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 distrobution 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

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

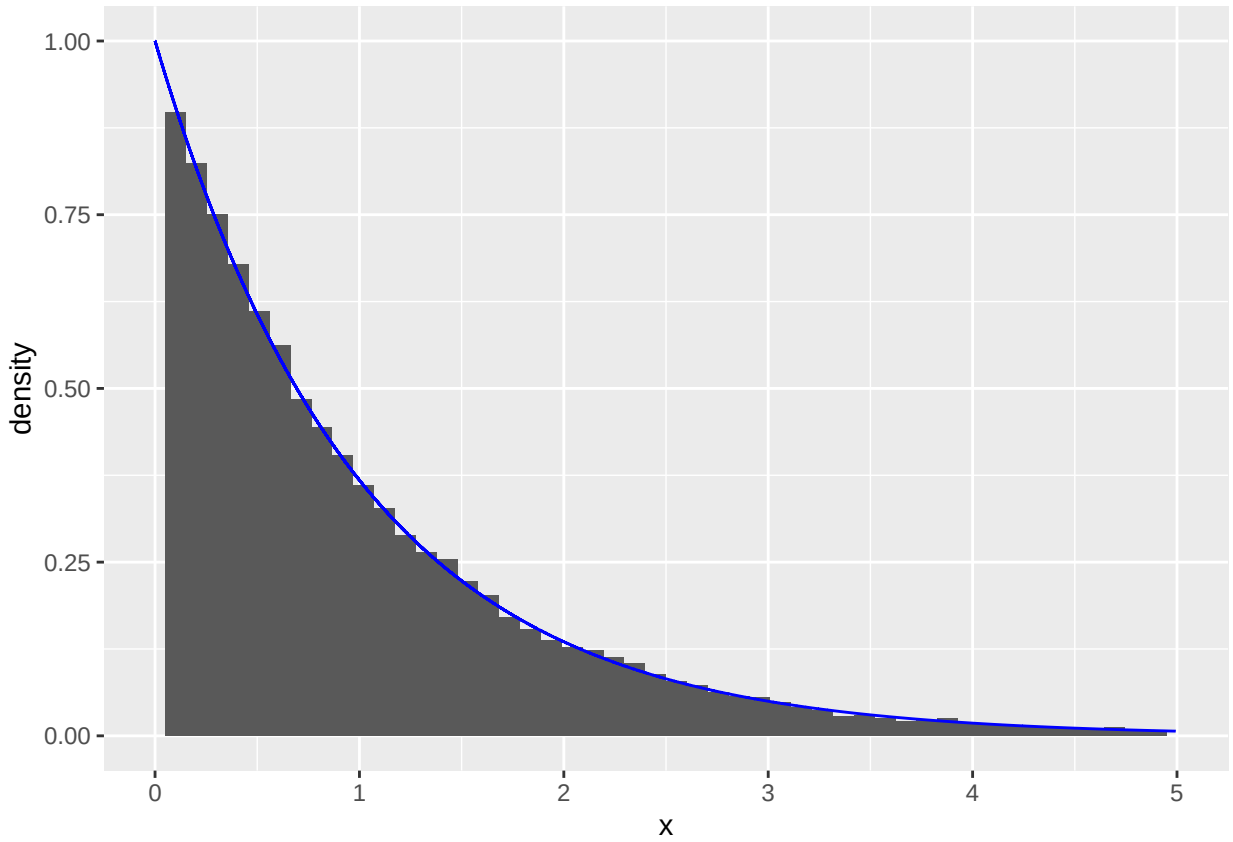
We now want to show that this is true using simulation from the inverse cumulative distribution

```
# 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]]
library(ggplot2)
ggplot(data.frame(x=x))+geom_histogram(aes(x=x, y = ..density..), bins = 50)+geom_line(data = data.fram
```

```
## Warning: Removed 376 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```

```
## Warning: Removed 376 row(s) containing missing values (geom_path).
```



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 2 a

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

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

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 2 b

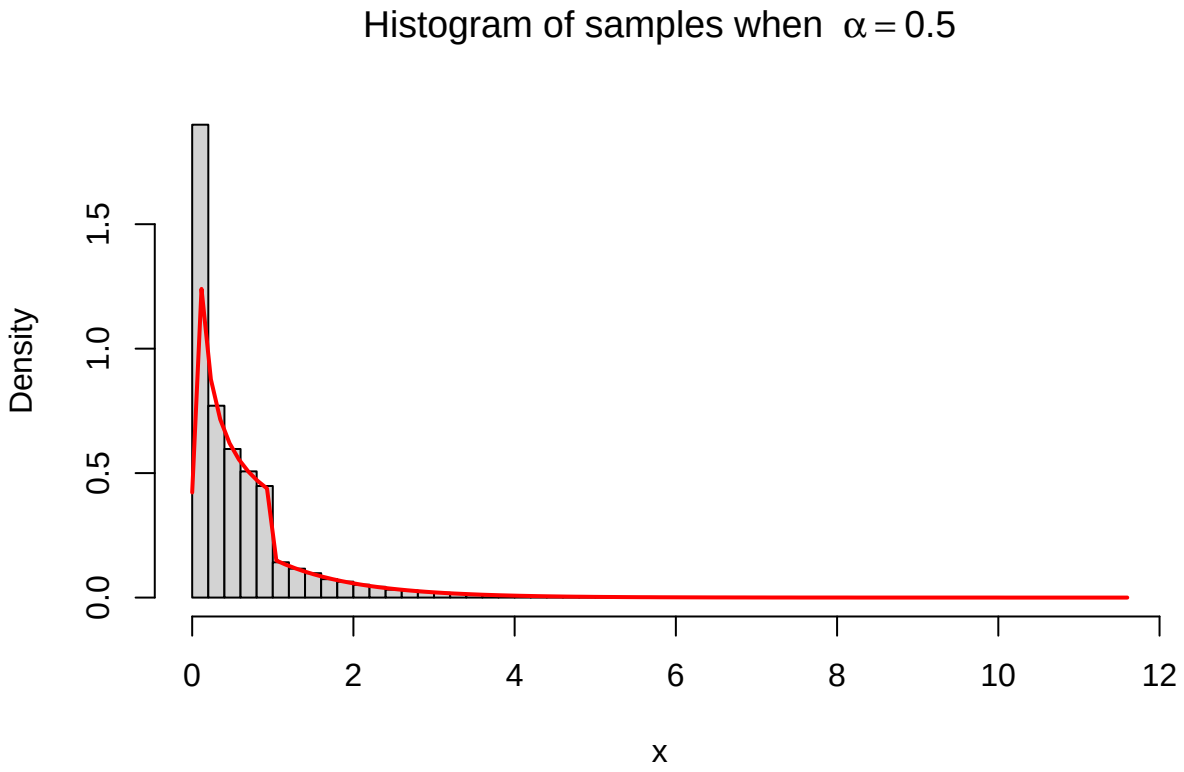
This distribution can be plotted by using the inverse cumulative distribution as before.

```
# 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 = 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]]

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



Question 3 a The normalizing constant is found by using that a PDF must integrate to 1, and therefore we get $c = \alpha$.

Question 3 b 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:

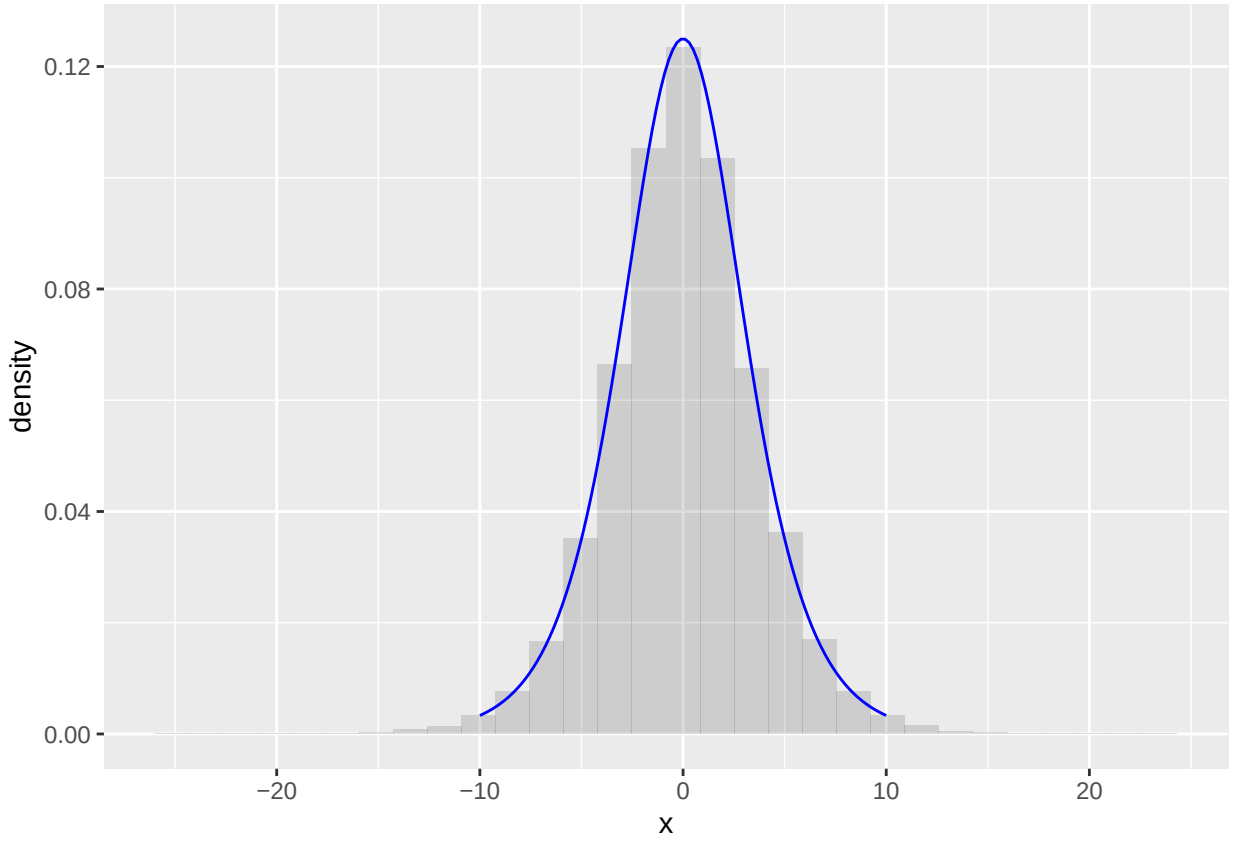
$$F^{-1}(x) = \frac{1}{\alpha} ln(\frac{-y}{y-1})$$

Question 3 c We want to use the inversion method to generate random samples from the distribution in question 3b, 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,
```

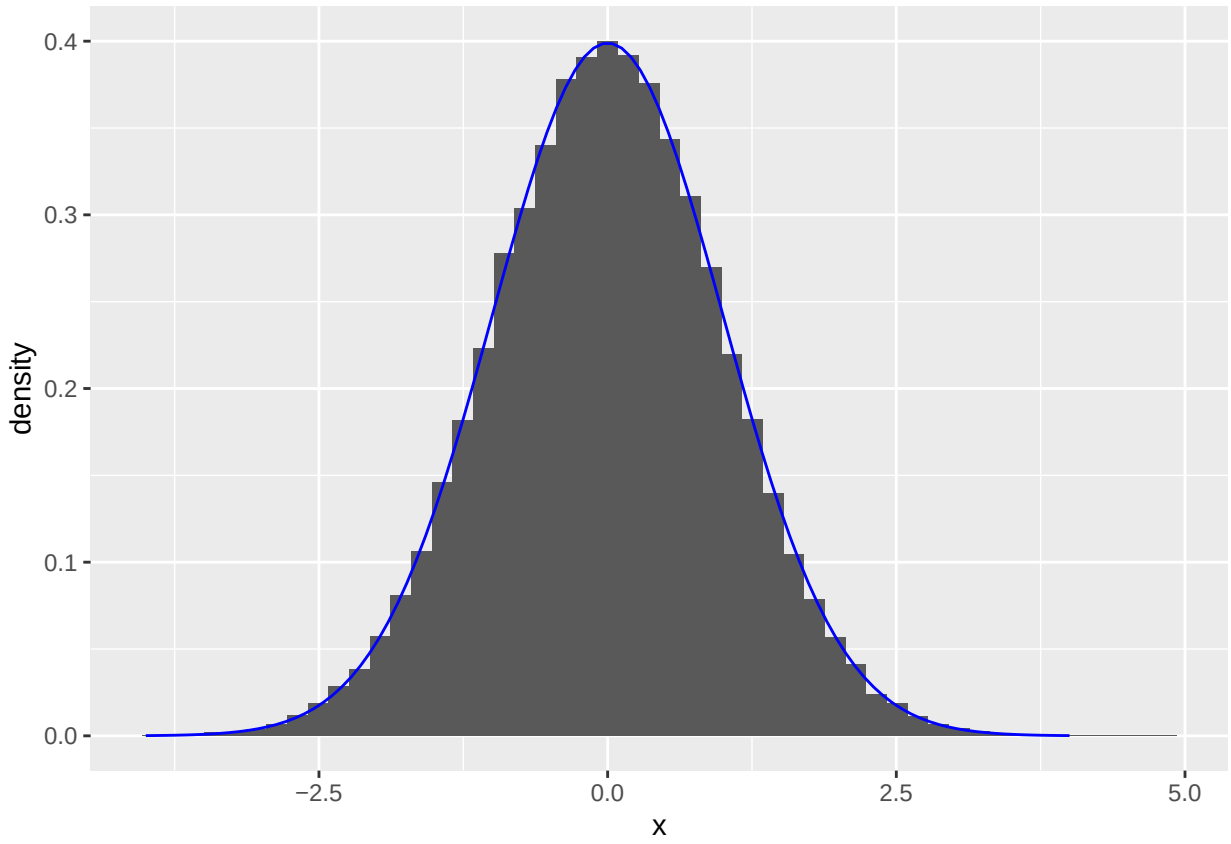
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



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

Question 4 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 = 100000
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,
```



Question 5

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 $y = \mu + M^T x$ because of how the Cholesky function is implemented in R.

```
# y: matrix to store the realisations of the multivariate normal distribution
n = 100000
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] 0.990185 1.987740
```

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

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

```
print(sigma)
```

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

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

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

```
print(covmatrix)
```

```
##          [,1]     [,2]
## [1,] 1.994406 1.008966
## [2,] 1.008966 3.035359
```

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