Compulsory exercise 1: Group 5

TMA4300 Computational Statistics V2022

Markus Johnsen Aase, Nora Ræhnebæk Aasen

10 februar, 2022

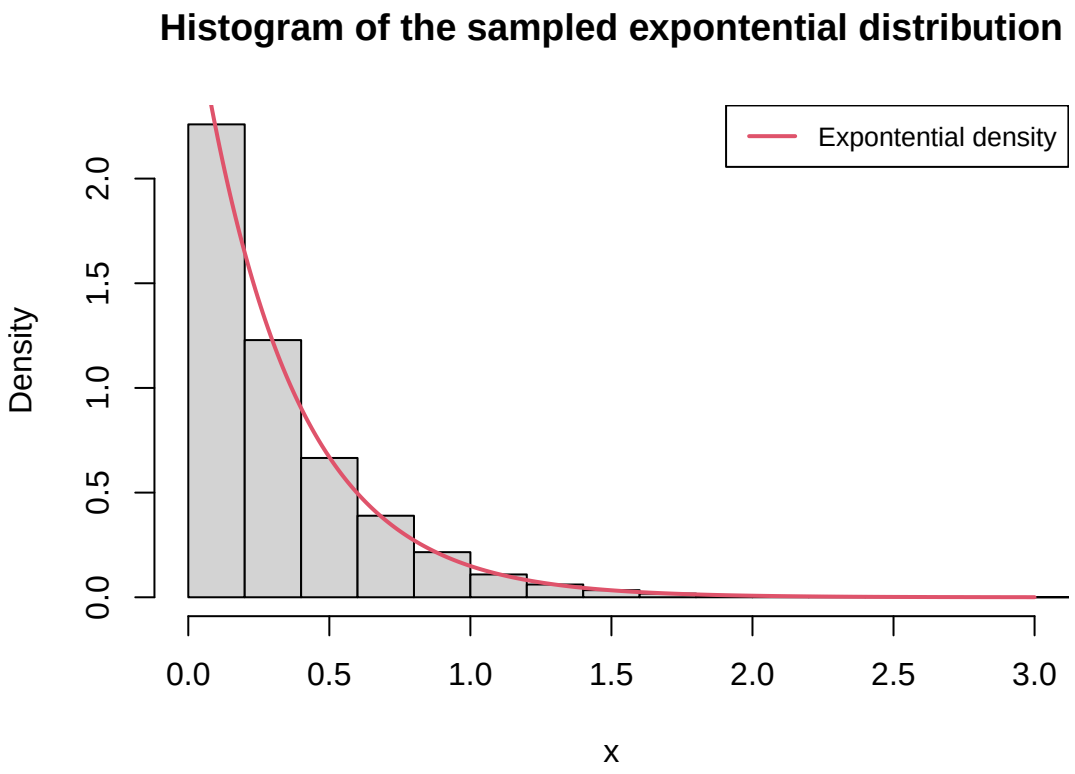
Problem A

1) Random samples from the exponential distribution

We want to generate samples from an exponential distribution with rate parameter λ, i.e. $X \sim \lambda e^{-\lambda x}$. This can be done using the inverse of $F(x) = 1 - e^{-\lambda x}$, which is $F^{-1}(y) = \frac{-\log(1-y)}{\lambda}$.

```
# A function that draws from an exponential distribution
# n: number of samples
# lambda: rate parameter

ranexp <- function(n,lambda=1){
  U = runif(n)              # Sample from U[0,1]
  V = -(log(1-U))/lambda    # Use the inverse of the cumulative exponential distribution
  return(V)                 # Return a vector of n independent samples from an exponential distribution
}
```

```
# Check that our function behaves as expected
x = ranexp(10000, lambda=3)

hist(x, freq = F, n=15, xlim = c(0,3), xlab = 'x', ylab = 'Density', main = 'Histogram of the sampled e
lines(seq(0,3,0.001), dexp(seq(0,3,0.001),rate = 3), col = 2, lwd = 2)
legend('topright',legend = ('Expontential density'), col = 2, lwd = 2, bty = 'l',cex = 0.8)
```



We see that the density matches our histogram, indicating that our function is successful at drawing random samples from an exponential distribution.

2)

For this problem we will draw from a density g given by

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

We start by calculating the normalizing constant c. Solving

$$\int_0^1 cx^{\alpha-1}dx + \int_1^\infty ce^{-x}dx = 1$$

yields $c = \frac{1}{\alpha^{-1}+e^{-1}}$.

Secondly, when calculating the cumulative distribution we have to split it into two cases: when $x \in (0,1)$ and $x \in [1,\infty)$.

$$G_1(x) = \int_0^x cs^{\alpha-1}ds = \frac{c}{\alpha}x^\alpha.$$

$$G_2(x) = c\left(\int_0^1 s^{\alpha-1}ds + \int_1^x e^{-x}ds\right) = c\left(\frac{1}{\alpha} + \frac{1}{e} - e^{-x}\right) = 1 - ce^{-x}.$$

Hence, the cumulative distribution is given by

$$G(x) = \begin{cases} \frac{c}{\alpha}x^{\alpha} & 0 < x < 1, \\ 1 - ce^{-x} & x \geq 1, \\ 0 & \text{otherwise.} \end{cases}$$

In order to find the inverse we must find the function $G^{-1}(x)$ such that $G(x) = y \implies x = G^{-1}(y)$. Note that this function will be different before and after the point $G(1)$. Calculating yields

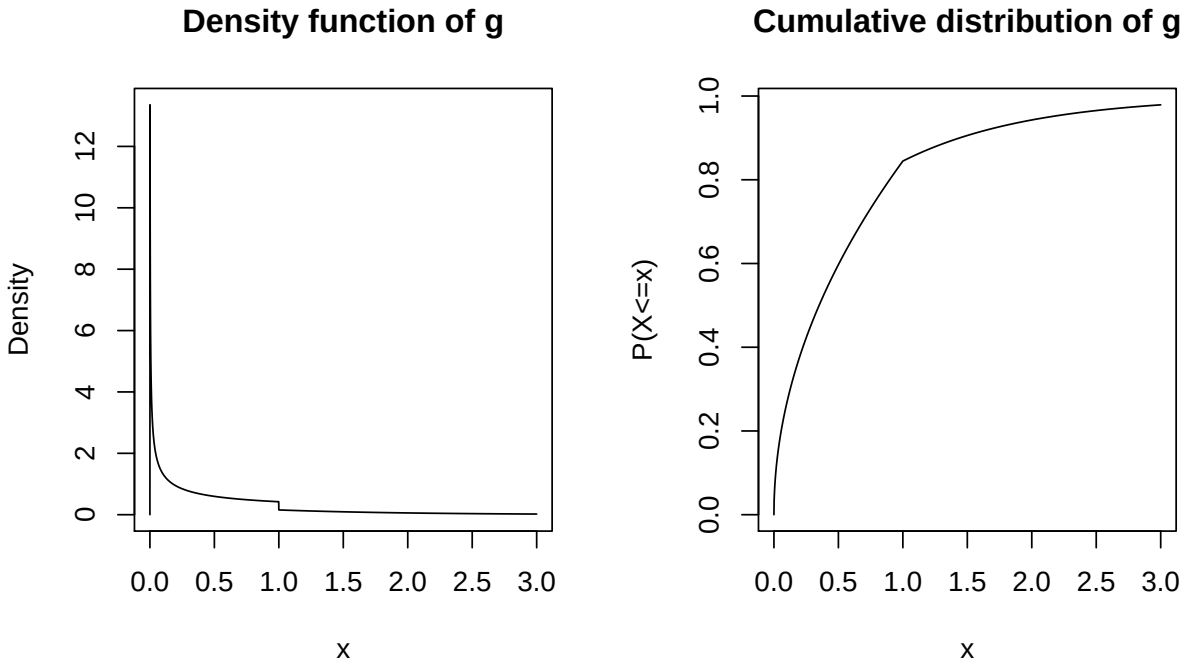
$$G^{-1}(y) = \begin{cases} (\frac{\alpha}{c}y)^{1/\alpha} & 0 < y < G(1), \\ -\log(\frac{1-y}{c}) & y \geq G(1), \\ 0 & \text{otherwise.} \end{cases}$$

```
# for all functions we first define a function that takes in one variable, x,
# and secondly a function that takes in a vector v.


# define the density function g
# x: a real number
# alpha: a real parameter-value in (0,1)

g1 <- function(x,alpha = 0.5){
 c = (alpha^(-1)+exp(-1))^(-1) # define the normalizing constant
 if(0<x & x<1){                # The if/else checks which function to apply
                               # depending on which domain x is in
   return(c*x^(alpha -1))
 }
 else if(x >= 1){
   return(c*exp(-x))
 }
 else if(x <= 0){
   return(0)
 }
}

# v: a vector of real numbers
# alpha: a real parameter-value in (0,1)

den_g <- function(v,alpha = 0.5){
  y = sapply(v,g1,alpha = alpha) # apply the function g1 to the vector v
  return(y)   # return a vector consisting of all elements of v evaluated in g1
}


# the cumulative distribution of g
# x: a real number
# alpha: a real parameter-value in (0,1)
```

```r
g2 <- function(x,alpha = 0.5){
  c = (alpha^(-1)+exp(-1))^(-1) # normalizing constant
  if(0<x & x<1){                # if/else to check which domain x lives in
    return((c/alpha)*x^(alpha))
  }
  else if(x >= 1){
    return(c*((1/alpha)+exp(-1)-exp(-x)))
  }
  else if(x <= 0){
    return(0)
  }
}

# v: a vector of real numbers
# alpha: a real parameter-value in (0,1)

cum_g <- function(v,alpha = 0.5){
  y = sapply(v,g2,alpha = alpha)
  return(y)
}

# the inverse of the cumulative distribution of g,
# defined in a similar matter as the previous two
# x: a real number
# alpha: a real parameter-value in (0,1)

g3 <- function(x,alpha = 0.5){
  c = (alpha^(-1)+exp(-1))^(-1)
  b = g2(1, alpha = alpha)
  if(0<x & x<b){
    return(((alpha/c)*x)^(1/alpha))
  }
  else if(x >= b){
    return(-log((1-x)/c))
  }
  else if(x <= 0){
    return(0)
  }
}

# v: a vector of real numbers
# alpha: a real parameter-value in (0,1)

inv_g <- function(v,alpha = 0.5){
  y = sapply(v,g3,alpha = alpha)
  return(y)
}

# A function that draws from the distribution g
# n: number of samples
# alpha: real-valued parameter in (0,1)

rang <- function(n,alpha=0.5){
```

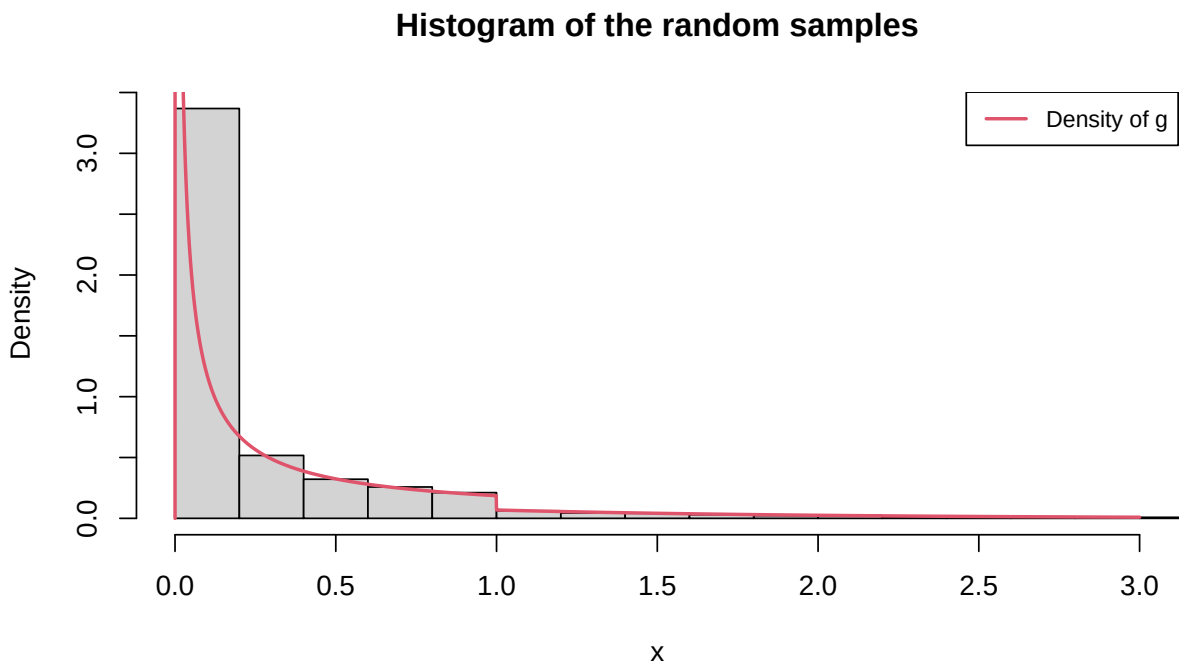
```
  U = runif(n)
  V = inv_g(U, alpha = alpha) # apply inv_g on the uniform
                              # random vector to simulate n samples from g
  return(V)
}
```

```
# Test that our functions behave as expected

# set the variables
alpha = 0.2
a = 0
b = 3
l = seq(a,b,0.001) # x-axis in the plot

# Check that the density and cumulative function are reasonable
par(mfrow = c(1,2))

plot(l,den_g(l), type = 'l', main = 'Density function of g', xlab = 'x', ylab = 'Density')

plot(l,cum_g(l), type = 'l', main = 'Cumulative distribution of g', xlab = 'x', ylab = 'P(X<=x)')
```




```
par(mfrow = c(1,1))

# Generate random samples from g
g_sample = rang(10000, alpha = alpha)

# Plot a histogram of the drawn samples and compare with the density function
hist(g_sample, freq = F, n=50, xlim = c(a,b), xlab = 'x', ylab = 'Density', main = 'Histogram of the ra
```

```
lines(l, den_g(l,alpha=alpha), col = 2, lwd = 2)
legend('topright',legend = ('Density of g'), col = 2, lwd = 2, bty = 'l',cex = 0.8)
```



We first the plot the density and cumulative distrribution to get familiar with the functions. We clearly see the breaking point at $x = 1$. The histogram appear consistent with the density and the function seem successful in simulating from g.

3)

We wish to sample from the function

$$f(x) = \frac{ce^{\alpha x}}{(1+e^{\alpha x})^2}, \quad -\infty < x < \infty.$$

We first calculate the normalizing constant c by solving

$$\int_{-\infty}^{\infty} \frac{e^{\alpha x}}{(1+e^{\alpha x})^2} dx = \left[\frac{e^{\alpha x}}{\alpha(1+e^{\alpha x})}\right]_{-\infty}^{\infty} = \frac{1}{c},$$

which yields $c = \alpha$. Then, we find the cumulative distribution and inverse cumulative similarly as in the previous problem.

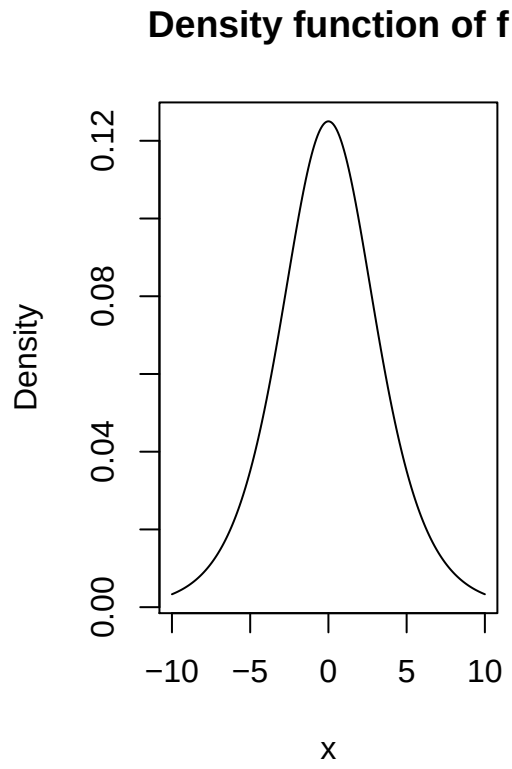
$$F(x) = \int_{-\infty}^{x} \frac{\alpha e^{\alpha s}}{(1+e^{\alpha s})^2} ds = \left[\frac{e^{\alpha s}}{1+e^{\alpha s}}\right]_{-\infty}^{x} = \frac{e^{\alpha x}}{1+e^{\alpha x}}.$$

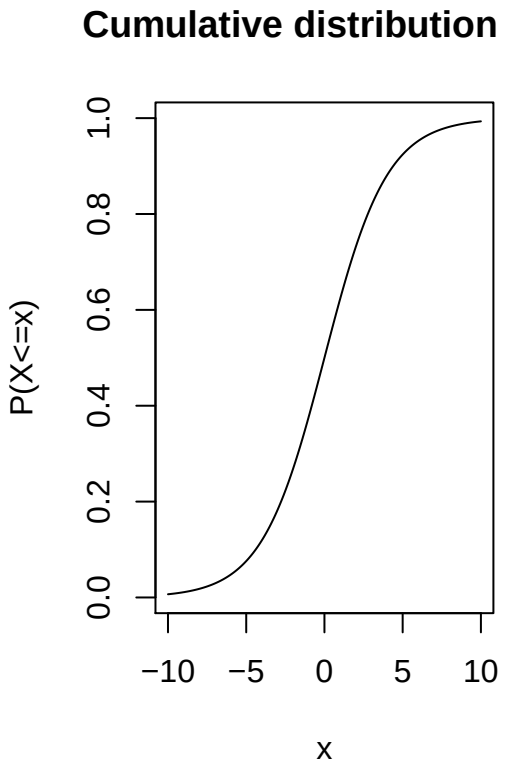
$$F^{-1}(y) = \frac{1}{\alpha} \log\left(\frac{1}{\frac{1}{y} - 1}\right).$$

```r
# As in the previous exercise we first define the functions for one variable,
# then apply it to a vector v in a separate function

# The density of f
# x: a real number
# alpha: a real parameter-value greater than 0

f1 <- function(x, alpha = 0.5){
  return((alpha*exp(alpha*x))/(1+exp(alpha*x))^2)
}

# v: a vector of real numbers
# alpha: a real parameter-value greater than 0

den_f <- function(v,alpha = 0.5){
  y = sapply(v,f1,alpha = alpha)
  return(y)
}

# The cumulative distribution of f
# x: a real number
# alpha: a real parameter-value greater than 0
f2 <- function(x, alpha = 0.5){
  return(exp(alpha*x)/(1+exp(alpha*x)))
}

# v: a vector of real numbers
# alpha: a real parameter-value greater than 0

cum_f <- function(v,alpha = 0.5){
  y = sapply(v,f2,alpha = alpha)
  return(y)
}

# The inverse cumulative of f
# x: a real number
# alpha: a real parameter-value greater than 0

f3 <- function(x, alpha = 0.5){
  return((1/alpha)*log(1/((1/x)-1)))
}

# v: a vector of real numbers
# alpha: a real parameter-value greater than 0

inv_f <- function(v,alpha = 0.5){
  y = sapply(v,f3,alpha = alpha)
  return(y)
}
```

```r
# A function that draws from the distribution f
# n: number of samples
# alpha: real-valued parameter greater than 0
ranf <- function(n,alpha=0.5){
  U = runif(n)
  V = inv_f(U, alpha = alpha)
  return(V)
}
```

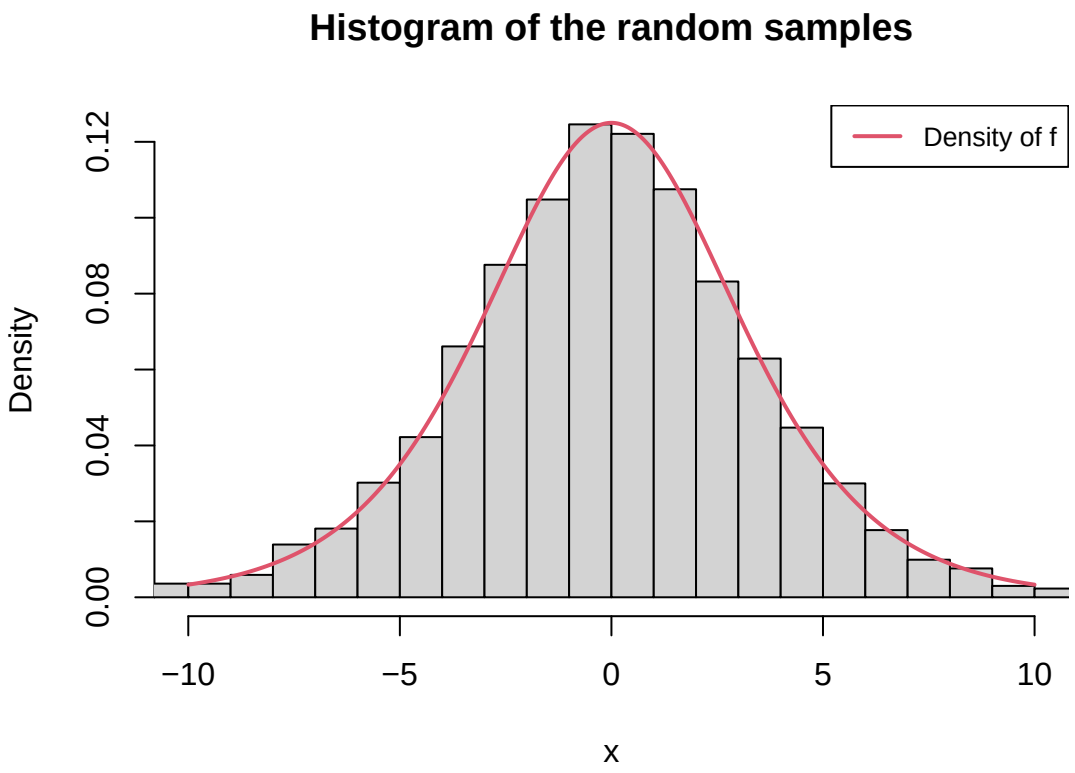
```r
# set the variables
alpha = 0.5
a = -10
b = 10
l = seq(a,b,0.01) # x-axis in plot

# Check that the functions are reasonable
par(mfrow = c(1,2))

plot(l,den_f(l), type = 'l', main = 'Density function of f', xlab = 'x', ylab = 'Density')

plot(l,cum_f(l), type = 'l', main = 'Cumulative distribution of f', xlab = 'x', ylab = 'P(X<=x)')
```






```r
par(mfrow = c(1,1))

# Generate random samples from f
```

```r
x = ranf(10000, alpha = alpha)

# Plot a histogram of the drawn samples and compare with the density function
hist(x, freq = F, n=50, xlim = c(a,b), xlab = 'x', ylab = 'Density', main = 'Histogram of the random sa
lines(l, den_f(l,alpha=alpha), col = 2, lwd = 2)
legend('topright',legend = ('Density of f'), col = 2, lwd = 2, bty = 'l',cex = 0.8)
```



Again, we first plot the density and cumulative distribution to get a visualization of the function we work with. From the histogram it appears that our function is successful at drawing samples from the density f.

4)

We wish to generate random variables X that are standard normally distributed, i.e. $X \sim \mathcal{N}(0,1)$. This is done using the Box-Muller algorithm. First, n samples are drawn from a standard uniform distribution and stored in a vector U which is then scaled by 2π. Since the uniform distribution is a scale family, we get $U \sim Unif[0, 2\pi]$. Secondly, we draw n samples from an exponential distribution with $\lambda = 0.5$ and store this in a vector V. Since U and V are independent, their joint distribution is given by

$$f_{U,V} = \frac{1}{2\pi} 2e^{-\frac{v}{2}}.$$

If we now choose $V = X^2 + Y^2$ and $U = \tan^{-1}(\frac{Y}{X})$, we get $|J| = -2$. Then,

$$f_{U,V}(u,v) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} = f_X(x) f_Y(y),$$

which we recognize as the joint distribution of two independent standard normal variables.

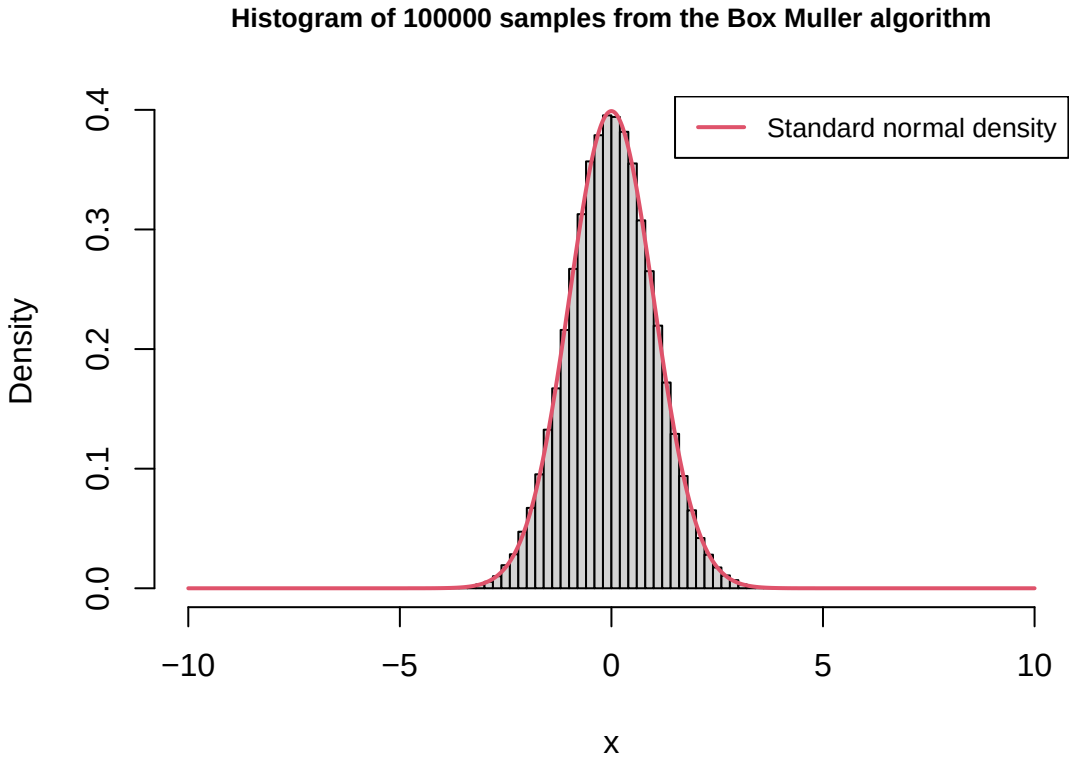
$$
\begin{aligned}
X^2 + Y^2 &= V \\
X^2 + X^2 \tan^2(U) &= V && \text{since} \quad \frac{Y}{X} = \tan(U) \\
X^2 \sec^2(U) &= V && \text{since} \quad 1 + \tan^2(U) = \sec^2(U) \\
X &= \sqrt{V}\cos(U) && \text{since} \quad \sec^2(U) = \frac{1}{\cos^2(U)}
\end{aligned}
$$

and similarly that $Y = \sqrt{V}\sin(U)$.

```
# the Box Muller sampling algorithm generates a vector of
# independent standard normally distributed values
# n: number of samples
BoxMuller <- function (n){
  u <- 2*pi*runif(n)              # n Unif[0,2*pi] random variables stored in a vector
  v <- ranexp(n, lambda = 0.5)    # n exp(1/2) random variables stored in a vector
  x <- sqrt(v) * cos(u)           # return a sample from a normal distribution
  return(x)
}
```

To check if we have implemented Box-Muller correctly we check the properties of 100000 samples.

```
# store 100000 samples in a vector called 'list'
list = BoxMuller(100000)

par(cex.main = 0.85)
# Plot a histogram of the drawn samples and compare with the density function
hist(list, freq = F, n=50, xlim = c(a,b), xlab = 'x', ylab = 'Density', main = 'Histogram of 100000 sam
lines(l, dnorm(l), col = 2, lwd = 2)
legend('topright',legend = ('Standard normal density'), col = 2, lwd = 2, bty = 'l',cex = 0.8)
```



The histogram show that our samples appear to be standard normal. The mean is -0.0032054 and the variance is 0.9973727, which is reasonably close to 0 and 1, respectively, as they should be.

Next we test the quantiles of our simulated standard normal values up against the theoretical quantiles at 0.1, 0.05 and 0.025.

```
# The different quantiles for normal distribution ( alpha = 0.1, 0.05, 0.025)
q = matrix(0, nrow = 2, ncol = 3)                       # make an empty matrix
q[1, ] = quantile(list, probs = c(0.9, 0.95, 0.975))    # calculate the quantiles from the sample
q[2, ] = qnorm(c(0.1, 0.05, 0.025),lower.tail = F)      # store the true quantile-values


colnames(q) = c(' alpha = 0.1', 'alpha = 0.05','alpha = 0.025')
rownames(q) = c('Estimated quantiles','Theoretical quantiles')


# present in a table with specified row- and columnnames
q
```

```
##                       alpha = 0.1 alpha = 0.05 alpha = 0.025
## Estimated quantiles      1.278591     1.637952      1.950646
## Theoretical quantiles    1.281552     1.644854      1.959964
```

As we can see, the theoretical quantiles and estimated quantiles match very well, and our algorithm appear to work as intended.

5)

We use Cholesky decomposition to generate normal realizations, and use the Box-Muller algortihm from the previous task to generate univariate standard normal variables.

```
# multi_func takes in the mu vector and covariance matrix and
# an epsilon value. To be used for Cholesky decomp.
# mu: mu vector of a given dimension d.
# cov: square covariance matrix (d x d)
# epsilon: small value to be used for adding small pertubation in Cholesky decomp.

multi_func <- function(mu, cov, epsilon = 0.0001){
  dim = length(mu)                        #find dimensions of mu-vector
  eigenvalues = eigen(cov)                #eigenvalues of covariance matrix.
  cov_mat = cov + epsilon*diag(dim)       #adding a small perturbation

  Cholesky_mat <- chol(cov_mat)           #Choleski factorization of the real
                                          #symmetric positive-definite square matrix.
  Cholesky_mat_T = t(Cholesky_mat)        #transposing it

  return(Cholesky_mat_T)                  #returning the transposed matrix for further use.
}
```

Here we implement the Box Muller to create realizations from standard normal.

We then go on to define the value x, as

$$\mathbf{x} = \mu + LR,$$

where:

$x \sim \mathcal{N}_n(\mu, \Sigma)$,

μ is an n-dimensional vector,

L is an $d \times d$ Cholesky matrix,

and R is an $d \times n$ matrix, where each column r is a standard multivariate normal vector, i.e. $r \sim \mathcal{N}_d(0, I)$.

```
# define a function that generates n samples of independent d-dimensional standard normal vectors
# n: number of samples
# d: dimension of each sample

stdmultinorm <- function(n,d){
  M = matrix(0,nrow = d, ncol = n)  # define an empty matrix for storing
  for (i in (1:n)){
    M[,i] = BoxMuller(d)            # let each column consist of a d-dimensional
                                    # standard normal vector, and generate n vectors
  }
  return(M)
}
```

Since we want independent vectors, where each vector is $\sim \mathcal{N}_d(0, I)$, we can generate d standard normal variables and store them in a vector. There is no covariance that need to be accounted for.

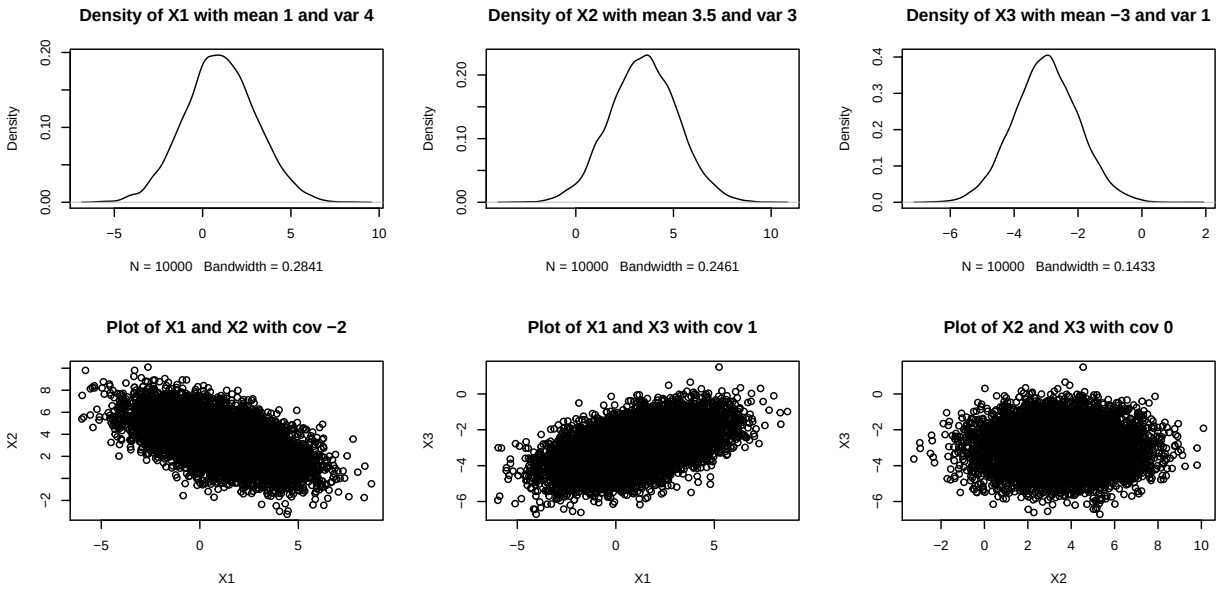
```r
# define a function multinorm that generates n samples from a normal distribution
# n: the number of samples
# mu: a given mean vector
# cov: a given covariance matrix

multinorm <- function(n,mu,cov){
  d = length(mu)                # set the dimensions of each sample
  C_mat = multi_func(mu,cov)    # calculate the Cholesky matrix
  r = stdmultinorm(n,d)         # sample n standard normal vectors of dimension d
  x = mu + C_mat  %*% r         # calculate a random normal vector with given mean and covariance
  return(x)
}

# Test for 3 dimensions
mu_vec = c(1, 3.5,-3)
cov_mat_0 = matrix(c(4, -2, 1, -2, 3, 0, 1, 0, 1), nrow = 3) # must be positive definite

x = multinorm(10000,mu_vec, cov_mat_0)

apply(x, 1, mean)
```

```
## [1]  1.012520  3.467867 -3.005480
```

```r
cov(t(x))
```

```
##           [,1]        [,2]        [,3]
## [1,]  4.038969 -1.96819660  1.05496993
## [2,] -1.968197  2.97737469 -0.02521539
## [3,]  1.054970 -0.02521539  1.02988265
```

```r
par(mfrow = c(2,3), cex = 0.6)
plot(density(x[1,]), main = "Density of X1 with mean 1 and var 4")
plot(density(x[2,]), main = "Density of X2 with mean 3.5 and var 3")
plot(density(x[3,]), main = "Density of X3 with mean -3 and var 1")
plot(x[1,],x[2,], main = "Plot of X1 and X2 with cov -2", xlab = 'X1', ylab = 'X2')
plot(x[1,],x[3,], main = "Plot of X1 and X3 with cov 1", xlab = 'X1', ylab = 'X3')
plot(x[2,],x[3,], main = "Plot of X2 and X3 with cov 0", xlab = 'X2', ylab = 'X3')
```



As we can see, the mean and covariance are close to what we set. Furthermore, the plots show that the density of each sample appear normally distributed, and the bivariate plots seem consistent with what we would expect from a normal distribution.

Problem B

1)

We wish to sample from a gamma distribution, with parameters $\alpha \in (0,1)$ and $\beta = 1$. This can be done using rejection sampling with the proposal distribution g from section A.2. First we wish to find a bound D such that $\frac{f(x;\alpha)}{g(x;\alpha)} \leq D(\alpha)$ for all $x > 0$.

For $x \in (0,1)$ we have

$$\frac{f(x)}{g(x)} = \frac{\frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}}{c x^{\alpha-1}} \leq \frac{\frac{1}{\Gamma(\alpha)}}{c} = \frac{1}{\Gamma(\alpha)}(\alpha^{-1} + e^{-1}),$$

since e^{-x} is decreasing on the domain, thus reacing its maximum at $e^0 = 1$.

For $x \geq 1$ we have

$$\frac{f(x)}{g(x)} = \frac{\frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}}{c e^{-x}} \leq \frac{\frac{1}{\Gamma(\alpha)}}{c} = \frac{1}{\Gamma(\alpha)}(\alpha^{-1} + e^{-1}),$$

since $x^{\alpha-1}$ is decreasing on the domain when $\alpha < 1$, thus reaching its maximum at $1^{\alpha-1} = 1$.

The acceptance probability p is then given by

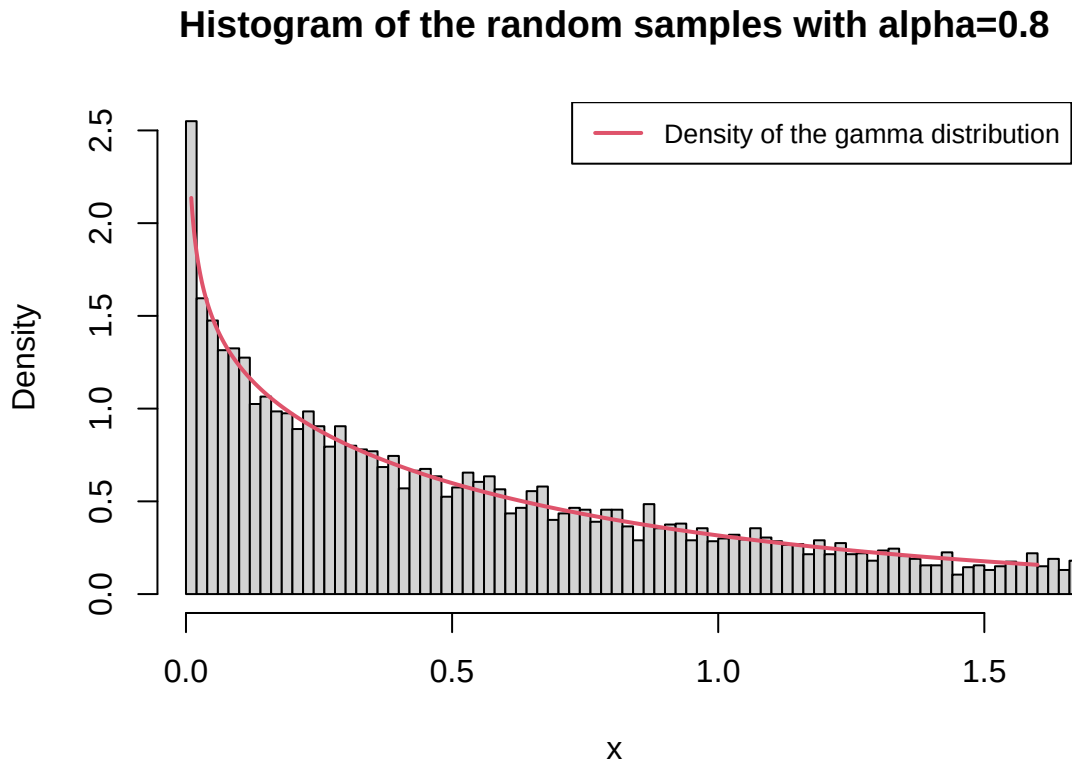
$$p(x;\alpha) = D(\alpha)^{-1} \frac{f(x;\alpha)}{g(x;\alpha)},$$

and the log-acceptance probability is

$$\log(p(x;\alpha)) = -\log(D(\alpha)) + \log(f(x;\alpha)) - \log(g(x;\alpha)).$$

```
# compute constant D (as a function of alpha)
const <- function(alpha){
  c = (alpha^(-1)+exp(-1))        # c is the normalizing constant of g
  return((1/gamma(alpha))*(c))
}
# compute the log of the constant D (as a function of alpha)
logconst <- function(alpha){
  c = (alpha^(-1)+exp(-1))        # c is the normalizing constant of g
  return(- lgamma(alpha) + log(c))
}

# function logg is the logarithm of the density g
# x: a real value
# alpha: real-valued parameter in (0,1)
logg <- function(x,alpha = 0.5){
  c = (alpha^(-1)+exp(-1))^(-1)
  if(0<x & x<1){
    return(log(c)+(alpha -1)*log(x))
  }
  else if(x >= 1){
    return(log(c)-x)
  }
}

# v: a real-valued vector
# alpha: real-valued parameter in (0,1)
den_logg <- function(v,alpha = 0.5){
  y = sapply(v,logg,alpha=alpha)
  return(y)
}

# generate random samples from a gamma distribution with beta=1
# n: number of samples
# alpha: a real-valued parameter on the interval (0,1)
ran_gamma1 = function(n, alpha){
  accepted = numeric(0)
  while(length(accepted) < n){
    # simulate from proposal distribution g
    x  <- rang(n, alpha)
    # generate from the unif[0,1]
    u <- runif(n)

    # compute log accept probability
    log_alpha <- -logconst(alpha)+dgamma(x,alpha,log = T)-den_logg(x,alpha)

    ind <- (u < exp(log_alpha))
    accepted <- c(accepted,x[ind])
  }
  return(list(z = accepted[1:n], alpha = exp(log_alpha)))
}
```

```r
# set the variables
n = 10000
alpha = 0.8
a = 0.01
b = 2*alpha
l = seq(a,b,0.001) # x-axis in plot

par(mfrow=c(1,1))

sample1 = ran_gamma1(n = n, alpha = alpha)
hist(sample1$z, freq = F, n=500, xlim = c(a,b), xlab = 'x', ylab = 'Density', main = 'Histogram of the
lines(l, dgamma(l, shape = alpha), col = 2, lwd = 2)
legend('topright',legend = ('Density of the gamma distribution'), col = 2, lwd = 2, bty = 'l',cex = 0.8)
```



From the plot, it seems that our function work as inteded.

2)

Using Ratio of uniform methods we want to find $a = \sqrt{\sup_x f^*(x)}$, $b_+ = \sqrt{\sup_{x\geq 0} x^2 f^*(x)}$ and $b_- = -\sqrt{\sup_{x\leq 0} x^2 f^*(x)}$, and then set $C_f \subset [0,a] \times [b_-, b_+]$.

This is found by derivating and setting equal to zero, as follows

$$\frac{d}{dx}f^*(x) = (\alpha-1)x^{\alpha-2}e^{-x} - x^{\alpha-1}e^{-x} = x^{\alpha-2}(\alpha-1-x)e^{-x} = 0 \iff x = \alpha - 1$$

and

$$\frac{d}{dx}x^2 f^*(x) = (\alpha+1)x^{\alpha}e^{-x} - x^{\alpha+1}e^{-x} = x^{\alpha}(\alpha+1-x)e^{-x} = 0 \iff x = \alpha + 1$$

in the cases when $x > 0$. We must keep in mind that $f^*(x) = 0$ for $x \leq 0$, and that $\alpha > 1$. From this it is clear that

$a = \sqrt{f^*(\alpha-1)}$,

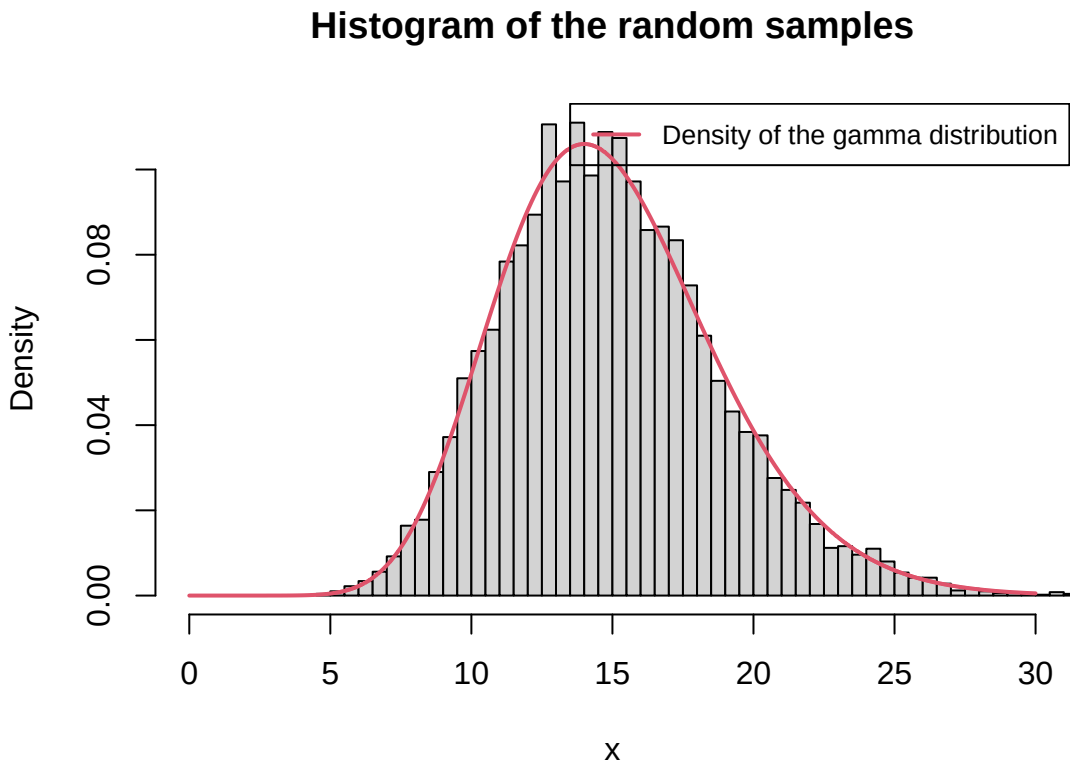
$b_+ = \sqrt{(\alpha+1)^2 f^*(\alpha+1)}$,

$b_- = 0$.

```
# use ratio of uniform methods to generate random samples from a gamma distribution with beta=1
# n: number of samples
# alpha: a real-valued parameter on the interval (1,2000]
# log.scale indicating if the function returns values on log-scale or not
ran_gamma2 <- function(n, alpha, log.scale = T){


  a = (1/2)*((alpha-1)*log(alpha-1)-(alpha-1))                          # log(a)
  b_plus = (1/2)*(2*log(alpha+1)+(alpha-1)*log(alpha+1)-(alpha + 1 )) # log(b+)

  y = numeric(n)    # empty vector to store the accepted values in
  count = 0         # counting how many times the algorithm runs

  for (j in (1:n)){
    i = 0                           # used to 'check' if we have accepted
    while (i == 0){
      count = count + 1

      U_1= a + log(runif(1))        # log(a*Unif[0,1])
      U_2 = b_plus+log(runif(1))    # log(b+*Unif[0,1])
      if (U_1 <= (1/2)*((alpha-1)*(U_2-U_1)-exp(U_2-U_1))){
        y[j] = U_2-U_1
        i = 1
      }
    }
  }
  if (isTRUE(log.scale))
    return(list(y = y, count = count))
  else
    return(list(y = exp(y), count = count))
}
```

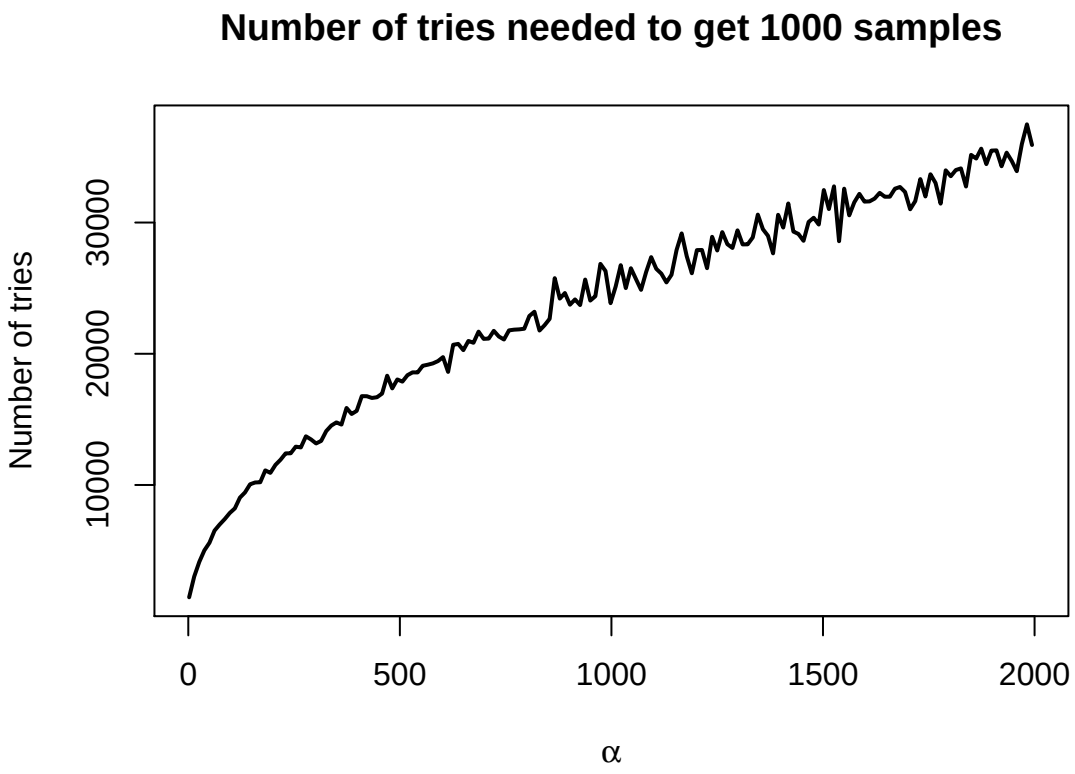
```
# set the variables
n = 10000
alpha = 15

a = 0
b = 2*alpha
```

```
l = seq(a,b,0.001)

hist(ran_gamma2(n,alpha, log.scale = F)$y, freq = F, n=100, xlim = c(a,b), xlab = 'x', ylab = 'Density'
lines(l, dgamma(l, shape = alpha), col = 2, lwd = 2)
legend('topright',legend = ('Density of the gamma distribution'), col = 2, lwd = 2, bty = 'l',cex = 0.8)
```



From the plot it seems reasonable that our function is successful at drawing from a gamma distribution. Next we want to measure the efficiency of the algorithm as α increases.

```
# we want to test how the efficiency of our algorithm relates to our choice of alpha
a = 2           # start at 2 since alpha is strictly greater than 1
b = 2000
l = seq(a,b,12) # check alpha values equal to 2+12n up to 2000


# make a for-loop that runs the random gamma variable generator for each value
# of alpha, and then store the count-value in the vector v
v = numeric(length(l))
for (i in (1:length(l))){
  x = ran_gamma2(1000,l[i],log.scale = T)
  v[i] = x$count
}
# plot l: the values of alpha, against v
plot(l,v,type = 'l', main = 'Number of tries needed to get 1000 samples', xlim = c(0,2000), xlab = expr
```



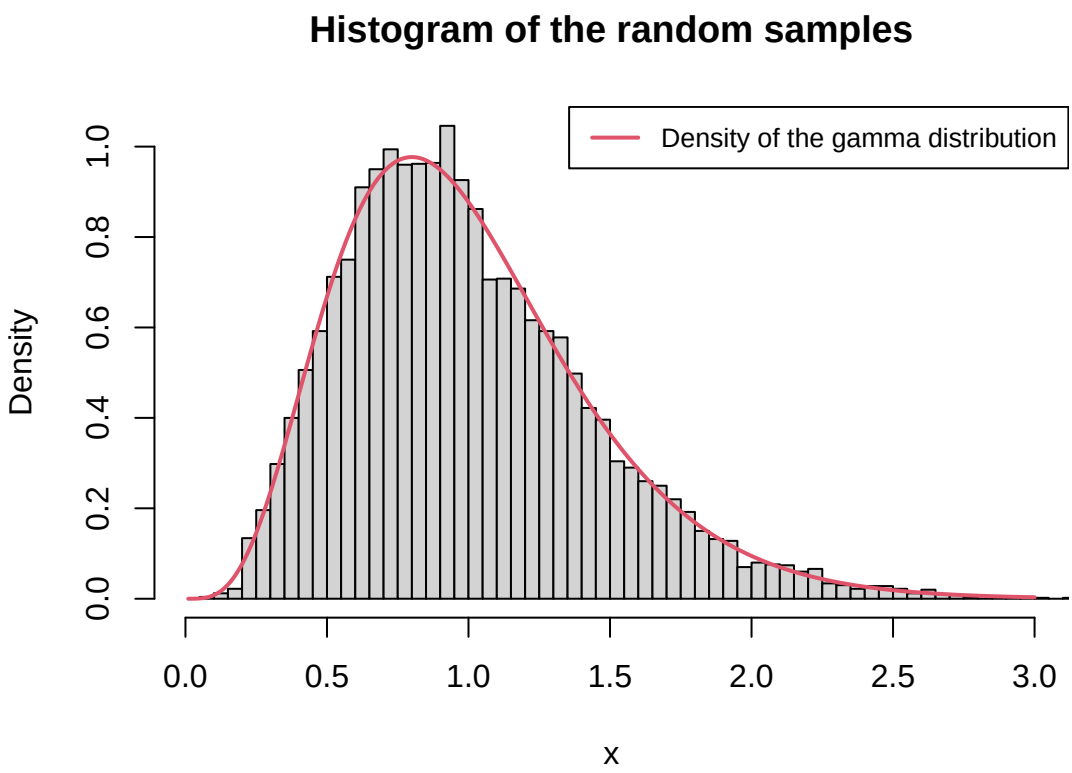
We check only every 12th whole number on the interval $(1, 2000]$ in order to reduce computational power, but from this we see that the algorithm is clearly less efficient for large values of α.

3)

Next we wish to combine our already made functions into one function that generates independent random variables $X \sim Gamma(\alpha, \beta)$ for $\alpha \in (0, 2000], \beta > 0$. We have already made functions that generate gamma distributed variables for $\beta = 1$ and $\alpha \in (0, 1) \cup (1, 2000]$. We exploit the fact that $Gamma(1, 1)$ is the same as $Exp(1)$, and we know how to draw from an exponential distribution thanks to problem A.1. Finally, to account for values of β besides 1, we exploit that β is an inverse scale parameter. Hence, if $X \sim Gamma(\alpha, 1)$ then $\frac{1}{\beta}X \sim Gamma(\alpha, \beta)$.

```
# the function ran_gamma generates independent samples from a gamma distribution
# n: number of samples
# alpha: a real-valued parameter from the interval (0,2000]
# beta: a real-valued parameter > 0
ran_gamma <- function(n,alpha,beta = 1){
  if(beta<0){
    print('Error: beta out of bounds')
    break
  }
  if ((alpha < 1) & (alpha > 0)){
    y = ran_gamma1(n,alpha)$z
  }
  else if ((alpha > 1) & (alpha <= 2000)){
```

```
    y = ran_gamma2(n, alpha,log.scale = F)$y
  }
  else if(alpha == 1){
    y = ranexp(n)
  }
  else{
    print('Error: alpha out of bounds')
    break
  }
  return((1/beta)*y)
}

# set variables and test that the function run as expected
n = 10000
alpha = 5
beta = 5

a = 0.01
b = 3*alpha/beta
l = seq(a,b,0.001)

hist(ran_gamma(n,alpha,beta), freq = F, n=100, xlim = c(a,b), xlab = 'x', ylab = 'Density', main = 'His
lines(l, dgamma(l, shape = alpha, scale = (1/beta)), col = 2, lwd = 2)
legend('topright',legend = ('Density of the gamma distribution'), col = 2, lwd = 2, bty = 'l',cex = 0.8)
```



From the plot, our function seems to be working as intended.

4)

Assume $X \sim Gamma(\alpha, 1)$ and $Y \sim Gamma(\beta, 1)$ are two independent random variables. We wish to show that $Z = \frac{X}{X+Y}$ has a $Beta(\alpha, \beta)$ distribution. Since X and Y are independent, their joint distribution is given by $f_{X,Y}(x, y) = f_X(x) \cdot f_Y(y)$, i.e.

$$f_{X,Y}(x, y) = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} y^{\beta-1} e^{-(x+y)}.$$

We wish to show that the density of Z is

$$f_Z(z) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} z^{\alpha-1}(1-z)^{\beta-1}, \quad z \in [0, 1].$$

This can be done using a change of variables in the density $f_{X,Y}$. Let $U = \frac{X}{X+Y}$ and $V = X + Y$. Then $X = UV$ and $Y = V - UV$, and

$$f_{U,V}(u, v) = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} (uv)^{\alpha-1} (v(1-u))^{\beta-1} e^{-v} |J|,$$

where $J = \begin{bmatrix} v & u \\ -v & 1-u \end{bmatrix}$ is the Jacobian, and $|J| = v$ is the determinant. Separating the terms with u and v leads to

$$f_{U,V}(u, v) = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} u^{\alpha-1}(1-u)^{\beta-1} v^{\alpha+\beta-1} e^{-v},$$

and multiplying by $1 = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha+\beta)}$ yields

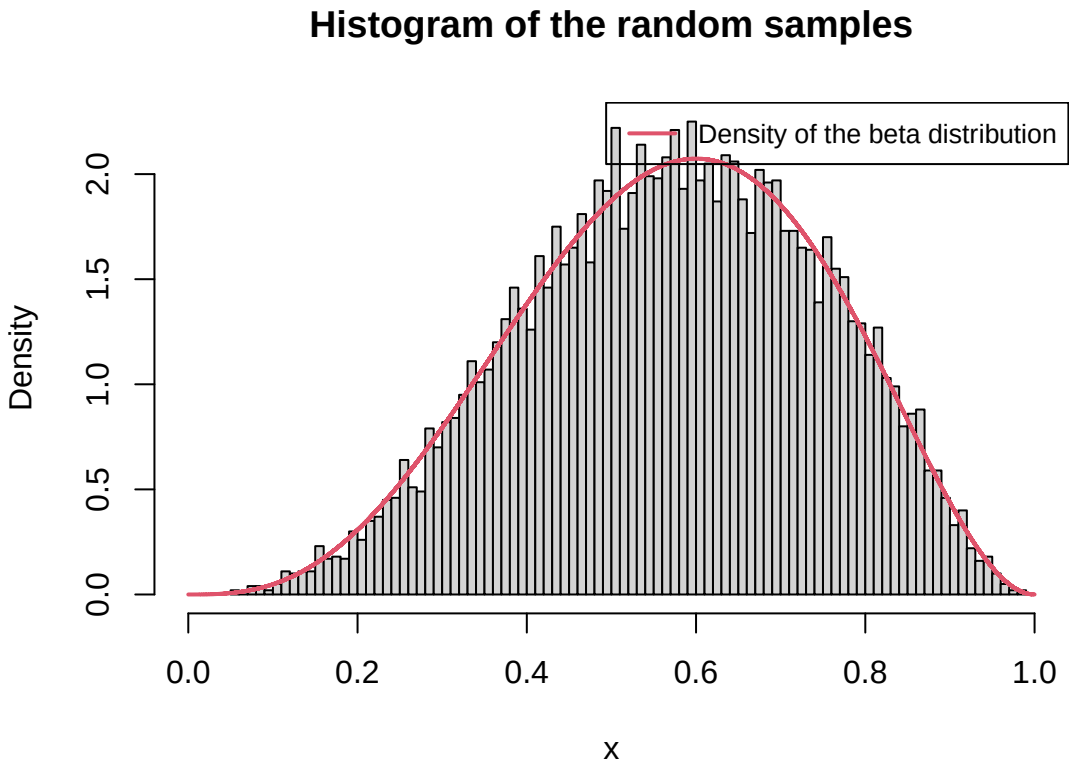
$$f_{U,V}(u, v) = \left(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} u^{\alpha-1}(1-u)^{\beta-1} \right) \left(\frac{1}{\Gamma(\alpha+\beta)} v^{\alpha+\beta-1} e^{-v} \right)$$

Hence U and V are independent, and $U \sim Beta(\alpha, \beta)$ which is what we wanted to show. Additionally, $V \sim Gamma(\alpha+\beta, 1)$ which is consistent with the behavior of the sum of two independent gamma distributed random variables.

Below we make and implement a function that exploits this relationship between gamma and beta distribution.

```
# this function draws independent samples from a Beta(alpha,beta) distribution
ran_beta <- function(n, alpha, beta){
  x = ran_gamma(n,alpha)
  y = ran_gamma(n,beta)

  return(x/(x+y))
}

n = 10000
alpha = 4
beta = 3

a = 0
b = 1
```

```r
l = seq(a,b,0.00001)

hist(ran_beta(n,alpha,beta), freq = F, n=100, xlim = c(a,b), xlab = 'x', ylab = 'Density', main = 'Hist
lines(l, dbeta(l, shape1 = alpha, shape2 = beta), col = 2, lwd = 2)
legend('topright',legend = ('Density of the beta distribution'), col = 2, lwd = 2, bty = 'l',cex = 0.8)
```



From the histogram, we see that the function works as it is supposed to.

Problem C

1)

We want to use Monte Carlo integration to find $\theta = P(X > 4)$ when $X \sim \mathcal{N}(0, 1)$.

```r
# Define the function MC taking in
# x: a vector of samples from a density f
# h: the function we want to find expectation of, i.e. E[h(x)]

MC <- function(x, h){
  x_h = sapply(x, h)              # use the function h on the samples x
  theta_hat = mean(x_h)           # calculate the mean of h(x)
  theta_var = (1/N)*var(x_h)      # calculate the variance of estimated mean of h(x)
  return(c(theta_hat, theta_var)) # return the mean and variance of the parameter
}
```

```r
# Define h as the indicator function over the event x>4
h_MC <- function(x){
  return(x>4)
}

# calculate the parameter theta = P(X>4) using Monte Carlo
N = 100000
x = BoxMuller(N)
theta1 = MC(x,h_MC)

CI1_l = (theta1[1] - qnorm(0.975)*sqrt(theta1[2]))
CI1_u = (theta1[1] + qnorm(0.975)*sqrt(theta1[2]))

c('True value' = pnorm(4,lower.tail = F),'Estimated theta' = theta1[1], 'Lower limit' = CI1_l, 'Upper li
```

```
##      True value Estimated theta     Lower limit     Upper limit
##    3.167124e-05    7.000000e-05    1.814578e-05    1.218542e-04
```

We use normal quantiles in our confidence interval. Since we have 100000 samples in an estimate of the mean, the central limit theorem ensures asymptotic normal distribution.

2)

In this task we wish to use importance sampling from the density

$$g(x) = \begin{cases} cxe^{-\frac{x^2}{2}} & x > 4 \\ 0 & \text{otherwise} \end{cases}$$

to estimate $\theta = P(X > 4)$. Calculations similar to what was done previously yields the normalizing constant $c = e^8$, and the inverse cumulative function

$$G^{-1}(x) = \sqrt{2(8 - \log(1 - x))}$$

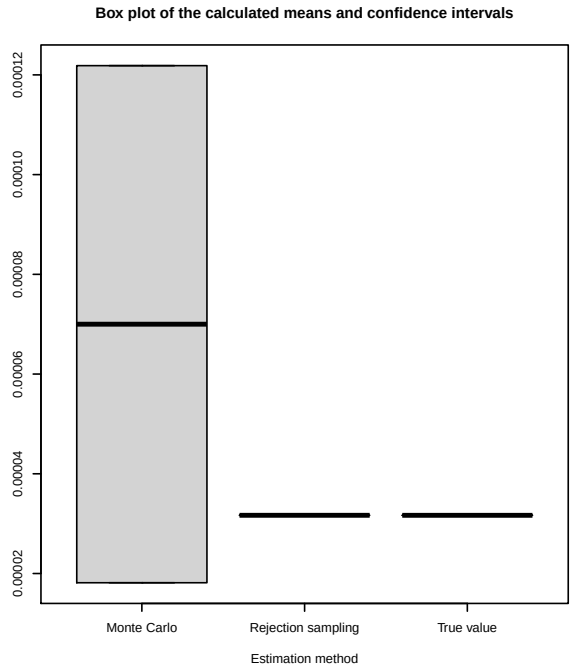
which can be used in the inverse sampling algorithm.

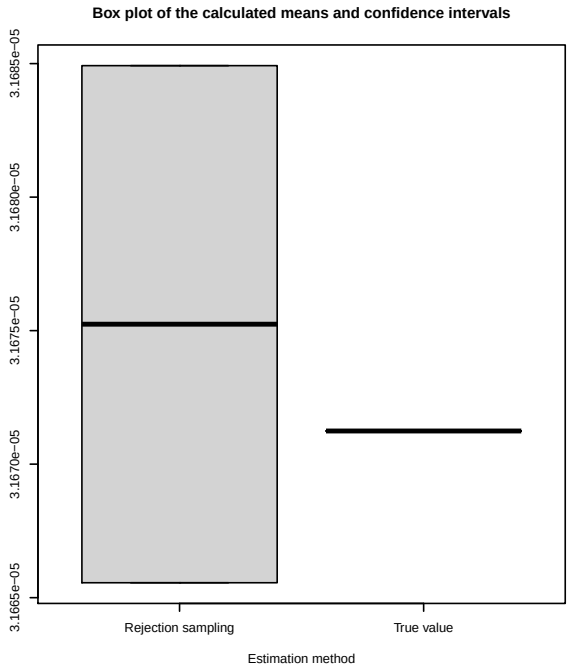
```r
# a general importance sampling functions which takes in:
# x: a vector of samples from the proposal density
# f: the target density
# g: the proposal density
# h: the function we want to find expectation of, i.e. E[h(x)]
importance <- function(x,f,g,h,return_vec = F){
  n = length(x)
  x_f = sapply(x,f)
  x_g = sapply(x,g)
  x_h = sapply(x,h)
  y = (x_h*(x_f/x_g))

  if(isTRUE(return_vec)){
    return(y)
  }
  else{
    return(c(mean(y),(var(y)/n)))
```

```
  }
}

# define the inverse cumulative of the proposal distribution g
inv_g2 <- function(x){
  return(sqrt(2*(8 - log(1-x))))
}

# Use the inverse CDF method to draw from the distribution g
rang2 <- function(n){
  U = runif(n)
  V = sapply(U,inv_g2)
  return(V)
}

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

# define the proposal density g
g_is <- function(x){
  return(exp(8)*x*exp(-(x^2)/2))
}

# define h as the indicator function over the event x>4
h_is <- function(x){
  return(x>4)
}

N = 100000
# calculate the parameter theta = P(X>4) using rejection sampling
theta2 = importance(rang2(N),dnorm,g_is,h_is)

CI2_l = (theta2[1] - qnorm(0.975)*sqrt(theta2[2]))
CI2_u = (theta2[1] + qnorm(0.975)*sqrt(theta2[2]))


c('True value' = pnorm(4,lower.tail = F),'Estimated theta' = theta2[1], 'Lower limit' = CI2_l, 'Upper li
```

```
##      True value Estimated theta     Lower limit     Upper limit
##    3.167124e-05    3.167524e-05    3.166556e-05    3.168492e-05
```

```
par(mfrow = c(1,2), cex = 0.5)
boxplot(c(CI1_l,CI1_u),c(CI2_l,CI2_u),pnorm(4,lower.tail = F),xlab = 'Estimation method', names = c('Mo
boxplot(c(CI2_l,CI2_u),pnorm(4,lower.tail = F),xlab = 'Estimation method', names = c('Rejection samplin
```






```
MC_N = (theta1[2]*N)/theta2[2]
```

As we can see from the Box plot, Monte Carlo have not even included the true value in the 95% confidence interval. When estimating such small probabilities using Monte Carlo, there is very high variance in the result between each time the algorithm runs. A targeted estimation method, such as importance sampling, yields significantly more accurate results, and is able to make a small condifence interval that still encaptures the true value.

If we wanted the same precision as importance sampling gives us after $n = 100000$ samples from our Monte Carlo estimate, we would need a sample size N such that

$$Var_{MC}(\bar{\theta}) = \frac{1}{N}\left[E_f[h(x)^2] - (E_f[h(x)])^2\right] \leq \frac{1}{100000}\left[E_g[(\frac{h(x)f(x)}{g(x)})^2] - (E_g[\frac{h(x)f(x)}{g(x)}])^2\right] = Var_{IS}(\bar{\theta})$$

We can estimate $\left[E_f[h(x)^2] - (E_f[h(x)])^2\right]$ and $\left[E_g[(\frac{h(x)f(x)}{g(x)})^2] - (E_g[\frac{h(x)f(x)}{g(x)}])^2\right]$ using our previously drawn samples. Doing this and then solving for N shows that we would need 2.8680168×10^{12}samples in order to achieve lower variance using Monte Carlo estimation.

3)

```
# sample using antithetic variates
rang3 <- function(n){
  U = runif(n)
  U2 = 1-U
  V = sapply(U,inv_g2)
  V2 = sapply(U2,inv_g2)
  return(c(V,V2))
```

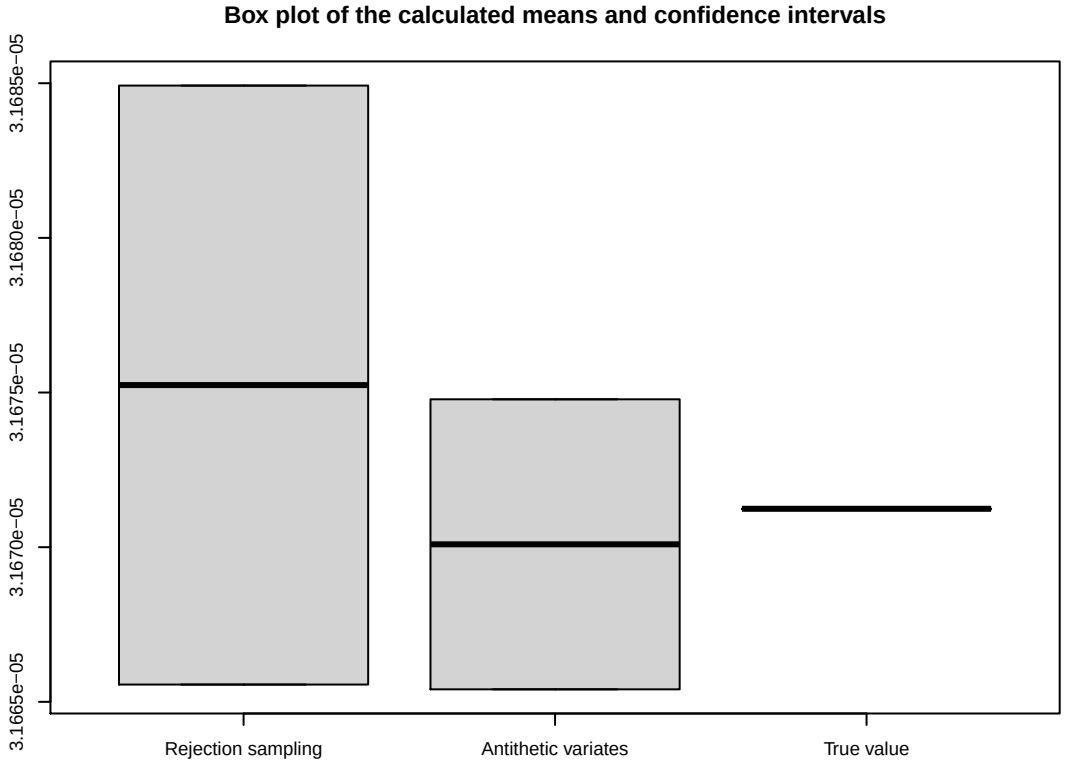
```
}
n = 50000
y = rang3(n)

theta3 = importance(y,dnorm,g_is,h_is, return_vec = T)
theta3_mean = mean(theta3)

th3_1 = theta3[1:n]
th3_2 = theta3[(n+1):(2*n)]
theta3_var = (1/(4*n))*(var(th3_1)+ var(th3_2)+2*cov(th3_1,th3_2))
theta3_var
```

```
## [1] 5.722056e-18
```

```
CI3_l = (theta3_mean - qnorm(0.975)*sqrt(theta3_var))
CI3_u = (theta3_mean + qnorm(0.975)*sqrt(theta3_var))

c('Lower limit' = CI3_l, 'Upper limit' = CI3_u)
```

```
##  Lower limit  Upper limit
## 3.166540e-05 3.167478e-05
```

```
par(cex = 0.6)

boxplot(c(CI2_l,CI2_u),c(CI3_l,CI3_u),pnorm(4,lower.tail = F), names = c('Rejection sampling', 'Antithe
```



Since we use both the random samples U and $1-U$ we get $2n$ random samples while only having to simulate half as many random numbers. Hence, in order to compare with our previous methods we only use $n = 50000$ samples for this algorithm.

The variance used in the confidence interval has to be calculated while keeping in mind that there is dependence in the samples when we use U and $1-U$. We can estimate the variance by looking at

$$Var_{IS}(\bar{\theta}) = Var_{IS}(\frac{\bar{\theta_1} + \bar{\theta_2}}{2}) = \frac{1}{4}(Var(\bar{\theta_1}) + Var(\bar{\theta_2}) + 2Cov(\bar{\theta_1}, \bar{\theta_2}))$$

where $\bar{\theta_1}$ are sampled from the transformed random vector U, and$\bar{\theta_2}$ are sampled from the transformed random vector $1-U$. We used built-in functions in R to find the variance and covariance of the vectors $Y_1 = \frac{h(U)f(U)}{g(U)}$ and $Y_2 = \frac{h(1-U)f(1-U)}{g(1-U)}$. Then we divided all by n as it is then vector mean we are interested in finding the variance of. From the box-plot we observe that this method yields an even more precise estimate than the two previous methods. This due to the negative correlation between U and $1-U$, thus making the term $2Cov(\bar{\theta_1}, \bar{\theta_2}))$ from the above expression negative, and hence causing a reduction in the overall variance of our estimator.

Problem D

1)

We want to sample from $f(\theta|y)$, our target distribution. We want to use the uniform distribution from 0 to 1 as the proposal distribution, $g(x)$. This yields

$$f(\theta|y) \leq c \cdot g(x)$$

Since the proposal density is uniform, this amounts to finding the maximum of the posterior density. Since $g(x)$ is a straight line at $y = 1$, c has to be bigger than or equal to maximum of $f(\theta|y)$. By maximizing $f(\theta|y)$ in the following way:

$$\frac{d}{d\theta} f(\theta|y) = 0$$

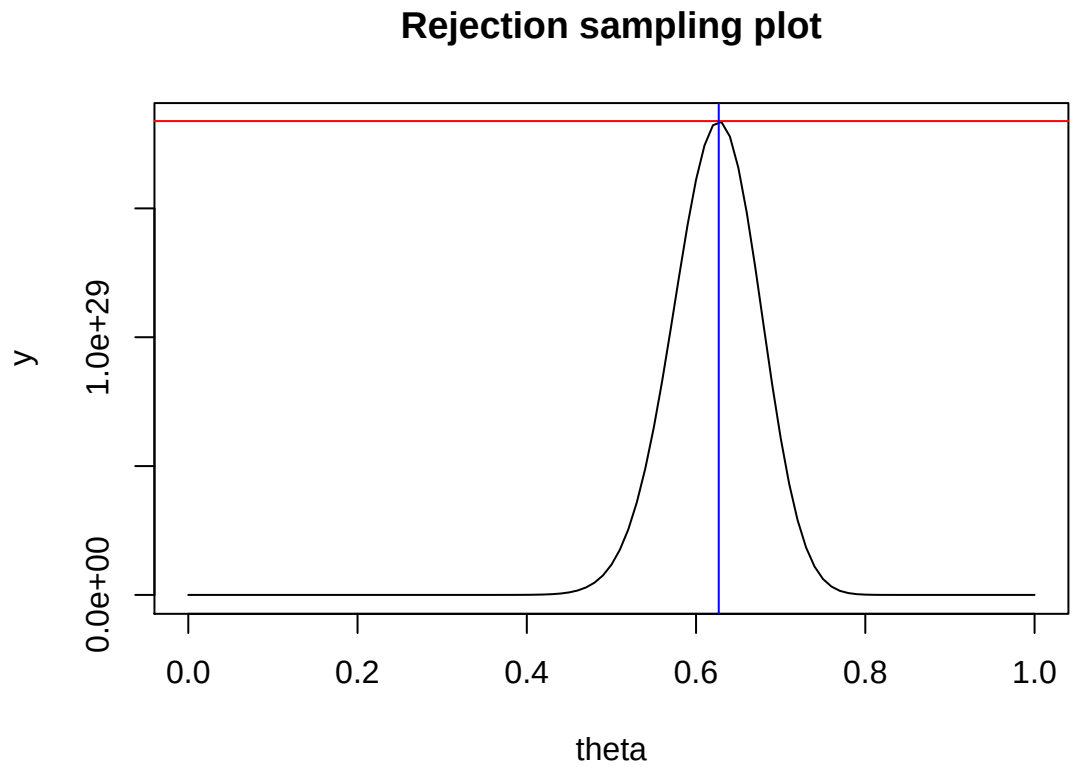
$$\theta \approx 0.63, \text{ when } \theta \in (0, 1)$$

we see that is reaches is maximum at $\theta = 0.6268214 \approx 0.63$.

```
y_1 = 125
y_2 = 18
y_3 = 20
y_4 = 34
y_vec = as.vector(c(y_1, y_2, y_3, y_4))  #y values.

f <- function(theta, y = c(125,18,20,34)){
return( (2 + theta)^(y[1])*(1 - theta)^(y[2] + y[3])*theta^(y[4]) )
}

max <- optimize(f,interval=c(0,1),maximum=TRUE)$maximum
c <- f(max)
c                              #maximum of posterior density.
```

```
## [1] 1.838839e+29
```

```r
max_f_y = 0.6268214            #theta value with yields maximum.

curve(((2+x)^(y_1))*((1-x)^(y_2+y_3))*x^(y_4), 0,1, xlab = "theta",
      ylab = "y", main = "Rejection sampling plot")                    #plot of posterior density.
abline(a = c, b = 0, col = "red")                                      #c-value needed for rejection
abline(v=max_f_y, col="blue")                                          #theta-value where maximum is
```




```r
target_dist <- function(theta){
  return(((2+theta)^y_1)*((1-theta)^(y_2+y_3))*theta^(y_4)) #posterior density of interest.
}

rej_sampling <- function(n){
  X = runif(n)       #simulate from proposal distribution g.
  U = runif(n)       # generate from the unif[0,1].

  count = 1          #counter to be used in "if"-statement below.
  accept = c()       #emtpy list, which will be filled with accepted theta values.

  while(count <= n & length(accept) < 1000){
  test_u = U[count]                                          #test_u for comparison.
  test_x = target_dist(X[count])/(c*dunif(X[count], 0, 1))   #find test_x for comparison (rejection samp
  if (test_u <= test_x){
    accept = rbind(accept, X[count])      #if value is accepted the value is added to "accept".
    count = count + 1                     #added to a counter for the while loop.
```

```
    if (length(accept) == 1 ){            #for task D-3, when the first accepted value gets added to th
      forste = count-1                    #variable forste: is count minus 1, because the
                                          #variable count starts at 1.
    }
  }

  count = count + 1
}

  return(list(accept, forste))           #returns a list with all the accepted values,
                                          #and the amount of draws needed from proposal
                                          #distribution to get an accepted theta-value.
}

accept = rej_sampling(10000)              #Run the rejection sampling function for n=10000 times.
list_theta_values = accept[[1]]           #Returns the list of accepted theta values.
rej_sample_mean = mean(list_theta_values) #Gives us the mean of the expected theta values.
rej_sample_mean
```

```
## [1] 0.6222301
```

Here we get 10000 samples from $f(\theta|y)$ by using rejection sampling where the mean is approximately 0.62.

2)

When we try integrate the posterior density from 0 to 1 we need to keep in mind we don't have the normalizing constant. The function $f(\theta|y)$ is only proportional to $(2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}, \theta \in (0,1)$

We use integration to solve for c:

$$\int_0^1 f(\theta|y) = \frac{1}{c}\int_0^1 (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}d\theta = 1$$

$$\int_0^1 f(\theta|y) = \frac{1}{c}\int_0^1 (2+\theta)^{125}(1-\theta)^{18+y20}\theta^{34}d\theta = 1$$

$$\frac{1}{c}(2.3577)\cdot 10^{28} = 1$$

$$c = 2.3577 \cdot 10^{28}$$

We now got the normalizing constant which can be used to check the value for the mean by numerical integration.

```
MC_mean = MC(rej_sampling(10000)[1], identity)[1]  #returns the mean when using Monte Carlo Integration

mean_func <- function(x){
  return(x*target_dist(x))                    #Function to be used for numerical integration.

}

mean_without = integrate(mean_func, lower = 0, upper = 1) #using numerical integration from 0 to 1, wit
```

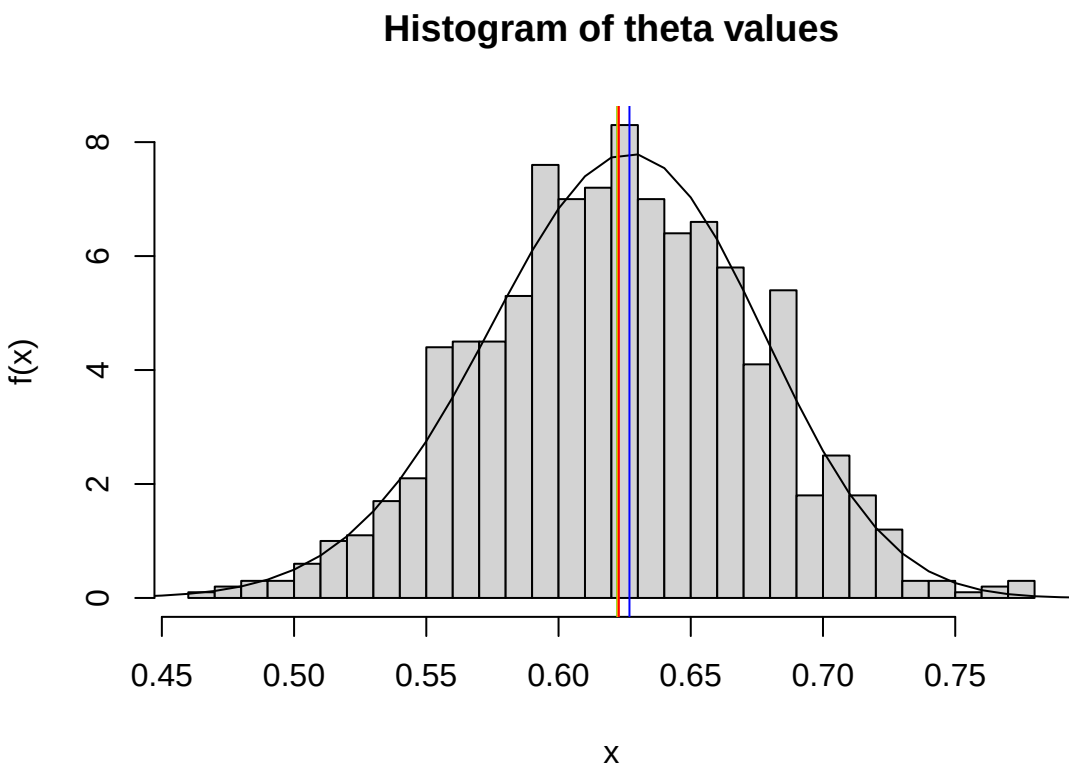
```r
norm_const = 2.3577*10^28                                    #normalizing constant.
integral_mean = mean_without$value/norm_const                #dividing by normalizing constant to obtain t
integral_mean
```

```
## [1] 0.6228049
```

```r
means = as.vector(c(max_f_y, rej_sample_mean, MC_mean, integral_mean))   #vector of the different means
means
```

```
## [1] 0.6268214 0.6222301 0.6223332 0.6228049
```

```r
hist(list_theta_values, freq = FALSE, breaks = 30, main = "Histogram of theta values"
     ,xlab="x",ylab="f(x)")                                          #histogram of accepted

curve((((2+x)^(y_1))*((1-x)^(y_2+y_3))*x^(y_4))/norm_const, 0,1, xlab = "theta", ylab = "y", add = TRUE)
abline(v=max_f_y, col="blue")              #blue line: theta value which maximizes posterior density.
abline(v=rej_sample_mean, col="green")     #green line: mean theta value by rejection sampling.
abline(v=MC_mean, col="orange")            #orange line: mean theta value by MC integration.
abline(v=integral_mean, col="red")         #red line: mean theta value by numerical integration.
```



3)

How many random numbers does your sampling algorithm need to generate on average in order to obtain one sample of $f(\theta|y)$. Derive your answer practically using your sampler and compare it with a theoretical result computed numerically.

We want to find how many draws from the proposal distribution it takes, on average, to get one draw from our distribution of interest, $f(\theta|y)$.

```
#average_num: function that takes in n.
#n: number of repetions to find mean value of how many draws from proposal distribution.

average_num <- function(n){
  i = 0                       #i: a counter
  number_of_trials = c()      #emtpy list to be filled with how many draws needed for one sample from our
  while(i <= n){
    number_of_trials = rbind(number_of_trials,
                             rej_sampling(100)[[2]]) #rej_sampling(100) because we will get an
    i = i + 1                                        #adding 1 to counter.

  }
  return(mean(number_of_trials))

}

average_num(10000) #returns the mean number of trials when trying 10000 times.
```

```
## [1] 7.909909
```

Running the function average_num(n) for $n = 10000$ we get that the average number of numbers from the proposal distribution need to get an accepted θ-value and thus a sample from $f(\theta|y)$ is approximately 7.7 numbers. This is the result when we use our sampler practically.

The theoretical numbers of samples needed to get one sample from the posterior can be quantified as π.

```
pi = c/norm_const      #when dividing the maximum of posterior f(theta|y) by the normalizing
                       #constant we get the real or true value of samples needed.
pi
```

```
## [1] 7.799292
```

We can see that our practical sampler and theoretical value are fairly close.

4)

We now want to repeat D-2, but instead of using a uniform prior ($(U(0,1))$) (equiv. to $Beta(1,1)$), we use $Beta(1,5)$ as the prior. Here we're going to use importance sampling weights

$$w(x_i) = \frac{f(x_i)}{g(x_i)}$$

and when inserting $f(x_i)$ (new posterior) and $g(x_i)$ (the previous posterior) things cancel out and we get:

$$w(x_i) = \frac{(1-x_i)^4}{Beta(1,5)}$$

```
w <- function(x){
  y <- ((1-x)^4)/(beta(1,5))

  return(y)
}
```

Let's implement the importance sampling algorithm to find the posterior mean, with the prior being $Beta(1,5)$ instead of $Beta(1,1)$.

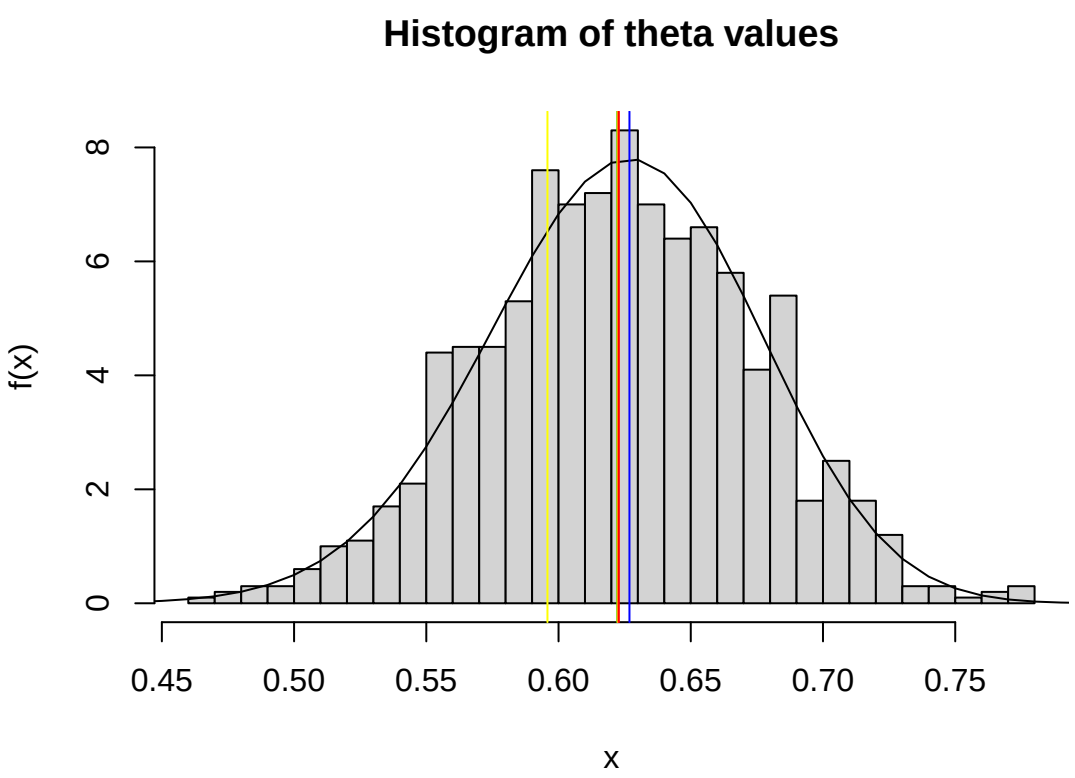
```
#IS_mean: function that takes in theta values (q), and returns mean.

IS_mean <- function(q){
  mu<-sum(q*w(q))/(sum(w(q)))      #weighted importance sampling
  return(mu)                       #return the new mu.
}

ISmean = IS_mean(list_theta_values)
ISmean
```

```
## [1] 0.5958116
```

We can observe that the posterior mean is a bit smaller when we change the prior to $Beta(1,5)$. Which indicates that the choice of prior clearly effects the posterior mean. This makes sense when we look at the priors used in task D.

When we use $Beta(1,1)$ as a prior it has expected value $E[X] = \frac{\alpha}{\alpha+\beta} = \frac{1}{2}$. However, when we use $Beta(1,5)$ as a prior, we get that $E[X] = \frac{\alpha}{\alpha+\beta} = \frac{1}{6}$. This is a bad choice of prior, as it weighs the mean down and hence leaving this estimate as the most inaccurate.

```
hist(list_theta_values, freq = FALSE, breaks = 30, main = "Histogram of theta values"
     ,xlab="x",ylab="f(x)")                                          #histogram of accepted

curve((((2+x)^(y_1))*((1-x)^(y_2+y_3))*x^(y_4))/norm_const, 0,1,
      xlab = "theta", ylab = "y", add = TRUE)                 #posterior density.

abline(v=max_f_y, col="blue")              #blue line: theta value which maximizes posterior density.
abline(v=rej_sample_mean, col="green")     #green line: mean theta value by rejection sampling.
abline(v=MC_mean, col="orange")            #orange line: mean theta value by MC integration.
abline(v=integral_mean, col="red")         #red line: mean theta value by numerical integration.
abline(v=ISmean, col="yellow")
```


Histogram of theta values
f(x)
0
2
4
6
8
0.45
0.50
0.55
0.60
0.65
0.70
0.75
x