Exercise 1 - TMA4300 Computer Intensive Statistical Methods

Problem A

Question 1

We first want to write an R function that generates samples from a exponential distribution with rate parameter λ. We know that the cumulative exponential distribution takes the form

$$F(x;\lambda) = 1 - \lambda e^{-\lambda x}, \quad x \geq 0$$

Computing the inverse cumulative function exploiting the uniform distribution $U \sim Unif(0,1)$ we can get

$$X = F^{-1}(u) = -ln(u)/\lambda, \quad 0 \leq u \leq 1$$

We now want to show that this is true using simulation from the inverse cumulative distribution

```
# n: number of samples to generate rate: the rate expdist
# returns a vector with generated random numbers from
# exponential distribution
expdist <- function(rate, n) {
    u <- runif(n)
    x <- -log((1/rate) * (1 - u))/rate
    list = list(x, u)
    return(list)
}
n <- 60000
lambda <- 1
outval <- expdist(lambda, n)
x = outval[[1]]
y = outval[[2]]
ggplot(data.frame(x = x)) + geom_histogram(aes(x = x, y = ..density..),
    bins = 50) + geom_line(data = data.frame(x = x, y = dexp(x,
    rate = 1)), aes(x = x, y = y), color = "blue") + xlim(0,
    5) + labs(title = "Exponential Distribution", x = "x", y = "Density")
```

From the plot one can see that the exponential distribtion that we generated and the exponential distribution in R closely follow each other, which means that our implementation is correct.

Question A2a

Let us consider a new example where the PDF has the form

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

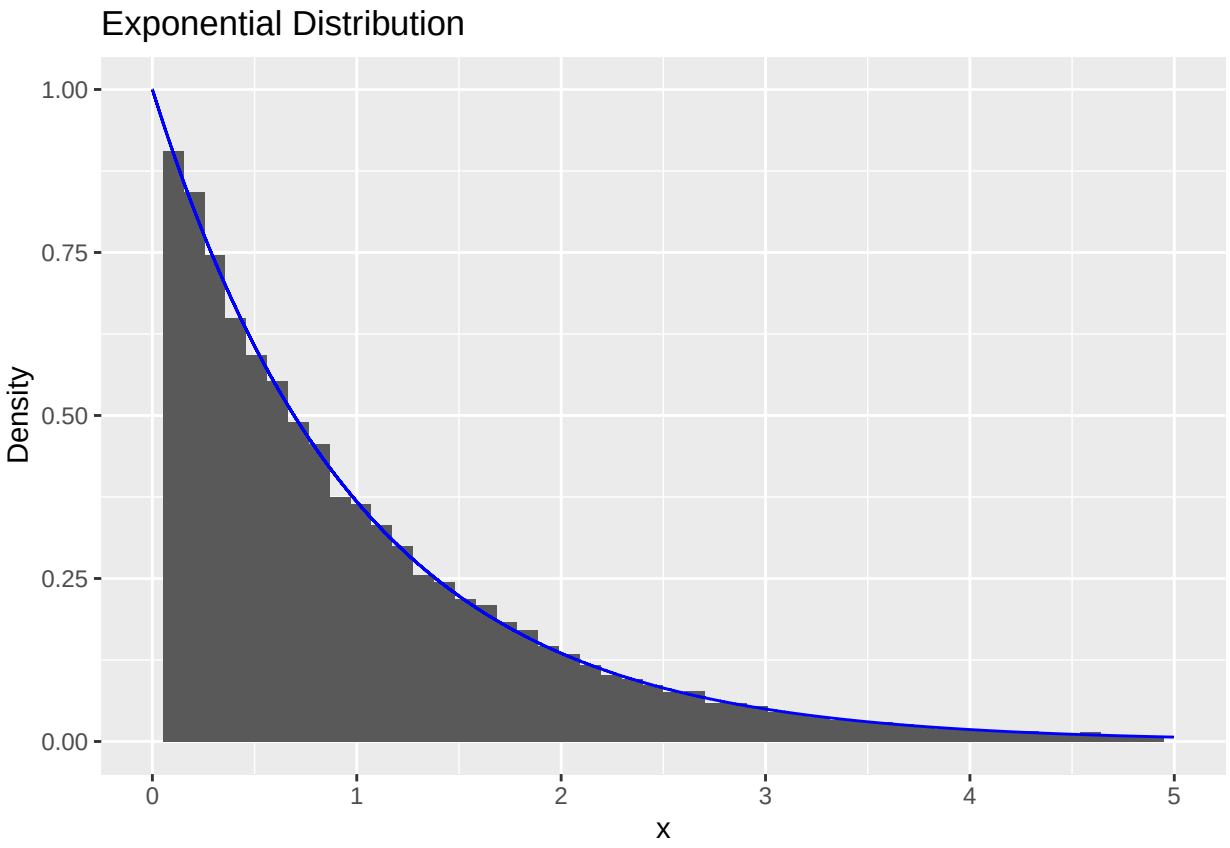


Figure 1: Theoretical and simulated exponential distribution when lambda = 1.

The cumulative function can be computed with integration

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

And similarly with the inverse cumulative function

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

with c as $\frac{e\alpha}{e+\alpha}$ in all the expressions, which is found by using that the density integrated over the whole space must be 1.

Question A2b

This distribution can be plotted by using the inverse cumulative distribution as before.

```
# ca: normalizing constant
ca <- function(alpha) {
    return(exp(1) * alpha/(exp(1) + alpha))
}
# alpha: parameter alpha n: number of samples to generate
# dist1 returns a vector of random numbers from the
# distribution g
dist1 <- function(alpha, n) {
    u <- runif(n)
    L = u < ca(alpha)/alpha
    res = c(1:n) * 0
    res[L] = (u[L] * alpha/ca(alpha))^(1/alpha)
    res[!L] = -log((1 - u[!L])/ca(alpha))
    return(list(res, u))
}

# g_test is the theoretical function to test the algorithm
# with
g_test = function(alpha, x) {
    res = c(1:length(x)) * 0
    L = x < 1 & x > 0
    res[L] = ca(alpha) * x[L]^(alpha - 1)
    res[!L] = ca(alpha) * exp(-x[!L])
    return(res)
}

n <- 60000
alpha <- 0.5
outval1 <- dist1(alpha, n)
x <- outval1[[1]]
y <- outval1[[2]]
```

```r
hist(x, breaks = 50, freq = FALSE, main = bquote("Histogram of samples from g when " ~
    alpha == .(alpha)))
curve(g_test(alpha, x), col = "red", lwd = 2, add = TRUE)
```

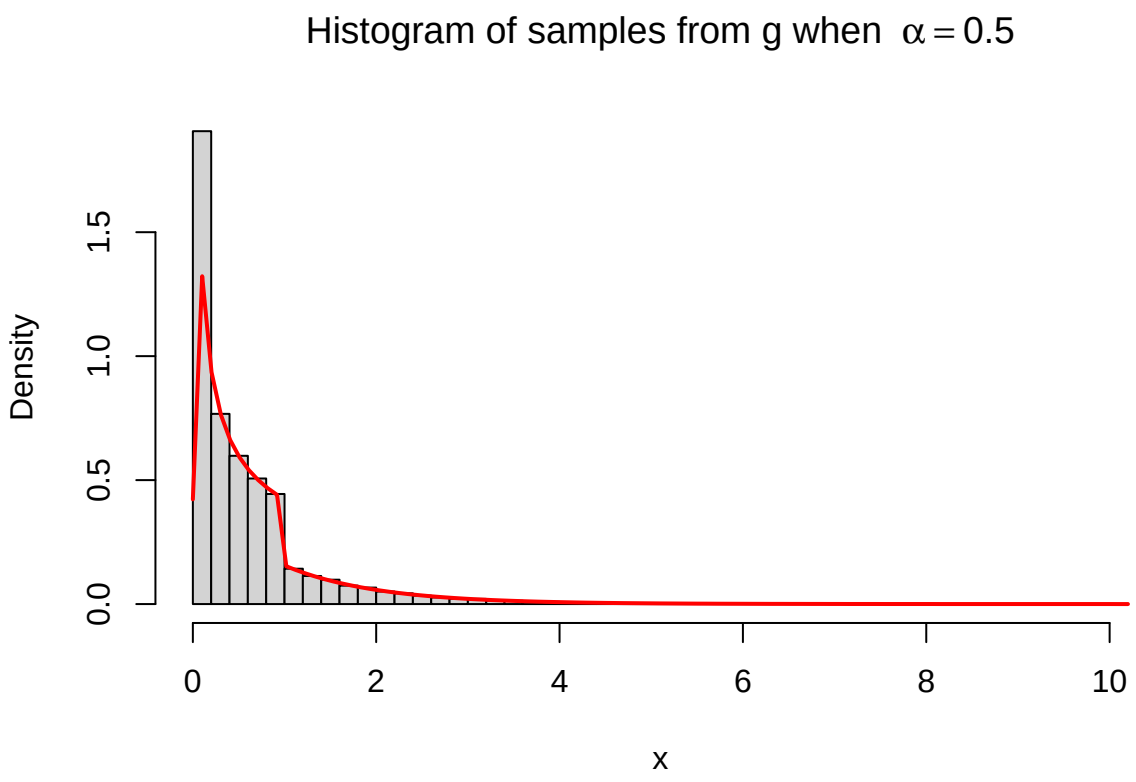


Figure 2: Simulated and theoretical values from our probability density function by the invesion method when alpha = 0.5.

Question A3a

The normalizing constant is found by using that a PDF must integrate to 1, and therefore we get $c = \alpha$.

Question A3b

By integrating the PDF from $-\inf$ to x, we find the CDF:

$$F(x) = \frac{-1}{1 + e^{\alpha x}} + 1$$

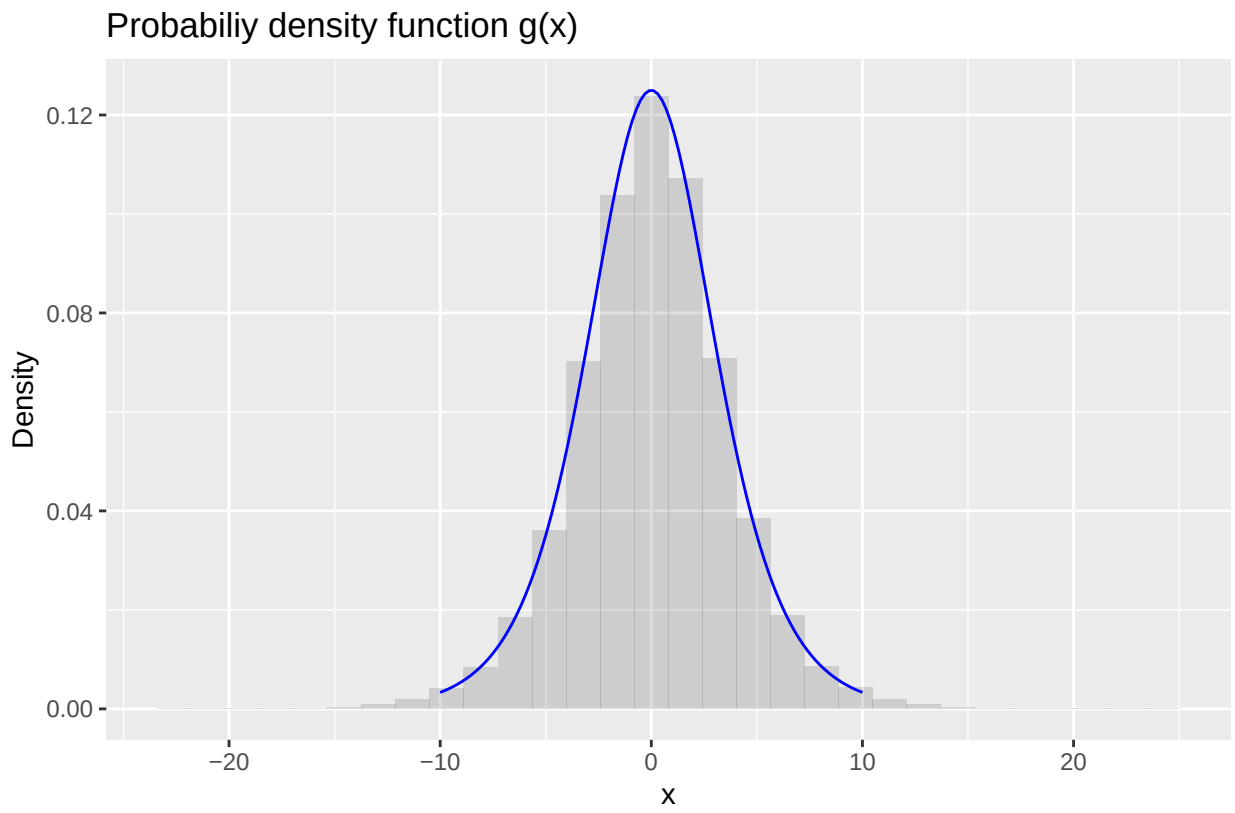
and the inverse of F is found by solving for x (where $F(x)$ is y:

$$F^{-1}(x) = \frac{1}{\alpha} ln(\frac{-y}{y-1})$$

Question A3c

We want to use the inversion method to generate random samples from the distribution in question A3b, and compare them with the theoretical values.

```
# alpha: the parameter alpha n: number of samples to be
# generated dist3 gives a vector of random variables from
# the distribution in question 3b
dist3 <- function(alpha, n) {
    u <- runif(n)
    x <- 1/alpha * log(-u/(u - 1))
    list = list(x, u)
    return(list)
}
# checkfunc3 gives values from the theoretical function
# that we want to check with
x <- seq(from = -10, to = 10, length.out = 100)
checkfunc3 <- function(alpha, x) {
    return(alpha * exp(alpha * x)/(1 + exp(alpha * x))^2)
}
outval3 <- dist3(alpha, n)
ggplot(data.frame(x = outval3[[1]], y = outval3[[2]])) + geom_histogram(aes(x = x,
    y = ..density..), alpha = 0.2) + geom_line(data = data.frame(x = x,
    y = checkfunc3(alpha, x)), aes(x = x, y = y), color = "blue") +
    labs(title = "Probabiliy density function g(x)", x = "x",
        y = "Density", caption = "Theoretical and simulated values of our probability density function
```



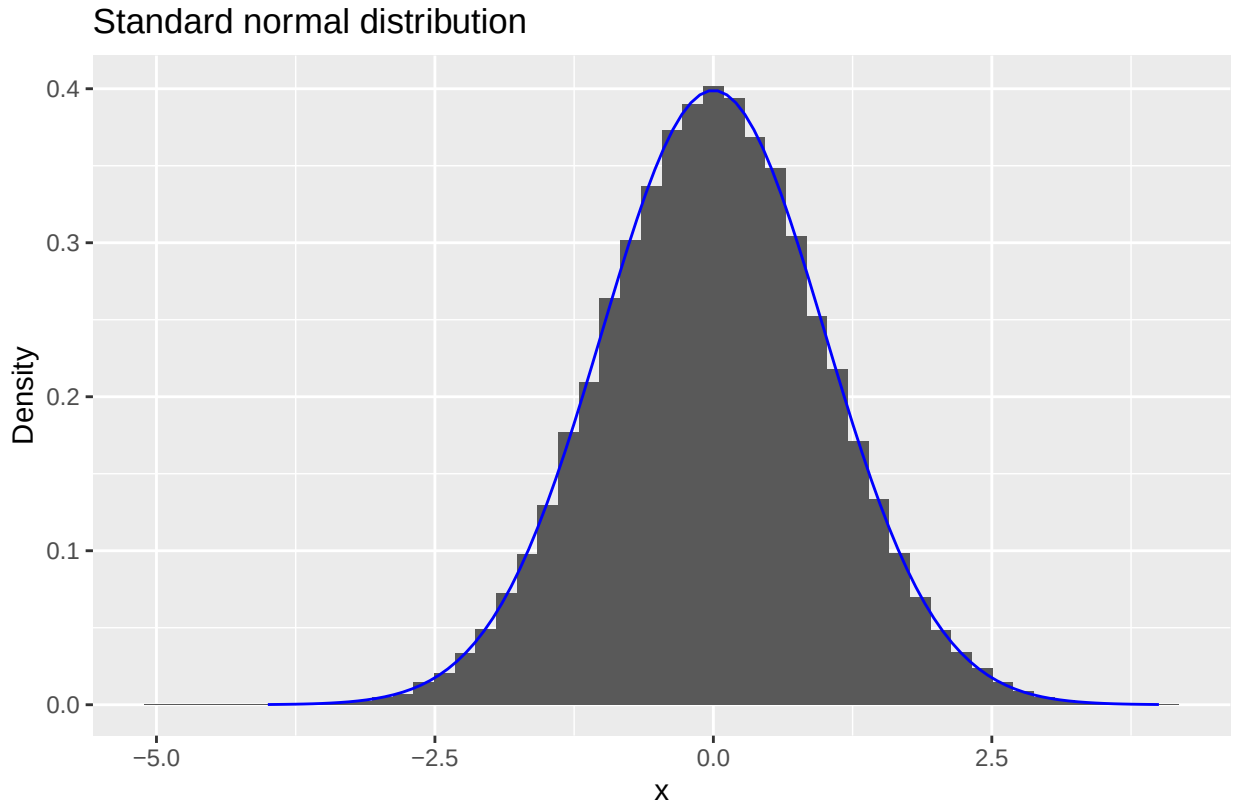
Theoretical and simulated values of our probability density function when alpha = 0.2.

From the plot we see that our simulation follows the theoretical values, although it could be better.

Question A4

We want to sample from the univariate normal distribution using the Box-Müller algorithm. We compare it to the normal distribution implemented in R.

```
# n: number of samples to generate mynormal samples from
# box-muller algorithm
mynormal <- function(n) {
    u1 = runif(n)
    u2 = runif(n)
    return(sqrt(-2 * log(u1)) * sin(2 * pi * u2))
}
n = 1e+05
mynormalout <- mynormal(n)
std_normal_data <- data.frame(x = mynormalout)
x <- seq(-4, 4, length = 100)
ggplot(std_normal_data) + geom_histogram(aes(x = x, y = ..density..),
    bins = 50) + geom_line(data = data.frame(x = x, y = dnorm(x)),
    aes(x = x, y = y), color = "blue") + labs(title = "Standard normal distribution",
    x = "x", y = "Density", caption = "Theoretical and simulated values of the standard normal distribu
```



Theoretical and simulated values of the standard normal distribution.

Question A5

We want to sample from a d-variate normal distribution with mean μ and covariance matrix Σ. We use the function mynormal with the Box-Muller algorithm to construct a vector with values of μ and a positive definite matrix $\Sigma = MM^T$, where M is a d by d matrix from a normal distribution. As we don't have given values ofμ and Σ, we just choose a μ and Σ such that Σ is positive definite by the Cholesky factorization. Note that in the code, sigma $= \mu^T\mu$ because of how the Cholesky function is implemented in R.

```
# mu: mean of the multivariate normal distribution sigma:
# covariance matrix of the multivariate normal distribution
dim = 2
mu <- c(1, 2)
M = chol(matrix(c(2, 1, 1, 3), dim, dim))
sigma <- t(M) %*% M
```

To simulate from a d-variate normal distribution, we use that $y = \mu + Mx$, where $\Sigma = MM^T$. We simulate n realisations from the multivariate normal distribution below. Note that the code uses $y = \mu + M^Tx$ because of how the Cholesky function is implemented in R.

```
# y: matrix to store the realisations of the multivariate
# normal distribution
n = 1e+05
y = matrix(0, dim, n)
for (i in 1:n) {
    y[, i] = mu + t(M) %*% mynormal(dim)
}
# means: vector with the mean of the simulated values
means = numeric(dim)
for (i in 1:dim) {
    means[i] = mean(y[i, ])
}
# covmatrix: matrix with the covariances of the simulated
# values
covmatrix = cov(t(y))
print("The chosen mean vector is: ")
```

```
## [1] "The chosen mean vector is: "
```

```
print(mu)
```

```
## [1] 1 2
```

```
print("The simulated mean vector is: ")
```

```
## [1] "The simulated mean vector is: "
```

```
print(means)
```

```
## [1] 1.004203 2.006795
```

```r
print("The chosen covariance matrix is: ")
```

```
## [1] "The chosen covariance matrix is: "
```

```r
print(sigma)
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    1    3
```

```r
print("The simulated covariance vector is: ")
```

```
## [1] "The simulated covariance vector is: "
```

```r
print(covmatrix)
```

```
##          [,1]     [,2]
## [1,] 1.993937 1.007558
## [2,] 1.007558 3.013150
```

We see that the chosen mean and covariance is almost equal to the simulated values, which shows that the implementation is correct.

Problem B

Question B1a

The acceptance probability has the form

$$P(U \leq \frac{f(x)}{cg(x)}) = \int_{-\infty}^{\infty} \frac{f(x)}{cg(x)} g(x)dx = \int_{-\infty}^{\infty} \frac{f(x)}{c} dx = \frac{1}{c}$$

where c is such that $\frac{f(x)}{g(x)} \leq c$. To find c, consider the expression $f(x) \leq cg(x)$. Then we get (for the nontrivial cases when $x > 0$)

$$\frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x} \leq \begin{cases} c\frac{e\alpha}{e+\alpha} x^{\alpha-1}, 0 < x < 1 \\ c\frac{e\alpha}{e+\alpha} e^{-x}, x \geq 1 \end{cases}$$

Rearranging the inequalities and using that e^{-x} and $x^{\alpha-1}$ are decreasing functions, we get the following upper bound for c:

$$c \geq \frac{1}{\Gamma(\alpha)} \frac{e+\alpha}{e\alpha}$$

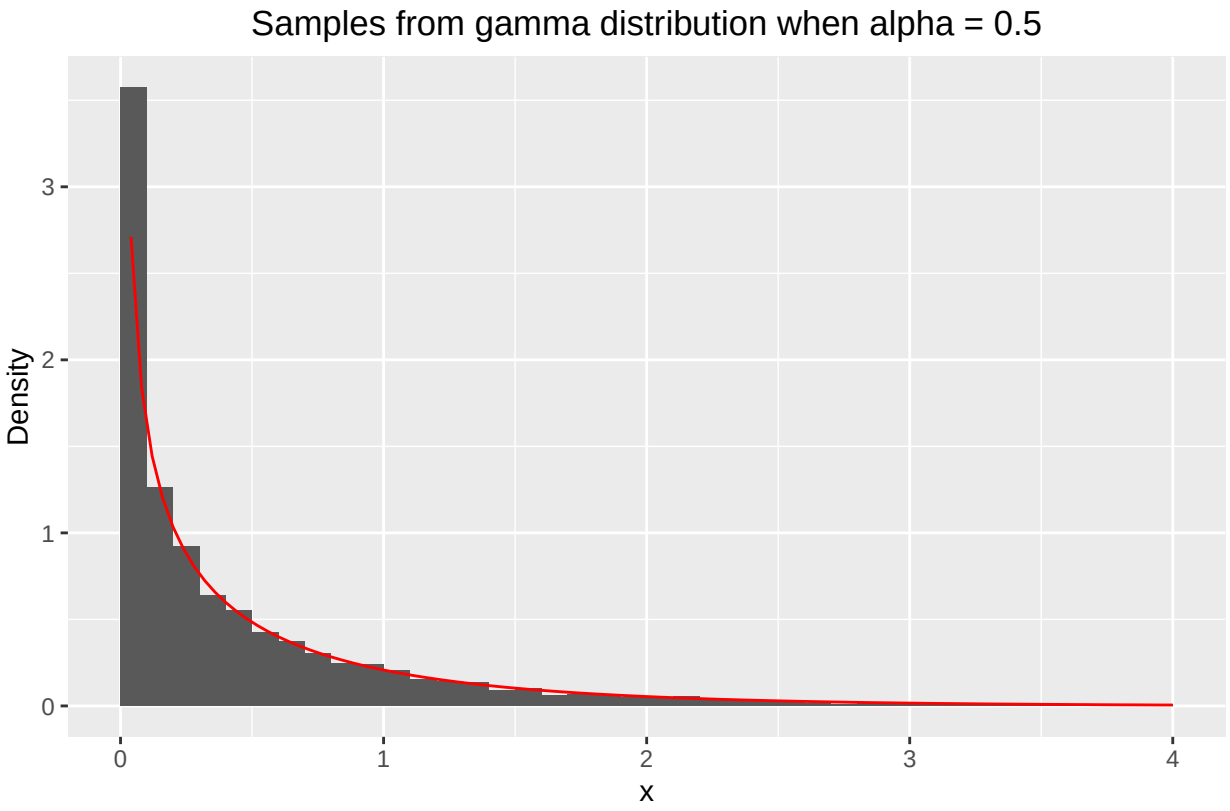
So we choose $c = \frac{1}{\Gamma(\alpha)} \frac{e+\alpha}{e\alpha}$ to satisfy our initial inequality. The acceptance probability is then $\frac{1}{c} = \frac{\Gamma(\alpha)e\alpha}{e+\alpha}$.

Question B1b

To generate n independent samples from f by rejection sampling, we need to find the expression $\alpha = \frac{1}{c}\frac{f(x)}{g(x)}$, with the proposal g from problem A2 and f as given in the exercise. We therefore use c as found in the previous exercise and we sample from g as in problem A2.

```
# f_b1 returns the value of f in this exercise, a gamma
# distribution
f_b1 <- function(alpha, x) {
    gamma = (1/gamma(alpha)) * x^(alpha - 1) * exp(-x)
    return((gamma) * (x > 0))
}

# acceptance_criterion returns the alpha to accept/reject
# in rejection sampling
acceptance_criterion <- function(alpha, x) {
    const = (exp(1) + alpha)/(gamma(alpha) * exp(1) * alpha)
    alpha = f_b1(alpha, x)/(const * g_test(alpha, x))
    return(alpha)
}

# rejection_b generates samples from f by rejection
# sampling n: number of samples to generate alpha: shape
# parameter
rejection_b <- function(alpha, n) {
    out = 1:n

    for (i in 1:n) {
        finished = 0
        while (finished == 0) {
            u = runif(1)
            x = dist1(alpha, 1)[[1]]
            acceptance = acceptance_criterion(alpha, x)
            if (u <= acceptance) {
                out[i] = x
                finished = 1
            }
        }
    }
    return(out)
}


# f_testb is the theoretical function to test the algorithm
# with
f_testb = function(x, alpha) {
    gamma = (1/gamma(alpha)) * x^(alpha - 1) * exp(-x)
    return((gamma) * (x > 0))
}

n = 10000
alpha = 0.5

samples_b1 <- rejection_b(alpha, n)
```

```r
x <- seq(0, 4, 0.001)
data_b1 <- data.frame(samples_b1 = samples_b1)
ggplot(data = data_b1, aes(data_b1$samples_b1)) + geom_histogram(aes(y = ..density..),
    breaks = seq(0, 4, by = 0.1)) + theme(plot.title = element_text(hjust = 0.5)) +
    xlim(c(0, 4)) + labs(title = "Samples from gamma distribution when alpha = 0.5",
    x = "x", y = "Density", caption = "Theoretical and simulated values from the gamma distribution fun
    stat_function(fun = f_testb, args = list(alpha), color = "red")
```



Theoretical and simulated values from the gamma distribution function when alpha = 0.5.

Question B2a

We can define the domain

$$C_f = \left\{(x_1, x_2) : 0 \leq x_1 \leq \sqrt{f^*\left(\frac{x_2}{x_1}\right)}\right\}$$

where

$$f(x) = \begin{cases} x^{\alpha-1}e^{-x}, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

We now want to define some upper and lower bounds such that

$$a = \sqrt{sup_x f(x)} = \sqrt{f(\alpha-1)}b_+ = \sqrt{sup_{x\geq 0} x^2 f(x)} = \sqrt{(\alpha+1)^2 f(\alpha+1)}b_- = \sqrt{sup_{x\leq 0} x^2 f(x)} = 0$$

The supremum for a and b_+ can easily be derived by finding extremal points of the function using differentiation and b_- follows from $f(x) = 0$ for $x \leq 0$.

Question B2b

To sample from the gamma distribution for large values of α, we use the ratio of uniforms method with the values of a, b_- and b_+ as above. To avoid computational errors following large numbers, we use a log transform. To sample from the uniform distribution between 0 and a_- and between 0 and b_+, we use that

$$x_1 = a \cdot u_1 x_2 = b_+ \cdot u_2$$

which is a transform from the standard uniform variables u_1, u_2 to a uniform distribution between 0 and some other limit. In the log scales, this becomes

$$\log(x_1) = \log(a) + \log(u_1) \log(x_2) = \log(b_+) + \log(u_2)$$

The acceptance criterion is then

$$x_1 \leq \sqrt{\frac{x_2^{\alpha-1} e^{-x_2}}{x_1^{\alpha-1} e^{-x_2}}}$$

and on the log scale we get

$$\log(x_1) \leq \frac{1}{2}(\alpha - 1)(\log(x_2) - \log(x_1)) - e^{\log(x_2) - log(x_1)}$$

```
n <- 10000
# f_gamma the kernel of a gamma function
f_gamma <- function(x, alpha) {
    return(x^(alpha - 1) * exp(-x) * (x > 0))
}

alpha <- 400

# gamma_b2b2 samples from gamma distribution with large
# alpha
gamma_b2b2 <- function(alpha, n) {
    x = 1:n
    loga = 0.5 * ((alpha - 1) * log(alpha - 1) + (1 - alpha))
    logb = 0.5 * ((alpha + 1) * log(alpha + 1) - 1 - alpha)
    trials = 1

    for (i in 1:n) {
        logx1 = loga + log(runif(1))
        logx2 = logb + log(runif(1))
        f_condition = 0.5 * ((alpha - 1) * (logx2 - logx1) -
            exp(logx2 - logx1))
        while (logx1 > f_condition) {
            logx1 = loga + log(runif(1))
            logx2 = logb + log(runif(1))
            f_condition = 0.5 * ((alpha - 1) * (logx2 - logx1) -
                exp(logx2 - logx1))
            trials = trials + 1
        }

        x[i] = exp(logx2 - logx1)
    }
```

```
    return(cbind(x, trials - 1))
}


samples_b2b = gamma_b2b2(alpha, n)[, 1]
hist(samples_b2b, breaks = 40, freq = FALSE, main = bquote("Gamma distribution when " ~
    alpha == .(alpha)))
curve(dgamma(x, alpha, 1), col = "red", lwd = 2, add = TRUE)
```

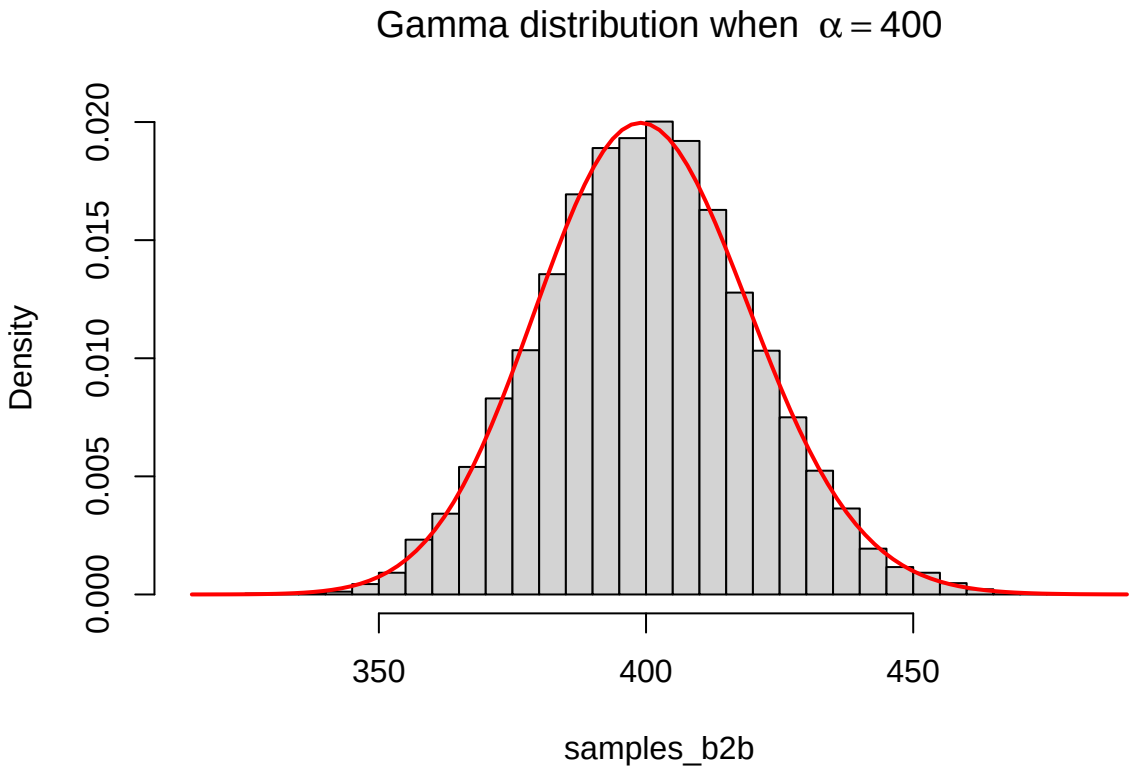
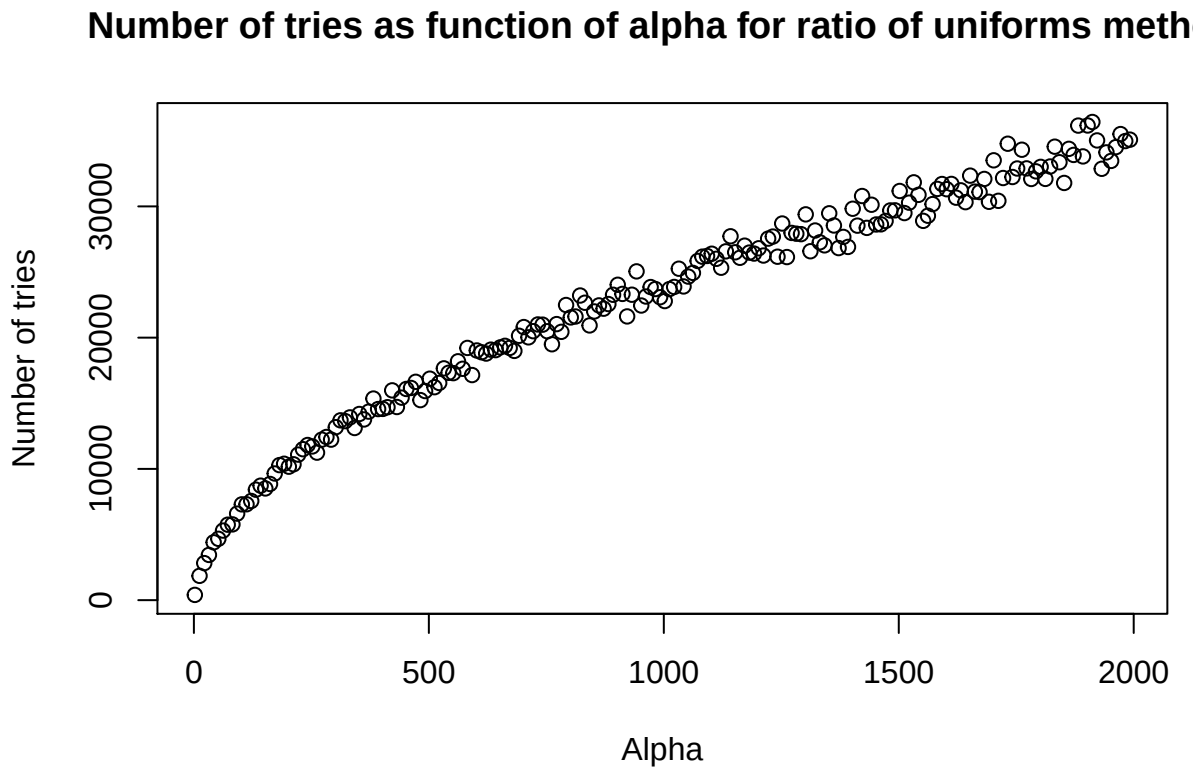


Figure 3: Theoretical and simulated values of the gamma distribution when alpha = 400.

```
test_tries <- function(n) {
    alpha = seq(2, 2000, 10)
    tryvector = 1:length(alpha)
    for (i in 1:length(alpha)) {
        tries = gamma_b2b2(alpha[i], n)[, 2][1]
        tryvector[i] = tries
    }
    return(cbind(alpha, tryvector))

}

test_try = test_tries(1000)
```

```r
plot(test_try[, 1], test_try[, 2], title(main = "Number of tries as function of alpha for ratio of unif
    xlab = "Alpha", ylab = "Number of tries")
```



From the plot we see that a larger value of α implies that we need more tries to accept one sample. This makes sense, as we are approaching a more extreme case when α becomes large.

Question B3

We want to generate n samples of the gamma distribution

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta}$$

for $\alpha > 0$ and $\beta > 0$. We use that $Y = \frac{X}{\beta}$ is Gamma(α, β) distributed if X is Gamma(α) distributed. To show this, use the moment generating function of Y and the properties of moment generating functions.

$$M_Y(t) = M_X\left(\frac{t}{\beta}\right) = \left(1 - \frac{t}{\beta}\right)^{-\alpha}$$

which we recognize as the MGF of a Gamma(α, β) variable. When $\alpha = 1$, we recognize this as the MGF of an Exponential(β) distribution, so therefore we can sample from the exponential distribution in this case. We use the rejection sampling algorithm for $\alpha \in (0, 1)$ and the ratio of uniforms algorithm for $\alpha > 1$.

```r
# Delete this before merging the files, only for help n:
# number of samples to generate rate: the rate expdist
# returns a vector with generated random numbers from
# exponential distribution
expdist <- function(rate, n) {
    u <- runif(n)
    x <- -log((1/rate) * (1 - u))/rate
    list = list(x, u)
    return(list)
}

# gammaab samples from the gamma distribution for all
# values of alpha and beta > 0
gammaab <- function(alpha, beta, n) {
    if (alpha > 0 && alpha < 1) {
        X = rejection_b(alpha, n)/beta
    } else if (alpha == 1) {
        X = expdist(1, n)[[1]]
    } else if (alpha > 1) {
        X = gamma_b2b2(alpha, n)[, 1]
    } else {
        print("alpha not valid")
        return(0)
    }
    Y = X/beta
    return(Y)
}

# theoretical_gamma is the theoretical function to test the
# algorithm with for alpha, beta > 0
theoretical_gamma = function(x, alpha, beta) {
    gamma = ((beta^alpha)/gamma(alpha)) * x^(alpha - 1) * exp(-beta *
        x)
    return((gamma) * (x > 0))
}

alpha = 2
beta = 2
n = 10000

samples_ab <- gammaab(alpha, beta, n)
hist(samples_ab, breaks = 40, freq = FALSE, main = bquote("Gamma distribution when " ~
    alpha == .(alpha) ~ beta == .(beta)))
curve(theoretical_gamma(x, alpha, beta), col = "red", lwd = 2,
    add = TRUE)
```

Question B4a

We want to show that if $x \sim \text{Gamma}(\alpha, 1)$ and $y \sim \text{Gamma}(\beta, 1)$ are independent, then $z = \frac{x}{x+y}$ is $\text{Beta}(\alpha, \beta)$ distributed. Define $v = x + y$, such that $x = zv$ and $y = v(1 - z)$. Then the jacobian of the transformation is v, so by using the transformation of variables formula, the joint distribution of z and v becomes

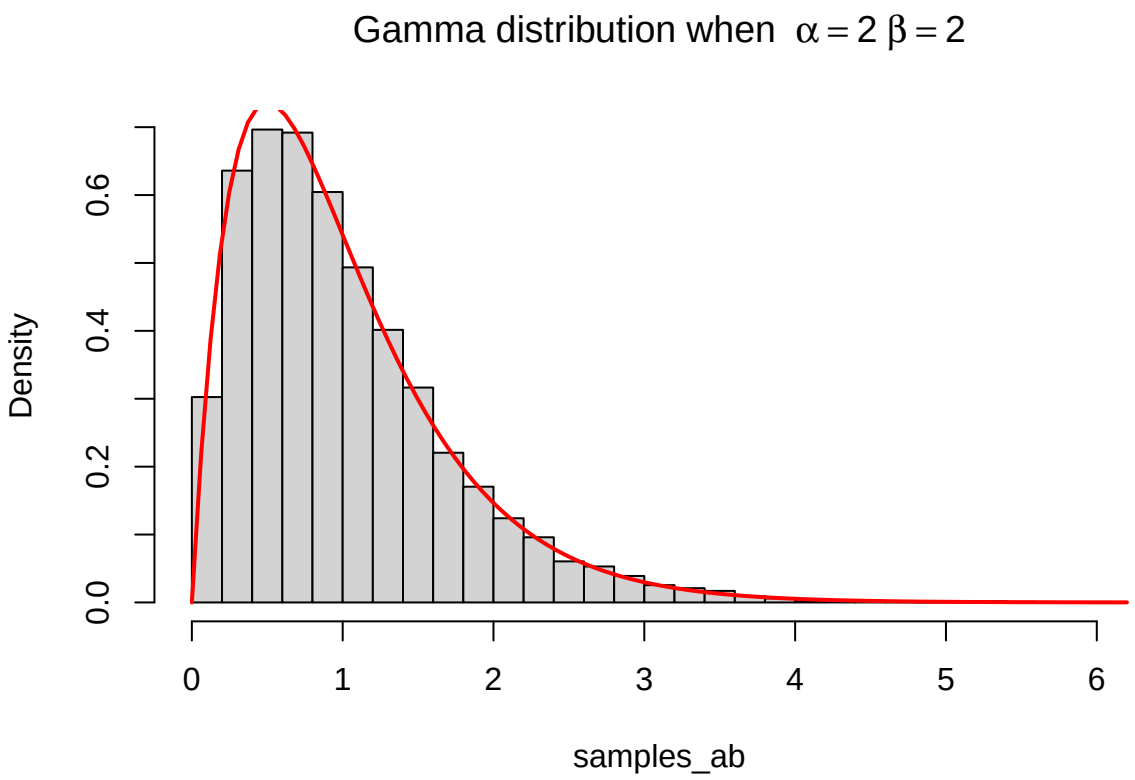


Figure 4: Theoretical and simulated values of gamma distribution when alpha = 2, beta = 2

$$f_{zv}(z,v) = \frac{v}{\Gamma(\alpha)\Gamma(\beta)}(vz)^{\alpha-1}(v(1-z))^{\beta-1}e^{-v}$$

which can be written as

$$\frac{v^{\alpha+\beta-1}}{\Gamma(\alpha)\Gamma(\beta)}z^{\alpha-1}(1-z)^{\beta-1}e^{-v}$$

and we see that this is just a gamma distribution and a beta distribution multiplied, so $v \sim \text{Gamma}(\alpha+\beta, 1)$ and $z \sim \text{Beta}(\alpha, \beta)$.

Question B4b

To sample from a $\text{Beta}(\alpha, \beta)$ distribution, we use the result above.

```
# sample_beta samples from beta(alpha, beta) distribution
# n: number of samples
sample_beta <- function(alpha, beta, n) {
    x = gammaab(alpha, 1, n)
    y = gammaab(beta, 1, n)
    z = x/(x + y)
    return(z)
}

alpha = 0.5
beta = 0.5
n = 10000
samples_beta <- sample_beta(alpha, beta, n)
hist(samples_beta, breaks = 40, freq = FALSE, main = bquote("Beta distribution when " ~
    alpha == .(alpha) ~ beta == .(beta)))
curve(dbeta(x, alpha, beta), col = "red", lwd = 2, add = TRUE)
```

Problem C

Question C1

We want to estimate $P(X > 4)$ by Monte Carlo integration, so we use that we can approximate the integral $\int h(x)f(x)dx$ with the sum $\frac{1}{N}\sum_{i=1}^{N} h(x_i)$. Since we only want a part of the domain, we use the indicator function $I(x > 4)$ as $h(x)$. We use the Box-Muller algorithm from before to draw samples from the normal distribution.

```
# indicator_func return
indicator_func <- function(x) {
    return(x > 4)
}

# monte_carlo estimates the probability and the variance n:
# number of samples to generate when estimating
rm("c")
monte_carlo <- function(n) {
```

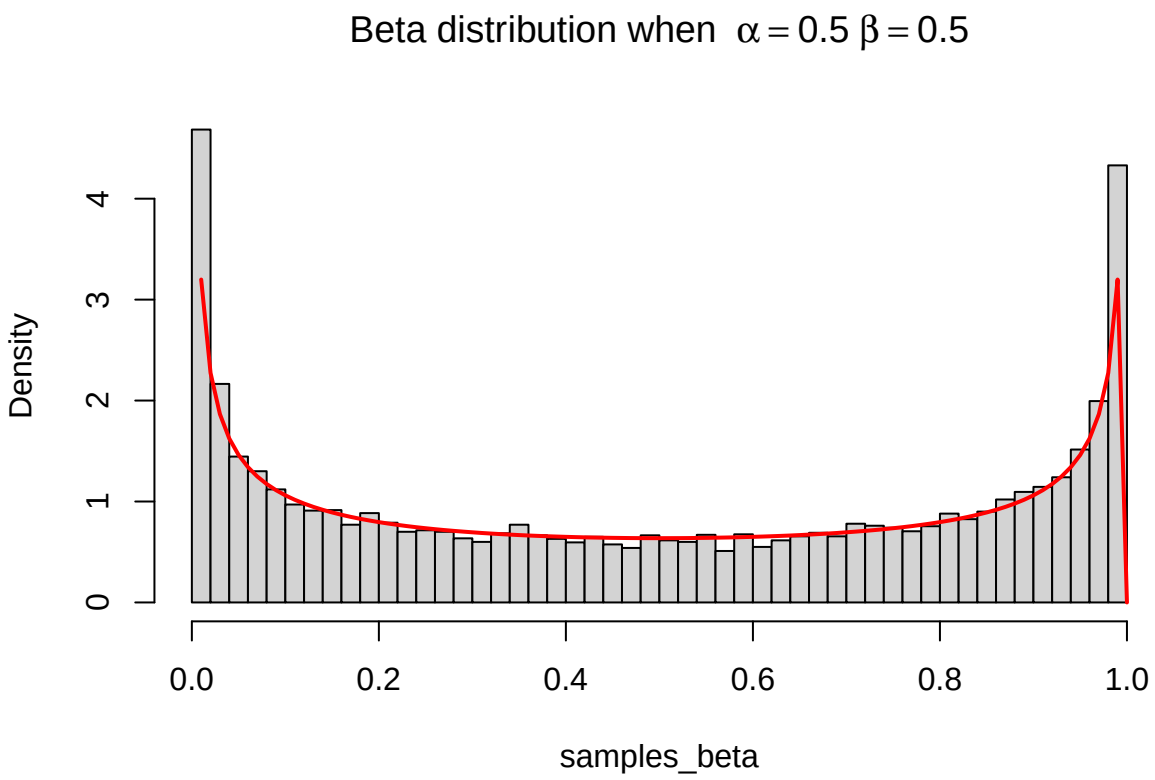


Figure 5: Theoretical and simulated values of the beta distribution when alpha = 0.5, beta = 0.5.

```
    normal = mynormal(n)
    h = indicator_func(normal)
    theta_estimate = (1/n) * sum(h)
    variance_estimate = var(h)/(n - 1)
    return(c(theta_estimate, variance_estimate))
}

n = 1e+05

monte_carlo_test <- monte_carlo(n)
# lower: lower limit of confidence interval upper: upper
# limit of confidence interval

lowermc <- monte_carlo_test[1] + qnorm(0.025) * sqrt(monte_carlo_test[2])

uppermc <- monte_carlo_test[1] + qnorm(0.975) * sqrt(monte_carlo_test[2])
```

The estimate of the probability $P(X > 4)$ is 10^{-5}. The confidence interval for θ is $[-9.5997378 \times 10^{-6}, 2.9599738 \times 10^{-5}]$.

Question C2

We want to use importance sampling to estimate $P(X > 4)$. To determine the normalizing constant c, we use that the density $g(x)$ must integrate to 1, and we get that $c = e^8$. To estimate $P(X > 4)$ by importance sampling, we can estimate $\int f(x)h(x)dx$ with $\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)h(x_i)}{g(x_i)}$, where x_i are sampled from the distribution $g(x)$, and in our case this is done by inversion sampling.

The function we want to sample from is

$$g(x) = \begin{cases} cxe^{-0.5x^2} & x > 4 \\ 0 & \text{otherwise} \end{cases}$$

By integrating from $t = 4$ to an arbitrary x, we find that the cumulative distribution is

$$G(x) = 1 - e^{8-0.5x^2}$$

Solving this equation for x we get

$$G^{-1}(u) = \sqrt{16 - 2log(1-u)}$$

```
# importance_sampling estimates the probability via the
# density g n: number of samples to generate
importance_sampling <- function(n) {
    u = runif(n)
    x = sqrt(16 - 2 * log(1 - u))
    f = 1/sqrt(2 * pi) * exp(-0.5 * x^2)
    g = x * exp(-0.5 * x^2 + 8)
    h = indicator_func(x)
    theta_estimate = (1/n) * sum(h * f/g)
    variance_estimate = (1/(n * (n - 1))) * sum(h * f/g - theta_estimate)^2
    return(c(theta_estimate, variance_estimate))
}
```

```
n = 10^5
importance_test <- importance_sampling(n)

# lower: lower limit of confidence interval upper: upper
# limit of confidence interval
lowerim <- importance_test[1] + qnorm(0.025) * sqrt(importance_test[2])
upperim <- importance_test[1] + qnorm(0.975) * sqrt(importance_test[2])
```

The estimate of $P(X > 4)$ with importance sampling is 3.167659×10^{-5}. The confidence interval for θ with importance sampling is $[3.167659 \times 10^{-5}, 3.167659 \times 10^{-5}]$. Compared to the confidence interval for Monte Carlo integration $[-9.5997378 \times 10^{-6}, 2.9599738 \times 10^{-5}]$, we see that the confidence interval from importance sampling is smaller, and therefore we prefer importance sampling over Monte Carlo integration for this problem.

We want to find how many samples we need to get the same precision for Monte Carlo integration as with importance sampling. We test with some values of n.

```
monte_carlo_test5 <- monte_carlo(10^5)
monte_carlo_test6 <- monte_carlo(10^6)
monte_carlo_test7 <- monte_carlo(10^7)

lowermc5 <- monte_carlo_test5[1] + qnorm(0.025) * sqrt(monte_carlo_test5[2])

uppermc5 <- monte_carlo_test5[1] + qnorm(0.975) * sqrt(monte_carlo_test5[2])

lowermc6 <- monte_carlo_test6[1] + qnorm(0.025) * sqrt(monte_carlo_test6[2])

uppermc6 <- monte_carlo_test6[1] + qnorm(0.975) * sqrt(monte_carlo_test6[2])

lowermc7 <- monte_carlo_test7[1] + qnorm(0.025) * sqrt(monte_carlo_test7[2])

uppermc7 <- monte_carlo_test5[1] + qnorm(0.975) * sqrt(monte_carlo_test7[2])
```

The confidence intervals for $n = 10^5, 10^6$ and 10^7 with Monte Carlo integration are respectively: $[0, 0]$, $[1.0456788 \times 10^{-5}, 2.7543212 \times 10^{-5}]$, and $[2.8021463 \times 10^{-5}, 3.4785366 \times 10^{-6}]$. We see that the confidence interval has a similar size for Monte Carlo integration as for importance sampling for $n = 10^6$, but with $n = 10^7$ the precision is even better.

Question C3a

We want to produce n pairs of antithetic variates from the specified distribution. The estimate is given by averaging the samples from importance sampling. The function below also estimates θ and the variance to be used in the confidence interval.

```
# antithet gives an estimate of the probability and the
# variance n: number of sample pairs to generate
antithet <- function(n) {
    u = runif(n)
    x1 = sqrt(16 - 2 * log(u))
    x2 = sqrt(16 - 2 * log(1 - u))
    f1 = (1/sqrt(2 * pi)) * exp(-0.5 * x1^2)
    f2 = (1/sqrt(2 * pi)) * exp(-0.5 * x2^2)
```

```
    g1 = x1 * exp(-0.5 * x1^2 + 8)
    g2 = x2 * exp(-0.5 * x2^2 + 8)
    h1 = indicator_func(x1)
    h2 = indicator_func(x2)

    theta_estimate1 = (1/n) * sum(h1 * f1/g1)
    theta_estimate2 = (1/n) * sum(h2 * f2/g2)
    theta_estimate = (theta_estimate1 + theta_estimate2)/2

    variance_estimate1 = (1/(n - 1)) * sum(h1 * f1/g1 - theta_estimate1)^2
    variance_estimate2 = (1/(n - 1)) * sum(h2 * f2/g2 - theta_estimate2)^2
    cov = (1/(n - 1)) * sum(h1 * f1/g1 - theta_estimate1) * sum(h2 *
        f2/g2 - theta_estimate2)
    variance_estimate = (1/(4 * n)) * (variance_estimate1 + variance_estimate2 +
        2 * cov)

    return(c(theta_estimate, variance_estimate))
}
```

Question C3b

Now we use the function above to estimate the probability $\theta = P(X > 4)$ and the variance of θ. Because the function gives pairs of random variables, we will get a similar result if we use antithetic variables with $n = 5 \cdot 10^4$ as if we use importance sampling with $n = 10^5$ variables.

```
antithetic_test <- antithet(5 * 10^4)

loweranti <- antithetic_test[1] + qnorm(0.025) * sqrt(antithetic_test[2])

upperanti <- antithetic_test[1] + qnorm(0.975) * sqrt(antithetic_test[2])
```

The estimate of the probability $P(X > 4)$ with antithetic variables is 3.1670406×10^{-5}. The confidence interval is $[3.1670406 \times 10^{-5}, 3.1670406 \times 10^{-5}]$. We see that as expected, the precision of the antithetic variables and the importance sampling is similar. We must use a value of n that is half as big for antithetic variables as for importance sampling because antithetic variables produces pairs of variables, and therefore the total amount of simulated samples will be the same in the end.

Problem D

Question D1

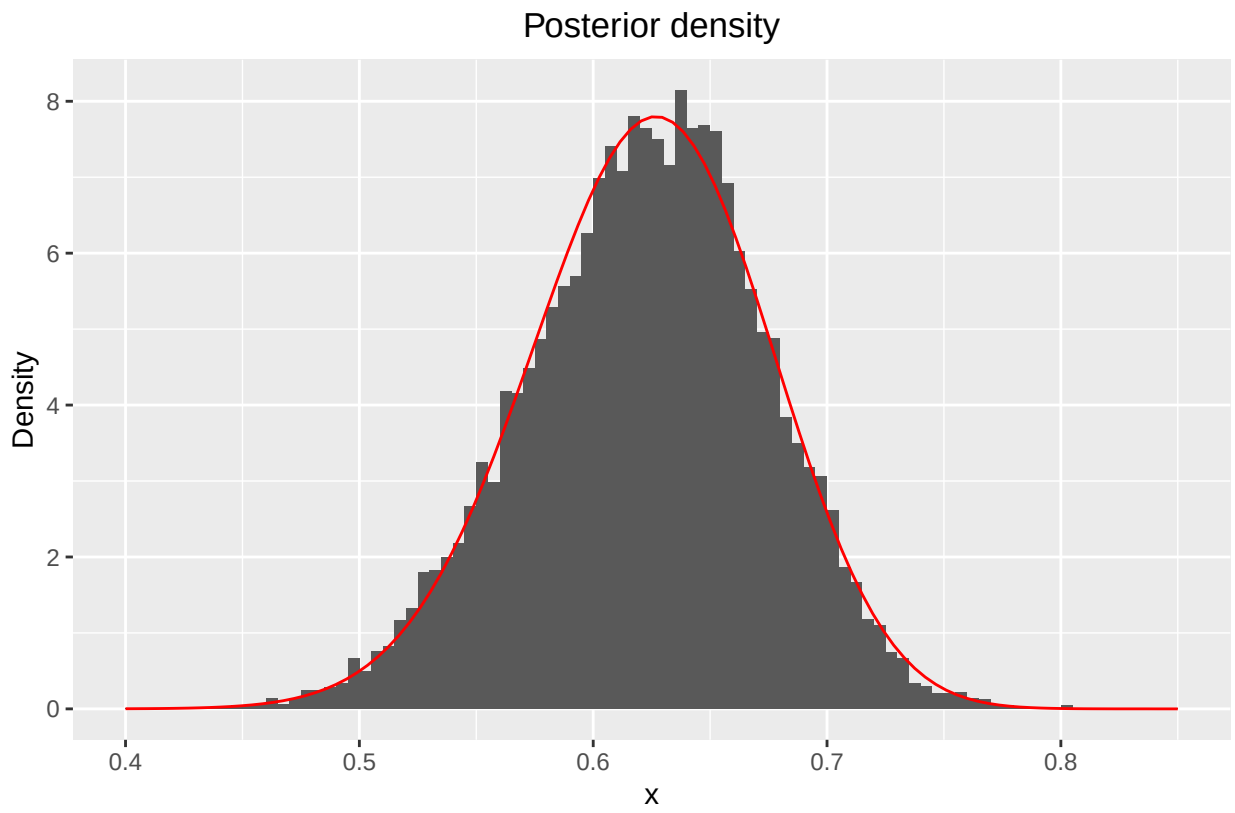
We want to sample from

$$f(\theta|\mathbf{y}) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$$

by using rejection sampling and using $g(\theta) = U(0,1)$ as the proposal density. We use the maximum of f as the constant c in rejection sampling, to ensure $\frac{f(\theta|y)}{g(\theta)} \leq c$.

```
# f is the posterior function to sample from, not
# normalized theta: parameter between 0 and 1, sampled from
# U(0,1) y: observed values
f <- function(theta, y) {
```

```
    return((2 + theta)^y[1] * (1 - theta)^(y[2] + y[3]) * theta^y[4])
}

# f_normalized is the posterior function, normalized
f_normalized <- function(theta, y) {
    c = integrate(f, 0, 1, y)$value
    return(f(theta, y)/c)
}

# alpha finds the acceptance probability alpha
alpha <- function(theta, y) {
    c = optimize(f, interval = c(0, 1), maximum = TRUE, y = y)$objective
    return(f(theta, y)/c)
}

# rejection_d samples from the posterior by rejection
# sampling n: number of samples to generate
rejection_d <- function(y, n) {
    out = 1:n
    g = function() {
        return(runif(1))
    }

    for (i in 1:n) {
        finished = 0
        while (finished == 0) {
            u = runif(1)
            theta = g()
            alpha = alpha(theta, y)
            if (u <= alpha) {
                out[i] = theta
                finished = 1
            }
        }
    }
    return(out)
}

y = c(125, 18, 20, 34)
n = 10000

samples_d1 <- rejection_d(y, n)
x <- seq(0, 1, 0.001)
data_d1 <- data.frame(samples_d1 = samples_d1)
ggplot(data = data_d1, aes(data_d1$samples_d1)) + geom_histogram(aes(y = ..density..),
    breaks = seq(0, 1, by = 0.005)) + theme(plot.title = element_text(hjust = 0.5)) +
    xlim(c(0.4, 0.85)) + labs(title = "Posterior density", x = "x",
    y = "Density", caption = "Posterior density for theta given y.") +
    stat_function(fun = f_normalized, args = list(y), color = "red")
```



Posterior density for theta given y.

From the plot we see that the theoretical and the simulated distribution follow each other.

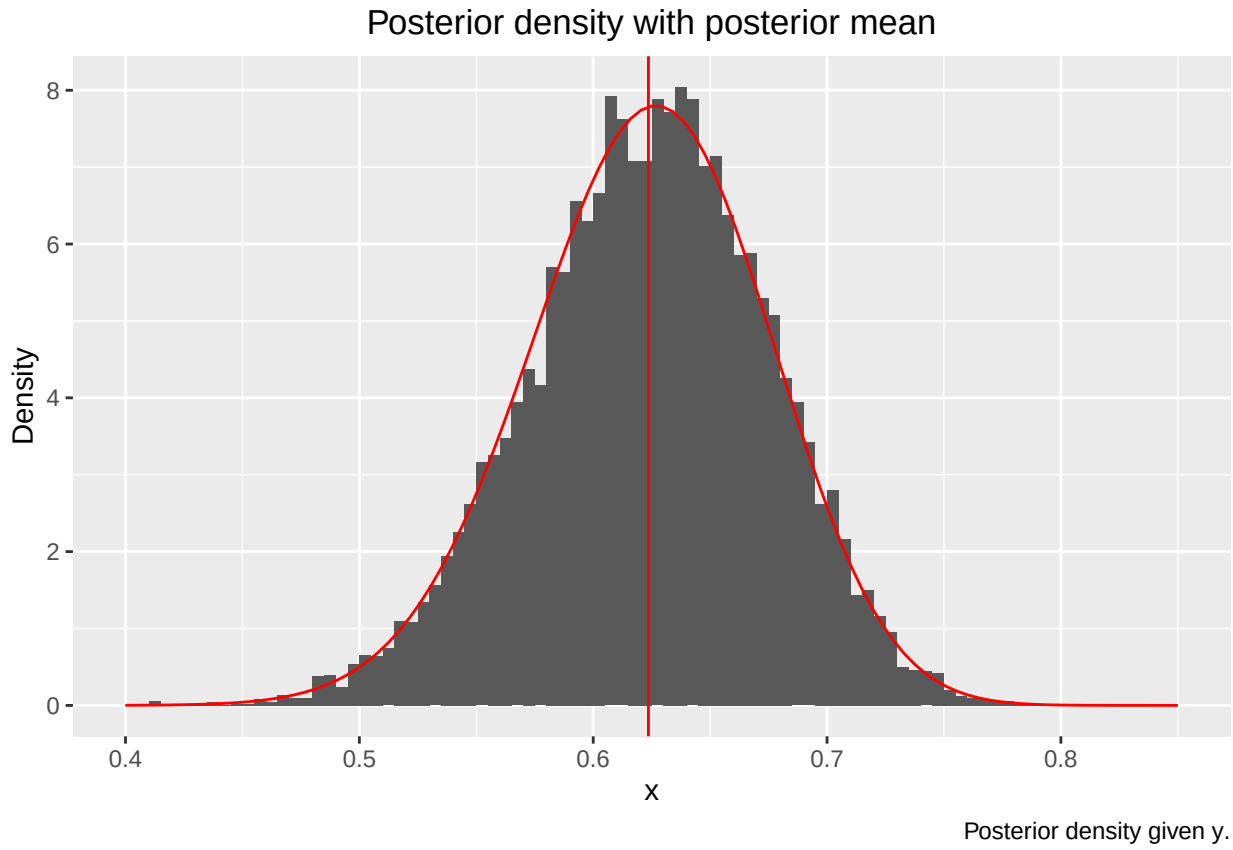
We then use Monte Carlo integration with $M = 10000$ samples to estimate the posterior mean of θ, that is, the integral $\int \theta f(\theta|y)d\theta$ approximated by $\frac{1}{N}\sum_{i=1}^{N}\theta_i$.

```
M = 10000
# post_mean_mc: the estimated posterior mean
thetas <- rejection_d(y, M)
post_mean_mc <- mean(thetas)

# inside_integral is what we want to integrate
inside_integral = function(theta, y) {
    return(theta * f_normalized(theta, y))
}

# numerical_mean is the mean found by numerical integration
numerical_mean <- integrate(inside_integral, 0, 1, y)[[1]]
```

The estimate of the mean of θ found by Monte Carlo integration is 0.623628. The estimate of the mean found by numerical integration is 0.6228061. The two values are very similar. We plot the distribution of the simulated values of θ together with the theoretical distribution and the Monte Carlo estimate.

```
x <- seq(0, 1, 0.001)
data_d1 <- data.frame(thetas = thetas)
ggplot(data = data_d1, aes(data_d1$thetas)) + geom_histogram(aes(y = ..density..),
    breaks = seq(0, 1, by = 0.005)) + theme(plot.title = element_text(hjust = 0.5)) +
    xlim(c(0.4, 0.85)) + labs(title = "Posterior density with posterior mean",
```

```
    x = "x", y = "Density", caption = "Posterior density given y.") +
    stat_function(fun = f_normalized, args = list(y), color = "red") +
    geom_vline(xintercept = post_mean_mc, color = "red")
```



Question D3

The overall acceptance probability is

$$P(U \leq \frac{f(\theta)}{cg(\theta)}) = \int_{-\infty}^{\infty} \frac{f(\theta)}{cg(\theta)} g(\theta) d\theta = \int_{-\infty}^{\infty} \frac{f(\theta)}{c} d\theta = \frac{1}{c}$$

which implies that the number of trials to simulate one sample is c.

We want to count how many trials we need to simulate one sample, and we check it with the theoretical value of c. We just copy the algorithm from before and add a counter.

```
# rejection_d_count samples from the posterior by rejection
# sampling and counts the number of trials to one
# acceptance n: number of samples to generate
rejection_d_count <- function(y, n) {
    out = 1:n
    g = function() {
        return(runif(1))
    }
```

```
    for (i in 1:n) {
        finished = 0
        count = 0
        while (finished == 0) {
            u = runif(1)
            theta = g()
            alpha = alpha(theta, y)
            count = count + 1
            if (u <= alpha) {
                out[i] = count
                count = 0
                finished = 1
            }
        }
    }
    return(out)
}

y = c(125, 18, 20, 34)
n = 10000

# simulated number of trials
number_of_trials <- mean(rejection_d_count(y, n))

# c_const finds the constant c, i.e. the numerical
# computation of the theoretical number of trials for one
# acceptance
c_const <- function(y) {
    c = optimize(f_normalized, interval = c(0, 1), maximum = TRUE,
        y = y)$objective
    return(c)
}

c_numerical <- c_const(y)
```

The number of trials in the simulation until one acceptance is 7.8049, and the numerical calcualtion of the theoretical number of trials is 7.7993075.

Question D4

We want to use the Beta(1,5) distribution as the prior, instead of the U(0,1) distribution and importance sampling with self-normalizing weights. The new prior with a Beta(1,5) distribution is:

$$f(\theta) = \frac{\Gamma(6)}{\Gamma(5)}(1-\theta)^4 = 5(1-\theta)^4$$

And the new posterior destribution becomes

$$f_{new}(\theta|\mathbf{y}) \propto f(\theta)f(\mathbf{y}|\theta) = 5(1-\theta)^4(2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$$

We use importance sampling with self-normalizing weights, such that the weights become

$$w(\theta_i) = \frac{f_{new}(\theta_i|\mathbf{y})}{f(\theta_i|\mathbf{y})}$$

and because our estimate is based on the samples of θ in part D 2, we use the unnormalized $f(\theta|\mathbf{y})$ as $g(\theta)$ from part D 2 in the expression for the weights $w(\theta_i)$. The estimate of the mean is then

$$\mu = \frac{\sum_{i=1}^{N} \theta_i w(\theta_i)}{\sum_{i=1}^{N} w(\theta_i)}$$

which is biased, but for $n \to \text{inf}$ it becomes approximately unbiased.

```
# fnew calculates the unnormalized new posterior
# distribution
fnew <- function(theta, y) {
    fnew = 5 * (1 - theta)^4 * (2 + theta)^y[1] * (1 - theta)^(y[2] +
        y[3]) * theta^y[4]
}

# weight calculates the self-normalizing weights to use in
# importance sampling
weight <- function(theta, y) {
    return(fnew(theta, y)/f(theta, y))
}

# fnew_normalized calculates the normalized new posterior
# distribution
fnew_normalized <- function(theta, y) {
    const = integrate(fnew, 0, 1, y)$value
    return(fnew(theta, y)/const)
}

fnew_normalized(0.5, y)
```

```
## [1] 1.378492
```

```
# importance_d4 estimates the mean via importance sampling
# y: observed values n: number of samples to use in the
# estimation
importance_d4 <- function(y, n) {
    theta = rejection_d(y, n)
    estimate = sum(theta * weight(theta, y))/sum(weight(theta,
        y))
    return(estimate)
}

n = 10000
simulated_d4 <- importance_d4(y, n)

# integrand_d4 is the expression to integrate
integrand_d4 <- function(theta, y) {
    return(fnew_normalized(theta, y) * theta)
}

# numerical_d4 numerically calculates the integral
numerical_d4 <- function(y) {
    return(integrate(integrand_d4, 0, 1, y)$value)
```

```
}

num_d4 <- numerical_d4(y)
```

The simulated expected value is 0.595192, and the numerical value is 0.5959316. We see that they are equal.