National Technical University of Athens
MSc - Data Science and Machine Learning

Computational Statistics and Stochastic Optimization

Semester Project

Vasileios Depastas
MSc student

A.M: 03400131
vasileiosdepastas@mail.ntua.gr

June 2022

Contents

List of Figures

Introduction

The present report employs a range of statistical and stochastic methods in order to solve a range of problems by yielding the power of the programming language R and whenever possible verify simulation results theoretically. We briefly describe below the four assignments this report deals with.
In the first exercise, initially we explore the rejection sampling method to sample from a known distribution. Next, we experiment with an expected value estimator and its Rao-Blackwellized version. Moreover, values from a statistical function of random variables following the uniform distribution are simulated and a histogram of the resulting distribution is generated. Finally, we calculate the function's standard error using Bootstrap and Jackknife methods and generate a histogram to compare with the previous method.
Following on, the second exercise examines the "Old Faithful geyser" dataset, which contains the times in minutes of consecutive explosions of a volcano in the USA. As a first step, we estimate the probability density function (pdf) of the data using an Epanechnikov kernel and determine the optimal kernel width by maximizing the cross-validated likelihood. The estimated pdf is plotted in two different ways and the results are compared. Lastly, the probability for two consecutive explosions to be more than 3.5 minutes apart is calculated by integrating an appropriate region under the estimated pdf curve as well as by simulating values from the estimated pdf and the two results are then compared.
Exercise three demonstrates how Maximum Likelihood is used for parameter estimation and subsequently the Expectation Maximization algorithm is employed for the same task when some information is missing.
Finally, exercise four explores the full space of possible models in the feature/variable selection problem given a diabetes prediction dataset containing ten variables by utilizing the BIC criterion. In addition to that, the Lasso method is applied for the same task and cross-validation is used to calculate the lambda penalty parameter of the method. Subsequently, based on two different parameter values, two models are specified by the inclusion of appropriate predictors to the models, and finally we compare all three models on their predictive power using 5-fold cross-validation and each model's Root Mean Square Error (RMSE) on the validation fold each time.

Exercise 1 - Stochastic simulation

a - Rejection sampling

The initial assignment of the first exercise is an application of rejection sampling. The distribution with pdf $f(x)$ that is being sampled with the rejection sampling method is a standard normal distribution $N(0,1)$ and a sample comprised of 1000 sample points is generated using code in the R programming language. The proposal distribution is the Cauchy distribution with pdf $g(x) = \frac{1}{\pi(1+x^2)}$, $x \in \mathbb{R}$, from which samples are taken using inverse sampling.

Let $X \sim N(0,1)$ be a random variable. Then its probability density function (pdf) is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right), \ x \in \mathbb{R}.$$

The rejection sampling algorithm is then implemented as detailed below.

Rejection sampling:

- **Step 1**

  Generate $Y \sim \text{Cauchy}(1,0)$ i.e. $Y \sim g(y)$ and set $y = Y$. This is done using inverse sampling, following the next three steps:

  1. generate $U \sim U(0,1)$
  2. $Y = \tan\left(\pi\left(U - \frac{1}{2}\right)\right)$
  3. $Y \sim \text{Cauchy}(1,0)$

  Therefore, by generating uniform sample points we were able to sample from g. Please note here that we used the following for the aforementioned inverse sampling method.

$$Y \sim \text{Cauchy}(1,0), \quad g(y) = \frac{1}{\pi\left(1+y^2\right)}, \quad y \in \mathbb{R},$$
$$G(y) = \int_{-\infty}^{y} \frac{1}{\pi\left(1+s^2\right)} ds = \frac{1}{2} + \frac{1}{\pi}\arctan x = u,$$
$$G^{-1}(y) = \tan\left(\pi\left(u - \frac{1}{2}\right)\right).$$

- **Step 2**

  Generate $U \sim U(0,1)$ and set $u = U$.

- **Step 3**

  if $u \leq \frac{f(y)}{M \cdot g(y)}$ then $X = y$ else go back to step 1.

To find the optimal M for the algorithm, we want to find the minimum M value that satisfies the condition $f(y) \leq Mg(y)$ or equally $M \geq \frac{f(y)}{g(y)} = h(y),\ \forall\, y \in \mathbb{R}$. Therefore, the minimum M value that satisfies the previous condition will also be the optimum M we are looking for. That is the global maxima of $h(y)$ then.
Taking the derivative of $h(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right) \pi\left(1+y^2\right)$ we obtain

$$h'(y) = \sqrt{\frac{\pi}{2}} \cdot y \cdot e^{-\frac{y^2}{2}} \left(1 - y^2\right),\ y \in \mathbb{R}.$$

We can easily see that $h'(y)$ has three real roots, $0, -1$ and 1. Its second derivative is

$$h''(y) = \sqrt{\frac{\pi}{2}} \cdot e^{-\frac{y^2}{2}} \left(1 - 4y^2 + y^4\right),\ y \in \mathbb{R},$$

and we can see that $h''(0) = \sqrt{\frac{\pi}{2}} > 0$ while $h''(1) = h''(-1) = -2\sqrt{\frac{\pi}{2}} \cdot e^{-\frac{1}{2}} < 0$, therefore we have a local minima at $y = 0$ and local maxima at $y = -1$ and $y = 1$ with h being an even function. By subtituting in h, we get $h(1) = h(-1) = \sqrt{2\pi}e^{-\frac{1}{2}} \approx 1.52$ and we can prove that those are also global maxima of the function. Thus, $M_{opt} = \sqrt{2\pi}e^{-\frac{1}{2}} \approx 1.52$. For demonstration purposes, we present a plot of the function below using R code.

```r
library(ggplot2)

# f(x)
f <- function(x) {
  return((1/sqrt(2*pi))*exp(-x^2/2))
}

# g(x)
g <- function(x) {
  return(1/(pi*(1+x^2)))
}

# show h(x)=f(x)/g(x) that is used to select the best M
x = seq(-5, 5, 0.05) # create a sequence from -5 to 5 with step 0.05
h <- function(x) f(x)/g(x) # define h function
h_values = h(x) # caluclate the values in order to find max
max_h = max(h_values) # get M as the maximum value of h
M_func <- function(x) max_h # define M constant function

figure2 <- ggplot(data = data.frame(x=x), mapping = aes(x=x)) +
  stat_function(fun=h, mapping = aes(color="h(x)"), size=1.1) +
  stat_function(fun=M_func, mapping = aes(color="optimal M"), size=1.1) +
  scale_color_manual(name = "Function:",
                     values = c("darkblue", "indianred"), # Color specification
                     labels = c("h(x)", "optimal M")) +
  theme(legend.position = c(0.9, 0.7)) +
  labs(title = "Graph of h(x) and optimal M value") # title


figure2 # show figure
```

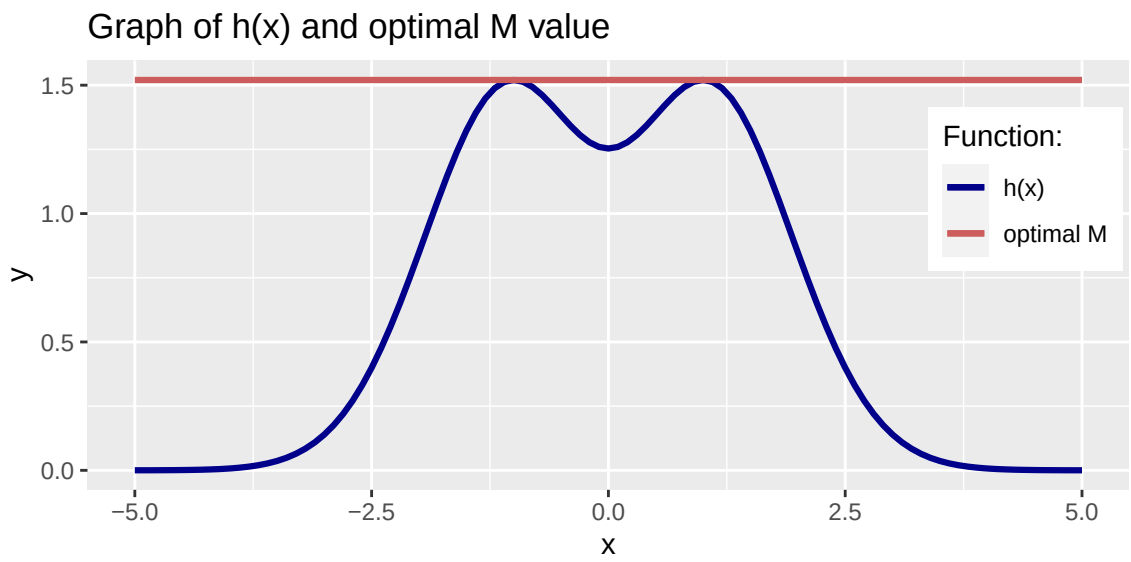

Figure 1: (1a) - Graph of $h(x)$ and optimal M

Selecting the optimal M, we can then verify graphically using R that

$$f(y) \leq M_{opt} \cdot g(y), \ \forall \ y \in \mathbb{R}.$$

```r
# show f(x) and M_optimal * g(x) in a single plot
G <- function(x) max_h * g(x) # G = M*g(x)

figure3 <- ggplot(data = data.frame(x=x), mapping = aes(x=x)) +
  stat_function(fun=f, mapping = aes(color="f(x)"), size=1.1) +
  stat_function(fun=G, mapping = aes(color="M*g(x)"), size=1.1) +
  scale_color_manual(name = "Function:",
                     values = c("darkblue", "indianred"), # Color specification
                     labels = c("f(x)", "M*g(x)")) +
  theme(legend.position = c(0.9, 0.7)) +
  labs(title = "Graph of f(x) and M*g(x) with optimal M") # title


figure3 # show figure
```

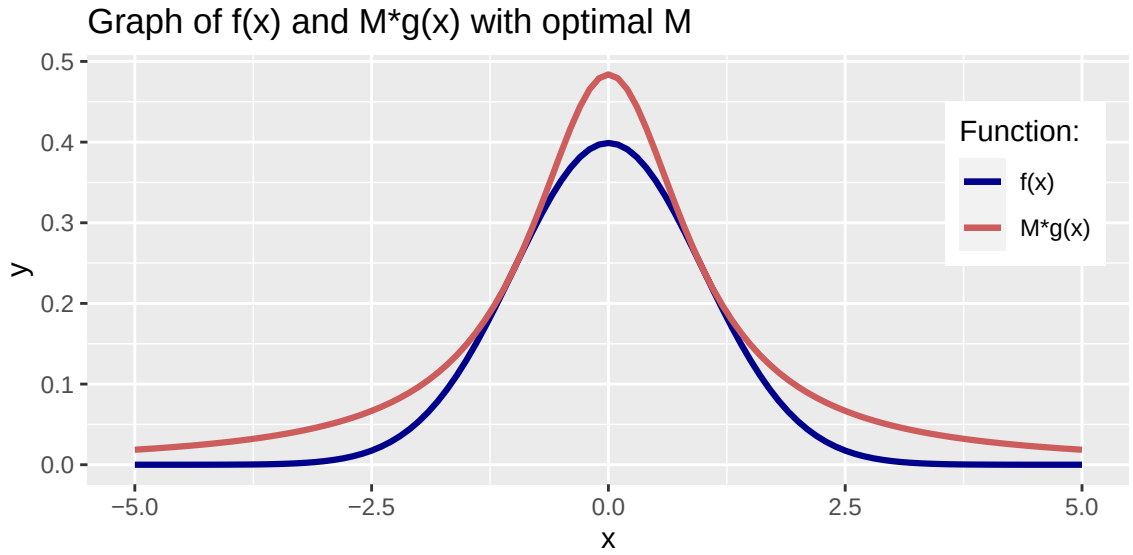

Figure 2: (1a) - Graph of $f(x)$ and $M_{opt}g(x)$

Once we have determined the M_{opt} value, we can go ahead and generate a histogram using 1000 simulated values from the standard normal distribution $N(0, 1)$, where we selected the optimal binwidth for the histogram as below

$$h_{opt} = 3.491 \cdot \sigma \cdot n^{-\frac{1}{3}} = 3.491 \cdot 1 \cdot 1000^{-\frac{1}{3}} \approx 0.3491.$$

Below is the code that generates the histogram of the simulated values using the rejection sampling method, the simulated probability density function and the standard normal probability density function in a single figure. We have also included a random seed at the top of the code block for reproducibility purposes.

```r
# 1.a - Rejection Sampling
# Sample from N(0,1) using utilizing a Cauchy proposal distribution

set.seed(42) # generate reproducible results across runs

# --- IMPORTS ---
library(ggplot2)

# --- FUNCTIONS ---
```

```r
# returns Cauchy sample derived from transformed inverse sample
cauchy_sampling <- function(uniform_sample) {
  transformed_sample <- tan(pi * (uniform_sample-1/2))
  return(transformed_sample)
}

# acceptance probability
acceptance_prob <- function(M, sample) {
  probability <- f(sample)/(M*g(sample)) # acceptance probability
  return(probability)
}

# --- MAIN ---
M = sqrt(2*pi/exp(1)) # optimal M constant multiplier
num_total_samples = 0 # counter of total samples generated
num_accepted_samples = 0 # counter of samples accepted (<= total)
required_samples = 1000 # number of accepted samples to be generated
accepted_samples = c() # save accepted samples' values

while (num_accepted_samples < required_samples){
  num_total_samples = num_total_samples + 1 # calculate all samples generated
  uniform_sample = runif(1, min=0, max=1) # sample from uniform distribution
  cauchy_sample = cauchy_sampling(uniform_sample) # transform to get cauchy sample
  accept_probability = acceptance_prob(M, cauchy_sample) # acceptance prob
  uniform_probability = runif(1, min=0, max=1) # gerete number in [0,1]
  if (accept_probability >= uniform_probability){
    num_accepted_samples  = num_accepted_samples + 1
    accepted_samples = append(accepted_samples, cauchy_sample)
  }
}

# --- PLOTTING ---

opt_h <- 3.491 * 1 * required_samples^(-1/3) # Normal dist, optimal bin width

# convert accepted samples list to df to use ggplot
df <- data.frame(x = accepted_samples)

figure <- ggplot(data=df, aes(x)) +
    geom_histogram(
      binwidth = opt_h, # bin width = optimal
      aes(y = ..density..),
      color = "white", # histogram bins borders
      fill = "indianred" # histogram bins fill color
    ) +
    geom_density(size=1.1,  # plot simulated distribution curve
                 kernel="gaussian",
                 mapping = aes(color="Simulated"),
                 key_glyph = draw_key_path) +
    stat_function(fun = f, # plot N(0,1) pdf curve
                  size=1.1,
                  mapping = aes(color="Standard normal"),
                  key_glyph = draw_key_path) +
    scale_color_manual(name = "Pdf:",
                       values = c("red", "darkblue"), # Color specification
                       labels = c("Simulated - Gaussian kernel", "Standard normal")) +
    theme(legend.position = c(0.8, 0.7)) +
    labs(title = "Simulated standard normal values histogram",
         subtitle = "Method: Rejection sampling") # title
```

```
figure # show figure
```

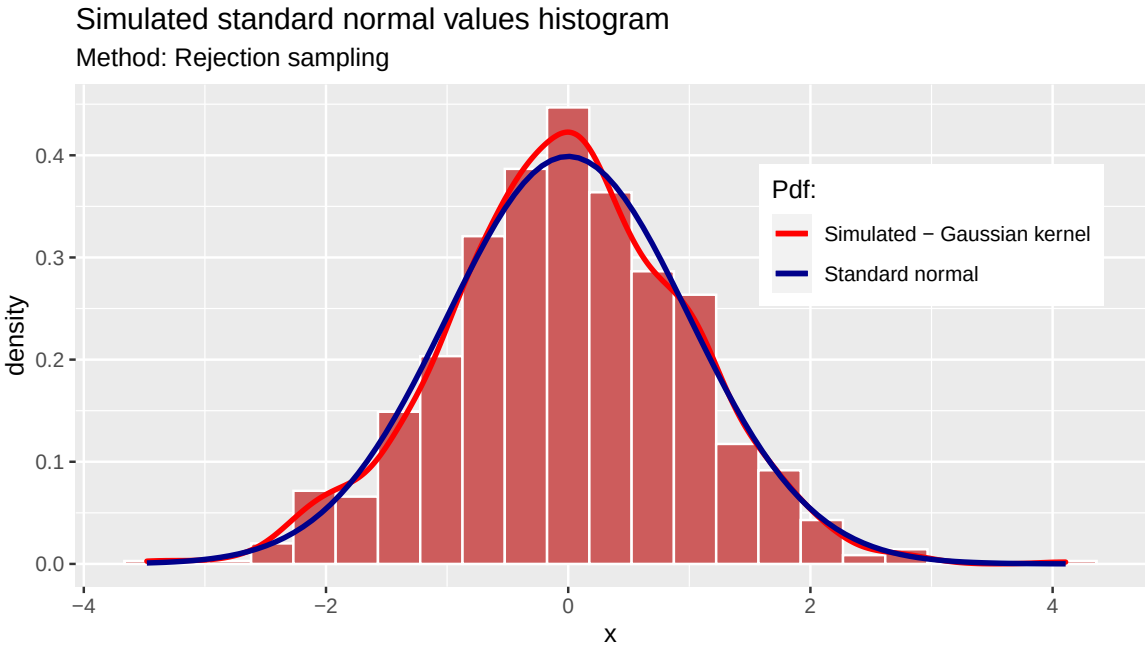

Figure 3: (1a) - Rejection sampling simulated histogram and pdfs

The simulated pdf curve has been generated using a Gaussian kernel and is compared with the standard normal pdf curve. It seems from the previous plot that the simulated distribution is very close to the one it approximates. By increasing the number n (here $n = 1000$) of the simulated values, we could further improve the simulated pdf.
Next, we calculate the acceptance probability for the rejection sampling algorithm, both theoretically as well as experimentally with R. The exact value for that probability will be

$$\mathbb{P}[\text{accept the value } X = y] = \int_{-\infty}^{\infty} \frac{f(y)}{M \cdot g(y)} g(y) \mathrm{d}y = \frac{1}{M},$$

since from every $y \sim g$ we only accept those with probability $\frac{f(y)}{M \cdot g(y)}$. Using the optimal M value $M_{opt} \approx 1.52$, it follows that

$$\mathbb{P}[\text{accept the value } X = y] = \frac{1}{M_{opt}} \approx 0.6577 \approx 65.77\%.$$

We can now experimentally calculate the acceptance probability value by dividing the number of accepted sample points by the total number of proposed sample points. Below, we can see the R code and the results generated.

```r
# --- PRINTING RESULTS ---
# Results of simulation
deviation = round(100*abs(round(100*num_accepted_samples/num_total_samples,2)-
round(100*1/M, 2))/round(100*1/M, 2), 2)
accpt_prob = round(100*num_accepted_samples/num_total_samples,2)

cat(sprintf("SIMULATION RESULTS
Total number of samples generated: %d
Total number of samples accepted: %d
Simulation acceptance probability (SAP): %.2f %%
Theoretical acceptance probability (TAP): %.2f %%
Deviation of SAP from TAP: %.2f %%
Estimation for mean from accepted sample points: %.4f
Estimation for standard deviation from accepted sample points: %.4f",
num_total_samples,
num_accepted_samples,
accpt_prob,
round(100*1/M, 2),
deviation,
mean(accepted_samples),
sd(accepted_samples)))

## SIMULATION RESULTS
## Total number of samples generated: 1516
## Total number of samples accepted: 1000
## Simulation acceptance probability (SAP): 65.96 %
## Theoretical acceptance probability (TAP): 65.77 %
## Deviation of SAP from TAP: 0.29 %
## Estimation for mean from accepted sample points: -0.0127
## Estimation for standard deviation from accepted sample points: 0.9901
```

Out of 1516 generated sample points, 1000 were accepted, leading to an acceptance probability for the simulation of

$$\frac{\#\ accepted\ sample\ points}{\#\ total\ examined\ points} = \frac{1000}{1516} = 65.96\%,$$

a value which is very close to the theoretical one, deviating only 0.29%.
Lastly, for the standard normal distribution $N(0,1)$ we have for its mean and standard deviation

$$\mu = 0,\ \sigma = 1.$$

We can see in the previous code block's output that the simulated sample's distribution parameters are a good approximation of the theoretical ones since we found

$$\overline{x} = -0.0127,\ s = 0.9901.$$

b - Rao-Blackwellized estimator

In this assignment, a known unbiased standard estimator of a distribution's parameter is assumed and then we are looking to find another unbiased estimator that has lower variance. The Rao-Blackwell theorem states that any unbiased estimator can be uniformly improved by conditioning on a sufficient statistic.
The standard estimator of the expected value of a random variable is the sample mean. We are going to find the Rao-Blackwellized version of that estimator using a mixture of distributions and we will demonstrate that the Rao-Blackwellized estimator has

lower variance than the standard estimator and the same expected value (both are unbiased). We will then verify those findings by simulating the estimators in R.
For the task at hand, let X be a random variable, where

$$X \sim NB(r, p),$$

and it can be derived as a mixture of distributions

$$\Lambda \sim \text{Gamma}\left(r, \frac{1-p}{p}\right) \text{ and } X|\Lambda = \lambda \sim \text{Poisson}(\lambda).$$

Initially, since X follows a negative binomial distribution with parameters r and p, we get

$$\mathbb{E}[X] = \frac{r \cdot (1-p)}{p} \text{ and } \text{Var}[X] = \frac{r \cdot (1-p)}{p^2},$$

where X is the number of failures before the r-th success is observed in a series of Bernoulli experiments with probability of success p. In order to estimate $\theta = \mathbb{E}[x]$, the standard estimator $\hat{\theta}$ that estimates the parameter θ is the sample mean

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

For the estimator's mean value and variance we get

$$\begin{aligned}\mathbb{E}[\hat{\theta}] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] &\overset{\mathbb{E}\ linearity}{=} \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[X_i\right] \overset{i.i.d.}{=} \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[x] \\ = \frac{1}{n}\sum_{i=1}^{n}\frac{r\cdot(1-p)}{p} = \frac{1}{n}\cdot n \cdot \frac{r(1-p)}{p} &= \frac{r(1-p)}{p}\end{aligned}$$

$$\begin{aligned}\text{Var}[\hat{\theta}] = \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n^2}\text{Var}\left[\sum_{i=1}^{n} X_i\right] &= \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}\left[X_i\right] \\ = \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}[X] = \frac{1}{n^2}\sum_{i=1}^{n}\frac{r\cdot(1-p)}{p^2} = \frac{1}{n^2}n\cdot\frac{r(1-p)}{p^2} &= \frac{r\cdot(1-p)}{n\cdot p^2}\end{aligned}$$

For $\Lambda \sim \text{Gamma}\left(r, \frac{1-p}{p}\right)$ we have

$$\mathbb{E}[\Lambda] = r\frac{1-p}{p} \quad \text{and} \quad \text{Var}[\Lambda] = r\frac{(1-p)^2}{p^2}.$$

For $X|\Lambda \sim \text{Poisson}(\lambda)$, we have

$$\mathbb{E}_{X|\Lambda}[X \mid \Lambda] = \lambda \quad \text{and} \quad \text{Var}_{X|\Lambda}[X \mid \Lambda] = \lambda.$$

Given that we are looking to estimate $\theta = \mathbb{E}[X]$, we can use $\theta^* = \mathbb{E}_{X|\Lambda}[\hat{\theta} \mid \mathbf{\Lambda}]$, where $\mathbf{\Lambda}$ is defined below, instead of the estimator $\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} X_i$ in order to find an estimator with the same bias but lower variance, i.e. $Bias(\theta^*) = Bias(\hat{\theta})$ and $\text{Var}[\theta^*] \leq \text{Var}[\hat{\theta}]$. Now, let

$$X \sim f_x(x), \text{ and } \boldsymbol{X} = (X_1, X_2, \ldots, X_n) \text{ a random sample },$$

$$\Lambda \sim f_\Lambda(\lambda), \text{ and } \mathbf{\Lambda} = (\Lambda_1, \Lambda_2, \ldots, \Lambda_n) \text{ a random sample .}$$

If θ^* is an estimator of $\theta = \mathbb{E}[X]$ and $\boldsymbol{X}$ can be simulated by the joint distribution $f_{X,\Lambda}(x, \lambda)$ satisfying

$$\int_\lambda f_{X,\Lambda}(x, \lambda) d\lambda = f_X(x),$$

then $\theta^* = \mathbb{E}_{X|\Lambda}[\hat{\theta} \mid \mathbf{\Lambda}]$ is the Rao-Blackwellized estimator. Next, we define the pdfs

$$f_{X|\Lambda}(x \mid \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}, \text{ since } X|\Lambda \sim \text{Poisson}(\lambda), \quad \lambda > 0,$$

$$f_\wedge(\lambda) = \frac{p^r}{\Gamma(r) \cdot (1-p)^r} \cdot \lambda^{r-1} e^{-\frac{p\lambda}{1-p}}, \quad r > 0, \ \frac{1-p}{p} > 0 \Rightarrow 0 < p < 1, \text{ since}$$

$$\Lambda \sim \text{Gamma}\left(r, \frac{1-p}{p}\right).$$

We also note here that if $Y \sim \text{Gamma}(k, \theta)$ then its pdf is

$$f_Y(y) = f(y; k, \theta) = \frac{y^{k-1} e^{-\frac{y}{\theta}}}{\theta^k \Gamma(k)}, \quad y > 0 \text{ and } k, \theta > 0.$$

Also, for $X \sim NB(r, p)$ we get for its pdf:

$$f_X(x) = f(x; r, p) = \begin{pmatrix} x+r-1 \\ r-1 \end{pmatrix} \cdot (1-p)^x \cdot p^r.$$

However,

$$\begin{pmatrix} x+r-1 \\ r-1 \end{pmatrix} = \frac{(x+r-1) \cdot (x+r-2) \cdots (r)}{x!} = \frac{\Gamma(x+r)}{x! \Gamma(r)},$$

therefore the previous pdf for the negative binomial distribution takes the following form

$$f_X(x) = f(x; r, p) = \frac{\Gamma(x+r)}{x! \Gamma(r)} \cdot (1-p)^x \cdot p^r.$$

Now, we calculate the joint distribution

$$f_{X,\Lambda}(x, \lambda) = f_{X|\Lambda}(x \mid \lambda) \cdot f_\Lambda(\lambda) = \frac{\lambda^x e^{-\lambda}}{x!} \cdot \frac{p^r}{\Gamma(r)(1-p)^r} \cdot \lambda^{r-1} \cdot e^{\frac{-p\lambda}{1-p}}, \lambda > 0.$$

Therefore

$$\begin{aligned} \textstyle\int_\lambda f_{X,\Lambda}(x, \lambda) d\lambda &= \int_0^{+\infty} \frac{\lambda^x e^{-\lambda}}{x!} \cdot \frac{p^r}{\Gamma(r)(1-p)^r} \cdot \lambda^{r-1} \cdot e^{-\frac{p\lambda}{1-p}} d\lambda \\ &= \frac{p^r}{x! \Gamma(r)(1-p)^r} \cdot \underbrace{\int_0^{+\infty} \lambda^x e^{-\lambda} \lambda^{r-1} \cdot e^{-\frac{p\lambda}{1-p}} d\lambda}_{I}. \end{aligned}$$

We can calculate the integral I now

$$I = \int_0^{+\infty} \lambda^{x+r-1} \cdot e^{-\lambda - \frac{p\lambda}{1-p}} d\lambda = \int_0^{+\infty} \lambda^{x+r-1} \cdot e^{-\frac{\lambda}{1-p}} d\lambda.$$

If we set $k = x + r$ and $\theta = 1 - p$, then

$$I = \int_0^{+\infty} \lambda^{k-1} \cdot e^{-\frac{\lambda}{\theta}} d\lambda,$$

and the expression in the integral has a form similar to Gamma distribution. We can rewrite I as

$$I = \theta^k \cdot \Gamma(k) \int_0^{+\infty} \frac{\lambda^{k-1} e^{-\frac{\lambda}{\theta}}}{\theta^k \Gamma(k)} d\lambda = \theta^k \cdot \Gamma(k) \cdot 1 = \theta^k \Gamma(k),$$

where the previous integral was equal to 1, since we integrated a Gamma pdf. Then,

$$I = \theta^k \cdot \Gamma(k) = (1-p)^{x+r} \Gamma(x+r),$$

and therefore

$$\begin{aligned}
\int_\lambda f_{X,\Lambda}(x,\lambda) d\lambda &= \frac{p^r}{x!\Gamma(r)(1-p)^r} \cdot I \\
&= \frac{p^r}{x!\Gamma(r)(1-p)^r} (1-p)^{x+r} \cdot \Gamma(x+r) \\
&= p^r (1-p)^x \cdot \frac{\Gamma(x+r)}{x! \cdot \Gamma(r)}
\end{aligned}$$

We showed before that for $X \sim NB(r,p)$,

$$f_X(x) = \frac{\Gamma(x+r)}{x!\Gamma(r)} \cdot (1-p)^x \cdot p^r$$

and thus we have proved that

$$\int_\lambda f_{X,\Lambda}(x,\lambda) d\lambda = f_X(x).$$

Then, we can state that $\theta^* = \mathbb{E}_{X|\Lambda}[\hat{\theta} \mid \mathbf{\Lambda}]$ is the Rao-Blackwellized estimator and

$$\begin{aligned}
\theta^* = \mathbb{E}_{X|\Lambda}[\hat{\theta} \mid \mathbf{\Lambda}] &= \mathbb{E}_{X|\Lambda} \left[\frac{1}{n} \sum_{i=1}^n X_i \mid \mathbf{\Lambda} \right] \\
&= \frac{1}{n} \mathbb{E}_{X|\Lambda} \left[\sum_{i=1}^n X_i \mid \mathbf{\Lambda} \right] \\
&= \frac{1}{n} \sum_{i=1}^n \mathbb{E}_{X|\Lambda} \left[X_i \mid \Lambda_i \right] = \\
&= \frac{1}{n} \cdot \sum_{i=1}^n \Lambda_i.
\end{aligned}$$

For its mean and variance we get

$$\mathbb{E}\left[\theta^*\right] = \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^n \Lambda_i \right] = \frac{1}{n} \mathbb{E}\left[\sum_{i=1}^n \Lambda_i \right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E}\left[\Lambda_i\right] =$$

$$= \frac{1}{n} \sum_{i=1}^n \mathbb{E}[\Lambda] = \frac{1}{\cancel{n}} \cdot \cancel{n} \cdot r \cdot \frac{1-p}{p} = r \cdot \frac{1-p}{p},$$

which is equal to $\mathbb{E}[\hat{\theta}] = \mathbb{E}[X]$, i.e. $\hat{\theta},\ \theta^*$ are both unbiased estimators of $\theta = \mathbb{E}[X]$.

$$\text{Var}\left[\theta^*\right] = \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\Lambda_i\right] = \frac{1}{n^2}\,\text{Var}\left[\sum_{i=1}^{n}\Lambda_i\right] =$$
$$= \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}\left[\Lambda_i\right] = \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}[\Lambda] =$$
$$= \frac{1}{n^{\not 2}}\cdot \not{n}\cdot r\cdot\frac{(1-p)^2}{p^2} = \frac{r(1-p)^2}{n\cdot p^2}.$$

For the ratio of the Rao-Blackwellized estimator's variance over the standard estimator's variance, we have

$$\frac{Var\left[\theta^*\right]}{Var[\hat{\theta}]} = \frac{\frac{r(1-p)^2}{n\cdot p^2}}{\frac{r(1-p)}{n\cdot p^2}} \Rightarrow$$

$$Var\left[\theta^*\right] = (1-p)Var[\hat{\theta}], \qquad \text{where } 0 < p < 1.$$

Therefore,

$$Var\left[\theta^*\right] < Var[\hat{\theta}].$$

Using $r = 1$ and $p = 0.5$ for the assignment at hand, we get

$$\mathbb{E}[X] = \frac{r\cdot(1-p)}{p} = 1 = \mathbb{E}[\hat{\theta}] = \mathbb{E}\left[\theta^*\right],$$

$$\text{Var}[X] = \frac{r(1-p)}{\rho^2} = 2 \quad , \ \text{Var}[\hat{\theta}] = \frac{r\cdot(1-p)}{n\cdot p^2} = \frac{1}{n} \quad \text{and} \ \ \text{Var}\left[\theta^*\right] = \frac{r(1-p)^2}{n\cdot p^2} = \frac{1}{2n},$$

where n is the sample size. Therefore, as we increase the sample size n, $\hat{\theta} \to \mathbb{E}[X] = 1$ and $\theta^* \to \mathbb{E}[X] = 1$, while the variance $\text{Var}\left[\theta^*\right]$ will be half the variance $\text{Var}[\hat{\theta}]$ for any n. For $n \to +\infty$, though, both $\text{Var}[\hat{\theta}], \text{Var}\left[\theta^*\right] \to 0$, with the variance of θ^* dropping at a faster rate (twice the rate of $\hat{\theta}$'s variance).
Next, we verify the previous theoretical results utilizing R code to perform the required simulations. First, we set a fixed sample size equal to $n = 5000$, and we take m such samples from the Negative Binomial distribution. The samples will be

$$\begin{aligned}
&\boldsymbol{X_1} = (X_{11}, X_{12}, \ldots X_{1n}),\\
&\boldsymbol{X_2} = (X_{21}, X_{22}, \ldots X_{2n}),\\
&\vdots\\
&\boldsymbol{X_m} = (X_{m1}, X_{m2}, \ldots X_{mn}).
\end{aligned}$$

We can calculate the standard estimator for each sample by taking the sample mean, i.e. $\hat{\theta}_j = \frac{1}{n}\sum_{i=1}^{n} X_{ji}$, $j = 1, 2, \ldots m$. This way, we end up with m different simulated values for the standard estimator $\hat{\theta}_1, \ldots \hat{\theta}_m$. To then estimate the standard estimator's mean value and variance through the simulation, we calculate with R code

$$\text{Approximation of } \mathbb{E}[\hat{\theta}]: \qquad \bar{\hat{\theta}} = \frac{1}{m}\sum_{j=1}^{m}\hat{\theta}_j$$

$$\text{Approximation of } \text{Var}[\hat{\theta}]: \quad s^2_{\hat{\theta}} = \frac{1}{m-1}\sum_{j=1}^{m}\left(\hat{\theta}_j - \bar{\hat{\theta}}\right)^2.$$

Similarly for the Rao-Blackwellized estimator, we simulate m samples of size $n = 5000$ from the Gamma distribution. The samples will be

$$\begin{aligned}
\boldsymbol{\Lambda_1} &= (\Lambda_{11}, \Lambda_{12}, \dots \Lambda_{1n}), \\
\boldsymbol{\Lambda_2} &= (\Lambda_{21}, \Lambda_{22}, \dots \Lambda_{2n}), \\
&\vdots \\
\boldsymbol{\Lambda_m} &= (\Lambda_{m1}, \Lambda_{m2}, \dots \Lambda_{mn}).
\end{aligned}$$

In a similar fashion as with the standard estimator, we can calculate the Rao-Blackwellized estimator for each sample by taking the sample mean, i.e. $\theta_j^* = \frac{1}{n}\sum_{i=1}^{n} \Lambda_{ji}$, $j = 1, 2, \dots m$. This way, we end up with m different simulated values for the Rao-Blackwellized estimator $\theta_1^*, \dots \theta_m^*$. To then estimate the Rao-Blackwellized estimator's mean value and variance through the simulation, we calculate with R code

$$\text{Approximation of } \mathbb{E}[\theta^*]: \quad \bar{\theta}^* = \frac{1}{m}\sum_{j=1}^{m} \theta_j^*$$

$$\text{Approximation of } \mathrm{Var}[\theta^*]: \quad s_{\theta^*}^2 = \frac{1}{m-1}\sum_{j=1}^{m} \left(\theta_j^* - \bar{\theta}^*\right)^2.$$

In the following code cell we repeat the described simulation process for a number of different m samples of size $n = 5000$. The selected range is a logarithmic one selecting about 40 different m values in the range $[2, 5000]$.

```
# 1.b - Rao-Blackwellization
library(ggplot2)

set.seed(42) # generate reproducible results across runs


log_seq <- function(start_point, end_point, num_points, log_base = 10) {
  sequence <- seq(log(start_point, log_base),
                  log(end_point, log_base), length.out=num_points)
  return(log_base ** sequence)
}

n = 5000 # number of sample points for sample X = (X1, X2, ..., Xn)
r = 1 # number of successes to observe occurring
p = 0.5 # probability of success in a trial
std_est_sample = c()
rao_est_sample = c()

std_mean = c()
rao_mean = c()

std_var = c()
rao_var = c()

std_var_theor = c()
rao_var_theor = c()

m = unique(floor(log_seq(2, 5000, num=40))) # selected a logarithmic range

for (i in m){

  # take i samples of size n from NB(r,p)
  x_nb = replicate(i, rnbinom(n=n, size=r, prob=p)) # m samples of size n each
```

```
  # take i samples of size n from Gamma(r,(1-p)/p)
  lambda_i = replicate(i, rgamma(n=n, shape=r, scale = (1-p)/p))

  # calculate means
  std_est = colMeans(x_nb) # mean of columns 1 x m, this is the std estimator
  std_cur_mean = mean(std_est) # mean of standard estimator's simulated values
  std_mean = append(std_mean, std_cur_mean) # add to vector

  rao_est = colMeans(lambda_i) # mean of columns 1 x m, this is rao estimator
  rao_cur_mean = mean(rao_est) # mean of rao estimator's simulated values
  rao_mean = append(rao_mean, rao_cur_mean) # add to vector

  # calculate variances
  std_cur_var = var(std_est) # std estimator variance
  std_var = append(std_var, std_cur_var)

  rao_cur_var = var(rao_est) # rao estimator variance
  rao_var = append(rao_var, rao_cur_var)
}
```

We can now plot for the different m values, the simulated mean values of the two estimators under evaluation.

```
# create a data frame with all the values
df = data.frame(sims=m, # num of simulations ie sample size
                std_means=std_mean,
                rao_means=rao_mean,
                std_vars=std_var,
                rao_vars=rao_var,
                real_mean = 1,
                std_vars_theor = 2/n,
                rao_vars_theor = 1/n)


colors <- c("Rao estimator mean" = "darkblue",
            "Standard estimator mean" = "indianred",
            "Real mean value" = "royalblue1")


figure1 <- ggplot(data = df, aes(x=sims)) +
  geom_line(aes(y=std_means, color='Standard estimator mean'), size=0.9) +
  geom_line(aes(y=rao_means, color='Rao estimator mean'), size=0.9) +
  geom_line(aes(y=real_mean, color='Real mean value'),linetype="dashed", size=0.9)+
  labs(
    title = "Mean of Standard and Rao-Blackwellized estimators",
    subtitle = "Simulation for variable number of samples of size n=5000 each",
    x = 'Number of samples m',
    y = 'Mean',
    color = 'Legend') +
    scale_color_manual(values = colors) +
    theme(legend.position = c(0.79, 0.20),
          legend.key.size = unit(0.5, 'cm'))

figure1 # show figure
```

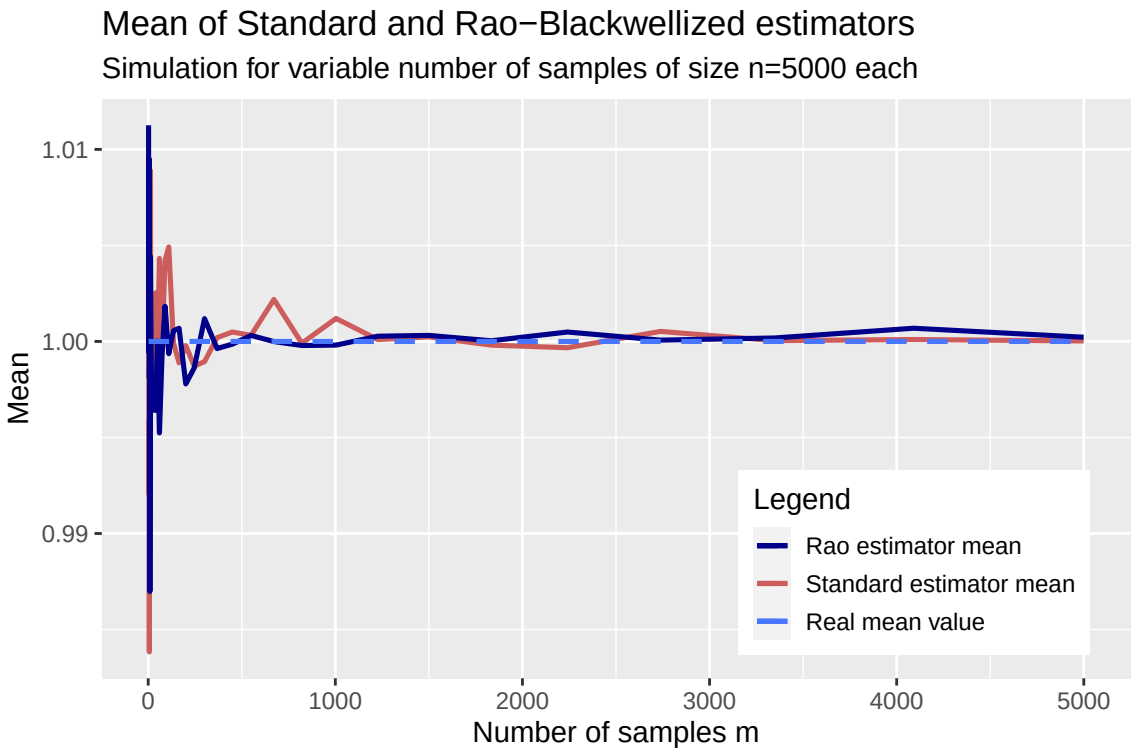

Figure 4: (1b) - Simulated estimators' mean - fixed sample size, variable samples number

We can see in the previous plot that both estimators converge to the theoretical mean value of 1. However, the Rao-Blackwellized estimator has lower variance (half of the standard estimator's), which is reflected in the plot, as the estimator generally presents smaller fluctuations/deviations from the horizontal line of the real value it approximates.
Next, we present a plot of the variance for the two estimators for different number of samples m of size n each.

```
colors <- c("Rao estimator variance" = "darkblue",
            "Standard estimator variance" = "indianred",
            "Real rao estimator variance" = "deepskyblue4",
            "Real standard estimator variance" = "magenta")

figure2 <- ggplot(data = df, aes(x=sims)) +
  geom_line(aes(y=std_vars, color='Standard estimator variance'), size=0.9) +
  geom_line(aes(y=rao_vars, color='Rao estimator variance'), size=0.9) +
  geom_line(aes(y=rao_vars_theor, color='Real rao estimator variance'),
  linetype="dashed", size=0.9) +
  geom_line(aes(y=std_vars_theor, color='Real standard estimator variance'),
  linetype="dashed", size=0.9) +
  labs(
    title = "Variance of Standard and Rao-Blackwellized estimators",
    subtitle = "Simulation for variable number of samples of size n=5000 each",
    x = 'Number of samples m',
    y = 'Variance',
    color = 'Legend') +
  #scale_y_continuous(trans='log10') +
  scale_color_manual(values = colors) +
  theme(legend.position = c(0.73, 0.81),
        legend.key.size = unit(0.35, 'cm'))
```

```
figure2 # show figure
```

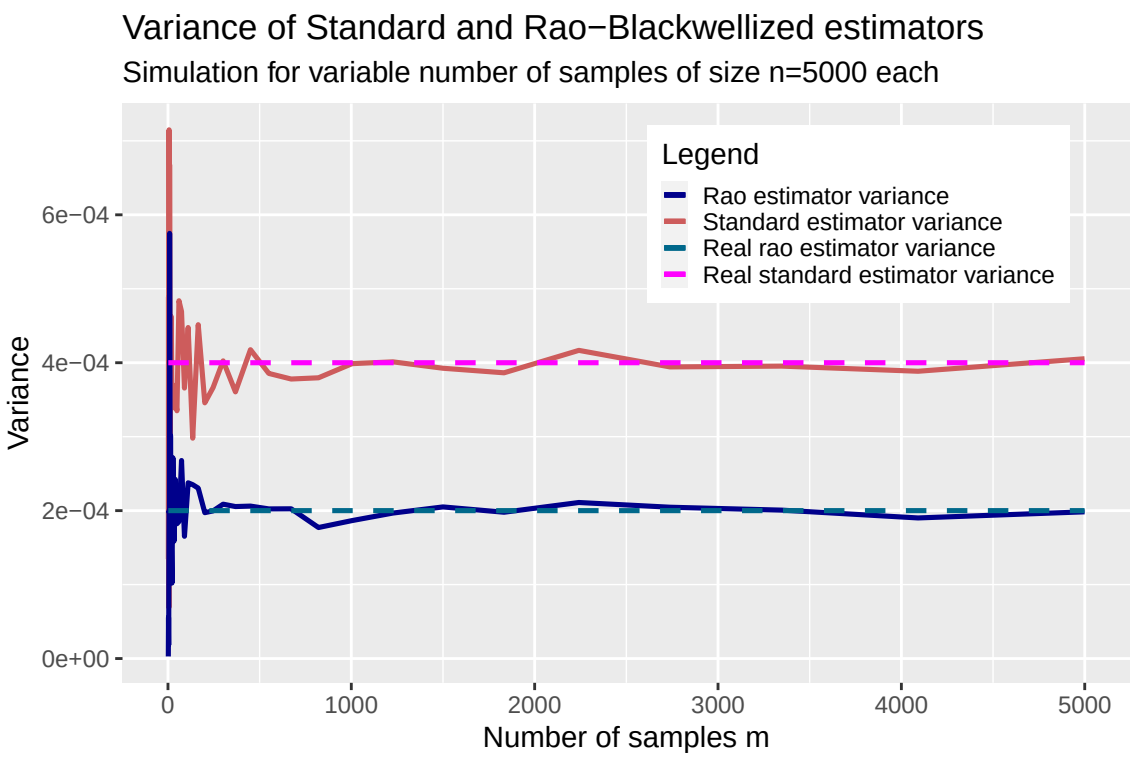

Figure 5: (1b) - Simulated estimators' variance - fixed sample size, variable samples number

In the previous plot, we can see that both estimators converge to their theoretical variance values. The Rao-Blackwellized estimator's variance have a similar form as the one for the standard estimator, however, for any m value on the horizontal axis, the variance of the Rao-Blackwellized one is approximately half the variance of the standard.
In addition to the previous simulation, we also perform another one, this time having a fixed number of samples k and a variable sample size n. We select $k = 300$ samples and select a range of around 150 distinct n values for each sample's size in the range $[2, 10000]$ using a logarithmic scale. The samples that are taken in this experiment will be for each n value

$$\begin{aligned}
\boldsymbol{X_1} &= (X_{11}, X_{12}, \dots X_{1n}),\\
\boldsymbol{X_2} &= (X_{21}, X_{22}, \dots X_{2n}),\\
&\vdots\\
\boldsymbol{X_k} &= (X_{k1}, X_{k2}, \dots X_{kn}),
\end{aligned}$$

indent and

$$\begin{aligned}
\boldsymbol{\Lambda_1} &= (\Lambda_{11}, \Lambda_{12}, \dots \Lambda_{1n}),\\
\boldsymbol{\Lambda_2} &= (\Lambda_{21}, \Lambda_{22}, \dots \Lambda_{2n}),\\
&\vdots\\
\boldsymbol{\Lambda_k} &= (\Lambda_{k1}, \Lambda_{k2}, \dots \Lambda_{kn}),
\end{aligned}$$

and we calculate the mean and variance of the estimators in the same way as in the previous simulation.

```
# NOW SIMULATE FOR VARIABLE SAMPLE SIZE n AND FIXED NUMBER OF SIMULATIONS


k = 300 # number of simulated samples
std_est_sample2 = c()
rao_est_sample2 = c()

std_mean2 = c()
rao_mean2 = c()

std_var2 = c()
rao_var2 = c()

std_var_theor2 = c()
rao_var_theor2 = c()
sample_size = c()


sequence =  unique(floor(log_seq(2, 10000, num=150))) # selected a logarithmic range

for (i in sequence){
  # take k samples of size i from NB(r,p)
  x_nb2 = replicate(k, rnbinom(n=i, size=r, prob=p)) #sample of size i

  # take k samples of size i of L_i points from Gamma(r,(1-p)/p)
  lambda_i2 = replicate(k, rgamma(n=i, shape=r, scale = (1-p)/p))

  # calculate means
  std_est2 = colMeans(x_nb2) # mean of columns 1 x m, this is the std estimator
  std_cur_mean2 = mean(std_est2) # mean of all the estimator's values
  std_mean2 = append(std_mean2, std_cur_mean2)

  rao_est2 = colMeans(lambda_i2) # mean of columns 1 x m, this is rao estimator
  rao_cur_mean2 = mean(rao_est2)
  rao_mean2 = append(rao_mean2, rao_cur_mean2)

  # calculate variances
  std_cur_var2 = var(std_est2)
  std_var2 = append(std_var2, std_cur_var2)

  rao_cur_var2 = var(rao_est2)
  rao_var2 = append(rao_var2, rao_cur_var2)

}

for (i in sequence){
  std_var_theor2 = append(std_var_theor2, 2/i)
  rao_var_theor2 = append(rao_var_theor2, 1/i)
  sample_size = append(sample_size, i)
}
```

Then, we generate the plot for the mean values of the estimators as well as the real mean value.

```
# create a data frame with all the values
df2 = data.frame(sims=sample_size, # num of simulations ie sample size
                 std_means=std_mean2,
                 rao_means=rao_mean2,
                 std_vars=std_var2,
                 rao_vars=rao_var2,
```

```
                real_mean = 1,
                std_vars_theor = std_var_theor2,
                rao_vars_theor = rao_var_theor2)


colors <- c("Rao estimator mean" = "darkblue",
            "Standard estimator mean" = "indianred",
            "Real mean value" = "royalblue1")

figure3 <- ggplot(data = df2, aes(x=sims)) +
  geom_line(aes(y=std_means, color='Standard estimator mean'), size=0.9) +
  geom_line(aes(y=rao_means, color='Rao estimator mean'), size=0.9) +
  geom_line(aes(y=real_mean, color='Real mean value'),linetype="dashed", size=0.9)+
  labs(
    title = "Mean of Standard and Rao-Blackwellized estimators",
    subtitle = "Simulation for k=300 samples of variable size",
    x = 'Sample size n',
    y = 'Mean value',
    color = 'Legend') +
  scale_color_manual(values = colors,
                     guide=guide_legend(override.aes=list(linetype=c(1,1,2),
                                                          lwd=c(1,1,0.5)))) +
  theme(legend.position = c(0.8, 0.83),
        legend.key.size = unit(0.5, 'cm'))

figure3 # show figure
```

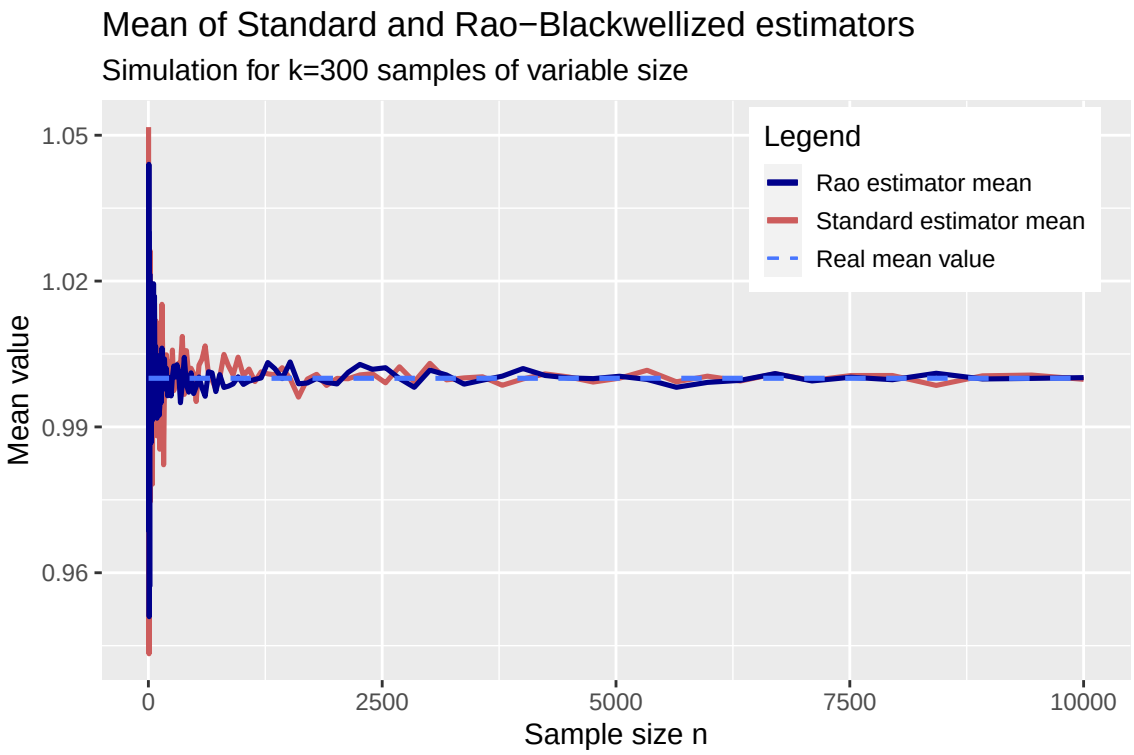

Figure 6: (1b) - Simulated estimators' mean - variable sample size, fixed samples number

Similarly to the previous simulation, the mean values of the Rao-Blackwellized simulator present lower variance and both estimators mean values converge to the real mean value. Next, we create a plot for the variance of the estimators.

```
colors <- c("Rao estimator variance" = "darkblue",
            "Standard estimator variance" = "indianred",
            "Real rao estimator variance" = "deepskyblue4",
            "Real standard estimator variance" = "magenta")

figure4 <- ggplot(data = df2, aes(x=sims)) +
  geom_line(aes(y=std_vars, color='Standard estimator variance'), size=0.9) +
  geom_line(aes(y=rao_vars, color='Rao estimator variance'), size=0.9) +
  geom_line(aes(y=rao_vars_theor, color='Real rao estimator variance'),
            linetype="dashed", size=0.9) +
  geom_line(aes(y=std_vars_theor,
            color='Real standard estimator variance'),
            linetype="dashed", size=0.9) +
  labs(
    title = "Variance of Standard and Rao-Blackwellized estimators ",
    subtitle = "Simulation for k=300 samples of variable size",
    x = 'Sample size n',
    y = 'Variance value',
    color = 'Legend') +
  scale_y_continuous(trans='log10') +
  scale_color_manual(values = colors,
                     guide=guide_legend(override.aes=list(linetype=c(1,1,2,2), lwd=c(1,1,0.5,0.5)
  theme(legend.position = c(0.72, 0.78),
        legend.key.size = unit(0.5, 'cm'))

figure4 # show figure
```

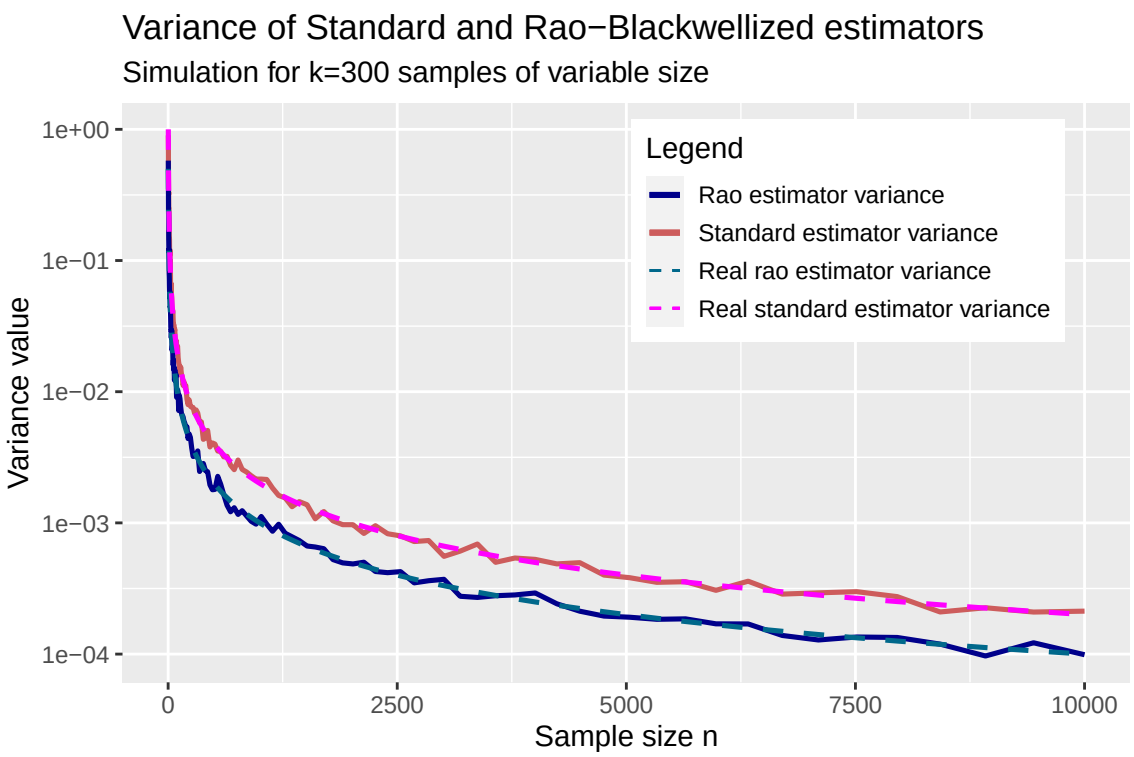

Figure 7: (1b) - Simulated estimators' variance - variable sample size, fixed samples number

In this case, since the sample size n is variable, the estimators' variance will also be variable depending on the n value. The real variance of the Rao-Blackwellized estimator is $\frac{1}{n}$ and the standard one's is twice the variance of the Rao-Blackwellized

estimatior, i.e. $\frac{2}{n}$. From the plot, we can see that for a fixed number of samples $k = 300$, both estimators' variances converge to the real values.

c - Statistical function simulation

Let $X_1, X_2, \ldots, X_n$ be a random sample and T be a statistical function

$$T = \frac{(X_1 + X_2 + \cdots + X_n)^2}{n}.$$

If $X_i \sim U(0,1),\ i = 1, 2, \ldots, n$, we are looking to simulate, using R code, 10000 values from T ($n = 80$) and construct the histogram of the simulated values, as well as calculate the mean and standard deviation of the simulated values and finally compare them with the theoretical mean and standard deviation of the random variable T.
Firstly, we can rewrite the $(X_1 + X_2 + \cdots + X_n)^2$ expression in a sum format

$$(X_1 + X_2 + \cdots + X_n)^2 = \sum_{i=1}^{n} X_i^2 + 2 \cdot \sum_{j=2}^{n} \left(X_j \cdot \sum_{i=1}^{j-1} X_i \right).$$

For its mean value it then follows

$$\begin{aligned}
\mathbb{E}\left[(X_1 + X_2 + \cdots + X_n)^2\right] &= \mathbb{E}\left[\sum_{i=1}^{n} X_i^2 + 2 \cdot \sum_{j=2}^{n} \left(X_j \cdot \sum_{i=1}^{j-1} X_i \right)\right] = \\
&= \sum_{i=1}^{n} \mathbb{E}\left[X_i^2\right] + 2 \sum_{j=2}^{n} \mathbb{E}\left[X_j \sum_{i=1}^{j-1} X_i\right] = \\
&= \sum_{i=1}^{n} \mathbb{E}\left[X_i^2\right] + 2 \sum_{j=2}^{n} \left(\mathbb{E}\left[X_j\right] \cdot \sum_{i=1}^{j-1} \mathbb{E}\left[X_i\right] \right) = \\
&= \sum_{i=1}^{n} \mathbb{E}\left[X^2\right] + 2 \cdot \sum_{j=2}^{n} \left(\mathbb{E}[X] \cdot \sum_{i=1}^{j-1} \mathbb{E}[X] \right) = \\
&= n \cdot \mathbb{E}\left[X^2\right] + 2 \cdot \sum_{j=2}^{n} \left(\mathbb{E}[X] \cdot (j-1) \cdot \mathbb{E}[X] \right) = \\
&= n \cdot \mathbb{E}\left[X^2\right] + \cancel{2} \cdot \frac{n \cdot (n-1)}{\cancel{2}} \cdot (\mathbb{E}[X])^2 = \\
&= n \cdot \mathbb{E}\left[X^2\right] + n \cdot (n-1) \cdot (\mathbb{E}[X])^2,
\end{aligned}$$

where n is known ($n = 80$), $\mathbb{E}[X] = \frac{1}{2}$ since $X \sim U(0,1)$ and we also used in the previous calculations the following

$$\mathbb{E}\left[X_1\right] = \mathbb{E}\left[X_2\right] = \cdots = \mathbb{E}\left[X_n\right] = \mathbb{E}[X].$$

Given the fact that all of the random variables X_i are independent and identically distributed, we also used in the previous mean value evaluation the fact that

$$\mathbb{E}\left[X_i \cdot X_j\right] = \mathbb{E}\left[X_i\right] \cdot \mathbb{E}\left[X_j\right] = \mathbb{E}[X] \cdot \mathbb{E}[X] = \left(\mathbb{E}[X]\right)^2, \text{ for } i \neq j,$$

and

$$\mathbb{E}\left[X_i \cdot X_i\right] = \mathbb{E}\left[X_i^2\right] = \mathbb{E}\left[X^2\right] = \mathrm{Var}[X] + (\mathbb{E}[X])^2 = \frac{1}{12} + \left(\frac{1}{2}\right)^2 = \frac{1}{3},$$

since $\text{Var}[X] = \frac{1}{12}$ considering that $X \sim U(0,1)$. We can now calculate $\mathbb{E}[T]$.

$$
\begin{aligned}
\mathbb{E}[T] &= \mathbb{E}\left[\frac{(X_1 + X_2 + \cdots + X_n)^2}{n}\right] = \frac{1}{n} \cdot \mathbb{E}\left[(X_1 + X_2 + \cdots + X_n)^2\right] = \\
&= \frac{1}{\not{n}} \cdot \left(\not{n} \cdot \mathbb{E}\left[X^2\right] + \not{n} \cdot (n-1) \cdot \left(\mathbb{E}\left[X\right]\right)^2\right) = \\
&= \mathbb{E} \cdot \left[X^2\right] + (n-1) \cdot \left(\mathbb{E}\left[X\right]\right)^2 = \\
&= \frac{1}{3} + (80-1) \cdot \left(\frac{1}{2}\right)^2 = \\
&= \frac{1}{3} + \frac{79}{4} = \\
&= \frac{241}{12} \\
&\simeq 20.083.
\end{aligned}
$$

That was the analytically computed mean value of the random variable T.
We then proceed to perform the required simulation in R. First, we define the statistical function T.

```r
# 1.c - Simulated samples

set.seed(42) # generate reproducible results across runs

# --- IMPORTS ---
library(ggplot2)

# --- FUNCTIONS ---

# T = (X1+X2+...Xn)^2 / n
T_statistical_function <- function(sample) {
  sample_size = length(sample)
  summ = sum(sample)
  T = summ^2 / sample_size
  return(T)
}
```

Then we perform the simulation.

```r
# --- MAIN ---
n = 80 # sample points X1, X2, ..., Xn
num_simulations = 10000 # number of times to simulate n values and subsequently T values
T_values = c() # save the simulated T values

for (i in 1:num_simulations){
  sample = runif(n, min=0, max=1) # sample from uniform distribution n values
  T = T_statistical_function(sample) # calculate T = (X1+...+Xn)^2 / n
  T_values = append(T_values, T) # add T values to the list of T values
}

simulated_mean_value = mean(T_values)
simulated_std_value = sd(T_values)
real_mean_value = 1/3 + (n-1)/4 # analytical calculation of E(T)
```

```
cat(sprintf("Simulated mean of T: %.4f
Real mean of T: %.4f
Deviation of simulated mean from theoretical: %.2f %%
Simulated standard deviation of T: %.4f",
simulated_mean_value,
real_mean_value,
round(100*abs((simulated_mean_value-real_mean_value)/real_mean_value), 2),
simulated_std_value))

## Simulated mean of T: 20.0877
## Real mean of T: 20.0833
## Deviation of simulated mean from theoretical: 0.02 %
## Simulated standard deviation of T: 2.5905
```

We can see from the printed results above that the approximation for the mean value of T through the simulation deviates 0.02% from the analytically computed value. Also the sample standard deviation was found to be $s = 2.59$.
Finally, we also draw a histogram of the simulated T values using R and the resulting simulated pdf from the histogram.

```
# --- PLOTTING ---

# convert T values list to df to use ggplot
df <- data.frame(T = T_values)

#optimal bin width hopt=3.491*s*n^(-1/3)
opt_h <- 3.491 * simulated_std_value * num_simulations^(-1/3)

figure <- ggplot(data=df, aes(x=T)) +
  geom_histogram(
    binwidth = opt_h, # bin width = optimal
    aes(y = ..density..),
    color = "white", # histogram bins borders
    fill = "indianred" # histogram bins fill color
  ) +
  geom_density(size = 1.1,
               color = "red") + # simulated T values dist. curve
  labs(title = "Simulated T values distribution histogram",
       subtitle = 'Gaussian kernel') # title

figure # show figure
```

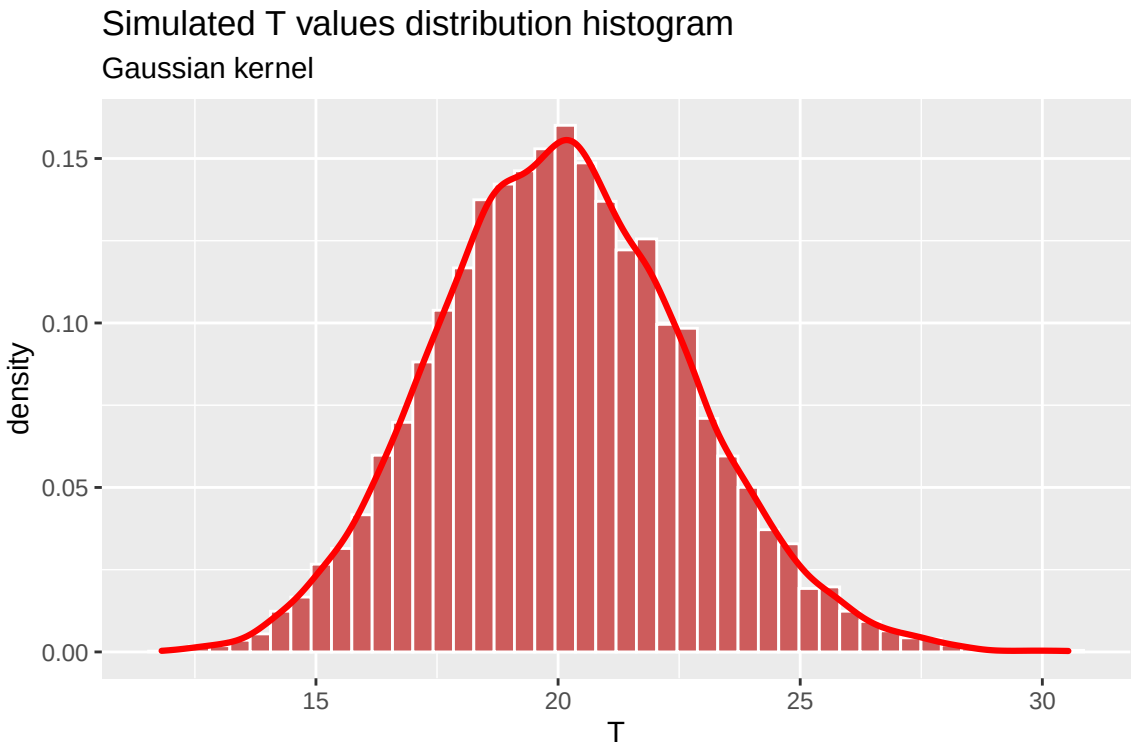

Figure 8: (1c) - Simulated T values histogram

d - Bootstrap and Jackknife resampling methods

In the present assignment, a set of 80 observations x_i, $i = 1, 2, ..., 80$ is given. Then the standard error of the previously presented statistical function $T = \frac{(X_1+X_2+\cdots+X_n)^2}{n}$ is estimated using the Bootstrap resampling method with a parameter $B = 10000$. Another estimation is performed then, using the Jackknife resampling method. Finally, by assuming the observations derived from the uniform distribution $U(0, 1)$, we compare the histogram of the simulated T values from those observations with the one presented in the previous exercise (c).
We start with the Bootstrap method first. Bootstrap is a resampling method that uses random sampling with replacement from a dataset in order to generate additional samples out of it. In the following, we describe the method.

Bootstrap:
Let $X_1, \ldots, X_n$ be a random sample from $f(x; \theta)$ and $\hat{\theta} = T(X_1, \ldots, X_n)$ an estimator. Then, in order to calculate the standard error of the statistical function T using the Bootstrap method, we follow the next two steps:

1. Create B bootstrap samples of size n. Each sample is taken by sampling with replacement from sample $X_1, \ldots, X_n$ with probability $\frac{1}{n}$ for each of its n elements.

2. For each of the B bootstrap samples $i : (X_1^*, \ldots X_n^*)$ we compute the value $\hat{\theta}_i^* = T(X_1^*, \ldots, X_n^*)$. Then obtain the following estimates for the standard error of $\hat{\theta}$

$$\text{se}(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}\left(\hat{\theta}_i^* - \overline{\hat{\theta}^*}\right)^2}, \text{ where}$$

$$\overline{\hat{\theta}^*} = \frac{1}{B}\sum_{i=1}^{B}\hat{\theta}_i^*.$$

We first define the R functions we will use, setting also the random seed for reproducibility purposes.

```r
# 1.d - Bootstrap-Jackknife

set.seed(42) # generate reproducible results across runs

# --- IMPORTS ---
library(ggplot2)

# --- FUNCTIONS ---

bootstrap_sampling <- function(original_sample){
  n = length(original_sample)
  bootstrap_sample = sample(original_sample, size=n, replace = TRUE)
  return(bootstrap_sample)
}

jackknife_sampling <- function(original_sample, exclude_index){
  jackknife_sample = original_sample[-exclude_index] # remove item with selected idx
  return(jackknife_sample)
}

# T = (X1+X2+...Xn)^2 / n
T_statistical_function <- function(sample) {
  sum = 0
  sample_size = length(sample)
  for(i in 1:sample_size){
    X_i = sample[i]
    sum = sum + X_i
  }
  T = sum^2 / sample_size
  return(T)
}
```

Then, we run the Bootstrap method to calculate the standard error.

```r
# --- MAIN ---
B = 10000 # number of bootstrap samples to generate
original_sample = readRDS("data1.rds") # read data file
n = length(original_sample) # sample length

# Bootstrap method
bootstrap_T_values = c() # save the T value for each bootstrap sample

for (i in 1:B){
  bootstrap_sample = bootstrap_sampling(original_sample) # generate bootstrap sample
  bootstrap_T_value =  T_statistical_function(bootstrap_sample) # T value
  bootstrap_T_values = append(bootstrap_T_values, bootstrap_T_value) # keep T values
}

bootstrap_se = sd(bootstrap_T_values) # bootstrap T standard error

# --- PRINTING RESULTS ---
cat(sprintf("Bootstrap T standard error: %.4f", bootstrap_se))

## Bootstrap T standard error: 2.5539
```

From the last code block, the standard error using Bootstrap is found to be

$$se_{\text{Bootstrap}}(\hat{\theta}) \approx 2.5539.$$

Next, the Jackknife resampling method is detailed for the same standard error evaluation. Jackknife is a resampling method that for a dataset of size n removes each time the $i-th$ observation $(0 \leq i \leq n)$, in order to generate a new sample. This way the total number of samples that have been generated from the initial sample are n.

Jackknife:
Similarly to Bootstrap, let $X_1, \ldots, X_n$ be a random sample from $f(x;\theta)$ and $\hat{\theta} = T(X_1, \ldots, X_n)$. We denote by $\hat{\theta}_{(i)} = T(X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$ the estimator's value calculated on the new sample $(X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$, ie. we have removed observation i from the original sample. Then, we can calculate the mean value of the n calculated $\hat{\theta}_{(i)}$ values as

$$\hat{\theta}_{(\cdot)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)}, \quad \text{where}$$

$$\hat{\theta}_{(i)} = T(X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n) = \frac{(X_1 + \ldots + X_{i-1} + X_{i+1} + \ldots + X_n)^2}{n-1}.$$

Subsequently, the Jackknife estimator $\hat{\theta}_J$ of the parameter θ is calculated as

$$\hat{\theta}_J = n\hat{\theta} - (n-1)\hat{\theta}_{(\cdot)}.$$

Intuitively, we repeat the estimation of the parameter θ n times, one for each sample created by leaving one observation out. We can also interpret the jackknife estimator as the corrected $\hat{\theta}$ using the estimators from the samples with one observation out each time. In the next code block, we implement the Jackknife method programmatically.

```
# Jackknife method
jackknife_T_values = c() # save the T value for each jackknife sample
original_sample_T_value = T_statistical_function(original_sample)

for (i in 1:n){ # remove 1 item each time for the original sample of size n
  jackknife_sample = jackknife_sampling(original_sample, i) # remove sample i
  jackknife_T_value = T_statistical_function(jackknife_sample) # calculate T
  jackknife_T_values = append(jackknife_T_values, jackknife_T_value) # save them
}

jackknife_mean_value = mean(jackknife_T_values)

# calculate jackknife estimator
jackknife_estimator = n * original_sample_T_value - (n-1) * jackknife_mean_value

# standard error calculation for jackknife estimator
summ = 0
for (i in 1:n){
  summ = summ + (jackknife_T_values[i] - jackknife_mean_value)^2
}

jackknife_estimator_variance = ((n-1)/(n)) * summ
jackknife_estimator_se = sqrt(jackknife_estimator_variance) # jackknife T se

# --- PRINTING RESULTS ---
cat(sprintf("Jackknife T standard error: %.4f", jackknife_estimator_se))

## Jackknife T standard error: 2.5470
```

The jackknife standard error was estimated to be $se_{\text{Jackknife}}(\hat{\theta}) \approx 2.5470$.
Compared to Bootstrap we can see a slight decrease in standard error of 0.27%. In general, the Jackknife method performs n repetitions while the Bootstrap method performs B repetitions with $B \geq n$ in general, which implies an increased computational cost for Bootstrap. However, Bootstrap is a method that works when Jackknife fails. The latter has also been proved to be an approximation of Bootstrap.
We now assume that the original observations have been sampled from the uniform distribution $U(0,1)$, and we will compare the histograms derived by applying the Bootstrap method for the statistical function T with the ones simulated in subsection (1c) based on the uniform assumption.

```
# --- PLOTTING ---

# 1c reimplementation

n = 80 # sample points X1, X2, ..., Xn
num_simulations = B # number of times to simulate n values and subsequently T values
T_values = c() # save the simulated T values

for (i in 1:num_simulations){
  sample = runif(n, min=0, max=1) # sample from uniform distribution n values
  T = T_statistical_function(sample) # calculate T = (X1+...+Xn)^2 / n
  T_values = append(T_values, T) # add T values to the list of T values
}
simulated_std_value = sd(T_values)

uniform_df <- data.frame(T_value = T_values)
#optimal bin width hopt=3.491*s*n^(-1/3)
uniform_opt_h <- 3.491 * simulated_std_value * B^(-1/3)


# Bootstrap

bootstrap_opt_h <- 3.491 * bootstrap_se * B^(-1/3) # hopt=3.491*s*n^(-1/3)

# convert accepted samples list to df to use ggplot
bootstrap_df <- data.frame(T_value = bootstrap_T_values)

# combine two dfs in one
# create new column denoting to which method the T value is about
uniform_df$Simulation = 'Uniform'
bootstrap_df$Simulation = 'Bootstrap'
df_combined = rbind(uniform_df, bootstrap_df)

# plot the bootstrap and uniform T values histoframs in a single figure
figure = ggplot(data=df_combined, aes(T_value, fill = Simulation)) +
  geom_histogram( # histogram bootstrap
    data = bootstrap_df,
    binwidth = bootstrap_opt_h,
    kernel = 'gaussian',
    aes(y = ..density..),
    alpha = 0.2
  ) +
  geom_histogram( # histogram uniform
    data = uniform_df,
    binwidth = bootstrap_opt_h,
    kernel = 'gaussian',
    aes(y = ..density..),
    alpha = 0.2
```

```
  ) +
  theme(legend.position = c(0.9, 0.7)) +
  labs(title = "T values histograms",
       subtitle = "Bootstrap vs Uniform simulation, Gaussian kernel")

figure # show figure
```

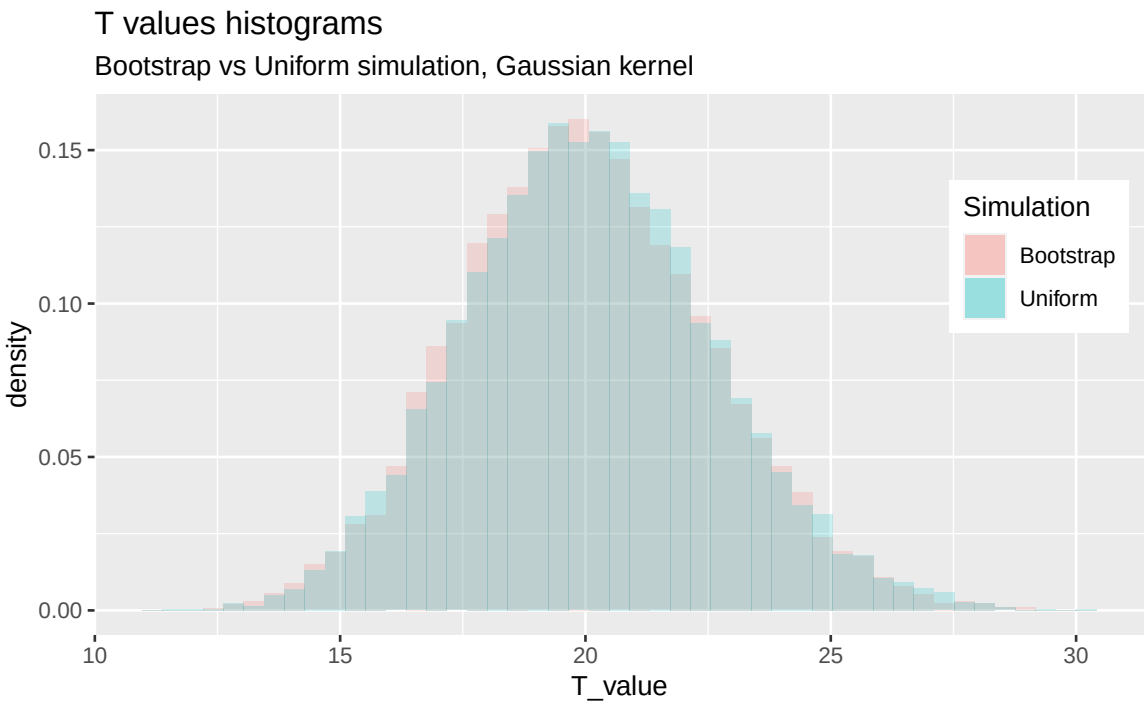

Figure 9: (1d) - T values histograms, Bootstrap vs Uniform

From the histograms plot, it follows that the Bootstrap method produces relatively similar results to the uniformly sampled values for the statistical function T. We can conclude, therefore, that the Bootstrap resampling method works very well in approximating the distribution of T from a single sample $(X_1, \ldots, X_n)$, yielding similar results to knowing the underlying distribution (uniform) of X, from which several samples $(X_1, \ldots, X_n)$ are generated to approximate T.

Exercise 2 - Density estimation

a - Optimum bandwidth selection and density plot

In the first assignment of the second exercise, a dataset consisting of 272 times in minutes of successive explosions concerning the volcano "Old Faithful geyser - Yellowston Natonal Park, Wyoming, USA" is provided. We first load the data with appropriate code in R and are looking to estimate the probability density function $f(x)$ of the data source. An Epanechnikov kernel is used for the estimation. In order to achieve this, we first calculate the best width h of the kernel maximizing the cross-validated likelihood programmatically.
The Epanechnikov kernel is defined as

$$K(u) = \begin{cases} \frac{3}{4}(1-u^2) & |u| \leq 1 \\ 0 & |u| \geq 1 \end{cases},$$

and its graph can be seen in the plot below generated with R.

```
# 2 - pdf estimation with Epanechnikov kernel

set.seed(42) # generate reproducible results across runs

# --- IMPORTS ---

library(ggplot2)

# --- FUNCTIONS ---

epanechnikov_kernel <- function(u){
  if ((u <= 1) && (u >= -1)){ #if |u|<=1
    kernel = (3/4) * (1-u^2) # Epanechnikov K(u) calculation
  } else {
    kernel = 0 # if |u|>1
  }
  return(kernel)
}


# --- MAIN ---
#2a

x_seq = seq(-2,2,0.1)
ep_kernel = c()
for (i in x_seq){
  ep_kernel =  append(ep_kernel, epanechnikov_kernel(i))
}

#plot the epanechnikov kernel
figure1 <- ggplot(data = data.frame(x=x_seq), aes(x=x_seq, y=ep_kernel)) +
  geom_line(color='indianred', aes( y=ep_kernel), size=1.1) +
  labs(title = "Graph of the epanechnikov kernel function", x="u", y = "K(u)")

figure1 # show figure
```

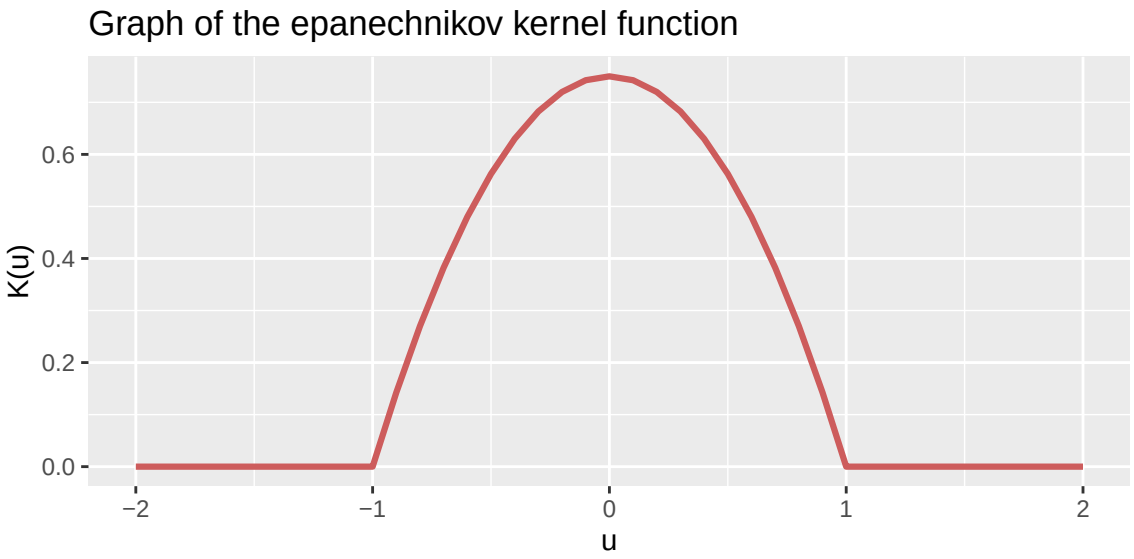

Figure 10: (2a) - Epanechnikov kernel density

The estimated probability density function $\hat{f}(x)$ of the actual pdf $f(x)$ using the Epanechnikov kernel is given by

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

where $(X_1, \dots X_n)$ is a random sample drawn from the probability density function $f(x)$ with X_i, $0 \leq i \leq n$, being n independent and identically distributed random variables. We assume that such a sample $(x_1, \dots x_n)$ is the one given.
For the given sample, we first determine the optimal h value that maximizes the cross-validated likelikood of the data. Towards that goal, we first pick a range of possible h values, in which we believe the optimal h_{opt} value lies into. Then, for every h value we can calculate the cross-validated likelihood as

$$L(h) = \prod_{j=1}^{n}\hat{f}_{h,j}\left(x_j\right),$$

where $\hat{f}_{h,j}\left(x_j\right)$ is the estimator on point x_j if we remove the j observation from the sample $(x_1, x_2, \cdots, x_n)$

$$\hat{f}_{h,j}\left(x_j\right) = \frac{1}{(n-1)\cdot h}\sum_{\substack{i=1\\ i\neq j}}^{n}K\left(\frac{x_j-x_i}{h}\right).$$

Instead of calculating the likelihood as a product of terms, we can instead calculate and maximize the log-likelihood since the logarithm is an increasing function. The calculation is easier and prevents any potential multiplication overflow, since the log-likelihood calculates a sum of terms instead.

$$LL(W) = \ln L(h) = \ln\left[\prod_{j=1}^{n}\hat{f}_{h,j}\left(x_i\right)\right] = \sum_{j=1}^{n}\left[\ln\hat{f}_{h,j}\left(x_j\right)\right].$$

Then

$$h_{opt} = \text{argmax}_h\{LL(h)\}.$$

Regarding the implementation with R, we first define some functions we are going to use.

```
# 2 - pdf estimation with Epanechnikov kernel

estimated_pdf <- function(x, sample, h){
  n = length(sample)
  summ = 0
  for (i in 1:n){ # calculate [((x-x_i)/h)]
    summ = summ + epanechnikov_kernel((x-sample[i])/h)
  }
  pdf = (1/(n * h)) * summ
  return(pdf)
}
```

Then, we load the data,

```
data = faithful$eruptions # load Old Faithful geyser data
n = length(data) # sample size
```

and print some useful properties of our data, in order to assist with selecting an appropriate range of values to test for h.

```
cat(sprintf("Max value in data: %.2f
Min value in data: %.2f
Max-min range in data: %.2f
Mean value of data: %.3f
Standard deviation of data: %.3f",
max(data),
min(data),
abs(max(data)-min(data)),
mean(data),
sd(data)))

## Max value in data: 5.10
## Min value in data: 1.60
## Max-min range in data: 3.50
## Mean value of data: 3.488
## Standard deviation of data: 1.141
```

Based on the previous code block's printed values regarding the min, max, mean and standard deviation of the data, we decided to explore a logarithmic range from 0.01 up to one standard deviation 1.141 for the possible h values considering the fact that most of the observations should be within one standard deviation apart from a certain observation. We also define the number of h values to consider in the range to be a relatively low number, in particular 30, for computational efficiency. For each of the candidate h values, we calculate the cross-validated log likelihood and the optimal h is the one that maximizes it. For demonstration purposes, we also present the graph of the log likelihood function with respect to the h value.

```
# we use the above printed results to select an appropriate range for h

log_seq <- function(start_point, end_point, num_points, log_base = 10) {
  sequence <- seq(log(start_point, log_base),
                  log(end_point, log_base), length.out=num_points)
  return(log_base ** sequence)
}
```

```r
h_seq = log_seq(0.01, sd(data), num=30) # selected a logarithmic range for h

cv_log_likelihood_seq = c()

for (h_index in 1:length(h_seq)){ # iterate over each h value in the range
  h = h_seq[h_index] # get the selected h value
  # cross-validation method, remove 1 observation each time
  cv_log_likelihood = 0 # take cross validated log likelihood
  for (i in 1:n){ # iterate over all observations
    x_i = data[i] # take the i observation
    cv_data = data[-i] # exclude i observation from sample
    cv_pdf_on_i = estimated_pdf(x_i, cv_data, h)
    cv_log_likelihood = cv_log_likelihood + log(cv_pdf_on_i) # log() = ln()
  }
  # print(paste("h=", h, " Log likelihood=", cv_log_likelihood))
  cv_log_likelihood_seq = append(cv_log_likelihood_seq, cv_log_likelihood)
}

# plot(x=h_seq[100:300], y=cv_log_likelihood_seq[100:300], type="l") # plot log likelihood functi


#plot the log likelihood vs h
df_ll = data.frame(x=h_seq, y=cv_log_likelihood_seq)
figure2 <- ggplot(data = df_ll, aes(x=x,y=y)) +
  geom_line(color='indianred', size=1.1) +
  labs(title = "Cross-validated log-likelihood vs h",
       x="h", y = "LL(h)") # title

figure2 # show figure
```

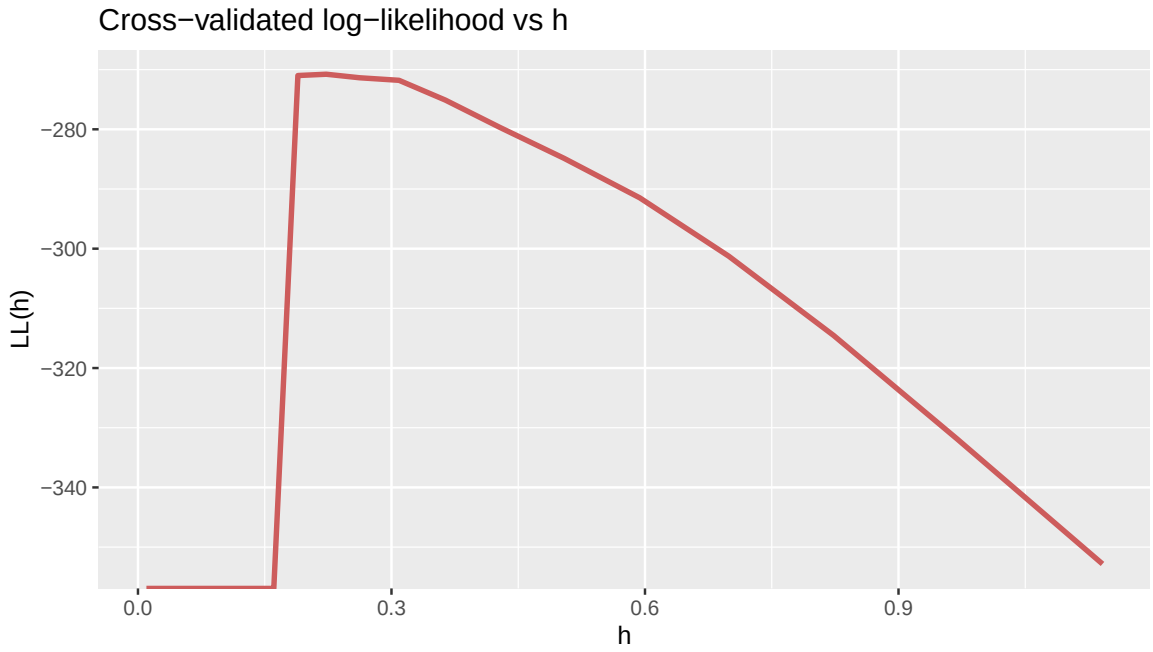

Figure 11: (2a) - Graph of cross-validated log-likelihood function $LL(h)$

It is clear from the graph that the maximum value of log-likelihood exists at some value between the range $[0.15, 0.3]$. To further narrow down the search range for h, we take the maximum and the second maximum values of log-likelihood for the h values

range selected previously, and then we again explore a logarithmic range between the two corresponding h values taking 30 points in between.
Plotting the log-likelihood function vs h for the second narrower range we get the following graph.

```
# to narrow it down we find the max and second max value
ll_max = max(cv_log_likelihood_seq) # max LL(h)
idx_max = which(cv_log_likelihood_seq == ll_max) # index of max
ll_2max = max(cv_log_likelihood_seq[cv_log_likelihood_seq != ll_max]) # 2nd max
idx_2max = which(cv_log_likelihood_seq == ll_2max) # index of max
# define new h range to perform a seach for the optimal h
h_start = min(h_seq[idx_max], h_seq[idx_2max])
h_end = max(h_seq[idx_max], h_seq[idx_2max])

h_seq = log_seq(h_start, h_end, num=30) # selected a logarithmic range for h

cv_log_likelihood_seq = c()

for (h_index in 1:length(h_seq)){
  h = h_seq[h_index]
  cv_log_likelihood = 0
  for (i in 1:n){
    x_i = data[i]
    cv_data = data[-i]
    cv_pdf_on_i = estimated_pdf(x_i, cv_data, h)
    cv_log_likelihood = cv_log_likelihood + log(cv_pdf_on_i)
  }
  cv_log_likelihood_seq = append(cv_log_likelihood_seq, cv_log_likelihood)
}

# plot again to get a more zoomed and accurate picture around the max
df_ll = data.frame(x=h_seq, y=cv_log_likelihood_seq)
figure3 <- ggplot(data = df_ll, aes(x=x,y=y)) +
  geom_line(color='indianred', size=1.1) +
  xlim(h_start,h_end) +
  ylim(min(cv_log_likelihood_seq), max(cv_log_likelihood_seq)) +
  labs(title = "Cross-validated log-likelihood vs h",
       x="h", y = "LL(h)") # title

figure3 # show figure
```

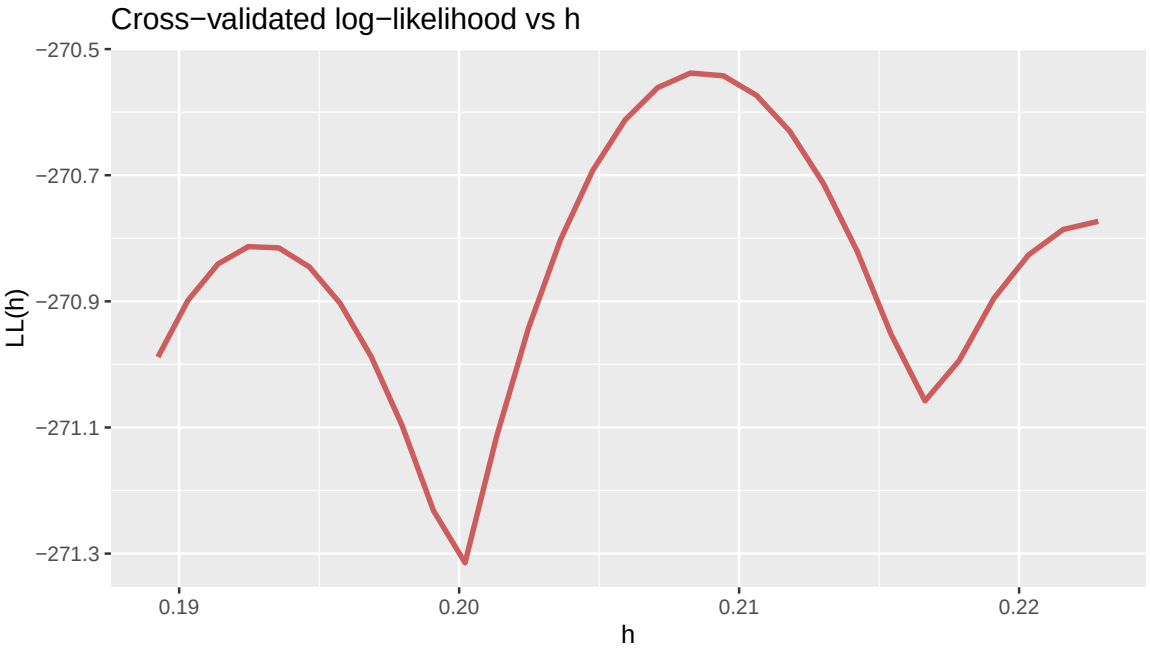

Figure 12: (2a) - Graph of cross-validated log-likelihood function $LL(h)$ (2)

```
max_log_likelihood = max(cv_log_likelihood_seq)
index_max_log_likelihood = which.max(cv_log_likelihood_seq) # index of max
h_optimal = h_seq[index_max_log_likelihood] # corresponding h for max log likelihood

print(paste('The optimal h =', h_optimal))

## [1] "The optimal h = 0.208263183574119"
```

The maximum value now seems to be around 0.21. In fact it's calculated value is $h_{opt} = 0.2082632$ and the cross-validated likelihood at that h value is approximately $LL(h_{opt}) \approx -270.538$.
Using the optimal h_{opt} value, we next proceed to creating a graph of the estimated probability density function $\hat{f}(x)$ using an Epanechnikov kernel and R's density function.

```
# background color only for the plot region
plot.new()
rect(par("usr")[1], par("usr")[3],
     par("usr")[2], par("usr")[4],
     col = "#ebebeb") # background color of ggplot


# add a new plot
par(new = TRUE)

plot(density(data, bw=h_optimal, kernel = "epanechnikov"),
     col='indianred',
     lwd=3,
     main="Estimated f(x) pdf using density function",
     sub='h=optimal=0.208, kernel=Epanechnikov',
     xlab='x (mins)') # plot pdf using optimal calculated h
grid(col = "white")
polygon(density(data, bw=h_optimal, kernel = "epanechnikov"),
        col = rgb(205/256, 92/256, 92/256, alpha = 0.6))  # color under curve
```

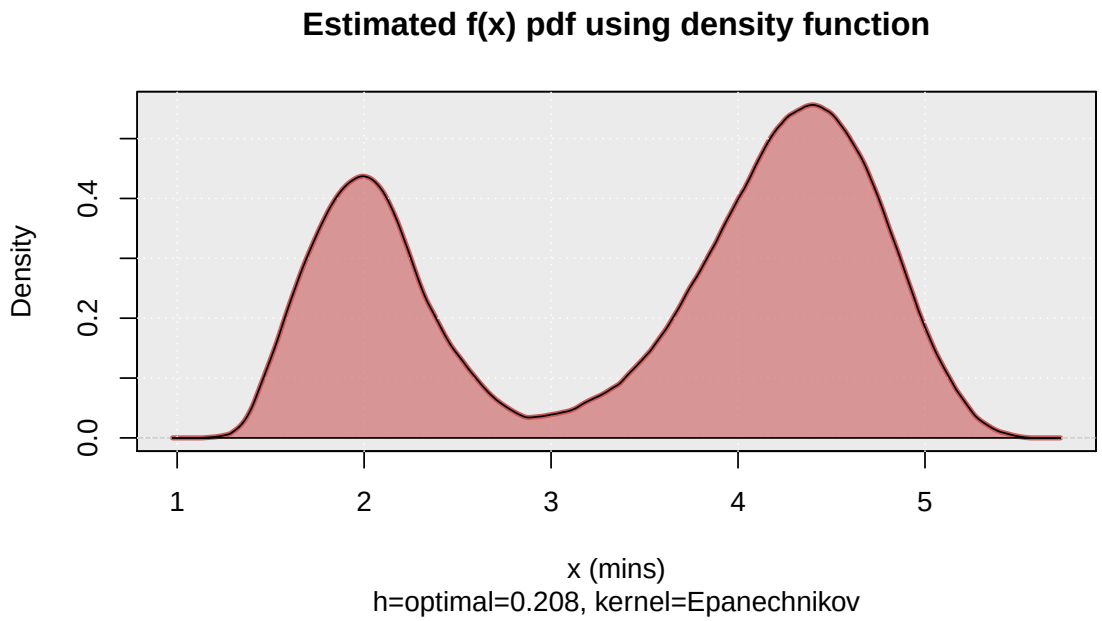

Figure 13: (2a) - Estimated pdf using density function, Epanechnikov kernel

The resulting probability density function plot for the optimal h has two peaks, the higher of which is around 4.3 mins, whereas the second one is around 2 mins. Therefore, two consecutive volcano explosions typically occur within a range of around 4.3 mins with high probability but also around 2 mins with slightly lower probability.

b - Custom pdf estimation and density plots comparison

We now define a custom function for the estimated pdf $\hat{f}(x)$ utilizing an Epanechnikov kernel and the previously presented equations, using $h = h_{opt}$. In the figure generated below, we also include the estimated pdf that was previously found using the density function and compare the two densities.

```
# below function is used in 2b and 2c so that a single pdf function
#includes the kernel implementation inside it too

full_pdf <- function(x, sample=faithful$eruptions, h=0.208){
  n = length(sample)
  summ = 0
  for (i in 1:n){ # calculate [((x-x_i)/h)]
    u = (x-sample[i])/h
    if ((u <= 1) & (u >= -1)){ #if |u|<=1
      kernel = (3/4) * (1-u^2) # Epanechnikov K(u) calculation
    } else {
      kernel = 0 # if |u|>1
    }
    summ = summ + kernel
  }
  pdf = (1/(n * h)) * summ
  return(pdf)
}

#2b
# get density function's x points (default = 512 points)
dens = density(data, bw=h_optimal, kernel = "epanechnikov")
```

```
x_seq = sort(dens$x)
pdf_values = c()
for (i in 1:length(x_seq)){
  x_i = x_seq[i]
  pdf = full_pdf(x_i) # estimated pdf value for each of the x points
  pdf_values = append(pdf_values, pdf) # add to a list
}


# background color only for the plot region
plot.new()
rect(par("usr")[1], par("usr")[3],
     par("usr")[2], par("usr")[4],
     col = "#ebebeb") # background color of ggplot


# add a new plot
par(new = TRUE, cex=0.8)

plot(density(data, bw=h_optimal, kernel = "epanechnikov"),
     col='indianred',
     ylim=c(0,max(pdf_values)),
     lwd=3,
     main='Estimated pdf for optimal h \n
          Custom vs R density function implementation',
     sub='h=optimal=0.208, kernel=Epanechnikov',
     xlab='x (mins)') # plot pdf using optimal calculated h
grid(col="white")
polygon(density(data, bw=h_optimal, kernel = "epanechnikov"),
        col = rgb(205/256, 92/256, 92/256, alpha = 0.6)) # color under curve
par(new=TRUE)
lines(x_seq, pdf_values, type="l", col='darkblue', lwd=3)
polygon(c(min(x_seq), x_seq, max(x_seq), min(x_seq)), c(0, pdf_values, 0, 0),
        col = rgb(55/256, 198/256, 255/256, alpha = 0.2)) # color under curve
legend(
  'topright',
  title = "Implementation",
  col = c('indianred', 'darkblue'),
  lty=1:1,
  legend = c('density \n function', 'custom')
)
```

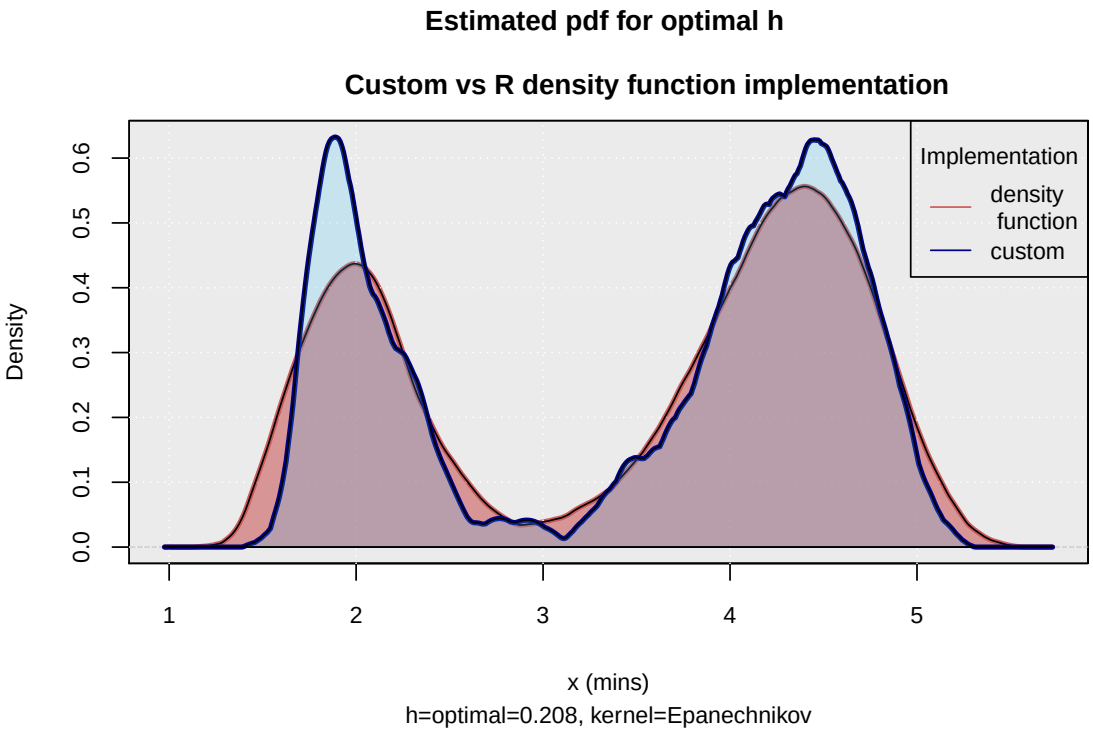

Figure 14: (2b) - Custom pdf vs density estimation, Epanechnikov kernel

The density function of R disperses the mass of the empirical distribution function over a regular grid of a selected number of points (default= 512) and then uses the fast Fourier transform to convolve this approximation with a discretized version of the selected kernel and finally uses linear approximation to evaluate the density at the specified points. This is why the the two densities evaluated (custom and the one with R's density function) differ in practice. Nevertheless, the overall form of the pdf is preserved as well as the values in minutes for which the pdf maximums are found.

c - Probability estimation with integration

We will now calculate the probability of successive volcano explosions taking place more than 3.5 minutes apart. This probability is approximated by using the estimated pdf $\hat{f}(x)$ instead of the real $f(x)$ as

$$\mathbb{P}[X > 3.5] = \int_{3.5}^{+\infty} \hat{f}(x)\, dx, \quad using\ h = h_{opt}.$$

Using the integrate function of R, we perform the calculation and present the results below.

```
#2c
probability = integrate(Vectorize(full_pdf), lower =3.5, upper = Inf)
cat(sprintf("Probability that the time interval between successive volcano
explosions is more than 3.5 minutes: %.5f with %.5f absolute error",
probability$value,
probability$abs.error))

## Probability that the time interval between successive volcano
## explosions is more than 3.5 minutes: 0.61205 with 0.00012 absolute error
```

based on the previous R code block's results, the probability is approximated as $\mathbb{P}[X > 3.5] \approx 61.2\%$ with an absolute error of circa 0.00012.

d - Probability estimation with estimated pdf sampling

We now proceed to estimating again the probability of successive volcano eruptions taking place after more than 3.5 mins by sampling 250 values from the estimated pdf $\hat{f}(x)$ and taking the proportion of the values that were below 3.5. In order to sample from $\hat{f}(x)$, we follow the steps below.

Kernel estimated pdf sampling:

- **Step 1**

  Choose an observation from the initial sample of n observations randomly, with probability $\frac{1}{n}$. We denote this by X.

- **Step 2**

  Sample a random variable Y from Epanechnikov kernel's probability distribution

$$K(y) = \begin{cases} \frac{3}{4}(1-y^2) & |y| \leq 1 \\ 0 & |y| \geq 1 \end{cases}.$$

  This is performed in two steps:

  1. Generate independent and identically distributed $U_1, U_2, U_3 \sim U(-1, 1)$
  2. If $|U_3| \geq |U_2|$ *and* $|U_3| \geq |U_1|$ then:
     $Y = U_2$
     else:
     $Y = U_3$.

- **Step 3**

  $Z = X + h \cdot Y \sim \hat{f}(x)$.

Therefore, we can sample from the estimated pdf using the described algorithm for $n = 272$ and $h = h_{opt} = 0.208$ a total of 250 times to get a simulated sample of size $m = 250$. We implement this sampling procedure using R and present the results below.

```
#2d
set.seed(42)
sample_size = 250 # number of sample points to simulate from estimated f
f_simul_values = c() # list of simulated f values

hY <- apply(matrix(runif(3*sample_size, -h_optimal, h_optimal), 3), 2, median)
x = sample(data, sample_size, replace = TRUE)
f_simul_values = x + hY # Z = X + h*Y sampling

simul_probability = length(which(f_simul_values > 3.5)) / sample_size

cat(sprintf("Simulated probability that the time interval between successive
volcano explosions is more than 3.5 minutes: %.5f",
simul_probability))

## Simulated probability that the time interval between successive
## volcano explosions is more than 3.5 minutes: 0.61200
```

Based on the previous code block's printed results, the probability derived from sampling has a very similar value to the one calculated with integration

$$\mathbb{P}_{integration}[X > 3.5] = 0.61204,$$
$$\mathbb{P}_{simulation}[X > 3.5] = 0.61200.$$

Exercise 3 - Maximum Likelihood and Expectation Maximization for parameter estimation

The third assignment of the report deals with the task of population(s) parameter estimation from which a set of observations has been generated. First, we examine the case where the information about which sub-population each observation belongs to is known. In that scenario, Maximum Likelihood Estimation is used to estimate the unknown populations parameters. Secondly, the case where there is no knowledge regarding which sub-population each observation has been derived from is also examined. In such a case, a mixture of different populations' distributions is assumed and Expectation-Maximization is employed in order to estimate the population parameters.

a - Maximum Likelihood Estimation

Two independent populations are assumed and a set of four observations is provided $\boldsymbol{X} = (x_1, x_2, x_3, x_4) = (2, 7, 3, 9)$. Observations $x_1 = 2, x_3 = 3$ are assumed to belong to the first population, whereas observations $x_2 = 7$ and $x_4 = 9$ belong to the second population. We also assume for the two populations

$$X_1, X_3 \sim \text{Poisson}(\lambda_1), \quad \lambda_1 > 0,$$
$$X_2, X_4 \sim \text{Poisson}(\lambda_2), \quad \lambda_2 > 0.$$

Moreover, π_1 is the probability for one observation to be drawn from the first population and $\pi_2 = 1 - \pi_1$ the corresponding probability for second one.
For the mixture of Poisson distributions, we have

$$f(x \mid \boldsymbol{\theta}) = \sum_{i=1}^{k} \pi_i \cdot f\left(x \mid \theta_i\right) = \sum_{i=1}^{k} \pi_i f\left(x \mid \lambda_i\right),$$

where $k = 2$ is the number of sub-populations. Given the two sub-populations are Poisson, the parameters are $\theta_1 = \lambda_1$, $\theta_2 = \lambda_2$ and $\boldsymbol{\theta} = (\pi_1, \pi_2, \lambda_1, \lambda_2)$ This is the set of parameters we are going to estimate and $\pi_i > 0$, $i = 1, 2$ and $\pi_1 + \pi_2 = 1$. The probability mass function of the two sub-papulations (Poisson) is

$$f\left(x \mid \lambda_i\right) = \mathbb{P}\left[X = x | \lambda_i\right] = \frac{{\lambda_i}^x \cdot e^{-\lambda_i}}{x!}.$$

Thcrefore

$$f\left(x | \lambda_1\right) = \frac{\lambda_1^x \cdot e^{-\lambda_1}}{x!},$$
$$f\left(x | \lambda_2\right) = \frac{\lambda_2^x \cdot e^{-\lambda_2}}{x!}.$$

Then for the mixture we get

$$f(x \mid \boldsymbol{\theta}) = \sum_{i=1}^{2} \pi_i f\left(x \mid \lambda_i\right) = \pi_1 \cdot f\left(x \mid \lambda_1\right) + \pi_2 f\left(x \mid \lambda_2\right) =$$
$$= \pi_1 \cdot \frac{\lambda_1^x \cdot e^{-\lambda_i}}{x!} + \pi_2 \cdot \frac{\lambda_2^x \cdot e^{-\lambda_2}}{x!}.$$

The likelikood of the given sample $\boldsymbol{X} = (x_2, x_2, x_3, x_4)$ is

$$L(\boldsymbol{\theta} \mid \boldsymbol{X}) = \prod_{i=1}^{4} f\left(x_i \mid \boldsymbol{\theta}\right).$$

We are looking to estimate the set of parameters $\boldsymbol{\theta}$ by maximizing the likelikood of the data observations. Given that the logarithmic function is increasing, we can maximize the log-likelihood instead for this maximization problem.

$$LL(\boldsymbol{\theta} \mid \boldsymbol{X}) = \ln L(\boldsymbol{\theta} \mid \boldsymbol{X}) = \ln \prod_{i=1}^{4} f\left(x_i \mid \boldsymbol{\theta}\right) = \sum_{i=1}^{4} \ln f\left(x_i \mid \boldsymbol{\theta}\right) =$$
$$= \sum_{i=1}^{4} \ln \left(\sum_{j=1}^{2} \pi_j f\left(x_i \mid \lambda_j\right) \right).$$

We can see in the previous expression that there is summation inside the logarithm which makes the analytical calculation of the log-likelihood non-trivial.
Next, we introduce the latent random variables $Z_{ij}, i = 1, 2, 3, 4, j = 1, 2$, where

$$Z_{ij} = \begin{cases} 1, & \text{if observation } i \text{ belongs to } j \text{ sub-population} \\ 0, & \text{else} \end{cases}.$$

We rewrite the log-likelikood now

$$LL(\boldsymbol{\theta} \mid \boldsymbol{X}, Z) = \sum_{i=1}^{4} \ln \left(\sum_{j=1}^{2} z_{ij} \pi_j f\left(x_i \mid \lambda_j\right) \right) =$$
$$= \sum_{i=1}^{4} \ln \left[\sum_{j=1}^{2} \left(z_{ij} \pi_j \frac{\lambda_j^{x_i} e^{-\lambda_j}}{x_i!} \right) \right] =$$
$$= \sum_{i=1}^{4} \ln \left(z_{i1} \pi_1 \frac{\lambda_1^{x_i} e^{-\lambda_1}}{x_i!} + z_{i_2} \pi_2 \cdot \frac{\lambda_2^{x_i} \cdot e^{-\lambda_2}}{x_i!} \right).$$

It is evident that having observed Z_{ij}, thus having knowledge regarding the sub-populations to which each observation belongs to, significantly simplifies the log-likelihood calculation. The problem is then converted to maximizing the log-likelihood for the two Poisson sub-populations separately.
Indeed, for $\boldsymbol{X} = (x_1, x_2, x_3, x_4) = (2, 7, 3, 9)$, where the first and third observations follow a Poisson distribution with parameter λ_1 and the second and fourth another Poisson with λ_2, it follows that

$$z_{11} = z_{31} = z_{22} = z_{42} = 1$$
$$z_{12} = z_{32} = z_{21} = z_{42} = 0,$$

and the log-likelinood becomes

$$\begin{aligned}
LL(\boldsymbol{\theta} \mid \boldsymbol{X}, Z) = & \ln\left(\pi_1 \frac{\lambda_1^{x_1} \cdot e^{-\lambda_1}}{x_1!}\right) + \ln\left(\pi \frac{\lambda_1 x_3 \cdot e^{-\lambda_1}}{x_3!}\right) \\
& + \ln\left(\pi_2 \frac{\lambda_2^{x_2} e^{-\lambda_2}}{x_2!}\right) + \ln\left(\pi_2 \frac{\lambda_2 \cdot e^{x_4} - \lambda_2}{x_4!}\right) \\
& = \ln \pi_1 + x_1 \ln \lambda_1 - \lambda_1 - \ln(x_1!) \\
& + \ln \pi_1 + x_3 \ln \lambda_1 - \lambda_1 - \ln(x_3!) \\
& + \ln \pi_2 + x_2 \ln \lambda_2 - \lambda_2 - \ln(x_2!) \\
& + \ln \pi_2 + x_4 \ln \lambda_2 - \lambda_2 - \ln(x_4!) \\
& = 2 \ln \pi_1 + 2 \ln \pi_2 + (x_1 + x_3) \ln \lambda_1 \\
& + (x_2 + x_4) \ln \lambda_2 - 2\lambda_1 - 2\lambda_2 - \ln\left(x_1! x_2! x_3! x_4!\right) \\
& = 2 \ln \pi_1 + 2 \ln\left(1 - \pi_1\right) + (x_1 + x_3) \ln \lambda_1 \\
& + (x_2 + x_4) \ln \lambda_2 - 2\lambda_1 - 2\lambda_2 - \ln\left(x_1! x_2! x_3! x_4!\right).
\end{aligned}$$

To maximize the log-likelihood with respect to each one of the parameters we are seeking to estimate, we calculate the appropriate partial derivatives and set them to zero.

$$\begin{aligned}
\frac{\partial LL(\boldsymbol{\theta} \mid \boldsymbol{X}, Z)}{\partial \lambda_1} = 0 \Rightarrow \frac{x_1 + x_3}{\lambda_1} - 2 = 0 \Rightarrow \lambda_1 = \frac{x_1 + x_3}{2} = \frac{2+3}{2} = 2.5, \\
\frac{\partial LL(\boldsymbol{\theta} \mid \boldsymbol{X}, Z)}{\partial \lambda_2} = 0 \Rightarrow \frac{x_2 + x_4}{\lambda_2} - 2 = 0 \Rightarrow \lambda_2 = \frac{x_2 + x_4}{2} = \frac{7+9}{2} = 8, \\
\frac{\partial LL(\boldsymbol{\theta} \mid \boldsymbol{X}, Z)}{\partial \pi_1} = 0 \Rightarrow \frac{2}{\pi_1} + \frac{2}{1 - \pi_1} \cdot (-1) = 0 \Rightarrow \frac{1}{\pi_1} = \frac{1}{1 - \pi_1} \\
\Rightarrow \pi_1 = 1 - \pi_1 \Rightarrow 2\pi_1 = 1 \Rightarrow \pi_1 = \frac{1}{2}.
\end{aligned}$$

Then $\pi_2 = 1 - \pi_1 = 1 - \frac{1}{2} = \frac{1}{2}$.

b - Expectation Maximization

In this case, Z_{ij} or equally the information regarding which sub-population each observation is drawn from is considered missing and the Expectation-Maximization algorithm that has two steps is used. On the E-step of the algorithm, we estimate the missing data with their conditional expected value, whereas on the M-step, those estimates are used in order to update the parameter values. It can be proved that on the M-step, only the calculation of weighted averages with the weights being the results of the E-step is required.
We had previously found the log-likelihood as

$$LL\left(\boldsymbol{\theta} | \boldsymbol{X}, Z\right) = \sum_{i=1}^{4} \underbrace{\ln\left(z_{i1} \pi_1 \frac{\lambda_1^{x_i} e^{-\lambda_1}}{x_i!} + z_{i2} \pi_2 \cdot \frac{\lambda_2^{x_i} e^{-\lambda_2}}{x_i!}\right)}_{A_i}.$$

However, for each $i = 1, 2, 3, 4$, if $z_{i1} = 0$, then $z_{i2} = 1$ or if $z_{ij} = 1$, then $z_2 = 0$, and vice versa, which means that each observation follows only one of the two Poisson distributions (sub-populations). We now state that the denoted expression A_i is equal to the following expression

$$A_i = \ln\left(z_{i1}\pi_1\frac{\lambda_1^{x_i}e^{-\lambda_1}}{x_i!} + z_{i2}\pi_2\frac{\lambda_2^{x_i}e^{-\lambda_2}}{x_i!}\right)$$
$$= z_{i1}\ln\left(\pi_1\frac{\lambda_1^{x_i}e^{-\lambda_1}}{x_i!}\right) + z_{12}\ln\left(\pi_2\frac{\lambda_2^{x_1}e^{-\lambda_2}}{x_i!}\right), \quad \forall\ i = 1,2,3,4$$

To prove this, we are distinguishing the following two cases.

- $z_{i1} = 0,\ z_{i2} = 1$

  $$A_i = \ln\left(0\cdot\pi_1\cdot\frac{\lambda_1^{x_i}e^{-\lambda_1}}{x_i!} + 1\cdot\pi_2\frac{\lambda_2^{x_i}e^{-\lambda_2}}{x_i!}\right) = 0\cdot\ln\left(\pi_1\frac{\lambda_1^{x_i}e^{-\lambda_1}}{x_i!}\right)+1\cdot\ln\left(\pi_2\frac{\lambda_2^{x_i}e^{-\lambda_2}}{x_i!}\right),$$

  which is true for any x_i.

- $z_{i1} = 1,\ z_{i2} = 0$

  $$A_i = \ln\left(1\cdot\pi_1\cdot\frac{\lambda_1^{x_i}e^{-\lambda_1}}{x_i!} + 0\cdot\pi_2\frac{\lambda_2^{x_i}e^{-\lambda_2}}{x_i!}\right) = 1\cdot\ln\left(\pi_1\frac{\lambda_1^{x_i}e^{-\lambda_1}}{x_i!}\right)+0\cdot\ln\left(\pi_2\frac{\lambda_2^{x_i}e^{-\lambda_2}}{x_i!}\right),$$

  which is true for any x_i.

Therefore, it has been proved that A_i can be rewritten as

$$A_i = z_{i1}\ln\left(\pi_1\frac{\lambda_1^{x_i}e^{-\lambda_1}}{x_i!}\right) + z_{12}\ln\left(\pi_2\frac{\lambda_2^{x_1}e^{-\lambda_2}}{x_i!}\right),$$

and then the log-likelihood is

$$\begin{aligned} LL\left(\boldsymbol{\theta}|\boldsymbol{X}, Z\right) &= \sum_{i=1}^{4} A_i \\ &= \sum_{i=1}^{4}\sum_{j=1}^{2}\left[z_{ij}\ln\left(\pi_j\frac{\lambda_j^{x_i}\cdot e^{-\lambda_j}}{x_i!}\right)\right] \\ &= \sum_{i=1}^{4}\sum_{j=1}^{2}\left[z_{ij}\ln\pi_j + z_{ij}\left(x_i\ln\lambda_j - \lambda_j - \ln(x_i!)\right)\right]. \end{aligned}$$

This form of log-likelihood could have also been used in the previous subsection for the maximum likelihood parameter estimation while knowing the sub-population of each observation, since it is now much easier to the calculate the log-likelikood's partial derivatives. Indeed

$$\frac{\partial LL\left(\boldsymbol{\theta}\mid\boldsymbol{X}, Z\right)}{\partial\lambda_j} = \sum_{i=1}^{4} z_{ij}\left(\frac{x_i}{\lambda_j} - 1\right) = 0 \Rightarrow$$
$$\sum_{i=1}^{4} z_{ij}\frac{x_i}{\lambda_j} - \sum_{i=1}^{4} z_{ij} = 0 \Rightarrow$$
$$\lambda j = \frac{\sum_{i=1}^{4} z_{ij}x_i}{\sum_{i=1}^{4} z_{ij}} \quad , \quad j = 1,2.$$

Also

$$\frac{\partial LL\left(\boldsymbol{\theta} \mid \boldsymbol{X}, Z\right)}{\partial \pi_1} = \sum_{i=1}^{4} \frac{\partial \left(z_{i1} \ln \pi_1 + z_{i2} \ln \left(1 - \pi_1\right)\right)}{\partial \pi_1} = \sum_{i=1}^{4} \left(\frac{z_{i1}}{\pi_1} - \frac{z_{i2}}{1 - \pi_1}\right) = 0 \Rightarrow$$

$$\frac{1}{\pi_1} \sum_{i=1}^{4} z_{i1} - \frac{1}{1 - \pi_i} \sum_{i=1}^{4} z_{i2} = 0 \Rightarrow$$

$$\sum_{i=1}^{4} z_{i1} - \pi_1 \sum_{i=1}^{4} z_{i1} - \pi_1 \sum_{i=1}^{4} z_{i2} \Rightarrow$$

$$\pi_1 = \frac{\sum_{i=1}^{4} z_{i1}}{\sum_{i=1}^{4} \left(z_{i1} + z_{i2}\right)}, \text{ and } \pi_2 = 1 - \pi_1.$$

The expected value of log-likelihood is

$$\mathbb{E}[LL(\boldsymbol{\theta} \mid \boldsymbol{X}, Z)] = \sum_{i=1}^{4} \sum_{j=1}^{2} \left(\mathbb{E}\left[z_{ij} \mid \boldsymbol{\theta}, \boldsymbol{X}\right] \ln \pi_j\right) + \sum_{i=1}^{4} \sum_{j=1}^{2} \left[\mathbb{E}\left[z_{ij} \mid \boldsymbol{\theta}, \boldsymbol{X}\right] \cdot \left(x_i \ln \lambda_j - \lambda_j - \ln\left(x_i!\right)\right)\right]$$

To calculate the expected value of the log-likelihood of the observed data and then calculate its maximum, first the expected value $\mathbb{E}\left[z_{ij} \mid \boldsymbol{\theta}, \boldsymbol{X}\right]$ needs to be calculated. Z_{ij} is a random variable that, given its definition, follows a Bernoulli distribution and therefore

$$\mathbb{E}\left[z_{ij} \mid \boldsymbol{\theta}, \boldsymbol{X}\right] = \mathbb{P}\left[Z_{ij} = 1 \mid \boldsymbol{\theta}, \boldsymbol{X}\right],$$

which denotes the probability for an observation i to be drawn from the sub-population j, given the set of observations $\boldsymbol{X}$. Then on the M-step, we are looking to maximize $\mathbb{E}[LL(\boldsymbol{\theta} \mid \boldsymbol{X}, Z)]$. Next, we describe the Expectation - Maximization algorithm broken down in four steps.

Expectation - Maximization:

- **Step 1**

  We assume $\pi_1, \ldots, \pi_k$ and $\theta_1, \ldots, \theta_k$ parameter values from the previous iteration of the algorithm. Those values can be initialized either randomly or by applying some other initialization logic.

- **Step 2, E-step**

  Calculate

  $$w_{ij} = \frac{\pi_j f\left(x_i \mid \theta_j\right)}{\sum_{j=1}^{k} \pi_j f\left(x_i \mid \theta_j\right)},$$

  where w_{ij} are the posterior probabilities $\mathbb{P}\left[Z_{ij} = 1 \mid \boldsymbol{\theta}, \boldsymbol{X}\right]$ for the observation i to belong to sub-population j.

- **Step 3, M-step**

  Find new estimations for the parameters

  $$\pi_j^{(new)} = \frac{\sum_{i=1}^{n} w_{ij}}{n} \text{ and } \theta_j^{(new)} = \frac{\sum_{i=1}^{n} w_{ij} x_i}{\sum_{i=1}^{n} w_{ij}}$$

- **Step 4**

  If a termination criterion is met then stop iterating, else go back to Step 2 (repeat E-step).

It is important to note here that the EM algorithm yields different solutions depending on the initialization. Therefore the initial values set at Step 1 can affect both the results but also the convergence speed, ie. the number of iterations until convergence.
For the task at hand, we have $k = 2$ and $\theta_1 = \lambda_1$, $\theta_2 = \lambda_2$ as well as $n = 4$ observations. Therefore, we get for the E and M-steps:

- **E-step**

$$w_{ij} = \frac{\pi_j f\left(x_i \mid \theta_j\right)}{\sum_{j=1}^{2} \pi_j \left(x_i \mid \theta_j\right)} = \frac{\pi_j f\left(x_i \mid \lambda_j\right)}{\sum_{j=1}^{2} \pi_j f\left(x_i \mid \lambda_j\right)},$$

  from which we derive

$$\begin{aligned}
w_{11} &= \frac{\pi_1 f\left(x_1 \mid \lambda_1\right)}{\pi_1 f\left(x_1 \mid \lambda_2\right) + \pi_2 f\left(x_1 \mid \lambda_2\right)}, \\
u_{12} &= \frac{\pi_2 f\left(x_1 \mid \lambda_2\right)}{\pi_1 f\left(x_1 \mid \lambda_2\right) + \pi_2 f\left(x_1 \mid \lambda_2\right)}, \\
w_{21} &= \frac{\pi_1 f\left(x_2 \mid \lambda_1\right)}{\pi_1 f\left(x_2 \mid \lambda_2\right) + \pi_2 f\left(x_2 \mid \lambda_2\right)}, \\
w_{22} &= \frac{\pi_2 \cdot f\left(x_2 \mid \lambda_2\right)}{\pi_1 f\left(x_2 \mid \lambda_1\right) + \pi_2 f\left(x_2 \mid \lambda_2\right)},
\end{aligned}$$

  and we can derive $w_{31}, w_{32}, w_{41}, w_{42}$ similarly for the remaining two observations, where $f\left(x_i \mid \lambda_j\right) \sim \text{Poisson}\left(\lambda_j\right)$, thus $f\left(x_i \mid \lambda_j\right) = \frac{\lambda_j{}^{x_i} \cdot e^{-\lambda_j}}{x_i!}$.

- **M-step**

  Update the parameter estimations

$$\begin{aligned}
\pi_j^{(\text{new})} &= \frac{\sum_{i=1}^{4} w_{ij}}{4}, \text{ and therefore we get} \\
\pi_1^{(new)} &= \frac{1}{4}\sum_{i=1}^{4} w_{i1} \text{ and } \pi_2^{(new)} = \frac{1}{4}\sum_{i=1}^{4} w_{i2}, \\
\theta_j^{(\text{new})} &= \lambda_j^{(\text{new})} = \frac{\sum_{i=1}^{4} w_{ij} x_i}{\sum_{i=1}^{1} w_{ij}}, \text{ hence we get} \\
\lambda_1^{(\text{new})} &= \frac{\sum_{i=1}^{1} w_{i1} x_i}{\sum_{i=1}^{1} w_{i1}} \text{ and } \lambda_2^{(\text{new})} = \frac{\sum_{i=1}^{4} w_{i2} x_i}{\sum_{i=1}^{n} w_{i2}}.
\end{aligned}$$

Below, we have included the R code to implement the EM algorithm. The previously described algorithm's implementation also includes a termination criterion; the algorithm terminates when the following condition is satisfied between two consecutive iterations (r) and $(r+1)$ of the algorithm

$$\left(\lambda_1^{(r+1)} - \lambda_1^{(r)}\right)^2 + \left(\lambda_2^{(r+1)} - \lambda_2^{(r)}\right)^2 \leq 10^{-10}.$$

The previous condition essentially quantifies the amount of difference in the estimation of parameters λ_1, λ_2 across two iterations. If that metric is below a threshold, then the update was not significant enough to justify additional iterations and the EM algorithm terminates.

```r
# 3b. EM for unknown parameter(s) estimation

set.seed(42) # generate reproducible results across runs

# --- IMPORTS ---

library(ggplot2)

# --- FUNCTIONS ---

EM_algorithm <- function(x, p, l, tolerance=1e-10){
  n = length(x) # length of sample / number of observations
  m = length(p) # number of populations/mixture distributions
  W = matrix(,nrow=n, ncol=m) # create empty matrix for w_ij
  # save in a list the p1,p2,l1,l2 values we find over iterations
  p1_values = c(p[1])
  p2_values = c(p[2])
  l1_values = c(l[1])
  l2_values = c(l[2])
  diffs = c()

  repeat{ # repeat for as long as the condition concerning tolerance is satisfied

    # E-step - recalculate W matrix values  (wij)
    for (i in 1:n){ # iterate over rows (observations)

      # calculate denominator
      denominator = 0 # this is the sum we see in the wij equation's denominator
      for (j in 1:m){ # iterate over columns (populations)
        denominator = denominator + p[j] * dpois(x[i], lambda=l[j])
      }

      # calculate wij
      for (j in 1:m){ # iterate over columns (populations)
        W[i,j] = p[j] * dpois(x[i], lambda=l[j]) / denominator # update matrix
      }
    }

    # M-step - update p and l values
    for (j in 1:m){

      p[j] = 0 # new estimations of p1, p2
      l[j] = 0 # new estimations of l1, l2
      for (i in 1:n){
        p[j] = p[j] + W[i, j]
        l[j] = l[j] + W[i, j] * x[i]
      }
      l[j] = l[j] / p[j]
      p[j] = p[j] / n
    }
    p1_values = append(p1_values, p[1])
    p2_values = append(p2_values, p[2])

    l1_values = append(l1_values, l[1])
    l2_values = append(l2_values, l[2])

    # repeat condition to exit
    l1_last_two_values = tail(l1_values, 2)
    l2_last_two_values = tail(l2_values, 2)
    difference = (l1_last_two_values[2] - l1_last_two_values[1])^2
```

```
    difference = difference + (l2_last_two_values[2] - l2_last_two_values[1])^2
    if (i==1){ # for the first iteration keep two diffs
      diffs = append(diffs, difference)
    }
    diffs = append(diffs, difference)
    if (difference <= tolerance){
      break # exit the repeat loop
    }
  } # end of repeat loop

  diffs = append(diffs, difference) # duplicate last diff
  return(list(p1=p1_values, p2=p2_values,
              l1=l1_values, l2=l2_values, diffs=diffs))
}
```

In the previous code block, we set the random seed for generating reproducible random results across runs, imported the ggplot library that will be used for plotting and defined a function that implements the EM algorithm. The function takes as input a vector of the values of the observations, another two vectors for the (π_1, π_2) and (λ_1, λ_2) values and a tolerance level with the default being 10^{-10}. Its output is a list that contains the final parameter estimates $(\pi_1, \pi_2, \lambda_1, \lambda_2)$ as well as the values of the termination criterion for each iteration to visualize how fast the algorithm converges. We also define four functions below. The first one prints the resulting π_1, π_2, λ_1 and λ_2 values after the convergence of the algorithm as well as the number of iterations until convergence. The subsequent ones are used for plotting the estimated parameter values per iteration until convergence as well as the convergence criterion values over iterations to demonstrate the speed of convergence.

```
print_results <- function(result){

  num_iterations = length(result$p1)

  p1=tail(result$p1, 1)
  p2=tail(result$p2, 1)
  l1=tail(result$l1, 1)
  l2=tail(result$l2, 1)

  cat(sprintf("Number of iterations before convergence: %d
  l1: %.3f
  l2: %.3f
  p1: %.3f
  p2: %.3f",
  num_iterations,
  l1,
  l2,
  p1,
  p2))

}

pi_plots <- function(result){

  df = data.frame(p1=result$p1, p2=result$p2,
                  l1=result$l1, l2=result$l2, diffs=result$diffs)

  # p1, p2 plots
  colors <- c("p1" = "darkblue", "p2" = "indianred")
```

```
    ggplot(data=df, aes(x=1:nrow(df))) +
    geom_line(aes(y=p1, color='p1'), size = 1) +
    geom_point(aes (y=p1, color='p1'), size = 2) +
    geom_line(aes(y=p2, color='p2'), size = 1) +
    geom_point(aes (y=p2, color='p2'), size = 2) +
    labs(
      title = 'p1 and p2 values over EM iterations',
      x = 'Iteration',
      y = 'p value',
      color = 'Legend') +
    scale_color_manual(values = colors) +
    theme(legend.position = c(0.9, 0.85))
}

lambda_plots <- function(result){

  df = data.frame(p1=result$p1, p2=result$p2,
                  l1=result$l1, l2=result$l2, diffs=result$diffs)

  # l1, l2 plots
  colors <- c("l1" = "darkblue", "l2" = "indianred")

    ggplot(data=df, aes(x=1:nrow(df))) +
    geom_line(aes(y=l1, color='l1'), size = 1) +
    geom_point(aes(y=l1, color='l1'), size = 2) +
    geom_line(aes(y=l2, color='l2'), size = 1) +
    geom_point(aes(y=l2, color='l2'), size = 2) +
    labs(
      title = 'l1 and l2 values over EM iterations',
      x = 'Iteration',
      y = 'l value',
      color = 'Legend') +
    scale_color_manual(values = colors) +
    theme(legend.position = c(0.9, 0.85))
}

convergence_plot <- function(result){
  df = data.frame(p1=result$p1, p2=result$p2,
                  l1=result$l1, l2=result$l2, diffs=result$diffs)

  # difference (convergence criterion) plot
  conv_limit <- function(x) 1e-10 # define M constant function

    ggplot(data=df, aes(x=1:nrow(df))) +
    geom_line(aes(y=diffs, color='criterion'), size = 1) +
    geom_point(aes(y=diffs, color='criterion'), size = 2) +
    scale_y_continuous(trans='log10') +
    stat_function(fun=conv_limit, mapping = aes(color="1e-10"), size=1.1) +
    scale_color_manual(name = "Function:",
                       values = c("darkblue", "indianred"), # Colors
                       labels = c("criterion", "1e-10")) +
    theme(legend.position = c(0.9, 0.7)) +
    labs(
      title = 'Convergence criterion value over EM iterations',
      x = 'Iteration',
      y = 'Criterion value')

}
```

As stated earlier in the report, the EM algorithm can lead to different results depend-

ing on the initialization of the values. As a first approach, we will assume that the parameter values from the previous subsection that were estimated with MLE with the knowledge of each observation's distribution can serve as a proper initialization. In this case we would get the following results for the parameters estimation.

```
# --- MAIN ---
#3b

sample = c(2,7,3,9) # data observations

# initialize p and lambda values
# run with the values that were found theoretically with MLE in 2a
p = c(0.5, 0.5) # p = [p1, p2]
l = c(2.5, 8) # lambda = [lambda1, lambda2]

result = EM_algorithm(sample, p, l) # EM algorithm

# ---  PRINT RESULTS ---
print_results(result)

## Number of iterations before convergence: 21
##   l1: 2.683
##   l2: 7.402
##   p1: 0.456
##   p2: 0.544
```

We then follow another approch by initializing the π and λ values randomly, choosing a value for π_1 from a uniform distribution in $[0, 1]$ and then setting $\pi_2 = 1 - \pi_2$, while we select λ_1 and λ_2 values uniformly in the range $[1, 20]$.

```
# initialize p and lambda values
p1 = runif(1, min=0, max=1) # random value for p1 in [0,1] and then p2=1-p1
p = c(p1, 1-p1) # p = [p1, p2]

# lambda = [lambda1, lambda2] random initialization with a value between [1,20]
l = runif(2, min=1, max=20)

result = EM_algorithm(sample, p, l) # EM algorithm

# ---  PRINT RESULTS ---
print_results(result)

## Number of iterations before convergence: 35
##   l1: 7.402
##   l2: 2.683
##   p1: 0.544
##   p2: 0.456
```

We can see that both initialization methods led essentially to the same result since we can interchange the indexes of the parameters of the two Poisson distributions without making any difference. The parameter values to which the algorithm converged are

$$\lambda_1 = 7.402, \quad \lambda_2 = 2.683,$$
$$\pi_1 = 0.544, \quad \pi_2 = 0.456.$$

However, the first initialization on the previously yielded results with the MLE converged faster as it took 21 iterations instead of 35. Below we display the values of the various parameters over the iterations of the EM algorithms for the latter (random) initialization method.

```
pi_plots(result) # display plot of converging values p1, p2
```

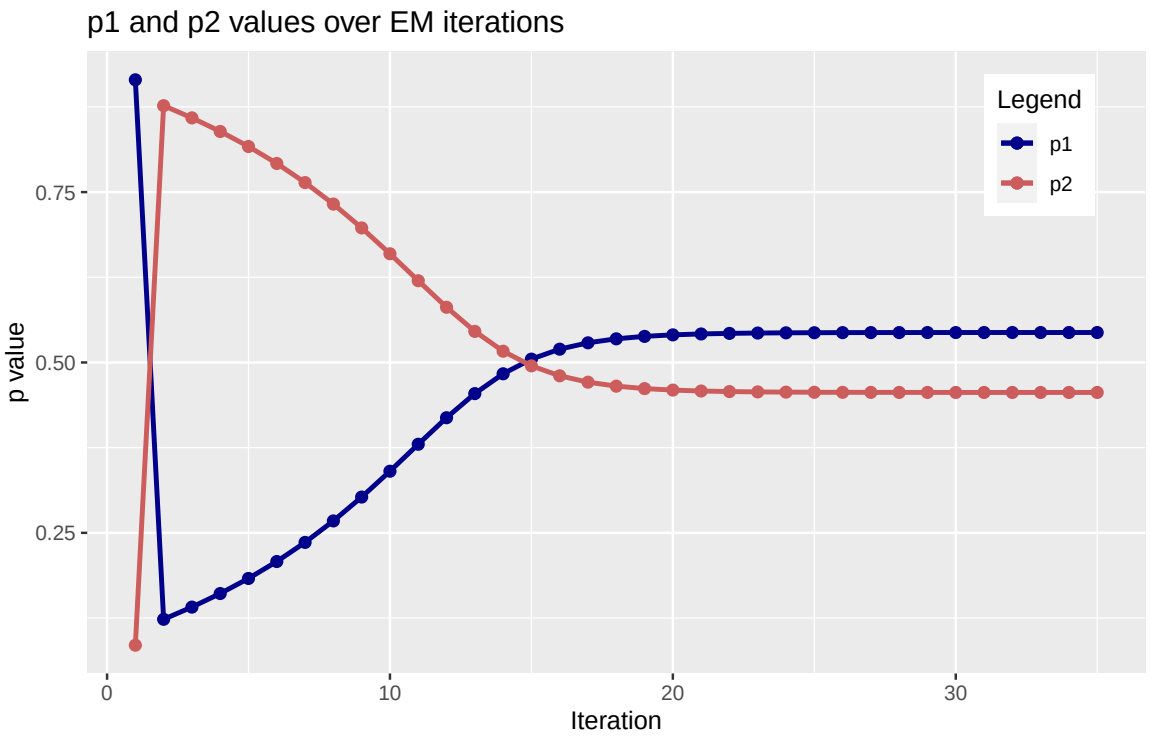

Figure 15: (3b) - p_1 and p_2 values over EM iterations

```
lambda_plots(result) # display plot of converging values l1, l2
```

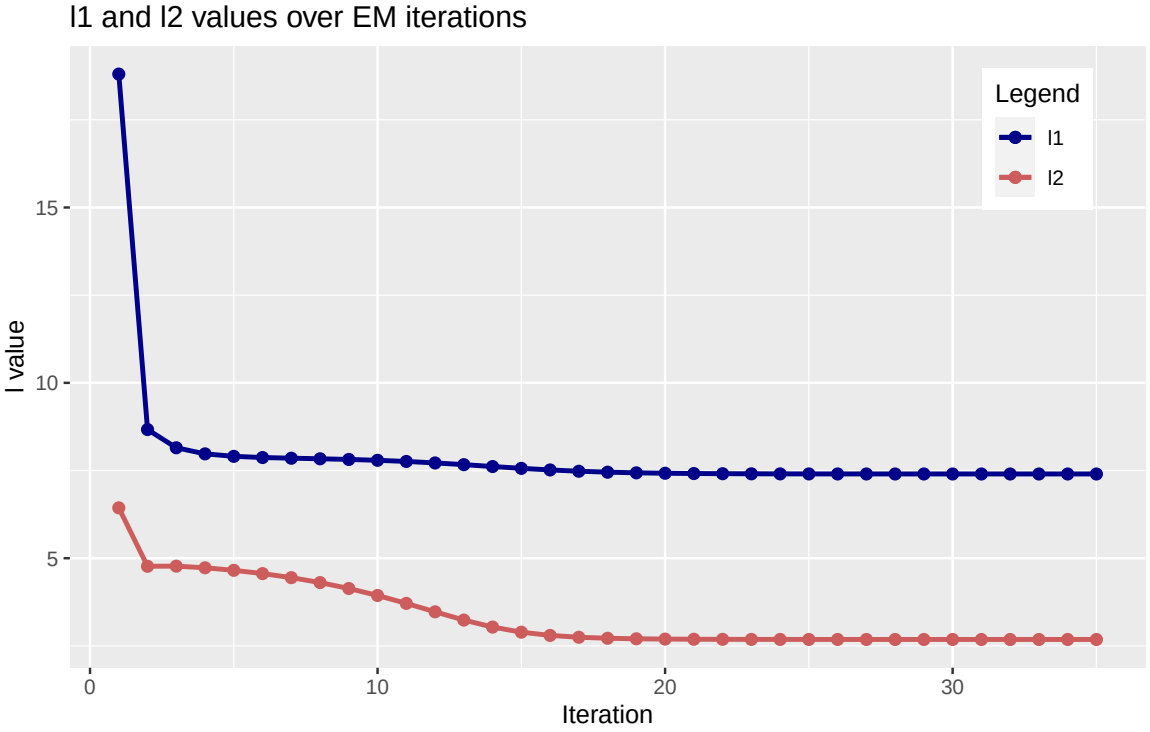

Figure 16: (3b) - l_1 and l_2 values over EM iterations

```
convergence_plot(result) # convergence criterion values in each iteration
```

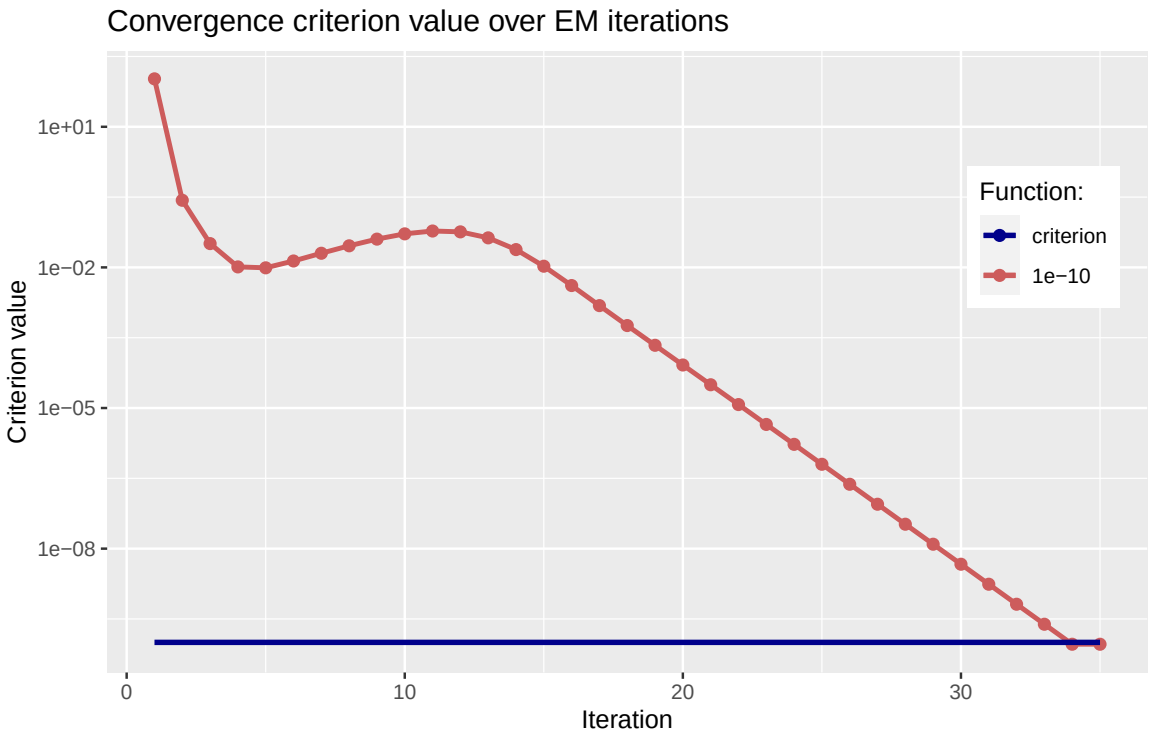

Figure 17: (3b) - Convergence criterion values over EM iterations

From all the generated plots we can observe that the convergence could also be achieved in less than 35 iterations if a more loose criterion was set for convergence. For instance, setting the convergence criterion to 10^{-4} instead of 10^{-10} could result in convergence within 20 iterations, while the resulting parameter values would be very similar.

Exercise 4 - Multiple linear regression model variable selection and evaluation

In the final section of this report, we explore the problem of feature/variable selection for a multiple linear regression model that predicts the level of progression of diabetes for 442 patients from a total of 10 predictor variables. The variables are age, sex, bmi (body mass index), map (mean arterial pressure) and six body urea measurements (tc, ldl, hdl, tch, ltg and glu). All predictor variable values have been normalized in order to sum to zero and have sum of squares equal to one. Therefore we have

$$Y : \text{target variable},$$

$$\underbrace{X_1}_{\text{age}}, \underbrace{X_2}_{\text{sex}}, \underbrace{X_3}_{\text{bmi}}, \underbrace{X_4}_{\text{map}}, \underbrace{X_5}_{\text{tc}}, \underbrace{X_6}_{\text{ldl}}, \underbrace{X_7}_{\text{hdl}}, \underbrace{X_8}_{\text{tch}}, \underbrace{X_9}_{\text{ltg}}, \underbrace{X_{10}}_{\text{glu}} : \text{predictor variables},$$

$$X_i^{(1)} + X_i^{(2)} + \cdots + X_i^{(n)} = 0, \ \textit{and}$$

$$\left(X_i^{(1)}\right)^2 + \left(X_i^{(2)}\right)^2 + \cdots + \left(X_i^{(n)}\right)^2 = 1, \text{where } n = 442, \ i = 1, 2, \ldots, 10.$$

The multiple linear regression model can then be written as

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon, \quad \epsilon \sim N(0, \sigma^2),$$

or equivalently

$$Y \sim N(\mu, \sigma^2), \quad \mathbb{E}(Y) = \mu = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p.$$

We are looking to estimate the parameters $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ with $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_p$ based on the available observations, and the fitted model is then

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \ldots + \hat{\beta}_p X_p,$$

where the estimated parameters $\hat{\beta}_j$ are calculated by minimizing the sum of squared errors, i.e.

$$\text{minimize} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)^2.$$

The fitted model to the training data then has the following parameters $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_p$ and is used to predict the target variable given a new observation j with its predictor variables values being $(X_{1,j}, X_{2,j}, \ldots, X_{p,j})$. The prediction of the model for the observation j will be

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_{1,i} + \hat{\beta}_2 X_{2,i} + \ldots + \hat{\beta}_p X_{p,i}.$$

In the problem explored here, the predictor variables are 10 in number, which means that $p = 10$.

a - Full enumeration of possible models

A first approach towards selecting a subset of the ten predictor variables set to construct a multiple linear regression model for predicting in the best possible way the target variable is to explore the full space of possible models i.e. explore all combinations of variables and evaluate the resulting models based on some criterion. The total number of possible models under consideration (i.e. total number of covariates' combinations) is 2^p, where $p = 10$ is the number of all predictor variables. Therefore there are $2^{10} = 1024$ models that we are going to take under consideration. We can represent the set of all such models by a vector of binary indicators $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p) \in \{0, 1\}^p$, denoting which of the $p = 10$ explanatory variables are considered in the linear predictor considering the following

$$\gamma_i = \begin{cases} 1, & \text{if variable } i \text{ is included in the model} \\ 0, & \text{otherwise} \end{cases}.$$

Given that the number of possible models seems to be explorable in a reasonable amount of time, we can work with the full enumeration method, i.e. consider all the models based on some criterion such as AIC or BIC. Some other alternative methods that can be more computationally efficient but do not guarantee that the final model will be the optimal one are stepwise methods, backward elimination and forward selection. The criterion we will use throughout this assignment is the Bayesian Information Criterion (BIC), which is defined as

$$BIC = k \cdot \ln(n) - 2 \ln(\hat{L}),$$

where $\hat{L}$ is the maximized likelihood function of the model M, i.e. $\hat{L} = p(\boldsymbol{x}|\hat{\boldsymbol{\theta}}, M)$. Here, $\hat{\boldsymbol{\theta}}$ are the parameter values maximizing the likelihood function, $\boldsymbol{x}$ is the observed data, n is the number of observations in $\boldsymbol{x}$ and k is the number of parameters estimated

by the model. In the multiple linear regression case, the parameters for estimation are the intercept, the predictor variables' multipliers, and the constant variance of the errors, therefore $p+2$ parameters in total, where p is the number of predictor variables included in the model. Models with lower BIC are generally preferred and therefore, k essentially penalizes the inclusion of a large number of predictor variables in the model by increasing BIC when more variables are included.
On the implementation side with R, we first set a random seed, import the required libraries for the task at hand and define two utility functions that assist to implement the binary indicators and enumerate all the models.

```
#4 - diabetes prediction models
set.seed(4) # generate reproducible results across runs

# --- IMPORTS ---
library(ggplot2)
#install.packages("lars") # install lars package just once and then comment out
library(lars)

## Loaded lars 1.2

#install glmnet package, run the below once and then comment out
#install.packages("glmnet", repos = "https://cran.us.r-project.org")

library(glmnet)

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

library(methods) # fetch also the methods of glmnet

# --- FUNCTIONS ---

# convert integer to binary list
int2bin <- function(integer, num_of_bits=10) {
  binary_vector = rev(as.numeric(intToBits(integer)))
  return(binary_vector[-(1:(length(binary_vector) - num_of_bits))])
}

# create string that says which variables are used
concat_string <- function(a_list) {
  full_string = ""
  for (i in 1:length(a_list)) {
    full_string = paste(full_string, a_list[i]) # add variable name
    if (i!=length(a_list)){
      full_string = paste(full_string, " + ") # add sign symbol between
    }
  }
  return(full_string)
}
```

We now load the data and print some useful information regarding the dataset.

```
# --- MAIN ---
# read data
data("diabetes")
attach(diabetes)

## The following object is masked _by_ .GlobalEnv:
##
##     x
```

```r
# create columns for all variables, first column=y=target variable
dataset = data.frame(cbind(y = diabetes$y, diabetes$x))

# display useful info for the dataset and variables
str(dataset) # variables and their types
```

```
## 'data.frame': 442 obs. of  11 variables:
##  $ y  : num  151 75 141 206 135 97 138 63 110 310 ...
##  $ age: num  0.03808 -0.00188 0.0853 -0.08906 0.00538 ...
##  $ sex: num  0.0507 -0.0446 0.0507 -0.0446 -0.0446 ...
##  $ bmi: num  0.0617 -0.0515 0.0445 -0.0116 -0.0364 ...
##  $ map: num  0.02187 -0.02633 -0.00567 -0.03666 0.02187 ...
##  $ tc : num  -0.04422 -0.00845 -0.0456 0.01219 0.00393 ...
##  $ ldl: num  -0.0348 -0.0192 -0.0342 0.025 0.0156 ...
##  $ hdl: num  -0.0434 0.07441 -0.03236 -0.03604 0.00814 ...
##  $ tch: num  -0.00259 -0.03949 -0.00259 0.03431 -0.00259 ...
##  $ ltg: num  0.01991 -0.06833 0.00286 0.02269 -0.03199 ...
##  $ glu: num  -0.01765 -0.0922 -0.02593 -0.00936 -0.04664 ...
```

```r
summary(dataset) # min, mean, max, median etc. per variable
```

```
##        y                age                  sex                 bmi          
##  Min.   : 25.0   Min.   :-0.107226   Min.   :-0.04464   Min.   :-0.090275  
##  1st Qu.: 87.0   1st Qu.:-0.037299   1st Qu.:-0.04464   1st Qu.:-0.034229  
##  Median :140.5   Median : 0.005383   Median :-0.04464   Median :-0.007284  
##  Mean   :152.1   Mean   : 0.000000   Mean   : 0.00000   Mean   : 0.000000  
##  3rd Qu.:211.5   3rd Qu.: 0.038076   3rd Qu.: 0.05068   3rd Qu.: 0.031248  
##  Max.   :346.0   Max.   : 0.110727   Max.   : 0.05068   Max.   : 0.170555  
##       map                  tc                  ldl          
##  Min.   :-0.112400   Min.   :-0.126781   Min.   :-0.115613  
##  1st Qu.:-0.036656   1st Qu.:-0.034248   1st Qu.:-0.030358  
##  Median :-0.005671   Median :-0.004321   Median :-0.003819  
##  Mean   : 0.000000   Mean   : 0.000000   Mean   : 0.000000  
##  3rd Qu.: 0.035644   3rd Qu.: 0.028358   3rd Qu.: 0.029844  
##  Max.   : 0.132044   Max.   : 0.153914   Max.   : 0.198788  
##       hdl                 tch                 ltg          
##  Min.   :-0.102307   Min.   :-0.076395   Min.   :-0.126097  
##  1st Qu.:-0.035117   1st Qu.:-0.039493   1st Qu.:-0.033249  
##  Median :-0.006584   Median :-0.002592   Median :-0.001948  
##  Mean   : 0.000000   Mean   : 0.000000   Mean   : 0.000000  
##  3rd Qu.: 0.029312   3rd Qu.: 0.034309   3rd Qu.: 0.032433  
##  Max.   : 0.181179   Max.   : 0.185234   Max.   : 0.133599  
##       glu          
##  Min.   :-0.137767  
##  1st Qu.:-0.033179  
##  Median :-0.001078  
##  Mean   : 0.000000  
##  3rd Qu.: 0.027917  
##  Max.   : 0.135612
```

```r
anyNA(dataset) # check for NA values -> 0 NA values
```

```
## [1] FALSE
```

We can see there are no missing values in the dataset, therefore no preprocessing is required in this case and we proceed with implementing the code to generate and evaluate all 1024 possible models.

```r
# 4a
dataset_columns = colnames(dataset) # 11 column names incl. target y
indep_vars = dataset_columns[-1] # keep 10 independent variables' names only
print(indep_vars)

##  [1] "age" "sex" "bmi" "map" "tc"  "ldl" "hdl" "tch" "ltg" "glu"

num_models = 2^length(indep_vars) # number of models 2^10

list_of_ = c(1)
models_df = data.frame(
  id = c(0:1023)
)
models_df$num_of_vars = NA
models_df$vars_used = NA
models_df$BIC = NA

# add the null model
models_df$vars_used[1] = 1
models_df$num_of_vars[1] = 0
null_model = lm( y~1, data = dataset["y"]) # null model
models_df$BIC[1] = BIC(null_model) # its BIC


for (i in 2:num_models){ # 2^10 = 1,024 models in total, null already indexed
  variables_mask = int2bin(i-1) # binary mask for variables selection

  # select the appropriate variables each time
  variables_selected = indep_vars[which(variables_mask==1)] # select vars
  models_df$num_of_vars[i] = sum(variables_mask) # update df on number of vars
  models_df$vars_used[i]=concat_string(variables_selected) # update df on vars used
  variables_selected = append(variables_selected, "y") # add y to selected vars
  sub_dataset = dataset[variables_selected] # dataset with selected vars and y

  # for each set of variables selected, use them all to fit linear model
  # this is done through the sub_dataset usage (limited dataset each time)
  model = lm( y~., data = sub_dataset)
  models_df$BIC[i] = BIC(model) # update df with BIC value for each model
}

best_model_index = which.min(models_df$BIC)
best_model_variables = models_df[best_model_index, ]$vars_used
best_model_BIC = models_df[best_model_index, ]$BIC

print("Best model with full enumeration - BIC criterion ")

## [1] "Best model with full enumeration - BIC criterion "

print(paste("Variables:", best_model_variables))

## [1] "Variables:  sex  +  bmi  +  map  +  hdl  +  ltg"

print(paste("BIC:", best_model_BIC))

## [1] "BIC: 4822.9019700381"

# now that we have found the best model, build it and name it M1
M1 = lm(y~sex+bmi+map+hdl+ltg, data=dataset)
summary(M1) # print info on the fitted model
```

```
##
## Call:
## lm(formula = y ~ sex + bmi + map + hdl + ltg, data = dataset)
##
## 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
```

All the possible models were saved in a dataframe denoted as models_df, where the number of variables, the exact variables used and the BIC score were kept. Then it is trivial to retrieve the model with the lowest BIC value and its variables set. We found the following through this method of full enumeration

$$BIC_{min} = 4822.902 \ ,$$

$$\text{Variables of the best model } M_1\text{: sex, bmi, map, hdl, ltg} \ ,$$

and the fitted model M_1 is

$$M_1 : \ \hat{Y} = 152.133 - 235.776X_2 + 523.562X_3 + 326.236X_4 - 289.117X_7 + 474.292X_9.$$

b - Lasso variable selection

Another method for variable selection is Lasso, which is a shrinkage method which can shrink some regression coefficients towards zero. It can achieve both variable selection by eliminating the coefficients of the variables to be removed by the model and regularization, so that the model doesn't overfit the data and can generalize relatively well to new unseen data. Lasso adds a regularization term in the sum of squared errors function that we are looking to minimize in order to obtain the parameter estimates $\hat{\beta}_j$ of β_j, $j = 1, 2, \ldots, p$.

$$\text{minimize} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)^2 + \lambda \sum_{j=1}^{p} |\hat{\beta}_j|,$$

where $p = 10$ in our case and λ is a parameter that measures the importance of the regularization term. By setting $\lambda = 0$, the problem is then converted to fitting the model by minimizing the sum of squared errors as described previously. Once we have performed the minimization, we can select variables for a multiple linear regression model by picking only the variables that have a non-zero coefficient. Then, a multiple linear model can be refitted to the data without the Lasso penalty term, minimizing the standard function of sum of squared errors.

Using the glmnet R library, a grid of λ values is selected and then through the minimization task of the loss function with the Lasso regularization/penalty term, the model is fitted to the data for each λ value, estimating this way all the model parameters. For the different λ values we then plot the 10 coefficient values in a logarithmic scale in order to demonstrate the effect of the λ regularization parameter to the coeffient values of the model.

```
# 4b

fit <- glmnet(x=dataset[, c('age', 'sex', 'bmi', 'map', 'tc',
                            'ldl', 'hdl', 'tch', 'ltg', 'glu')],
              y=dataset[, 'y'],
              alpha=1) # fit is an object of class glmnet

# background color only for the plot region
plot.new()
rect(par("usr")[1], par("usr")[3],
     par("usr")[2], par("usr")[4],
     col = "#ebebeb") # background color of ggplot
# add a new plot
par(new = TRUE, cex=0.8)

plot(fit, xvar='lambda', label=TRUE) # visualize coefficients vs log(lambda) values
title("Coefficients vs log(lambda)", line = 2.5, adj = 0)
grid(col = "white") # set plot grid to white
```

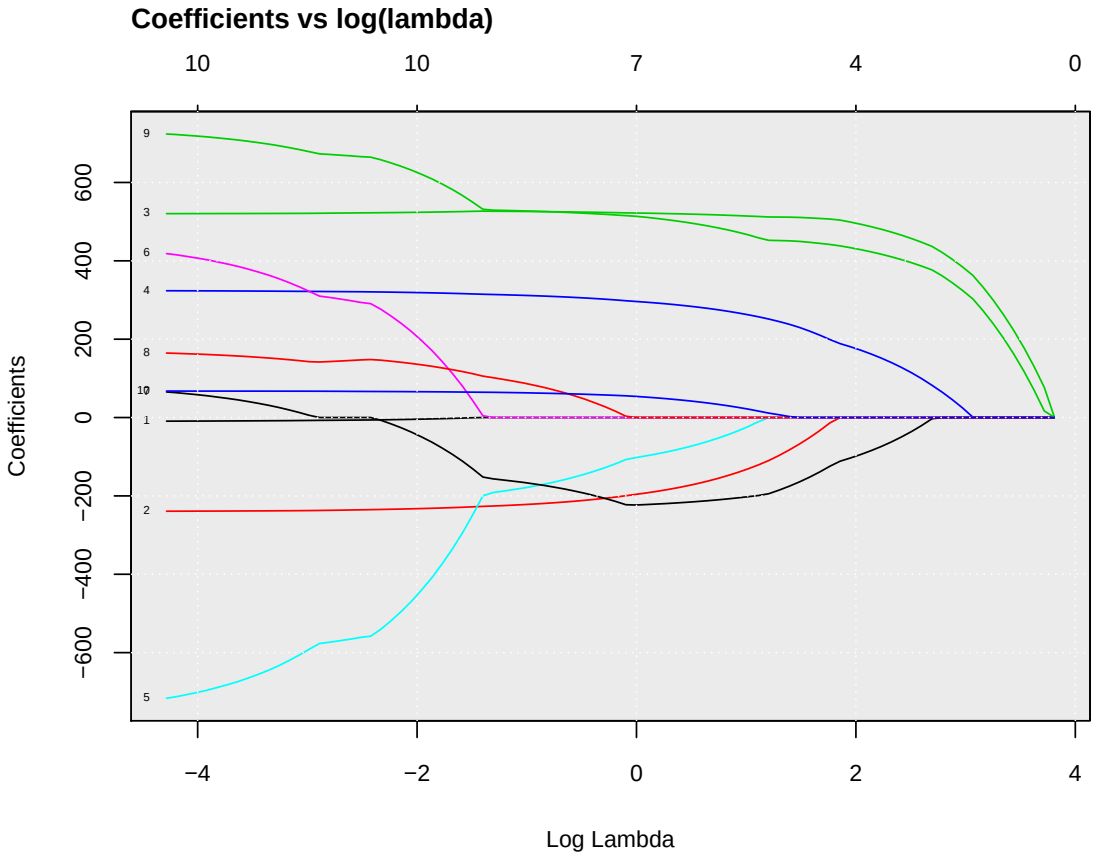

Figure 18: (4b) - Coefficients values vs log(lambda)

We can see from the previous plot, that as the λ value increases, and hence $\log(\lambda)$ too, the coefficients tend to drop in value in general and at different rates, reaching zero at some point. There is also a certain λ value, beyond which all coefficients become zero, which means that the model then becomes the constant model. Since we cannot select the λ value arbitrarily judging from the plot, we employ cross-validation to select the λ parameter.
The function cv.glmnet of the glmnet R library uses 10 folds by default for the cross validation. Cross validation is the process of randomly splitting a dataset in k non-overlapping groups/folds of approximately equal size, where the $k-1$ first folds are used to train/fit a model and the last one that is left out of the training process is used to evaluate the fitted model. The evaluation is based on a metric such as mean squared error (MSE), i.e. $\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}\left(Y_i - \hat{Y}_i\right)^2$, where m is assumed to be the number of observations in the evaluation fold. By repeating this process k times so that each fold has been used as an evaluation fold exactly one time, we can average the MSE values and name this as cross-validated MSE (CV-MSE).
Cv.glmnet creates a custom range of λ values, calculates the CV-MSE and then finds the model over the range of λ values with the lowest CV-MSE. Apart from calculating the minimum CV-MSE and the corresponding λ, it can also calculate the CV-MSE value that is one standard deviation apart from the minimum and the corresponding λ. Below, we present a plot of the CV-MSE over the selected by cv.glmnet range of λ values.

```
# cross-validation stochasticness -> setting seed for reproducible results
set.seed(4)
y=dataset[, c('y')]
x=dataset[, c('age', 'sex', 'bmi', 'map', 'tc', 'ldl', 'hdl', 'tch', 'ltg', 'glu')]
cvfit <- cv.glmnet(as.matrix(x), as.matrix(y), alpha=1, type.measure = "mse")

plot.new()
rect(par("usr")[1], par("usr")[3],
     par("usr")[2], par("usr")[4],
     col = "#ebebeb") # background color of ggplot
# add a new plot
par(new = TRUE, cex=0.8)

plot(cvfit, ylab='CV-MSE') # CV-MSE vs log(lambda)
title("CV-MSE vs log(lambda)", line = 2.5, adj = 0)
grid(col = "white") # set plot grid to white
```

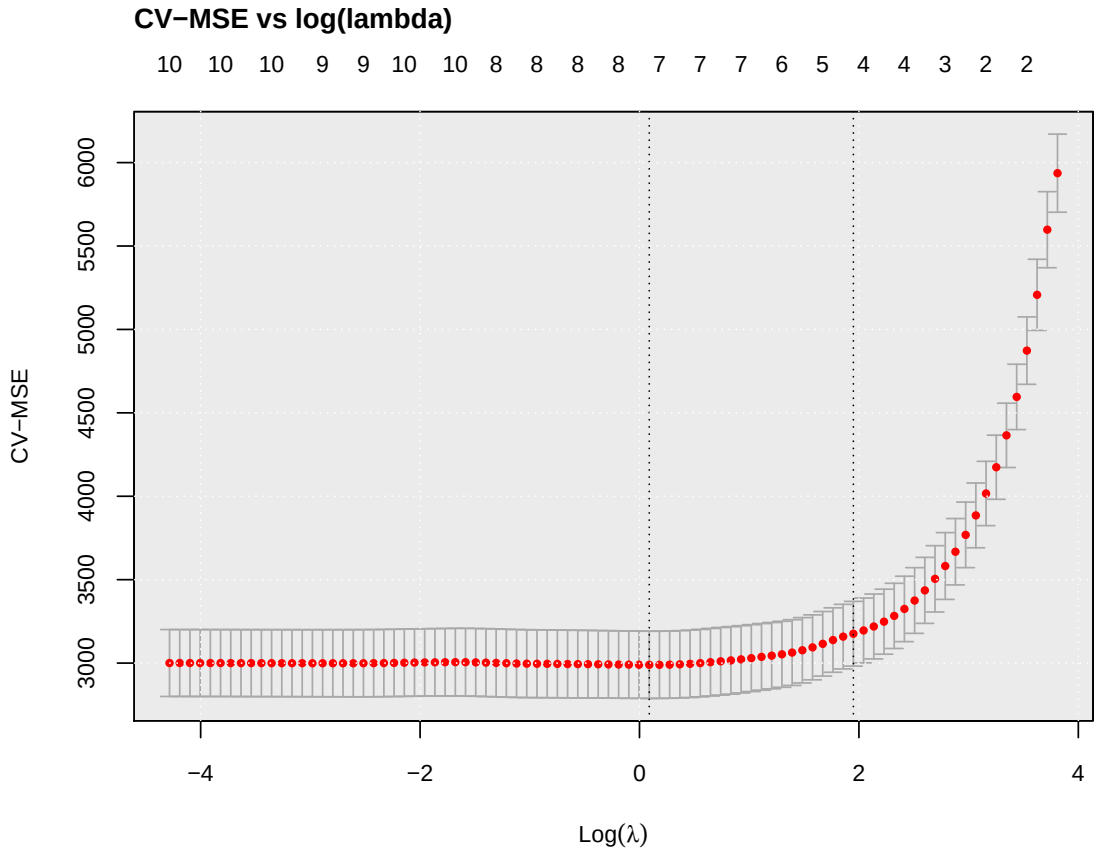

Figure 19: (4b) - CV-MSE vs log(lambda)

On the previous plot, there are two vertical dashed lines. The left one denotes the $\log(\lambda)$ value for which the CV-MSE is minimum, whereas the right one corresponds to the CV-MSE value that is one standard deviation away from the previous minimum CV-MSE value. We will call the model that corresponds to the minimum CV-MSE, M_2, and the model the CV-MSE of which is one standard deviation apart, M_3. Below, we print a summary for M_2.

```
# lambda.min is the value of  that gives minimum mean cross-validated error,
# while lambda.1se is the value of  that gives the most regularized model such
# that the cross-validated error is within one standard error of the minimum

# M2
l_min = cvfit$lambda.min # lambda that gives minimum mean cv error

M2_coeffs = coef(cvfit, s = "lambda.min") # check the coefficients
print("Coefficients of fitted Lasso model:")

## [1] "Coefficients of fitted Lasso model:"

print(M2_coeffs)

## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  152.13348
## age            .
## sex         -192.47489
## bmi          521.64423
```

```
## map          294.40618
## tc           -97.72765
## ldl            .
## hdl         -222.10137
## tch            .
## ltg          511.10045
## glu           52.15790

print(paste('Lambda value for min CV-MSE:', l_min))

## [1] "Lambda value for min CV-MSE: 1.0929306547027"

M2 = lm(y~sex+bmi+map+tc+hdl+ltg+glu, data=dataset) # define model at mean cv error
summary(M2)

##
## Call:
## lm(formula = y ~ sex + bmi + map + tc + hdl + ltg + glu, data = dataset)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -153.966  -39.064   -1.881   38.406  151.076
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  152.133      2.575  59.073  < 2e-16 ***
## sex         -232.747     60.678  -3.836 0.000144 ***
## bmi          526.434     66.184   7.954 1.58e-14 ***
## map          315.366     63.923   4.934 1.15e-06 ***
## tc          -146.347     68.071  -2.150 0.032113 *
## hdl         -235.299     69.810  -3.371 0.000817 ***
## ltg          540.186     78.039   6.922 1.61e-11 ***
## glu           72.181     65.234   1.107 0.269123
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54.14 on 434 degrees of freedom
## Multiple R-squared:  0.5146,Adjusted R-squared:  0.5068
## F-statistic: 65.73 on 7 and 434 DF,  p-value: < 2.2e-16
```

From the printed results of the previous code block, we can see that the λ value that corresponds to the minimum CV-MSE value is $\lambda = 1.093$ and leads to zero the coefficients of age, ldl and tch. Therefore, the model M_2 can then be built by taking into consideration the variable selection that the Lasso method yielded. The Lasso method only contributes to variable selection and the resulting model is fitted to the data again without the Lasso regularization term to form the gitted M_2 model

$$\begin{aligned} M_2 : \hat{Y} &= 152.133 - 233.747X_2 + 526.434X_3 + 315.366X_4 \\ &\quad - 146.347X_5 - 235.299X_7 + 540.186X_9 + 72.181X_{10}. \end{aligned}$$

The λ value for the minimum CV-MSE is relatively low and close to zero and therefore by selecting a CV-MSE value that is a standard deviation apart, we expect to get a model with more coefficients being zero, i.e. a more simple model. By using the variables with non-zero coefficients we then fit the resulting multiple linear model to the data and get the resulting fitted M_3 model.

```r
# M3
l_1se = cvfit$lambda.1se

M3_coeffs = coef(cvfit, s = "lambda.1se") # check the coefficients
print("Coefficients of fitted Lasso model:")

## [1] "Coefficients of fitted Lasso model:"

print(M3_coeffs)

## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     s1
## (Intercept)  152.1335
## age            .
## sex            .
## bmi          498.9191
## map          180.6812
## tc             .
## ldl            .
## hdl         -103.3210
## tch            .
## ltg          433.5648
## glu            .

print(paste('Lambda value for CV-MSE that is 1 sd apart from min:', l_min))

## [1] "Lambda value for CV-MSE that is 1 sd apart from min: 1.0929306547027"

# we can see age, tc, ldl, tch, glu coefficients reached 0
# and we will exclude them from the model

M3 = lm(y~bmi+map+hdl+ltg, data=dataset) # define model at 1 se with some vars removed
summary(M3)

##
## Call:
## lm(formula = y ~ bmi + map + hdl + ltg, data = dataset)
##
## 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
```

From the printed summary for model M_3, we can see that it includes only 4 variables, bmi, map, hdl and ltg. The fitted model to the data using the sum of squared errors is the following then

$$M_3 : \hat{Y} = 152.133 + 555.279X_3 + 269.676X_4 - 193.954X_7 + 484.979X_9.$$

We note here that the described CV-MSE method can yield different results in every run since the split of the data in folds is a random process. Here, we set a random seed to have reproducible results, however, we are aware that setting another seed or no seed at all would yield different results, possibly resulting in slightly different M_2 and M_3 models with different sets of selected variables. One way we could deal with this is running the cross-validation method multiple times for each λ value and averaging the (already averaged k times) CV-MSE values.

c - Models' predictive power evaluation

By now, three models have been defined and fitted to training data, M_1, M_2 and M_3 and in this assignment we are looking to evaluate those models' predictive power based on Root Mean Square Error (RMSE) with a 5-fold cross-validation method. RMSE is measured over a set of n test observations as

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)}.$$

In this case, we are using a 5-fold cross-validation, which means that we divide the dataset in 5 folds and retrain/refit each model 5 times, and evaluate each on the left-out evaluation/test fold, averaging the RMSE values of each evaluation iteration at the end. A random seed is set for reproducibility reasons in this case, since we shuffle the data before splitting the dataset into 5 folds in order to assign data to folds randomly.

```
# 4c

RMSE_func <- function(y_preds, y_real){
  diff = y_preds - y_real
  diff_sq = diff**2
  RMSE = sqrt(sum(diff_sq)/length(diff_sq))
  return(RMSE)
}

set.seed(4)

folds_num = 5
dataset_shuffled = dataset[sample(nrow(dataset)), ] # shuffle data randomly
# 5 folds creation
folds = cut(seq(1, nrow(dataset_shuffled)), breaks=folds_num, labels=FALSE)

RMSE1_seq = c() # keep the RMSE value on the test folds for the 5 folds tested
RMSE2_seq = c()
RMSE3_seq = c()

# performing 5-fold cv
for(i in 1:folds_num){
  # segment your data by fold using the which() function
  test_idx = which(folds==i, arr.ind=TRUE) # get indexes for test set
  test_set = dataset_shuffled[test_idx, ] # get the test fold for test set
  train_set = dataset_shuffled[-test_idx, ] # get the rest folds for train set

  M1_preds = predict(M1, test_set) # predictions of M1 on test set
  RMSE1 = RMSE_func(M1_preds, test_set[['y']])
```

```
  RMSE1_seq = append(RMSE1_seq, RMSE1)

  M2_preds = predict(M2, test_set) # predictions of M2 on test set
  RMSE2 = RMSE_func(M2_preds, test_set[['y']])
  RMSE2_seq = append(RMSE2_seq, RMSE2)

  M3_preds = predict(M3, test_set) # predictions of M3 on test set
  RMSE3 = RMSE_func(M3_preds, test_set[['y']])
  RMSE3_seq = append(RMSE3_seq, RMSE3)

}

RMSE1_mean = mean(RMSE1_seq)
RMSE2_mean = mean(RMSE2_seq)
RMSE3_mean = mean(RMSE3_seq)

cat(sprintf("Models evaluation, average RMSE:
M1: %.5f
M2: %.5f
M3: %.5f",
RMSE1_mean,
RMSE2_mean,
RMSE3_mean))

## Models evaluation, average RMSE:
## M1: 53.94393
## M2: 53.61246
## M3: 54.86858
```

The results of the previous code block show that M_2 is the best model based on the cross-validated RMSE, i.e. has the minimum 5-cv RMSE. Second comes M_1 and last M_3. Model M_1 was expected to score high since it was derived by the full exploration of the models' space, however, since the evaluation criterion has changed from BIC to RMSE, model M_2 yields a better performance. BIC penalizes the usage of extra variables and hence the BIC value of M_2 is lower than that of M_1.
The following table summarizes the results for all three models considered.

Model	Number of variables	Variables	5-cv average RMSE
M_1	5	sex, bmi, map, hdl, ltg	53.94393
M_2	7	sex, bmi, map, tc, hdl, ltg, mu	53.61246
M_3	4	bmi, map, hdl, ltg	54.86858

Table 1: Models comparison