
Note: This document is a part of the lectures given to students of IIT Guwahati during the Jan-May 2018 Semester.

We will look at algorithms at the core of Monte Carlo simulation for generating uniformly distributed random variables and methods for transforming these variables to other distribution. These algorithms may be executed millions of times in the course of a simulation, making efficient implementation especially important.

Random Number Generator (General Consideration): At the heart of nearly all Monte Carlo simulation is a sequence of apparently random numbers used to drive the simulation. We will treat this driving sequence as though it was genuinely random. Modern pseudonumber generators are sufficiently good at replicating genuine randomness (even though they are produced by completely deterministic algorithm). Before discussing sequences that appear to be random but are not, we should specify what we mean by a generator of a genuinely random number. We aim at a mechanism for producing a sequence of random variables $U_1, U_2, \ldots$ with the property that:

1. Each U_i is uniformly distributed between 0 and 1.
2. The U_i are mutually independent.

Note:

1. Property 1 is convenient but can be arbitrarily normalized. This means that values uniformly distributed between 0 and $1/2$ would be just as useful. Uniform random variables on the unit interval can be transformed into samples from essentially any other distribution.
2. Property 2 is more important. In other words, all pairs of values should be uncorrelated and more generally the that values of U_i should not be predictable from $U_1, U_2, \ldots, U_{i-1}$.

A random number generator (often called the pseudonumber generator to emphasize that it only mimics randomness) produces a finite sequence of numbers $u_1, u_2, \ldots, u_k$. An effective generator therefore produces values that appear consistent with properties 1 and 2. If the number of values of k is large, the fraction of values falling in any subinterval should be approximately the length of the subinterval-this is uniformity. Independence suggests that there should be no discernible pattern among the values. More precisely, statistical tests for independence would not easily reject segments of the sequence $u_1, u_2, \ldots, u_k$.

Linear Congruence Generator: A linear congruence generator is recurrence relation of the following form:

$$\begin{aligned} x_{i+1} &= ax_i \bmod m, \\ u_{i+1} &= x_{i+1}/m. \end{aligned}$$

Here the multiplier a and the modulus m are integer constants that determine the values generated, given an initial value (seed) x_0. The seed is an integer between 1 and $m-1$ and is ordinarily specified by users. The operation $y \bmod m$ returns the reminder of y (an integer) after division by m. In other words $y \bmod m = y - \lfloor y/m \rfloor m$ where $\lfloor x \rfloor$ denoted the greatest integer less than or equal to x. For example, 7 mod 5 is 2 and 43 mod 5 is 3 . Because the result of mod m is always an integer between 0 and $m-1$ the output values u_i are produced are always between 0 and $(m-1)/m$. In particular they lie in the unit interval. Simplicity and effectiveness has lead to wide usage of linear congruence generators. Notice that it has the form $x_{i+1} = f(x_i)$ and $u_{i+1} = g(x_{i+1})$ for some deterministic functions f and g.

1. Example: Suppose $a = 6$ and $m = 11$. If $x_0 = 1$ then we get the sequence $1, 6, 3, 7, 9, 10, 5, 8, 4, 2, 1, 6$ and the entire sequence repeats. All digits are included before a value was repeated.

2. Example: Suppose $m = 11$, $a = 3$. $x_0 = 1$ give the sequence $1, 3, 9, 5, 4, 1$ and $x_0 = 2$ gives $2, 6, 7, 10, 8, 2$. We see that regardless of the choice of x_0, $a = 3$ produces five distinct numbers before repeating, unlike the case of $a = 6$.

A linear congruence generator that produces all $(m - 1)$ distinct values before repeating is said to have full period.

1. Period Length: Any random number generator so generated will eventually repeat itself. Other things being equal, generators with longer periods are preferred. Longest possible period for a linear congruential generator with modulus m is $m - 1$.

2. Reproductivity: A genuine random sequence should not be reproducible easily. This goal can be accomplished with a linear congruential generator by using the same seed x_0.

3. Speed: This is needed for random number generators since they may be called thousands or even millions of times in a single simulation.

4. Portability: An algorithm for generating random numbers should produce the same sequence of values on all computing platforms.

5. Randomness: Depends on (1) Theoretical Properties (2) Statistical Tests.

General Linear Congruence Generators: The general linear congruence generator takes the form:

$$\begin{aligned} x_{i+1} &= (ax_i + c) \text{ mod } m \\ u_{i+1} &= x_{i+1}/m.. \end{aligned}$$

This is sometimes called a mixed linear congruential generator and the multiplicative case in the previous section is a pure linear congruential generator. Like a and m, the parameter c must be an integer. Simple conditions that ensures that the generator has full period, that is, the number of distinct values generated from any x_0 is $(m - 1)$.

1. If $c \neq 0$ the conditions are:

   (a) c and m are relatively prime.

   (b) Every prime that divides m divides $(a - 1)$.

   (c) $(a - 1)$ is divisible by 4 if m is.

   A simple consequence is that if m is a power of 2, the generator has a full period, if c is odd and $a = 4n + 1$ for some integer n.

2. If $c = 0$ and m is prime, full period is achieved from any $x_0 \neq 0$ if:

   (a) $a^{m-1} - 1$ is a multiple of m.

   (b) $a^j - 1$ is not a multiple of m for $j = 1 : m - 2$.

   A number a satisfying these two properties is called a primitive root of m. When $c = 0$, the sequence x_i becomes $\{x_0, ax_0, a^2x_0 \ldots\} (\text{mod } m)$ and returns at x_0 at smallest k for which $a^k x_0$ mod $m = x_0$, that is, the smallest k for which a^k mod $m = 1$.