# Exercise 2 - Random Number Generation through CDF and acceptance-rejection sampling

Florian Stadler

2024-10-07

## Contents

## 1 Random Number Generation Algorithm

The key idea of pseudo random number generation (PRNG) is to create numbers, that look like truely random generated numbers(RNG). As true RNGs are usually bound to real physical processes, they are not easily availabe. The Linear Congruential Random Number Generation (LCRNG) Algorithm is one of many PRNG algorithms. LCRNG has that name since it is based on a linear formula. Inputs are the modules (m), multiplier(a) and the increment (c). Based on a seeds, the algorithm produces a pseudo random sample of size n of numbers between 0 and m-1. The algorithm is “Pseudo”, since the sequence is deterministic to those who know the seed and the function definition. If n is larger than m, the sequence will repeat itself since the algorithm is cyclic.

Lets look at the LCRNG-variation of the slides:

```
lcrng <- function(n,m,a,c=0,x0){
  us <- numeric(n)
  for (i in 1:n){

    x0<- (a*x0+c) %% m
    us[i] = x0
    }
return (us)
}
lcrng(6,7,3,1,2)
```

```
## [1] 0 1 4 6 5 2
```

We created a pseudo random sample of 6 numbers ranging between 0 and 6. In this case, we got a sample without replacing. However, this is not always be the case as repetition occurs for some sets of inputs.

### 1.1 Trying and comparing different values of m, a to illustrate the behaviour of the method as described on slide 18.

We will now show some cases, where the cycle-length is smaller than m or even 1. We will also show the reptitiveness of the algorithm:

```
lcrng(16,16,5,1,3)
```

```
##  [1]  0  1  6 15 12 13  2 11  8  9 14  7  4  5 10  3
```

```
lcrng(16,6,2,1,3)
```

```
##  [1] 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3
```

```
lcrng(16,9,4,1,3)
```

```
##  [1] 4 8 6 7 2 0 1 5 3 4 8 6 7 2 0 1
```

```
lcrng(16,4,2,1,3)
```

```
##  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

## 2 Exponential Distribution

The exponential distribution has the cdf

$$F(x) = 1 - e^{-\lambda x},\ \lambda > 0.$$

### 2.1 Assume you can generate easily uniform random variables. How can you obtain then a random sample from the exponential distribution?

If we have a uniform random variable, we can get a random sample from the exponential distribution by applying the inverse of the cdf on a uniform random variable u. The inverse is

$$F^{-1}(u) = -\frac{1}{\lambda}\log(1-u)$$

We will now write an R function, which utilizes the inverse to create a random sample. With runif(n) we get a uniform random sample of size n.

```r
inv.exp.cdf <- function(x, lambda){
  return( -1/lambda * log(1-x) )
}
random.sample.exp <- function(n,lambda=1){
  u <- runif(n)
  value <- inv.exp.cdf(u, lambda)
  return(value)
}
random.sample.exp(1,0.1)
```

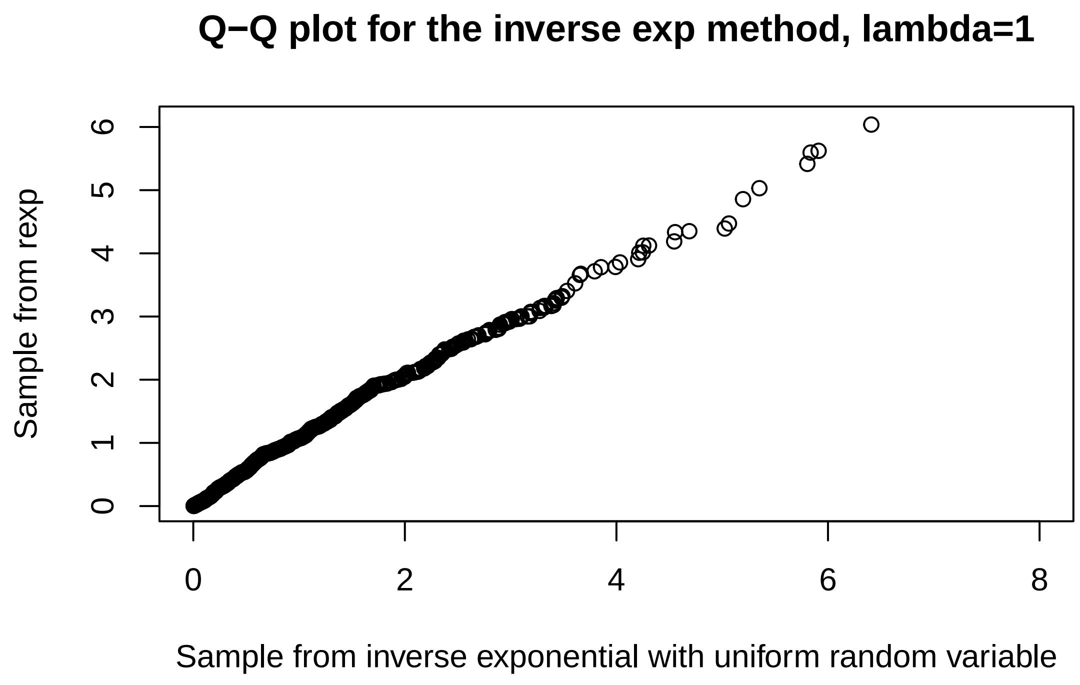
```
## [1] 19.541
```

random.sample.exp gives us a random number in a certain range. The distribution of the obtained sample values is dependend on lambda. To investigate this, we will create random sample of size n=1000 for 3 different lambdas and will evaluate their distribtion by comparing the results to a sample that is generated by a exponential distribution. For this we use the base R function rexp. We have set the default lambda value of our inverse method to 1 as this is also the default value of the R function rexp.

```r
set.seed(11835945)
sample.rexp <- rexp(1000)
set.seed(11835945)
sample.inv.exp <- random.sample.exp(1000)
```

We created the sample for n=1000 with $\lambda$=1. Lets look at the qqplots:

```r
qqplot(sample.rexp, sample.inv.exp, main="Q-Q plot for the inverse exp method, lambda=1",
ylab = "Sample from rexp",
xlab = "Sample from inverse exponential with uniform random variable",xlim=c(0,8))
```



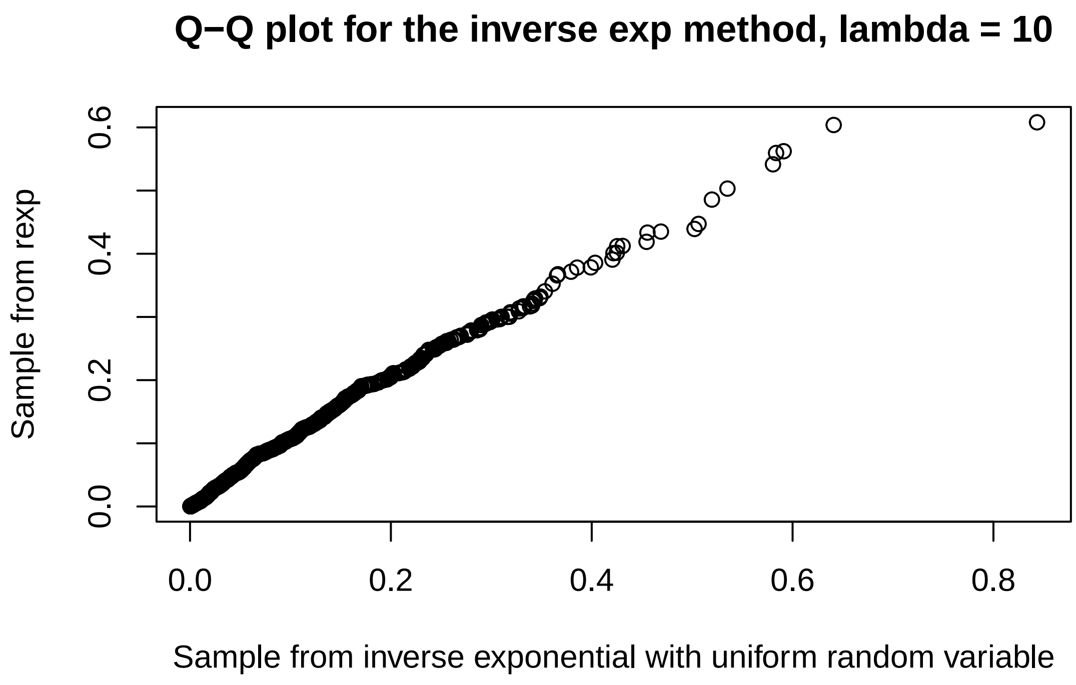
Q-Q (Quantile-Quantile) Plots are used to to compare two probability distributions against each other. In this case, this is done for the 2 samples, one that was created by the exponential distribution, and the other that was created by the inverse exponential with a uniform random variable. As can be seen in the plot, they follow a good linear relationships, which indicates that our created sample with our inverse method is exponentially distributed. Due to variance the outer data points may be not perfectly linear. However, this would correct the more data points are used.

Now lets look at a bigger and smaller lambda, I choose to multiply them by factors 10 and 1 over 10.

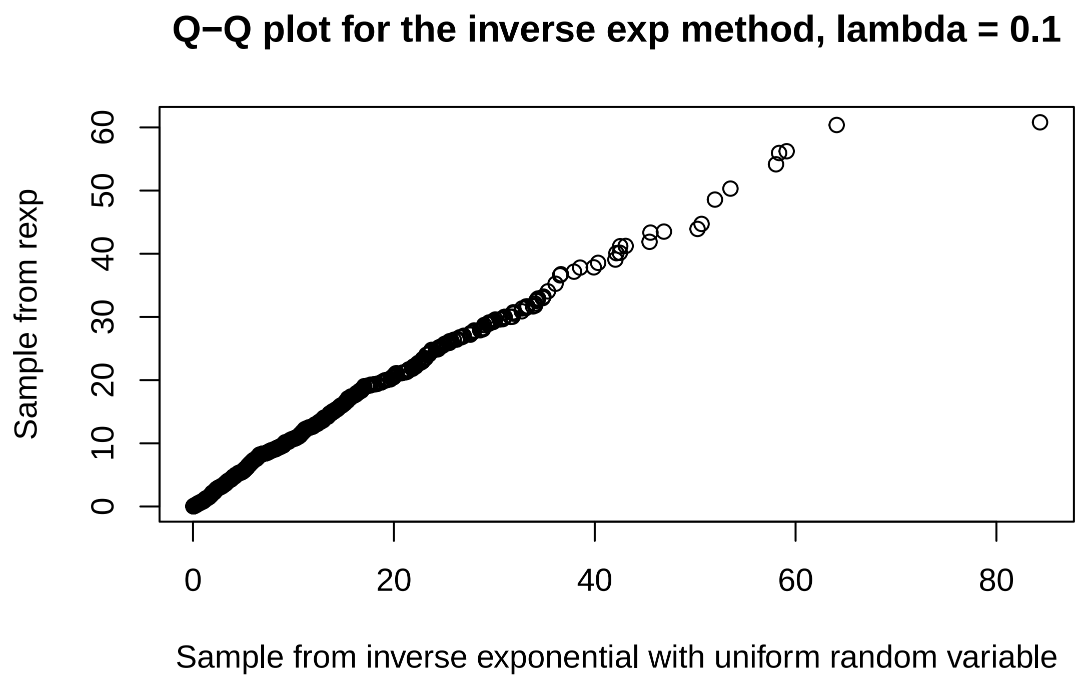
```
set.seed(11835945)
sample.rexp_10 <- rexp(1000,10)
set.seed(11835945)
sample.inv.exp_10 <- random.sample.exp(1000,10)

set.seed(11835945)
sample.rexp_0.1 <- rexp(1000,.1)
set.seed(11835945)
sample.inv.exp_0.1 <- random.sample.exp(1000,.1)
```

```
qqplot(sample.rexp_10, sample.inv.exp_10, main="Q-Q plot for the inverse exp method, lambda = 10",
ylab = "Sample from rexp",
xlab = "Sample from inverse exponential with uniform random variable")
```



```
qqplot(sample.rexp_0.1, sample.inv.exp_0.1, main="Q-Q plot for the inverse exp method, lambda = 0.1",
ylab = "Sample from rexp",
xlab = "Sample from inverse exponential with uniform random variable")
```



Since we used the same seed, the Linearity of the Q-Q plot is the same with the plot of $\lambda = 1$. However, due to the definition of the exponential function, our value range is scaled by the inverse of $\lambda$. Therefore, the data sets are indirectly proportionally scaled by $\lambda$.

## 3 Beta distribution

The Beta distribution has the following pdf:

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

## Write a function which uses an acceptance-rejection approach to sample from a beta distribution. Argue what is a natural candidate for a proposal distribution. A natural candidate for a proposal density g should have a similiar shape to the beta distribution. Furthermore it must fullfil $f(x) \leq cg(x), c > 1$. Since the beta distribution is within the interval $[0, 1]$, the uniform distribution would be suitable. Additionally, if one sets $\alpha$ and $\beta$ to 1, the beta distribtuion is exactly the uniform distribution due to the properties of the $\Gamma$ function.For other parameters the uniform dustribution serves as a good baseline.

We will implement the acceptance-rejection approach to sample from a beta distrubution and furthermore compare a sample of our method to a sample of the base r function rbeta. I choose $\alpha = 2, \beta = 2$ as defaut values, since this will be necessary for a later task.

```
library(assert)
find.best.constant<- function(alpha,beta){
  assert(alpha>1)
  assert(beta>1)
  x.max <- (alpha - 1) / (alpha + beta - 2)
  value<-dbeta(x.max,alpha,beta)
```

```r
  return(value)
}

beta.acc_rej <- function(n, alpha=2, beta=2) {
accepted <- 0
x <- numeric(n)
while(accepted < n) {
u <- runif(1)
y <- runif(1)
CC <- find.best.constant(alpha,beta)
if (dbeta(y, alpha, beta) / (CC * dunif(y)) >= u) {
accepted <- accepted + 1
x[accepted] <- y
}
}
return(x)
}
```

### 3.1 How to choose the constant of the acceptance-rejection method

First, I want to explain how I chose the best constant for the acceptance - rejection approach for the gamma distribution. For arbitrary $\alpha > 1, \beta > 1$, we need to find the argument $x_{max} \in [0, 1]$, in which the Beta distribution achieves its maximum value. By derivation of the beta density function we get that this point is reached at

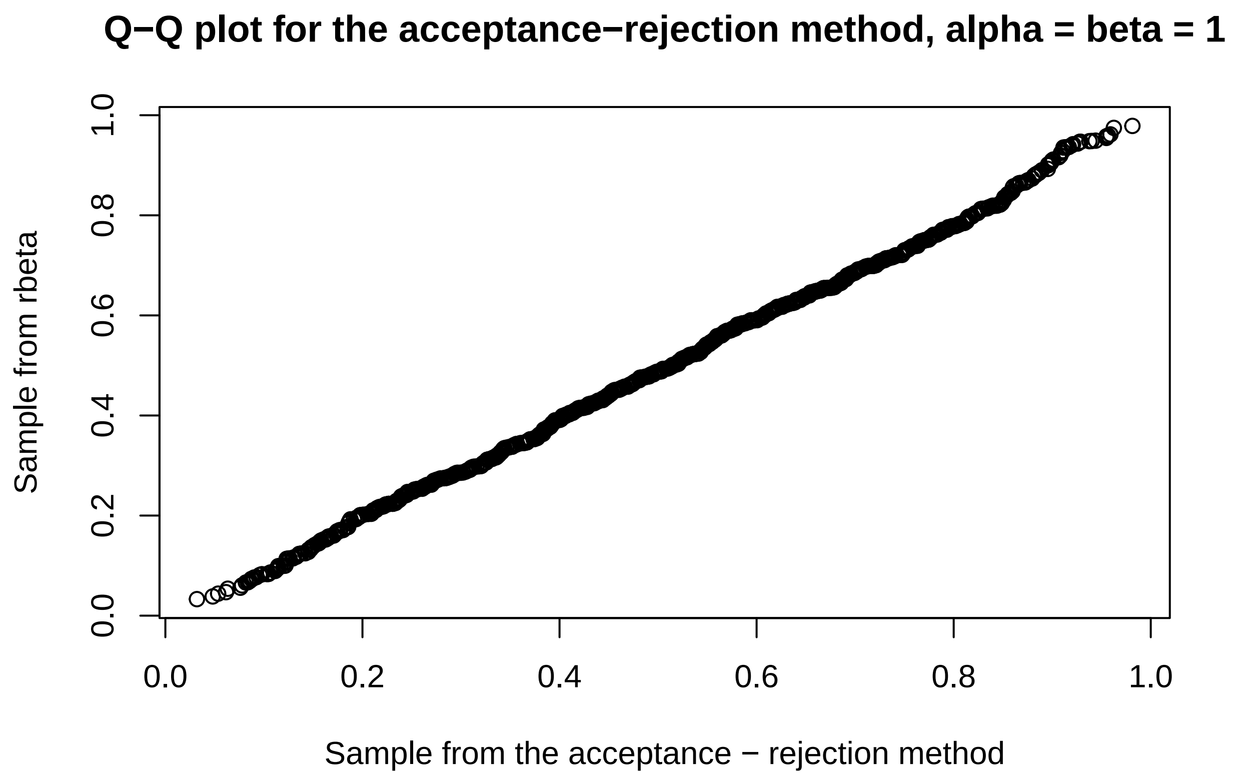
$$x_{max} = \frac{(\alpha - 1)}{(\alpha + \beta - 2)}, \text{for } \alpha > 1, \beta > 1.$$

Then we simply calculate the value of the beta function at this point. we Then have the best possible constant for our method for our set of constants. For the case where both parameters are 2, $x_{max} = 0.5$ and $B(x_{max}) =$1.5. This is what happens in my function find.best.constant. With this value, we keep the rejection proportion as small as possible.

### 3.2 Sampling quality of the acceptance-rejection approach

To check the distribution of my sampling method compared to R's, I will look at the Q-Q plots of 2 samples - one created by my method, one created by the rbeta function.

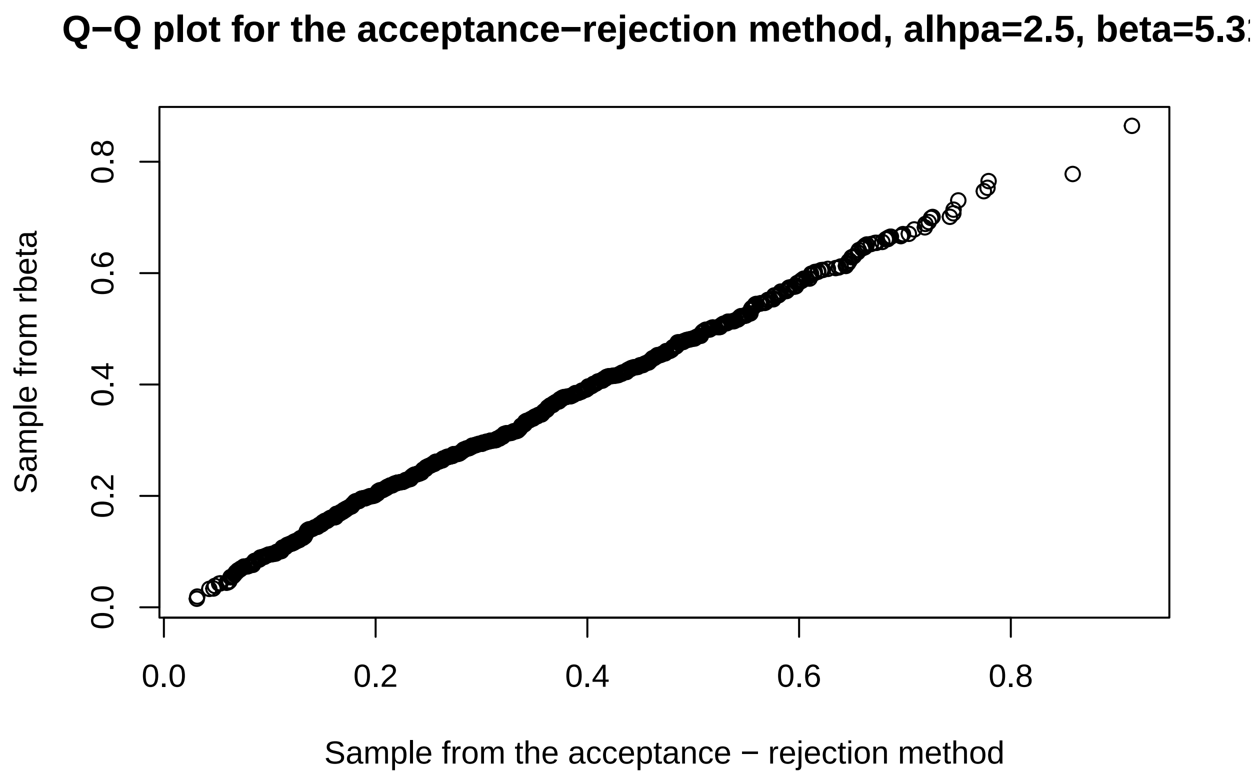
```r
set.seed(11835945)
sample.acc<-beta.acc_rej(1000,2,2)
set.seed(11835945)
sample.base<- rbeta(1000,2,2)
unif<- runif(1000)
qqplot(sample.base, sample.acc, main="Q-Q plot for the acceptance-rejection method, alpha = beta = 1",
ylab = "Sample from rbeta",
xlab = "Sample from the acceptance - rejection method")
```



We can see a good linear relation between the samples. Therefore, the distributions of the samples conicide and therefore our method generates a beta distributed sample.

When choosing other parameters, we observe the same effect:

```
set.seed(11835945)
sample.acc2<-beta.acc_rej(1000,2.5,5.31)
set.seed(11835945)
sample.base2<- rbeta(1000,2.5,5.31)
unif<- runif(1000)
qqplot(sample.base2, sample.acc2, main="Q-Q plot for the acceptance-rejection method, alhpa=2.5, beta=5
ylab = "Sample from rbeta",
xlab = "Sample from the acceptance - rejection method")
```



We observe the same relationship, however, for these parameters, fewer Values above ~0.775 occur.

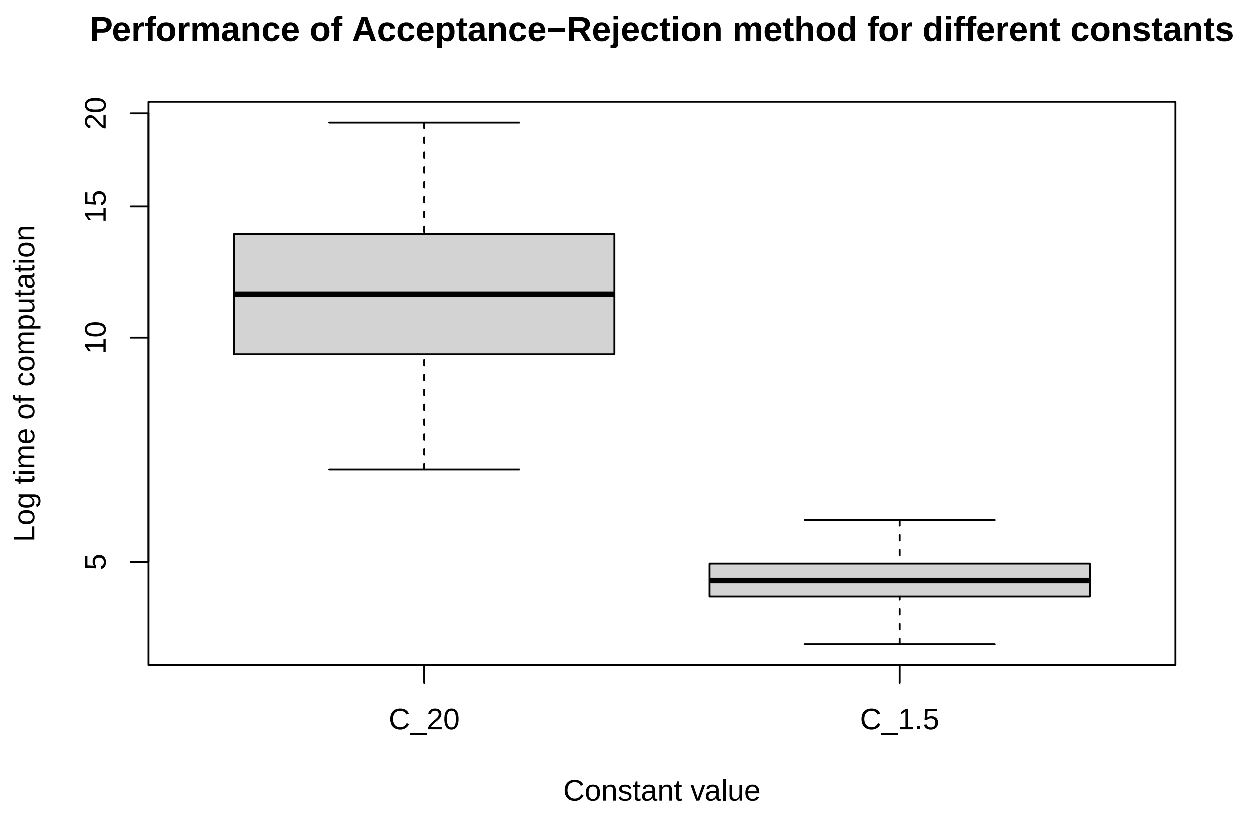
## 3.3 Performance of different constants

At last, I want to check the performance of the best constant versus a very high constant.

```
beta.acc_rej_bad <- function(n, alpha=2, beta=2) {
accepted <- 0
x <- numeric(n)
while(accepted < n) {
u <- runif(1)
y <- runif(1)
CC <- 20
if (dbeta(y, alpha, beta) / (CC * dunif(y)) >= u) {
accepted <- accepted + 1
x[accepted] <- y
}
}
return(x)
}
library(microbenchmark)

benchmark <- microbenchmark(
  C_20 = beta.acc_rej_bad(100,2,2),
  C_1.5 = beta.acc_rej(100,2,2))
```

```r
boxplot(benchmark, main="Performance of Acceptance-Rejection method for different constants",
        xlab="Constant value",ylab="Log time of computation",outline=FALSE)
```



We can see, there is a very significant difference in computation time when choosing bad constants for the acceptance-rejection method. In this example, parameters were set to $\beta = \alpha = 2$.