Contents

"Computational Statistics Term Project"

Karpontinis Dimitris

June 30, 2022

1 Exercise 1

1.1 1.a

In this particular exercise we simulate values from the Normal distribution using the rejection sampling method. Specifically we will instead acquire samples from the Cauchy distribution, using the inversion sampling method.

At this point we go into detail on the sampling methods used.

- Rejection Sampling: This method states that in order for us to generate samples from an r.v. with pdf $f(x)$ we must use another r.v. with pdf $g(x)$ such that for some $M \in (1, +\infty)$ it holds that: $f(x) \leq M\ g(x)$

  Specifically M must satisfy: $M = \min\{K > 1 : \frac{f(x)}{g(x)} \leq K,\ \forall x \in \mathbb{R}\}$
  We are actually interested in the number of tries until we find a value x that would be accepted as a value from the desired r.v.. This means that the new r.v. of interest is a geometric random variable with probability of sucess equal to 1/M and mean value equal to M. The above statement (i.e. the calculation of the probability of success) is in the lecture notes (Stochastic Simulation page 14).

  In our case the initial variable of interest follows a standardised normal distribution while the second r.v. follows a cauchy distribution. The value of M is taken to be the smallest one that satisfies the above condition.
  It easily follows that $M = \max_{x \in \mathbb{R}}\{\frac{f(x)}{g(x)}, g(x) \neq 0\}$
  Below we find this value explicitly:

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

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

$$h(x) \coloneqq \frac{\sqrt{\pi}(1+x^2)}{\sqrt{2}\ e^{\frac{x^2}{2}}}$$

Therefore by demanding the derivative of h be equal to zero we have:

$$\begin{aligned}
\frac{dh(x)}{dx} = 0 \iff \\
\frac{2xe^{\frac{x^2}{2}} - (1+x^2)\, xe^{\frac{x^2}{2}}}{e^{x^2}} = 0 \iff \\
2xe^{\frac{x^2}{2}} - (1+x^2)\, xe^{\frac{x^2}{2}} = 0 \iff \\
x\, e^{\frac{x^2}{2}} \left(2 - 1 - x^2\right) = 0 \iff \\
x\left(1 - x^2\right) = 0 \iff \\
x \in \{-1,\ 0,\ 1\}
\end{aligned}$$

We observe that: $h(-1) = h(1) = \sqrt{\frac{2\pi}{e}}$

Additionally:

1. $\forall\ x < -1$ it holds:

$$h'(x) > 0 \iff$$

$$h(x) < h(-1)$$

2. $\forall\ x \in (-1, 0)$ it holds:

$$h'(x) < 0 \iff$$

$$h(x) < h(-1)$$

3. $\forall\ x \in (0, 1)$ it holds:

$$h'(x) > 0 \iff$$

$$h(x) < h(1)$$

4. $\forall\ x > 1$ it holds:

$$h'(x) < 0 \iff$$

$$h(x) < h(1)$$

Therefore $h(x) \leq h(1) = h(-1),\ \forall\ x \in \mathbb{R}$

Which means that $\max_{x \in \mathbb{R}} h(x) = h(\pm 1) = M$

- Inversion Sampling This sampling method is based upon the following supposition:

  Proposition 1. *Let X be a random variable with cdf F, let U be a r.v. s.t. $U \sim U(0,1)$. Let Y be a r.v. s.t. $Y \coloneqq F^{-1}(U)$,*
  where F^{-1} is the inverse of F.

  Then: $F^{-1}(U) \overset{d}{=} X$

  Proof. F has an inverse as it is weakly monotic and right continuous, as a cdf. The inverse is defined as: $F^{-1}(u) = \inf\{x \in \mathbb{R} : F(x) \geq u\}$

  It suffices to show that: $F_Y(x) = F_X(x), \forall\ x \in \mathbb{R}$, where F_X, F_Y, cdfs for X, Y respectivelly.

  $$\mathbb{P}[Y \leq x] = \mathbb{P}[F_X^{-1}(U) \leq x] \iff$$

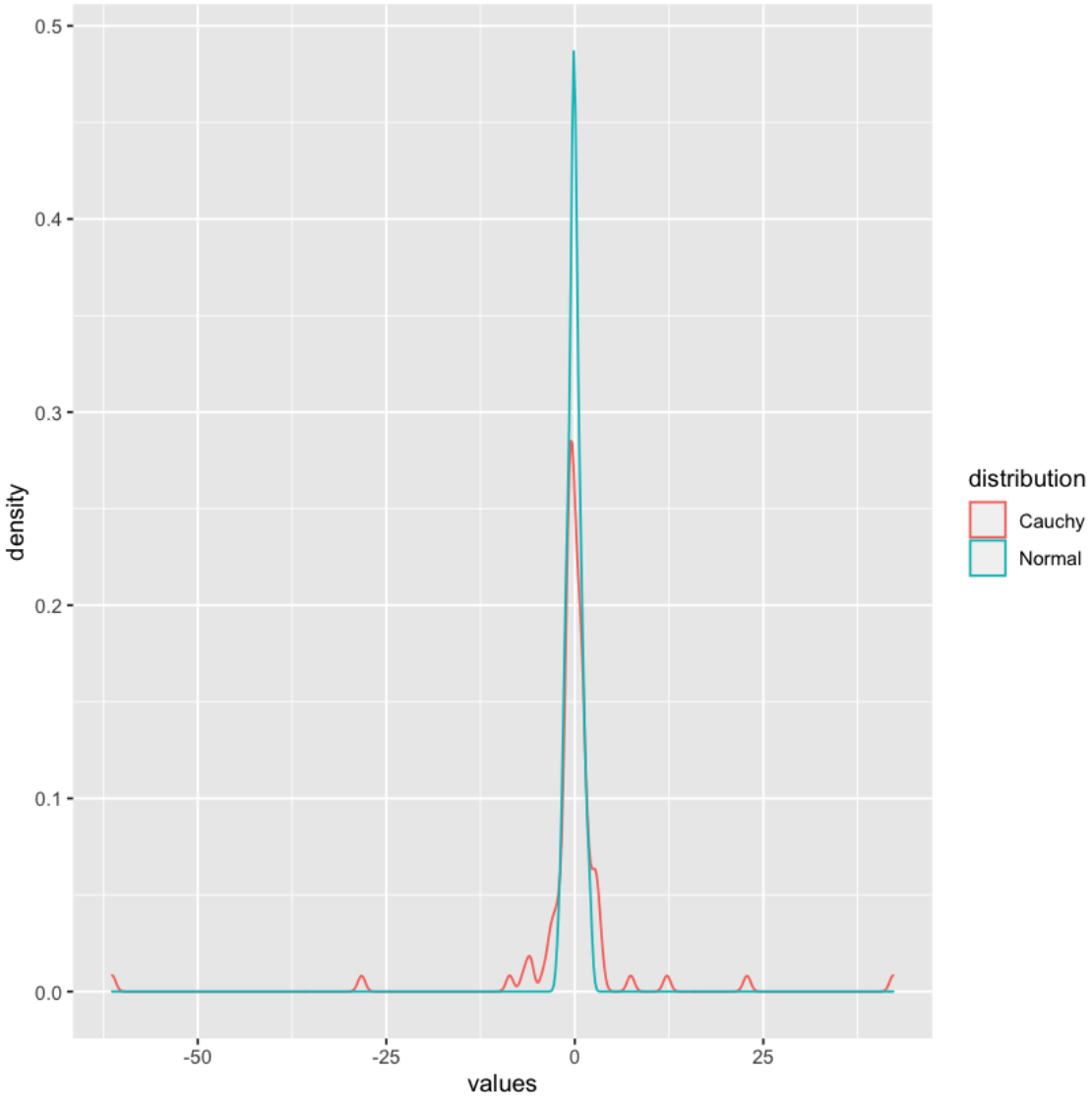
  $$\mathbb{P}[Y \leq x] = \mathbb{P}[U \leq F_X(x)] \overset{U \sim U(0,1)}{\iff}$$
  $$\mathbb{P}[Y \leq x] = F_X(x) \iff$$
  $$F_Y(x) = F_X(x)$$

  □

  What this proposition tells us, is that we can sample values from X by taking values from Y.
  In other words, we can instead sample values from a standardised uniform distribution and then apply the inverse cdf of X to acquire samples from X.

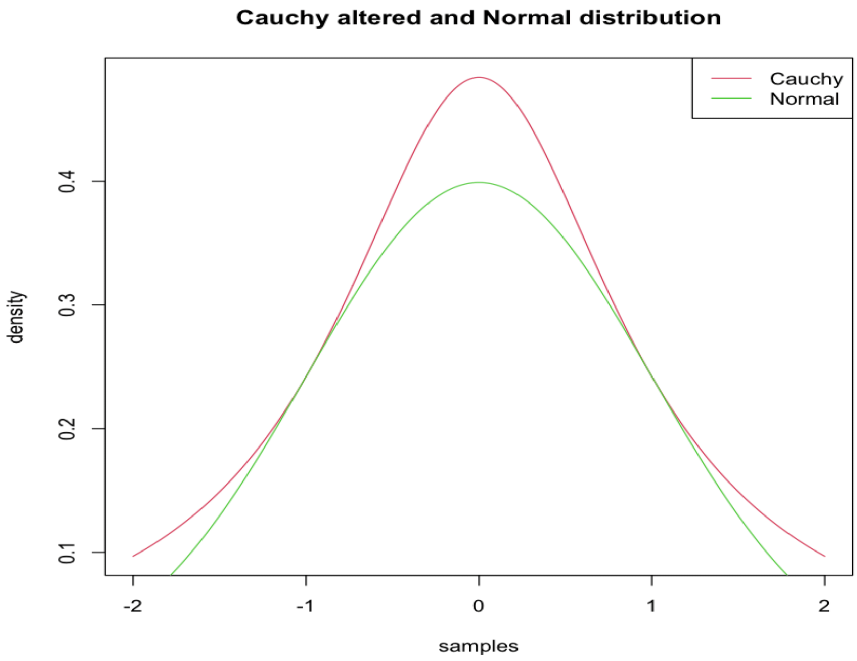
The code used to implement the explained sampling methods is given below:
Firstly, we plot the two distributions

```
normal_values = rnorm(100) # Sample from normal distribution
cauchy_values = rcauchy(100) # Sample from Cauchy distribution
dat = data.frame(distribution = factor(rep(c('Normal', 'Cauchy'), each = 100
# Store values in a data.frame
ggplot(dat, aes(x = values, colour = distribution)) + geom_density()
# Plot the two densities
```



We can see that for $M = 1$ the normal distribution's pdf is mainly above cauchy's distribution. Multiplying g with the theoretical M value will produce a graph in which $M * g(x) \geq f(x)\ \forall x \in \mathbb{R}$.

Indeed this can be seen by the following graph:



After that, we focus on the actual sampling methods, implemented below:

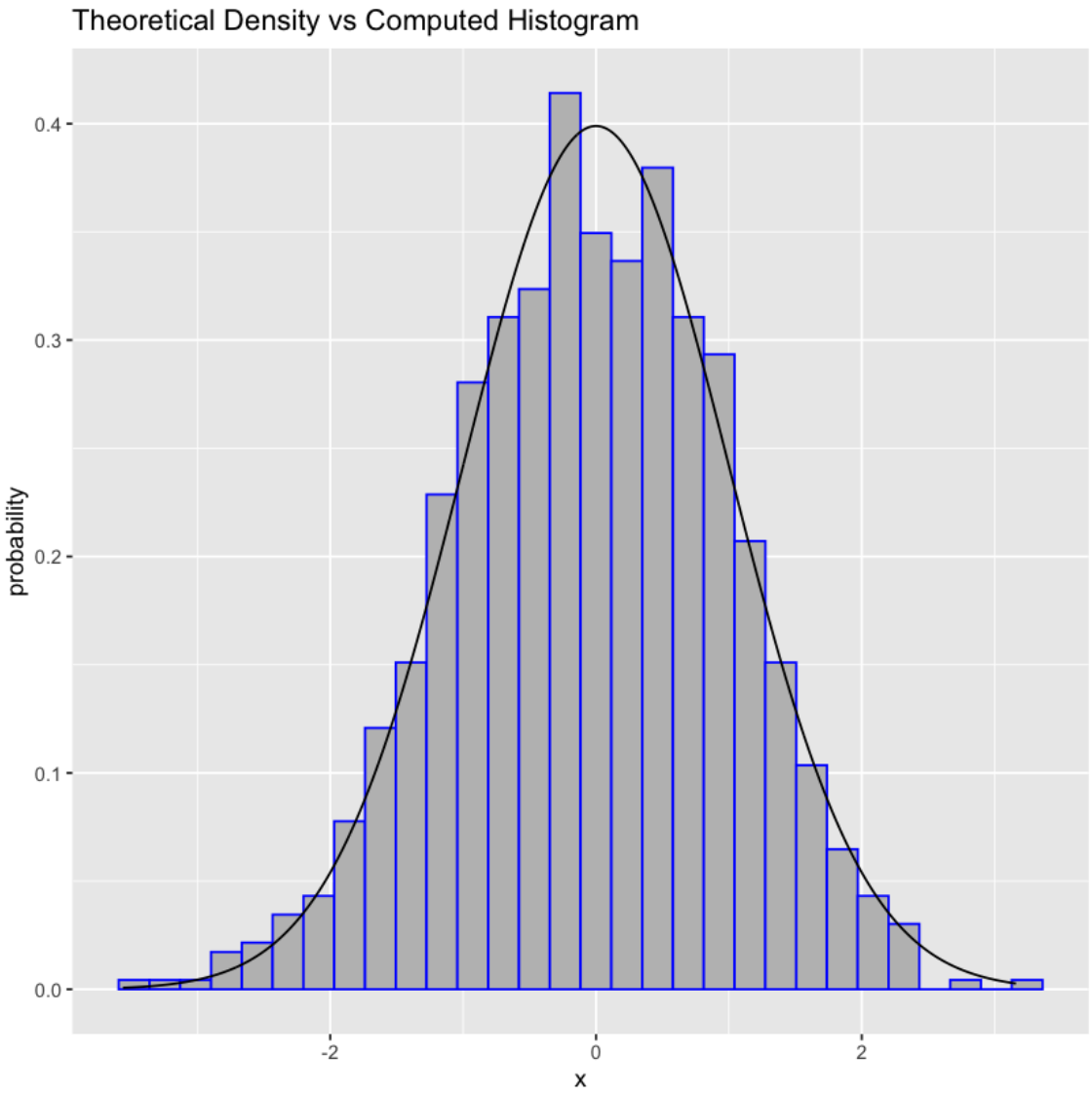
```
counter = 0 # Number of trials
acc_samples = c() # sequence of acepted values
while (length(acc_samples) < 1000){

counter = counter + 1
quant_value = runif(1) # Random value from standardized uniform
inv_sample = qcauchy(quant_value) # Use the quantile function to find x
temp = runif(1) # Random value from standardized uniform
if (dnorm(inv_sample) / (sqrt(pi*2) * dcauchy(inv_sample) / exp(0.5)) > temp
    # Check the acceptance condition
     acc_samples = append(acc_samples, inv_sample)
    # Add samples if condition holds
}
```

```
}

print((length(acc_samples) / counter)) # = 0.665779
# Computed probability of geometric
print(sqrt(exp(1)/(2 * pi))) # = 0.6577446
# Theoretical probability of geometric
print(mean(acc_samples)) # = 0.0532029
print(var(acc_samples)) # = 0.9945255
```

The results show that indeed the produced samples do approximate the mean and variance of the standardised normal distribution, as well as the probability of sucess for the geometric distribution.
Finally we show below the histogram produced by the sampled values asa long as the theoretical pdf.

```
ggplot(dat, aes(x = acc_samples)) + geom_histogram(aes(y = ..density..), col
# Plot the histogram from computed values along with the theoretical pdf
```

The figure produced is shown below.



As expected, the produced samples do indeed aproximate satisfiably the theoretical distribution.

1.2 1.b

In this exercise we shall use the Rao Blackwellization method in order to produce a better estimation of a desired r.v..
Firstly we present a brief reminder of the method:

Let X be a r.v., $g : D_g \to R_g, X \in D_g$ and δ an estimator of $g(X)$
It holds that:

$$\mathbb{V}_X[\delta] = \mathbb{E}_Y[\mathbb{V}_{X|Y}[\delta|Y]] + V_Y[\mathbb{E}_{X|Y}[\delta|Y]] \overset{\mathbb{V}\geq 0}{\Longrightarrow}$$
$$\mathbb{V}[\delta] \geq \mathbb{V}_Y[\mathbb{E}_{X|Y}[\delta|Y]]$$

Where Y r.v. such that X, Y dependent.
The above equality is called *total variance law* and it assures us that the estimator $\mathbb{E}_{X|Y}[\delta|Y]$ has smaller variance than the initial estimator.
Also it can be shown that $\mathbb{E}[\delta] = \mathbb{E}_{\mathbb{X}|\mathbb{Y}}[\delta|\mathbb{Y}]$, meaning that the two estimaators have the same bias.
The method of computing the second estimator to decrease our uncertainty (variance) in regards with the first estimator is called Rao-Blackwellization method.

In our case we want to estimate the mean value of an r.v. $X \sim \mathbb{NBIN}(r, p)$. A negative binomial distribution is used to express the number of successes before a specified number of failures (r) occurs. Our initial estimator is the sample mean. However we can improve the certainty of the estimator by using Rao-Blackwellization.
Specifically it holds that if:

$$Y \sim \Gamma(r, \frac{1-p}{p}) \implies X|Y = \lambda \sim Poisson(\lambda)$$

Therefore instead of using the sample mean to estimate the mean value of X we compute the average of $\bar{X}$ w.r.t. a Poisson distribution.
That is, we calculate :

$$\mathbb{E}_{X|Y}[\delta|Y = y] = \frac{1}{n}\mathbb{E}_{X|Y}[\sum_{i=1}^{n} X_i|Y = \lambda] \iff$$
$$\mathbb{E}_{X|Y}[\delta|Y = y] = \sum_{i=1}^{n} \frac{1}{n}\mathbb{E}_{X|Y}[X_i|Y = \lambda] \underset{call\ them\ X}{\overset{X_i i.i.d}{\iff}}$$
$$\mathbb{E}_{X|Y}[\delta|Y = y] = \mathbb{E}_{X|Y}[X|Y = \lambda] \overset{X|Y=\lambda \sim Poisson(\lambda)}{\iff}$$
$$\mathbb{E}_{X|Y}[\delta|Y = y] = \lambda$$

Let $\{(X_1, Y_1),\ (X_2, Y_2),\ (X_3, Y_3),\ \ldots,\ (X_n, Y_n)\}$ be a sample,
$X_i \stackrel{d}{=} X,\ Y_i \stackrel{d}{=} Y$
The Rao-Blackwellised version of our estimator is:

$$\delta^* = \mathbb{E}_{X|Y}[n^{-1}\sum_{i=1}^{n} X_i | Y = Y_i] = \sum_{i=1}^{n} \frac{Y_i}{n}$$

The above result is implemented by sampling values from the $\Gamma, \mathbb{NBINOM}$ distributions and then computing the corresponding estimation.

In order to validate that $\mathbb{E}[\delta^*] = \mathbb{E}[\delta],\ \mathbb{V}[\delta^*] \leq \mathbb{V}[\delta]$, we will take a plethora of samples each of which will produce a different value for $\delta,\ \delta*$.
Finally we shall approximate the mean and variance of each estimator thus:

$$\mathbb{E}[\delta] \approx \frac{\sum_{i=1}^{n} \delta_i}{n}$$

$$\mathbb{E}[\delta^*] \approx \frac{\sum_{i=1}^{n} \delta_i^*}{n}$$

$$\mathbb{V}[\delta] \approx \sqrt{\frac{\sum_{i=1}^{n} (\delta_i - \hat{\delta})^2}{(n-1)}}$$

$$\mathbb{V}[\delta^*] \approx \sqrt{\frac{\sum_{i=1}^{n} (\delta_i^* - \hat{\delta}^*)^2}{(n-1)}}$$

This is achieved by the following code:

```
    initial_estimators = c()
    rao_blackwell_estimators = c()

    for (i in 1 : 5000){
    # Take different sets of samples
    nsample_temp = rnbinom(5000, size = 1, prob = 0.5)
    # Individual negative binomial sample

    gsample_temp = rgamma(5000, shape = 1, rate = 1)
    # Individual gamma sample

    initial_estimators = append(initial_estimators, mean(nsample_temp))
    rao_blackwell_estimators = append(rao_blackwell_estimators, mean(psample_tem
    # Append each estimator to the vector of corresponding estimator values

}

mean(initial_estimators)
# Approximate mean of initial estimator
mean(rao_blackwell_estimators)
# Approximate mean of rao blackwell estimator


var(negative_samples)
# Approximate variance of initial estimator
var(gamma_samples)
# Approximate variance of rao blackwell estimator
```

1.3 1.c

In this part of the exercise we sample values from the r.v. $T = \frac{(\sum_{i=1}^{n} X_i)^2}{n}$, $X_i \sim U(0,1)$. After having collected 10^4 samples from T we find each mean value and variance.
Finally we compare the estimated mean with the theoretical mean.
Before presenting the code that achieves the above procedure, we calculate the theoretical mean value of T.

$$\mathbb{E}[T] = \mathbb{E}[n^{-1}(\sum_{i=1}^{n} X_i)^2] \iff$$

$$\mathbb{E}[T] = \frac{1}{n}\left(\mathbb{V}[\sum_{i=1}^{n}(X_i)] + \left(\mathbb{E}[\sum_{i=1}^{n} X_i]\right)^2\right) \underset{call\ them\ X}{\overset{X_i i.i.d}{\iff}}$$

$$\mathbb{E}[T] = \frac{1}{n}\left(\sum_{i=1}^{n}\mathbb{V}[(X)] + \left(\sum_{i=1}^{n}\mathbb{E}[X]\right)^2\right) \iff$$

$$\mathbb{E}[T] = \frac{1}{n}(\frac{n}{12} + \frac{n^2}{4}) \iff$$

$$\mathbb{E}[T] = (\frac{1}{12} + \frac{n}{4}) \iff$$

$$\mathbb{E}[T] \approx 20.083$$

At this point we present the code that implements the previously described process.

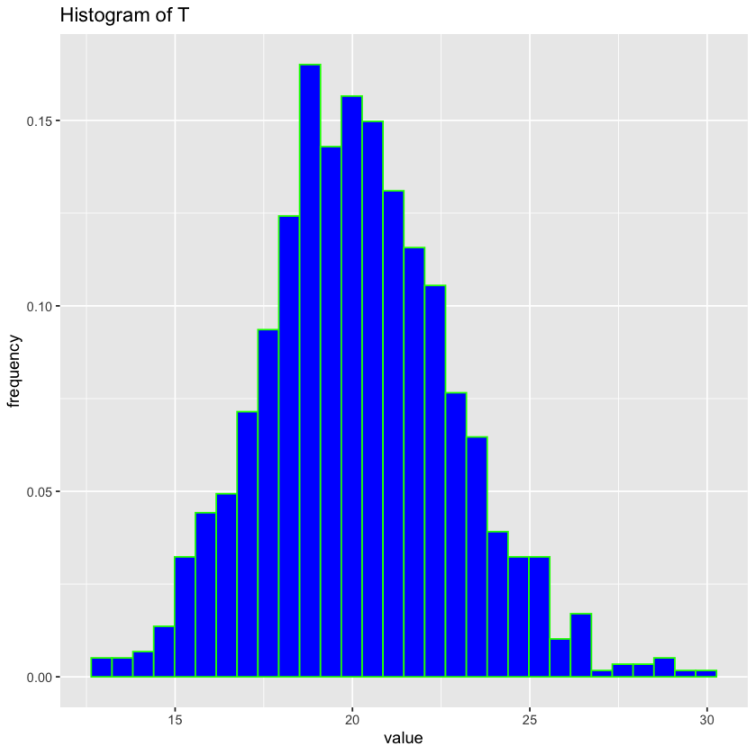
```
sample_means = c()

for (i in 1:10000){
    uni_sample = sum(runif(80))**2/ 80  # Take sample of T
    sample_means = append(sample_means, uni_sample)
    # append sample to samples vector
    }

estimated_mean = mean(sample_means)  # Calculate mean of T
estimated_sd = sd(sample_means)  # Calculate standard
deviation of T

real_mean = 20.083

print((estimated_mean - real_mean) / real_mean)
# percentage error of estimation
```

```
print(estimated_se)
```

The results are:
Percentage Error: 0.0047
Estimated Deviation: 2.5918
As we can see the estimation of the mean value does indeed converges to the real value.

Finally we present the histogram produced from the sampled values of T, as well as the code that implemented it.

```
dat_temp = data.frame(sample_means)
# Store samples to a data frame

(ggplot(dat_temp) + geom_histogram(
    aes(x = sample_means, y = ..density..),
    colour = 'green', fill = 'blue')
    + labs(title="Histogram of T", x = "value", y = "frequency"))
# Create density histogram for the stored values.
```



1.4 1.d

In this part of the exercise we shall estimate the quantity T of the previous step with the use of the Boostrap and Jackknife estimators.
Both methods produce a number of samples from the original dataset and return one final estimation of the desired parameter.

Let $(S_1, S_2, \ldots, S_k, k \in \mathbb{N})$ samples and
$T(S_1) = \widehat{T}_1,\ T(S_2) = \widehat{T}_2, \ldots,\ T(S_k) = \widehat{T}_k$ the corresponding estimates of T.

The Bootstrap method relies on sampling from our original sample multiple times (B) with repetition n values, where n the sample length of our original data, using the empirical distribution of our data.

$$F_n(x) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{x=x_i\}}}{n}$$

Our final estimation is given as $\widehat{T}_b = \frac{\sum_{i=1}^{B} T_i}{B}$

The standard error of the bootstrap method is given by:

$$se(\widehat{T}_b) = \sqrt{\frac{1}{B-1} \sum_{i=1}^{B} (\widehat{T}_i - \widehat{T}_b)}$$

The Jackknife method creates n samples of $n-1$ data points by removing one observation for every sample.

Our final estimation is given as $\widehat{T}_J = \widehat{T} + (n-1)\frac{\sum_{i=1}^{n} \widehat{T}_i}{n}$, where $\widehat{T}$ is the initial estimation given by using the original sample, and $\widehat{T}_i$ the estimation of the i-th sample.
The standard error of the Jackknife method is given by:

$$se(\widehat{T}_J) = \sqrt{\frac{n-1}{n} \sum_{i=1}^{n} (\widehat{T}_i - \bar{\widehat{T}}_{(.)})},\ \bar{T}_{(.)} \coloneqq \frac{1}{n} \sum_{i=1}^{n} \widehat{T}_i$$

Below is the code that calculates the standard error of the estimation of T of these two methods.

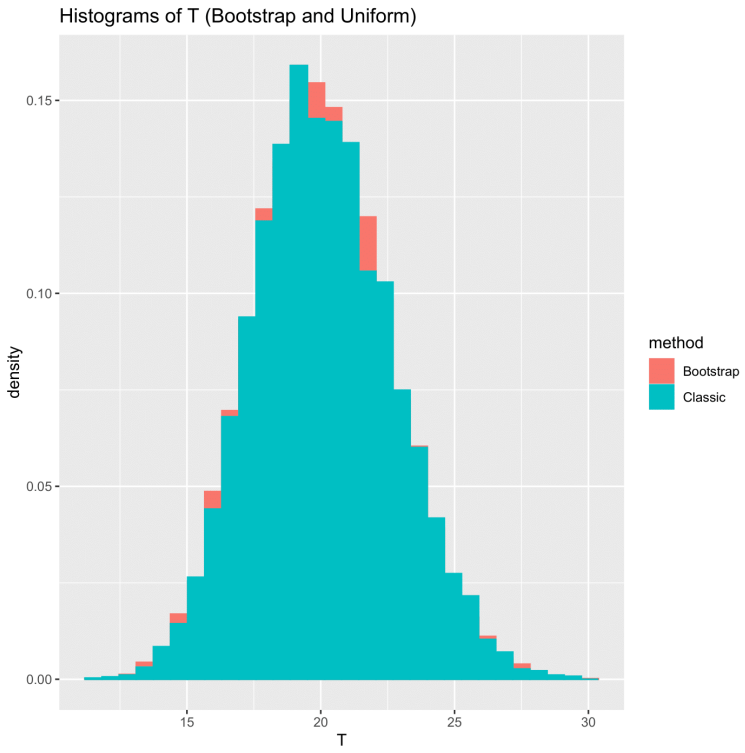
```
filename = file.choose('data1.rds')
file_rds = readRDS(filename)
# Read the data

# Bootstrap
samples = c()

for (i in 1: 10000){
    sample_temp = sample(file_rds , 80, replace=TRUE)
    # Generate a sample

    sample_est = sum(sample_temp) ** 2 / 80
    # Compute sample estimation

    estimations_b = append(samples , sample_est)
    # Collect the samples
}


bootstrap_est = mean(estimations_b)
bootstrap_se = sqrt(sum((estimations_b - bootstrap_est)^2)/ (B - 1))
print(bootstrap_se)
print((bootstrap_est - 20.083)/ 20.083)
# Print the standard error

#jackknife

samples_jk = c()
for (i in 1:80){
    jk_sample = file_rds[-i]
    # Remove an observation and store the new sample

    sample_est = sum(jk_sample)**2/79
    # Compute sample estimation

    estimations_jk= append(samples_jk , sample_est)
    # Collect samples
    }

mean_est = mean(estimations_jk)
# Compute mean of estimations
```

```
jack_knife_se = sqrt(79/80 * sum((estimations_jk - mean_est) ** 2))

print(jack_knife_se)
```

The results are:
Bootstrap standard error : 2.5473
Bootstrap percentage error: -0.002069
Jacknife standard error : 2.5470

At this point we are given the information that our dataset contains values from the standard uniform distribution.
The percentage error is calculated between the final bootstrap estimation and the theoretical value of the mean of T, given that $X_i \sim U(0,1)$. We note that the botstrap method seems to have smaller percentage error than the original uniform sampling method.

We show the histograms of the samples of T acquired by the bootstrap method and the uniform sampling of the previous task.



As we can see the two methods produce similar histograms as it was indeed expected.

2 Exercise 2

2.1 2.a

In this exercise we shall aproximate the density of the dataset "Old Faithful geyser data" by using a mixture of epechnikov kernels.
Specifically the estimated pdf is given by:

$$\hat{f}(x) = \frac{\sum_{j=1}^{n} K_{epan}\left(\frac{x-x_j}{h}\right)}{n\ h}$$

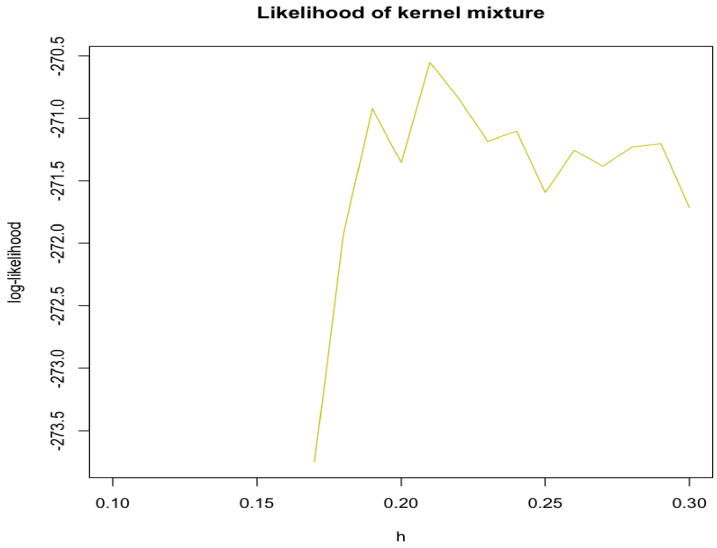
In order to aproximate the optimal h we use the following functions (one for each data point in our sample).

$$\hat{f}_{h,-i}(x) = \frac{\sum_{j=1,j\neq i}^{n} K_{epan}\left(\frac{x-x_j}{h}\right)}{(n-1)\ h}$$

We now compute the maximum likelihood function ($L(h,i)$), using the formula: $L(h,i) = \prod_{i=1}^{n} f_{h,-i}(x_i)$

The maximum h is found approximately by plotting values of $\log(L)$ as function of h. The code and graph for this estimation is given below:

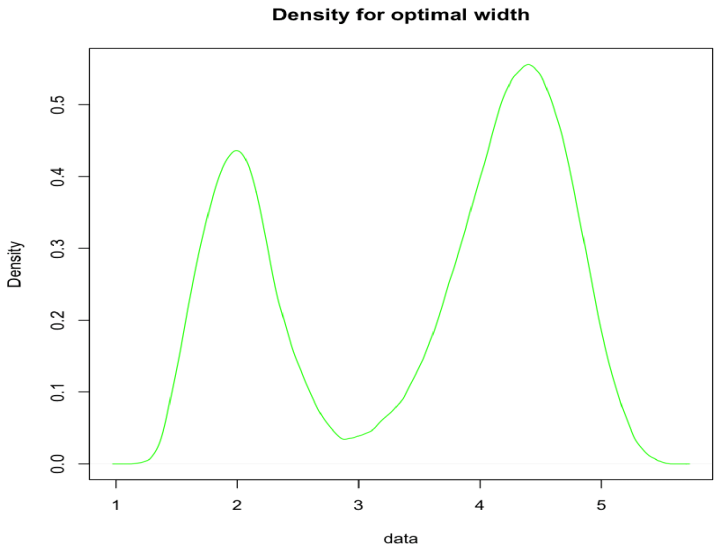
```
l = function(h){

return(sum(log(vf_hat(data, h=h))))
# Epanechnikov log likelihood as a funtion of h
}

h = seq(from = 0.1, to = 0.3, by = 0.01)
# Create a sequence of h values

l_vec = Vectorize(l)
# Vectorize the functionm so it can take the whole sequence at once

l_val = l_vec(h)
# Store the vectorized functions output

plot(h, l_val, type='l', col='#cbcb09', xlab='h', ylab='log-likelihood', mai
# Plot the log likelihood as a funtion of h
```



From the graph above we approximate the optimal width to be 0.21.

Using this approximated value as well as the code below we produce an estimation of the pdf using the R function density.

```
data = faithful$eruptions
# Collect data
estimated_density = density(data, kernel = "epanechnikov", bw = 0.21)
# Compute density using the density function.

plot(estimated_density, col='green', main="Density for optimal width", xlab=
# Plot the result.
```



2.2 2.b

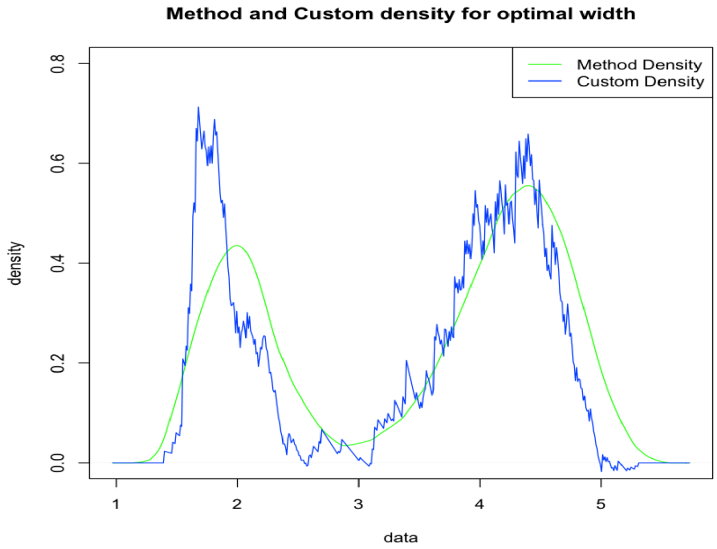
At this section we plot the custom density function using the optimal width at the first step and compare the results with the pdf found using the density method.
The function used is given below:

```
f_epanechnikov = function(x)
{
data = faithful$eruptions
h_opt = 0.210
n = length(data)

centered_data = (x-data)/h_opt
filter = abs(centered_data)<1
f = 0.75 * sum(1 - centered_data[filter]*2)

f =  f / (h_opt * n)
return(f)
}
```

Before plotting the function, we must first transform it so that it can accept a sequence of data points and return the correspoinding sequence of results. This can be achieved using the "Vectorize" command.

```
vf_hat = Vectorize(f_epanechnikov)

custom_density = vf_hat(estimated_density$x) # compute estimated pdf values

plot(estimated_density, col='green',
main="Density for optimal width",
xlab="data", ylab="density")

lines(estimated_density$x, custom_density, col="#0059ff",
main="Estimated density", xlab="data", ylab="density", lty=1, type= "l")
```

The corresponding graph is shown below:



As we can observe there is a considerable difference between the density created by the density method and the on created customly. This result is expected as the density function uses further transfomations to smooth and

otherwise process the returned density.

2.3 2.c

Using the R command "integrate" with inputs the vectorized estimated function and the lower, upper limit of the integral we have:

```
integrate(vf_hat,lower = 3.5, upper = Inf)
```

The result is: 0.6120.

2.4 2.d

In order to sample values from the estimated pdf function ($\hat{f}$) we shall use the r.vs $X \sim U(\{x_1, x_2, \ldots, x_n\}), x_i$ *observation*, $Y \sim Epanechnikov(\mu = 0, h = 1)$ Specifically the following proposition holds:

Proposition 2. *Let* $X \sim U(\{x_1, x_2, \ldots, x_n\}), x_i \in \mathbb{R}, i = 1, 2, \ldots, n,$
$Y \sim Epanechnikov(\mu = 0, h = 1)$ *random variables.*
Let $Z = X + h * Y, h \in \mathbb{R}$. *Then* Z *has pdf* f

Proof. First we show a relation between the cumulative probability functions.

$$
\begin{aligned}
&F_Z(x) = \mathbb{P}[Z \leq x] \iff \\
&F_Z(x) = \mathbb{P}[X + h * Y \leq x] \iff \\
&F_Z(x) = \sum_{i=1}^{n} \mathbb{P}[x_i + h * Y \leq x | X = x_i] \mathbb{P}[X = x_i] \iff \\
&F_Z(x) = \sum_{i=1}^{n} \frac{\mathbb{P}[Y \leq \frac{x - x_i}{h} | X = x_i]}{n} \iff \\
&F_Z(x) = \sum_{i=1}^{n} \frac{F_{Y|X=x_i}(\frac{x-x_i}{h})}{n} \overset{Y,X independent}{\iff} \\
&F_Z(x) = \sum_{i=1}^{n} \frac{F_Y(\frac{x-x_i}{h})}{n}
\end{aligned}
$$

By taking the derivative of the above expression we have:

$$
\begin{aligned}
\frac{dF_Z(x)}{dx} = \frac{d}{dx} \sum_{i=1}^{n} \frac{F_Y(\frac{x-x_i}{h})}{n} &\iff \\
f_Z(x) = \sum_{i=1}^{n} \frac{f_Y(\frac{x-x_i}{h})}{nh} &\iff \\
f_Z = f &
\end{aligned}
$$

□

Below we show the code written in R that implements the aforementioned sampling.

```
data = faithful$eruptions

center = data[sample(1:length(data), 250, replace=TRUE)]
epan_sample = repan(250, mu = 0, r = 1) # Sample values from epanechnikov

h_opt = 0.204 # calculated previously
final_sample = h_opt * epan_sample + center

condition = final_sample >= 3.5
probability = length(final_sample[condition])/ length(final_sample)
print(probability)
```

The last two lines are used to recompute the probabilty $\mathbb{P}[Z \geq 3.5]$
The result is 0.616 which is similar with the probability found using the integrate command above.
We note here that by increasing the number of generated samples the difference between the two results tends to zero.

3 Exercise 3

3.1 3.1

In this first task it is known to us to which distribution each data point belongs. Therefore the required values can easily be computed by using the maximum likelihood estimation. Specifically we maximize:

$$f_1(x|\lambda_1) = \frac{\lambda_1^x \exp^{-\lambda_1}}{x!},\ f_2(x|\lambda_2) = \frac{\lambda_2^x \exp^{-\lambda_2}}{x!}$$

Given that both distributioons are poisson with a diffferent shape parameter, we shall show the maximization process only for one of them. So we have:

$$L(\vec{x}|\lambda_1) \coloneqq f_1(\vec{x}|\lambda_1) \overset{x_i i.i.d.}{\iff}$$

$$L(\vec{x}|\lambda_1) = \prod_{i=1}^{4} f_1(x_i|\lambda_1) \iff$$

$$l(\vec{x}|\lambda_1) \coloneqq \log(L(\vec{x}|\lambda_1)) = \sum_{i=1}^{4} \log(f_1(x_i|\lambda_1)) \iff$$

$$l(\vec{x}|\lambda_1) = \sum_{i=1}^{4} (x_i \log(\lambda_1) - \lambda_1 - \log(x_i!)) \iff$$

$$\frac{\partial l(\vec{x}|\lambda_1)}{\partial \lambda_1} = 0 \iff$$

$$\sum_{i=1}^{4} \left(\frac{x_i}{\lambda_1} - 1 \right) = 0 \iff$$

$$\sum_{i=1}^{4} \left(\frac{x_i}{\lambda_1} \right) = 4 \iff$$

$$\lambda_1 = \frac{\sum_{i=1}^{4} x_i}{4} \iff$$

$$\widehat{\lambda_1} = \bar{x}$$

So finally we have $\widehat{\lambda_1} = \frac{3+2}{2} = 2.5,\ \widehat{\lambda_2} = \frac{7+9}{2} = 8$

3.2 3.2

At this point we assume that the subcategory for each data point is unknown. For this reason, we implement the EM algorithm in order to find the distibution from which each point was generated. The EM algorithm comprises of the two following steps:

- Expectation step:
  In this step we estimate the value of our unobserved data i.e. the class to which each data point belongs to, w.r.t. the value of the parameters during this iteration. In other words we compute the expected value: $\mathbb{E}[l(\vec{X}, Z|\ \vec{\theta})]$ where $Z = (\vec{Z_1}, \vec{Z_2}, \vec{Z_3}, \vec{Z_4})$ is a matrix of vectors $(\vec{Z_i})$, where $\vec{Z_i} \in \{0,1\}^2$ denotes the class for each observation and $\vec{X}$ the observations.

  Let $\vec{V_i} \coloneqq (\mathbb{P}[x_i, \lambda_1], \mathbb{P}[x_i, \lambda_2])\ , i = 1, 2, 3, 4$

The calculation is shown below:

$$L(\vec{X}, Z|\vec{\theta}) = \prod_{i=1}^{4} \vec{V_i}\vec{Z_i} \iff$$

$$L(\vec{X}, Z|\vec{\theta}) = \prod_{i=1}^{4} \sum_{j=1}^{2} (z_{ij}\ \pi_j\ \mathbb{P}[x_i|\lambda_j]) \iff$$

$$L(\vec{X}, Z|\vec{\theta}) = \prod_{i=1}^{4} \left(\sum_{j=1}^{2} \left(\frac{z_{ij}\ \pi_j \exp^{-\lambda_j}\ \lambda_j^{x_i}}{x_i!} \right) \right) \iff$$

$$l(\vec{X}, Z|\vec{\theta}) = \log(L(\vec{X}, \vec{Z}|\vec{\theta})) = \sum_{i=1}^{4} \log \left(\sum_{j=1}^{2} \left(\frac{z_{ij}\pi_j \exp^{-\lambda_j}\ \lambda_j^{x_i}}{x_i!} \right) \right) \quad (1)$$

Where z_{ij} is a bernouli random variable that expresses whether the i-th observation belongs to the j-th class.

Remark 3. *We observe that:*
$L(X_i|\vec{\theta}) = L(X_i, \vec{Z_i} = (1,0)|\ \vec{\theta}) + L(X_i, \vec{Z_i} = (0,1)|\ \vec{\theta}),\ i = 1,2,3,4$

Lemma 4. *Jensen's Inequality*
Let $g : D_g \to R_g$ *be a concave function*
Let also $a_k > 0, x_k \in D_g,\ k = 1,2,\ldots,N,\ N \in \mathbb{N}$
Then:

$$g(\sum_{k=1}^{N} a_k\ x_k) \geq \sum_{k=1}^{N} a_k g\,(x_k)$$

In our case we have the concave function log so :

$$(1) \implies l(\vec{X}, \vec{Z}|\vec{\theta}) \geq \sum_{i=1}^{4}\sum_{k=1}^{2} z_{ik} \log \left(\frac{\pi_k \exp^{-\lambda_k}\ \lambda_k^{x_i}}{x_i!} \right) \iff$$

$$l(\vec{X}, \vec{Z}|\vec{\theta}) \geq \sum_{i=1}^{4}\sum_{k=1}^{2} z_{ik} \left[\log \pi_k - \lambda_k + x_i \log(\lambda_k) - log(x_i!)\right] \quad (2)$$

For reasons explained later on we shall instead compute the expected value of the quantity on the right side of (2)

$$\mathbb{E}[l(\vec{X}, Z|\ \vec{\theta})] \geq \mathbb{E}[\sum_{i=1}^{4}\sum_{k=1}^{2} z_{ik} \left[\log \pi_k - \lambda_k + x_i \log(\lambda_k) - log(x_i!)\right]] \iff$$

$$\mathbb{E}[l(\vec{X}, Z|\ \vec{\theta})] \geq \sum_{i=1}^{4}\sum_{k=1}^{2} \mathbb{E}[z_{ik}] \left(\log \pi_k - \lambda_k + x_i \log(\lambda_k) - log(x_i!)\right) =: Q(\vec{\theta})$$

Where:

$$\mathbb{E}[z_{ik}] \coloneqq \mathbb{P}[z_{ik} = 1 | x_i] \iff$$
$$\mathbb{E}[z_{ik}] = \frac{\mathbb{P}[z_{ik} = 1,\ x_i]}{\mathbb{P}[x_i]} \iff$$

$$\mathbb{E}[z_{ik}] = \frac{\pi_k \mathbb{P}[x_i | z_{ik} = 1]}{\sum_{m=1}^{2} \pi_m \mathbb{P}[x_i | z_{im} = 1]} \in (0, 1)$$

Where $\mathbb{E}[z_{ik}]$ is equal to the probability, that observation i is generated from class j, provided that we see observation i.
It holds that:

$$\frac{\partial \mathbb{E}[l(\vec{X}, Z | \vec{\theta})]}{\partial \lambda_k} = 0 \iff \frac{\partial Q(\vec{\theta})}{\partial \lambda_k} = 0,$$

$$\vec{\theta} = (\lambda_1, \lambda_2, \ldots, \lambda_k, \pi_1, \pi_2, \ldots, \pi_k)$$

It suffices to show that:

$$\frac{\partial \mathbb{E}[l(X_i, \vec{Z_i} | \vec{\theta})]}{\partial \lambda_k} = 0 \iff \sum_{j=1}^{2} \mathbb{E}[z_{ij}] \ \frac{\partial \log\left(\pi_j \ \mathbb{P}[x_i | \lambda_j]\right)}{\partial \lambda_k} = 0, \forall\ i \in \ \{1, 2, 3, 4\}$$

Indeed:

$$\frac{\partial \mathbb{E}[l(X_i, \vec{Z}_i|\vec{\theta})]}{\partial \lambda_k} = 0 \overset{l \in C^1}{\iff}$$

$$\mathbb{E}\left[\frac{\partial l(X_i, \vec{Z}_i|\vec{\theta})}{\partial \lambda_k}\right] = 0 \iff$$

$$\mathbb{E}\left[\frac{\partial \log\left(\sum_{j=1}^{2} \pi_j z_{ij}\ \mathbb{P}[x_i|\lambda_j]\right)}{\partial \lambda_k}\right] = 0 \iff$$

$$\mathbb{E}\left[\frac{z_{ik}\pi_k}{\sum_{j=1}^{2} z_{ij}\pi_j \mathbb{P}[x_i|\lambda_j]} \frac{\partial \mathbb{P}[x_i|\lambda_k]}{\partial \lambda_k}\right] = 0 \iff$$

$$\frac{\pi_k\ \mathbb{E}[z_{ik}]\ \partial \mathbb{P}[x_i|\lambda_k]}{\partial \lambda_k} = 0$$

On the other hand:

$$\frac{\partial Q(\vec{\theta})}{\partial \lambda_k} = 0 \iff$$

$$\sum_{j=1}^{2} \mathbb{E}[z_{ij}]\ \frac{\partial \log\left(\pi_j\ \mathbb{P}[x_i|\lambda_j]\right)}{\partial \lambda_k} = 0 \iff$$

$$\frac{\mathbb{E}[z_{ik}]\pi_k}{\pi_k \mathbb{P}[x_i|\lambda_k]} \frac{\partial \mathbb{P}[x_i|\lambda_k]}{\partial \lambda_k} = 0 \iff$$

$$\frac{\mathbb{E}[z_{ik}]\ \pi_k\ \partial \mathbb{P}[x_i|\lambda_k]}{\partial \lambda_k} = 0$$

[1] Therefore for simplicity of computations we choose to differentiate the right part of Jensen's inequality. For this reason in the expectation step above we also took the expectation of the right side of (2).

- Maximization Step:

  We first differentiate Q with respect to λ_k and demand the value be equal to 0.

[1] $\mathbb{E}[z_{ij}]$ is fixed, as it is estimated during the expectation step.

$$\frac{\partial Q}{\partial \lambda_k} = 0 \iff$$

$$\sum_{i=1}^{4} \frac{\partial \left(\sum_{j=1}^{2} \mathbb{E}[z_{ij}] \log\left(\pi_j \ \mathbb{P}[x_i|\lambda_j]\right) \right)}{\partial \lambda_k} = 0 \iff$$

$$\sum_{i=1}^{4} \sum_{j=1}^{2} \mathbb{E}[z_{ij}] \frac{\partial \left(\log\left(\pi_j \ \mathbb{P}[x_i|\lambda_j]\right) \right)}{\partial \lambda_k} = 0 \iff$$

$$\sum_{i=1}^{4} \mathbb{E}[z_{ik}] \frac{\partial \left(\log \pi_k - \lambda_k + x_i \log(\lambda_k) - log(x_i!) \right)}{\partial \lambda_k} = 0 \iff$$

$$\sum_{i=1}^{4} \mathbb{E}[z_{ik}] \left(\frac{x_i}{\lambda_k} - 1 \right) = 0 \iff$$

$$\widehat{\lambda_k} = \frac{\sum_{i=1}^{4} \mathbb{E}[z_{ik}] x_i}{\sum_{i=1}^{4} \mathbb{E}[z_{ik}]}$$

Secondly we want to estimate the parameter $\pi_k, k = 1, 2, 3, 4$.
The maxizimization problem for this parameter is written as such:

$\max_{\pi_k \in [0,1]} Q(\vec{\theta}), \sum_{k=1}^{2} \pi_k = 1.$

Using lagrange multipliers the above problem can be rewritten as such:
$\max_{\pi_k \in [0,1]} \left(Q(\vec{\theta}) - \delta(\sum_{k=1}^{2} \pi_k - 1) \right)$

By demanding the derivative of $G(\vec{\theta}) \coloneqq Q(\vec{\theta}) - \delta(\sum_{k=1}^{2} \pi_k - 1)$ w.r.t. π_k is equal to 0, we have:

$$\frac{\partial G(\vec{\theta})}{\partial \pi_k} = 0 \iff$$

$$\frac{\partial Q(\vec{\theta})}{\partial \pi_k} - \delta = 0 \iff$$

$$\sum_{i=1}^{4} \mathbb{E}[z_{ik}] \frac{\partial \left(\log \pi_k - \lambda_k + x_i \log(\lambda_k) - log(x_i!) \right)}{\partial \pi_k} - \delta = 0 \iff$$

$$\sum_{i=1}^{4} \frac{\mathbb{P}[z_{ik} = 1|x_i]}{\pi_k} = \delta \qquad (*)$$

Before finding an estimation for the priors, we need to find δ

$$
\begin{aligned}
(*) \implies \pi_k \delta &= \sum_{i=1}^{4} \mathbb{P}[z_{ik} = 1 | x_i] \iff \\
\sum_{k=1}^{2} \pi_k \delta &= \sum_{i=1}^{4} \sum_{k=1}^{2} \mathbb{P}[z_{ik} = 1 | x_i] \iff \\
\delta &= 4
\end{aligned}
$$

Finally we can estimate the priors as:

$$\widehat{\pi_k} = \sum_{i=1}^{4} \frac{\mathbb{P}[z_{ik} = 1 | x_i]}{4}$$

The above is implemented using the above R code:

```
X = c(2, 7, 3, 9)
lambda_1 = runif(1, 0, 20)
lambda_2 = runif(1, 0, 20)
priors = runif(1, 0, 1)

criterion = TRUE
i = 0
while (criterion){

i = i + 1
# E_step
prob_distr_1 = dpois(X, lambda = lambda_1)
prob_distr_2 = dpois(X, lambda = lambda_2)

reg_sum = priors[1] * prob_distr_1 + priors[2] * prob_distr_2

prob_distr_1 = (priors[1] * prob_distr_1) /reg_sum
prob_distr_2 = (priors[2] * prob_distr_2) / reg_sum

# M_Step
lambda_1_previous = lambda_1
lambda_2_previous = lambda_2

lambda_1 = sum(prob_distr_1 * X) / sum(prob_distr_1)

lambda_2 = sum(prob_distr_2 * X)/ sum(prob_distr_2)
priors = c(sum(prob_distr_1)/ 4, sum(prob_distr_2)/4)

criterion = ((lambda_1 - lambda_1_previous) ^ 2 + (lambda_2 - lambda_2_previ

}

print(lambda_1)
print(lambda_2)
print(priors)
```

The results are $\lambda_1 = 2.6832, \lambda_2 = 7.4018, \pi = (0.4560, 0.5440)$

4 Exercise 4

4.1 4.a

In this exercise we explore different lineal models and related techniques in order to eventually find the best model (for different metrics) for our data. In the first

section we shall fit all possible linears models given our explanatory variables and select the one with the best BIC score.
BIC is used to minimize the number of parameters of our model in an attempt to prevent overfitting.
While a models accuracy on the training data increases with the number of parameters (i.e. exploratory variables), the same is not true about the test set. Therefore metrics such as BIC and AIC are used in order to prevent this phenomenon from happening by using a penalty factor.
Specifically BIC is defined as: $BIC = -\ln(\widehat{L}) + k \ln n$

Where $\widehat{L}$ is the estimated value of the likelihood function after maximization w.r.t the distribution parameters, k the number of variables used and n the number of data points in our sample.
We can see that by minimizing the BIC we seek to increase the likelihood of our data while simultaneously have a small number of exploratory variables.
The penalty mentioned above is given by the term $k \ln n$ since it increases (we want to minimize) the value of BIC.

Below is the code used to find the best of all possible linear models given the set of explanatory variables.

```
find_model_BIC = function(){

    best_model = NULL

    for (i in 1:10){

        z = (combinations(10, i))
        # Find all possible variable combinations of i elements from 10 total
        rows = nrow(z)
        # Number of rows in the combinations library is
        # equal to the number combinations
        for (j in 1 : rows){

            model = lm(y ~ x[, c(z[j, ])])
            # Fit the model
            if (is.null(best_model)){

                best_model = model
                # Initialize best_model variable
            }else{
                    if (BIC(model) < BIC(best_model)){
                        best_model = model
                        # Change best model in case of smaller BIC score
                    }
            }
```

```
        }

}
return(best_model)
}

m1 = find_model_BIC()
names(m1$coefficients) = c("(Intercept)", "sex", "bmi", "map", "hdl", "ltg")
# Change coefficient names for display purposes.
```

The last line shows specifically which variables have been selected as the most suitable using the BIC metric. Furthermore the model weigths corresponding to each variable are:

M1 Model Weights	
Variable Name	Weight
Intercept	152.1334
sex	235.7756
bmi	523.5623
map	326.2358
hdl	289.1169
ltg	474.2918

4.2 4.b

After looking for the best linear model we shall slightly alternate our model of choice by using a regularization method. Specifically we shall use the l_1 regularization to fit the various models and select the one which minimizes the augmented mse loss function. The l_1 or Lasso regression model weights are given by minimizing the following loss function w.r.t. the weights.

$L(\vec{w}) = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{n} w_j \, x_{ij} \right)^2 + \lambda \sum_{j=1}^{n} |w_j|$ The value of λ will be computed using cross validation. Specifically the cross validation method will create a number of samples k of approximately the same size. Each time $k-1$ of them will be merged and used as the training set and one will be used as the test set. This way there will be n computed losses one for each different test set used. The value of the mean square error will be computed as the mean of these n losses.
After applying this method for a number of different λ values we select the one that minimizes the average MSE or CV-MSE.

The procedure explained above will be done internally by the R library glmnet and specifically by the function cv.glmnet.

This function will return the weights for the λ value that minimizes CV-MSE as well as the weights for the value whose corresponding CV-MSE is one standard deviation away from the optimal λ's CV-MSE.

The code achieving this is given below:

```
cv = cv.glmnet(x, y, type.measure = "mse")
# Find the two lambda's

m2 = coef(cv, s = "lambda.min")
# Store model for lambda minimizing CV-MSE.

m3 = coef(cv, s = "lambda.1se")
# Store model for lambda 1 sd away from minimum CV-MSE. Result is a sparse m

m2 = as.matrix(m2)
m3 = as.matrix(m3)
# Transform sprase matrices to matrices

m2 = m2[t(m2) != 0, ]
m3 = m3[t(m3) != 0, ]
# Keep only non zero
```

The model weights as well as the repsective exploratory variables are given below:

Model Weights		
Model	Variable Name	Weight
M1	Intercept	152.1334
M1	sex	235.7756
M1	bmi	523.5623
M1	map	326.2358
M1	hdl	289.1169
M1	ltg	474.2918
M2	Intercept	152.1335
M2	sex	-199.6136
M2	bmi	522.3339
M2	map	298.1229
M2	tc	-107.2595
M2	hdl	-222.5078
M2	tch	3.4363
M2	ltg	515.3141
M2	glu	55.4275
M3	Intercept	152.1335
M3	sex	-51.3350
M3	bmi	509.5096
M3	map	220.6414
M3	hdl	-152.1941
M3	ltg	447.2292

Finally the models found above are retrained using five fold cross validation (split to five equal parts of the dataset) and root mean square loss for within fold evaluating.
The code is given below:

```
diabetes = diabetes[sample.int(length(y), length(y)),]
# Shufffle the data

folds = cut(seq(from = 1, to = length(y)),breaks=5,labels=FALSE)
# Create the 5 folds.

m1_error = c()
m2_error = c()
m3_error = c()

for (i in 1:5){

    train_folds = diabetes[folds != i, ]
```

```
    # Create the train set

    test_folds = diabetes[folds == i, ]
    # Create the test set

    m1_train = train_folds$x[, names(coef(m1))[-1]]
    m2_train = train_folds$x[, names(m2[t(m2) != 0, ])[-1]]
    m3_train = train_folds$x[, names(m3[t(m3) != 0, ])[-1]]
    # Only keep the variables that each model uses.

    m1_meta = lm(train_folds$y ~., m1_train)
    m2_meta = lm(train_folds$y ~., m2_train)
    m3_meta = lm(train_folds$y ~., m3_train)
    # Fit the linear model

    y_true = test_folds$y

    m1_test = test_folds$x[, names(coef(m1))[-1]]
    m2_test = test_folds$x[, names(m2[t(m2) != 0, ])[-1]]
    m3_test = test_folds$x[, names(m3[t(m3) != 0, ])[-1]]
    # Only keep the variables that each model uses.

    m1_pred = predict(m1_meta, m1_test)
    m2_pred = predict(m2_meta, m2_test)
    m3_pred = predict(m3_meta, m3_test)
    # Predict test labels

    rmse_m1 = rmse(y_true, m1_pred)
    rmse_m2 = rmse(y_true, m2_pred)
    rmse_m3 = rmse(y_true, m3_pred)
    # compute within fold loss

    m1_error = append(m1_error, rmse_m1)
    m2_error = append(m2_error, rmse_m2)
    m3_error = append(m3_error, rmse_m3)
    # Collect losses
}

m1_error = mean(m1_error)
m2_error = mean(m2_error)
m3_error = mean(m3_error)
# Compute average error per model.

which.min(c(m1_error, m2_error, m3_error))
# Find the model with minimum average error.
```

Depending on the shuffling of the data set the best model seems to be either the first or the second one. However the second model, which is the model selected using ridge regression with the optimal λ, appears to be the best one in the majority of shuffles.