x_i < z_i, \lambda_0^{(t)}) = \frac{P(x_i < z_i|x_i, \lambda_0^{(t)})P(x_i, \lambda_0^{(t)})}{P(x_i < z_i, \lambda_0^{(t)})} &= \frac{P(x_i, \lambda_0^{(t)})}{P(x_i < z_i, \lambda_0^{(t)})} \\ &= \frac{f(x_i, \lambda_0^{(t)})}{\int_0^{z_i} f(x_i)dx_i} = \frac{f(x_i, \lambda_0^{(t)})}{F(z_i)}, \end{aligned}$$

where $F(\cdot)$ is the cumulative distribution function of the exponential distribution. We find

$$E[x_i|z_i, u_i, \lambda_0^{(t)}] = u_i z_i + (1 - u_i)\frac{1}{F(z_i)}\int_0^{z_i} x_i f(x_i, \lambda_0^{(t)})dx_i$$

Using integration by parts in the integral, we find the expression

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

Similarily, for y_i, we find

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

Putting it all together,

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

3b)

Define $Q(\lambda_0, \lambda_1|\lambda_0^{(t)}, \lambda_1^{(t)}) = \mathrm{E}[\ln(f(\boldsymbol{x}, \boldsymbol{y}|\lambda_0, \lambda_1))|\boldsymbol{z}, \boldsymbol{u}, \lambda_0^{(t)}, \lambda_1^{(t)}]$. In the M-step of the EM algorithm, we want to find $\mathrm{argmax}_{\lambda_0,\lambda_1} Q(\lambda_0, \lambda_1|\lambda_0^{(t)}, \lambda_1^{(t)})$, so we take the partial derivative of Equation 1 w.r.t. λ_0 and λ_1:

$$\begin{aligned} \partial_{\lambda_0} Q(\lambda_0, \lambda_1|\lambda_0^{(t)}, \lambda_1^{(t)}) &= \frac{n}{\lambda_0} - \sum_{i=1}^{n}\left(u_i z_i + (1 - u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{\exp(\lambda_0^{(t)} z_i) - 1}\right)\right) \\ \partial_{\lambda_1} Q(\lambda_0, \lambda_1|\lambda_0^{(t)}, \lambda_1^{(t)}) &= \frac{n}{\lambda_1} - \sum_{i=1}^{n}\left((1 - u_i)z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{\exp(\lambda_1^{(t)} z_i) - 1}\right)\right) \end{aligned}$$

We set the partial derivatives equal to zero and solve for λ_0 and λ_1:

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

The last part of the M-step in the algorithm is to set $\lambda_0 = \lambda_0^{(t+1)}$ and $\lambda_1 = \lambda_1^{(t+1)}$, which is a procedure we repeat until convergence.

We import the data vectors $\boldsymbol{u}$ and $\boldsymbol{z}$.

```r
u_data <- scan(file="https://www.math.ntnu.no/emner/TMA4300/2024v/u.txt")
z_data <- scan(file="https://www.math.ntnu.no/emner/TMA4300/2024v/z.txt")
```

We implement the recursion, and find the maximum likelihood estimates of λ_0 and λ_1.

```r
update_lambda <- function(lambda0,lambda1,z,u){
  n = length(z)
  sum0 = u %*% z + (1-u) %*% (1/lambda0 - z/(exp(lambda0*z)-1))
  sum1 = (1-u) %*% z + u %*% (1/lambda1 - z/(exp(lambda1*z)-1))

  return (list(lambda0_next = n/sum0, lambda1_next = n/sum1))
}

findMLE <- function(lambda0,lambda1,z,u,tol,iterList = 0){
  lambda0_prev = lambda0
  lambda1_prev = lambda1

  est = update_lambda(lambda0_prev,lambda1_prev,z,u)
  iterList = rbind(iterList,c(est$lambda0_next, est$lambda1_next))

  if (abs(lambda0_prev - est$lambda0_next) < tol &&
      abs(lambda1_prev - est$lambda1_next) < tol){
    return (list(lambda0MLE = est$lambda0_next,lambda1MLE = est$lambda1_next,
                 lambdaVals = iterList))
  }
  return (findMLE(est$lambda0_next,est$lambda1_next,z,u,tol,iterList))
}

est = findMLE(1,1,z_data,u_data,1e-8,data.frame(lambda1 = 1, lambda0 = 1))
cat("The estimated value of lambda_0 is ", est$lambda0MLE,
    ", and the estimated value of lambda_1 is ",est$lambda1MLE)
```

```
## The estimated value of lambda_0 is  3.465735 , and the estimated value of lambda_1 is  9.353215
```

We see that the maximum likelihood estimates found by the EM-algorithm are $\hat{\lambda_0} = 3.465735$ and $\hat{\lambda_1} = 9.353215$. The algorithm was terminated when the absolute difference in λ estimates from one iteration to the next was less than 10^{-8}. The initial value for both λ's was set to be 1. We visualize the convergence of the algorithm in Figure 1.

```r
plot(est$lambdaVals[,2],type = "b",col = "red",
     xlab = "Iterations",ylab = "Estimate")
lines(est$lambdaVals[,1],type = "b",col = "darkblue")
legend("bottomright",legend = c(expression(lambda_0),expression(lambda_1)),
       col = c("darkblue","red"),lty = 1,pch = 1)
```

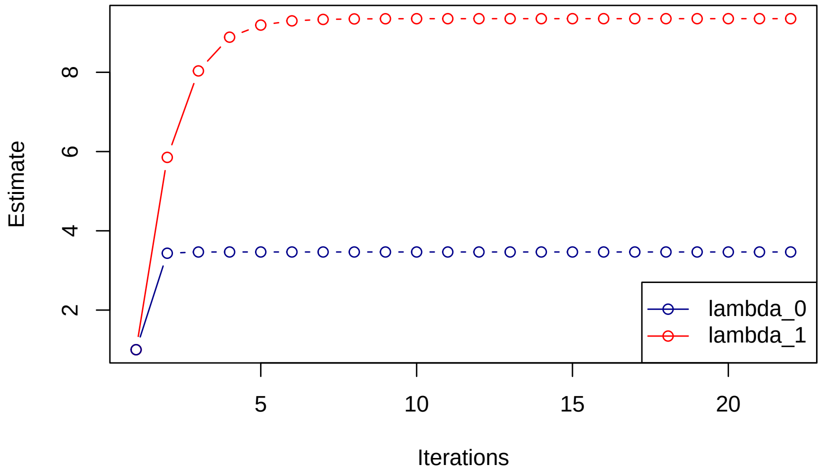

Figure 1: Convergence of λ values in the EM algorithm.

Figure 1 shows that the value of λ_0 stabilized after 1 or 2 iterations, while the value of λ_1 stabilized at around 6 or 7 iterations. In this particular setting, the EM algorithm converged very quickly.

3c)

We turn to bootstrapping in order to estimate the standard deviations and the biases of each of the λ estimates. The bootstrapping algorithm used is presented in short here:

```
1. Input: $\boldsymbol{u}, \boldsymbol{z}$: data vectors of length $n$, $B$: number of bootstrap samples
2. for $1, \dots, B$:
    (a) $u^* \leftarrow n$ samples with replacement from $\boldsymbol{u}$
    (b) $z^* \leftarrow n$ samples with replacement from $\boldsymbol{z}$
    (c) $\lambda_0^*, \lambda_1^* \leftarrow$ bootstrap estimates of $\lambda_0$, $\lambda_1$ from the EM-algorithm using $\boldsymbol{u}^*, \boldsymbol{z}^*$
3. end
```

```r
bootstrapEM <- function(B){
  vals = matrix(NA,nrow = B,ncol = 2)
  n = length(u_data)
  for (i in 1:B){
    idx = sample(1:n,n,replace = T)
    lambdaEst = findMLE(1,1,z_data[idx],u_data[idx],1e-5)
    vals[i,] = c(lambdaEst$lambda0MLE, lambdaEst$lambda1MLE)
  }
  return (vals)
}

set.seed(4)
booty = bootstrapEM(10000)
print("Standard deviation:")
```

```
## [1] "Standard deviation:"
```

```
apply(booty,2,sd) #Standard deviation
```

```
## [1] 0.2464255 0.8031727
```

```
print("Bias:")
```

```
## [1] "Bias:"
```

```
apply(booty,2,mean) - c(est$lambda0MLE,est$lambda1MLE) #Bias
```

```
## [1] 0.01662213 0.08493141
```

```
print("Correlation")
```

```
## [1] "Correlation"
```

```
cor(booty[,1],booty[,2])
```

```
## [1] -0.009880386
```

```
print("Relative size of bias ")
```

```
## [1] "Relative size of bias "
```

```
(apply(booty,2,mean)-c(est$lambda0MLE,est$lambda1MLE))/c(est$lambda0MLE,est$lambda1MLE)
```

```
## [1] 0.004796133 0.009080451
```

We see that for $\hat{\lambda_0}$, the estimated standard deviation is found to be 0.2464 and the estimated bias is found to be 0.01662. For $\hat{\lambda_1}$, the estimated standard deviation is found to be 0.8032, and the estimated bias is found to be 0.08493. The correlation between the vectors containing the estimates for λ_0 and λ_1 was found to be -0.00988, which indicates very little to no correlation, as expected, since $\boldsymbol{X}$ and $\boldsymbol{Y}$ are independent. The bias in both cases are small, amounting to approximately 0.48 of $\hat{\lambda_0}$ and 0.91 of $\hat{\lambda_1}$. We therefore prefer to use the maximum likelihood estimates, since using bias-corrected estimates would increase the variance.

3d)

To find an analytical formula for the joint density $f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1)$, we exploit the fact that $u_i = 1$ when $x_i > y_i$ and $u_i = 0$ when $x_i < y_i$:

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

The likelihood takes the form

$$\begin{aligned}
L(\lambda_0, \lambda_1|\boldsymbol{z}, \boldsymbol{u}) = \prod_{i=1}^{n} f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1) \\
= \prod_{i:u_i=1} u_i(1 - \exp(-\lambda_1 z_i))(\lambda_0 \exp(-\lambda_0 z_i)) \cdot \prod_{i:u_i=0} (1 - u_i)(1 - \exp(-\lambda_0 z_i))(\lambda_1 \exp(-\lambda_1 z_i))
\end{aligned}$$

Taking the logarithm, we find the log-likelihood

$$\begin{aligned}
l(\lambda_0, \lambda_1|\boldsymbol{z}, \boldsymbol{u}) = &\sum_{i:u_i=1} (\ln(\lambda_0) - \lambda_0 z_i + \ln(1 - \exp(-\lambda_1 z_i))) \\
&+ \sum_{i:u_i=0} (\ln(\lambda_1) - \lambda_1 z_i + \ln(1 - \exp(-\lambda_0 z_i)))
\end{aligned}$$

We could take the partial derivatives w.r.t. λ_0, λ_1, and set the resulting terms equal to zero to try and find the maximum likelihood estimates, but the resulting equations have no closed form solution. Instead, we turn to numerical optimization for finding the maximum likelihood estimates. We use the `optim` function with the default gradient approximation, and optimize the negative log-likelihood.

```r
loglik <- function(lambda){
  sum1 = u_data %*% (log(lambda[1]) - lambda[1]*z_data + log(1-exp(-lambda[2]*z_data)))
  sum2 = (1 - u_data) %*% (log(lambda[2]) - lambda[2]*z_data + log(1-exp(-lambda[1]*z_data)))
  return (-(sum1+sum2))
}

optim(c(1,1),loglik)$par
```

```
## [1] 3.465890 9.351103
```

We see that with the initial parameters $(1, 1)$, the optimization procedure finds the maximum likelihood estimators $\hat{\lambda_0} = 3.46589$ and $\hat{\lambda_1} = 9.3511$. Optimizing the likelihood this way has some advantages over using the EM-algorithm. For one, it is usually faster. In our case, we were able to calculate the recursion in the M-step analytically, but if the situation was different we might have had to optimize it numerically. Optimization of the likelihood might also be conceptually more intuitive than using the EM-algorithm, which requires special tailoring for each new problem. The optimization procedure used also has an option to report the estimated Hessian at the maximum likelihood estimates, so we can find the estimated standard errors from this. With the EM-algorithm, we had to use bootstrapping to estimate the standard errors.