31st EDITION

CRC

STANDARD MATHEMATICAL TABLES AND FORMULAE

DANIEL ZWILLINGER

CHAPMAN & HALL/CRC

A CRC Press Company

Boca Raton London New York Washington, D.C.

Table of Contents

Chapter 7

Probability and Statistics

1-58488-291-3/02/$0.00+$1.50

7.6 RANDOM NUMBER GENERATION

7.6.1 METHODS OF PSEUDORANDOM NUMBER GENERATION

In Monte Carlo applications, and other computational situations where randomness is required, one must appeal to random numbers for assistance. While it has been argued that numbers measured from a physical process known to be random should be used, it has been infinitely more practical to use simple recursions that produce numbers that behave as random in applications and with respect to statistical tests of randomness. These are so-called *pseudorandom numbers* and are produced by a pseudorandom number generator (PRNG). Depending on the application, either integers in some range or floating point numbers in $[0, 1)$ are the desired output from a PRNG. Since most PRNGs use integer recursions, a conversion into integers in a desired range or into a floating point number in $[0, 1)$ is required. If x_n is an integer produced by some PRNG in the range $0 \le x_n \le M - 1$, then an integer in the range $0 \le x_n \le N - 1$, with $N \le M$, is given by $y_n = \lfloor \frac{N}{M} x_n \rfloor$. If $N \ll M$, then $y_n = x_n \pmod{N}$ may be used. Alternately, if a floating point value in $[0, 1)$ is desired, let $y_n = x_n / M$.

7.6.1.1 Linear congruential generators

Perhaps the oldest generator still in use is the *linear congruential generator* (LCG). The underlying integer recursion for LCGs is

$$x_n = a x_{n-1} + b \pmod{M}. \tag{7.6.1}$$

Equation (7.6.1) defines a periodic sequence of integers modulo M starting with x_0, the initial seed. The constants of the recursion are referred to as the *modulus* M, *multiplier* a, and *additive constant* b. If $M = 2^m$, a very efficient implementation is possible. Alternately, there are theoretical reasons why choosing M prime is optimal. Hence, the only moduli that are used in practical implementations are $M = 2^m$ or the prime $M = 2^p - 1$ (i.e., M is a Mersenne prime). With a Mersenne prime or any modulus "close to" 2^p, modular multiplication can be implemented at about twice the computational cost of multiplication modulo 2^p.

Equation (7.6.1) yields a sequence $\{x_n\}$ whose period, denoted $\mathrm{Per}(x_n)$, depends on M, a, and b. The values of the maximal period for the three most common cases used and the conditions required to obtain them are

a	b	M	$\mathrm{Per}(x_n)$
Primitive root of M	Anything	Prime	$M - 1$
3 or 5 (mod 8)	0	2^m	2^{m-2}
1 (mod 4)	1 (mod 2)	2^m	2^m

A major shortcoming of LCGs modulo a power-of-two compared with prime modulus LCGs derives from the following theorem for LCGs:

THEOREM 7.6.1

Define the following LCG sequence: $x_n = ax_{n-1} + b \pmod{M_1}$. *If* M_2 *divides* M_1 *then* $y_n = x_n \pmod{M_2}$ *satisfies* $y_n = ay_{n-1} + b \pmod{M_2}$.

Theorem 7.6.1 implies that the k least-significant bits of any power-of-two modulus LCG with $\mathrm{Per}(x_n) = 2^m = M$ has $\mathrm{Per}(y_n) = 2^k, 0 < k \le m$. Since a long period is crucial in PRNGs, when these types of LCGs are employed in a manner that makes use of only a few least-significant-bits, their quality may be compromised. When M is prime, no such problem arises.

Since LCGs are in such common usage, here is a list of parameter values mentioned in the literature. The Park–Miller LCG is widely considered a minimally acceptable PRNG. Using any values other than those in the following table may result in a "weaker" LCG.

a	b	M	Source
7^5	0	$2^{31}-1$	Park–Miller
131	0	2^{35}	Neave
16333	25887	2^{15}	Oakenfull
3432	6789	9973	Oakenfull
171	0	30269	Wichman–Hill

7.6.1.2 Shift-register generators

Another popular method of generating pseudorandom numbers is using binary shift-register sequences to produce pseudorandom bits. A *binary shift-register sequence* (SRS) is defined by a binary recursion of the type,

$$x_n = x_{n-j_1} \oplus x_{n-j_2} \oplus \cdots \oplus x_{n-j_k}, \qquad j_1 < j_2 < \cdots < j_k = \ell, \tag{7.6.2}$$

where $\oplus$ is the exclusive "or" operation. Note that $x \oplus y \equiv x + y \pmod 2$. Thus the new bit, x_n, is produced by adding k previously computed bits together modulo 2. The implementation of this recurrence requires keeping the last ℓ bits from the sequence in a shift register, hence the name. The longest possible period is equal to the number of non-zero ℓ-dimensional binary vectors, namely $2^{\ell} - 1$.

A sufficient condition for achieving $\mathrm{Per}(x_n) = 2^{\ell} - 1$ is that the characteristic polynomial, corresponding to Equation (7.6.2), be primitive modulo 2. Since primitive trinomials of nearly all degrees of interest have been found, SRSs are usually implemented using two-term recursions of the form,

$$x_n = x_{n-k} \oplus x_{n-\ell}, \qquad 0 < k < \ell. \tag{7.6.3}$$

In these two-term recursions, k is the *lag* and ℓ is the register length. Proper choice of the pair (ℓ, k) leads to SRSs with $\mathrm{Per}(x_n) = 2^{\ell} - 1$. Here is a list with suitable (ℓ, k) pairs:

Primitive trinomial exponents					
(5,2)	(7,1)	(7,3)	(17,3)	(17,5)	(17,6)
(31,3)	(31,6)	(31,7)	(31,13)	(127,1)	(521,32)

7.6.1.3 Lagged-Fibonacci generators

Another way of producing pseudorandom numbers uses lagged-Fibonacci generators. The term "lagged-Fibonacci" refers to two-term recurrences of the form,

$$x_n = x_{n-k} \diamond x_{n-\ell}, \qquad 0 < k < \ell, \tag{7.6.4}$$

where $\diamond$ refers to one of the three common methods of combination: (1) addition modulo 2^m, (2) multiplication modulo 2^m, or (3) bitwise exclusive 'OR'ing of m-long bit vectors. Combination method (3) can be thought of as a special implementation of a two-term shift-register sequence.

Using combination method (1) leads to *additive lagged-Fibonacci sequences* (ALFSs). If x_n is given by

$$x_n = x_{n-k} + x_{n-\ell} \pmod{2^m}, \qquad 0 < k < \ell, \tag{7.6.5}$$

then the maximal period is $\text{Per}(x_n) = (2^\ell - 1)2^{m-1}$.

ALFSs are especially suitable for producing floating point deviates using the real-valued recursion $y_n = y_{n-k} + y_{n-\ell} \pmod 1$. This circumvents the need to convert from integers to floating point values and allows floating point hardware to be used. One caution with ALFSs is that Theorem 7.6.1 holds, and so the low-order bits have periods that are shorter than the maximal period. However, this is not nearly the problem as in the LCG case. With ALFSs, the j least-significant bits will have period $(2^\ell - 1)2^{j-1}$, so, if ℓ is large, there really is no problem. Note that one can use the table of primitive trinomial exponents to find (ℓ, k) pairs that give maximal period ALFSs.

7.6.1.4 Non-linear generators

A recent development among PRNGs are non-linear integer recurrences. For example, if in Equation (7.6.4) "$\diamond$" referred to multiplication modulo 2^m, then this recurrence would be a *multiplicative lagged-Fibonacci generator* (MLFG), a non-linear generator. The mathematical structure of non-linear generators is qualitatively different than that of linear generators. Thus, their defects and deficiencies are thought to be complementary to their linear counterparts.

The maximal period of a MLFG is $\text{Per}(x_n) = (2^\ell - 1)2^{m-3}$, a factor of 4 shorter than the corresponding ALFS. However, there are benefits to using multiplication as the combining function due to the bit mixing achieved. Because of this, the perceived quality of the MLFG is considered superior to an ALFS with the same lag, ℓ.

We conclude by defining two non-linear generators, the *inversive congruential generators* (ICGs), which were designed as non-linear analogs of the LCG.

1. The *implicit ICG* is defined by the following recurrence that is almost that of an LCG

$$x_n = a\overline{x_{n-1}} + b \pmod M. \tag{7.6.6}$$

The difference is that we must also take the multiplicative inverse of x_{n-1}, which is defined by $\overline{x_{n-1}}\, x_{n-1} \equiv 1 \pmod M$, and $\overline{0} = 0$. This recurrence is indeed non-linear, and avoids some of the problems inherent in linear recurrences, such as the fact that linear tuples must lie on hyperplanes.

2. The *explicit ICG* is

$$x_n = \overline{an + b} \pmod{M}. \tag{7.6.7}$$

One drawback of ICGs is the cost of inversion, which is $O(\log_2 M)$ times the cost of multiplication modulo M.

7.6.2 GENERATING NON-UNIFORM RANDOM VARIABLES

Suppose we want deviates from a distribution with probability density function $f(x)$ and distribution function $F(x) = \int_{-\infty}^{x} f(u)\, du$. In the following "$y$ is $U[0,1)$" means y is uniformly distributed on $[0,1)$.

Two general techniques for converting uniform random variables into those from other distributions are as follows:

1. The *inverse transform method*:

   If y is $U[0,1)$, then the random variable $F^{-1}(y)$ will have its density equal to $f(x)$. (Note that $F^{-1}(y)$ exists since $0 \le F(x) \le 1$.)

2. The *acceptance-rejection method*:

   Suppose the density can be written as $f(x) = Ch(x)g(x)$ where $h(x)$ is the density of a computable random variable, the function g satisfies $0 < g(x) \le 1$, and $C^{-1} = \int_{-\infty}^{\infty} h(u)g(u)\, du$ is a normalization constant. If x is $U[0,1)$, y has density $h(x)$, and if $x < g(y)$, then x has density $f(x)$. Thus one generates $\{x, y\}$ pairs, rejecting both if $x \ge g(y)$ and returning x if $x < g(y)$.

Examples of the inverse transform method:

1. *Exponential distribution*: The exponential distribution with rate λ has $f(x) = \lambda e^{-\lambda x}$ (for $x \ge 0$) and $F(x) = 1 - e^{-\lambda x}$. Thus $u = F(x)$ can be solved to give $x = F^{-1}(u) = -\lambda^{-1} \ln(1 - u)$. If u is $U[0,1)$, then so is $1 - u$. Hence $x = -\lambda^{-1} \ln u$ is exponentially distributed with rate λ.
2. *Normal distribution*: Suppose the z_i's are normally distributed with density function $f(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$. The polar transformation then gives random variables $r = \sqrt{z_1^2 + z_2^2}$ (exponentially distributed with $\lambda = 2$) and $\theta = \tan^{-1}(z_2/z_1)$ (uniformly distributed on $[-\frac{\pi}{2}, \frac{\pi}{2}]$). Inverting these relationships results in $z_1 = \sqrt{-2\ln x_1} \cos 2\pi x_2$ and $z_2 = \sqrt{-2\ln x_1} \sin 2\pi x_2$; each is normally distributed when x_1 and x_2 are $U[0,1)$. (This is the *Box–Muller technique*.)

Examples of the rejection method:

1. *Exponential distribution with $\lambda = 1$*:

   (a) Generate random numbers $\{U_i\}_{i=1}^{N}$ uniformly in $[0,1]$, stopping at $N = \min\{n \mid U_1 \ge U_2 \ge U_{n-1} < U_n\}$.

   (b) If N is even, accept that run, and go to step (c). If N is odd reject the run, and return to step (a).

(c) Set X equal to the number of failed runs plus U_1 (the first random number in the successful run).

2. *Normal distribution*:

   (a) Select two random variables (V_1, V_2) from $U[0,1)$. Form $R = V_1^2 + V_2^2$.
   (b) If $R > 1$, then reject the (V_1, V_2) pair, and select another pair.
   (c) If $R < 1$, then $x = V_1\sqrt{-2\dfrac{\ln R}{R}}$ has a $N(0,1)$ distribution.

3. *Normal distribution*:

   (a) Select two exponentially distributed random variables with rate 1: (V_1, V_2).
   (b) If $V_2 \geq (V_1-1)^2/2$, then reject the (V_1, V_2) pair, and select another pair.
   (c) Otherwise, V_1 has a $N(0,1)$ distribution.

4. *Cauchy distribution*: To generate values of X from $f(x) = \frac{1}{\pi(1+x^2)}$ on $-\infty < x < \infty$,

   (a) Generate random numbers U_1, U_2 (uniform on $[0,1)$), and set $Y_1 = U_1 - \frac{1}{2}, Y_2 = U_2 - \frac{1}{2}$.
   (b) If $Y_1^2 + Y_2^2 \leq \frac{1}{4}$, then return $X = Y_1/Y_2$. Otherwise return to step (a).

   To generate values of X from a Cauchy distribution with parameters β and θ, $f(x) = \dfrac{\beta}{\pi\left[\beta^2 + (x-\theta)^2\right]}$, for $-\infty < x < \infty$, construct X as above, and then use $\beta X + \theta$.

7.6.2.1 Discrete random variables

The density function of a discrete random variable that attains finitely many values can be represented as a vector $\mathbf{p} = (p_0, p_1, \ldots, p_{n-1}, p_n)$ by defining the probabilities $P(x = j) = p_j$ (for $j = 0, \ldots, n$). The distribution function can be defined by the vector $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}, 1)$, where $c_j = \sum_{i=0}^{j} p_i$. Given this representation of $F(x)$, we can apply the inverse transform by computing x to be $U[0,1)$, and then finding the index j so that $c_j \leq x < c_{j+1}$. In this case event j will have occurred. Examples:

1. (Binomial distribution) The binomial distribution with n trials of mean p has $p_j = \binom{n}{j}p^j(1-p)^{n-j}$, for $j = 0, \ldots, n$.

   (a) As an example, consider the result of flipping a fair coin. In 2 flips, the probability of obtaining $(0,1,2)$ heads is $\mathbf{p} = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$. Hence $\mathbf{c} = (\frac{1}{4}, \frac{3}{4}, 1)$. If x (chosen from $U[0,1)$) turns out to be say, 0.4, then "1 head" is returned (since $\frac{1}{4} \leq 0.4 < \frac{3}{4}$).
   (b) Note that, when n is large, it is costly to compute the density and distribution vectors. When n is large and relatively few binomially distributed pseudorandom numbers are desired, an alternative is to use the normal approximation to the binomial.

(c) Alternately, one can form the sum $\sum_{i=1}^{n} \lfloor u_i + p \rfloor$, where each u_i is $U[0,1)$.

2. (Geometric distribution) To simulate a value from $P(X = i) = p(1-p)^{i-1}$ for $i \geq 1$, use $X = 1 + \left\lceil \frac{\log U}{\log(1-p)} \right\rceil$.

3. (Poisson distribution) The Poisson distribution with mean λ has $p_j = \lambda^j e^{-\lambda}/j!$ for $j \geq 0$. The Poisson distribution counts the number of events in a unit time interval if the times are exponentially distributed with rate λ. Thus if the times t_i are exponentially distributed with rate λ, then j will be Poisson distributed with mean λ when $\sum_{i=0}^{j} t_i \leq 1 \leq \sum_{i=0}^{j+1} t_i$. Since $t_i = -\lambda^{-1} \ln u_i$, where u_i is $U[0,1)$, the previous equation may be written as $\prod_{i=0}^{j} u_i \geq e^{-\lambda} \geq \prod_{i=0}^{j+1} u_i$. This allows us to compute Poisson random variables by iteratively computing $P_j = \prod_{i=0}^{j} u_i$ until $P_j < e^{-\lambda}$. The first such j that makes this inequality true will have the desired distribution.

Random variables can be simulated using the following table (each U and U_i is uniform on the interval $[0,1)$):

Distribution	Density	Formula for deviate
Binomial	$p_j = \binom{n}{j} p^j (1-p)^{n-j}$	$\sum_{i=1}^{n} \lfloor U_i + p \rfloor$
Cauchy	$f(x) = \frac{\sigma}{\pi(x^2+\sigma^2)}$	$\sigma \tan(\pi U)$
Exponential	$f(x) = \lambda e^{-\lambda x}$	$-\lambda^{-1} \ln U$
Pareto	$f(x) = ab^a / x^{a+1}$	$b/U^{1/a}$
Rayleigh	$f(x) = x/\sigma e^{-x^2/2\sigma^2}$	$\sigma\sqrt{-\ln U}$

7.6.2.2 Testing pseudorandom numbers

The prudent way to check a complicated computation that makes use of pseudorandom numbers is to run it several times with different types of pseudorandom number generators and see if the results appear consistent across the generators. The fact that this is not always possible or practical has led researchers to develop statistical tests of randomness that should be passed by general purpose pseudorandom number generators. Some common tests are the spectral test, the equidistribution test, the serial test, the runs test, the coupon collector test, and the birthday spacing test.