1 Generating (pseudo)random numbers

Here we discuss methods to generate pseudorandom numbers from the uniform distribution on $(0, 1)$. If random numbers from another distribution is needed, then appropriate transformation methods (will be discussed in Chapter 3) are applied to uniform pseudorandom numbers. It is best to avoid the end points 0 and 1 when generating from $U(0, 1)$; this is not a theoretical, but a practical issue.

2 Linear Congruential Generators

Introduced by D. H. Lehmer in 1949.

$$x_n \equiv ax_{n-1} + c \mod M \tag{1}$$

where a is called the multiplier, c is called the shift or increment, and M is called the modulus of the generator. The generator produces a sequence of integers $x_1, x_2, ...$ between 0 and $M - 1$, once a seed x_0 is chosen. Uniform random numbers u_n between 0 and 1 are obtained by

$$u_n = \frac{x_n}{M}.$$

Lehmer first considered LCG's where $c = 0$. Such generators are called multiplicative congruential generators (MCG).

2.1 Period

Clearly, one can only produce at most M distinct integers, $\{0, 1, ..., M - 1\}$ in some order, using an LCG. If an MCG is used, then we can have at most $M-1$ distinct (non-zero) integers (omitting zero). The period of a generator is defined as the number of distinct x_n before the generator starts repeating itself.

Example 1 *Consider an LCG with* $M = 16, a = 5, c = 4$:

$$x_n \equiv 5x_{n-1} + 4 \mod 16$$

$$\begin{aligned} x_0 &= 0 \to 0, 4, 8, 12, 0 \\ x_0 &= 1 \to 1, 9, 1 \\ x_0 &= 3 \to 3, 3 \end{aligned}$$

Observe that the period of a sequence may depend on the starting value x_0. Having a short period is certainly not desirable. One can obtain full period by the following choice

$$\begin{aligned} x_n &\equiv x_{n-1} + 1 \mod M \\ x_0 &= 1 \end{aligned}$$

but clearly, the sequence this LCG generates

$$1, 2, 3, ...$$

is far from a random looking sequence!

Theorem 2 (Greenberger; Hull, Dobell, 1962) *The period of LCG is M iff*

1. *$gcd(c, M) = 1$*
2. *$a \equiv 1 \bmod p$ for each prime factor p of M*
3. *$a \equiv 1 \bmod 4$ if 4 divides M*

Example 3 *ANSI C rand() function for Unix, BSD version.*

$$M = 2^{31}, a = 1103515245, c = 12345, x_0 = 12345$$

2.2 Other LCG's

Some popular LCG's that do not have full period.

1. M = prime; a is a primitive root $\bmod M$ ($a^{M-1} \equiv 1$ and $a^x \neq 1 \bmod M$ for any $x < M - 1$); $c = 0; x_0 \neq 0$.
   Then the period is $M - 1$.

   Example 4 *Maple's random number generator.* $M = 10^{12} - 11; a = 427419669081; c = 0$

   Example 5 *A commonly used generator.* $M = 2^{31} - 1$ *(Mersenne prime);* $a = 7^5; c = 0$

2. $M = 2^N; a \equiv 5 \bmod 8; c = 0; x_0$ is odd. Then the period is $M/4 = 2^{N-2}$.

   Example 6 *Cray Supercomputers.* $M = 2^{48}; a = 44485709377909$. *The seed specifies the lower 32 bits of x_0, with the lowest bit set to prevent the seed taking an even value. The upper 16 bits are set to 0.*

Theorem 7 (Marsaglia) *The output of any LCG lies on a simple lattice in a k-space with axes representing successive numbers in the output.*

Example 8 *Consider the simple MCG with $M = 31, a = 3$ and $x_0 = 9$. The period is 30, and 3 is a primitive root* $\bmod 31$. *The numbers are*
{27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1, 3, 9, ***27****}. The output may look like a random sample from the discrete uniform distribution over the integers 1 to 30. However, if we partition the list as (27,19),(19,26),... and plot the pairs we obtain Figure 1. A non-overlapping partition gives Figure 2.*

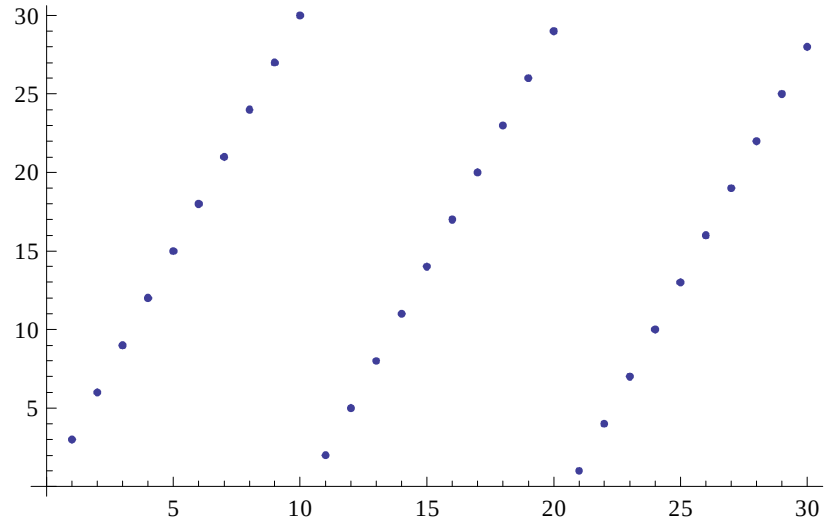


Figure 1: A simple MCG. Overlapping pairs.

Example 9 (RANDU) *A faulty MCG which was widely used in 1960's, on IBM 360.370.*

$$\begin{aligned} x_n &\equiv 65539 x_{n-1} \bmod 2^{31} \\ &\equiv (65539)^2 x_{n-2} \\ &\equiv (2^{16}+3)^2 x_{n-2} \\ &\equiv 2^{32} x_{n-2} + 6 \cdot 2^{16} x_{n-2} + 9 x_{n-2} \\ &\equiv 6(\underbrace{2^{16} x_{n-2} + 3 x_{n-2}}_{65539 x_{n-2} \equiv x_{n-1}}) - 9 x_{n-2} \\ &\equiv 6 x_{n-1} - 9 x_{n-2} \end{aligned}$$

therefore

$$x_n - 6x_{n-1} + 9x_{n-2} = k 2^{31}$$

for some integer k. Divide both sides by 2^{31} to get

$$u_n - 6u_{n-1} + 9u_{n-2} = k$$

where $0 < u_i < 1$. Then, $k < 10$ and $k > -6$, and thus the triplets must lie on no more than 15 planes in R^3.

2.3 Computer implementation of LCG's

Rounding error in x_n may propagate and produce a different sequence than intended due to the recurrence of LCG's. Several algebraic tricks have been

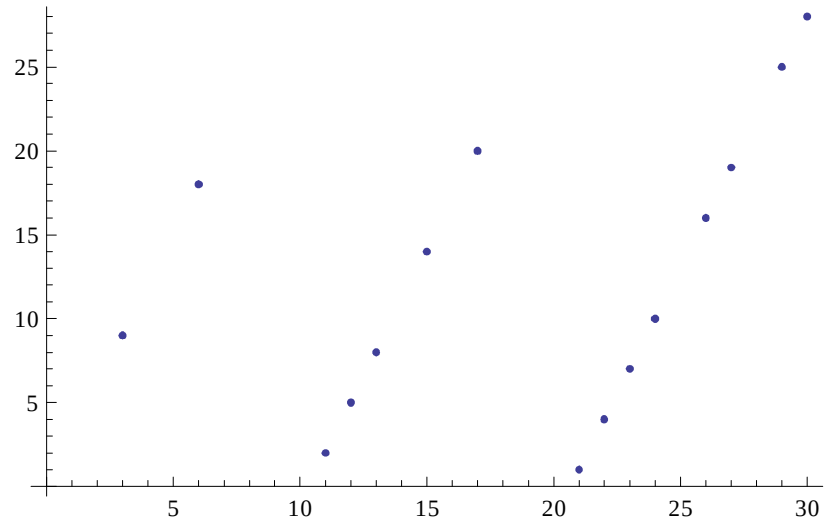


Figure 2: A simple MCG. Non-overlapping pairs.

used in the past so that computations are exact. Good implementation of an LCG depends on the type of architecture, the precision available, the base of the arithmetic, etc. See [1], and [3] for more.

3 Congruential Generators with Dependence on More Terms

3.1 Quadratic generators

They have the form

$$x_n \equiv ax_{n-1}^2 + bx_{n-1} + c (\mathrm{mod}\, M)$$

An example is a generator by R. Coveyou who suggested

$$\begin{aligned} x_n &\equiv x_{n-1}(x_{n-1}+1) \bmod 2^e \\ x_0 &\equiv 2 \bmod 4 \end{aligned}$$

3.2 Lagged Fibonacci generators

The Fibonacci generator $x_n \equiv x_{n-1} + x_{n-2} (\mathrm{mod}\, M)$ is not a good generator; it fails several statistical tests. However, a modification is commonly used today:

$$x_n \equiv x_{n-r} + x_{n-s} (\mathrm{mod}\, M)$$

where $n \geq r, 0 < s < r$, and r & s are the lags. The low-order bits of these generators are not very random if lags are small.

Another generalization is to replace addition by another binary operation, such as $-, \times$, or exclusive or if $M = 2^e$ (the operation then is equivalent to addition mod 2)

Example 10 $r = 607; s = 273; m = 2^{31}$;*operation is* $-$, *i.e.,*

$$x_n \equiv x_{n-607} + x_{n-273} (\mathrm{mod}\, 2^{31})$$

has period $(2^{607} - 1)2^{31-1} \approx 10^{191}$, *if at least one member of the initial sequence* $x_0, ..., x_{606}$ *is odd. There are other pairs of lags* (r, s) *that have period* $(2^r - 1)2^{31-1}$; *see [2].*

3.3 Multiple Recursive generators

These have the form

$$x_n \equiv a_1 x_{n-1} + ... + a_k x_{n-k} \ (\mathrm{mod}\, p)$$

where p is prime. It is possible to find $a_1, ..., a_k$ such that the sequence has period length $p^k - 1$. Initial values $x_0, ..., x_{k-1}$ can be chosen arbitrarily, but not all zero. A special case, $p = 2$, is commonly used today.

3.3.1 Feedback Shift Register Generators

Introduced by Tausworthe, 1965. Consider the polynomial

$$f(z) = z^k - (a_1 z^{k-1} + ... + a_{k-1} z + a_k)$$

over the field defined over the integers $\{0, 1\}$. Then, the period of

$$x_n \equiv a_1 x_{n-1} + ... + a_k x_{n-k} \ (\mathrm{mod}\, 2)$$

is $2^k - 1$ iff the polynomial is irreducible (provided not all initial values x_i are zero.)

The sequence of binary digits $x_n, n = 1, ...$, is then partitioned into blocks, and each block is transformed to an integer. For example, if blocks of size 4 are formed $[0, 1, 1, 0], [1, 1, 1, 0], ...$, then the first block is converted to an integer as $0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 0$.

If the partition into blocks is done with a delay between blocks, then we obtain the so-called generalized feedback shift register generators (GFSR). A good example is tt800 and Mersenne Twister (these generators have a further modification of the basic FSR generators.)

4 Inversive Congruential Generators

Introduced by Eichenauer & Lehn (1986). They have the form

$$x_{n+1} \equiv a x_n^{-1} + c \pmod{M}$$

where x_n^{-1} is the multiplicative inverse of $x_n \bmod M$, if it exists, or else it is zero. These generators do not yield regular planes like LCG's, and several tests of randomness reveal their superior properties over LCG's. However, they are computationally demanding.

5 Other non-linear generators

$$x_{n+1} \equiv g(x_n) \mod M$$

where g is some function. Examples are

1. $g(x) = x^2$; introduced by Blum, Blum, Shub, 1986)
2. $g(x) = ax^{-1} + bx + c$; introduced by Kato, Wu, Yanagihara, 1996)

6 Combining Generators

Wichmann and Hill (1982, 84) introduced the following generator

$$\begin{aligned} x_n &\equiv 171 x_{n-1} \bmod 30269 \\ y_n &\equiv 172 y_{n-1} \bmod 30307 \\ z_n &\equiv 170 z_{n-1} \bmod 30323 \end{aligned}$$

and

$$u_n \equiv \frac{x_n}{30269} + \frac{y_n}{30307} + \frac{z_n}{30323} \bmod 1$$

The period of this generator is 10^{12}.

L'Ecuyer suggested the following generator

$$\begin{aligned} x_n &\equiv 40014 x_{n-1} \bmod 2147483563 \\ y_n &\equiv 40692 y_{n-1} \bmod 2147483399 \\ z_n &\equiv x_n - y_n \end{aligned}$$

where if $z_n < 1$ then $z_n = z_n + 2147483562$, and

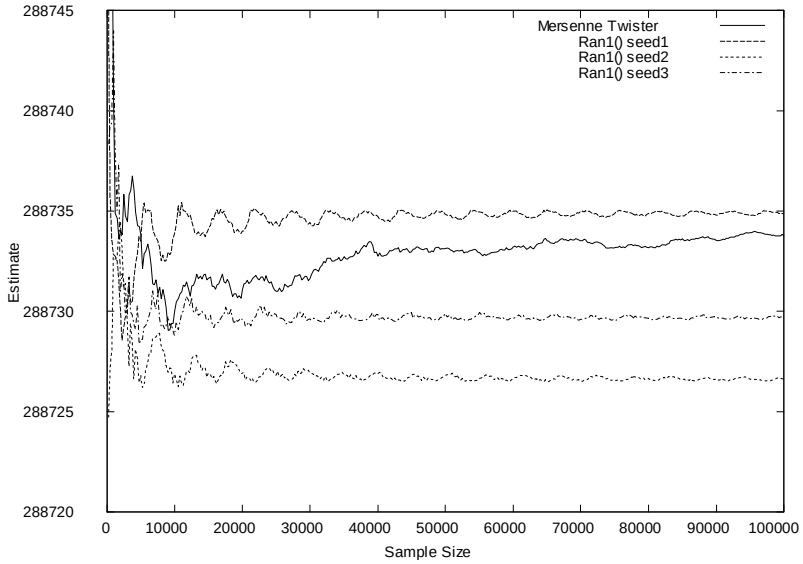
$$u_n = 4.656613 z_n \times 10^{-10}.$$

The period of this generator is about 10^{18}.

See [3] (pages 52, 53) for a C code for another combined generator suggested by L'Ecuyer.

6.1 A cautionary tale

A combined random number generator, ran1(), that was given in Numerical Recipes in C, First Edition (Press et. al. 1988), used two LCG's to construct a uniform number, and then a third LCG to shuffle the order numbers are generated. However, after this generator was applied to pricing collateralized mortgage obligations (CMO), it was discovered to be faulty. The generator has a cycle of $360 \times 10,800$, and since CMO is a 360-dimensional problem, there are only 10,800 different paths. The following graph shows how estimates obtained from Ran1 converges to different values for different seeds. The estimates given by Mersenne Twister can be taken as the "true" price of the CMO. See [4], [5].



Ran1() with different seeds and Mersenne Twister. 360 months.

7 Suggestions

- Use at least two different types of well tested random number generators. Having the Mersenne Twister and one of the combined generators in your library is probably a good idea. Subtle correlations in a generator can always cause problems in a particular problem.
- Parallel generators
  - Blocking, Leap-frogging, Parameterization
    (SPRNG library - http://sprng.cs.fsu.edu/)

References

[1] James E. Gentle. Random Number Generation and Monte Carlo Methods. Springer. 1998.

[2] Donald E. Knuth. The Art of Computer Programming, Vol 2. Addison Wesley.

[3] Paul Glasserman. Monte Carlo Methods in Fianncial Engineering. Springer. 2004.

[4] A. Tajima, S. Ninomiya, S. Tezuka, "On the Anomaly of Ran1() in Monte Carlo Pricing of Financial Derivatives", Proceedings of the 1996 Winter Simulation Conference.

[5] G. Ökten, M. Willyard, "Parameterization based on randomized quasi-Monte Carlo methods", Parallel and Distributed Computing in Finance, Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium.