National Technical University of Athens

Interdisciplinary Master's Programme in

Data Science and Machine Learning

Computational Statistics and Stochastic Optimization

Semester Assignment

Written by: Christos Nikou
ID: 03400146

email:christosnikou@mail.ntua.gr

June 29, 2022

Contents

1 Stochastic Simulation

1.1 Rejection Sampling

In this task we use the technique of rejection sampling to generate 1000 values from the normal distribution $\mathcal{N}(0,1)$. We denote by $f(x)$ the density function of the normal distribution which is given by

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}},$$

for $x \in \mathbb{R}$. The proposal distribution is the Cauchy distribution with density $g(x) = \frac{1}{\pi(1+x^2)}$, $x \in \mathbb{R}$. In rejection sampling we seek for the optimal $M \geq 1$ such that $f(x) \leq Mg(x)$ for all x for which $f(x) > 0$. Since $g(x) > 0$ for all $x \in \mathbb{R}$, we write

$$f(x) \leq Mg(x) \iff \frac{f(x)}{g(x)} \leq M. \tag{1.1.1}$$

Now, the optimal M^* that satisfies (1.1.1) can be described as

$$M^* = \sup_{x\in\mathbb{R}} \frac{f(x)}{g(x)}. \tag{1.1.2}$$

To determine M^*, we define $h(x) = f(x)/g(x)$. By differentiating h and equating to zero, we obtain

$$\begin{aligned} h'(x) = \left(\frac{f(x)}{g(x)}\right)' &= \frac{f'(x)g(x) - g'(x)f(x)}{g^2(x)} = 0 \\ &\iff f'(x)g(x) = g'(x)f(x). \end{aligned} \tag{1.1.3}$$

Substituting f, g and the derivatives

$$f'(x) = -\frac{x}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \quad g'(x) = -\frac{2x\pi}{\pi^2(1+x^2)^2} \tag{1.1.4}$$

in (1.1.3) we get that

$$h'(x) = 0 \iff -\frac{x}{\cancel{\sqrt{2\pi}}} \cancel{e^{-\frac{x^2}{2}}} \frac{1}{\cancel{\pi(1+x^2)}} = -\frac{2\cancel{x}\pi}{\pi^{\cancel{2}}(1+x^2)^{\cancel{2}}} \frac{1}{\cancel{\sqrt{2\pi}}} \cancel{e^{-\frac{x^2}{2}}} \tag{1.1.5}$$

$$\iff 1 = \frac{2\cancel{\pi}}{\cancel{\pi}(1+x^2)} \iff 1 + x^2 = 2 \iff x = \pm 1. \tag{1.1.6}$$

Hence, h has two stationary points; 1 and -1. To examine whether these points are local maximums we examine the sign of the derivative h'. By the calculations in (1.1.5) and (1.1.6) we have that

$$\begin{aligned} h'(x) > 0 &\iff 1 + x^2 > 2 \\ &\iff x^2 > 1 \iff x \in (-\infty, -1) \cup (1, +\infty). \end{aligned}$$

Therefore, we deduce that h is strictly increasing on $(-\infty,-1) \cup (1,+\infty)$ and strictly decreasing on $(-1,1)$. Hence, the stationary points 1,-1 are the only possible global maximums of h on $\mathbb{R}$. By observing that h is symmetric; i.e. $h(x) = h(-x)$ for all $x \in \mathbb{R}$ we get that $h(1) = h(-1)$. After these considerations, we can write (1.1.2) as

$$\begin{aligned} M^* = \max_{x\in\mathbb{R}} \frac{f(x)}{g(x)} = \frac{f(1)}{g(1)} &= \frac{2\pi}{\sqrt{2\pi}} e^{-1/2} \\ &= \sqrt{\frac{2\pi}{e}} \approx 1.5203. \end{aligned}$$

Using *ggplot2* in R and the following commands we can produce the graph of the function h and verify our theoretical findings.

```r
# Exercise 1 - Stochastic Simulation
# Importing libraries
library(data.table)
library(ggplot2)
library(latex2exp) # For LaTeX commands in plot titles
M <- sqrt(2*pi/exp(1)) # Optimal M
# Graph of h
dt <- data.table(x=-5:5)
ggplot(dt,aes(x))+
  stat_function(fun = function(x) dnorm(x)/dcauchy(x),
                colour = "blue", size = 1)+ylab("h(x)")+
  labs(title = TeX("The graph of function $h(x)=f(x)/g(x)$"))
```

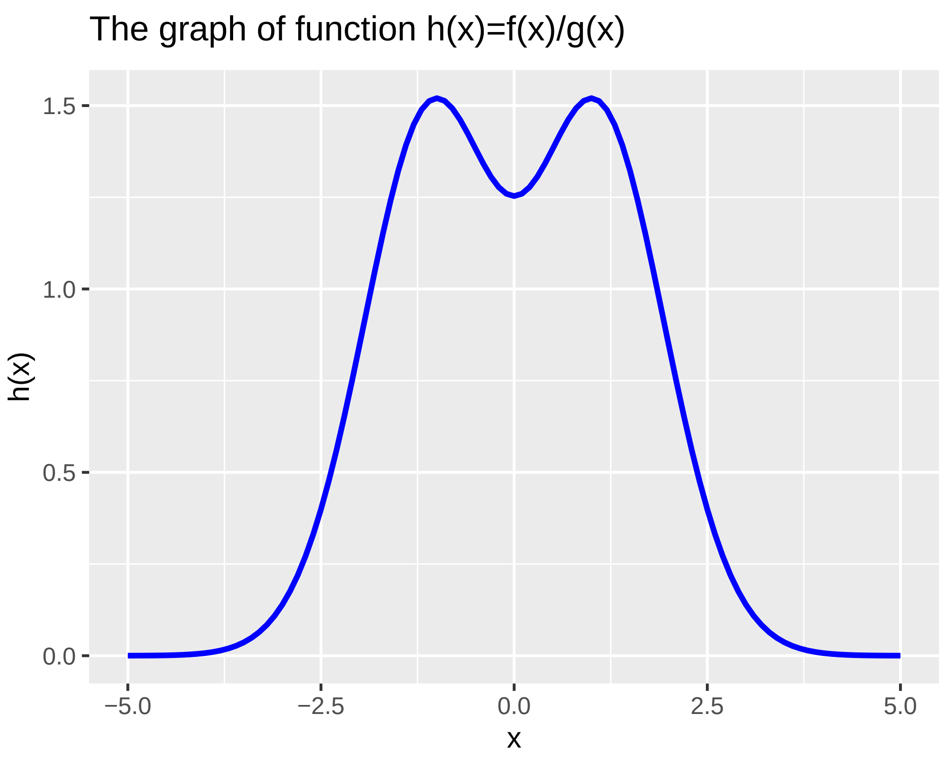

Figure 1: The graph of the function $h(x) = f(x)/g(x)$.

In what follows we denote by X the random variable with $X \sim \mathcal{N}(0, 1)$ and by Y the random variable with the Cauchy distribution. To simulate from $\mathcal{N}(0, 1)$ using the rejection sampling technique we apply the following algorithm:

Algorithm 1 Rejection Sampling Algorithm

```
1: Generate $Y \sim g(y)$ and set $y = Y$.
2: Generate $U \sim U(0,1)$ and set $u = U$
3: if $u \leq \frac{f(y)}{Mg(y)}$ then
4:     $X = y$
5: else
6:     go back to Step 1.
7: end if
```

Now, to sample from Y in step 1 in Algorithm 1 we use the inversion method. To this end we first calculate the inverse function of the Cauchy distribution G. For this, we have that

$$G(x) = \int_{-\infty}^{x} \frac{1}{\pi(1+t^2)}\, dt = \frac{1}{\pi} \arctan t \Big|_{-\infty}^{x} = \frac{1}{\pi}\Big(\arctan(x) + \frac{\pi}{2}\Big).$$

Therefore, to find the the inverse function G^{-1} we write

$$u = \frac{1}{\pi}\arctan(x) + \frac{1}{2} \iff \arctan(x) = \pi(u - \frac{1}{2}). \tag{1.1.7}$$

To solve for x in (1.1.7) the quantity $\pi(u - 1/2)$ must lie in the domain of $\tan u$ which is $\big(-\pi/2, \pi/2\big)$. Solving the following inequality

$$\begin{aligned} -\frac{\pi}{2} < \pi(u - \frac{1}{2}) < \frac{\pi}{2} &\iff -\frac{1}{2} < u - \frac{1}{2} < \frac{1}{2} \\ &\iff 0 < u < 1, \end{aligned}$$

we find that we can solve (1.1.7) for every $u \in (0,1)$. Hence, solving for x in (1.1.7) we have that

$$\arctan(x) = \pi(u - \frac{1}{2}) \implies x = \tan\Big(\pi(u - \frac{1}{2})\Big),$$

which in turn implies that $G^{-1}(u) = \tan\big(\pi(u - \frac{1}{2})\big)$ for every $u \in (0,1)$. Now, to sample from Y we draw a random point u from $U \sim U(0,1)$ and we set $y = \tan\big(\pi(u - \frac{1}{2})\big)$. Below we present the implementation of Algorithm (1.1.7) in R using the inverse method to sample from the Cauchy distribution. In order to produce the same output across multiple runs we set a random seed equal to 42. Moreover, we count the trials and rejections made by the algorithm for the condition 3 in Algorithm 1 and estimate the probability of acceptance by the ratio (1-rejections)/trials.

```r
samples <- 1000 # Number of samples to generate

rejection_simulation <- function(num_samples, M,seed = 42){
  set.seed(seed) # Set seed for reproducibility
  sampled <- 0 # Current number of samples
  rejections <- 0
  trials <- 0
  samples = c()
  while (sampled < num_samples){
    accept = FALSE
    while(!accept){
      u <- runif(2)
      y <- tan(pi*(u[1]-1/2)) # Inversion sampling
      trials <- trials +1
      if (M*dcauchy(y)*u[2] <= dnorm(y)){
        accept = TRUE
        samples <- append(samples,y)
        sampled <- sampled +1
      }
      else{rejections <- rejections + 1}
    }
  }
  info <- c("samples", "trials", "rejections")
  l <- list(samples = samples, trials = trials,
            rejections = rejections, info = info)
  return(l)}
l <- rejection_simulation(num_samples = 1000, M = M)
print(paste("Probability of acceptance: ", 1-l$rejections/l$trials))

## [1] "Probability of acceptance:  0.659630606860158"

print(paste("Theoretical acceptance: ", 1/M))

## [1] "Theoretical acceptance:  0.657744623479457"
```

As we can see from the above output, the probability of acceptance of the algorithm is 0.659630 while the theoretical probability is equal to $1/M = 0.657744$. Now, to determine the average number of trials for an acceptance we introduce a new variable N to count the number of trials till the first accepted step. Since the the probability of acceptance is equal to $p = 1/M \approx 0.6577$, then N is just a geometric distribution with probability of success $p = 1/M$, i.e. $N \sim \text{Geom}(p)$, with support $\{1, 2, \dots\}$. Therefore, the average number of trials for an acceptance is equal to $\mathbb{E}(N) = 1/p \approx 1.5203469$. Below we plot a histogram of the samples obtained from the rejection sampling. For the histogram we use the optimal bin width $h_{\text{opt}} = 3.491n^{-1/3}$ for the normal distribution $\mathcal{N}(0, 1)$, proven in the lectures of this course.

```
# Visualize the results of sampling
h_opt <- 3.491/(1000)^(1/3) # optimal bin-width for Normal Distribution
dataset <- data.frame(X = get("samples",l))
m <- ggplot(dataset, aes(x=X))
m + geom_histogram(aes(y = ..density..),binwidth = h_opt,
                   color = "#24438C",fill = "#77AABB")+
  geom_density(alpha = 0.5, size = 1, aes(color = "Estimated Density"),
               key_glyph = draw_key_path)+
  stat_function(fun = dnorm,size = 1.2, aes(color = "Normal PDF"),
  key_glyph =draw_key_path)+
  labs(title = TeX("Histogram of samples generated from $N(0,1)$"),
       subtitle = "Method: Rejection sampling")+
  scale_colour_manual("Densities",values = c("blue","black"))+
  theme(legend.position = c(0.85, 0.85))
```

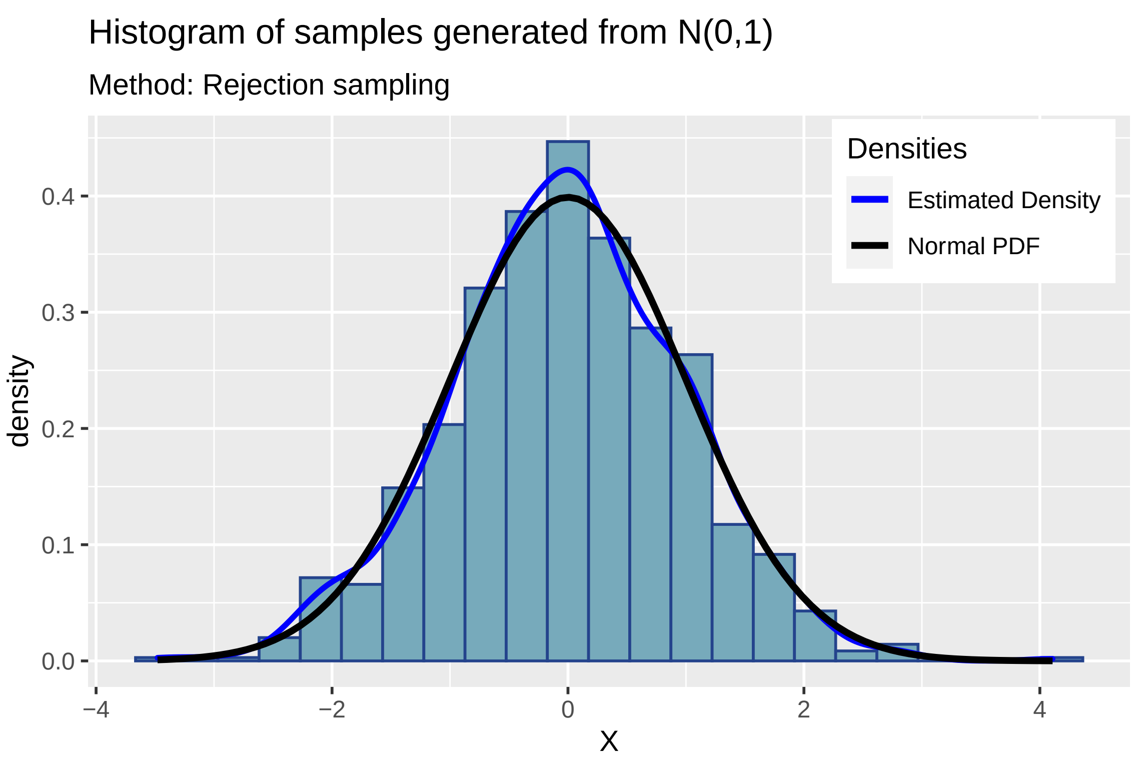

Figure 2: Comparison of the estimated distribution obtained using rejection sample and the actual distribution of $N(0,1)$.

Figure 2 shows that the estimated density obtained from the samples is almost identical to the actual normal distribution. With the following commands we calculate the sample mean and standard deviation.

```r
print(paste("- Estimated mean:", mean(l$samples)))

## [1] "- Estimated mean: -0.0126922109893221"

print(paste("- Estimated standard deviation", sd(l$samples)))

## [1] "- Estimated standard deviation 0.990052838995922"
```

1.2 Rao - Blackwell estimator

1.2.1 Theoretical results

In this task we use the Rao - Blackwellized form of the sample mean $\bar{X}$ to obtain an estimator for the negative binomial with lower variance. More precisely, let $X \sim \text{NB}(r,p)$ be a random variable from the negative binomial distribution with size r and probability p. Here we assume that X represents the number of failures which occur in a sequence of Bernoulli trials before the r-th success. The probability mass function in this case is given by

$$f_X(k;r,p) = \mathbb{P}(X=k) = \frac{\Gamma(k+r)}{k!\Gamma(r)}(1-p)^k p^r, \tag{1.2.1}$$

for $k = 0, 1, 2, \ldots$. The mean and variance are equal to

$$\mathbb{E}(X) = \frac{r(1-p)}{p}, \quad \operatorname{Var}(X) = \frac{r(1-p)^2}{p^2}. \tag{1.2.2}$$

The sample mean estimator is given by

$$\delta_n = \frac{(X_1 + \cdots + X_n)}{n},$$

where $\mathbf{X} = (X_1, \ldots, X_n)$ are i.i.d from $\mathrm{NB}(r, p)$. Using the linearity of the expected value and the i.i.d assumption we calculate the expected value of $\bar{X}$ as

$$\begin{aligned}\mathbb{E}(\delta_n) &= \mathbb{E}\left(\sum_{k=1}^{n} \frac{X_k}{n}\right) = \frac{1}{n}\sum_{k=1}^{n}\mathbb{E}(X_k) \\ &= \frac{1}{n}\sum_{k=1}^{n}\mathbb{E}(X) = \frac{\cancel{n}}{\cancel{n}}\mathbb{E}(X) = \frac{r(1-p)}{p}.\end{aligned} \tag{1.2.3}$$

Similarly, for the variance we have that

$$\begin{aligned}\operatorname{Var}(\delta_n) &= \operatorname{Var}\left(\frac{1}{n}\sum_{k=1}^{n} X_k\right) = \frac{1}{n^2}\operatorname{Var}\left(\sum_{k=1}^{n} X_k\right) \\ &= \frac{1}{n^2}\left(\sum_{k=1}^{n}\operatorname{Var}(X_k) + 2\sum_{i\neq j}\operatorname{Cov}(X_i, X_j)\right).\end{aligned}$$

Now, the fact that X_i, X_j are independent for $i \neq j$ implies that $\operatorname{Cov}(X_i, X_j) = 0$ for $i \neq j$. Hence

$$\operatorname{Var}(\delta_n) = \frac{1}{n^2}\sum_{k=1}^{n}\operatorname{Var}(X_k) = \frac{1}{n}\operatorname{Var}(X) = \frac{1}{n}\frac{r(1-p)}{p^2}. \tag{1.2.4}$$

A negative binomial distribution can also arise as a mixture of a Poisson $X \mid \Lambda = \lambda \sim \text{Poisson}(\lambda)$ distribution with mean λ distributed as a gamma distribution $\Lambda \sim \text{Gamma}\big(r, \frac{1-p}{p}\big)$, with scale parameter $(1-p)/p$ and shape parameter r. The density function of Λ is given by

$$f_\Lambda(x) = \frac{p^r}{\Gamma(r)(1-p)^r}x^{r-1}e^{-\frac{px}{1-p}}, \quad x > 0, \tag{1.2.5}$$

with $\mathbb{E}(\Lambda) = \frac{r(1-p)}{p}$ and $\operatorname{Var}(\Lambda) = \frac{r(1-p)^2}{p^2}$. The conditional probability mass function $f_{X \mid \Lambda}$ is given by

$$f_{X \mid \Lambda}(x \mid \lambda) = \begin{cases} \frac{f(x,\lambda)}{f_\Lambda(\lambda)}, & f_\Lambda(\lambda) > 0 \\ 0, & f_\Lambda(\lambda) = 0 \end{cases}. \tag{1.2.6}$$

But, since $f_\Lambda(\lambda) > 0$ for every $\lambda > 0$ we can write the conditional probability mass function as $\frac{f(x,\lambda)}{f_\Lambda(\lambda)}$. Now, using the fact that $X \mid \Lambda = \lambda \sim \text{Poisson}(\lambda)$ we have that

$$f_{X \mid \Lambda}(x \mid \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad x = 0, 1, \ldots, \tag{1.2.7}$$

with $\mathbb{E}(X \mid \Lambda = \lambda) = \lambda$ and $\operatorname{Var}(X \mid \Lambda = \lambda) = \lambda$. Combining (1.2.6), (1.2.7) we get that

$$\begin{aligned}f(x, \lambda) &= f_{X \mid \Lambda}(x \mid \lambda) \cdot f_\Lambda(\lambda) \\ &= \frac{\lambda^x e^{-\lambda}}{x!} \cdot \frac{p^r}{\Gamma(r)(1-p)^r}\lambda^{r-1}e^{-\frac{p\lambda}{1-p}}, \quad x \in \mathbb{N}_0,\ \lambda > 0.\end{aligned} \tag{1.2.8}$$

Therefore, integrating with respect to λ,

$$\begin{aligned}
\int_0^\infty f(x,\lambda)\,d\lambda &= \int_0^\infty \frac{\lambda^x e^{-\lambda}}{x!}\cdot\frac{p^r}{\Gamma(r)(1-p)^r}\lambda^{r-1}e^{-\frac{p\lambda}{1-p}}\,d\lambda \\
&= \frac{1}{x!\Gamma(r)}p^r(1-p)^x\int_0^\infty \frac{\lambda^{x+r-1}}{(1-p)^{x+r}}e^{\cancelto{-\frac{\lambda}{1-p}}{-\lambda-\frac{p\lambda}{1-p}}}\,d\lambda \\
&= \frac{\Gamma(x+r)}{x!\Gamma(r)}p^r(1-p)^x\cancelto{1}{\int_0^\infty \underbrace{\frac{\lambda^{x+r-1}}{\Gamma(x+r)(1-p)^{x+r}}e^{-\frac{\lambda}{1-p}}}_{\text{Gamma}(x+r,1-p)}\,d\lambda} \\
&= \frac{\Gamma(x+r)}{x!\Gamma(r)}p^r(1-p)^x,
\end{aligned}$$

we observe that the marginal probability mass function of (X,Λ) is equal to the probability mass function of $f_X(x;r,p)$ as described in (1.2.1)[1]. Then, we know from the course lectures that if $\mathbf{X}=(X_1,\ldots,X_n)$, $\mathbf{\Lambda}=(\Lambda_1,\ldots,\Lambda_n)$ are two random samples from X,Λ, respectively, then the estimator $\delta_n^*=\mathbb{E}_{X\,|\,\Lambda}(\delta_n\,|\,\mathbf{\Lambda})$ dominates δ_n in terms of variance, while the bias is the same. In this special case, we can compute δ_n^* explicitly by writing

$$\begin{aligned}
\delta_n^* = \mathbb{E}_{X\,|\,\Lambda}(\delta_n\,|\,\mathbf{\Lambda}) &= \mathbb{E}_{X\,|\,\Lambda}\left(\frac{1}{n}\sum_{k=1}^n X_k\,|\,\Lambda_k\right) \\
&\overset{\text{linearity}}{=} \frac{1}{n}\sum_{k=1}^n \mathbb{E}\Big(X_k\,|\,\Lambda_k\Big) \\
&\overset{X_k\sim X\text{i.i.d}}{=} \frac{1}{n}\sum_{k=1}^n \mathbb{E}\Big(X\,|\,\Lambda_k\Big),
\end{aligned}$$

and using the fact that for fixed $\Lambda=\lambda$ we have $X\,|\Lambda_i\sim\text{Poisson}(\lambda)$ we conclude that the random variable $\mathbb{E}(X\,|\,\Lambda_i)$ is equal to Λ_i. In other words, $\delta_n^*=n^{-1}\sum_{i=1}^n\Lambda_i$. Now, its easy to calculate explicitly the mean and variance of the estimator δ_n^*. For the mean we write

$$\begin{aligned}
\mathbb{E}(\delta_n^*) = \mathbb{E}\left(\frac{1}{n}\sum_{k=1}^n\Lambda_k\right) &= \frac{1}{n}\sum_{k=1}^n\mathbb{E}(\Lambda_k) \\
&= \frac{1}{n}\sum_{k=1}^n\mathbb{E}(\Lambda) = \mathbb{E}(\Lambda) = \frac{r(1-p)}{p^2},
\end{aligned}$$

and for the variance, using the i.i.d assumption of $\mathbf{\Lambda}=(\Lambda_1,\ldots,\Lambda_n)$ we can write

$$\begin{aligned}
\text{Var}(\delta_n^*) = \text{Var}\left(\frac{1}{n}\sum_{k=1}^n\Lambda_k\right) &= \frac{1}{n^2}\text{Var}\left(\sum_{k=1}^n\Lambda_k\right) \\
&= \frac{1}{n^2}\left(\sum_{k=1}^n\text{Var}(\Lambda_k)+2\sum_{i\neq j}\cancelto{0}{\underbrace{\text{Cov}(\Lambda_i,\Lambda_j)}_{\text{independence}}}\right) \\
&= \frac{1}{n^2}\sum_{k=1}^n\text{Var}(\Lambda_k)\underset{\text{distributed}}{\overset{\text{identically}}{=}}\frac{1}{n}\text{Var}(\Lambda) = \frac{1}{n}\frac{r(1-p)^2}{p}. \qquad (1.2.9)
\end{aligned}$$

[1]The same result could also be obtained using the Law of total probability.

Therefore, by (1.2.4) and (1.2.9) we see that $\mathrm{Var}(\delta_n^*) = (1-p)\mathrm{Var}(\delta_n)$, hence the variance is reduced by a factor of $0 < 1-p < 1$. Now, to produce a sample of (X_i, Λ_i) by (1.2.8) its suffices to produce a sample $\mathbf{\Lambda} = (\Lambda_1, \dots, \Lambda_n)$, where $\Lambda \sim \mathrm{Gamma}(r, \frac{1-p}{p})$ and then produce the sample $\mathbf{X} = (X_1, \dots, X_n)$ by using $X \mid \Lambda = \lambda \sim \mathrm{Poisson}(\lambda)$.

1.2.2 Implementation - Experiments

Having developed the theory around the estimator δ_n and its Blackwellized form δ_n^* we now experimentally verify our theoretical results. To this end, we begin by defining a function named *conditional_sampling* to sample from (X_i, Λ_i) using the conditional relation $\Lambda \sim \mathrm{Gamma}(r, \frac{1-p}{p})$, $X \mid \Lambda = \lambda \sim \mathrm{Poisson}(\lambda)$. The function's arguments are the number of samples n to be returned, the parameters r, p which are initialized to $r = 1, p = 0.5$, and a random state which is initialized to 42 for reproducible code. The function returns a list which contains the samples from Λ and the samples from X.

```r
conditional_sampling <- function(n=5000,r=1,p=0.5,seed = 42){
  set.seed(seed)
  lambdas <- rgamma(n=n,shape = r, scale = (1-p)/p) # Sample lambdas
  x <- rep(0,n) # initialize an array of length n to store x values
  for(i in 1:n){ # conditional sampling from Poisson
    x[i] <- rpois(1,lambdas[i])
  }
  info <- c("x", "lambdas")
  l <- list(info = info, x = x,
            lambdas = lambdas)
  return (l)}
```

With the following block of code we generate samples of size $20, 30, 40, \dots, 5000$ iteratively, using the function *conditional_sampling*. For each iteration, we keep track of the respective sampling mean values for λ's and x's. By the law of large numbers we expect to have (with probability 1)

$$\delta_n \xrightarrow{n\to\infty} \mathbb{E}(X) = \frac{r(1-p)}{p} = 1, \quad \delta_n^* \xrightarrow{n\to\infty} \mathbb{E}(X) = 1, \tag{1.2.10}$$

but since the variance of the estimator δ_n^* is lower than δ_n we expect from the former to have milder fluctuations around the limit of convergence $\mathbb{E}(X) = 1$. Below we see the corresponding code in *R* and in Figure 3 the convergence described in (1.2.10)

```r
max_sample <- 5000 # maximum number of samples
num_samples <- seq(20,max_sample,10) # generate the sequence 20,30,...,5000
iterations <- length(num_samples)
mean_x <- rep(0,iterations) # array to store the mean values of X
mean_lambdas <- rep(0,iterations) # array to store the mean values of lambdas

for(i in 1:iterations){ # Iterative application of conditional_sampling
  samples <- conditional_sampling(n=num_samples[i])
  mean_x[i] <- mean(samples$x)
  mean_lambdas[i] <- mean(samples$lambdas)
}
# Merge results to data tables for grouped visualization
```

```r
d1 <- data.table(values = mean_lambdas, Mean = rep("lambdas", iterations),
                 Iterations = num_samples)
d2 <- data.table(values = mean_x, Mean = rep("X's", iterations),
                 Iterations = num_samples)
dt <- rbindlist(list(d1,d2))
# Plot the Figure
ggplot(data = dt, aes(x= Iterations, y = values, color = Mean))+
  geom_line(size = 0.4)+
  labs(title = TeX("The converge of $\\delta_n$ and $\\delta_n^*$ to $E(X)=1$."),
       x = "Number of samples", y = "Mean value")+
  scale_colour_manual("Mean of",values = c("#4FA3AB","#F07B7B"))+
  theme(legend.position = c(0.88, 0.86))+
  scale_fill_discrete(labels=c("Delta star",'Delta'))
```

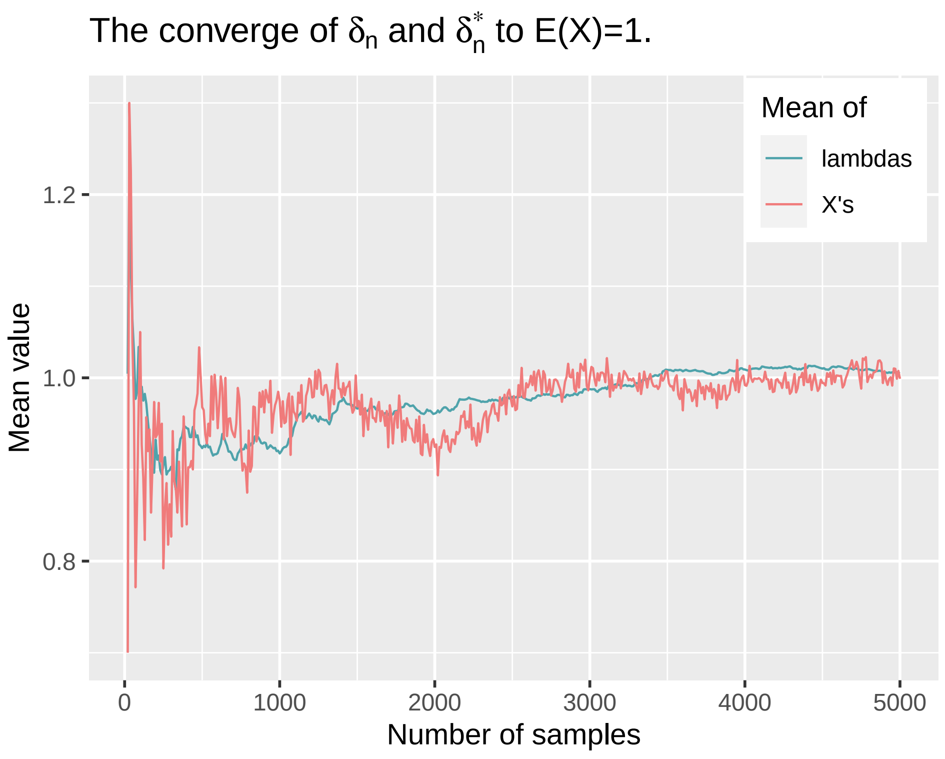

Figure 3: The convergence of δ_n and δ_n^* to $\mathbb{E}(X) = 1$. The variances are equal to $\text{Var}(\delta_n) = \frac{2}{n}$ and $\text{Var}(\delta_n^*) = \frac{1}{n}$. The convergence of δ_n has more rapid fluctuations than δ_n^* due to its higher variance.

Below we generate two sequences of 5000 samples, one from $X \sim \text{NB}(1, 0.5)$ and one from $\Lambda \sim \text{Gamma}(1, 1)$ and we compare the sample variances using the function *var* from R. We expect the variance of the samples of X's to be close to 2 and from Λ's close to 1. Below you can see the code and corresponding results printed on screen.

```r
# Comparison of variances
samples <- conditional_sampling(n=5000)
sprintf("- The variance of samples from X is %.3f",var(samples$x))

## [1] "- The variance of samples from X is 2.039"
```

```
sprintf("- The variance of samples from L is %.3f",var(samples$lambdas))

## [1] "- The variance of samples from L is 1.031"
```

In the last experiment we generate a sequence of samples from both estimators δ_n, δ_n^* and we compare with our theoretical findings in 1.2.1. More precisely, for a fixed number $n = 5000$ by iterating over the set of integers $I = \{20, 21, \ldots, 3000\}$ we generate for every $i \in I$ two sequence of points $(\delta_{1,5000}, \ldots, \delta_{i,5000})$ and $(\delta^*_{1,5000}, \ldots \delta^*_{i,5000})$ from δ_{5000} and δ^*_{5000}, respectively, and we calculate their variance by using the command *var* in R. Then, for each $i \in I$ we obtain two estimates v_i and v_i^* for $\text{Var}(\delta_{5000})$ and $\text{Var}(\delta^*_{5000})$, respectively. By (1.2.4) and (1.2.9) we expect to have for each i

$$v_i \approx \frac{2}{5000} = 4 \times 10^{-4}, \quad v_i^* \approx \frac{1}{5000} = 2 \times 10^{-4}. \tag{1.2.11}$$

In Figure 4 you can see the result of this experiment. Below is the code that was developed in R for this experiment.

```
n <- 5000
p <- 0.5
r <- 1
sequence <- seq(20,3000,1)
iterations <- length(sequence)
delta <- rep(0,3000)
delta_star <- rep(0,3000)
var_delta <- rep(0,iterations)
var_delta_star <- rep(0, iterations)

for(i in 1:19){
  delta_star[i] <- mean(rgamma(n = n, shape = r,
                              scale = (1-p)/p))
  delta[i] <- mean(rnbinom(n=n, size = r,
                          prob = p))
}

for(i in 1:iterations){
  delta_star[sequence[i]] <- mean(rgamma(n = n, shape = r,
                                        scale = (1-p)/p))
  delta[sequence[i]] <- mean(rnbinom(n=n, size = r,
                                    prob = p))
  var_delta[i] <- var(delta[1:sequence[i]])
  var_delta_star[i] <- var(delta_star[1:sequence[i]])
}
d1 <- data.table(values = var_delta,
                 Mean = rep("Simulated delta",iterations),
                 sample_size = sequence)
d2 <- data.table(values = rep((1/n)*r*(1-p)/p^2,iterations),
                 Mean = rep("Delta"),
                 sample_size = sequence)
d3 <- data.table(values = var_delta_star,
                 Mean = rep("Simulated delta star",
                            iterations),
                  sample_size = sequence)
d4 <- data.table(values = rep((1/n)*r*(1-p)^2/p^2,iterations),
                  Mean = rep("Delta star"),
                  sample_size = sequence)
dt <- rbindlist(list(d1,d2,d3,d4))
ggplot(data = dt, aes(x= sample_size, y = values, color = Mean))+
  geom_line(size = 0.4)+
  labs(title = TeX("Simulation: Variance of $\\delta_n$ and $\\delta_n^*$."),
                    subtitle = TeX("$n=5000$"),
       x = "Number of samples", y = "Variance")+
  scale_colour_manual("Variance of",values = c("#4FA3AB","#4FA3AB",
                                                "#F07B7B", "#7AB163"))+
  theme(legend.position = c(0.82, 0.85))+
  scale_fill_discrete(labels=c(TeX("Delta"), 'Delta Star',
                                "Simulated delta",
                                "Simulated delta star"))+
    theme(legend.key.height= unit(0.3, 'cm'),
          legend.key.width= unit(0.5, 'cm'))
```

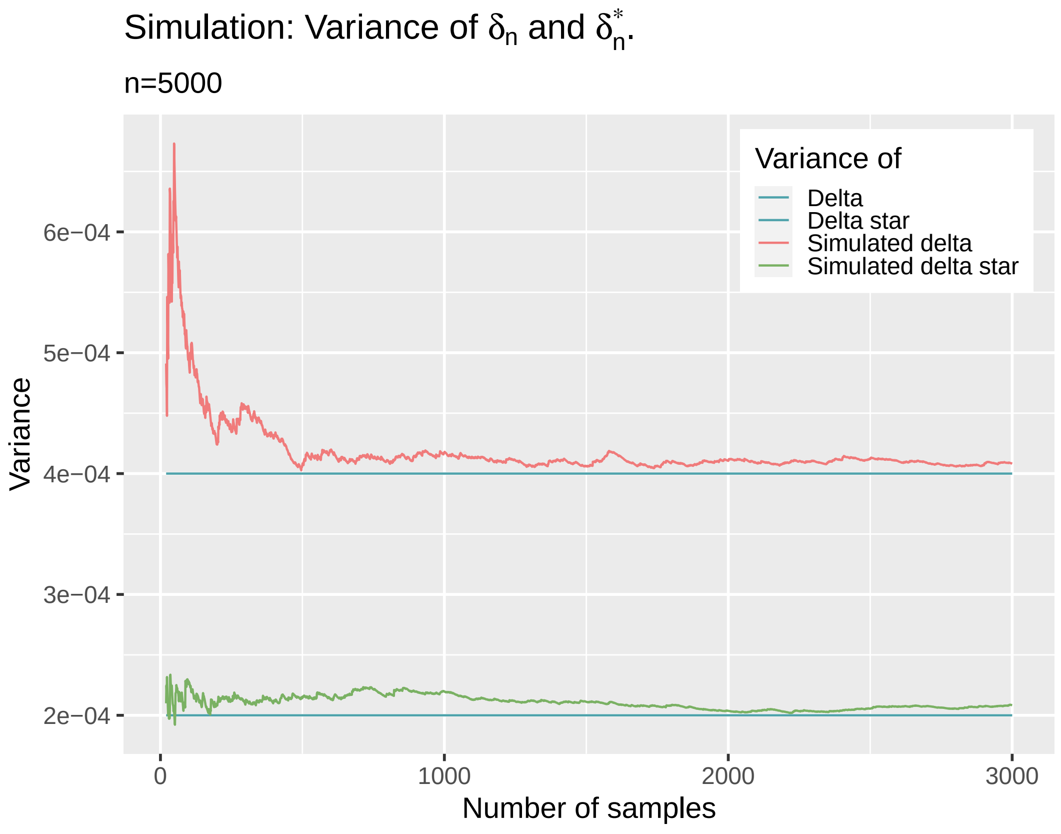

Figure 4: Simulation to verify the theoretical findings. A number of samples from 20 to 3000 for both δ_{5000} and δ^*_{5000} are generated. The sampling variances for the two generated sequences are expected to approximate the true variances $\text{Var}(\delta_{5000}) = 4 \times 10^{-4}$ and $\text{Var}(\delta^*_{5000}) = 2 \times 10^{-2}$.

1.3 T simulation

The next simulation task refers to the estimator

$$T_n = \frac{(X_1 + \cdots + X_n)^2}{n}, \quad X_i \sim U(0,1). \tag{1.3.1}$$

Our goal is to compute the expected value of T_n and compare the result with the mean value of a sample from (1.3.1) with $n = 10000$. In addition, we plot a histogram of the obtained sample and calculate its standard deviation. We first begin with the theoretical part where we calculate the expected value of T_n and then compare the theoretical result with the experimental results.

1.3.1 Calculating the expected value

To calculate the expected value we first write

$$nT_n = \left(\sum_{k=1}^{n} X_k\right)^2 = \sum_{k=1}^{n} X_k^2 + 2\sum_{i \neq j} X_i X_j.$$

Therefore, since $X_1, \ldots, X_n$ are i.i.d, using the linearity of the expected value we have

$$\begin{aligned}
\mathbb{E}(T_n) &= \mathbb{E}\left(\sum_{k=1}^{n} X_k^2 + 2\sum_{i \neq j} X_i X_j\right) \\
&= \mathbb{E}\left(\sum_{k=1}^{n} X_k^2\right) + 2\,\mathbb{E}\left(\sum_{i \neq j} X_i X_j\right) \\
&= \sum_{k=1}^{n} \mathbb{E}(X_k^2) + 2\sum_{i \neq j} \mathbb{E}(X_i X_j) \\
&\overset{\text{i.i.d}}{=} n\,\mathbb{E}(X^2) + 2\sum_{i \neq j} \mathbb{E}(X_i)\,\mathbb{E}(X_j).
\end{aligned} \tag{1.3.2}$$

Now, on the the second summand in (1.3.2) we have $\binom{n}{2}$ pairs, hence (1.3.2) is equivalent to

$$\begin{aligned}
n\,\mathbb{E}(T_n) &= n\,\mathbb{E}(X^2) + 2\binom{n}{2}\,\mathbb{E}^2(X) \\
&= n\,\mathbb{E}(X^2) + \cancel{2}\frac{n!}{(n-2)!\cancel{2!}}\,\mathbb{E}^2(X) \\
&= n\,\mathbb{E}(X^2) + n(n-1)\,\mathbb{E}^2(X),
\end{aligned} \tag{1.3.3}$$

where $X \sim U(0,1)$. To calculate $\mathbb{E}(X)$, $\mathbb{E}(X^2)$ we have that the density function of X is given by

$$f_X(x) = \begin{cases} 1, & 0 \le x \le 1 \\ 0, & |x| > 1 \end{cases}.$$

Therefore,

$$\mathbb{E}(X) = \int_0^1 x\,dx = \frac{x^2}{2}\bigg|_0^1 = \frac{1}{2},$$

and

$$\mathbb{E}(X^2) = \int_0^1 x^2\,dx = \frac{x^3}{3}\bigg|_0^1 = \frac{1}{3}.$$

Hence, substituting $\mathbb{E}(X^2) = 1/3$, $\mathbb{E}^2(X) = 1/4$ in (1.3.3) we conclude that

$$\begin{aligned} n\,\mathbb{E}(T_n) &= \frac{n}{3} + \frac{n^2}{4} - \frac{n}{4} \implies \\ \mathbb{E}(T_n) &= \frac{n}{4} + \frac{1}{12}. \end{aligned}$$

In particular, for a sample of size $n = 80$ we have that $\mathbb{E}(T_{80}) \approx 20.0833333$.

1.3.2 Implementation - Experiments

For the implementation we generate a sample of 10000 values from (1.3.1) for $n = 80$ and by using the commands *mean, var* we estimate the sampling mean and standard deviation of the obtained sequence of 10000 points. To sample from $U(0,1)$ we use the function *runif* provided by R. Moreover, we compare the estimated mean with the true mean given by $\mathbb{E}(T_{80}) \approx 20.0833333$. Below you can see the results and the corresponding code produced in R.

```
# Function to simulate from T
T_simulation <- function(n=80, num_samples=10000, seed = 42){
  set.seed(seed) # Set seed to 42, reproducibility
  samples <- c() # Samples from T_80
  for(i in 1:num_samples){
    samples <- append(samples, sum(runif(n))^2/n)
  }
  estimated_mean <- mean(samples)
  estimated_sd <- sd(samples)
  true_mean <- n/4+1/12
  info <- c("estimated mean", "estimated sd", "samples", "true mean")
  l <- list(estimated_mean = estimated_mean,
            estimated_sd = estimated_sd, samples = samples,
            info = info, true_mean = true_mean)
}
l <- T_simulation()
print(paste("- Sample mean:", l$estimated_mean))

## [1] "- Sample mean: 20.0877226309024"

print(paste("- True mean: ", l$true_mean))

## [1] "- True mean:  20.0833333333333"

print(paste("- Estimated standard deviation: ", l$estimated_sd))

## [1] "- Estimated standard deviation:  2.59046906669014"
```

As we can see the sample mean is equal to 20.087 which is close to the true mean 20.0833333. The standard deviation is equal to 2.59. With the following commands we generate a histogram for the values obtained from the sampling.

```r
# Histogram of samples
data <- data.table(samples = l$samples)
ggplot(data = data, aes(x = samples))+
  geom_histogram(aes(x = samples),bins=30,
                 color = "#24438C",fill = "#77AABB")+
  labs(title = TeX("Histogram for 10000 samples from $T_{80}$."),
       x=TeX("Value of $T_{80}$"))
```

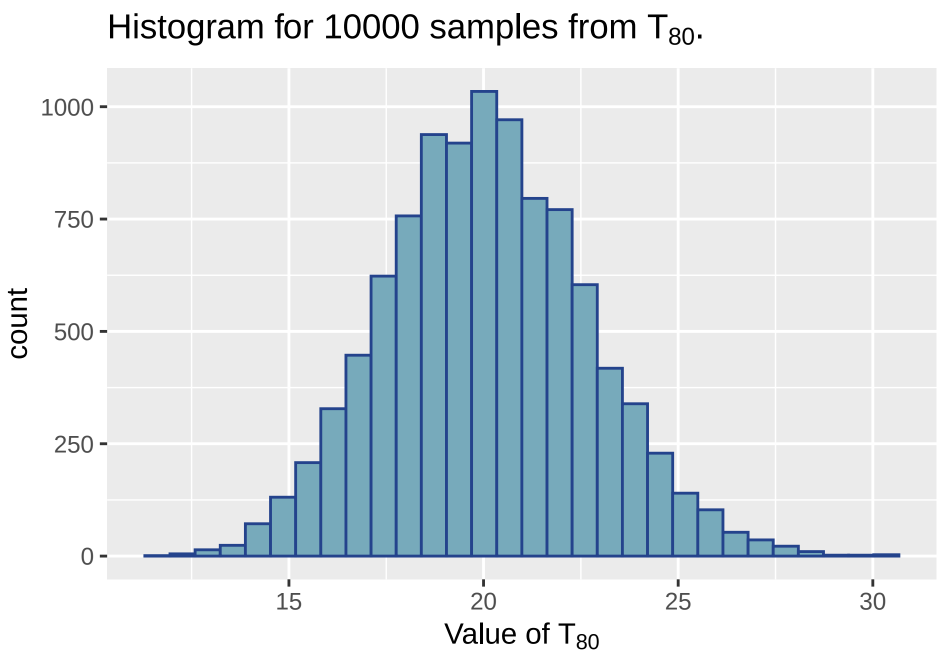

Figure 5: The resulting histogram from the 10000 samples for the estimator $T_{80} = (80)^{-1}\left(\sum_{k=1}^{80} X_k\right)^2$, where $X_1, \ldots, X_{80} \sim U(0,1)$.

1.4 Bootstrap - Jackknife resampling

For this task using the dataset in http://www.math.ntua.gr/~fouskakis/Computational_Stats/data1.rds we use two different methods; Bootstrap and Jackknife resampling in order to estimate the standard error (se) of T_n in (1.3.1) and we compare the results of the sampling methods. We start by reading the dataset from the aforementioned link.

```r
data <- readRDS("data1.rds")
print(data[1:5])

## [1] 0.4481623 0.8563335 0.6211611 0.5276296 0.3102156
```

For the rest of this subsection we denote the estimator T_n by $\hat{\theta}$ and the unknown parameter by θ. In other words, we define

$$\hat{\theta} = \frac{(X_1 + \cdots + X_n)^2}{n}. \tag{1.4.1}$$

1.4.1 Jackknife

The core idea of the Jackknife ([7, 5]) estimator is that it systematically leaves out each observation from the dataset and calculating the parameter estimate over the remaining observations and then aggregating these calculations. In this way, we can get information about the variability of the estimator. More precisely, the Jackknife estimator $\hat{\theta}_J$ of the parameter θ is given by

$$\hat{\theta}_J = n\hat{\theta} - (n-1)\bar{\hat{\theta}}_{(.)}, \tag{1.4.2}$$

where $\bar{\hat{\theta}}_{(.)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)}$ and $\hat{\theta}_{(i)}$ is calculated as in (1.4.1) except we have omitted the i-th observation. In other words,

$$\hat{\theta}_{(i)} = \frac{1}{n-1}\left(\sum_{\substack{j=1,\\ j\neq i}}^{n} X_j\right)^2.$$

An alternative and useful expression of (1.4.2) is through the use of pseudo-values $p_i = n\hat{\theta} - (n-1)\hat{\theta}_{(i)}$. Then, (1.4.2) can be written as

$$\hat{\theta}_J = \frac{1}{n}\sum_{i=1}^{n} p_i. \tag{1.4.3}$$

Now, since X_i's are independent p_i's are also independent (and therefore uncorrelated) and since their variances are the same we have that

$$\begin{aligned}
\operatorname{Var}(\hat{\theta}_j) &= \operatorname{Var}\left(\frac{1}{n}\sum_{i=1}^{n} p_i\right) = \frac{1}{n^2}\operatorname{Var}\left(\sum_{i=1}^{n} p_i\right) \\
&= \frac{1}{n^2}\left(\sum_{i=1}^{n}\operatorname{Var}(p_i) + 2\sum_{i\neq j}\cancelto{0}{\operatorname{Cov}(p_i,p_j)}\right) \\
&= \frac{1}{n}\operatorname{Var}(p_1).
\end{aligned}$$

Using the unbiased estimator $S_p^2 = \frac{1}{n-1}\sum_{i=1}^{n}(p_i - \bar{p})^2$ of the pseudo-values, we can calculate the standard error $S_{\hat{\theta}_J}^2$ of the Jackknife estimator as

$$\begin{aligned}
S_{\hat{\theta}_J}^2 &= \frac{1}{n(n-1)}\sum_{i=1}^{n}(p_i - \bar{p})^2 \\
&= \frac{1}{n(n-1)}\sum_{i=1}^{n}\left(n\hat{\theta} - (n-1)\hat{\theta}_{(i)} - \frac{1}{n}\sum_{j=1}^{n}(n\hat{\theta} - (n-1)\hat{\theta}_{(j)})\right)^2 \\
&= \frac{1}{n(n-1)}\sum_{i=1}^{n}\left(\cancel{n\hat{\theta}} - (n-1)\hat{\theta}_{(i)} - \cancel{n\hat{\theta}} + \frac{n-1}{n}\sum_{j=1}^{n}\hat{\theta}_{(j)}\right)^2 \\
&= \frac{1}{n(n-1)}\sum_{i=1}^{n}\left(\frac{1}{n}\sum_{j=1}^{n}\hat{\theta}_{(j)} - \hat{\theta}_{(i)}\right)\Bigg]^2 \\
&= \frac{(n-1)^2}{n(n-1)}\sum_{i=1}^{n}\left(\frac{1}{n}\sum_{j=1}^{n}\hat{\theta}_{(j)} - \hat{\theta}_{(i)}\right)^2 \\
&= \frac{n-1}{n}\sum_{i=1}^{n}\left(\frac{1}{n}\sum_{j=1}^{n}\hat{\theta}_{(j)} - \hat{\theta}_{(i)}\right)^2.
\end{aligned}$$

Concluding the previous calculations we write

$$S^2_{\hat{\theta}_J} = \frac{n-1}{n} \sum_{i=1}^{n} \left(\bar{\hat{\theta}}_{(.)} - \hat{\theta}_{(i)} \right)^2, \tag{1.4.4}$$

hence the standard error is given by

$$S_{\hat{\theta}_J} = \sqrt{\frac{n-1}{n} \sum_{i=1}^{n} \left(\bar{\hat{\theta}}_{(.)} - \hat{\theta}_{(i)} \right)^2}. \tag{1.4.5}$$

Using (1.4.5) we implement a custom function named *T_jackknife* to calculate the standard error $S_{\hat{\theta}_J}$ by using our dataset. Below we see the main body of the function and the resulting standard error printed on screen.

```
# Jackknife resampling
T_jackknife <- function(samples){
  n <- length(samples) # Size of sample
  theta_ommited <- rep(0,n) # Array containing theta's with ith obs. omitted
  for(i in 1:n){
    theta_ommited[i] <- (1/(n-1))*sum(samples[-i])^2
  }
  # eq: (1.4.5)
  return(sqrt(((n-1)/n)*sum((mean(theta_ommited)-theta_ommited)^2)))
}
print(paste("- Standard error using Jackknife sampling: ", T_jackknife(data)))

## [1] "- Standard error using Jackknife sampling:  2.54701029014747"
```

1.4.2 Bootstrap

Bootstrap ([2]) is a resampling method which uses random sampling with replacement within the existing dataset. Bootstraping can be used in several cases; i.e. s.e and bias estimation, hypothesis testing, confidence intervals and approximating distributions. In our case, we will utilize bootstraping to estimate the standard error of the estimator $\hat{\theta}$ in (1.4.1).

For a sample of size n, we sample n elements from the initial sample with equal probability $1/n$ and replacement. Therefore, in one bootstrap sample we obtain a sequence of points $(X_1^*, \ldots, X_n^*)$ from $(X_1, \ldots, X_n)$. We say that $(X_1^*, \ldots, X_n^*)$ is a bootstrap sample. Having this terminology in mind, we now describe in pseudocode the bootstrap algorithm for the estimation of the standard error for a generic estimator $\hat{\theta} = T(X_1, \ldots, X_n)$.

Algorithm 2 Bootstrap algorithm

```
1: Create $B$ bootstrap samples.
2: for each bootstrap sample $(X_1^*, \ldots, X_n^*)$ do
3:     Compute $\hat{\theta}_i^* = T(X_1^*, \ldots, X_n^*)$
4: end for
5: Compute the standard error $\text{se}(\hat{\theta})$ as
6:     $\text{se}(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}(\hat{\theta}_i^* - \bar{\hat{\theta}}^*)^2}$, where $\bar{\hat{\theta}}^* = B^{-1}\sum_{j=1}^{B}\hat{\theta}_j^*$.
```

Now, using the estimator $\hat{\theta}$ in (1.4.1) we implement Algorithm 2 in R to obtain an estimate of the standard error. As in the case of Jackknife, we create a custom function to handle the resampling.

The function *T_bootstrap* has as arguments the dataset for which the standard error would be estimated, the number of bootstrap samples which is denoted by *B* and initialized to 10000 and the seed which by default is equal to 42. To generate a bootstrap sample we make use of the *sample* method provided by R.

```r
# Bootstrap resampling
T_bootstrap <- function(samples, B=10000, seed = 42){
  set.seed(42)
  bootstrap_samples <- c() # Array to store bootstrap samples, size Bx80
  for(i in 1:B){ # Generating a bootstrap sample
    # and append new sample to array
    bootstrap_samples <- c(bootstrap_samples,
                           sample(samples,
                                  size=length(samples),replace = TRUE))
  }
  # Reshape array 1x(Bx80)->(Bx80)
  bootstrap_samples <- matrix(bootstrap_samples, nrow = B, byrow = T)
  # Calculate the mean row-wise
  T_on_bootstrap <- apply(bootstrap_samples, 1, sum)^2/length(samples)
  info <- c("samples", "Estimated sdt")
  l <- list(info = info, samples = T_on_bootstrap,
            estimated_sd =
              sqrt((1/(B-1))*sum((T_on_bootstrap-mean(T_on_bootstrap))^2)))
  return(l)}
```

Using a for loop, we first generate all B bootstrap samples of size equal to the length of the input data (variable *samples*). The samples are initially stored in a one dimension array. Having generated all bootstrap samples we reshape the one dimension array to a matrix of size $B \times 80$ to have all of the samples in a matrix form. In a variable called *T_on_bootstrap* we store the values of the estimator $\hat{\theta}$ (1.4.1) evaluated on each bootstrap sample. The function returns a list which contains the value of $\hat{\theta}$ on each bootstrap sample and the standard error evaluated as in (1.4.5). Below you can see the estimated standard error printed on screen.

```r
tic <- Sys.time()
bootstrap <- T_bootstrap(data)
tac <- Sys.time()
```

```
print(sprintf("- Bootstrap resampling finished in %.5f sec(s)", tac-tic))

## [1] "- Bootstrap resampling finished in 29.47606 sec(s)"

print(paste("- Standard error: ",bootstrap$estimated_sd))

## [1] "- Standard error:  2.55394669962387"
```

1.4.3 Comparing the results

In Table 1 we summarize the results of the two previous methods.

Table 1: Comparison of standard errors

Method	**Standard error** se$(\hat{\theta})$
Jackknife	2.547
Bootstrap	2.553

Now, using the knowledge that our samples have a $U(0,1)$ distribution we compare the samples obtained in Subsection 1.3.2 with the corresponding bootstrap samples by generating a histogram.

```
# Compare histograms with simulation sampling & Bootstrap sampling
simul <- T_simulation()
dt_1 <- data.table(samples = simul$samples,
                   Method = rep("R - Simulation",length(simul$samples)))
dt_2 <- data.table(samples = bootstrap$samples,
                   Method = rep("Bootstrap", length(bootstrap$samples)))
dt <- rbindlist(list(dt_1,dt_2))
ggplot(dt,aes(x = samples, color = Method, fill = Method))+
  geom_histogram(position = "dodge", bins =30)+
  labs(x = "Estimator Value",y = "Number of samples")+
  theme(legend.position = c(0.86, 0.83))
```

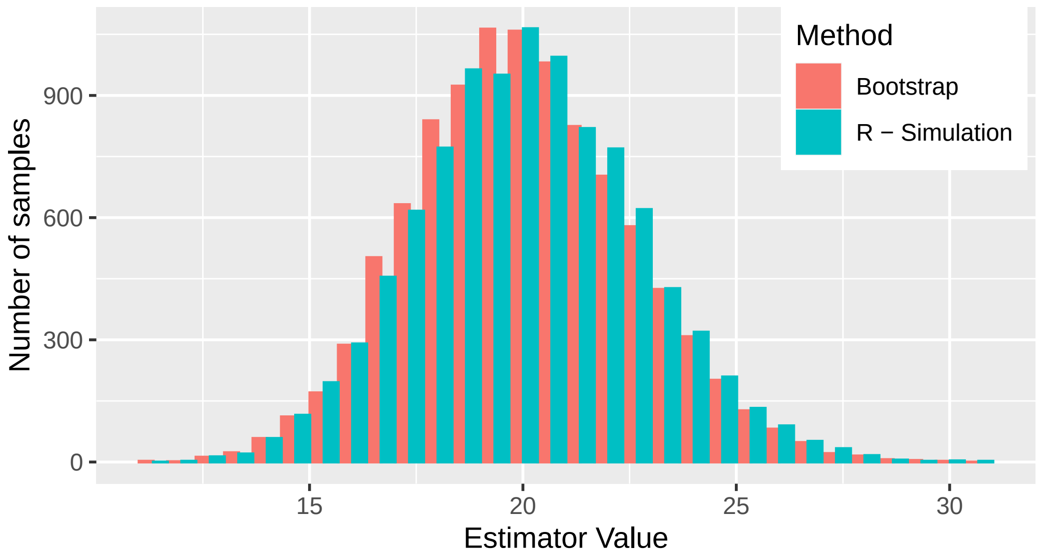

Figure 6: Histogram comparison for the estimator $\hat{\theta}$. The first method uses the function runif from R to sample from $U(0,1)$ in order to generate 10000 samples from $\hat{\theta}$. Bootstrap makes no use of the underlying distribution and generates 10000 samples for $\hat{\theta}$ with resampling.

2 Density Estimation

In exercise 2 we make use of density estimation techniques to estimate the density of waiting times between two consecutive eruptions for the Old Faithful geyser located in Yellowstone National Park, Wyoming, USA. The Old Faithful geyser data contains 272 waiting times (in mins) between two consecutive eruptions. With the following commands we load the data in R and we print the first five entries as well as the mean and standard deviation of our samples.

```r
# Exercise 2 - Density Estimation
# Importing libraries & data
library(data.table)
library(ggplot2)
library(latex2exp)
sample <- faithful$eruptions

print(paste("- Mean:", mean(sample)))

## [1] "- Mean: 3.48778308823529"

print(paste("- Standard deviation:", sd(sample)))

## [1] "- Standard deviation: 1.14137125110521"

print(sample[1:5])

## [1] 3.600 1.800 3.333 2.283 4.533
```

2.1 Bandwidth selection

In density estimation we have an i.i.d sample $(X_1, \dots, X_n)$[2] drawn for some univariate distribution with unknown density f. Our main goal is to estimate the shape of this density f. To this end, we form its kernel density estimator

$$\hat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{2.1.1}$$

where $K(x)$ is a function called kernel and satisfies the following properties:

(i) $K(x) \geq 0,$ for all $x \in \mathbb{R}$,

(ii) $\int\limits_{-\infty}^{\infty} K(x)\, dx = 1,$

(iii) $\int\limits_{-\infty}^{\infty} xK(x)\, dx = 0,$

(iv) $\int\limits_{-\infty}^{\infty} x^2K(x)\, dx < \infty,$

[2]In our case is the sample consisting of the 272 waiting times.

$h > 0$ is a smoothing parameter called the bandwidth. There are several options for the kernel $K(x)$. In this case we will make use of the Epanechnikov kernel which is given by

$$K(x) = \begin{cases} \frac{3}{4}(1-x^2), & |x| \leq 1. \\ 0, & |x| > 1. \end{cases} \tag{2.1.2}$$

One criterion to determine the suitable bandwidth h in order to approximate the unknown density f is by maximizing the cross-validated likelihood given by

$$L(h) = \prod_{i=1}^{n} \hat{f}_{h,i}(x_i), \tag{2.1.3}$$

where by $\hat{f}_{h,i}$ we denote the estimate of $\hat{f}$ overall all samples except for the i-th sample. Since the natural logarithm is increasing, it is suffice to maximize the cross-validated log-likelihood given by

$$l(h) = \sum_{i=1}^{n} \log \hat{f}_{h,i}(x_i). \tag{2.1.4}$$

Instead of analytically maximize $l(h)$ we will proceed with a heuristic algorithm to find the most suitable bandwidth h given our samples. To this end, we define four utility functions. The function named *epanechnikov* which calculates the value of the Epanechnikov kernel according to (2.1.2) at a given point x. The function *density_estimate* which takes as input a point x, the bandwidth h, a sample of points, and returns the value of the kernel density estimator $\hat{f}_h$ at x according to (2.1.1) for that given sample. The function *cv_log_likelihood* which takes as input the bandwidth h, a sample of points and returns the value of the cross-validated log-likelihood as described in (2.1.4). The function *logspace* which generates a number of points indicated by the argument *num* in the interval (*start,stop*) in logarithmic scale. Below you can see the main body of these functions.

```r
# Utility functions
epanechnikov <- function(x){ # Epanechnikov kernel evaluated at x
  return(ifelse(abs(x)<= 1, (3/4)*(1-x^2), 0))}
density_estimate <- function(x,h,sample){ # Density estimator
  n <- length(sample)
  epan <- rep(0,n)
  for(i in 1:n){
    epan[i]<-epanechnikov((x-sample[i])/h)
  }
  return((1/n)*sum(epan)*1/h)}
cv_log_likelihood <- function(h,sample){ # Cross-validated log-likelihood
  n <- length(sample)
  cv_log <- rep(0,n)
  for(i in 1:n){
    cv_log[i] <- log(density_estimate(sample[i],h,sample[-i]))
  }
  return(sum(cv_log))}
logspace <- function(start, stop, num = 50, base = 10.0) {
  s <- seq(log(start, base), log(stop, base), length.out=num)
  return(base ^ s)}
```

To find the optimal bandwidth h we first perform two passes through certain intervals and search for the points where the cross-validated log-likelihood achieves its maximum value. Since the standard deviation of our sample is equal to 1.14 there is no point to examine values of h larger than 2 for example. This is because for large values of h the kernel density estimator would be the constant estimator which is only useful when the underlying distribution is uniform.

In the first pass using the *logspace* function we generate 100 points in the interval $(0.17, 2)$ log scale with base 10, and we calculate the value of the cross-validated log-likelihood on these points using the utility functions defined previously. At the end, we print the point/points where the maximum value was/were spotted. Below you can see the code that performs the 1st pass and the corresponding results.

```r
tic = Sys.time()
# 1st pass
x <- logspace(start = 0.17, stop =2, num =100)
values <- rep(0, length(x))
for(i in 1:length(x)){values[i] = cv_log_likelihood(x[i],sample)}
toc <- Sys.time()
print(sprintf("- 1st pass finished in %.3f sec(s) and %d were examined.",
              toc-tic,length(x)))

## [1] "- 1st pass finished in 21.248 sec(s) and 100 were examined."

x1 <- which(values == max(values), arr.ind = TRUE)
print(sprintf("- Maximum value %.4f at point %.2f",
              max(values),x[x1]))

## [1] "- Maximum value -270.5506 at point 0.21"
```

As we can see the maximum value is at point $x = 0.21$ with value -270.5538. In the second pass the examine 100 points in the interval $[0.2, 0.22]$.

```r
tic = Sys.time()
x <- logspace(start=0.20,stop=0.22, num =100)
values <- rep(0,length(x))
for(i in 1:length(x)){values[i] = cv_log_likelihood(x[i],sample)}
toc <- Sys.time()
print(sprintf("- 1st pass finished in %.3f sec(s) and %d were examined.",
              toc-tic,length(x)))

## [1] "- 1st pass finished in 19.012 sec(s) and 100 were examined."

x1 <- which(values == max(values), arr.ind = TRUE)
print(sprintf("- Maximum value %.4f at point %.2f",
              max(values),x[x1]))

## [1] "- Maximum value -270.5365 at point 0.21"

d <- data.frame(cbind(x,values))
ggplot(d, aes(x = x, y = values))+
  geom_path(col = "#E75177", size = 1.2)+xlab("h")+ylab("log-likelikehood")+
  labs(title =
         TeX("CV log-likelihood $\\sum_{i=1}^n\\log \\hat{f}_{h,i}(x_i)$"))+
    theme(plot.title = element_text(size=10))
```

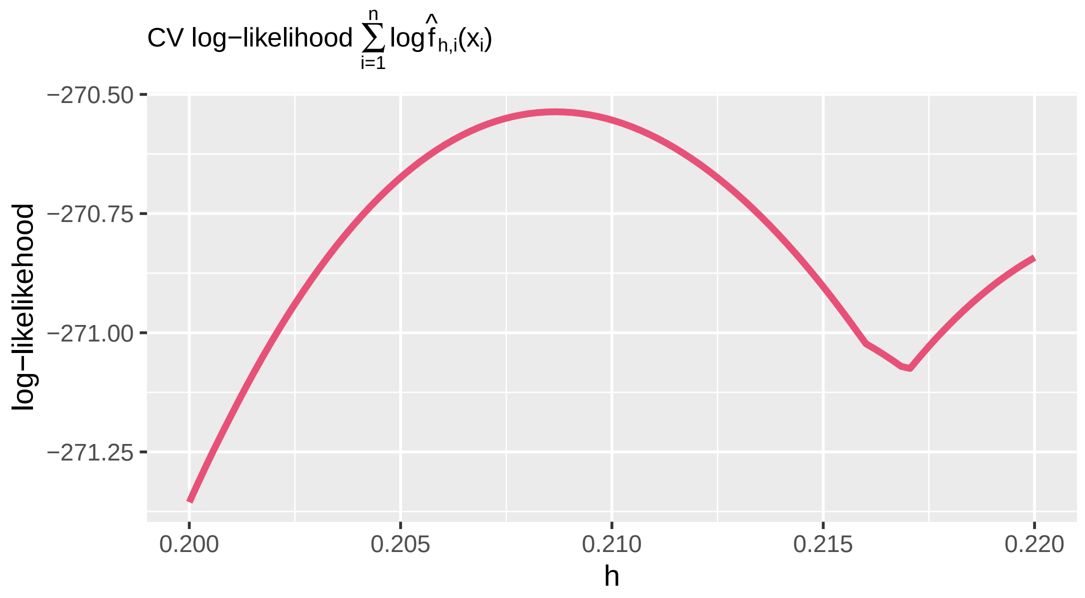

Figure 7: The graph of the cross-validated log-likelihood $\sum_{i=1}^{n} \log \hat{f}_{h,i}(x_i)$ evaluated at 100 points in the interval closed interval with points $0.2, 0.22$ using the custom function cv_log_loglikelihood.

Figure 7 shows that the maximum value of the cross-validated log-likelihood lies in the closed interval $[0.205, 0.21]$. To get a better estimate of the optimum h we perform a binary search in the interval $[0.205, 0.21]$. For this purpose, we define two custom functions named *binary_search* and *search_for_maximum*. The *binary_search* function takes as input an interval (α, β) and returns the endpoints of the interval $[\alpha, \frac{\alpha+\beta}{2}]$ if

$$\max\{l(\alpha), l(\frac{\alpha+\beta}{2})\} > \max\{l(\frac{\alpha+\beta}{2}), l(\beta)\}, \tag{2.1.5}$$

otherwise it returns the endpoints of the interval $[\frac{\alpha+\beta}{2}, \beta]$. The function *search_for_maximum* takes the endpoints of an interval (α, β) and a tolerance which is initialized to 10^{-8} and it calls the *binary_search* function iteratively, until the length of the interval becomes smaller than 10^{-8}. At the end of the iterations, the algorithm obtains an interval $[c, d] \subseteq [0.205, 0.21]$ with $d - c < 10^{-8}$. Then, the function returns the optimal bandwidth h_{opt} according to the condition

$$h_{\text{opt}} = \arg\max\{l(i) : i = c, d, (c+d)/2\}. \tag{2.1.6}$$

Below we define the two aforementioned functions and use them to obtain the optimum bandwidth h_{opt} as described in 2.1.6.

```r
binary_search <- function(a,b,sample){ # Determine the new interval
  mid_point <- (a+b)/2
  ifelse(cv_log_likelihood(a,sample)>cv_log_likelihood(b,sample), # eq. (2.1.5)
         return(c(a,mid_point)), return(c(mid_point,b)))}
search_for_maximum <- function(a,b,sample,tol=1e-8){
  while(abs(a-b)>tol){ # while length of the interval (a,b) is larger than tol
    interval <- binary_search(a,b,sample)
    a <- interval[1]
    b <- interval[2]
  }
  # Find the point that satisfies eq. (2.1.6)
  points <- c(a, (a+b)/2, b)
  values <- c(cv_log_likelihood(a, sample),
              cv_log_likelihood((a+b)/2,sample),
              cv_log_likelihood(b,sample))
  index <- which(values == max(values),arr.ind = TRUE)
  return(points[index])}
#Find optimal h
h_opt <- search_for_maximum(a=0.20,b=0.21,sample)
print(paste("- The optimal bandwidth is:",h_opt))

## [1] "- The optimal bandwidth is: 0.208661231994629"
```

Now, since we have determined the $h_{\text{opt}} \approx 0.20866$ using the function *density* we can plot the kernel density function as described in (2.1.1) using as bandwidth the h_{opt} and K the Epanechnikov kernel in (2.1.2).

```
estimated_density <- density(sample, kernel = "epanechnikov", bw = h_opt,
                             n = 2048)
d <- data.frame(cbind(estimated_density$x,estimated_density$y))
ggplot(d, aes(x = estimated_density$x, y = estimated_density$y))+
  geom_path(col = "#E75177")+xlab("x")+ylab("Likelihood")+
  labs(title = TeX("Estimated Density for bandwidth $h=0.20866$"))+
  theme(text=element_text(size = 9),
        plot.subtitle = element_text(size = 9))
```

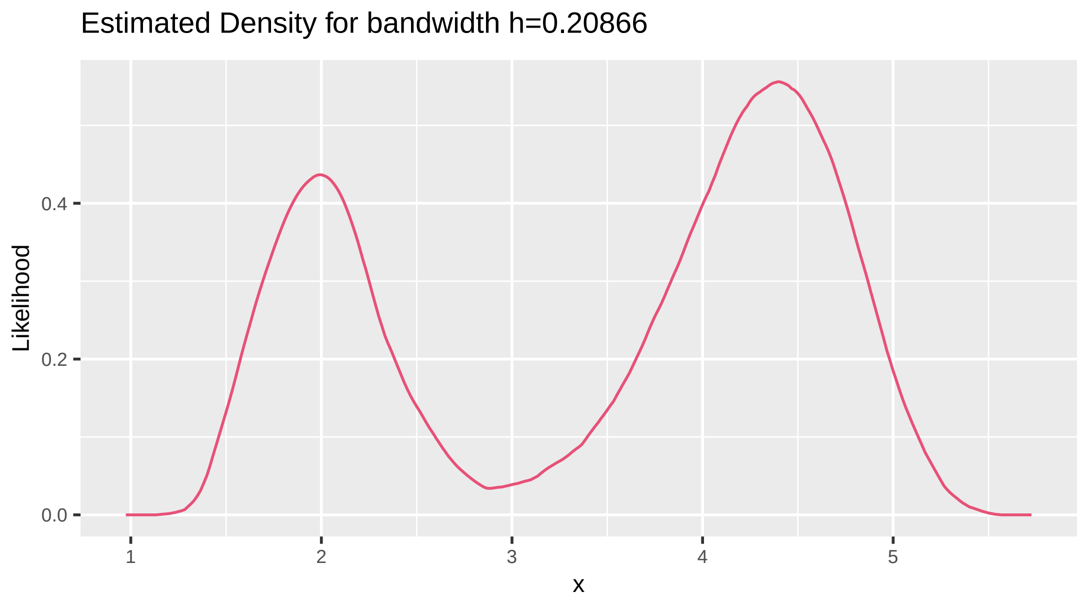

Figure 8: The kernel density estimator using $\hat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)$ with the $h_{\text{opt}} \approx 0.20866$ and the Epanechnikov kernel K using the density function provided by R.

2.2 Comparing densities

Having calculated the optimum bandwidth we now visualize the estimated kernel density using a custom function that calculates $\hat{f}_{h_{\text{opt}}}(x)$ at a given input point x. The function is the same with the custom function *density_estimate* that we used to calculate h_{opt} but this time the bandwidth h is equal to h_{opt}. We recall the main body of this function in the following separated block of code.

```r
# Evaluate the Epanechnikov kernel at point x
epanechnikov <- function(x){
  return(ifelse(abs(x)<= 1, (3/4)*(1-x^2), 0))
}
# Evaluate the estimated kernel density at x for given
# bandwidth h, the optimal one is h_opt = 0.20866.
density_estimate <- function(x,h,sample){
  n <- length(sample)
  epan <- rep(0,n)
  for(i in 1:n){
    epan[i]<-epanechnikov((x-sample[i])/h)
  }
  return((1/n)*sum(epan)*1/h)
}
```

Using the generated points of the *density* method. We compare the graph of the density obtained when using the custom functions *epanechnikov* and *density_estimate* with the graph of the density obtained with the *density* method of R.

```r
# Compare densities
num_pts <- length(estimated_density$x)
custom_estimations <- rep(0,num_pts)
for(i in 1:num_pts){
  custom_estimations[i]<-density_estimate(estimated_density$x[i]
                                          ,h_opt,sample)}
dt_1 <- data.table(x = estimated_density$x,
                   y = custom_estimations, Estimation = "Custom")
dt_2 <- data.table(x = estimated_density$x,
                   y = estimated_density$y, Estimation ="Actual")
d <- rbindlist(list(dt_1,dt_2))
ggplot(d, aes(x = x, y = y), color = Estimation)+
  geom_path(aes(color = Estimation),size=0.8)+xlab("x")+ylab("Density")+
  labs(title = "Estimated density vs actual density")+
  scale_colour_manual("Density",values = c("#E75177","#4FA3AB"))+
  theme(legend.position = c(0.89, 0.82))
```

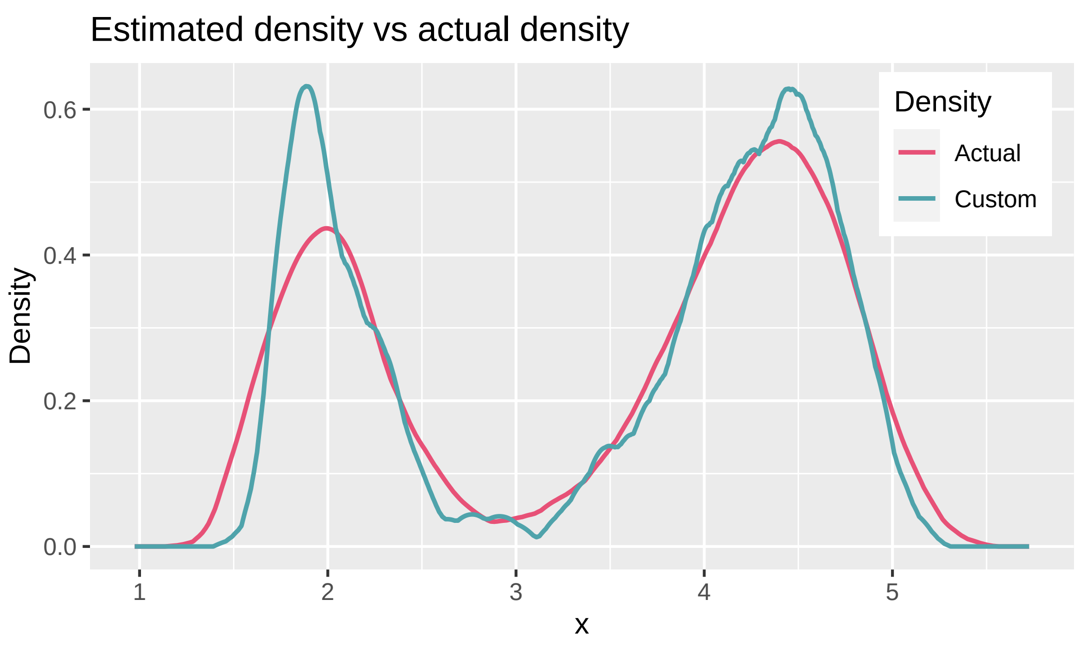

Figure 9: The graph of the kernel density estimate with $h_{\text{opt}} \approx 0.20866$ using two different methods. The red line corresponds to the graph using the fitted estimate of R for h_{opt}. The light-blue line corresponds to the graph using the custom functions epanechnikov and density estimate.

2.3 Calculating probabilities through integration

Assuming that the time between two consecutive eruptions of the old Faithful geyser is modeled by a random variable X we estimate the probability of not having any eruptions under 3.5 minutes. In other words, we estimate the quantity $\mathbb{P}(X \geq 3.5)$. To this end, using the kernel density estimate $\hat{f}_{h_{\text{opt}}}$ obtained in Subsection 2.1 we can approximate $\mathbb{P}(X \geq 3.5)$ as

$$\mathbb{P}(X \geq 3.5) \approx \int_0^\infty \hat{f}_{h_{\text{opt}}}(x)\,dx. \tag{2.3.1}$$

Below we calculate the integral in (2.1.3) using the *integrate* method provided by R.

```
# Calculate P(X>3.5)
final_density <- function(x, seed = 42){
  set.seed(seed)
  return(density_estimate(x,h_opt,sample))
}
int <- integrate(Vectorize(final_density),lower =3.5, upper = Inf)
print(paste("- Probability using integration", int$value))

## [1] "- Probability using integration 0.612047140232332"
```

2.4 Calculating probabilities through sampling

An alternative and perhaps a more intuitive way to estimate the probability in (2.3.1) is to use a sampling approach. More precisely, we can sample a sequence of points $\hat{x}_1, \ldots, \hat{x}_N$ from $\hat{f}_{h_{\text{opt}}}(x)$ and calculate the $\mathbb{P}(X \geq 3.5)$ as

$$\mathbb{P}(X \geq 3.5) \approx \frac{\big|\{i \in \mathbb{N} : \hat{x}_i \geq 3.5\}\big|}{N}, \tag{2.3.2}$$

where by $|A|$ we denote the cardinal number of the set A. The only problem we have to surpass at this point is how to sample from $\hat{f}_{h_{\text{opt}}}(x)$. To this end, suppose that our initial samples that we used to construct $\hat{f}_{h_{\text{opt}}}(x)$ are $(x_1, \ldots, x_n)$. Denote by Z the uniform random variable on the set $\{x_1, \ldots, x_n\}$; i.e.

$$\mathbb{P}(Z = x_i) = \frac{1}{n}, \tag{2.3.3}$$

for every $i = 1, \ldots, n$. Moreover, denote by Y the random variable with density function the Epanechnikov kernel $f_Y(y) = \frac{3}{4}(1 - y^2) \cdot \mathbb{1}_{|y| \leq 1}$[3] Then, we claim that the density function of $X = Z + h_{\text{opt}}Y$ is $\hat{f}_{h_{\text{opt}}}$. Indeed, to prove this we first calculate the distribution F_X of X. For fixed $x \in \mathbb{R}$, using the law of total probability, we write

$$\begin{aligned}
F_X(x) &= \mathbb{P}(X \leq x) = \mathbb{P}(Z + h_{\text{opt}}Y \leq x) \\
&= \sum_{i=1}^{n} \mathbb{P}(x_i + h_{\text{opt}}Y \leq x \,|\, Z = x_i) \cdot \cancelto{1/n}{\mathbb{P}(Z = x_i)} \\
&= \frac{1}{n}\sum_{i=1}^{n} \mathbb{P}(h_{\text{opt}}Y \leq x - x_i) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{P}\left(Y \leq \frac{x - x_i}{h_{\text{opt}}}\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} F_Y\left(\frac{x - x_i}{h_{\text{opt}}}\right).
\end{aligned}$$

Therefore, we deduce that

$$F_X(x) = \frac{1}{n}\sum_{i=1}^{n} F_Y\left(\frac{x - x_i}{h_{\text{opt}}}\right). \tag{2.3.4}$$

But now, since the density $f_Y(y)$ is continuous the distribution $F_Y(y)$ is everywhere differentiable and $F_Y'(y) = f_Y(y)$ for all $y \in \mathbb{R}$. Hence, differentiating (2.3.4) with respect to x we get

$$\begin{aligned}
\frac{\partial}{\partial x}F_X(x) &= \frac{\partial}{\partial x}\left(\frac{1}{n}\sum_{i=1}^{n} F_Y\left(\frac{x - x_i}{h_{\text{opt}}}\right)\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial x}F_Y\left(\frac{x - x_i}{h_{\text{opt}}}\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} f_Y\left(\frac{x - x_i}{h_{\text{opt}}}\right)\frac{\partial}{\partial x}\left(\frac{x - x_i}{h_{\text{opt}}}\right) \\
&= \frac{1}{nh_{\text{opt}}}\sum_{i=1}^{n} f_Y\left(\frac{x - x_i}{h_{\text{opt}}}\right).
\end{aligned}$$

But now, since f_Y is the Epanechnikov kernel and $\partial/\partial x F_X(x) = f_X(x)$ is the density of X we deduce that

$$f_X(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)\hat{f}_{h_{\text{opt}}}(x),$$

and therefore the random variable X has $\hat{f}_{h_{\text{opt}}}$ as density. Now, the relation $X = Z + \hat{f}_{h_{\text{opt}}}Y$ gives a clear cut way to sample from $\hat{f}_{h_{\text{opt}}}$. In the next algorithm we describe this sampling method.

[3]By $\mathbb{1}_A$ we denote the indicator function on the set A.

Algorithm 3 Sampling from $\hat{f}_{h_{\text{opt}}}$

```
Require: the number of samples $M$.
Require: the initial sample points $x_1, \ldots, x_n$.
Require: the optimal bandwidth $h_{\text{opt}}$.
1: for each $i = 1, \ldots, M$ do
2:     Draw a sample $z_i$ with equal probability from $\{x_1, \ldots, x_n\}$.
3:     Draw a sample $y_i \sim Y$.
4:     $\hat{x_i} \leftarrow z_i + h_{\text{opt}} y_i$.
5: end for
6: return the sample $\{\hat{x_1}, \ldots, \hat{x_M}\}$.
```

Now, to sample from $\{x_1, \ldots, x_n\}$ with equal probability R provides the method *sample*. However, there is no method to sample from Y with Y having the Epanechnikov kernel as density. Luckily, there is a nice description of Y; let Y_1, Y_2, Y_3 be three independent draws from $U(-1, 1)$. Then, $Y_\delta \overset{d}{=} Y$, where Y_δ is the median of Y_1, Y_2, Y_3. To prove this we show that the densities of Y_δ and Y coincide. To this end, let $-1 \leq t \leq 1$. Then, the set $\{Y_\delta \leq t\}$ can be decomposed as

$$\begin{aligned}\{Y_\delta \leq t\} = &\underbrace{\{\text{Exactly two of the } Y_i\text{'s are lower or equal than } t\}}_{A_t} \\ &\bigcup \underbrace{\{\text{All three of } Y_i\text{'s are lower or equal than } t\}}_{B_t}.\end{aligned} \tag{2.3.5}$$

Indeed, since Y_δ is the median of Y_i's then $Y_\delta \leq t$ implies that at least two of Y_1, Y_2, Y_3 will be $\leq t$. Conversely, if exactly two of Y_1, Y_2, Y_3 is lower or equal than t then the median Y_δ will also be lower or equal than t. In the case where all three Y_1, Y_2, Y_3 are lower or equal than t it obviously implies that the median Y_δ will also be lower or equal than t. Therefore, (2.3.5) is true. Now, to calculate $\mathbb{P}(Y_\delta \leq t)$ we only need to find $\mathbb{P}(A_t), \mathbb{P}(B_t)$. The probability of exactly two of the Y_i's are lower or equal than t is a binomial distribution of three independent Bernoulli trials with success probability $p = \mathbb{P}(Y_i \leq t)$, $i = 1, 2, 3$. Therefore,

$$\mathbb{P}(A_t) = \binom{3}{2} \mathbb{P}(Y_1 \leq t)^2 \cdot (1 - \mathbb{P}(Y_1 \leq t)). \tag{2.3.6}$$

But since Y_1, Y_2, Y_3 are i.i.d from $U(-1, 1)$ we have that

$$\mathbb{P}(Y_i \leq t) = \int_{-1}^{t} \frac{1}{2}\, ds = \frac{t}{2} + \frac{1}{2}.$$

Therefore, substituting in (2.3.6) we get

$$\begin{aligned}\mathbb{P}(A_t) &= \binom{3}{2}\left(\frac{t}{2} + \frac{1}{2}\right)^2 \left(\frac{1}{2} - \frac{t}{2}\right) \\ &= -3\left(\frac{t^2}{4} + \frac{1}{4}\right)\left(\frac{t}{2} + \frac{1}{2}\right) \\ &= \underbrace{-3\left(\frac{t^3}{8} + \frac{t^2}{8} - \frac{t}{8} - \frac{1}{8}\right)}_{h(t)}.\end{aligned}$$

Differentiating $h(t)$ with respect to t, we obtain

$$h'(t) = -3\left(\frac{3t^2}{8} + \frac{t}{4} - \frac{1}{8}\right). \tag{2.3.7}$$

For $\mathbb{P}(B_t)$ we have that

$$\begin{aligned}\mathbb{P}(B_t) &= \mathbb{P}(Y_1, Y_2, Y_3 \le t) \\ &\overset{\text{i.i.d}}{=} \mathbb{P}(Y_1 \le t)^3 = \underbrace{\left(\frac{t}{2} + \frac{1}{2}\right)^3}_{g(t)}.\end{aligned}$$

Differentiating $g(t)$ we get

$$g'(t) = \frac{3}{2}\left(\frac{t}{2} + \frac{1}{2}\right)^2 = \frac{3}{2}\left(\frac{t^2}{4} + \frac{t}{2} + \frac{1}{4}\right). \tag{2.3.8}$$

By (2.3.5), (2.3.7) and (2.3.8) we can calculate the density f_{Y_δ} of Y_δ as

$$\begin{aligned}f_{Y_\delta}(t) = \frac{\partial \mathbb{P}(Y_\delta \le t)}{\partial t} &= \frac{\partial}{\partial t}(h(t) + g(t)) \\ &= h'(t) + g'(t) \\ &= -\frac{9t^2}{8} - \frac{3t}{4} + \frac{3}{8} + \frac{3t^2}{8} + \frac{3t}{4} + \frac{3}{8} \\ &= -\frac{6t^2}{8} + \frac{6}{8} = \frac{3}{4}(1 - t^2).\end{aligned}$$

Therefore, we deduce that the density of Y_δ is equal to the Epanechnikov kernel, and hence Y and Y_δ have the same distribution, i.e. $Y_\delta \overset{d}{=} Y$.

The fact that the Epanechnikov kernel is the median of three independent draws from $U(-1, 1)$ allows us to construct a one line code to obtain samples from Y. For n number of samples we simply generate a matrix of size $3 \times n$ where each entry is drawn independently from $U(-1, 1)$. Then, we use the *apply* method column-wise to calculate the median for each triple of the i.i.d. draws from $U(-1, 1)$. The resulting $1 \times n$ is a sample from Y. To draw from $h_{\text{opt}}Y$ as described in Algorithm 3 we simply draw from $U(-h_{\text{opt}}, h_{\text{opt}})$ instead of $U(-1, 1)$. Below we implement the method of Algorithm 3 to sample 250 values from $\hat{f}_{h_{\text{opt}}}$ and we calculate the probability $\mathbb{P}(X \ge 3.5)$ as the ratio in (2.3.2).

```r
# Calculate P(X>3.5) through simulation

# Sample from the estimated f
# Step 1 - Pick at random a point Z from sample
# Step 2 - Pick at random a point Y from kernel
# Step 3 - Then X = Z+h_optY is a sample from the estimated kernel density f
set.seed(42)
n <- 250 # Number of samples
y <- apply(matrix(runif(3*n,-h_opt,h_opt),3), 2, median)#Epanechnikov sampling
x <- sample(sample,size=n,replace = TRUE)
z <- x+y
p1 <- int$value # Probability through integration
p2 <- length(z[z>3.5])/length(z) # Probability through simulation
print(paste("- Estimated probability through simulation",
    p2))
```

```
## [1] "- Estimated probability through simulation 0.612"
```

As we can see the estimate for $\mathbb{P}(X \geq 3.5)$ is almost the same in both methods. The integration method gives $\mathbb{P}(X \geq 3.5) \approx 0.6120471$ and the simulation method $\mathbb{P}(X \geq 3.5) \approx 0.612$.

3 Expectation - Maximization

3.1 Poisson Mixture - Theoretical Results

In this task we are given two populations ω_1, ω_2 and the observations $\mathbf{x} = (x_1, x_2, x_3, x_4) = (2, 7, 3, 9)$. In general, the observations $\mathbf{x} = (x_1, x_2, x_3, x_4)$ arise from a Poisson mixture X of the following form

$$\mathbb{P}(X = x) = \pi_1 \mathbb{P}(X = x \,|\, x \in \omega_1) + \pi_2 \mathbb{P}(X = x \,|\, x \in \omega_2), \tag{3.1.1}$$

for $x = 0, 1, \ldots,$ where $\pi_1, \pi_2 \geq 0$, $\pi_1 + \pi_2 = 1$ and $X \,|\, \omega_1 \sim \text{Poisson}(\lambda_1)$, $X \,|\, \omega_2 \sim \text{Poisson}(\lambda_2)$. Intuitively, this means that a new observation x drawn X is drawn with probability π_1 from a poisson distribution with parameter λ_1 and with probability π_2 from a poisson distribution with parameter λ_2. Our goal in this task, based on our sample of observations $(2, 7, 3, 9)$, is to estimate the parameters $\tilde{\theta} = (\pi_1, \pi_2, \lambda_1, \lambda_2)$ in each of the following cases:

(i) **Case 1:** Using the prior knowledge that the observations $(x_1, x_3) = (2, 3)$ belong to the population ω_1 and the observations $(x_2, x_4) = (7, 9)$ are drawn from ω_2.

(ii) **Case 2:** Without having any prior knowledge to which population the each of observations $\mathbf{x} = (x_1, x_2, x_3, x_4)$ belongs to.

For the first case, since we have this prior knowledge in our hands, we can proceed by estimating the vector of parameters $\tilde{\theta}$ by maximizing the likelihood (MLE). For the second case, we estimate the parameters via the Expectation - Maximization (EM) algorithm [1].

Before we proceed with the estimation of $\tilde{\theta}$ in each of the two cases, we first derive some theoretical results concerning the Likelihood $L(\tilde{\theta} \,|\, \mathbf{x})$ and log-Likelihood $l(\tilde{\theta} \,|\, \mathbf{x})$. In general, the likelihood $L(\tilde{\theta} \,|\, \mathbf{x})$ is given by

$$L(\tilde{\theta} \,|\, \mathbf{x}) = \prod_{i=1}^{4} f_X(x_i; \tilde{\theta}) = \prod_{i=1}^{4} \left(\pi_1 \cdot \frac{e^{-\lambda_1} \lambda_1^{x_i}}{x!} + \pi_2 \cdot \frac{e^{-\lambda_2} \lambda_2^{x_2}}{x!} \right), \tag{3.1.2}$$

where $f_X(x; \tilde{\theta})$ is the density function of X with parameters $\tilde{\theta}$. Taking logarithms in (3.1.2) we have that the log-likelihood $l(\tilde{\theta} \,|\, \mathbf{x})$ is given by

$$l(\tilde{\theta} \,|\, \mathbf{x}) = \sum_{i=1}^{4} \log \left(\pi_1 \cdot \frac{e^{-\lambda_1} \lambda_1^{x_i}}{x!} + \pi_2 \cdot \frac{e^{-\lambda_2} \lambda_2^{x_2}}{x!} \right). \tag{3.1.3}$$

Since the logarithm is increasing, to maximize $L(\tilde{\theta} \,|\, \mathbf{x})$ with respect to $\tilde{\theta}$ its suffices to maximize $l(\tilde{\theta} \,|\, \mathbf{x})$ with respect $\tilde{\theta}$. However, maximizing (3.1.3) with respect to $\tilde{\theta}$ is difficult due to the summands inside the logarithm. One way to tackle this problem and simplify (3.1.3) is by introducing the latent variables z_{ij}, where

$$z_{ij} = \begin{cases} 1, & \text{the } i\text{-th observation belongs to } \omega_j. \\ 0, & \text{otherwise.} \end{cases} \quad , \tag{3.1.4}$$

for $i = 1, 2, 3, 4$ and $j = 1, 2$. Using the latent variables we can write

$$l(\tilde{\theta}\,|\,\mathbf{x}) = \sum_{i=1}^{4} \log\left(z_{i1}\pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x!} + z_{i2}\pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_2}}{x!} \right). \tag{3.1.5}$$

Now, we claim that for fixed i,

$$\log\left(z_{i1}\pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x!} + z_{i2}\pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_2}}{x!} \right) = z_{i1}\log\left(\pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x!} \right) + z_{i2}\log\left(\pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_2}}{x!} \right). \tag{3.1.6}$$

To this end, observe that $z_{i1} = 1 \iff z_{i2} = 0$. Therefore, in the case where $z_{i1} = 1$ we have that (3.1.6) reduces to

$$\log\left(\pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x!} + \cancelto{0}{z_{i2}\pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_2}}{x!}} \right) = \log\left(\pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x!} \right) + \cancelto{0}{z_{i2}\log\left(\pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_2}}{x!} \right)}.$$

and in the case where $z_{i1} = 0$ (and hence $z_{i2} = 1$) we have that

$$\log\left(\cancelto{0}{z_{i1}\pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x!}} + \pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_2}}{x!} \right) = \cancelto{0}{z_{i1}\log\left(\pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x!} \right)} + \log\left(\pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_2}}{x!} \right),$$

which shows that (3.1.5) is true in either case. Now, applying (3.1.6) to (3.1.5) we obtain

$$\begin{aligned} l(\tilde{\theta}\,|\,\mathbf{x}) &= \sum_{i=1}^{4} \left[z_{i1}\log\left(\pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x!} \right) + z_{i2}\log\left(\pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_2}}{x!} \right) \right] \qquad (3.1.7) \\ &= \sum_{i=1}^{4} \Big(z_{i1}\log\pi_1 - \lambda_1 z_{i1} + z_{i1}x_i\log\lambda_1 - z_{i1}\log x_i! + z_{i2}\log\pi_2 - \lambda_2 z_{i2} + z_{i2}x_i\log\lambda_2 - z_{i2}\log x_i! \Big). \end{aligned}$$

Now, its a lot easier to maximize (3.1.7) with respect to $\tilde{\theta}$ instead of (3.1.5). Indeed, taking partial derivatives we see that

$$\frac{\partial}{\partial\pi_1} l(\tilde{\theta}\,|\,\mathbf{x}) = \sum_{i=1}^{4} \frac{z_{i1}}{\pi_1}, \quad \frac{\partial}{\partial\pi_2} l(\tilde{\theta}\,|\,\mathbf{x}) = \sum_{i=1}^{4} \frac{z_{i2}}{\pi_2}. \tag{3.1.8}$$

And,

$$\frac{\partial}{\partial\lambda_1} l(\tilde{\theta}\,|\,\mathbf{x}) = \sum_{i=1}^{4} z_{i1}\frac{x_i}{\lambda_1} - \sum_{i=1}^{4} z_{i1}, \quad \frac{\partial}{\partial\lambda_2} l(\tilde{\theta}\,|\,\mathbf{x}) = \sum_{i=1}^{4} z_{i2}\frac{x_i}{\lambda_2} - \sum_{i=1}^{4} z_{i2}. \tag{3.1.9}$$

Equating to zero we obtain the following estimations for λ_1, λ_2

$$\hat{\lambda_1} = \frac{\sum_{i=1}^{4} z_{i1}x_i}{\sum_{i=1}^{4} z_{i1}}, \quad \hat{\lambda_2} = \frac{\sum_{i=1}^{4} z_{i2}x_i}{\sum_{i=1}^{4} z_{i2}}. \tag{3.1.10}$$

For π_1, π_2 denote by $A = \sum\limits_{i=1}^{4} z_{i1}$, $B = \sum\limits_{i=1}^{4} z_{i2}$. Then, substituting $\pi_2 = (1 - \pi_1)$ in (3.1.7) and taking the partial derivative with respect to π_1 we obtain

$$\begin{aligned} \frac{\partial}{\partial\pi_1} l(\tilde{\theta}\,|\,\mathbf{x}) = 0 &\iff \frac{A}{\pi_1} - \frac{B}{1 - \pi_1} = 0 \\ &\iff \pi_1 = \frac{A}{A + B}, \end{aligned}$$

which leads to a unique solution for the estimates of π_1, π_2

$$\hat{\pi_1} = \frac{A}{A + B} = \frac{\sum_{i=1}^{4} z_{i1}}{\sum_{i=1}^{4}(z_{i1} + z_{i2})}, \quad \hat{\pi_2} = \frac{B}{A + B} = \frac{\sum_{i=1}^{4} z_{i2}}{\sum_{i=1}^{4}(z_{i1} + z_{i2})}. \tag{3.1.11}$$

3.2 Estimating parameters through MLE

Now, using the prior knowledge that the samples $(x_1, x_3) = (2, 3)$ belong to the population ω_1 and $(x_2, x_4) = (7, 9)$ to ω_2 we estimate the vector of parameters $\tilde{\theta}$ by maximizing the log-likelihood $l(\tilde{\theta} \mid \mathbf{x})$ in (3.1.3). To this end, we make use of the theoretical results that we developed in Subsection 3.1.

In the case where we know to which population each observation belongs to we can explicitly calculate the estimates by making use of the relations in (3.1.8), (3.1.10). Indeed, since (x_1, x_3) belong to ω_1 we have that $z_{11} = z_{13} = 1$ and since (x_2, x_4) to ω_2 we have that $z_{22} = z_{42} = 1$. Therefore, the estimates for λ_1, λ_2 and π_1, π_2 obtained by maximizing the likelihood are the following

$$\hat{\lambda_1} = \frac{x_1 + x_3}{2} = \frac{5}{2} = 2.5, \quad \hat{\lambda_2} = \frac{x_2 + x_4}{2} = \frac{16}{2} = 8, \tag{3.2.1}$$

and for π_1, π_2

$$\hat{\pi}_1 = \frac{\sum_{i=1}^4 z_{i1}}{\sum_{i=1}^4 (z_{i1} + z_{i2})} = \frac{1}{2}, \quad \hat{\pi}_2 = \frac{\sum_{i=1}^4 z_{i2}}{\sum_{i=1}^4 (z_{i1} + z_{i2})} = \frac{1}{2}. \tag{3.2.2}$$

3.3 Estimating parameters through EM

The Expectation-Maximization (EM) algorithm introduced in 1997 in [1]. It is an iterative method to find (local) maximum likelihood estimates of parameters in statistical models, where the model depends on unobserved latent variables. It provides an alternative method to find suitable estimates for the parametric vector $\boldsymbol{\theta}$ when the analytical approach to the maximization problem $\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \mathbf{X})$ is intractable. EM finds good estimates for $\boldsymbol{\theta}$ by augmenting the data $\mathbf{X}$ by adding some latent variables $\mathbf{Z}$.

Formally, suppose that our data $(\mathbf{X}, \mathbf{Z})$ depends on a vector of parameters $\boldsymbol{\theta}$. In this scenario $\mathbf{X}$ denotes the observed data while $\mathbf{Z}$ denotes the missing (or latent) data. EM starts with an initial guess $\boldsymbol{\theta}^{(0)}$ for the parametric vector and it proceeds iteratively to find better estimates for $\boldsymbol{\theta}$. Each iteration of the algorithm involves two steps:

(i) **E-step:** During the r-th iteration, this step involves the calculation of the expected value of the log-likelihood of $\boldsymbol{\theta}$ for the complete data w.r.t the conditional distribution $\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}^{(r)}$. Formally, in E-step we calculate the function $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(r)})$ where

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(r)}) &= \int_{\boldsymbol{Z}} \log L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Z}) \cdot f(\boldsymbol{Z} \mid \boldsymbol{\theta}^{(r)}, \boldsymbol{X}) \, d\boldsymbol{Z} \\ &= \mathbb{E}_{\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\theta}^{(r)}} \Big(\log L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Z}) \Big). \end{aligned} \tag{3.2.4}$$

(ii) **M-step:** The new estimates for $\boldsymbol{\theta}$ are given by maximizing $Q(\boldsymbol{\theta}, \theta^{(\mathbf{r})})$; i.e.

$$\boldsymbol{\theta}^{(r+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(r)}). \tag{3.2.3}$$

Several termination criteria can be used; e.g.

$$\left| \frac{l^{(r+1)} - l^{(r)}}{l^{(r+1)}} \right| \leq \text{tolerance}, \tag{3.2.5}$$

where $l^{(r)}$ is the log-likelihood of the complete data after iteration r. Another criterion involves the ℓ_∞-norm of the parametric vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$; i.e.

$$\max_{1\leq j\leq p} \left|\theta_j^{(r+1)} - \theta_j^{(r)}\right| \leq \text{tolerance}, \tag{3.2.6}$$

or the ℓ_2-norm of $\boldsymbol{\theta}$,

$$\sum_{j=1}^{p} \left(\theta_j^{(r+1)} - \theta_j^{(r)}\right)^2 \leq \text{tolerance}. \tag{3.2.7}$$

Below we summarize in pseudocode the algorithm using the ℓ_2-norm as termination criterion since is the criterion that we use to estimate $\boldsymbol{\theta} = (\lambda_1, \lambda_2, \pi_1, \pi_2)$.

Algorithm 4 Expectation-Maximization algorithm

```
Require: $\boldsymbol{\theta}^{(0)}$, tolerance.
1: $r \leftarrow 0$.
2: repeat
3:     E-step: Compute $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(r))}$.
4:     M-step: $\boldsymbol{\theta}^{(r+1)} \leftarrow \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}; \boldsymbol{\theta}^{(r)})$.
5:     $r \leftarrow r + 1$.
6: until $\sum_j^p \left(\theta_j^{(r)} - \theta_j^{(r-1)}\right)^2 \leq$tolerance.
7: return $\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(r)}$.
```

In our case the $\boldsymbol{X}$ corresponds to the observations $\boldsymbol{x} = (x_1, x_2, x_3, x_4) = (2, 7, 3, 9)$., where each x_i arises from the poisson mixture

$$\begin{aligned} \mathbb{P}(X = x_i) &= \pi_1 \mathbb{P}(X = x \,|\, x \in \omega_1) + \pi_2 \mathbb{P}(X = x \,|\, x \in \omega_2) \\ &= \pi_1 \cdot \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x_i!} + \pi_2 \cdot \frac{e^{-\lambda_2}\lambda_2^{x_i}}{x_i!}, \end{aligned} \tag{3.2.8}$$

$X \,|\, \omega_i \sim \text{Poisson}(\lambda_i)$, $\pi_1 + \pi_2 = 1$ and $i = 1, 2, 3, 4$. As opposed to Subsection 3.2 the information from which Poisson each observation is generated is unknown. To augment our data we introduce the latent variables $\boldsymbol{Z} = (z_{ij})$, $i = 1, 2, 3, 4$, $j = 1, 2$ as defined in (3.1.4). In (3.1.7) we showed that the log-likelihood for $\boldsymbol{\theta} = (\lambda_1, \lambda_2, \pi_1, \pi_2)$ and $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ can be written as

$$\begin{aligned} l(\boldsymbol{\theta} \,|\, \boldsymbol{x}, \boldsymbol{Z}) &= \sum_{i=1}^{4} z_{i1} \log \pi_1 - \lambda_1 z_{i1} + z_{i1} x_i \log \lambda_1 - z_{i1} \log x_i! \\ &= \sum_{i=1}^{4} z_{i2} \log \pi_2 - \lambda_2 z_{i2} + z_{i2} x_i \log \lambda_2 - z_{i2} \log x_i!. \end{aligned} \tag{3.2.9}$$

Now, (3.2.9) allows us to find a simple expression for $Q(\boldsymbol{\theta}; \boldsymbol{\theta}^{(r)})$ in the r-th iteration. Indeed, since $Q(\boldsymbol{\theta}; \boldsymbol{\theta}^{(r)})$ involves the calculation of the expected value of $l(\boldsymbol{\theta} \,|\, \boldsymbol{x}, \boldsymbol{Z})$ w.r.t the conditional distribution

$\boldsymbol{Z} \mid \boldsymbol{x}, \boldsymbol{\theta}^{(r)}$ we have that

$$\begin{aligned} Q(\boldsymbol{\theta}; \boldsymbol{\theta}^{(r)}) &= \mathbb{E}_{\boldsymbol{Z} \mid \boldsymbol{x}, \boldsymbol{\theta}^{(r)}} \left(l(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{Z}) \right) \\ & \sum_{i=1}^{4} \omega_{i1}^{(r)} \log \pi_1^{(r)} - \lambda_1 \omega_{i1}^{(r)} + \omega_{i1}^{(r)} x_i \log \lambda_1 - \omega_{i1}^{(r)} \log x_i! \\ &+ \sum_{i=1}^{4} \omega_{i2}^{(r)} \log \pi_2^{(r)} - \lambda_2 \omega_{i2}^{(r)} + \omega_{i2}^{(r)} x_i \log \lambda_2 - \omega_{i2}^{(r)} \log x_i!, \end{aligned} \tag{3.2.10}$$

where $\omega_{ij}^{(r)} = \mathbb{E}_{z_{ij}} \left(z_{ij} \mid \boldsymbol{x}; \boldsymbol{\theta}^{(r)} \right)$. But now, $z_{ij} \mid \boldsymbol{x}; \boldsymbol{\theta}^{(r)}$ is a Bernoulli trial with success probability $\mathbb{P}(z_{ij} = 1 \mid \boldsymbol{x}; \boldsymbol{\theta}^{(r)})$. Therefore,

$$\omega_{ij}^{(r)} = \mathbb{E}_{z_{ij}}(z_{ij} \mid \boldsymbol{x}; \boldsymbol{\theta}^{(r)}) = \mathbb{P}(z_{ij} = 1 \mid \boldsymbol{x}; \boldsymbol{\theta}^{(r)}). \tag{3.2.11}$$

Since, z_{ij} depends only on x_i we have that $\omega_{ij}^{(r)} = \mathbb{P}(z_{ij} = 1 \mid x_i; \boldsymbol{\theta}^{(r)})$. Hence, using the law of total probability we write

$$\begin{aligned} \omega_{ij}^{(r)} = \mathbb{P}(z_{ij} = 1 \mid x_i; \boldsymbol{\theta}^{(r)}) &= \frac{\mathbb{P}(z_{ij} = 1, X_i = x_i \mid \boldsymbol{\theta}^{(r)})}{\mathbb{P}\left(X_i = x_i \mid \boldsymbol{\theta}^{(r)} \right)} \\ &= \frac{\mathbb{P}(X_i = x_i \mid z_{ij} = 1; \boldsymbol{\theta}^{(r)}) \mathbb{P}(z_{ij} = 1 \mid \boldsymbol{\theta}^{(r)})}{\sum_{k=1}^{2} \mathbb{P}(X_i = x_i \mid z_{ik} = 1; \boldsymbol{\theta}^{(r)}) \cdot \mathbb{P}(z_{ij} = 1 \mid \boldsymbol{\theta}^{(r)})} \\ &= \frac{\pi_j^{(r)} f_{X_i \mid \omega_j}(x_i; \boldsymbol{\theta}^{(r)})}{\sum_{k=1}^{2} \pi_k^{(r)} f_{X_i \mid \omega_k}(x_i; \boldsymbol{\theta}^{(r)})}, \end{aligned} \tag{3.2.12}$$

where by $f_{X_i \mid \omega_k}(x_i; \boldsymbol{\theta}^{(r)})$ we denote the probability mass function of $X_i \mid \omega_k \sim \text{Poisson}(\lambda_k^{(r)})$ given the estimates $(\lambda_1, \lambda_1, \pi_1, \pi_2)$ for the parametric vector $\boldsymbol{\theta}^{(r)}$ during the r-th iteration. Finally, taking partial derivatives with respect to each coordinate of $\boldsymbol{\theta} = (\lambda_1, \lambda_2, \pi_1, \pi_2)$ in (3.2.10) we get the new estimates $\boldsymbol{\theta}^{(r+1)}$ for the $\boldsymbol{\theta}$ in the r-th iteration. The calculations are essentially the same as shown in equations (3.1.8), (3.1.9), (3.1.10) and (3.1.11) but with ω_{ij} instead of z_{ij}. Hence, we obtain the following relations

$$\lambda_1^{(r+1)} = \frac{\sum_{i=1}^{4} \omega_{i1}^{(r)} x_i}{\sum_{i=1}^{4} \omega_{i1}^{(r)}}, \quad \lambda_2^{(r+1)} = \frac{\sum_{i=1}^{4} \omega_{i2}^{(r)} x_i}{\sum_{i=1}^{4} \omega_{i2}^{(r)}}, \tag{3.2.13}$$

and

$$\pi_1^{(r)} = \frac{\sum_{i=1}^{4} \omega_{i1}^{(r)}}{\sum_{i=1}^{4} (\omega_{i1}^{(r)} + \omega_{i2}^{(r)})}, \quad \pi_2^{(r)} = \frac{\sum_{i=1}^{4} \omega_{i2}^{(r)}}{\sum_{i=1}^{4} (\omega_{i1}^{(r)} + \omega_{i2}^{(r)})}. \tag{3.2.14}$$

But since $\omega_{i1}^{(r)} + \omega_{i2}^{(r)} = \mathbb{P}(z_{i1} = 1 \mid x_i; \boldsymbol{\theta}^{(r)}) + \mathbb{P}(z_{i2} = 1 \mid x_i; \boldsymbol{\theta}^{(r)}) = 1$ we can write (3.2.14) as

$$\pi_1^{(r)} = \frac{\sum_{i=1}^{4} \omega_{i1}^{(r)}}{4}, \quad \pi_2^{(r)} = \frac{\sum_{i=1}^{4} \omega_{i2}^{(r)}}{4}. \tag{3.2.15}$$

Summarizing the above results we obtain the following simplified version of EM in our case of the mixture of two Poisson's with parametric $\boldsymbol{\theta} = (\lambda_1, \lambda_2, \theta_1, \theta_2)$ and $\boldsymbol{x} = (x_1, x_2, x_3, x_4) = (2, 7, 3, 9)$ the observed data.

Algorithm 5 EM algorithm to estimate $\boldsymbol{\theta} = (\lambda_1, \lambda_2, \theta_1, \theta_2)$

```
Require: $\boldsymbol{\theta}^{(0)} = (\lambda_1^{(0)}, \lambda_2^{(0)}, \theta_1^{(0)}, \theta_2^{(0)})$, tolerance.
1: $r \leftarrow 0$.
2: repeat
3:     E-step: Compute the weights $\omega_{ij}^{(r)}$, $i = 1,2,3,4,\ j = 1,2$ according to (3.2.12).
4:     M-step: Compute $\boldsymbol{\theta}^{(r+1)} = (\lambda_1^{(r+1)}, \lambda_2^{(r+1)}, \theta_1^{(r+1)}, \theta_2^{(r+1)})$ according to (3.2.14).
5:     $r \leftarrow r + 1$.
6: until $\sum_{j=1}^{2} \left(\lambda_j^{(r)} - \lambda_j^{(r-1)}\right)^2 \leq$tolerance.
7: return $\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(r)}$.
```

Below we implement Algorithm 4 in R. To initialize the parametric vector $\boldsymbol{\theta}^{(0)}$ we choose $\pi_1^{(0)}$ to be any number selected in $(0,1)$ according to $U(0,1)$ and we set $\pi_2^{(0)} = 1 - \pi_1^{(0)}$. For the parameters $\lambda_1^{(0)}, \lambda_2^{(0)}$ we randomly choose two points uniformly in $(0,10)$. Algorithm 4 is implemented by a custom function named *EM_poisson_mixture*. It takes as arguments the given samples $\boldsymbol{x} = (2,7,3,9)$, the initial parameters $\boldsymbol{\pi}^{(0)} = (\pi_1^{(0)}, \pi_2^{(0)})$, $\boldsymbol{\lambda}^{(0)} = (\lambda_1^{(0)}, \lambda_2^{(0)})$ and the tolerance which is set to 10^{-10}. The functions returns the number of iterations needed to converge according to the termination condition, the final estimates for $(\hat{\lambda}_1, \hat{\lambda}_2)$, $(\hat{\pi}_1, \hat{\pi}_2)$ and a matrix containing the updates for (λ_1, λ_2) in each iteration. Below you can see the main body of the function and the corresponding results.

```r
# Implementation of EM for Poisson Mixture
samples <- c(2,7,3,9) # The observed data

EM_poisson_mixture <- function(samples,p,l,tol=1e-10){
  n <- length(samples)
  lambdas <- matrix(l,ncol = 2) # Matrix to store the lambda1,lambda2
  reps <- 0 # Counter for iterations
  repeat{
    # E - Step
    w <- matrix(rep(0,2*n), ncol = 2)
    for(i in 1:n){ # fill the matrix w
      p_total <- p[1]*dpois(samples[i], l[1]) +
        p[2]*dpois(samples[i],l[2])
      w[i,1] <- p[1]*dpois(samples[i],l[1])/p_total # eq. (3.2.12)
      w[i,2] <- p[2]*dpois(samples[i], l[2])/p_total} # eq. (3.2.12)
    # M - Step
    p_new <- apply(w,2,mean) # eq. (3.2.15)
    l_new <- apply(diag(samples) %*% w,2,sum)/apply(w,2,sum) # eq. (3.2.13)
    lambdas <- rbind(lambdas, l_new)
    reps <- reps+1
    if(sum((l_new - l)^2) <= tol){
      break
    }else{
      p <- p_new
      l <- l_new
    }}
  return(list(p=p_new, l=l_new, reps = reps,
              lambdas = lambdas))}
```

```r
set.seed(42) # For reproducability
p <- runif(1)
l <- runif(2, min =1, max = 10)

result <- EM_poisson_mixture(samples, p = c(p,1-p), l,
                             tol=1e-10)

print(sprintf("- EM converged after %d iterations",
              result$reps))

## [1] "- EM converged after 23 iterations"

sprintf("- Estimate for (p1,p2)=(%.4f, %.4f)",
        result$p[1],result$p[2])

## [1] "- Estimate for (p1,p2)=(0.5440, 0.4560)"

sprintf("- Estimate for (l1,l2)=(%.4f, %.4f)",
        result$l[1],result$l[2])

## [1] "- Estimate for (l1,l2)=(7.4017, 2.6831)"
```

As we can see the algorithm converged after 23 iterations with $(\hat{\pi}_1, \hat{\pi}_2) = (0.5440, 0.4560)$ and $(\hat{\lambda}_1, \hat{\lambda}_2) = (7.4017, 2.6831)$, estimates which are reasonable if we consider the estimates found in Subsection 3.1 using the prior knowledge. In Table 2 we see the estimates for (λ_1, λ_2) in some iterations of the algorithm.

Table 2: The values of the estimates for (λ_1, λ_2)

Iteration	$\lambda_1^{(r)}$	$\lambda_2^{(r)}$
r=0	9.433679	3.575256
r=5	7.310406	2.652603
r= 10	7.394137	2.679885
r=15	7.401101	2.682865
r=20	7.401692	2.683129
r=23	7.401734	2.683149

4 Linear regression & Variable selection

4.1 The framework of linear regression

In this task we are concerned with a regression task. We produce three different linear models using different criteria. We use the method of Lasso for variable selection and exploit techniques of cross validation to choose the best model with respect to its predictive capability.

In the linear regression framework we consider an experiment in which p characteristics of n samples are measured. The data from this experiment are denoted by $\mathbf{X}$, where

$$\mathbf{X} = \begin{pmatrix} X_{11} & \dots & X_{1p} \\ \vdots & \ddots & \vdots \\ X_{n1} & \dots & X_{np} \end{pmatrix}.$$

The matrix $\mathbf{X}$ is called the *design matrix*. Additional information of the samples is available in the form of $\mathbf{Y} = (Y_1, \dots Y_n)$ referred to as the *response variable*. The aim of linear regression analysis is to explain $\mathbf{Y}$ in terms of $\mathbf{X}$ through a relationship of the form

$$Y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{1p} + \epsilon_i, \quad i = 1, \dots, p, \tag{4.1.1}$$

where $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_p)^T$ is the *regression parameter*. The parameter β_j, $j = 1, \dots, p$ represents the effect size of covariate j on the response. That is, for each unit change in covariate j the observed change in response is equal to β_j. The summand ϵ_i in (4.1.1) is the *error* term. It is assumed that $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ and that the ϵ_i's are independent, i.e. $\mathbb{E}(e_i e_j) = \mathbb{E}(e_i)\,\mathbb{E}(e_j)$ and

$$\text{Cov}(\epsilon_i, \epsilon_j) = \begin{cases} \sigma^2, & i = j \\ 0, & i \neq j \end{cases}.$$

Now, eq (4.1.1) is often written in a condensed matrix form:

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & X_{11} & \dots & X_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & \dots & X_{np} \end{pmatrix}}_{\mathbf{X}'} \cdot \begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_p \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix} \implies \mathbf{Y} = \mathbf{X}' \cdot \boldsymbol{\beta} + \epsilon. \tag{4.1.2}$$

The estimated parameters $\hat{\boldsymbol{\beta}} = (\beta_0, \beta_1, \dots, \beta_p)$ are determined by minimizing the sum of squares

$$\text{SSE} = \sum_{i=1}^{n} (Y_i - X_i'^T \cdot \boldsymbol{\beta})^2 = \left\| \boldsymbol{Y} - \boldsymbol{X}' \cdot \boldsymbol{\beta} \right\|_2^2, \tag{4.1.3}$$

where $X_i'^T = (1, X_{i1}, \dots, X_{ip})$, $i = 1, \dots, n$. It is proven that $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$.

In our case, the dataset consists of ten baseline - independent variables, age, sex, body mass index, average blood pressure, and six blood serum measurements (tc, ldl, hdl, tch, ltg, glu) obtained for each $n = 442$ diabetes patients, as well as the response interest, a quantitative measure of disease progression one year after baseline. The values corresponding to each independent variable have been normalized in such a way that their sum is equal to 0 and their respective sum of squares equal to 1. With the following lines of code we load the data using the library *lars*. Moreover, we import all the necessary packages that we will use throughout this task and perform some data pre-processing to store the data in a *data.table* having as columns the target response variable $\boldsymbol{Y}$ and the 10 independent variables, and as rows the 442 observations.

```r
# Exercise 4 - Variable selection
# Importing libraries
library(lars)

## Loaded lars 1.2

library(data.table)
library(sys)

data("diabetes") # Getting the data
N <- length(diabetes$y)
# Storing data in a data table
dt <- data.table(y = diabetes$y, age = diabetes$x[1:N,1],
                 sex = diabetes$x[1:N,2], bmi = diabetes$x[1:N,3],
                 map = diabetes$x[1:N,4], tc = diabetes$x[1:N,5],
                 ldl = diabetes$x[1:N,6], hdl = diabetes$x[1:N,7],
                 tch = diabetes$x[1:N,8],ltg = diabetes$x[1:N,9],
                 glu = diabetes$x[1:N,10])
```

In the following subsections we construct three different models and compare them according to their predictive power to obtain the final model. The first model is obtained via minimizing the *BIC* index. The other two models are obtained by variable selection according to *LASSO* regression. We first begin by determining the first model.

4.2 Full enumeration - Minimizing *BIC*

The first model is defined to be the model M_1 for which the quantity

$$\mathrm{BIC}(M) = k \log n - 2 \log \left(\hat{L}\right), \tag{4.2.1}$$

for a given model M is minimized. In (4.2.1), $\hat{L}$ is the value of the likelihood function of the model M, i.e. $\hat{L} = p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}}, M)$, where $\hat{\boldsymbol{\theta}}$ are the parameter values that maximize the likelihood function. By $\boldsymbol{x}$ we denote the observed data, n is the number of data points in $\boldsymbol{x}$ and k is the number of parameters (the coefficients of the independent variables) estimated by the model. The purpose of BIC is to combat overfitting. In general, an increased number of estimated parameters leads to overfitting resulting in poor generalization. The purpose of the quantity in (4.2.1) is to achieve a balance between the number of parameters k and the value of the maximized likelihood. In other words, minimizing BIC results in models that still describe the training data but at the same time exhibit good generalization.

To determine M_1, we denote by $\mathcal{M}$ the family of all possible models for our task at hand. Since we have 10 independent variables the cardinal number of $\mathcal{M}$ will be equal to $|\mathcal{M}| = 2^{10} = 1024$. In other words, the number of all possible linear models that we can generate for our problem are 1024. These models are generated by deciding each independent variable whether to include it or not in the final model. Hence, since we have 10 independent variables we have 2^{10} options in total. Since the number 1024 is small we can examine all models one by one and find the one with the minimum BIC. This procedure is handled by a custom function named *Minimize_bic*. It takes as arguments the *dataset* to fit the models and the name of the response variable which is "y" in our case, and returns the model M_1 with the minimum BIC. The function is designed in such a way in order to work for other target names as well. Therefore, before entering the enumeration process it checks whether the user has given a different target name other than "y". Moreover, it places the target variable in the first column of the *data.table*. The enumeration process is handled as follows: Assuming our data consists of n observations and p independent variables; using the method *expand.grid*, the function first creates a binary matrix called *binarization* of dimensions $2^p \times p$. Each row consists of p columns with zeros or ones. A value of 1 in the k-th column, for $1 \leq k \leq p$, indicates that the k-th independent variable should be included in the model. Therefore, by iterating over all rows of this binary matrix we examine all possible models. A temporary variable keeps track of the value of BIC for the model chosen in each iteration and it is compared to the current best model's BIC obtained until that point. Below you can see the main body of the function *Minimize_bic* and the obtained results for the *diabetes* dataset.

```r
# Find the best model by minimizing BIC score
# Full enumeration
Minimize_bic <- function(data, target_name = "y"){
  dt <- data
  variables <- names(data)
  # Change the target name to "y"
  target_pos <- which(names(data) == target_name)
  variables <- replace(variables, c(1,target_pos),
                       c(target_name,variables[1]))
  variables[1] <- "y"
  names(dt) <- variables
  num_variables <- length(variables)
  dt <- dt[,..variables]
  x <- replicate(length(variables)-1,c(0,1))
  # Create the binary matrix using expand.grid
  binarization <- expand.grid(lapply(seq_len(ncol(x)),
                                     function(i) x[,i]))

  num_models <- length(binarization[,1]) # Number of total models
  best_model <- lm(y~., data = dt[,"y"]) # The constant model
  min_bic <- BIC(best_model) # Its BIC value
  best_variables <- c()

  tic <- Sys.time() # To measure the execution time
  for(i in 2:num_models){ # Examine all models and compare BIC's
    indices <- which(binarization[i,]>0)+1
    temp_variables <- c("y", variables[indices])
    temp_model <- lm(y~., data = dt[,..temp_variables])
    temp_bic <- BIC(temp_model)
    if(temp_bic<min_bic){
      best_model <- temp_model
      min_bic <-temp_bic
      best_variables <- temp_variables[-1]
    }
  }
  tac <- Sys.time()
  # Print the results on screen
  print(sprintf("- Exhaustive search finished in %.5f sec(s).",
                tac-tic))
  print(sprintf("- %d models were examined!", num_models))
  print(sprintf("- The best model contains the variables %s",
          paste(best_variables,collapse=",")))
  print(sprintf("- BIC value: %.4f",BIC(best_model)))
  return(best_model) # Return the best model
}
M1 <- Minimize_bic(dt) # Apply full enumeration

## [1] "- Exhaustive search finished in 1.93025 sec(s)."
## [1] "- 1024 models were examined!"
## [1] "- The best model contains the variables sex,bmi,map,hdl,ltg"
## [1] "- BIC value: 4822.9020"
```

As we can see from the above results, the model M_1 contains the variables sex, bmi, map, hdl and ltg. The minimum BIC value among all models is equal to 4822.9020. Below we can see the summary of M_1.

```
summary(M1)

##
## Call:
## lm(formula = y ~ ., data = dt[, ..temp_variables])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -150.03  -39.32   -0.56   37.31  148.38
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  152.133      2.585  58.849  < 2e-16 ***
## sex         -235.776     60.469  -3.899 0.000112 ***
## bmi          523.562     65.294   8.019 9.94e-15 ***
## map          326.236     63.084   5.171 3.55e-07 ***
## hdl         -289.117     65.645  -4.404 1.34e-05 ***
## ltg          474.292     65.683   7.221 2.32e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54.35 on 436 degrees of freedom
## Multiple R-squared:  0.5086,Adjusted R-squared:  0.503
## F-statistic: 90.26 on 5 and 436 DF,  p-value: < 2.2e-16
```

Looking at p - values we see that every variable is statistically significant. The residual standard error is 54.35 on 436 degrees of freedom. The equation for M_1 is given by

$$\begin{aligned}\hat{Y} &= 152.133 - 235.776 \cdot \text{sex} + 523.562 \cdot \text{bmi} \\ &\quad + 326.236 \cdot \text{map} - 289.117 \cdot \text{hdl} + 474.292 \cdot \text{ltg}.\end{aligned} \tag{4.2.2}$$

4.3 Variable selection with LASSO

To determine M_2 and M_3 we use the LASSO method. Instead of minimizing the sum of squares (SSE) given in (4.1.3), this method adds a penalty term to the SSE to combat overffiting and multicollinearity[4]. Formally, lasso regression finds an estimator of the parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^T$ through the minimization of

$$J(\beta_1, \ldots, \beta_p) = \left\| \boldsymbol{Y} - \boldsymbol{X}' \cdot \boldsymbol{\beta} \right\|_2^2 + \lambda \sum_{i=1}^{p} |\beta_i| \,, \tag{4.3.1}$$

where $\lambda > 0$ is a fixed parameter called the *penalty* parameter. Observe that in the case where $\lambda = 0$ then (4.3.1) coincides with the minimization of the SSE in (4.1.3). The penalty term added in (4.3.1) is essentially the ℓ_1 - norm of the parameters $(\beta_1, \ldots, \beta_p)$. Minimizing J with respect to $\beta_1, \ldots, \beta_p$ results in bounded values for the ℓ_1 - norm. This means that the penalty term does not allow the parameters to get high values. High values of the penalty parameter $\lambda > 0$ correspond to adding greater penalty to the parameters $\beta_1, \ldots, \beta_p$ meaning that more estimated parameters will be close to zero. In this way, we are reducing the capacity of the model in order to combat overfitting. In other words, we restrict our initial model space into a space that contains models that exhibit lower variance in the cost of higher bias. On the other hand, small values for the parameter λ corresponds to a larger space of models. Therefore, in this case we expect models with low bias but with higher variance. The goal of LASSO regression is to achieve a compromise between bias and variance in order to obtain a model that fits the data well but without completely losing its predictive power. To implement the LASSO method in R we use *glmnet* library. With the following and the help of *glmnet* we produce 88 values for the penalty parameter λ and we plot the values of the estimated parameters $\beta_1, \ldots, \beta_{10}$ for each value of λ.

```
# Importing the required package
library(glmnet)

## Loading required package:  Matrix
## Loaded glmnet 4.1-2

library(ggplot2)
library(tidyr)

##
## Attaching package:  'tidyr'
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack

library(dplyr)

##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:data.table':
##
##     between, first, last
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

[4]Multicollinearity is when two or more variables are linearly dependent with each other.

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggrepel)
library(latex2exp)
lasso <- glmnet(data.matrix(dt[,2:11]),dt$y)
betas = as.matrix(lasso$beta)
lambdas = lasso$lambda
names(lambdas) = colnames(betas)

as.data.frame(betas) %>% # Creating a dataframe for grouped plot
  tibble::rownames_to_column("variable") %>%
  pivot_longer(-variable) %>%
  mutate(lambda=lambdas[name]) %>%
  ggplot(aes(x=lambda,y=value,col=variable)) + geom_line() +
  geom_label_repel(data=~subset(.x,lambda==min(lambda)),
                   aes(label=variable),nudge_x=-0.5) +
  scale_x_log10()+
  labs(x=TeX("Penalty parameter $\\lambda$"),
       y="Value of estimated parameter")+
  theme(legend.position = "none")
```

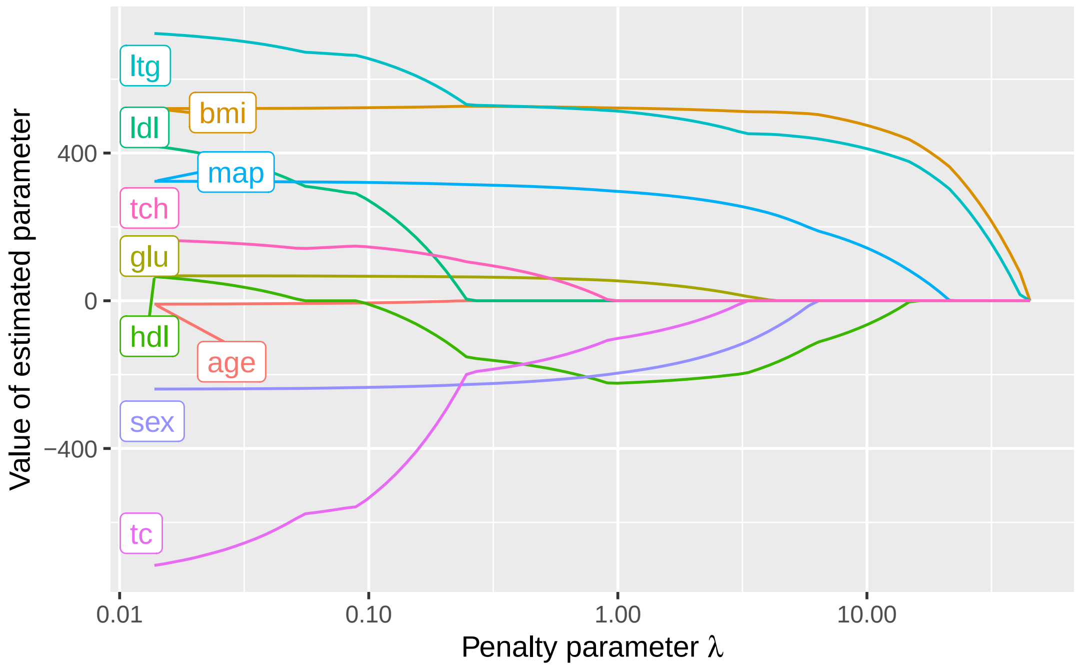

Figure 10: The values of the estimated parameters $\beta_1, \ldots, \beta_{10}$ with respect to the penalty parameter λ.

The values for the penalty parameter λ in Figure 10 start from 0.013 and end at 45.16. We observe that as the value of the penalty parameter grows then more parameter coefficients become zero. This is of course expected since by adding more penalty we restrict our model space. The four last variables that remain until they also become zero are bmi, ltg, map and hdl. As we can see from the summary presented previously for M_1 these variables seem to be the most significant. To select the most suitable values for the parameter λ we perform a *cross - validation* and then select the λ's that minimize the CV-MSE. More formally, using the *cv* method of *glmnet* we split our initial dataset consisting of the 442 observations into 10 folds. Then, for every parameter λ we obtain a model by minimizing the eq (4.3.1) in each of the 10 folds. For each fold we calculate the quantity CV-MSE which is the average mean squared error of the models obtained for the chosen penalty parameter λ trained on each fold. Below you can see the code produced in R for the *cv* method. We set the random state to be equal to 413 to obtain the same folds independently of each run.

```
# Performing a cross validation to find
# the suitable values for the penalty parameter
set.seed(413)
lasso2 <- cv.glmnet(data.matrix(dt[,2:11]), dt$y,
                    num_folds = 10)
plot(lasso2)
```

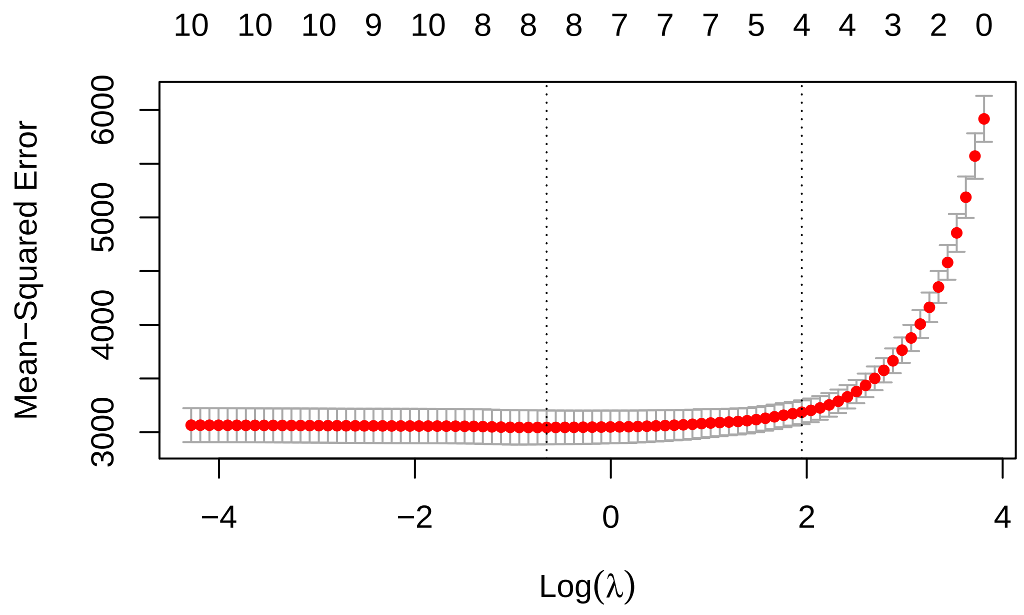

Figure 11: The graph of the CV-MSE with respect to each penalty parameter λ (in log-scale). Two special values along the λ sequence are indicated by the vertical dotted lines. The left dotted line is the value of λ that gives the minimum CV-MSE, while the right dotted line is the value of λ that gives the most regularized model such that the CV-MSE is within one standard error of the minimum. Above the parallel axis we can see the number of parameters included in the model with respect to the sequence λ.

Now, the model M_2 is the model that corresponds to the penalty parameter $\lambda > 0$ that achieves the minimum CV-MSE and M_3 is the model that corresponds to the penalty parameter which is one standard error of the minimum CV-MSE. Below we see these values.

```r
print(paste("- Minimum CV-MSE:", lasso2$lambda.min))

## [1] "- Minimum CV-MSE: 0.519230606443435"

print(paste("- One standard deviation:",lasso2$lambda.1se))

## [1] "- One standard deviation: 7.02543814837611"
```

The number $\lambda_{\text{min}} = 0.51923$ corresponds to a model with low bias but low variance while the number $\lambda_{\text{se}} = 7.02543$ corresponds to a model with higher bias and lower variance. Below we can also see the independent variables that correspond to the non zero estimated parameters $\beta_1, \dots, \beta_{10}$ for λ_{min} and λ_{se}.

```r
# M2
coefs1 <- coef(lasso2, s = "lambda.min")
coefs1 <- c("y",rownames(coefs1)[coefs1[,1]!=0][-1])
print(sprintf("- lmin: %s",
                  paste(coefs1[-1],collapse=",")))

## [1] "- lmin: sex,bmi,map,tc,hdl,tch,ltg,glu"

# M3
coefs2 <- coef(lasso2)
coefs2 <- c("y",rownames(coefs2)[coefs2[,1]!=0][-1])
print(sprintf("- lse: %s",
              paste(coefs2[-1],collapse=",")))

## [1] "- lse: bmi,map,hdl,ltg"
```

As we can see the model that corresponds to λ_{se} has the variables bmi, map, hdl, ltg. Although, this result depends on this particular random state that we used. For different values of random state we get either the variables bmi, map, hdl, ltg or the model with variables sex, bmi, map, hdl, ltg, which is the same as M_1. To have a variety of models we proceed with the model containing the variables bmi, map, hdl, ltg as M_3.

4.4 Model selection through 5-fold CV

Now, in order to decide which variables are the best to include in our model we will proceed by a 5-fold cross validation. More precisely, we create a custom function named *cross_val* that takes as input the data and a list containing the variables of each model. It splits the dataset into 5 folds and performs a cross validation. Denote the folds of the dataset by $C_1, \dots C_5$ and by M_1, M_2, M_3 the three models (according to the variables that will be included). In the i-th iteration the models are trained on the data samples from $\bigcup\limits_{\substack{j=1 \\ j\neq i}}^{5} C_j$ according to the minimization of the SSE (without the penalty term). Then, each model is tested on C_i and the root mean squared error is calculated:

$$\text{RMSE}(M_k)_i = \sqrt{\frac{\sum_{\boldsymbol{x}\in C_i}(\hat{y}_k(\boldsymbol{x}) - y(\boldsymbol{x}))^2)}{|C_i|}}, \quad k = 1, 2, 3, \tag{4.1.4}$$

where by $\hat{y}_k(x)$ we denote the prediction of the model when the input is $\boldsymbol{x} = (x_1, \ldots, x_{10})$ and by $y(\boldsymbol{x})$ the true value corresponding to $\boldsymbol{x}$. The function returns the average of the RMSE scores for each model with respect to each iteration. Moreover, we create a custom function named *rmse* to calculate the root mean squared error as shown in (4.1.4). Below you can see the corresponding code and the results printed on screen.

```
rmse <- function(actual,preds){
  return(sqrt(mean((actual-preds)^2)))}

cross_val <- function(data, models, num_folds = 5, seed=42){
  set.seed(seed)
  dt_shuffled <- data[sample(nrow(data)),]
  folds <- cut(seq(1,nrow(dt_shuffled)),
               breaks = num_folds, labels = FALSE)
  RMSE <- rep(0,length(models))
  for(i in 1:length(models)){
    names(RMSE)[i] <- paste("M", as.character(i), sep ="")
  }
  for(i in 1:num_folds){
    indices <- which(folds == i, arr.ind = TRUE)
    test_data <- dt_shuffled[c(indices),]
    train_data <- dt_shuffled[-c(indices),]

    for(i in 1:length(models)){
      variables <- c(array(unlist(models[i])),"y")
      model <- lm(y~., data = train_data[,..variables])
      preds <- predict(model, test_data[,-"y"])
      RMSE[i] <- RMSE[i] + rmse(array(unlist(test_data[,"y"])),preds)
    }}
  return(RMSE/num_folds)}
M1_vars <- names(M1$coefficients)[-1]
M2_vars <- coefs1[-1]
M3_vars <- coefs2[-1]
models <- list(M1 = M1_vars, M2 = M2_vars,
               M3 = M3_vars)
RMSE <- cross_val(dt, models)
RMSE

##       M1       M2       M3
## 55.44567 55.48494 56.09423
```

In Table 3 we give a summary of results obtained from the cross - validation. As we can see M_1 has the lowest CV-MSE and therefore has the best predictive capabilities.

Table 3: The Models M_1, M_2, M_3

Model	**Variables**	**CV-MSE**
M_1	sex, bmi, map, hdl and ltg	55.44567
M_2	All variables	55.48494
M_3	bmi, map, hdl, ltg	56.09423

For completion, below we present the summary of the models M_2 and M_3.

```
M2 <- lm(y~., data = dt[,..coefs1])
summary(M2)

##
## Call:
## lm(formula = y ~ ., data = dt[, ..coefs1])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -153.135  -38.273   -1.242   37.870  153.602
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  152.133      2.576  59.062  < 2e-16 ***
## sex         -236.851     60.854  -3.892 0.000115 ***
## bmi          528.630     66.239   7.981 1.32e-14 ***
## map          320.896     64.220   4.997 8.47e-07 ***
## tc          -229.533    113.500  -2.022 0.043758 *
## hdl         -125.494    138.725  -0.905 0.366168
## tch          146.504    159.936   0.916 0.360169
## ltg          535.644     78.210   6.849 2.56e-11 ***
## glu           68.158     65.394   1.042 0.297866
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54.15 on 433 degrees of freedom
## Multiple R-squared:  0.5155,Adjusted R-squared:  0.5066
## F-statistic: 57.59 on 8 and 433 DF,  p-value: < 2.2e-16

M3 <- lm(y~., data = dt[,..coefs2])
summary(M3)

##
## Call:
## lm(formula = y ~ ., data = dt[, ..coefs2])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -142.104  -40.308   -3.297   39.140  151.833
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  152.133      2.627  57.916  < 2e-16 ***
## bmi          555.279     65.829   8.435 4.88e-16 ***
## map          269.676     62.383   4.323 1.91e-05 ***
## hdl         -193.954     61.922  -3.132  0.00185 **
## ltg          484.979     66.684   7.273 1.64e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 55.23 on 437 degrees of freedom
## Multiple R-squared:  0.4915,Adjusted R-squared:  0.4868
## F-statistic: 105.6 on 4 and 437 DF,  p-value: < 2.2e-16
```

References

[1] Arthur P Dempster, Nan M Laird, and Donald B Rubin. “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[2] Bradley Efron. “Bootstrap methods: another look at the jackknife”. In: *Breakthroughs in statistics*. Springer, 1992, pp. 569–593.

[3] Peter E Hart, David G Stork, and Richard O Duda. *Pattern classification*. Wiley Hoboken, 2000.

[4] Paul G Hoel, Sidney C Port, and Charles J Stone. “Introduction to Probability Theory, 1971”. In: *Houghto n Mifflin, Boston, MA* ().

[5] Maurice H Quenouille. “Notes on bias in estimation”. In: *Biometrika* 43.3/4 (1956), pp. 353–360.

[6] Sheldon M Ross. *A first course in probability*. Pearson, 2014.

[7] John Tukey. “Bias and confidence in not quite large samples”. In: *Ann. Math. Statist.* 29 (1958), p. 614.

[8] Aad W Van der Vaart. *Asymptotic statistics*. Vol. 3. Cambridge university press, 2000.

[9] Wessel N van Wieringen. “Lecture notes on ridge regression”. In: *arXiv preprint arXiv:1509.09169* (2015).