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

Yannik Gaebel

2022-10-25

```
set.seed(12208157)
```

## Task 1. Linear Congruential Random Number Generation Algorithm

Concept of Linear Congruential Random Number Generation Algorithm:

The Linear Congruential Random Number Generation Algorithm generates a sequence of pseudo random numbers based on the formula:

$x_{n+1} = (a * x_n + c)(mod\, m)$

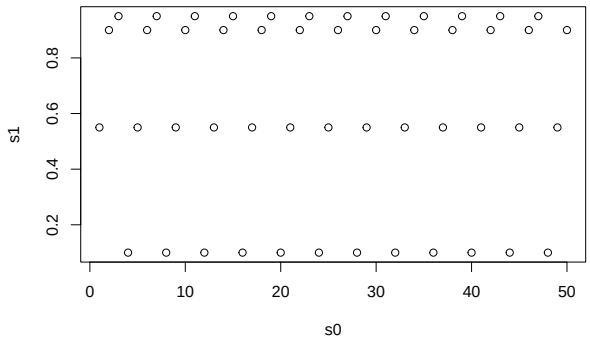
The algorithm starts with a seed $x_0$ and generates the rest of the sequence based on it. It has three parameters that need to be set. The modulus $m$ is supposed to be a large integer, a prime number preferably. A multiplier $a$ and an increment $c$. The algorithm then generates a cyclic sequence, which means that at some point it will repeat itself. If the choice for $a$, $m$ and $c$ are appropriate, this sequence is very long.

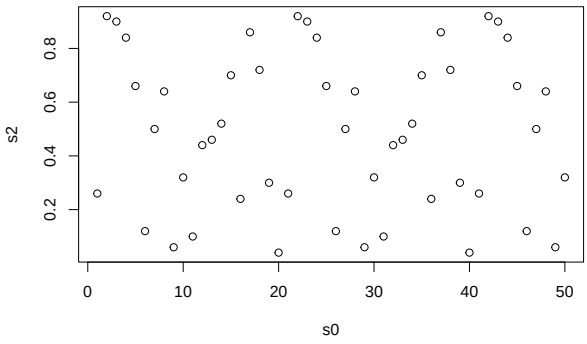
```
# Code from lecture slides

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

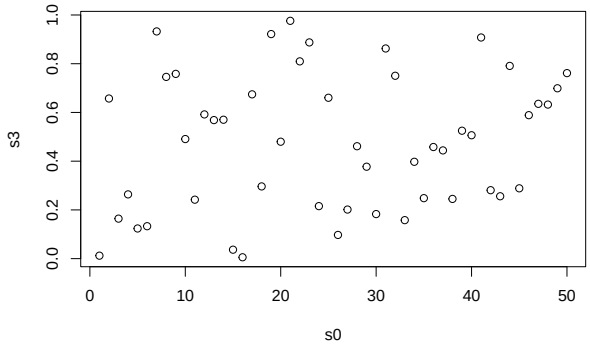
Try out and compare different values of $m$ and $a$:

The losgest possible sequence that can can be generated by the algorithm is $m$, so $m$ should be a large value.

```
n <- 50 # length of random sequence
s0 <- 1:50

s1 = round(mc.gen(n,m=20,a=3,c=5,x0=2),4)
s2 = round(mc.gen(n,m=50,a=3,c=7,x0=2),4)
s3 = round(mc.gen(n,m=123456,a=300,c=5,x0=5),4)

plot(s0,s1)
plot(s0,s2)
plot(s0,s3)
```



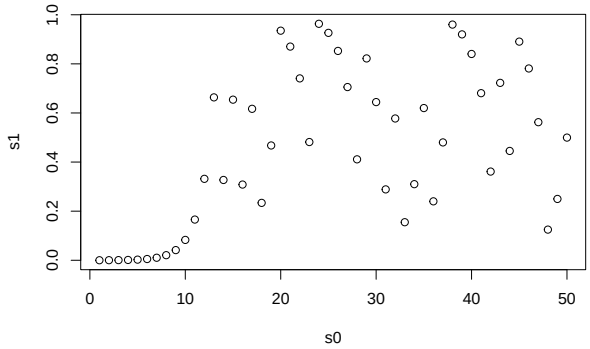




We can see that the bigger $m$ is the longer the cycles get. In the first to plots it is easy to spot the repeating cycle. In the last plot no cycle is visible.

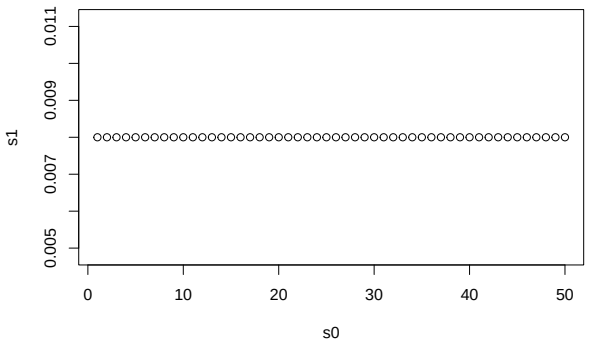
$a$ should not be to small, as you can see in the following plot:

```
n <- 50 # length of random sequence
s0 <- 1:50
s1 = round(mc.gen(n,m=123456,a=2,c=5,x0=5),4)
plot(s0,s1)
```




$a$ should not be a divisor of $m$:

```r
n <- 50 # length of random sequence
s0 <- 1:50
s1 = round(mc.gen(n,m=390625,a=625,c=5,x0=5),4)
plot(s0,s1)
```




## Task 2.

The cdf for the exponential distribution is given as:

$$F_X(x) = 1 - exp(-\lambda * x),\ \lambda > 0$$

To generate a random sample from the exponential distribution using the uniform distribution, the quantile function needs to be computed. To do this we need to invert the cdf of the exponential distribution. The mathematical formula then becomes:

$$F_X(x)^{-1} = \frac{-ln(1-x)}{\lambda},\ 0 \leq x \leq 1$$

To generate a random number given $u = unif[0,1]$ we need to calculate:

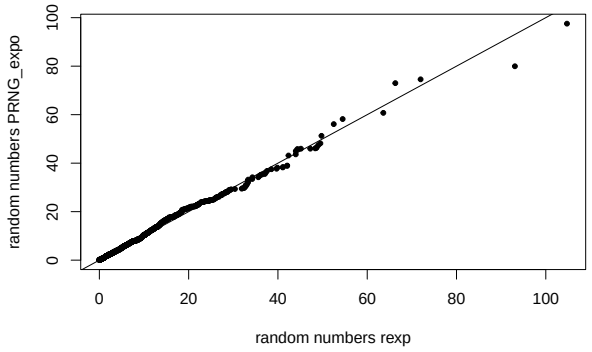
$$F_X(u)^{-1} = \frac{-ln(1-u)}{\lambda}$$

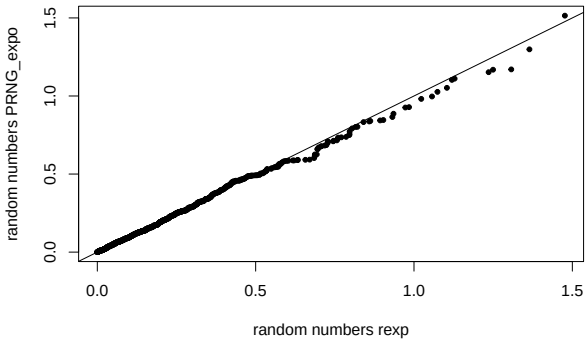
Function to generate a random sequence from the exponential distribution:

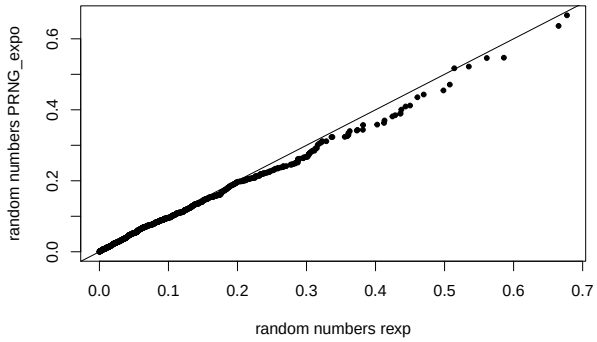
```r
PRNG_expo <- function(n,y){
  result <- numeric(n)
  for (i in 1:n){
    u = runif(1)
    result[i] <- (-1*log(1-u))/y
  }
  return (result)
}
```

Plot QQ-Plots:

```r
n <- 1000
y <- 0.1

RN_my_function <- PRNG_expo(n,y)
RN_r_function <- rexp(n,y)

qqplot(RN_r_function, RN_my_function, pch = 20, xlab = "random numbers rexp", ylab = "random numbers PRI
abline(0,1)

y <- 5

RN_my_function <- PRNG_expo(n,y)
RN_r_function <- rexp(n,y)

qqplot(RN_r_function, RN_my_function, pch = 20, xlab = "random numbers rexp", ylab = "random numbers PRI
abline(0,1)

y <- 10

RN_my_function <- PRNG_expo(n,y)
RN_r_function <- rexp(n,y)

qqplot(RN_r_function, RN_my_function, pch = 20, xlab = "random numbers rexp", ylab = "random numbers PRI
abline(0,1)
```




The method works well. When the values get very high they start to deviate more.

## Task 3. Acceptance-rejection approach with beta distribution

For the proposal distribution we need a function that larger than the target function when multiplied by a constant C. For a good acceptance rate, it should be only slightly larger that the target function. The greater the difference between the functions the lower the acceptance rate is going to be. As a simple proposal distribution we could take the uniform distribution.

Function that uses acceptance-rejection approach to sample from beta distribution:

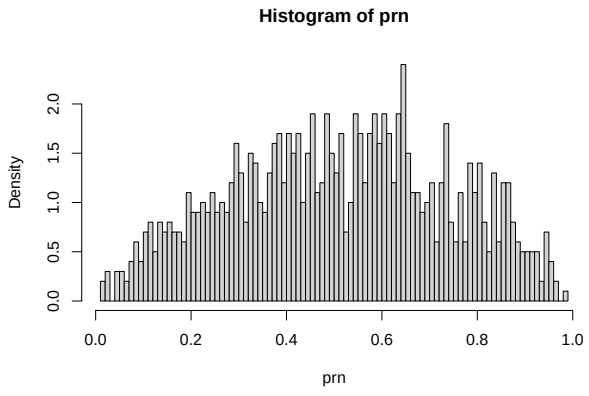
```
accept_reject <- function(n,alpha,beta,C){
  iter <- 0
  accepted <- 0
  result <- numeric(n)

  while(accepted<n){
    u <- runif(1)
    iter <- iter + 1
    y <- runif(1)
    if (dbeta(y,alpha,beta)/(C*dunif(y)) >= u){
      accepted <- accepted+1
      result[accepted] <- y
    }
  }

  print(paste("Rejection proportion: ", round(1-(n/iter),2)))
  return (result)
}
```

The maximum when alpha and beta are 2 is 1.5. So we will take 1.5 as the c value at fist:

```
prn <- accept_reject(1000,2,2,1.5)
```

```
## [1] "Rejection proportion:  0.34"
```

```
hist(prn,breaks=100,probability = TRUE)
```

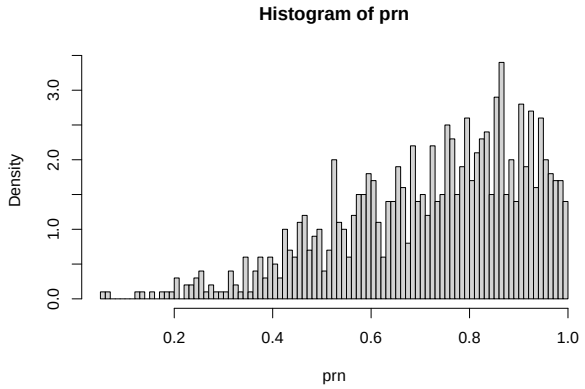



Try other values for alpha and beta:

```
prn <- accept_reject(1000,3,1,2)
```

```
## [1] "Rejection proportion:  0.56"
```

```
hist(prn,breaks=100,probability = TRUE)
```




```
prn <- accept_reject(1000,5,5,4)
```

```
## [1] "Rejection proportion:  0.75"
```

```
hist(prn,breaks=100,probability = TRUE)
```