Project 1

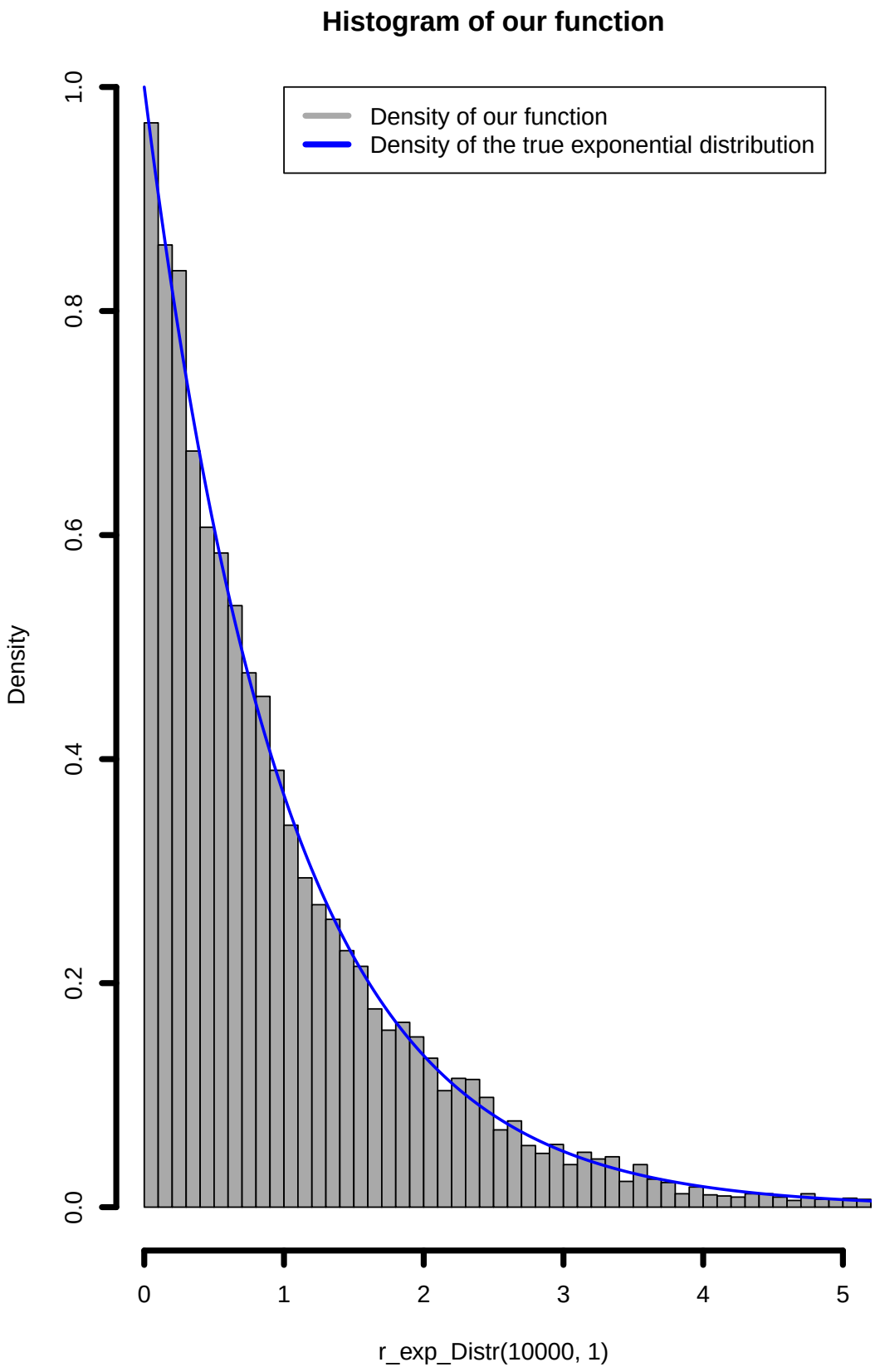
Martinius Singdahlsen, Ola Rasmussen, Johan Lagardére

Problem A)

1.

$$f(x) = \lambda e^{-\lambda x} \quad , \quad x > 0$$
$$\Rightarrow F(x) = 1 - e^{-\lambda x}$$
$$\Rightarrow F^{-1}(u) = -\frac{1}{\lambda} log(u)$$

```
set.seed(98)  #Very nice seed. Will use in all problems

# Inversion sampling returning F inverse of u
r_exp_Distr <- function(n, rate) {
    u <- log(runif(n))
    x <- -(1/rate) * u
    return(x)
}

# Making the histogram of our function
hist(r_exp_Distr(10000, 1), main = paste("Histogram of our function"),
    freq = F, col = "darkgray", lwd = 4, breaks = 100, xlim = c(0,
        5))
# Showing the true exponential distribution
lines(x = seq(0, 10, length.out = 10000), dexp(x = seq(0, 10,
    length.out = 10000), rate = 1), col = "blue", lwd = 2)
legend(1, 1, c("Density of our function", "Density of the true exponential distribution"
    col = c("darkgray", "blue"), lty = c(1, 1), lwd = c(4, 4))
```


Histogram of our function
Density of our function
Density of the true exponential distribution
Density
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
r_exp_Distr(10000, 1)

2.

(a)

$$g(x) = \begin{cases} cx^{\alpha-1} & ,0 < x < 1 \\ ce^{-x} & ,1 \leq x \\ 0 & ,\text{elsewhere} \end{cases}$$

$$\begin{aligned} G(x) &= \int_0^{\infty} g(y)\,dy \\ &= \begin{cases} \int_0^x cy^{\alpha-1}\,dy & ,0 < x < 1 \\ \int_0^1 cy^{\alpha-1}\,dy + \int_1^x ce^{-y}\,dy & ,1 \leq x \end{cases} \\ &= \begin{cases} \frac{c}{\alpha}\Big[y^{\alpha}\Big]_0^x & ,0 < x < 1 \\ \frac{c}{\alpha}\Big[y^{\alpha}\Big]_0^1 + c\int_{-1}^{-x} -e^u\,du, \text{ where } u = -y & ,1 \leq x \end{cases} \\ &= \begin{cases} \frac{c}{\alpha}x^{\alpha} & ,0 < x < 1 \\ \frac{c}{\alpha} - c\Big[y^{\alpha}\Big]_{-1}^{-x} & ,1 \leq x \end{cases} \\ &= \begin{cases} \frac{c}{\alpha}x^{\alpha} & ,0 < x < 1 \\ \frac{c}{\alpha} - c\left(e^{-x} - \frac{1}{e}\right) & ,1 \leq x \end{cases} \end{aligned}$$

Now that we have found the CDF, we will derive the normalizing constant $c(\alpha)$. To do that we need to set the $CDF = 1$ when x goes to infinity.

$$\begin{aligned} \lim_{x\to+\infty} G(x) = 1 \Leftrightarrow \frac{c}{\alpha} + \lim_{x\to+\infty}\Big[ce^{-y}\Big]_0^x &= 1 \\ \Rightarrow \underline{\underline{c(\alpha) = \frac{\alpha e}{\alpha + e}}} & \end{aligned}$$

When finding the inverse CDF, $G^{-1}(u)$, we need to find the inverse of the function when $0 < x < 1$ and when $1 \leq x$

When $0 < x < 1$:

$$\begin{aligned} u &= \frac{c}{\alpha}x^{\alpha} \\ \frac{\alpha}{c}u &= x^{\alpha} \\ x &= \left(\frac{\alpha}{c}u\right)^{\frac{1}{\alpha}} \\ &= \left(\frac{\alpha}{\frac{\alpha e}{\alpha+e}}u\right)^{\frac{1}{\alpha}} \\ &= \left(\frac{\alpha+e}{e}u\right)^{\frac{1}{\alpha}} \end{aligned}$$

$0 < x < 1 \Leftrightarrow 0 < u < \frac{c}{\alpha} 1^{\alpha} = \frac{\frac{\alpha e}{\alpha+e}}{\alpha} = \frac{e}{\alpha+e}$

So $0 < u < \frac{e}{\alpha+e}$

When $1 \leq x$:

$$\begin{aligned}
u &= \frac{c}{\alpha} - c\left(e^{-x} - \frac{1}{e}\right) \\
\frac{u - \frac{c}{\alpha}}{-c} &= e^{-x} - \frac{1}{e} \\
\frac{1}{\alpha} - \frac{u}{c} + \frac{1}{e} &= e^{-x} \\
x &= -ln\left(\frac{1}{\alpha} - \frac{u}{\frac{\alpha e}{\alpha+e}} + \frac{1}{e}\right) \\
&= -ln\left(\frac{1}{\alpha} - \frac{\alpha+e}{\alpha e}u + \frac{1}{e}\right) \\
&= -ln\left(\frac{1}{e} + \frac{e-(\alpha+e)u}{\alpha e}\right) \\
&= -ln\left(\frac{(e+\alpha)-(\alpha+e)u}{\alpha e}\right) \\
&= ln\left(\frac{\alpha e}{(e+\alpha)-(\alpha+e)u}\right) \\
&= ln\left(\frac{\alpha e}{(e+\alpha)(1-u)}\right)
\end{aligned}$$

$1 \leq x < \infty \Leftrightarrow = \frac{e}{\alpha+e} < u < 1$

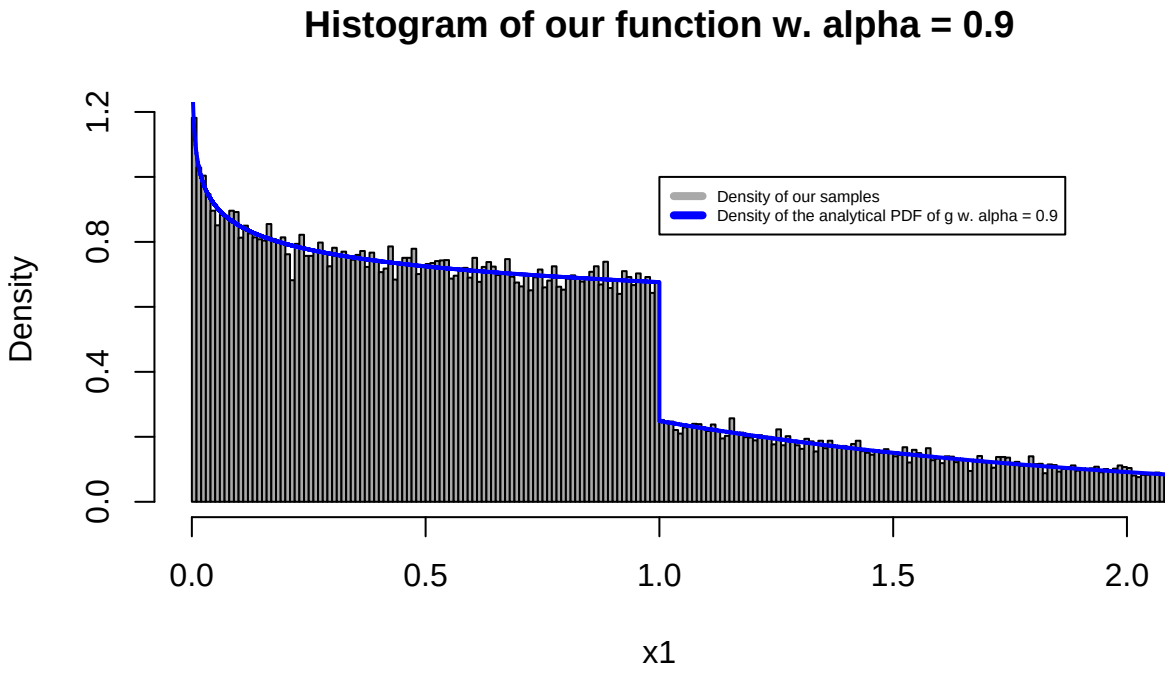
So our inverse becomes:

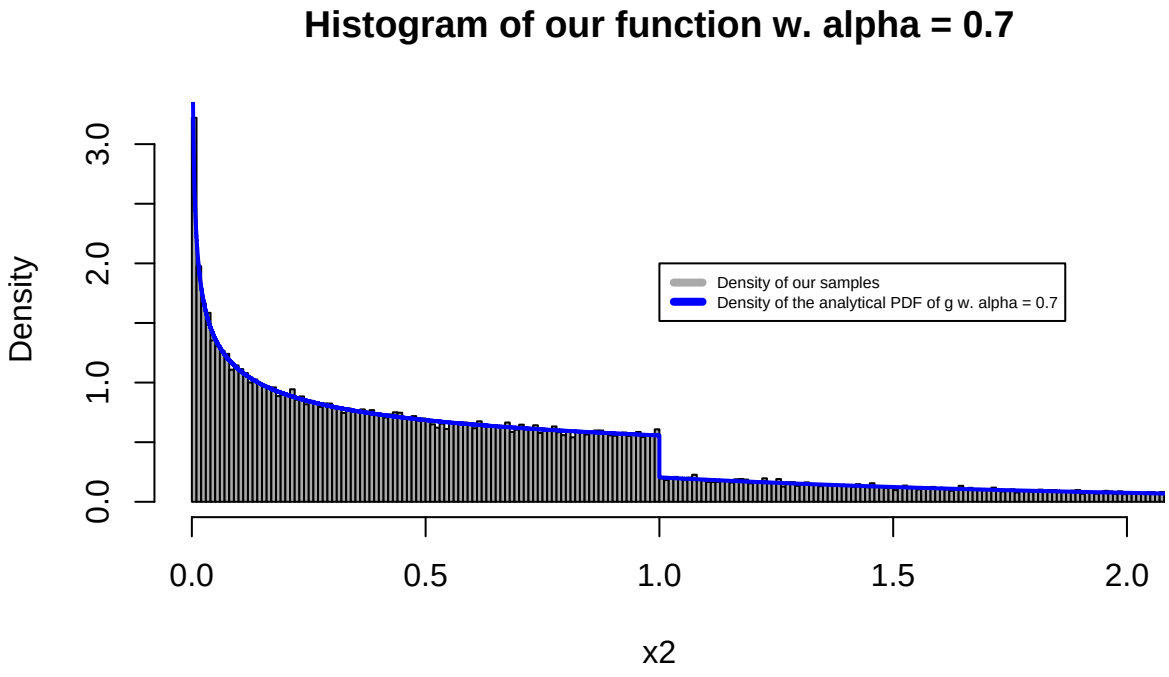
$$G^{-1}(u) = \begin{cases} \left(\frac{\alpha+e}{e}u\right)^{\frac{1}{\alpha}} & , 0 < u < \frac{e}{\alpha+e} \\ ln\left(\frac{\alpha e}{(e+\alpha)(1-u)}\right) & , \frac{e}{\alpha+e} \leq u < 1 \end{cases}$$

(b) Code

```
# Analytical function
analytical_g <- function(x, alpha) {

    # Normalizing constant
    c <- (alpha * exp(1))/(alpha + exp(1))

    # Creating density vector
    dens <- vector(length = length(x))

    # Finding values when x<1
    dens[x < 1] <- c * (x[x < 1])^(alpha - 1)

    # Finding values when x>=1
    dens[x >= 1] <- c * exp(-x[x >= 1])

    return(as.double(dens))
}

# Samples using inversion sampling
sample_g <- function(n, alpha) {
    # Drawing samples
    u <- runif(n, 0, 1)

    samples <- vector(length = n)

    # Finding values when u<e/(a+e)
    samples[u < (exp(1)/(alpha + exp(1)))] <- (((alpha + exp(1))/exp(1)) *
        u[u < (exp(1)/(alpha + exp(1)))])^(1/alpha)

    # Finding values when u>=e/(a+e)
    samples[u >= (exp(1)/(alpha + exp(1)))] <- log((alpha * exp(1))/((exp(1) +
        alpha) * (1 - u[u >= (exp(1)/(alpha + exp(1)))])))

    return(samples)
}
```

```r
set.seed(98)
n <- 1e+05
alpha1 <- 0.9
alpha2 <- 0.7
x1 <- sample_g(n, alpha1)
x2 <- sample_g(n, alpha2)

# Making histograms
par(mfrow = c(2, 1))
hist(x1, main = paste("Histogram of our function w. alpha =",
    alpha1), freq = F, breaks = 1000, xlim = c(0, 2), col = "darkgray")
lines(x1[order(x1)], analytical_g(sort(x1), alpha1), col = "blue",
    lwd = 2)
legend(1, 1, c("Density of our samples", paste("Density of the analytical PDF of g w. al
    alpha1)), col = c("darkgray", "blue"), lty = c(1, 1), lwd = c(4,
    4), cex = 0.5)

alpha <- 0.7
x2 <- sample_g(n, alpha)
hist(x2, main = paste("Histogram of our function w. alpha =",
    alpha2), freq = F, breaks = 1000, xlim = c(0, 2), col = "darkgray")
lines(x2[order(x2)], analytical_g(sort(x2), alpha2), col = "blue",
    lwd = 2)
legend(1, 2, c("Density of our samples", paste("Density of the analytical PDF of g w. al
    alpha2)), col = c("darkgray", "blue"), lty = c(1, 1), lwd = c(4,
    4), cex = 0.5)
```





Looking at this histogram, we can see that our sampler is working properly, even for different values of alpha.

3.

(a) We will first find the normalizing constant to the function defined in the problem. We will do this by integrating over its domain. First we make the substitution $u = 1 + e^{ax}$. Observe that this does alter where we evaluate the integral:

$$\begin{aligned} 1 &= c \int_R \frac{c * e^{ax}}{(1 + e^{ax})^2} dx \\ &= c \int_1^{1+e^{ar}} \frac{du}{a * u^2} \\ &= (c/a) \left[\frac{-1}{u} \right]_1^{1+e^{ar}} \\ &= (c/a)(1 - 1/(1 + e^{ar})) \end{aligned}$$

Taking the limit of when r goes to infinity we thus obtain $c = a$.

(b) Finding the cumulative distribution, F, is done similar, only with different evaluations of the integral. Here again we do the substitution $u = 1 + e^{at}$.

$$\begin{aligned} F(x) &= \int_{-\infty}^{x} \frac{a \cdot e^{at}}{(1 + e^{at})^2} dx \\ &= \int_1^{1+e^{ax}} \frac{a \cdot du}{a \cdot u^2} \\ &= - \left[\frac{1}{u} \right]_1^{1+e^{ax}} \\ &= - \left(\frac{1}{1 + e^{ax}} - 1 \right) \\ &= 1 - \frac{1}{1 + e^{ax}} \end{aligned}$$

This does look right as we can observe that our cumulative distribution does integrate to one when x goes to infinity. To apply inverse sampling we need to find the inverse of the cumulative distribution. Setting w equal to the cumulative distribution we obtain,

$$
\begin{aligned}
w &= 1 - \frac{1}{1+e^{ax}} \\
\Rightarrow \frac{1}{1+e^{ax}} &= 1 - w \\
\Rightarrow 1 + e^{ax} &= \frac{1}{1-w} \\
\Rightarrow e^{ax} &= \frac{1}{1-w} - 1 \\
\Rightarrow x &= (1/a) ln(\frac{1}{1-w} - 1) \\
F^{-1}(x) &= (1/a) ln(\frac{1}{1-x} - 1)
\end{aligned}
$$

(c) Now we will code a function that simulates n draws with specified $a > 0$ from the distribution. We will also get an indicator if it works by calculating its mean.

```
sim_double_exp <- function(n, a) {
    veac <- (1/a) * log(1/(1 - runif(n)) - 1)

    return(veac)
}
```

Interestingly enough the mean of this distribution for all $a > 0$ is zero. This is because that the distribution is symmetric:

$$
\begin{aligned}
f(-x) &= \frac{a \cdot e^{-ax}}{(1+e^{-ax})^2} \\
&= \frac{a \cdot e^{-ax}}{1 + 2e^{-ax} + e^{-2ax}} \\
&= \frac{a \cdot e^{-ax}}{e^{-2ax}(e^{2ax} + 2e^{a*x} + 1)} \\
&= \frac{a \cdot e^{-ax} \cdot e^{2ax}}{(e^{ax}+1)^2} \\
&= \frac{a \cdot e^{ax}}{(1+e^{ax})^2} \\
&= f(x)
\end{aligned}
$$

We will therefore get an indicator if the code works properly if the empirical mean is zero for different values of a. This will give us an indicator that it is symmetric for multiple values of a. Underneath we have calculated the empirical mean of the distribution five times where $n = 10000$ for the five calculations, however a differs and is equal to 1/2, 1, 5, 15, and 50.

```r
set.seed(98)

# MEans for different a values
mean_0.5 <- mean(sim_double_exp(n = 10000, 1/2))
mean_1 <- mean(sim_double_exp(n = 10000, 1))
mean_5 <- mean(sim_double_exp(n = 10000, 5))
mean_15 <- mean(sim_double_exp(n = 10000, 15))
mean_50 <- mean(sim_double_exp(n = 10000, 50))

cat("Mean for a=0.5: ", mean_0.5, "\nMean for a=1:   ", mean_1,
    "\nMean for a=5:  ", mean_5, "\nMean for a=15:  ", mean_15,
    "\nMean for a=50: ", mean_50)
```

```
## Mean for a=0.5:  0.03035133
## Mean for a=1:    0.02031997
## Mean for a=5:   -0.0003879047
## Mean for a=15:   0.0002223044
## Mean for a=50:  -2.456647e-05
```

We observe that the greater a is the closer our empirical mean becomes zero. This does coincide with visual graphing of the probability distribution as when a becomes smaller, the function flattens out. Then there is more variance and we expect a worse estimate. It is clear that this is evidence supporting that our coded function works.

4.

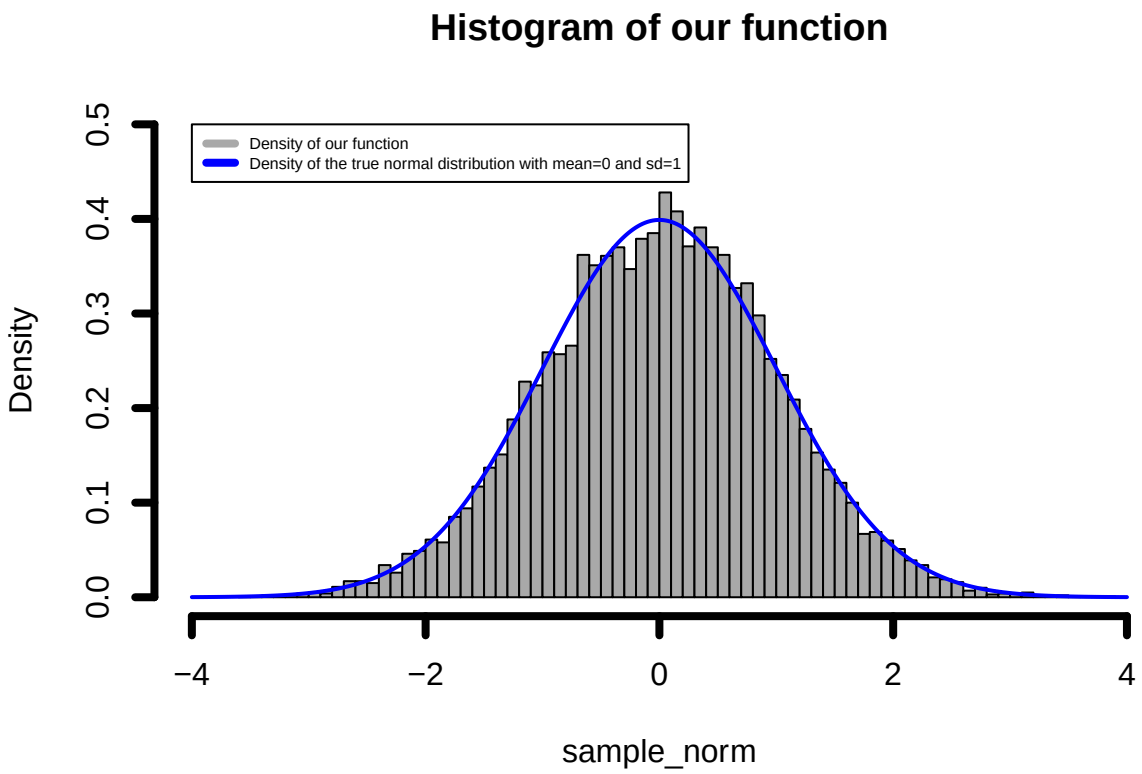
We need to simulate the standard normal distribution to be able to do task (C). This will be done by applying the Box-Muller algorithm. Specifically we have to generate a vector of n independent standard normal. The code first checks if it is an odd number, and if it is it ads a single standard normal to the return vector. If $n = 1$ this vector is returned. If $n \neq 1$ it generates an even number of independent standard normal's and returns all the generated standard normal's in a vector.

```r
box_m <- function(n) {
    r_numbers <- c()

    if (n%%2 == 1) {
        r_2 <- -2 * log(runif(1))
        theta <- 2 * pi * runif(1)
        x_1 <- sin(theta) * sqrt(r_2)
        r_numbers <- c(r_numbers, x_1)
    }
```

```
    if (n == 1) {
        return(r_numbers)
    }

    for (i in 1:(n%/%2)) {
        r_2 <- -2 * log(runif(1))
        theta <- 2 * pi * runif(1)
        x_1 <- sin(theta) * sqrt(r_2)
        x_2 <- cos(theta) * sqrt(r_2)
        r_numbers <- c(r_numbers, x_1, x_2)
    }

    return(r_numbers)
}
```

To check if it works properly we will make a histogram of 10000 simulated variables and calculate its empirical mean and variance.

```
set.seed(98)
n = 10000
sample_norm <- box_m(n)

# Making the histogram of our function
hist(sample_norm, main = paste("Histogram of our function"),
    freq = F, col = "darkgray", lwd = 4, breaks = 100, xlim = c(-4,
        4), ylim = c(0, 0.5))
# Showing the true normal distribution
lines(x = seq(-4, 4, length.out = n), dnorm(x = seq(-4, 4, length.out = n),
    mean = 0, sd = 1), col = "blue", lwd = 2)
legend(-4, 0.5, c("Density of our function", "Density of the true normal distribution wi
    col = c("darkgray", "blue"), lty = c(1, 1), lwd = c(4, 4),
    cex = 0.5)
```



This histogram of the drawn random numbers from the box_m function does resemble a normal distribution. This leads us to believe that the function works. Further more we know that the mean of a standard normal is zero and its variance is one. We will use the same $n = 10000$ samples to see if we get corresponding values.

```
cat("Mean:", mean(sample_norm), "\nVar: ", var(sample_norm))
```

```
## Mean: 0.003345845
## Var:  0.9823667
```

We can see that the variance and mean does correspond with the true values of the standard normal, thus further suporting that our code works properly.

5.

To simulate a draw of a d-variate normal distribution with given mean and variance, we will use the function defined to simulate n independent random normal's and preform a linear transformation. Specifically a linear transform on the form,

$$Z = \mu + M^{1/2}X$$

, where μ is a vector representing the mean of Z, $M^{1/2}$ is the symmetric square root of the variance matrix and X is a vector of n independent draws form a standard normal. Note that the symmetric square root always exists of a variance matrix since it is symmetric and positive-semi-definite matrix. Z then has mean μ and variance matrix $\Sigma = M^{1/2}t(M^{1/2}) = M^{1/2}M^{1/2}$.

```
d_variate <- function(u, S) {
    z <- u + mhalf(S) %*% box_m(length(u))
    return(z)
}
```

We will get an indicator if the function works by calculating the mean and variance of 1000 realizations of the function.

```
mean <- c(1, 2, 3)
corM <- matrix(c(3, 1/2, 1, 1/2, 2, -1/2, 1, -1/2, 1), nrow = 3,
    ncol = 3)

sample_20 <- c(1, 1, 1)
for (i in 1:1000) {
    sample_20 <- matrix(c(as.vector(sample_20), d_variate(mean,
        corM)), byrow = FALSE, ncol = i + 1, nrow = 3)
}
sample_20 <- sample_20[, -1]

mean_ofmat <- round(c(mean(sample_20[1, ]), mean(sample_20[2,
    ]), mean(sample_20[3, ])), digits = 3)
var_ofmat <- var((t(sample_20)))
V <- round(var_ofmat, digits = 2)
names <- c("X_1", "X_2", "X_3")
```

The known mean and variance matrix is:

$$\mu = (1, 2, 3)$$

$$\Sigma = \begin{bmatrix} 3 & 1/2 & 1 \\ 1/2 & 2 & -1/2 \\ 1 & -1/2 & 1 \end{bmatrix}$$

and the variance and mean of the simulated d-variate random normal can be seen in the matrix coded below. Here the second row is the mean and the 4th to 7th row is the variance matrix:

```
cat("Mean vector:     ", c(mean_ofmat), "\n", "\nVariance matrix:  ",
    matrix(V[c(1, 2, 3), 1]), "\n                  ", matrix(V[c(1,
        2, 3), 2]), "\n                 ", matrix(V[c(1, 2, 3),
        2]))
```

```
## Mean vector:      0.965 2.056 2.97
##
## Variance matrix:   3.1 0.45 1.08
##                   0.45 1.99 -0.49
##                   0.45 1.99 -0.49
```

As we can see this is quite similar to what we putted into the function that generates the d-variate normal distribution. This leads us to believe that the code works.

Problem B)

1.

We consider a gamma distribution with parameters $\alpha \in (0,1)$ and $\beta = 1$ and we use these functions to define the acceptance probability:

$$f(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x} & ,0 < x \\ 0 & ,elsewhere \end{cases}$$

$$g(x) = \begin{cases} cx^{\alpha-1} & ,0 < x < 1 \\ ce^{-x} & ,1 \leq x \\ 0 & ,elsewhere \end{cases}$$

We can do that because we have $g(x) > 0$ whenever $f(x) > 0$ so there exists a $c > 1$ such that $\frac{f(x)}{g(x)} \leq c, x : f(x) > 0$.

(a) The rejection sampling algorithm is used like this:I take a sample $x \sim g(x)$ (we can use the one we got from the problem 1) then we generate $u \sim U[0,1]$ and compare it to $\alpha = \frac{f(x)}{cg(x)}$,if $u \leq \alpha$ then x is a samples of $f(x)$. If it isn't the case we redo all of this until it is. So now we would like to find the overall acceptance probability:

$$\begin{aligned} P_a &= P\left(U \leq \frac{1}{c}\frac{f(x)}{g(x)}\right) \\ &= \int_{-\infty}^{\infty} \frac{f(x)}{c.g(x)} g(x)dx \\ &= c^{-1} \int_{-\infty}^{\infty} f(x)dx \\ &= c^{-1} \end{aligned}$$

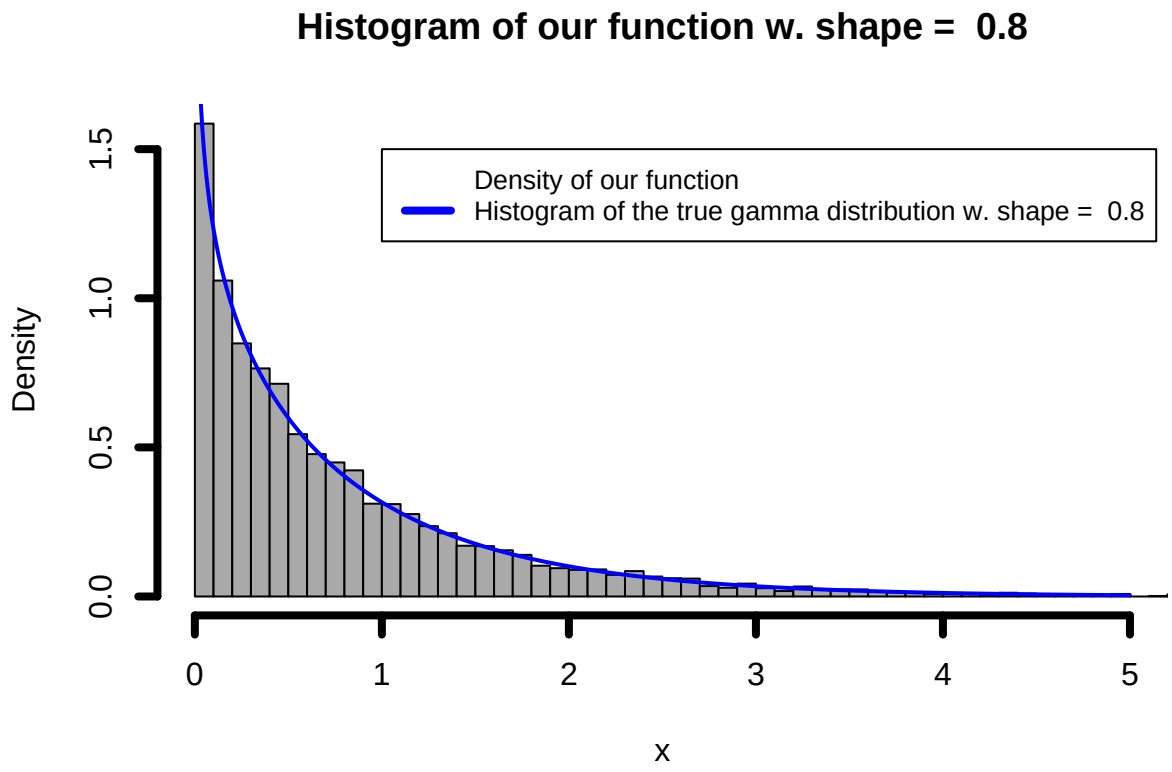
And so we have to found the constant c which has to be more than $\frac{f(x)}{g(x)}$ $\forall x : f(x) > 0$,and so the lowest ratio has to be the sup of this quantity:

$$
\begin{aligned}
c &= \sup_x \frac{f(x)}{g(x)} \\
&= \sup_x \begin{cases} \frac{\alpha+e}{\alpha e}\frac{e^{-x}}{\Gamma(\alpha)}, & 0 < x < 1 \\ \frac{\alpha+e}{\alpha e}\frac{x^{\alpha-1}}{\Gamma(\alpha)}, & 1 \leq x \end{cases} \\
&= \sup_x \begin{cases} \frac{e^{-x}}{\Gamma(\alpha)}\left(\frac{1}{\alpha}+\frac{1}{e}\right), & 0 < x < 1 \\ \frac{x^{\alpha-1}}{\Gamma(\alpha)}\left(\frac{1}{\alpha}+\frac{1}{e}\right), & 1 \leq x \end{cases} \\
\Rightarrow c &= \frac{\left(\frac{1}{\alpha}+\frac{1}{e}\right)}{\Gamma(\alpha)} \\
&= \frac{(\alpha+e)}{\alpha e\Gamma(\alpha)} \\
\Rightarrow P_{a,optimal} &= \frac{\alpha e\Gamma(\alpha)}{(\alpha+e)}
\end{aligned}
$$

(b) So we can use this c to implement a sampling algorithm giving n independent samples from f.

```
set.seed(98)
# We can define the density of a Gamma function thanks to
# the function :
rejection_sampling <- function(n, alpha) {
    c = (1/alpha + exp(-1))/gamma(alpha)
    number_it = 0
    samples <- vector()
    while (number_it <= n) {

        finished = 0
        while (finished == 0) {
            u <- runif(1)
            x = sample_g(1, alpha)
            f = dgamma(x, alpha)
            g = analytical_g(x, alpha)
            acceptance = 1/c * (f/g)

            if (u <= acceptance) {
                finished = 1
                samples <- append(samples, x)
            }
            number_it = number_it + 1

        }
    }
```

```
    return(samples)
}
n = 10000
alpha = 0.8
x = rejection_sampling(n, alpha)

# Making the histogram of our function
hist(x, main = paste("Histogram of our function w. shape = ",
    alpha), freq = F, col = "darkgray", lwd = 4, breaks = 100,
    xlim = c(0, 5))

# Showing the true exponential distribution
lines(x = seq(0, 5, length.out = n), dgamma(x = seq(0, 5, length.out = n),
    alpha), col = "blue", lwd = 2)

legend(1, 1.5, c("Density of our function", paste("Histogram of the true gamma distribut
    alpha)), col = c("darkgray", "blue"), lty = c(0.5, 1), lwd = c(4,
    4), cex = 0.8)
```



2.

$$C_f = \{(x_1, x_2) : 0 \leq x_1 \leq \sqrt{f^*\left(\frac{x_2}{x1}\right)}\} \quad where$$
$$f^*(x) = \begin{cases} x^{\alpha-1}e^{-x} & , if\ 0 < x \\ 0 & , elsewhere \end{cases}$$

(a) We search $a = \sqrt{\sup_x f^*(x)}$ to do so we calculate:

$$f'^*(x) = (\alpha-1)x^{\alpha-2}e^{-x} - x^{\alpha-1}e^{-x} = ((\alpha-1)-x)x^{\alpha-2}e^{-x}$$

who cancels out when $x = \alpha - 1$ which exists because $\alpha > 1$ and also we have:

$$\begin{cases} f'^*(x) > 0 & , if\ x < \alpha - 1 \\ f'^*(x) < 0 & , if\ x > \alpha - 1 \end{cases}$$
$$so\ \ that : \sup_x f^*(x) = f^*(^\alpha\!-1) = (\alpha-1)\alpha - 1e^{-(\alpha-1)}$$
$$\Rightarrow a = \sqrt{\sup_x f^*(x)} = (\alpha-1)^{\frac{\alpha-1}{2}} e^{-\frac{(\alpha-1)}{2}}$$

Now we search $b_+ = \sqrt{\sup_{x\geq 0}(x^2 f^*(x))}$, and so we define:

$$\begin{aligned} g^*(x) &= x^2 f^*(x) \\ &= \begin{cases} x^{\alpha+1}e^{-x} & , if\ 0 < x \\ 0 & , \text{elsewhere} \end{cases} \end{aligned}$$

and we calculate :

$$\begin{aligned} g'^*(x) &= (\alpha+1)x^\alpha e^{-x} - x^{\alpha+1}e^{-x} \\ &= ((\alpha+1)-x)x^\alpha e^{-x} \\ &= 0 \\ &\Leftrightarrow \\ & x = \alpha + 1 \text{ which exist because } \alpha > 1 \end{aligned}$$

And we also have:

$$\begin{cases} g'^*(x) > 0 & , if\ x < \alpha + 1 \\ g'^*(x) < 0 & , if\ x > \alpha + 1 \end{cases}$$
$$so\ \ that : \sup_x g^*(x) = g^*(\alpha+1) = (\alpha+1)\alpha + 1e^{-(\alpha+1)}$$
$$\Rightarrow b_+ = \sqrt{\sup_x g^*(x)} = (\alpha+1)^{\frac{\alpha+1}{2}} e^{-\frac{(\alpha+1)}{2}}$$

Finally we have $b_- = -\sqrt{\sup_{x\leq 0}(x^2 f^*(x))} = 0$. Then we will take $y = \frac{x_1}{x_2} \sim Gamma(\alpha, \beta = 1)$. And to select x_1 and x_2 are sampled uniformly inside C_f .To do so we generate $x_1 \sim U[0, a]$ and$x_2 \sim U[b_-, b_+]$ and we select only $(x_1, x_2) \in C_f$. The problem is that we have $(\alpha + 1)^{\alpha+1}$ which will dominate the exponential function so that when α grow it will grow exponentially causing a numerical overflow. So we will choose to sample in log-scale so that we will have:

$$\begin{aligned} x_1 &= aU[0,1] \\ \Rightarrow ln(x_1) &= ln(a) + ln\left(U[0,1]\right) \\ x_2 &= b_- + (b_+ - b_-)U[0,1] \quad but \quad as \quad b_- = 0 \\ \Rightarrow ln(x_2) &= ln(b_+) + ln\left(U[0,1]\right) \end{aligned}$$

so that the condition with the logarithm become:

$$\begin{aligned} ln(x_1) &\leq ln\left(\sqrt{f^*\left(\frac{x_2}{x_1}\right)}\right) \\ \Rightarrow 2ln(x_1) &\leq (\alpha - 1)(ln(x_2) - ln(x_1)) - e^{ln(x_2)-ln(x_1)} \end{aligned}$$

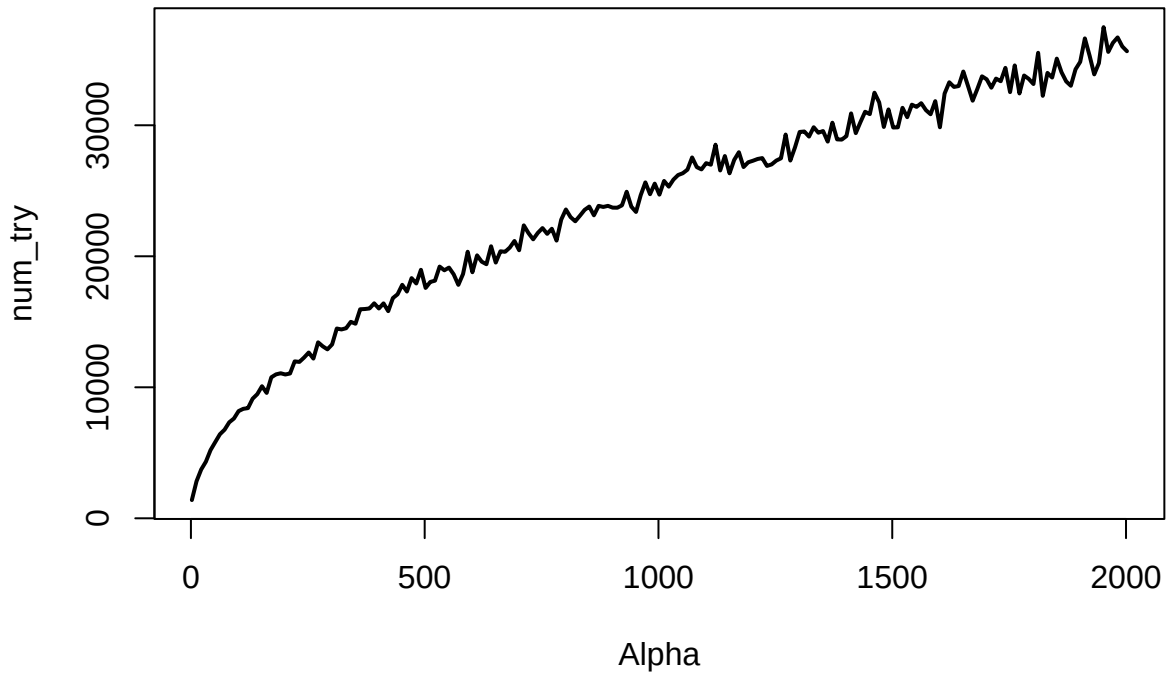
Also we finally we will have:

$$\begin{aligned} ln(a) &= \frac{\alpha - 1}{2}(ln(\alpha - 1) - 1) \\ ln(b_+) &= \frac{\alpha + 1}{2}(ln(\alpha + 1) - 1) \end{aligned}$$

(b) Code

```
set.seed(98)
# function for n samples from ratio uniform in log-scale
ratio_uniform <- function(n, alpha) {
    loga = (log(alpha - 1) - 1) * (alpha - 1)/2
    logbplus = (log(alpha + 1) - 1) * (alpha + 1)/2
    tried = 0
    samples <- vector()

    while (length(samples) < n) {

        tried = tried + 1
        logx1 = loga + log(runif(1))
        logx2 = logbplus + log(runif(1))

        if (2 * logx1 <= (alpha - 1) * (logx2 - logx1) - exp(logx2 -
            logx1)) {
```

```
            samples <- append(samples, exp(logx2 - logx1))
        }
    }
    return(list(samples = samples, tried = tried))
}

n = 1000
Alpha <- seq(from = 2, to = 2002, 10)
num_try <- vector()
for (i in 1:length(Alpha)) {
    num_try <- append(num_try, ratio_uniform(n, Alpha[i])$tried)
}

plot(Alpha, num_try, type = "l", lwd = 2)
```



So by the results we can see that the number of try grow as a $log(\alpha)$ so that when we raise the number α it will have little impact in the number of iterations!

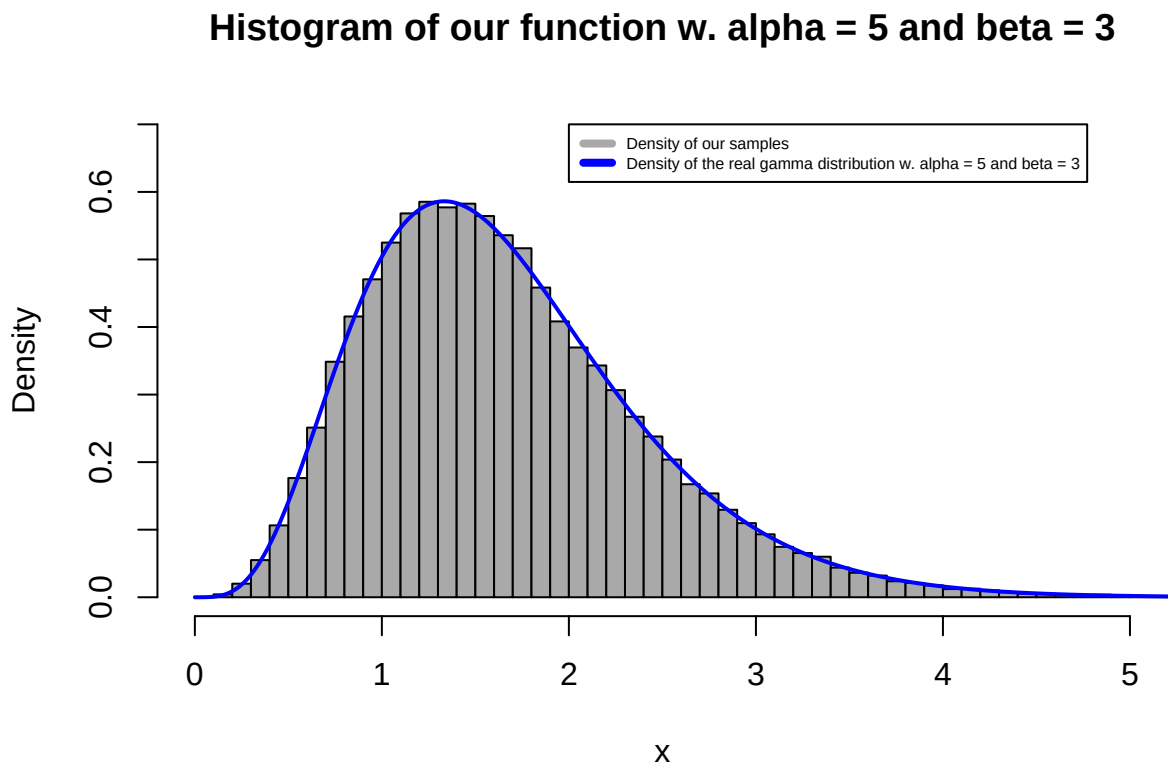
3.

For now we will try to generates samples from a $Gamma(\alpha, \beta)$ distribution. Or as the gamma distribution parameter *beta* is an inverse scale parameter, simulating a $Gamma(\alpha, \beta)$ is equivalent to simulate $Gamma(\alpha, 1)$ and divide by β, also a $Gamma(1, 1)$ is the same as an exponential distribution so that we can use the function of the Problem A.

```
# function for n samples from a Gamma distribution
Gamma_sample <- function(n, alpha, beta) {
    samples = vector(length = n)
    if (alpha == 1) {
        samples = sample_expo(n, 1)
    } else if (alpha < 1) {
        samples = rejection_sampling(n, alpha)
    } else (alpha > 1)
    {
        samples = ratio_uniform(n, alpha)$samples
    }
    samples = samples/beta
    return(samples)
}
```

We can then try to use it on for example Gamma(5,3) and see if it works well:

```
set.seed(98)
n = 1e+05
alpha = 5
beta = 3
x = Gamma_sample(n, alpha, beta)

# We do an histogram and we also do the pdf to compare if
# it fits well
hist(x, main = paste("Histogram of our function w. alpha =",
    alpha, "and beta =", beta), freq = F, breaks = 100, xlim = c(0,
    5), ylim = c(0, 0.7), col = "darkgray")
lines(x = seq(0, 10, length.out = 10000), dgamma(x = seq(0, 10,
    length.out = 10000), shape = alpha, rate = beta), col = "blue",
    lwd = 2)
legend(2, 0.7, c("Density of our samples", paste("Density of the real gamma distribution
    alpha, "and beta =", beta)), col = c("darkgray", "blue"),
    lty = c(1, 1), lwd = c(4, 4), cex = 0.5)
```



4.

(a) So we consider $x \sim Gamma(\alpha, 1)$ and $y \sim Gamma(\beta, 1)$ independently sampled and $z = \frac{x}{x+y}$, we want to prove that $z \sim beta(\alpha, \beta)$.

Firstly as they are Gamma distribution, we have $f_x(x) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}$ and $f_y(y) = \frac{1}{\Gamma(\beta)} y^{\beta-1} e^{-y}$ and we can have the joint distribution depending from x and y:

$$\begin{aligned} f_{x,y}(x,y) &= f_x(x).f_y(y) \\ &= \frac{1}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} y^{\beta-1} e^{-(x+y)}, \quad \forall (x,y) \in \mathbf{R}_+{}^2 \end{aligned}$$

so we want to go from x and y to z and we define the equivalence with functions f_1 and f_2:

$$\begin{cases} z = f_1(x,y) = \frac{x}{x+y} \\ z_2 = f_2(x,y) = x + y \end{cases} \Leftrightarrow \begin{cases} x = f_1^{-1}(x,y) = z.z_2 \\ y = f_2^{-1}(x,y) = z_2 - z.z_2 \end{cases}$$

If we want to go from (x, y) to (z, z_2), we will have to calculate the Jacobi determinant of this transformation:

$$
\begin{aligned}
detJ_{x,y}(z,z_2) &= det\begin{pmatrix} \frac{\partial x}{\partial z} & \frac{\partial x}{\partial z_2} \\ \frac{\partial y}{\partial z} & \frac{\partial y}{\partial z_2} \end{pmatrix} \\
&= det\begin{pmatrix} z_2 & z \\ -z_2 & 1-z \end{pmatrix} \\
&= z_2(1-z) + zz_2
\end{aligned}
$$

So we use the bivariate transformations:

$$
\begin{aligned}
f_{z,z_2}(z,z_2) &= |detJ_{x,y}|(z,z_2) f_{x,y}(x,y) \\
&= |detJ_{x,y}(z,z_2)| f_{x,y}(zz_2, z_2 - z.z_2) \\
&= |z_2| \frac{1}{\Gamma(\alpha)\Gamma(\beta)} (zz_2)^{\alpha-1}(z_2 - zz_2)^{\beta-1} e^{-z_2}
\end{aligned}
$$

Finally we can use the total probability theorem and we can conclude:

$$
\begin{aligned}
f_z(z) &= \int_{-\infty}^{\infty} f_{z,z_2}(z,z_2) dz_2 \\
&= \frac{1}{\Gamma(\alpha)\Gamma(\beta)} z^{\alpha-1}(1-z)^{\beta-1} \int_0^{\infty} z_2^{\alpha+\beta-1} e^{-z_2} dz_2 \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} z^{\alpha-1}(1-z)^{\beta-1}
\end{aligned}
$$

$$
\begin{aligned}
&because \ \ we \ \ define \ \ \Gamma: \\
&\qquad z_2 \mapsto \int_0^{\infty} z_2^{\lambda-1} e^{-z_2} dz_2 \text{ for a } Gamma(\lambda) \\
&\Rightarrow z \sim beta(\alpha, \beta)
\end{aligned}
$$

(b) Now we want to generate n independent samples from a beta distribution, to do so we generate $x \sim Gamma(\alpha, 1)$ and $y \sim Gamma(\beta, 1)$and we calculate $z = \frac{x}{x+y} \sim beta(\alpha, \beta)$:

```
Beta_samples <- function(n, alpha, beta) {
    x = Gamma_sample(n, alpha, 1)
    y = Gamma_sample(n, beta, 1)
    z = x/(x + y)
    return(z)
}
```

We then evaluate like we did before and we plot with alpha=5 and beta=3:

```r
set.seed(98)
n = 1e+05
alpha = 5
beta = 3

x = Beta_samples(n, alpha, beta)
meantheo = alpha/(alpha + beta)
cat("Theoritical mean:", meantheo, "\nSample mean:     ", mean(x))
```

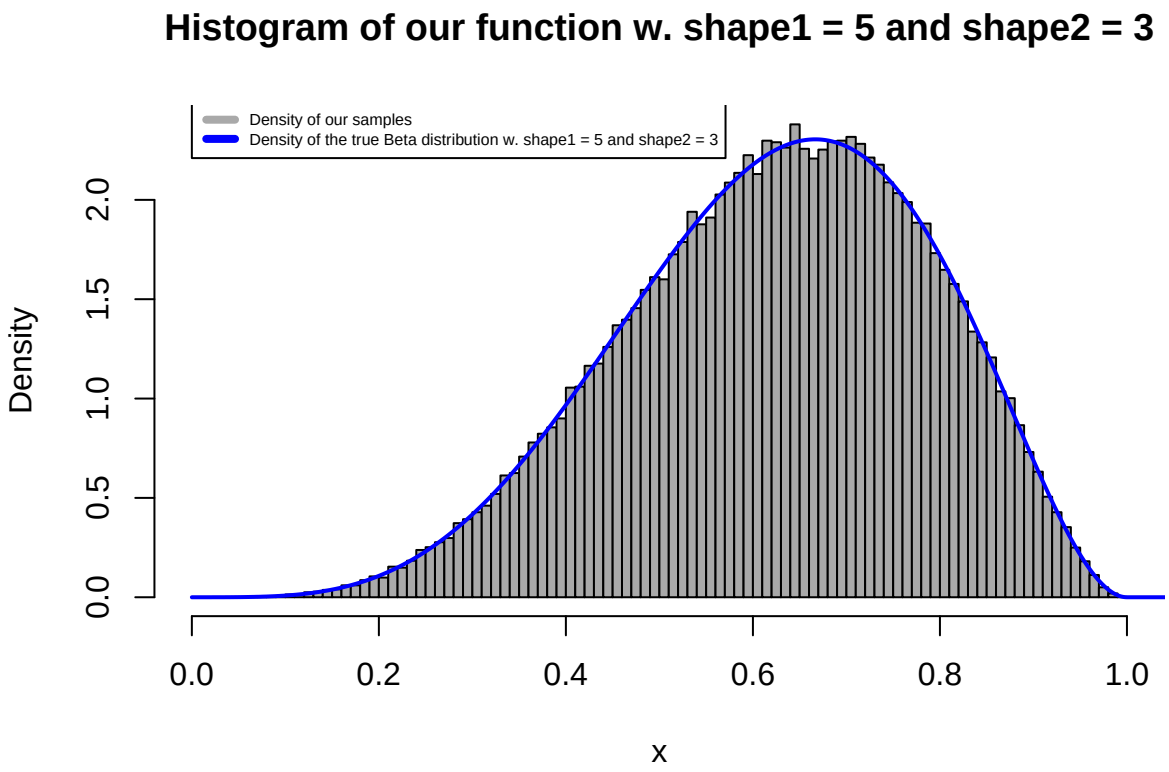
```
## Theoritical mean: 0.625
## Sample mean:      0.6247891
```

```r
vartheo = alpha * beta/((alpha + beta)^2 * (alpha + beta + 1))
cat("Theoretical variance:", vartheo, "\nSample variance:     ",
    var(x))
```

```
## Theoretical variance: 0.02604167
## Sample variance:      0.02601471
```

We have almost the same mean and variance. We can look at the distribution compared to the pdf:

```r
# We do an histogram and we also do the pdf to compare if
# it fits well
hist(x, main = paste("Histogram of our function w. shape1 =",
    alpha, "and shape2 =", beta), freq = F, breaks = 100, xlim = c(0,
    1), col = "darkgray")
lines(x = seq(0, 10, length.out = 10000), dbeta(x = seq(0, 10,
    length.out = 10000), shape1 = alpha, shape2 = beta), col = "blue",
    lwd = 2)
legend(0, 2.5, c("Density of our samples", paste("Density of the true Beta distribution
    alpha, "and shape2 =", beta)), col = c("darkgray", "blue"),
    lty = c(1, 1), lwd = c(4, 4), cex = 0.5)
```



So we have the samples and the pdf which are distributed the same. Our function seems to return samples like the theory predict it.

Problem C)

1.

We have from (A3) a random normal vector of n components. We will use this to simulate random draws from a standard normal. To calculate a 95% confidence interval, we will use the central limit theorem since we are taking the mean of a function with unknown probability density. We also do calculate the empirical variance, such that we obtain a student-t distribution. However since n is so large it is approximately a standard normal distribution. We will use this approximation to obtain the confidence interval in all tasks in (C).

```
set.seed(98)

n = 1e+05
# generating samples
normal <- box_m(n)

# Monte Carlo estimate function (making it a function to
# check that it works)
MC_function <- function(x, data) {
    return(mean(0 + (normal > x)))
}

MC_es <- MC_function(4, normal)

# Empirical Variance estimation
var_1 <- 0
for (i in 1:n) {
    var_1 <- var_1 + ((0 + (normal[i] > 4)) - MC_es)^2
}
h_var <- 1/((n - 1)) * var_1

# conf interval
conf_interval2 <- c(MC_es - 1.962 * sqrt(h_var/n), MC_es + 1.96 *
    sqrt(h_var/n))
```

The Monte Carlo estimate, Variance of the indicator function, and the 95% confidence interval using the empirical variance is given in the matrix bellow:

```
cat("Monte Carlo:       ", MC_es, "\nVar of I(x):       ", h_var,
    "\nConfidence interval", "(", conf_interval2[1], ",", conf_interval2[2],
    ")")
```

```
## Monte Carlo:        3e-05
## Var of I(x):        2.99994e-05
## Confidence interval ( -3.982497e-06 , 6.394786e-05 )
```

We can see that the confidence interval stretches over to the negative numbers. We know that the integral of a positive function can not be negative. However the value we are approximating is really small as it is equal to the integral from 4 to infinity of the standard Gaussian.

To check that the coded function works we will use 10000 samples with $x = 0$. This is the same as approximating the integral of the standard Gaussian from zero to infinity. We know that this value is 1/2. Using the same samples we get:

```
MC_function(0, normal)
```

```
## [1] 0.49927
```

This supports that our function works properly.

2.

To sample from g(x) we first find the normalizing constant and then find the inverse of the CDF of g(x).

Finding the normalizing constant through integrating the function g(x) over its domain:

$$\begin{aligned} 1 &= \int_4^r cx \cdot e^{-1/2x^2} dx \\ &= c \int_8^{1/2r^2} e^{-u} du \end{aligned}$$

Here we have done the substitution $u = 1/2x^2$. Thus we obtain,

$$\begin{aligned} 1 &= c[-e^{-u}]_8^{1/2r^2} \\ &= c(e^{-8} - e^{-1/2r^2}) \end{aligned}$$

Taking the limit when r goes to infinity we therefore end up with,

$$c = e^8$$

Finding the CDF of g, $G(x)$:

$$
\begin{aligned}
G(x) &= (c) \cdot \int_4^x t \cdot e^{-(1/2)\cdot t^2} dt \\
&= (c) \cdot \int_8^{(1/2)x^2} e^{-u} du \\
&= (c) \cdot (-e^{-(1/2)\cdot x^2} + e^{-8}) \\
&= 1 - e^{-(1/2)x^2+8}
\end{aligned}
$$

To be able to draw from $g(x)$ we find the inverse of $G(x)$:

$$
\begin{aligned}
y &= 1 - exp(-(1/2)x^2 + 8) \\
\rightarrow epx(-(1/2)x^2 + 8) &= 1 - y \\
\rightarrow -(1/2)x^2 + 8 &= ln(1-y) \\
\rightarrow x^2 &= -2(ln(1-y) - 8) \\
\rightarrow x &= sqrt(-2ln(1-y) + 16)
\end{aligned}
$$

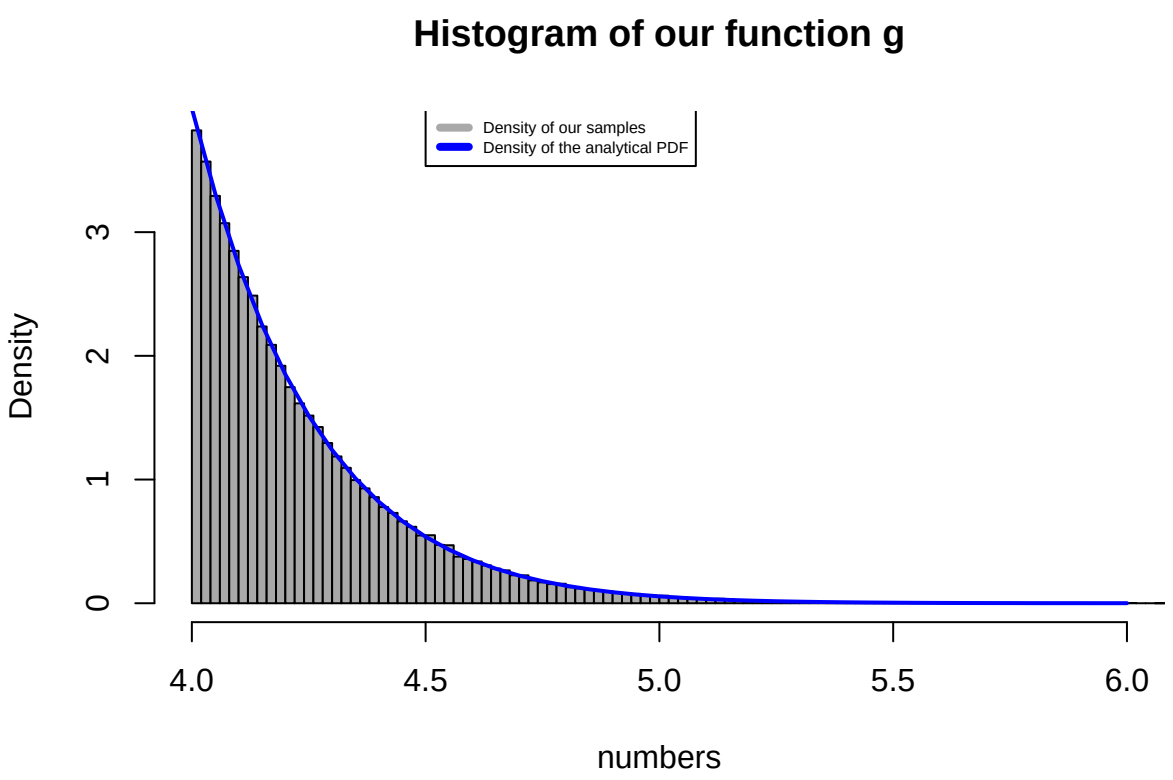
Here i continue by defining the PDF and CDF inverse of g as well as the standard normal. Note that the function "dnorm" is used. This is just the probability density function of the standard normal, so there is no random sampling with this function. This is done such that we can apply importance sampling to the Monte Carlo estimate.

```
set.seed(98)

montecarlo_es_IS <- function(n) {
    # The function that the task asks for (MC with
    # Important S) here n is samples

    # Here we have our different functions (R -> R
    # functions) that we need
    g_CDFinverse <- function(x) {
        sqrt(-2 * log(1 - x) + 16)
    }  #G CDF inverse
    g_PDF <- function(x) {
        (0 + (x > 4)) * x * exp(-(1/2) * x^2 + 8)
    }  #g PDF
    hw_func <- function(x) {
        (0 + (x > 4)) * (dnorm(x, 0, 1))/(g_PDF(x))
    }  #weight * I(x>4)

    vector_of_samples <- hw_func(g_CDFinverse(runif(n)))  #sampling from g and evaluati
    # weights and the indicator function

    Important_es <- mean(vector_of_samples)  #Calculating mean (aka MC)
    Important_var <- var(vector_of_samples)  #The variance for conf interval
```

```
    return(c(Important_es, Important_var))
}

N <- 1e+05
Data <- montecarlo_es_IS(N)
Important_es <- Data[1]
Important_var <- Data[2]
var_conf <- Important_var/N
Important_conf <- c(Important_es - 1.962 * sqrt(Important_var/N),
    Important_es + 1.96 * sqrt(Important_var/N))  #Conf interval
```

We will let the task be the test for the function itself. We expect that the estimate is similar, however the variance to be reduced. To get an indicator if the sampling works we will make a histogram of samples for the function g and graph the function itself besides.

```
set.seed(98)

# Dropped the Identity function here (it made the graph
# weird?)
g_PDF_1 <- function(x) {
    x * exp(-(1/2) * x^2 + 8)
}
numbers_1 <- seq(4, 6, by = 0.05)
evaluete_func <- g_PDF_1(numbers_1)

g_CDFinverse <- function(x) {
    sqrt(-2 * log(1 - x) + 16)
}
numbers <- g_CDFinverse(runif(1e+05))

hist(numbers, main = paste("Histogram of our function g"), freq = F,
    breaks = 100, xlim = c(4, 6), col = "darkgray")
lines(numbers_1, evaluete_func, col = "blue", lwd = 2)
legend(4.5, 4, c("Density of our samples", paste("Density of the analytical PDF")),
    col = c("darkgray", "blue"), lty = c(1, 1), lwd = c(4, 4),
    cex = 0.5)
```



As we can see, the sampling from g does resemble the probability density function itself.

The Monte Carlo estimate using important sampling, Variance of the product of the indicator function and weight, and the 95% confidence interval using the empirical variance is given in the matrix bellow:

```
cat("MC Importance sampling:", Important_es, "\nVar of I(x)*w(x):      ",
    Important_var, "\nConfidence interval:   ", "(", Important_conf[1],
    ",", Important_conf[2], ")")
```

```
## MC Importance sampling: 3.166897e-05
## Var of I(x)*w(x):       2.425843e-12
## Confidence interval:    ( 3.16593e-05 , 3.167862e-05 )
```

We can clearly observe that our estimate is much better with importance sampling compared to without. Here we do not get a 95% confidence interval that goes into the negative numbers.

Since we are interested in how many samples we need to make of the Monte Carlo estimate of θ to have the same precision of the same estimate, however when importance sampling is applied, we want to solve,

$$\hat{Var}[\frac{1}{R}\sum_{i=0}^{R} I(x_i > 4)] = \hat{Var}[\frac{1}{N}\sum_{i=0}^{N} I(x_i > 4)w(x_i)]$$

for R. In this expression N is given and equal to 100000. We know the value of the right hand side from this task so it simplifies to,

$$\frac{1}{R}\hat{Var}[I(x > 4)] = 2.453573e - 17$$

```
Important_var/N
```

```
## [1] 2.425843e-17
```

The ideal way of finding R would be to increase R until we got equality. Howevewr by doing this we have to generate new samples of a standard normal. This turns out to be impractical since R is many times greater than N, and computing that many standard normal samples will take way to long. Therefor we will solve the following equation:

$$\frac{1}{R}(\hat{Var}_{n=10000}[I(x > 4)]) = 2.453573e - 17$$

Here the empirical variance is of the 10000 samples, that we have already calculated. Thus we end up with,

$$R = \lceil(\frac{\hat{Var}_{n=10000}[I(x > 4)]}{2.453573e - 17})\rceil$$

which is the value:

```
ceiling((h_var)/(Important_var/N))
```

```
## [1] 1.236659e+12
```

3.

(a) Creating a modified function that generates n pairs of antithetic varieties:

```
set.seed(98)

montecarlo_es_IS_nc <- function(n) {
    # modified version, input is n pairs

    # Here we have our different functions (R -> R
```

```
    # functions) that we need
    g_CDFinverse <- function(x) {
        sqrt(-2 * log(1 - x) + 16)
    }  #G CDF
    g_PDF <- function(x) {
        (0 + (x > 4)) * x * exp(-(1/2) * x^2 + 8)
    }  #g PDF
    hw_func <- function(x) {
        (0 + (x > 4)) * (dnorm(x, 0, 1))/(g_PDF(x))
    }  #weight * I(x>4)

    uniforms <- runif(n)
    uniforms_nc <- c(uniforms, 1 - uniforms)

    vector_of_samples_nc <- hw_func(g_CDFinverse(uniforms_nc))  #sampling from g and ev
    # in weights and the indicator function

    Important_es_nc <- mean(vector_of_samples_nc)  #Calculating mean (aka MC)

    Important_var_nc <- var(hw_func(g_CDFinverse(uniforms)) +
        hw_func(g_CDFinverse(1 - uniforms)))  #The variance for conf interval. It is co
    return(c(Important_es_nc, Important_var_nc))
}
```

To check that this function works we will look at the result of the task. However, we will also create a smaller part of the code underneath to check that the variables actually become negatively correlated.

```
# Here i have coded a function that we will check works.
# That the negatively correlated variables actually are
# negatively correlated.
sampl_anti <- function(n) {
    uni <- runif(n)
    pairs_unif <- c(uni, 1 - uni)
    return(pairs_unif)
}

g_CDFinverse <- function(x) {
    sqrt(-2 * log(1 - x) + 16)
}  #g function sampling
a <- g_CDFinverse(sampl_anti(50000))  #negatively correlated samples
c(cor(g_CDFinverse(runif(5e+05)), g_CDFinverse(runif(5e+05))),
    cor(a[1:50000], a[50001:1e+05]))
```

```
## [1]  6.766198e-05 -6.866528e-01
```

As we can see there is a clear difference between the non correlated and negatively correlated variables.

(b) We will then estimate θ with 50000 pairs to get 100000 variables where there are pairwise negatively correlated to find an estimate with less variance. This will be a fair comparison of the two methods previously discussed since we use the same amount of draws. To find the variance of the negatively correlated functions we have to show a result first. If we let Y and X be correlated variables form the distribution in (C.2), we can derive the variance,

$$Var[\frac{1}{N}(\sum_{i=1}^{N/2} w(X_i)I(X_i > 4))+\sum_{i=1}^{N/2} w(Y_i)I(Y_i > 4))] = \frac{1}{N^2}Var[\sum_{i=1}^{N/2} w(X_i)I(X_i > 4)+w(Y_i)I(Y_i > 4)]$$

Here we have partitioned the sum such that the negatively correlated variables stand together. Thus the summand of the sum are independent variables and we can exploit this:

$$\begin{aligned}\frac{1}{N^2}Var[\sum_{i=1}^{N/2} w(X_i)I(X_i > 4) + w(Y_i)I(Y_i > 4)] &= \frac{N/2}{N^2}Var[w(X)I(X > 4) + w(Y)I(Y > 4)] \\ &= \frac{1}{2N}Var[w(X)I(X > 4) + w(Y)I(Y > 4)]\end{aligned}$$

The last line here is what i will use for the code. However if we continue the derivation we can observe that negatively correlating our variables pairwise does indeed lower the variance. Here i have used that the variance of $w(X)I(X > 4)$ and $w(Y)I(Y > 4)$ are equal:

$$\begin{aligned}\Rightarrow &= \frac{1}{2N}Var[w(X)I(X > 4) + w(Y)I(Y > 4)] \\ &= \frac{1}{2N}(Var[w(X)I(X > 4)] + Var[w(Y)I(Y > 4)] + 2Cov[w(X)I(X > 4), w(Y)I(Y > 4)]) \\ &= \frac{1}{N}(Var[w(X)I(X > 4)] + Cov[w(X)I(X > 4), w(Y)I(Y > 4)])\end{aligned}$$

We can see that since X and Y are negatively correlated that we will subtract something positive. Hence, lower variance. Implementing this we get the following code bellow.

```
set.seed(98)
n_2 = 50000

generate_the_mean_var <- montecarlo_es_IS_nc(n_2)  #running the function defined

Important_es_anti <- generate_the_mean_var[1]
```

```
Important_var_anti <- generate_the_mean_var[2]
conf_interval_anti <- c(Important_es_anti - 1.962 * sqrt(Important_var_anti/(4 *
    n_2)), Important_es_anti + 1.962 * sqrt(Important_var_anti/(4 *
    n_2)))
```

The Monte Carlo estimate using important sampling whit pairwise negatively correlated variables, the variance of $w(X)I(X > 4) + w(Y)I(Y > 4)$, and the 95% confidence interval is given in the matrix bellow:

```
cat("MC IS NC:           ", Important_es_anti, "\nVar when NC:        ",
    Important_var_anti, "\nConfidence interval:", "(", conf_interval_anti[1],
    ",", conf_interval_anti[2], ")")
```

```
## MC IS NC:            3.167227e-05
## Var when NC:         1.145145e-12
## Confidence interval: ( 3.166757e-05 , 3.167696e-05 )
```

Note that to calculate the variance of the mean of the pairwise correlated variables we end up with,

$$\frac{1}{2N} Var[w(X)I(X > 4) + w(Y)I(Y > 4)]$$

Such that we divide by $2N$ compared to N. This is mentioned to show the there is a difference of the variances presented in (C3) compared to (C2) and (C1).

It might not be clear that using negatively correlated variables does help the estimate. However if we calculate the lengths of the confidence intervals and look at there difference it becomes apparent. Recall that this will be the same as,

$$\frac{\hat{Var}[(1/N)\sum_{i=1}^{N/2} w_i]}{\hat{Var}[(1/N)\sum_{i=1}^{N} v_i]}$$

where w_i represents the sum of the pairwise negatively correlated variables and v_i are not correlated. All are drawn form the same distribution applying important sampling. The result becomes:

```
(Important_conf[2] - Important_conf[1])/(conf_interval_anti[2] -
    conf_interval_anti[1])
```

```
## [1] 2.057286
```

We can now clearly see that the confidence interval is clearly smaller and we have a better estimate of θ and we have evidence for the for the function that generates pairwise negatively variables working properly.

Problem D)

1.

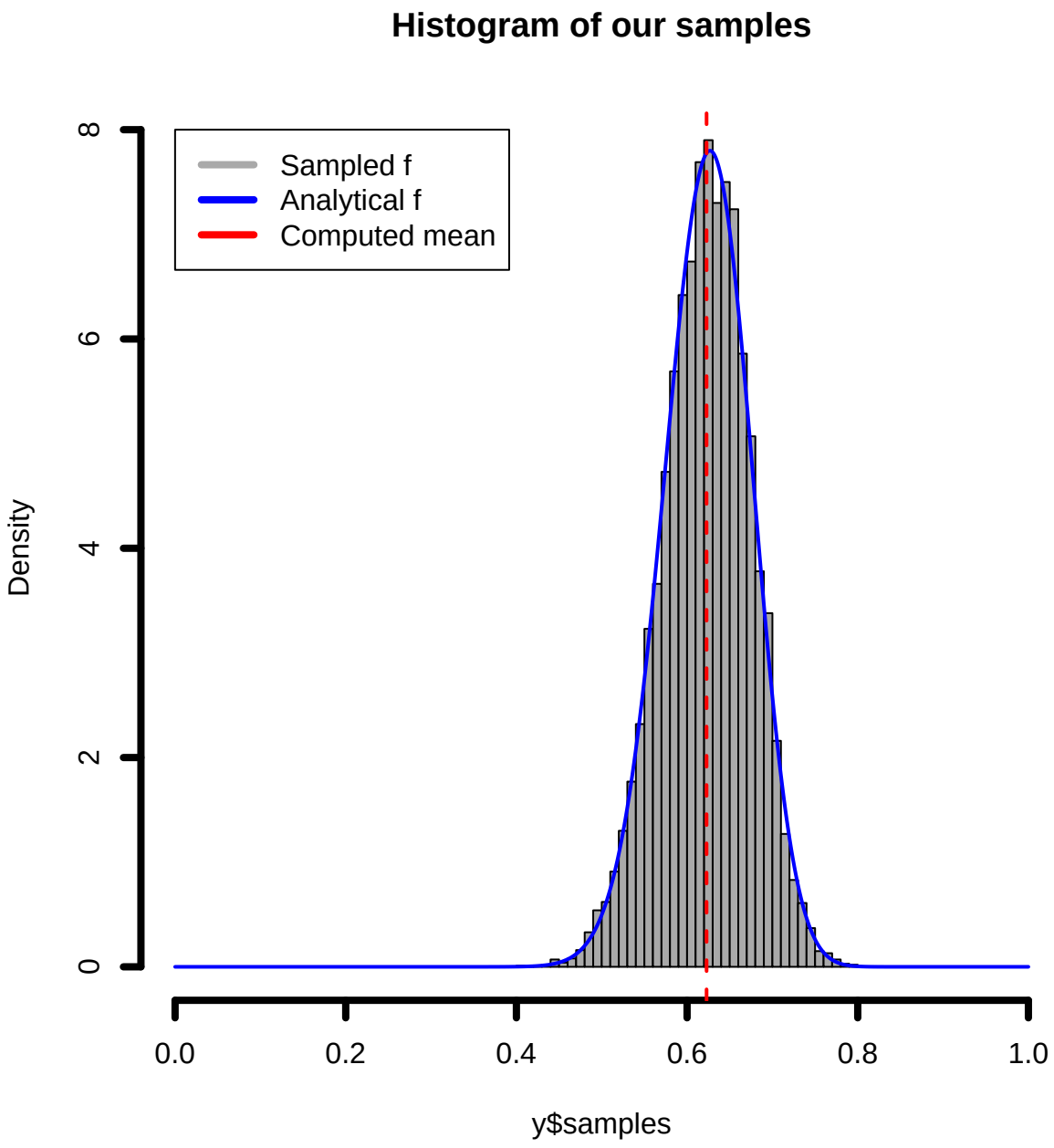
```r
set.seed(98)

# Defining the posterior distribution
f <- function(theta) {
    return((2 + theta)^125 * (1 - theta)^38 * theta^34)
}

# finding the highest value of the posterior distribution
c <- optimize(f, c(0, 1), maximum = T)

rej_samp <- function(n) {
    samples <- rep(NA, n)
    number_it <- 0

    for (i in 1:n) {
        when_done <- 0
        while (when_done == 0) {
            # Prop. sample
            theta_sample <- runif(1)
            # Accept prob.
            alpha <- f(theta_sample)/c$objective
            u <- runif(1)
            # accept if uniform lower than alpha
            if (u <= alpha) {
                samples[i] <- theta_sample
                when_done <- 1
            }
            # Adding iteration
            number_it <- number_it + 1
        }
    }

    return(list(samples = samples, number_iterations = number_it))
}
```

2.

```
n = 10000
# Used to find the mean
find_mean <- function(theta) {
    return(theta * f(theta))
}
# finding normalizing constant
norm_const <- integrate(f, 0, 1)$value

# PDF of posterior
f_density <- function(theta) {
    return(f(theta)/norm_const)
}
y <- rej_samp(n)
cat("Computed mean: ", sum(y$samples)/n, "\nTheoretical mean: ",
    integrate(find_mean, 0, 1)$value/norm_const)
```

```
## Computed mean:  0.6228497
## Theoretical mean:  0.6228061
```

```
hist(y$samples, main = "Histogram of our samples", freq = F,
    col = "darkgray", lwd = 4, xlim = c(0, 1), ylim = c(0, 8),
    breaks = 30)
lines(x = seq(0, 1, length.out = n), f_density(seq(0, 1, length.out = n)),
    col = "blue", lwd = 2)
abline(v = sum(y$samples)/n, col = "red", lwd = 2, lty = 2)
legend(0, 8, c("Sampled f", "Analytical f", "Computed mean"),
    col = c("darkgray", "blue", "red"), lty = c(1, 1), lwd = c(4,
        4))
```



See that our samples fit the theoretical function well, and the estimated means are similar, so our function is working properly

3.

The expected number of random numbers our rejection sampling alg. has to make is found using the formula:

$$c \geq \frac{f(x)}{g(x)}$$

where we find the smallest c possible for the algorithm to be as efficient as it can be. We know that $g(x) = U(0,1) = 1$. So:

$$\begin{aligned} c &= max\left(\frac{f(x)}{g(x)}\right) \\ &= max(f(x)) \end{aligned}$$

```
cat("Average random numbers generated: ", y$number_iterations/n,
    "\nTheoretical expected number: ", c$objective/norm_const)
```

```
## Average random numbers generated:  7.845
## Theoretical expected number:  7.799308
```

We find that the average random numbers generated by our function and the theoretical expected numbers of random numbers are similar, so our function is working.

4.

Now we want to use $Beta(1,5) = \frac{\Gamma(1)\Gamma(5)}{\Gamma(6)}$ as our prior instead of $Beta(1,1)$ that we used in D:1-3. So the prior distribution becomes $\frac{1}{Beta(1,5)}(1-\theta)^4\theta^0 = \frac{\Gamma(6)}{\Gamma(1)\Gamma(5)}(1-\theta)^4\theta^0$

Our new posterior density then becomes

$$\begin{aligned} f_{new}(\theta|y) \propto f(y|\theta) \cdot Beta(1,5) &= (2+\theta)^{125}(1-\theta)^{38}\theta^{34} \cdot \frac{\Gamma(6)}{\Gamma(1)\Gamma(5)}(1-\theta)^4\theta^0 \\ &= (2+\theta)^{125}(1-\theta)^{42}\theta^{34}\frac{\Gamma(6)}{\Gamma(1)\Gamma(5)} \end{aligned}$$

In the following calculations and code, we will use our old posterior density from D:1-3, $f_{old}(\theta|y) \propto (2+\theta)^{125}(1-\theta)^{38}\theta^{34}$, as our proposal density to improve the accuracy.

The importance sampling posterior mean is given by $\hat{\mu}_{IS} = \frac{1}{n}\sum_{i=1}^{n}\theta_i w_i$, where the w_i's are given by

$$w_i = \frac{f_{new}(\theta|y)}{f_{old}(\theta|y)} \propto \frac{\Gamma(6)}{\Gamma(1)\Gamma(5)}(1-\theta)^4$$

which is just the density of the Beta distribution.

```r
target_post <- function(theta) {
    return(f_density(theta) * dbeta(theta, 1, 5))
}

new_norm_const <- integrate(target_post, 0, 1)$value

target_post_normz <- function(theta) {
    return(target_post(theta)/new_norm_const)
}

target_post_mean <- function(theta) {
    return(theta * target_post_normz(theta))
}

w <- target_post_normz(y$samples)/f_density(y$samples)

cat("Importance sampling posterior mean:", sum(y$samples * w)/n,
    "\nTheoretical mean:", integrate(target_post_mean, 0, 1)$value)
```

```
## Importance sampling posterior mean: 0.5959028
## Theoretical mean: 0.5959316
```

We find that our algorithm creates a mean similar to the theoretical, so the bias introduced by the new samples are small.