Exercice n°2

Random Number Generation through CDF and acceptance-rejection sampling

Grégoire de Lambertye

2022-10-24

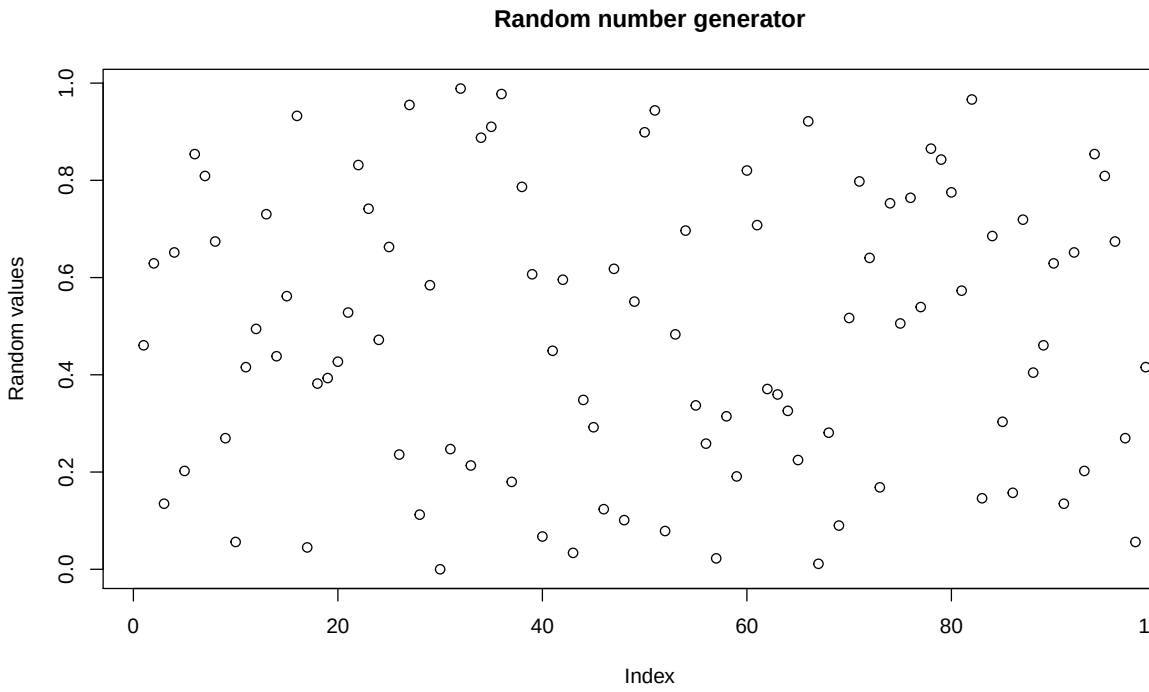
1 Pseudo-random number generation with Linear Congruential Random Number Generation Algorithm

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 $x_n + 1 = a * x_n + c \mod m$ and $x_0 = seed$.

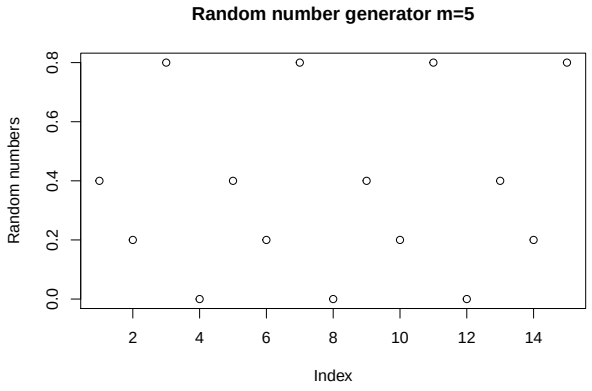
Here is a linear congruential number generator :

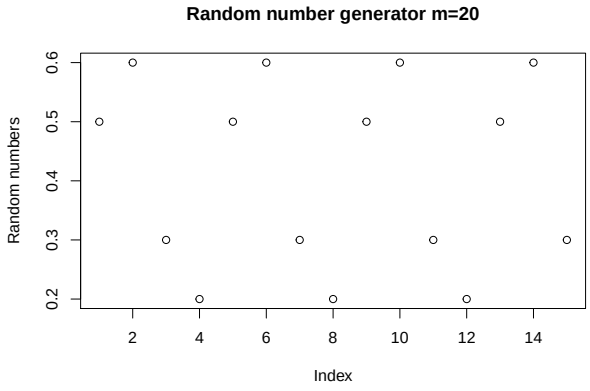
```
congruential_gen <- function(n,m,a,c=0,x0)
{
  us <- numeric(n)
  us[0] <- x0
  for (i in 1:n)
  {
    x0 <- (a*x0+c) %% m
    us[i] <- x0 / m
  }
  return(us)
}
```

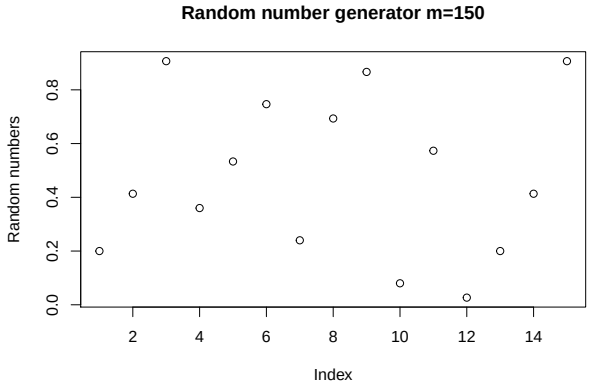


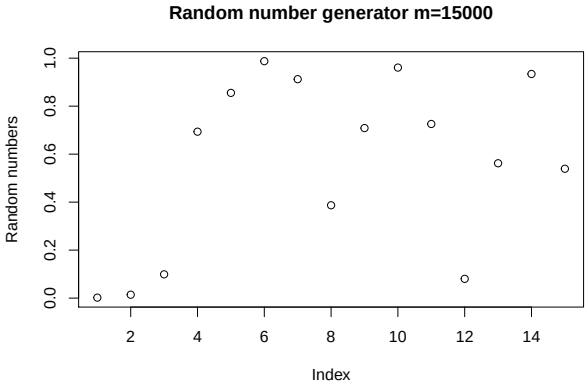
That we can observe here
The modulus number m as a huge influence on the sequence and act as a "maximal number generated" before the mapping between 0 and 1. It also a major parameter for the sequence length. With a small m, it's easy to recognize the sequence quickly:

```
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")
plot(congruential_gen(15,15000,7,2,4), main="Random number generator m=15000", ylab="Random numbers")
```









It seems pretty easy to understand the sequence of the 2 firsts plot. For the third ans fourth sample we would need more data to be able to recognize the sequence. The bigger m is the more data we need.

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.

```
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:

$$u = 1 - exp(-\lambda x)$$

$$exp(-\lambda x) = 1 - u$$

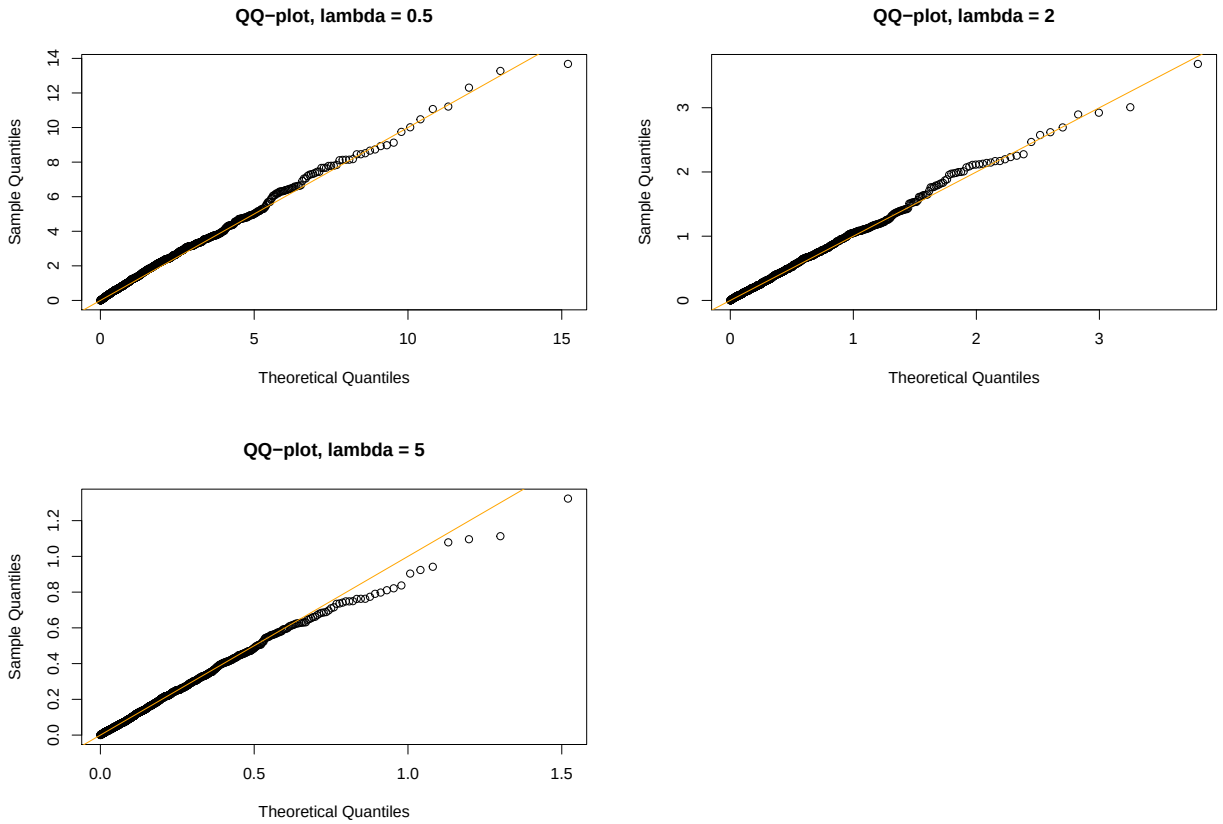
$$-\lambda x = ln(1 - u)$$

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

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

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 once are well aligned, we can conclude that our samples follow an exponential distribution. for $\lambda = 5$ our QQ plot seems a little bit left-skewed but that might come from the seed (other seed don't show this bias).

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:

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

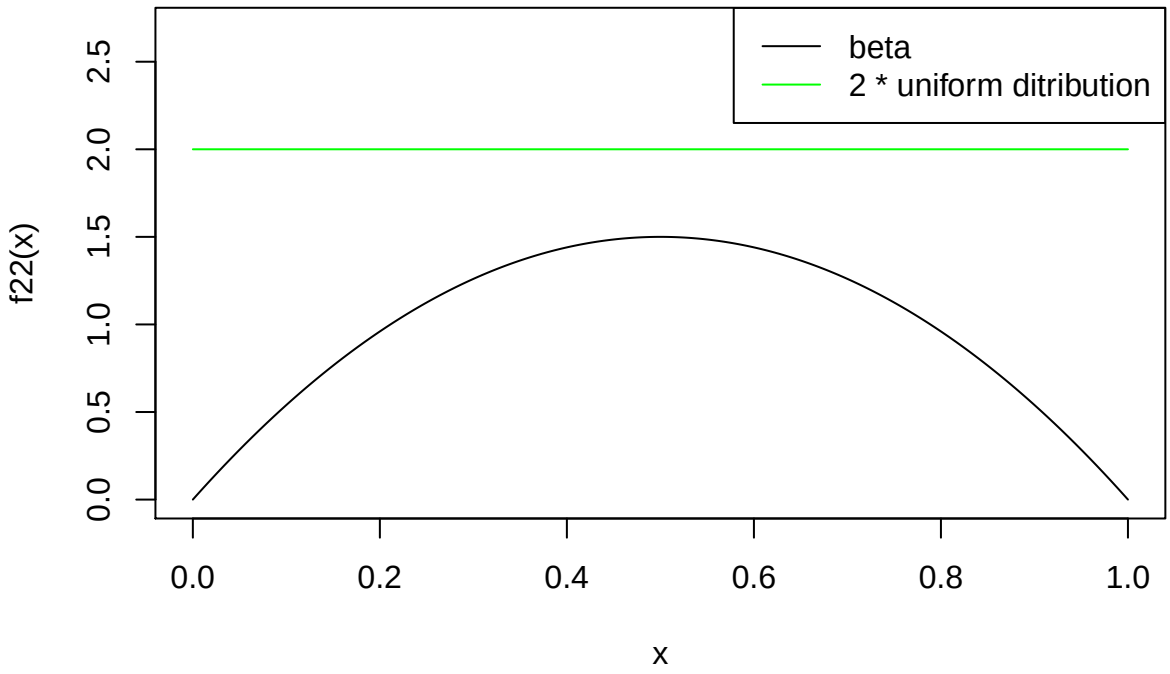
Γ is the gamma function define as :

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

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

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

In this case, a good distribution could be the normal distribution but we can also take the uniform distribution wich will be easier to use and more adaptive to other gamma parameters.



We print in black the beta (2,2) distribution. A good C value would be the maximum of the distribution but it would demand to derive the distribution and recompute the maximal. A easier candidate is the maximum between alpha and beta (here 2). We determined this optimum value for C after trying different value for alpha and beta. With this distribution, we have a proportion of accepted point of 50.4%.

```
## [1] "Acceptance proportion:  50.4032 %"
```

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

```
beta_accept_rej <- function(n, alpha, beta){
  iter <- 0; accepted <- 0
  x <- numeric(n)
  CC <- max(alpha, beta)
  while(accepted < n) {
    iter <- iter + 1
```

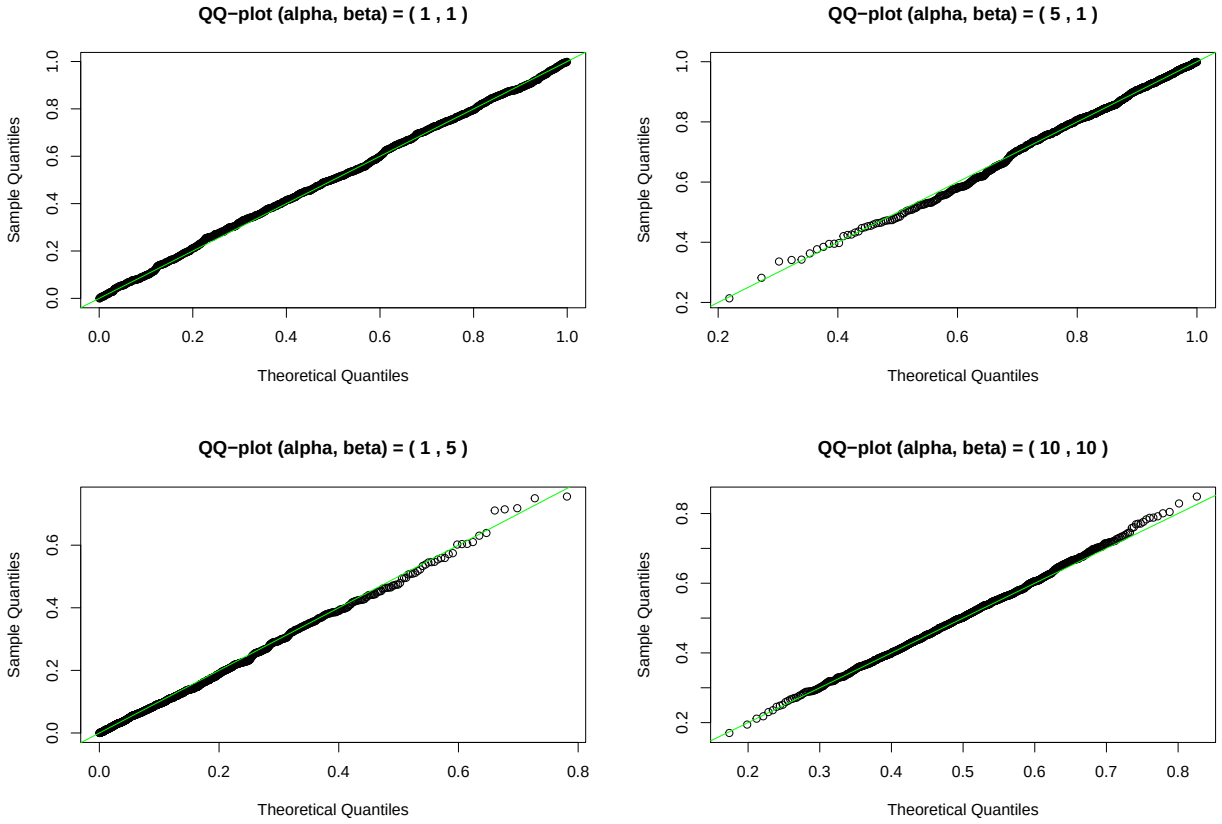
```
    u <- runif(1)
    y <- runif(1)
    if (u <= dbeta(y, alpha, beta) / (CC * dunif(y))) {
      accepted <- accepted+ 1
      x[accepted] <- y
      }
    }
  print(paste('Acceptance proportion: ', round(n / iter * 100,4), '%'))
  return(x)
}
```

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

## [1] "Alpha =  5 , Beta =  1"
## [1] "Acceptance proportion:  20.1898 %"

## [1] "Alpha =  1 , Beta =  5"
## [1] "Acceptance proportion:  19.8216 %"

## [1] "Alpha =  10 , Beta =  10"
## [1] "Acceptance proportion:  10.1368 %"
```



The QQ plots show that the random samples follow Beta distributions with the selected parameters. However, the proportion of accepted point decrease as C grows. Using the maximum of the distribution for the C value couldn't help us in every situation (alpha = 5 & beta = 1 for example).