# Generating random number
# (A Brief Note)

Alireza Sheikh-Zadeh, PhD

One of the most important advantages of using simulation models to characterize system performance is the flexibility that simulation allows for incorporating randomness into the modeling.

## Pseudorandom Numbers

A sequence of pseudorandom numbers is a sequence of numbers between 0 and 1 having the same relevant statistical properties as a sequence of truly uniformly random numbers.

The algorithms that produce pseudorandom numbers are called random number generators.

## Random Number Generators

Over the history of scientific computing, there have been a wide variety of techniques and algorithms proposed and used for generating pseudorandom numbers. A common technique that has been used (and is still in use) within a number of simulation environments is called the linear congruential generator (LCG).

## LCG Algorithm

LCG is a recursive algorithm for producing a sequence of pseudorandom numbers. Each new pseudorandom number from the algorithm depends on the previous pseudorandom number. Thus, a starting value called the seed is required. Given the value of the seed, the rest of the sequence of pseudorandom numbers can be completely determined by the algorithm.

An LCG defines a sequence of integers, $R_0$, $R_1$, … between 0 and m-1 according to the following recursive relationship, where i = 0, 1, 2, …:

$$R_{i+1} = (aR_i + c) \bmod m$$

where $R_0$ is called the seed of the sequence, a is called the constant multiplier, c is called the increment, and m is called the modulus and mod finds the remainder after division of one number by another. (m, a, c, $R_0$) are integers with $a > 0$, $c \geq 0$, $m > a$, $m > c$, $m > R_0$, and $0 \leq R_i \leq m - 1$.

To compute the corresponding pseudorandom uniform number, we use

$$U_i = \frac{R_i}{m}$$

Notice that an LCG defines a sequence of integers and subsequently a sequence of real (rational) numbers that can be considered pseudorandom numbers.

**Example: Simple LCG**

Consider an LCG with parameters (m = 8, a = 5, c = 1, R0 = 5). Compute the first nine values for Ri and Ui from the defined sequence.

R_1 = (5R0 + 1) mod 8 = 26 mod 8 = 2 then U1 = 0.25

```
a= 5
c = 1
m=8
R0 = 5
R1 = (a*R0 + c) %% m
(U1 = R1/m)

## [1] 0.25
```

R2 = (5R1 + 1) mod 8 = 11 mod 8 = 3 then U2 = 0.375

R3 = (5R2 + 1) mod 8 = 16 mod 8 = 0 then U3 = 0.0

R4 = (5R3 + 1) mod 8 = 1 mod 8 = 1 then U4 = 0.125

R5 = 6 then U5 = 0.75

R6 = 7 then U6 = 0.875

R7 = 4 then U7 = 0.5

R8 = 5 then U8 = 0.625

R9 = 2 then U9 = 0.25

```
# The following function is an implementation of a linear congruential
generator with the given parameters above.
lcg.rand <- function(n) {

  U <- vector(length = n)

  m <- 8
  a <- 5
  c <- 1

  # Set the seed (R0)
```

```
  R = 5

  for (i in 1:n) {
    R <- (a * R + c) %% m
    U[i] <- R / m
  }

  return(U)
}

# Print 9 random numbers on the interval [0, 1)

lcg.rand(9)

## [1] 0.250 0.375 0.000 0.125 0.750 0.875 0.500 0.625 0.250
```

Note: Larger M is better. If m is small, there will be gaps on the interval [0, 1) and if m is large, then the $U_i$ will be more densely distributed on [0, 1).

Notice that if a sequence generates the same value as a previously generated value, then the sequence will repeat or cycle. An important property of an LCG is that it has a long cycle, as close to length m as possible. The length of the cycle is called the period of the LCG.

Ideally, the period of the LCG is equal to m. If this occurs, the LCG is said to achieve its full period. As can be seen in the example, the LCG is full period. Until recently, most computers were 32-bit machines and thus a common value for $m$ is $2^{31} - 1 = 2,147,483,647$, which represents the largest integer number on a 32-bit computer using 2's complement integer arithmetic. This choice of m also happens to be a **prime number**, which leads to special properties.

## LCG Full Period Conditions

An LCG has a full period if and only if the following three conditions hold:

1. The only positive integer that (exactly) divides both m and c is 1 (i.e., c and m have no common factors other than 1).
2. If q is a prime number that divides m then q should divide (a - 1) (i.e., (a - 1) is a multiple of every prime number that divides m).
3. If 4 divides m, then 4 should divide (a - 1) (i.e., (a - 1) is a multiple of 4 if m is a multiple of 4).

**Practice:** Use the above conditions to check if the LCG of (m=8, a = 5, and c = 1) should reach its full period.

## Testing Random Numbers

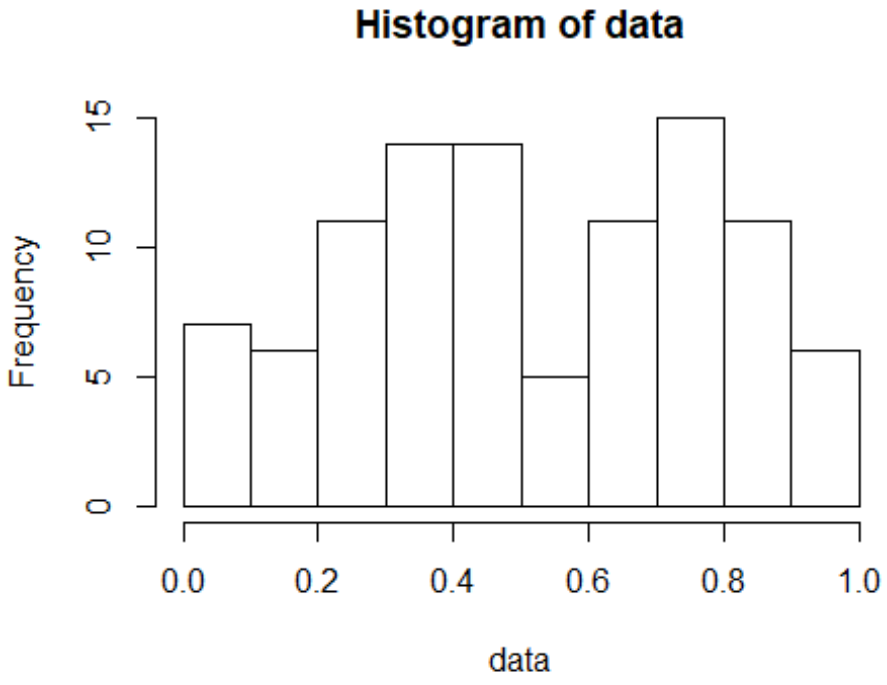
Essentially a random number generator is supposed to produce sequences of numbers that appear to be independent and identically distributed (IID) Uniform(0, 1) random variables. The hypothesis that a sample from the generator is IID U(0, 1) must be made. Then, the hypothesis is subjected to various statistical tests.

## Chi-Squared Goodness-of-Fit Test

The chi-square test divides the range of the data into, k, intervals, and tests if the number of observations that fall in each interval is close to the expected number that should fall in the interval, given the hypothesized distribution is the correct model.

**Example:** Suppose we have 100 observations from a pseudorandom number generator. Perform a chi-squared test that the numbers appear Uniform(0, 1).

```
set.seed(1)
nsim = 100
data <- runif(nsim)
# H0: Data is uniformly distributed.

b = seq(0,1, by = 0.1)
h = hist(data, b, right = FALSE)
```




```
h$counts
```

```
##  [1]  7  6 11 14 14  5 11 15 11  6
```

```
chisq.test(h$counts)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  h$counts
## X-squared = 12.6, df = 9, p-value = 0.1816
```

```
# Test Statistics Sum((observed - expected)^2/expected )

o = h$counts
e = 10 # We expected 10 random number in each interval
chi.stat = sum((o-e)^2/e)
chi.stat
```

```
## [1] 12.6
```

```
# degree of freedom : k-s-1 = 10-0-1 = 9 # k is the number of intervals, s is
the number of parametrs.
# In the uniform distribution s = 0. In Poisson, s = 1. In Normal, s = 1.
pvalue = pchisq(chi.stat, 9, lower.tail = F)
pvalue
```

```
## [1] 0.1815566
```

```
# Fail to reject H0.
```

There are also some other useful tests, which we do not cover in this class. For instance:

- Kolmogorov–Smirnov (K-S), which has the same null hypothesis as the Chi-sq test. test
- Correlation tests and runs tests for testing the Independence.
- Poker test and gap test for testing the existence of Patterns.

## Generating Random Variates From Distributions

In the simulation, pseudorandom numbers serve as the foundation for generating samples from probability distribution models. We will now assume that the random number generator produces sequences of uniform numbers ($U_i$ between 0 and 1). We now want to take the $U_i$ and utilize them to generate from probability distributions.

The realized value from a probability distribution is called a random variate. Simulations use many different probability distributions as inputs. Thus, methods for generating random variates from distributions are required, that is, how do you generate samples from probability distributions? There are four basic methods for producing random variates:

1. Inverse transform or inverse CDF method
2. Convolution
3. Acceptance/rejection
4. Mixture and truncated distributions.

In this course, we only cover the inverse transform method.

## Inverse Transform

The inverse transform method is the preferred method for generating random variates, provided that the inverse transform of the CDF (cumulative density function) can be easily derived or computed numerically.

The key advantage for the inverse transform method is that for every $U_i$ generated, a corresponding $X_i$ will be produced.

The example below illustrates the inverse transform method for the exponential distribution.

**Example: Inverse CDF Method for Exponential Distribution**

The exponential distribution is often used to model the time until an event (e.g., time until failure and time until an arrival) and has the form:

$f(x) = \lambda e^{-\lambda x}$ if $x \geq 0$

Derive the inverse CDF and present an inverse CDF algorithm for the exponential distribution.

To apply the inverse CDF method, you must first compute the CDF.

$$F(X) = P(X \leq x) = \int_0^x \lambda\, e^{-\lambda x} = 1 - e^{-\lambda x}$$

Thus, the CDF of the exponential distribution is

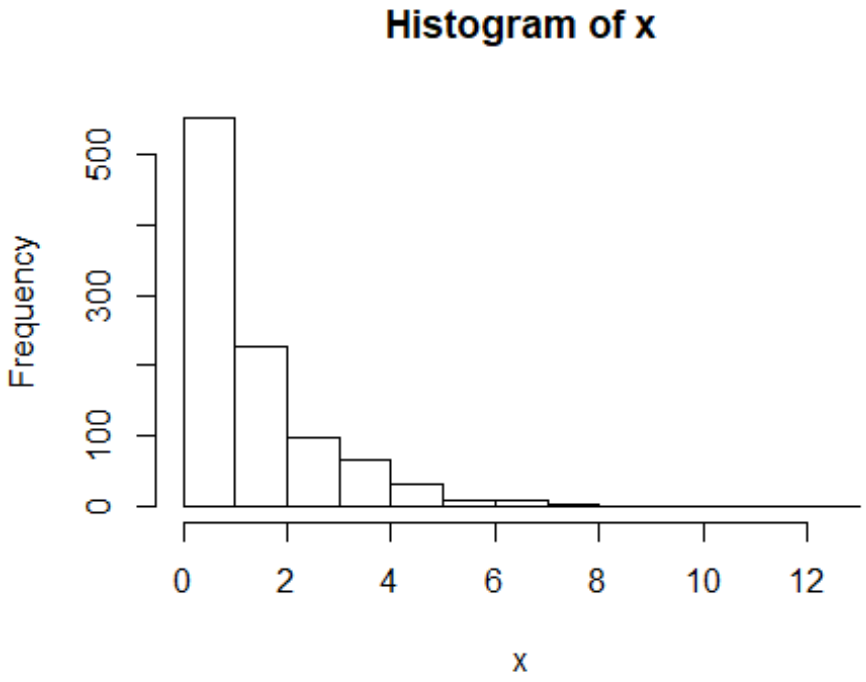
$$F(x) = 1 - e^{-\lambda x} \text{ if } x \geq 0$$

Now the inverse of the CDF can be derived by setting $u = F(x)$ and solving for $x = F^{????1}(u)$.

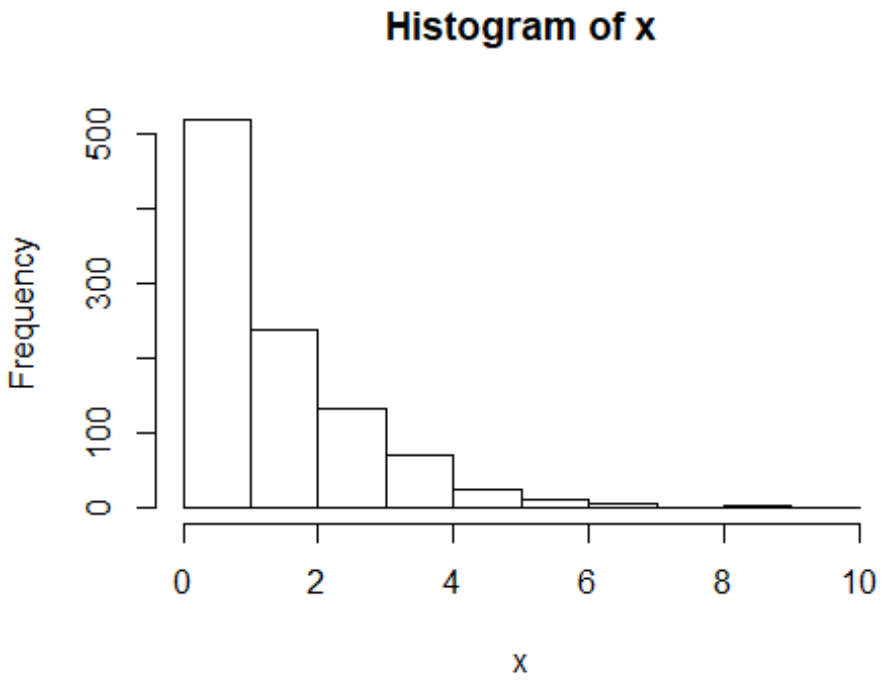
$$u = 1 - e^{-\lambda x}$$

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

```
# Generate 1000 exponential random numbers with lambda = 0.75

# first, we make 1000 uniform random numbers
u = runif(1000)

# then, we implement the inverse formula
lambda = 0.75
x = (-1/lambda)*log(1-u)

hist(x)
```




```
# you can compare it with the exponential random number generator in R
x = rexp(1000, lambda)
hist(x)
```


Histogram of x
Frequency
0
100
300
500
0
2
4
6
8
10
x