Exercice n°2

Random Number Generation through CDF and acceptance-rejection sampling

Grégoire de Lambertye

2022-10-12

1 Pseudo-random number generation with Linear Congruential Random Number Generation Algorithm

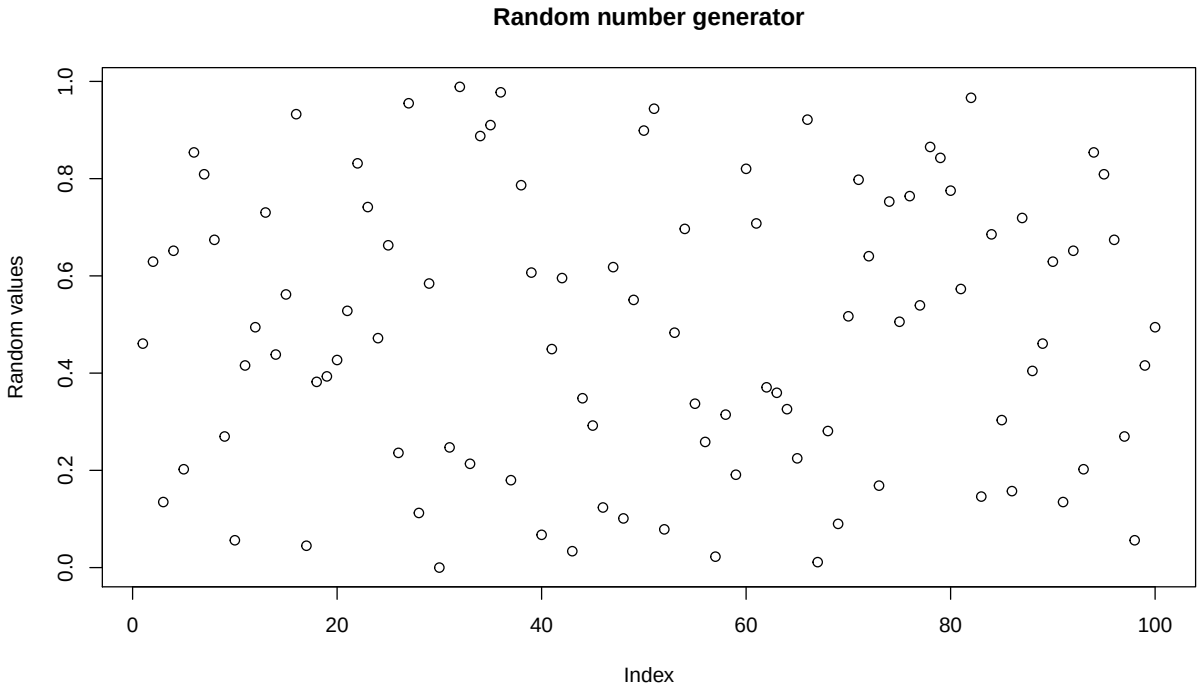
Pseudo-random number generators (PRNG) are algorithms used in computer science to simulate random number generation. They are called pseudo because they use recursive process to generate numbers and they are initialized with a seed. They are by this way reproducible and deterministic but they look random.

The main idea of the linear congruential random number generator is to define a sequence based on the linear formula $x_n + 1 = a * x_n + c \mod m$

Here is a linear congruential number generator :

That we can observe here

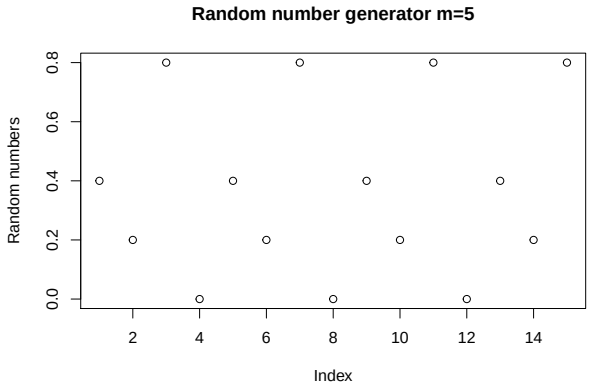
```
plot(congruential_gen(100,178,3,44,72), main="Random number generator", ylab="Random values")
```

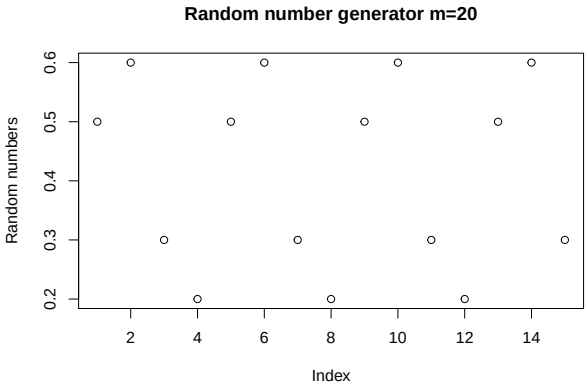


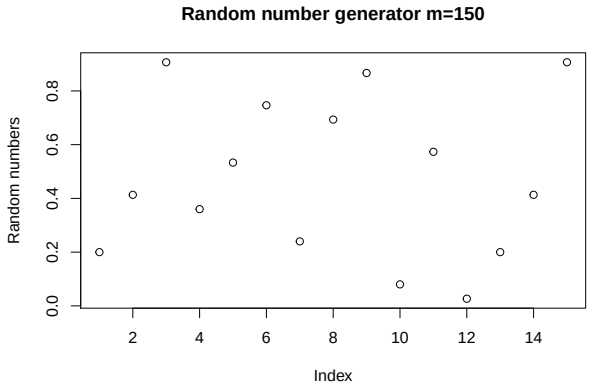
The modulus number m as a huge influence on the sequence and act as a “maximal number generated”

befor the mapping between 0 and 1. It also a major parameter for the sequence length. With a small m, it's easy to recognize the sequence quickly:

```
plot(congruential_gen(15,5,7,2,0), main="Random number generator m=5", ylab="Random numbers")
plot(congruential_gen(15,20,7,2,4), main="Random number generator m=20", ylab="Random numbers")
plot(congruential_gen(15,150,7,2,4), main="Random number generator m=150", ylab="Random numbers")
```







It seems pretty easy to understand the sequence of the 2 firsts plot. For the m=150 sample we would need more data to be able to recognize the sequence.

2 Exponential distribution

The exponential distribution has the following cdf:

$$F(x) = 1 - exp(-\lambda x), \lambda > 0 \text{ and } x \in [0 : \infty]$$

```
expf <- function(x, lambda){
  return(1-exp(-lambda*x))
}
```

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

To generate uniform random variables it's convenient to use runif(). This function provided by R can generate random numbers that follow a uniform distribution.

```r
set.seed(12202211)
sample <- runif(100,0,1)
```

A way to obtain a random sample from the exponential distribution is to use the *quantile function* $Q_x(p)$ defined as the smallest x, such that $F_X(x) = u$ whith $u \in [0:1]$

$$Q_X(u) = min\{x \in R | F_X(x) = u\} = F_X^{-1}(x)$$

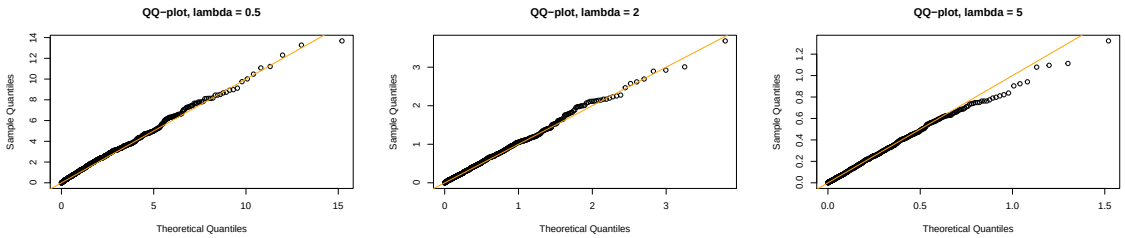
We can compute $X \sim Exp(\lambda)$ with this workaround:

$$u = 1 - exp(-\lambda x)exp(-\lambda x) = 1 - u - \lambda x = ln(1-u)x = 1 - \frac{ln(1-u)}{\lambda}$$

```r
library("SciViews")
set.seed(12202211)
exp_distrib <- function(lambda,n){
  return(-1 / lambda * ln(1 - runif(n, 0, 1)))
}

lambda_list <- c(0.5,2,5)

for(l in lambda_list){
  qqplot(qexp(ppoints(1000), rate=l), exp_distrib(l,1000) , xlab="Theoretical Quantiles", ylab= "Sample
  abline(a=0, b=1, col='orange')
}
```



The theorical quantile and the computed onces arewell aligned, we can conlude that our samples follow an exponential distribution.

3 Acceptance-Rejection approach

We will try to use the Acceptance-Rejection method to approach a beta distribution. The Beta distribution has the following pdf:

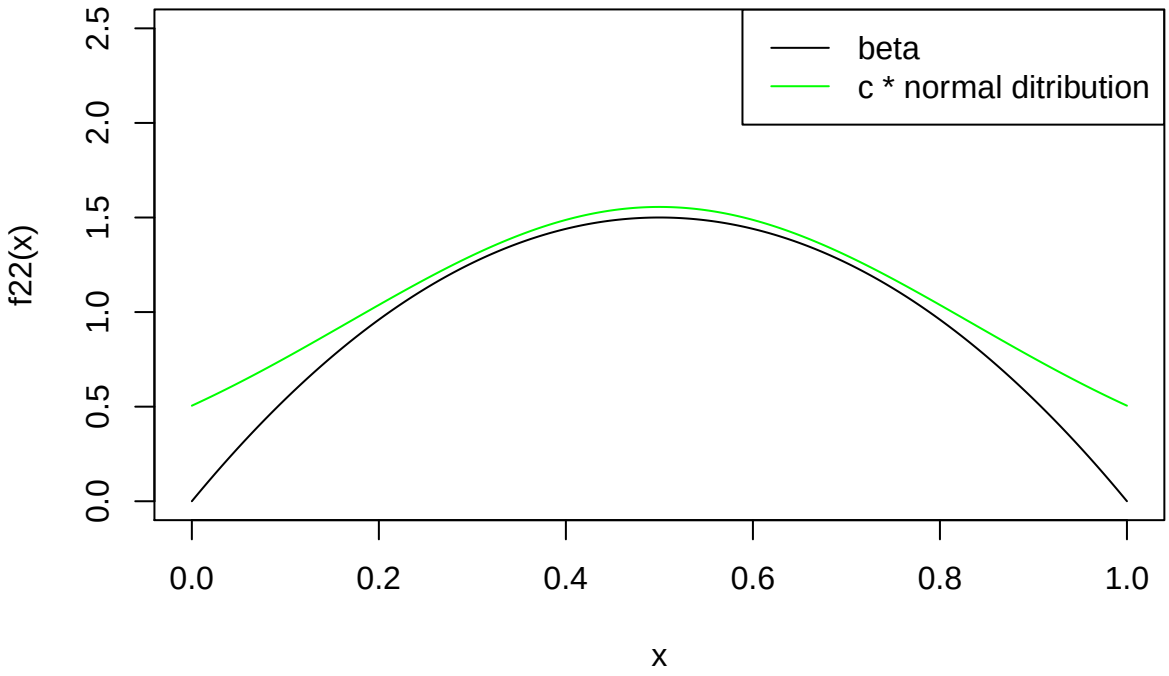
$$f(x;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

Γ is the gamma function define as :

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$$

We will focus on the case where $a = 2$ and $b = 2$ i.e.

$$f(x;2,2) = \frac{\Gamma(4)}{\Gamma(2)\Gamma(2)} x(1-x) = 6 * x(1-x)$$



We print in black the beta (2,2) distribution. After trying different values for c and the standard deviation, the candidate c=1.3 and $X \sim \mathcal{N}(0.5, 1/3)$ seems to fit the curve well (in green).

```
beta22_approach <- function(n, alpha=2, beta=2){
  iter <- 0; accepted <- 0
  x <- numeric(n)
  CC <- 1.3
  while(accepted < n) {
    iter <- iter + 1
    u <- runif(1)
    y <- runif(1)
    if (u <= dbeta(y, alpha, beta) / (CC * dnorm(y,0.5,1/3))) {
      accepted <- accepted+ 1
      x[accepted] <- y
      }
    }
  print(paste('Acceptance proportion: ', round(n / iter * 100,4), '%'))
  return(x)
}

invisible(beta22_approach(1000))
```

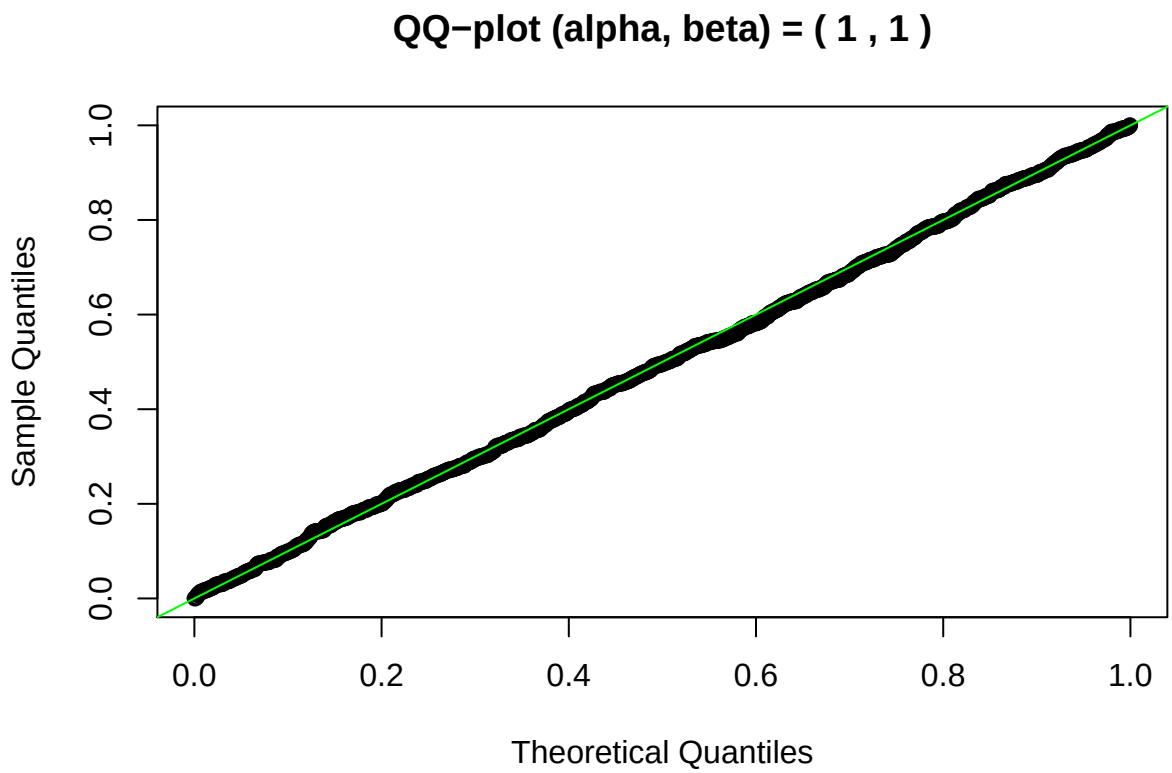
```
## [1] "Acceptance proportion:  82.9187 %"
```

```
beta_accept_rej <- function(n, alpha, beta){
  iter <- 0; accepted <- 0
```

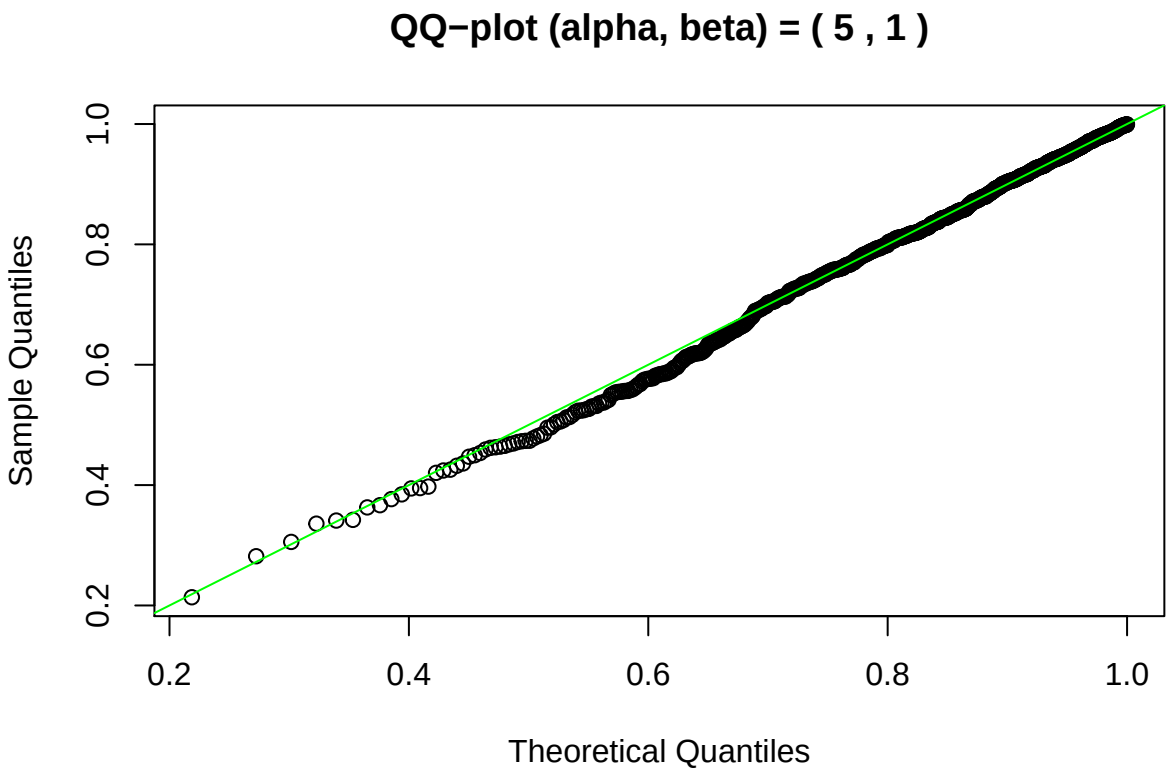
```
  x <- numeric(n)
  CC <- max(alpha, beta)
  while(accepted < n) {
    iter <- iter + 1
    u <- runif(1)
    y <- runif(1)
    if (u <= dbeta(y, alpha, beta) / (CC * dunif(y))) {
      accepted <- accepted+ 1
      x[accepted] <- y
      }
    }
  print(paste('Acceptance proportion: ', round(n / iter * 100,4), '%'))
  return(x)
}
```

We will finally test in for some parameter choices: $(\alpha, \beta) \in \{(1,1),(5,1),(1,5),(10,10)\}$

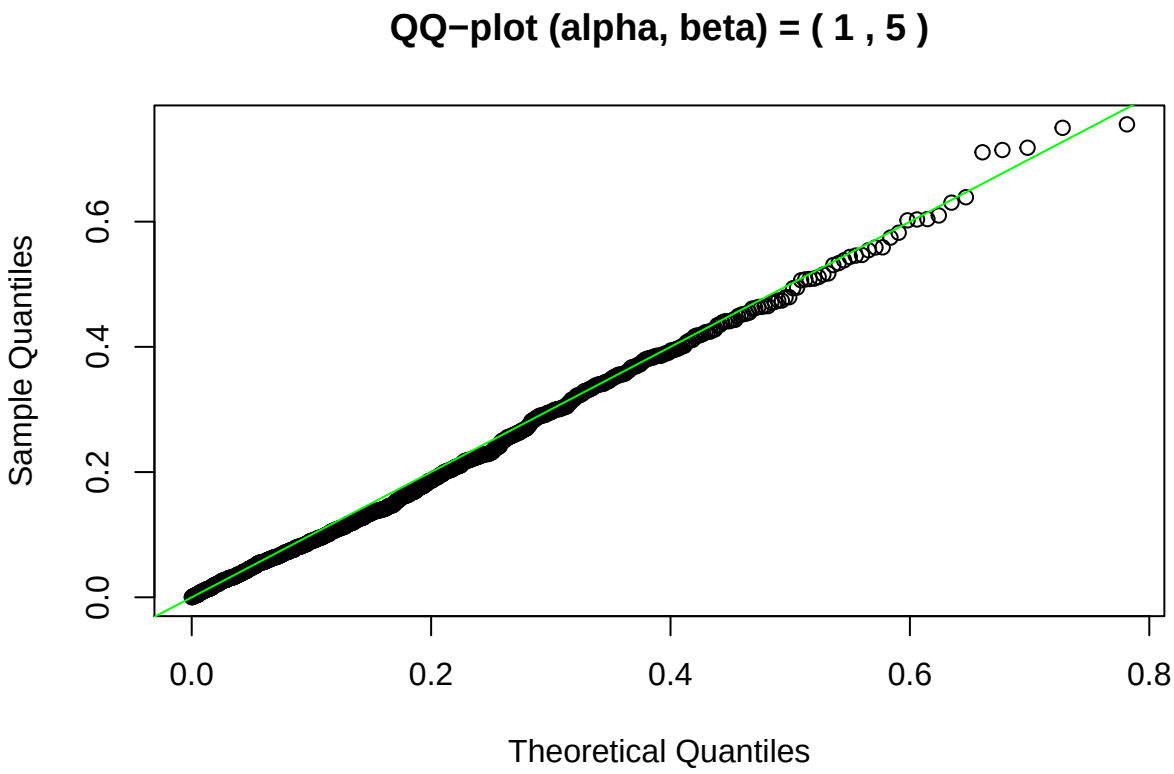
```
## [1] "Alpha =  1 , Beta =  1"
## [1] "Acceptance proportion:  100 %"
```



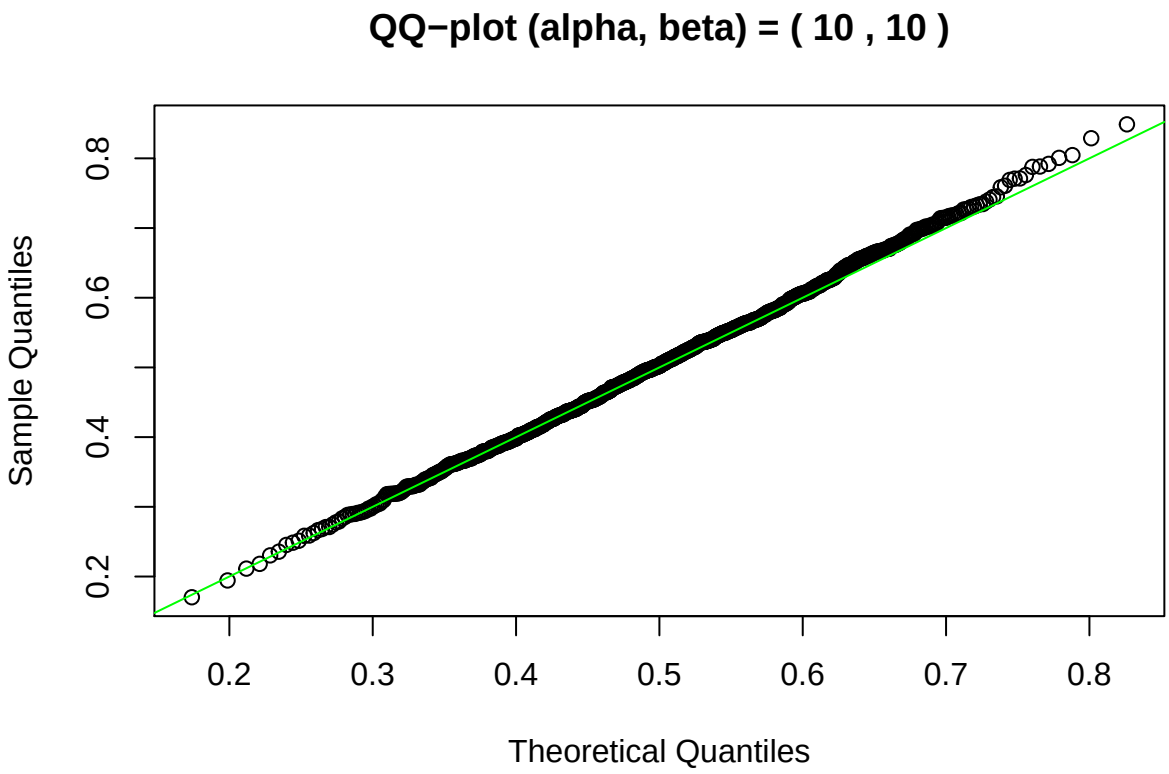

```
## [1] "Alpha =  5 , Beta =  1"
## [1] "Acceptance proportion:  20.2963 %"
```




```
## [1] "Alpha =  1 , Beta =  5"
## [1] "Acceptance proportion:  19.2604 %"
```




```
## [1] "Alpha =  10 , Beta =  10"
## [1] "Acceptance proportion:  10.0867 %"
```



The QQ plots show that the random sample follows a Beta distribution with the selected parameters. However, the acceptance proportions are rather low and decrease as $c = \max(\alpha, \beta)$ grows, which corresponds to the growth of the distribution parameters. The shape of the density of the Beta distribution makes it difficult to choose any other proposal distribution than our choice.