

# Random Number Generation

Pierre L'Ecuyer<sup>1</sup>

## 1 Introduction

The fields of probability and statistics are built over the abstract concepts of probability space and random variable. This has given rise to elegant and powerful mathematical theory, but exact implementation of these concepts on conventional computers seems impossible. In practice, random variables and other random objects are *simulated* by *deterministic algorithms*. The purpose of these algorithms is to produce sequences of numbers or objects whose behavior is very hard to distinguish from that of their “truly random” counterparts, at least for the application of interest. Key requirements may differ depending on the context. For Monte Carlo methods, the main goal is to reproduce the statistical properties on which these methods are based, so that the Monte Carlo estimators behave as expected, whereas for gambling machines and cryptology, observing the sequence of output values for some time should provide no practical advantage for predicting the forthcoming numbers better than by just guessing at random.

In computational statistics, random variate generation is usually made in two steps: (1) generating imitations of independent and identically distributed (i.i.d.) random variables having the uniform distribution over the interval  $(0, 1)$  and (2) applying transformations to these i.i.d.  $U(0, 1)$  random variates to generate (or imitate) random variates and random vectors from arbitrary distributions. These two steps are essentially independent and the world's best experts on them are two different groups of scientists, with little overlap. The expression (*pseudo*)*random number generator* (RNG) usually refers to an algorithm used for step (1).

In principle, the simplest way of generating a random variate  $X$  with distribution function  $F$  from a  $U(0, 1)$  random variate  $U$  is to apply the inverse of  $F$  to  $U$ :

$$X = F^{-1}(U) \stackrel{\text{def}}{=} \min\{x \mid F(x) \geq U\}. \quad (1)$$

This is the *inversion* method. It is easily seen that  $X$  has the desired distribution:  $P[X \leq x] = P[F^{-1}(U) \leq x] = P[U \leq F(x)] = F(x)$ . Other methods are sometimes preferable when  $F^{-1}$  is too difficult or expensive to compute, as will be seen later.

The remainder of this chapter is organized as follows. In the next section, we give a definition and the main requirements of a uniform RNG. Generators based on linear recurrences modulo a large integer  $m$ , their lattice structure and quality criteria, and their implementation, are covered in Sect. 3. In Sect. 4, we have a similar discussion for RNGs based on linear recurrences modulo 2. Nonlinear RNGs are briefly presented in Sect. 5. In Sect. 6, we discuss empirical statistical testing of RNGs and give some examples. Sect. 7 contains a few pointers to recommended RNGs and software. In Sect. 8, we cover non-uniform random variate generators. We first discuss inversion and its implementation in various settings. We then explain the rejection, ratio-of-uniform, composition and convolution methods, provide pointers to other methods that apply in special cases, **and discuss multivariate distributions.**

---

Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal (Québec), H9S 5B8, Canada. <http://www.iro.umontreal.ca/~lecuyer>

Important basic references that we recommend are Knuth (1998), L'Ecuyer (1994, 1998), Niederreiter (1992), and Tezuka (1995) for uniform RNGs, and Devroye (1986, 2006), Gentle (2003), and Hörmann et al. (2004) for non-uniform RNGs.

## 2 Uniform Random Number Generators

### 2.1 Physical Devices

Random numbers can be generated via physical mechanisms such as the timing between successive events in atomic decay, thermal noise in semiconductors, **photon counting and photon trajectory detectors**, and the like. A key issue when constructing a RNG based on a physical device is that a “random” or “chaotic” output does not suffice; the numbers produced must be, at least to a good approximation, realizations of *independent and uniformly distributed* random variables. If the device generates a stream of bits, which is typical, then each bit should be 0 or 1 with equal probability, and be independent of all the other bits. In general, this cannot be proved, so one must rely on the results of empirical statistical testing to get convinced that the output values have the desired statistical behavior, **at least approximately**. Not all these devices are reliable, **but some are and, as far as we know, they pass all statistical tests that can be run in reasonable time**.

For computational statistics, physical devices have several disadvantages compared to a good algorithmic RNG that stands in a few lines of code. For example, (a) they are much more cumbersome to install and run; (b) they are more costly; (c) they are slower; (d) they cannot reproduce exactly the same sequence twice. Item (d) is important in several contexts, including program verification and debugging as well as comparison of similar systems by simulation with common random numbers to reduce the variance (Bratley et al., 1987, Fishman, 1996, Law and Kelton, 2000). Nevertheless, these physical RNGs can be useful for selecting the seed of an algorithmic RNG, more particularly for applications in cryptology and for gaming machines, where frequent reseeding of the RNG with an external source of entropy (or randomness) is important. A good algorithmic RNG whose seed is selected at random can be viewed as an extensor of randomness, stretching a short random seed into a long sequence of *pseudorandom* numbers.

### 2.2 Generators Based on a Deterministic Recurrence

RNGs used for simulation and other statistical applications are almost always based on deterministic algorithms that fit the following framework, taken from L'Ecuyer (1994): a RNG is a structure  $(\mathcal{S}, \mu, f, \mathcal{U}, g)$  where  $\mathcal{S}$  is a finite set of *states* (the *state space*),  $\mu$  is a probability distribution on  $\mathcal{S}$  used to select the *initial state* (or *seed*)  $s_0$ ,  $f : \mathcal{S} \rightarrow \mathcal{S}$  is the *transition function*,  $\mathcal{U}$  is the *output space*, and  $g : \mathcal{S} \rightarrow \mathcal{U}$  is the *output function*. Usually,  $\mathcal{U} = (0, 1)$ , and we shall assume henceforth that this is the case. The state of the RNG evolves according to the recurrence  $s_i = f(s_{i-1})$ , for  $i \geq 1$ , and the *output* at step  $i$  is  $u_i = g(s_i) \in \mathcal{U}$ . The output values  $u_0, u_1, u_2, \dots$  are the so-called *random numbers* produced by the RNG.

Because  $\mathcal{S}$  is finite, there must be some finite  $l \geq 0$  and  $j > 0$  such that  $s_{l+j} = s_l$ . Then, for all  $i \geq l$ , one has  $s_{i+j} = s_i$  and  $u_{i+j} = u_i$ , because both  $f$  and  $g$  are deterministic. That is, the state and output sequences are eventually periodic. The smallest positive  $j$  for which this happens is called the *period* of the RNG, and is denoted by  $\rho$ . When  $l = 0$ , the sequence is said to be *purely periodic*. Obviously,  $\rho \leq |\mathcal{S}|$ , the cardinality of  $\mathcal{S}$ . If the state has a  $k$ -bit representation on the computer, then  $\rho \leq 2^k$ . Good RNGs are designed so that their period  $\rho$  is not far from that upper bound. In general, the value of  $\rho$  may depend on the seed  $s_0$ , but good RNGs are normally designed so that the period is the same for all admissible seeds.

In practical implementations, it is important that the output be *strictly* between 0 and 1, because  $F^{-1}(U)$  is often infinite when  $U$  is 0 or 1. All good implementations take care of that. However, for the mathematical analysis of RNGs, we often assume that the output space is  $[0, 1)$  (i.e., 0 is admissible), because this simplifies the analysis considerably without making much difference in the mathematical structure of the generator.

## 2.3 Quality Criteria

What important quality criteria should we consider when designing RNGs? An extremely *long period* is obviously essential, to make sure that no wrap-around over the cycle can occur in practice. The length of the period must be guaranteed by a mathematical proof. The RNG must also be *efficient* (run fast and use only a small amount of memory), *repeatable* (able to reproduce exactly the same sequence as many times as we want), and *portable* (work the same way in different software/hardware environments). The availability of efficient *jump-ahead* methods that can quickly compute  $s_{i+\nu}$  given  $s_i$ , for any large  $\nu$  and any  $i$ , is also very useful, because it permits one to partition the RNG sequence into long disjoint *streams* and *substreams* of random numbers, to create an arbitrary number of *virtual generators* from a single RNG (Law and Kelton, 2000, L'Ecuyer et al., 2002, L'Ecuyer, 2008). These virtual generators can be used on parallel processors or to support different sources of randomness in a large simulation model, for example.

To show that these properties are *not sufficient*, consider a RNG with state space  $\mathcal{S} = \{0, \dots, 2^{1000} - 1\}$ , transition function  $s_{i+1} = f(s_i) = (s_i + 1) \bmod 2^{1000}$ , and  $u_i = g(s_i) = s_i/2^{1000}$ . This RNG has period  $2^{1000}$  and enjoys all the nice properties described in the preceding paragraph, but it is far from imitating “randomness.”

A sequence of real-valued random variables  $u_0, u_1, u_2, \dots$  are i.i.d.  $U(0, 1)$  if and only if for all integers  $i \geq 0$  and  $t > 0$ , the vector  $\mathbf{u}_{i,t} = (u_i, \dots, u_{i+t-1})$  is uniformly distributed over the  $t$ -dimensional unit hypercube  $(0, 1)^t$ . Of course, this cannot hold for algorithmic RNGs because any vector of  $t$  successive values produced by the generator must belong to the *finite set*

$$\Psi_t = \{(u_0, \dots, u_{t-1}) : s_0 \in \mathcal{S}\},$$

which is the set of all vectors of  $t$  successive output values, from all possible initial states. Here we interpret  $\Psi_t$  as a *multiset*, which means that the vectors are counted as many times as they appear, and the cardinality of  $\Psi_t$  is exactly equal to that of  $\mathcal{S}$ .

Suppose we select the seed  $s_0$  at random, uniformly over  $\mathcal{S}$ . This can be approximated by using some physical device, for example. Then, the vector  $\mathbf{u}_{0,t}$  has the uniform distribution over the finite set  $\Psi_t$ . And if the sequence is purely periodic for all  $s_0$ ,  $\mathbf{u}_{i,t} = (u_i, \dots, u_{i+t-1})$  is also uniformly distributed over  $\Psi_t$  for all  $i \geq 0$ . Since the goal is to approximate the uniform distribution over  $(0, 1)^t$ , it immediately becomes apparent that  $\Psi_t$  should be evenly spread over this unit hypercube. In other words,  $\Psi_t$  *approximates*  $(0, 1)^t$  as the *sample space* from which the vectors of successive output values are drawn randomly, so it must be a good approximation of  $(0, 1)^t$  in some sense. The design of good-quality RNGs must therefore involve practical ways of measuring the uniformity of the corresponding sets  $\Psi_t$  even when they have huge cardinalities. In fact, a large state space  $\mathcal{S}$  is necessary to obtain a long period, but an even more important reason for having a huge number of states is to make sure that  $\Psi_t$  can be large enough to provide a good uniform coverage of the unit hypercube, at least for moderate values of  $t$ .

More generally, we may also want to measure the uniformity of sets of the form

$$\Psi_I = \{(u_{i_1}, \dots, u_{i_t}) \mid s_0 \in \mathcal{S}\},$$

where  $I = \{i_1, \dots, i_t\}$  is a fixed set of non-negative integers such that  $0 \leq i_1 < \dots < i_t$ . As a special case, we recover  $\Psi_t = \Psi_I$  when  $I = \{0, \dots, t-1\}$ . **Of course, there are so many such sets  $\Psi_I$  that we cannot examine the uniformity over all of them, but we can do it over a selected class  $\mathcal{J}$  of such sets deemed more important.**

The uniformity of a set  $\Psi_I$  is typically assessed by measuring the *discrepancy* between the empirical distribution of its points and the uniform distribution over  $(0, 1)^t$  (Niederreiter, 1992, L'Ecuyer and Lemieux, 2002, L'Ecuyer, 2009). Discrepancy measures are equivalent to goodness-of-fit test statistics for the multivariate uniform distribution. They can be defined in many different ways. The choice of a specific definition typically depends on the mathematical structure of the RNG to be studied and the reason for this is very pragmatic: we must be able to compute these measures quickly even when  $\mathcal{S}$  has very large cardinality, **for instance  $2^{200}$  or more.** This obviously excludes any method that requires explicit generation of the sequence over its entire period. The selected discrepancy measure is usually computed for each set  $I$  in a predefined class  $\mathcal{J}$ , these values are weighted or normalized by factors that depend on  $I$ , and the worst-case (or average) over  $\mathcal{J}$  is adopted as a *figure of merit* used to rank RNGs. The choice of  $\mathcal{J}$  and of the weights are arbitrary. Typically,

$\mathcal{J}$  would contain sets  $I$  such that  $t$  and  $i_t - i_1$  are rather small. Examples of such figures of merit will be given when we discuss specific classes of RNGs.

## 2.4 Statistical Testing

Good RNGs are designed based on mathematical analysis of their properties, then implemented and submitted to batteries of *empirical statistical tests*. These tests try to detect empirical evidence against the null hypothesis  $\mathcal{H}_0$ : “the  $u_i$  are realizations of i.i.d.  $U(0, 1)$  random variables.” A test can be defined by any function  $T$  that maps a sequence  $u_0, u_1, \dots$  in  $(0, 1)$  to a real number  $X$ , and for which a good approximation is available for the distribution of the random variable  $X$  under  $\mathcal{H}_0$ . For the test to be implementable,  $X$  must depend on only a finite (but perhaps random) number of  $u_i$ 's. Passing many tests may improve one's confidence in the RNG, but never guarantees that the RNG is foolproof for all kinds of simulations.

Building a RNG that *passes all statistical tests* is an impossible dream. Consider, for example, the class of all tests that examine the first (most significant)  $b$  bits of  $n$  successive output values,  $u_0, \dots, u_{n-1}$ , and return a binary value  $X \in \{0, 1\}$ . Select  $\alpha \in (0, 1)$  so that  $\alpha 2^{nb}$  is an integer and let  $\mathcal{T}_{n,b,\alpha}$  be the set of tests in this class that return  $X = 1$  for *exactly*  $\alpha 2^{nb}$  of the  $2^{nb}$  possible output sequences. We say that the sequence *fails the test* when  $X = 1$ . This  $\mathcal{T}_{n,b,\alpha}$  is the set of all statistical tests of (exact) level  $\alpha$ . Its cardinality is equal to the number of ways of choosing  $\alpha 2^{nb}$  distinct objects among  $2^{nb}$ . The chosen objects are the sequences that fail the test. For any given output sequence, the number of tests in  $\mathcal{T}_{n,b,\alpha}$  that return 1 for this sequence is equal to the number of ways of choosing the other  $\alpha 2^{nb} - 1$  sequences that also fail the test. This is the number of ways of choosing  $\alpha 2^{nb} - 1$  distinct objects among  $2^{nb} - 1$ . In other words, as pointed out by Leeb (1995), every output sequence fails exactly the same number of tests! Viewed from a different angle, it is a restatement of the well-known fact that under  $\mathcal{H}_0$ , each of the  $2^{nb}$  possible sequences has the same probability of occurring, so one may argue that none should be considered more random than any other (Knuth, 1998).

This viewpoint seems to lead into a dead end. For statistical testing to be meaningful, all tests should not be considered on equal footing. So which ones are more important? *Any answer is tainted with arbitrariness.* However, for large values of  $n$ , the number of tests is huge and all but a tiny fraction are too complicated even to be implemented. So we may say that *bad* RNGs are those that fail simple tests, whereas *good* RNGs fail only complicated tests that are hard to find and run. This common-sense compromise has been generally adopted in one way or another.

Experience shows that RNGs with very long periods, good structure of their set  $\Psi_t$ , and based on recurrences that are not too simplistic, pass most reasonable tests, whereas RNGs with short periods or bad structures are usually easy to crack by standard statistical tests. For sensitive applications, it is a good idea, when this is possible, to apply additional statistical tests designed in close relation with the random variable of interest (e.g., based on a *simplification* of the stochastic model being simulated, and for which the theoretical distribution can be computed).

Our discussion of statistical tests continues in Sect. 6. A key reference is L'Ecuyer and Simard (2007).

## 2.5 Cryptographically Strong Generators

One way of defining an ideal RNG would be that no statistical test can distinguish its output sequence from an i.i.d.  $U(0, 1)$  sequence. If an unlimited computing time is available, no finite-state RNG can satisfy this requirement, because by running it long enough one can eventually figure out its periodicity. But what if we impose a limit on the computing time? This can be analyzed formally in the framework of asymptotic *computational complexity* theory, under the familiar “rough-cut” assumption that polynomial-time algorithms are practical and others are not.

Consider a family of RNGs  $\{\mathcal{G}_k = (\mathcal{S}_k, \mu_k, f_k, \mathcal{U}_k, g_k), k = 1, 2, \dots\}$  where  $\mathcal{S}_k$  of cardinality  $2^k$  (i.e.,  $\mathcal{G}_k$  has a  $k$ -bit state). Suppose that the transition and output functions  $f$  and  $g$  can be computed in time bounded by a polynomial in  $k$ . Let  $\mathcal{T}$  be the class of statistical tests that run in time bounded by a polynomial in  $k$  and try to differentiate between the output sequence of the RNG and an i.i.d.  $U(0, 1)$  sequence. The RNG family is called *polynomial-time perfect* if there is a constant  $\epsilon > 0$  such that for all  $k$ , no test in  $\mathcal{T}$  can differentiate correctly

with probability larger than  $1/2 + e^{-k\epsilon}$ . This is equivalent to asking that no polynomial-time algorithm can predict any given bit of  $u_i$  with probability of success larger than  $1/2 + e^{-k\epsilon}$ , after observing  $u_0, \dots, u_{i-1}$ . This links unpredictability with statistical uniformity and independence. For the proofs and additional details, see, e.g. Blum et al. (1986), L'Ecuyer and Proulx (1989), Lagarias (1993), and Luby (1996). This theoretical framework has been used to define a notion of reliable RNG in the context of cryptography. But the guarantee is only asymptotic; it does not necessarily tell what value of  $k$  is large enough for the RNG to be secure in practice. Moreover, specific RNG families have been proved to be polynomial-time perfect only under yet unproven conjectures. So far, no one has been able to prove even their existence. Most RNGs discussed in the remainder of this chapter are known *not* to be polynomial-time perfect. However, they are fast, convenient, and have good enough statistical properties when their parameters are chosen carefully.

### 3 Linear Recurrences Modulo $m$

#### 3.1 The Multiple Recursive Generator

The most widely used RNGs are based on the linear recurrence

$$x_i = (a_1x_{i-1} + \dots + a_kx_{i-k}) \bmod m, \quad (2)$$

where  $m$  and  $k$  are positive integers called the *modulus* and the *order*, and the *coefficients*  $a_1, \dots, a_k$  are in  $\mathbb{Z}_m$ , interpreted as the set  $\{0, \dots, m-1\}$  on which all operations are performed with reduction modulo  $m$ . The *state* at step  $i$  is  $s_i = \mathbf{x}_i = (x_{i-k+1}, \dots, x_i)^\top$ . When  $m$  is a prime number, the finite ring  $\mathbb{Z}_m$  is a finite field and it is possible to choose the coefficients  $a_j$  so that the period reaches  $\rho = m^k - 1$  (the largest possible value) (Knuth, 1998). This maximal period is achieved if and only if the characteristic polynomial of the recurrence,  $P(z) = z^k - a_1z^{k-1} - \dots - a_k$ , is a primitive polynomial over  $\mathbb{Z}_m$ , i.e., if and only if the smallest positive integer  $\nu$  such that  $(z^\nu \bmod P(z)) \bmod m = 1$  is  $\nu = m^k - 1$ . Knuth (1998) explains how to verify this for a given  $P(z)$ . For  $k > 1$ , for  $P(z)$  to be a primitive polynomial, it is necessary that  $a_k$  and at least another coefficient  $a_j$  be nonzero. Finding primitive polynomials of this form is generally easy and they yield the simplified recurrence:

$$x_n = (a_r x_{n-r} + a_k x_{n-k}) \bmod m. \quad (3)$$

A *multiple recursive generator* (MRG) uses (2) with a large value of  $m$  and defines the output as  $u_i = x_i/m$ . For  $k = 1$ , this is the classical *linear congruential generator* (LCG). In practice, the output function is modified slightly to make sure that  $u_i$  never takes the value 0 or 1 (e.g., one may define  $u_i = (x_i + 1)/(m + 1)$ , or  $u_i = x_i/(m + 1)$  if  $x_i > 0$  and  $u_i = m/(m + 1)$  otherwise) but to simplify the theoretical analysis, we will follow the common convention of assuming that  $u_i = x_i/m$  (in which case  $u_i$  *does* take the value 0 occasionally).

#### 3.2 The Lattice Structure

Let  $\mathbf{e}_i$  denote the  $i$ th unit vector in  $k$  dimensions, with a 1 in position  $i$  and 0's elsewhere. Denote by  $x_{i,0}, x_{i,1}, x_{i,2}, \dots$  the values of  $x_0, x_1, x_2, \dots$  produced by the recurrence (2) when the initial state  $\mathbf{x}_0$  is  $\mathbf{e}_i$ . An arbitrary initial state  $\mathbf{x}_0 = (z_1, \dots, z_k)^\top$  can be written as the linear combination  $z_1\mathbf{e}_1 + \dots + z_k\mathbf{e}_k$  and the corresponding sequence is a linear combination of the sequences  $(x_{i,0}, x_{i,1}, \dots)$ , with reduction of the coordinates modulo  $m$ . Conversely, any such linear combination reduced modulo  $m$  is a sequence that can be obtained from some initial state  $\mathbf{x}_0 \in \mathcal{S} = \mathbb{Z}_m^k$ . If we divide everything by  $m$  we find that for the MRG, for each  $t \geq 1$ ,  $\Psi_t = L_t \cap [0, 1)^t$  where

$$L_t = \left\{ \mathbf{v} = \sum_{i=1}^t z_i \mathbf{v}_i \mid z_i \in \mathbb{Z} \right\},$$

is a  $t$ -dimensional *lattice* in  $\mathbb{R}^t$ , with basis

$$\begin{aligned}
\mathbf{v}_1 &= (1, 0, \dots, 0, x_{1,k}, \dots, x_{1,t-1})^\top / m \\
&\vdots \\
\mathbf{v}_k &= (0, 0, \dots, 1, x_{k,k}, \dots, x_{k,t-1})^\top / m \\
\mathbf{v}_{k+1} &= (0, 0, \dots, 0, 1, \dots, 0)^\top \\
&\vdots \\
\mathbf{v}_t &= (0, 0, \dots, 0, 0, \dots, 1)^\top.
\end{aligned}$$

For  $t \leq k$ ,  $L_t$  contains all vectors whose coordinates are multiples of  $1/m$ . For  $t > k$ , it contains a fraction  $m^{k-t}$  of those vectors.

This lattice structure implies that the points of  $\Psi_t$  are distributed according to a very regular pattern, in equidistant parallel hyperplanes. Graphical illustrations of this, usually for LCGs, can be found in a myriad of papers and books; e.g., Gentle (2003), Law and Kelton (2000), and L'Ecuyer (1998). Define the *dual lattice* to  $L_t$  as

$$L_t^* = \{\mathbf{h} \in \mathbb{R}^t : \mathbf{h}^\top \mathbf{v} \in \mathbb{Z} \text{ for all } \mathbf{v} \in L_t\}.$$

Each  $\mathbf{h} \in L_t^*$  is a normal vector that defines a family of equidistant parallel hyperplanes, at distance  $1/\|\mathbf{h}\|_2$  apart, and these hyperplanes cover all the points of  $L_t$  unless  $\mathbf{h}$  is an integer multiple of some other vector  $\mathbf{h}' \in L_t^*$ . Therefore, if  $\ell_t$  is the Euclidean length of a shortest non-zero vector  $\mathbf{h}$  in  $L_t^*$ , then there is a family of hyperplanes at distance  $1/\ell_t$  apart that cover all the points of  $L_t$ . A small  $\ell_t$  means there are thick slices of empty space between the hyperplanes and we want to avoid that. A large  $\ell_t$  means a better (more uniform) coverage of the unit hypercube by the point set  $\Psi_t$ . Computing the value of  $1/\ell_t$  is often called the *spectral test* (Knuth, 1998, Fishman, 1996).

The lattice property holds as well for the point sets  $\Psi_I$  formed by values at arbitrary lags defined by a fixed set of indices  $I = \{i_1, \dots, i_t\}$ . One has  $\Psi_I = L_I \cap [0, 1)^t$  for some lattice  $L_I$ , and the largest distance between successive hyperplanes for a family of hyperplanes that cover all the points of  $L_I$  is  $1/\ell_I$ , where  $\ell_I$  is the Euclidean length of a shortest nonzero vector in  $L_I^*$ , the dual lattice to  $L_I$ .

The lattice  $L_I$  and its dual can be constructed as explained in Couture and L'Ecuyer (1996) and L'Ecuyer and Couture (1997). Finding the shortest nonzero vector in a lattice with basis  $\mathbf{v}_1, \dots, \mathbf{v}_t$  can be formulated as an integer programming problem with a quadratic objective function:

$$\text{Minimize } \|\mathbf{v}\|^2 = \sum_{i=1}^t \sum_{j=1}^t z_i \mathbf{v}_i^\top \mathbf{v}_j z_j$$

subject to  $z_1, \dots, z_t$  integers and not all zero. This problem can be solved by a branch-and-bound algorithm (Fincke and Pohst, 1985, L'Ecuyer and Couture, 1997, Tezuka, 1995).

For any given dimension  $t$  and  $m^k$  points per unit of volume, there is an absolute upper bound on the best possible value of  $\ell_I$  (Conway and Sloane, 1999, Knuth, 1998, L'Ecuyer, 1999b). Let  $\ell_t^*(m^k)$  denote such an upper bound. To define a figure of merit that takes into account several sets  $I$ , in different numbers of dimensions, it is common practice to divide  $\ell_I$  by an upper bound, to obtain a standardized value between 0 and 1, and then take the worst case over a given class  $\mathcal{J}$  of sets  $I$ . This gives a figure of merit of the form

$$M_{\mathcal{J}} = \min_{I \in \mathcal{J}} \ell_I / \ell_{|I|}^*(m^k).$$

A value of  $M_{\mathcal{J}}$  too close to zero means that  $L_I$  has a bad lattice structure for at least one of the selected sets  $I$ . We want a value as close to 1 as possible. Computer searches for good MRGs with respect to this criterion have been reported by L'Ecuyer et al. (1993), L'Ecuyer and Andres (1997), L'Ecuyer (1999a), for example. In most cases,  $\mathcal{J}$  was simply the sets of the form  $I = \{1, \dots, t\}$  for  $t \leq t_1$ , where  $t_1$  was an arbitrary integer ranging from 8 to 45. L'Ecuyer and Lemieux (2000) also consider the small dimensional sets  $I$  with indices not too far apart. They suggest taking  $\mathcal{J} = \{\{0, 1, \dots, i\} : i < t_1\} \cup \{\{i_1, i_2\} : 0 = i_1 < i_2 < t_2\} \cup \dots \cup \{\{i_1, \dots, i_d\} : 0 = i_1 < \dots < i_d < t_d\}$  for some positive integers  $d, t_1, \dots, t_d$ . We could also take a weighted average instead of the minimum in the definition of  $M_{\mathcal{J}}$ .

An important observation is that for  $t > k$ , the  $t$ -dimensional vector  $\mathbf{h} = (-1, a_1, \dots, a_k, 0, \dots, 0)^\top$  always belong to  $L_t^*$ , because for any vector  $\mathbf{v} \in L_t$ , the first  $k+1$  coordinates of  $m\mathbf{v}$  must satisfy the recurrence (2),

which implies that  $(-1, a_1, \dots, a_k, 0, \dots, 0)\mathbf{v}$  must be an integer. Therefore, one always has  $\ell_i^2 \leq 1 + a_1^2 + \dots + a_k^2$ . Likewise, if  $I$  contains 0 and all indices  $j$  such that  $a_{k-j} \neq 0$ , then  $\ell_I^2 \leq 1 + a_1^2 + \dots + a_k^2$  (L'Ecuyer, 1997). This means that the sum of squares of the coefficients  $a_j$  *must be large* if we want to have any chance that the lattice structure be good.

Constructing MRGs with only two nonzero coefficients and taking these coefficients small has been a very popular idea, because this makes the implementation easier and faster (Deng and Lin, 2000, Knuth, 1998). However, the MRGs thus obtained have a bad structure. As a worst-case illustration, consider the widely-available additive or subtractive *lagged-Fibonacci* generator, based on the recurrence (2) where the two coefficients  $a_r$  and  $a_k$  are both equal to  $\pm 1$ . In this case, whenever  $I$  contains  $\{0, k-r, k\}$ , one has  $\ell_I^2 \leq 3$ , so the distance between the hyperplanes is at least  $1/\sqrt{3}$ . In particular, for  $I = \{0, k-r, k\}$ , all the points of  $\Psi_I$  (aside from the zero vector) are contained in only two planes! This type of structure can have a dramatic effect on certain simulation problems and is a good reason for staying away from these lagged-Fibonacci generators, regardless of their parameters. **They fail several simple empirical statistical tests (L'Ecuyer and Simard, 2007).**

A similar problem occurs for the “fast MRG” proposed by Deng and Lin (2000), based on the recurrence

$$x_i = (-x_{i-1} + ax_{i-k}) \bmod m = ((m-1)x_{i-1} + ax_{i-k}) \bmod m,$$

with  $a^2 < m$ . If  $a$  is small, the bound  $\ell_I^2 \leq 1 + a^2$  implies a bad lattice structure for  $I = \{0, k-1, k\}$ . A more detailed analysis by L'Ecuyer and Touzin (2004) shows that this type of generator cannot have a good lattice structure even if the condition  $a^2 < m$  is removed. Another special case proposed by Deng and Xu (2003) has the form

$$x_i = a(x_{i-j_2} + \dots + x_{i-j_t}) \bmod m. \quad (4)$$

In this case, for  $I = \{0, k-j_{t-1}, \dots, k-j_2, k\}$ , the vectors  $(1, a, \dots, a)$  and  $(a^*, 1, \dots, 1)$  both belong to the dual lattice  $L_I^*$ , where  $a^*$  is the multiplicative inverse of  $a$  modulo  $m$ . So neither  $a$  nor  $a^*$  should be small.

To get around this structural problem when  $I$  contains certain sets of indices, Lüscher (1994) and Knuth (1998) recommend to skip some of the output values to break up the bad vectors. For the lagged-Fibonacci generator, for example, one can output  $k$  successive values produced by the recurrence, then skip the next  $d$  values, output the next  $k$ , skip the next  $d$ , and so on. A large value of  $d$  (e.g.,  $d = 5k$  or more) may get rid of the bad structure, but slows down the generator. See Wegenkittl and Matsumoto (1999) for further discussion.

### 3.3 MRG Implementation Techniques

The modulus  $m$  is often taken as a large prime number close to the largest integer directly representable on the computer (e.g., equal or near  $2^{31} - 1$  for 32-bit computers). Since each  $x_{i-j}$  can be as large as  $m-1$ , one must be careful in computing the right side of (2) because the product  $a_j x_{i-j}$  is typically not representable as an ordinary integer. Various techniques for computing this product modulo  $m$  are discussed and compared by Fishman (1996), L'Ecuyer and Tezuka (1991), L'Ecuyer (1999a), and L'Ecuyer and Simard (1999). Note that if  $a_j = m - a'_j > 0$ , using  $a_j$  is equivalent to using the negative coefficient  $-a'_j$ , which is sometimes more convenient from the implementation viewpoint. In what follows, we assume that  $a_j$  can be either positive or negative.

One approach is to perform the arithmetic modulo  $m$  in 64-bit (double precision) floating-point arithmetic (L'Ecuyer, 1999a). Under this representation, assuming that the usual IEEE floating-point standard is respected, all positive integers up to  $2^{53}$  are represented exactly. Then, if each coefficient  $a_j$  is selected to satisfy  $|a_j|(m-1) \leq 2^{53}$ , the product  $|a_j|x_{i-j}$  will always be represented exactly and  $z_j = |a_j|x_{i-j} \bmod m$  can be computed by the instructions

$$y = |a_j|x_{i-j}; \quad z_j = y - m[y/m].$$

Similarly, if  $(|a_1| + \dots + |a_k|)(m-1) \leq 2^{53}$ ,  $a_1 x_{i-1} + \dots + a_k x_{i-k}$  will always be represented exactly.

A second technique, called *approximate factoring* (L'Ecuyer and Côté, 1991), uses only the integer representation and works under the condition that  $|a_j| = i$  or  $|a_j| = \lfloor m/i \rfloor$  for some integer  $i < \sqrt{m}$ . One precomputes  $q_j = \lfloor m/|a_j| \rfloor$  and  $r_j = m \bmod |a_j|$ . Then,  $z_j = |a_j|x_{i-j} \bmod m$  can be computed by

$$y = \lfloor x_{i-j}/q_j \rfloor; \quad z = |a_j|(x_{i-j} - yq_j) - yr_j;$$



if  $z < 0$  then  $z_j = z + m$  else  $z_j = z$ .

All quantities involved in these computations are integers between  $-m$  and  $m$ , so no overflow can occur if  $m$  can be represented as an ordinary integer (e.g.,  $m < 2^{31}$  on a 32-bit computer).

The *powers-of-two decomposition* approach selects coefficients  $a_j$  that can be written as a sum or difference of a small number of powers of 2 (Wu, 1997, L'Ecuyer and Simard, 1999, L'Ecuyer and Touzin, 2000). For example, one may take  $a_j = \pm 2^q \pm 2^r$  and  $m = 2^e - h$  for some positive integers  $q, r, e$ , and  $h$ . To compute  $y = 2^q x \bmod m$ , decompose  $x = z_0 + 2^{e-q} z_1$  (where  $z_0 = x \bmod 2^{e-q}$ ) and observe that

$$y = 2^q(z_0 + 2^{e-q} z_1) \bmod (2^e - h) = (2^q z_0 + h z_1) \bmod (2^e - h).$$

Suppose now that

$$h < 2^q \quad \text{and} \quad h(2^q - (h+1)2^{-e+q}) < m. \quad (5)$$

Then,  $2^q z_0 < m$  and  $h z_1 < m$ , so  $y$  can be computed by shifts, masks, additions, subtractions, and a single multiplication by  $h$ . Intermediate results never exceed  $2m - 1$ . Things simplify further if  $q = 0$  or  $q = 1$  or  $h = 1$ . For  $h = 1$ ,  $y$  is obtained simply by swapping the blocks of bits  $z_0$  and  $z_1$  (Wu, 1997). L'Ecuyer and Simard (1999) pointed out that LCGs with parameters of the form  $m = 2^e - 1$  and  $a = \pm 2^q \pm 2^r$  have bad statistical properties because the recurrence does not “mix the bits” well enough. However, good and fast (combined) MRGs can be obtained via the power-of-two decomposition method, as explained in L'Ecuyer and Touzin (2000).

Another idea to improve efficiency is to take all nonzero coefficients  $a_j$  equal to the same  $a$ , as in (4) (Marsaglia, 1996, Deng and Xu, 2003). Then, computing the right side of (2) requires a single multiplication. [Deng and Xu \(2003\) and Deng \(2005\) provide specific parameter sets and concrete implementations for MRGs of this type, for prime  \$m\$  near  \$2^{31}\$ , and for  \$k\$  ranging from 102 to 1597.](#)

One may be tempted to take  $m$  equal to a power of two, say  $m = 2^e$ , because then the “ $\bmod m$ ” operation is much easier: it suffices to keep the  $e$  least significant bits and mask-out all others. However, taking a power-of-two modulus is not recommended because it has several strong disadvantages in terms of the quality of the RNG (L'Ecuyer, 1990, 1998). In particular, the least significant bits have very short periodicity and the period of the recurrence (2) cannot exceed  $(2^k - 1)2^{e-1}$  if  $k > 1$ , and  $2^{e-2}$  if  $k = 1$  and  $e \geq 4$ . The maximal period achievable with  $k = 7$  and  $m = 2^{31}$ , for example, is more than  $2^{180}$  times smaller than the maximal period achievable with  $k = 7$  and  $m = 2^{31} - 1$  (a prime number).

### 3.4 Combined MRGs and LCGs

The conditions that make MRG implementations run faster (e.g., only two nonzero coefficients both close to zero) conflict with those required for having a good lattice structure and statistical robustness. *Combined MRGs* are one solution to this problem. Consider  $J$  distinct MRGs evolving in parallel, based on the recurrences

$$x_{j,i} = (a_{j,1}x_{j,i-1} + \cdots + a_{j,k}x_{j,i-k}) \bmod m_j \quad (6)$$

where  $a_{j,k} \neq 0$ , for  $j = 1, \dots, J$ . Let  $\delta_1, \dots, \delta_J$  be arbitrary integers,

$$z_i = (\delta_1 x_{1,i} + \cdots + \delta_J x_{J,i}) \bmod m_1, \quad u_i = z_i / m_1, \quad (7)$$

and

$$w_i = (\delta_1 x_{1,i} / m_1 + \cdots + \delta_J x_{J,i} / m_J) \bmod 1. \quad (8)$$

This defines two RNGs, with output sequences  $\{u_i, i \geq 0\}$  and  $\{w_i, i \geq 0\}$ .

Suppose that the  $m_j$  are pairwise relatively prime, that  $\delta_j$  and  $m_j$  have no common factor for each  $j$ , and that each recurrence (6) is purely periodic with period  $\rho_j$ . Let  $m = m_1 \cdots m_J$  and let  $\rho$  be the least common multiple of  $\rho_1, \dots, \rho_J$ . [Under these conditions, L'Ecuyer and Tezuka \(1991\) and L'Ecuyer \(1996a\) proved the following:](#) (a) the sequence (8) is exactly equivalent to the output sequence of a MRG with (composite) modulus  $m$  and coefficients  $a_j$  that can be computed explicitly as explained by L'Ecuyer (1996a); (b) the two sequences in (7) and (8) have period  $\rho$ ; and (c) if both sequences have the same initial state, then  $u_i = w_i + \epsilon_i$



where  $\max_{i \geq 0} |\epsilon_i|$  can be bounded explicitly by a constant  $\epsilon$  which is very small when the  $m_j$  are close to each other.

Thus, these combined MRGs can be viewed as practical ways of implementing an MRG with a large  $m$  and several large nonzero coefficients. The idea is to cleverly select the components so that: (1) each one is easy to implement efficiently (e.g., has only two small nonzero coefficients) and (2) the MRG that corresponds to the combination has a good lattice structure. If each  $m_j$  is prime and if each component  $j$  has maximal period  $\rho_j = m_j^k - 1$ , then each  $\rho_j$  is even and  $\rho$  cannot exceed  $\rho_1 \cdots \rho_J / 2^{J-1}$ . Tables of good parameters for combined MRGs of different sizes that reach this upper bound are given in L'Ecuyer (1999a) and L'Ecuyer and Touzin (2000), together with C implementations.

### 3.5 Jumping Ahead

The recurrence (2) can be written in matrix form as

$$\mathbf{x}_i = \mathbf{A}\mathbf{x}_{i-1} \bmod m = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix} \mathbf{x}_{i-1} \bmod m.$$

To jump ahead directly from  $\mathbf{x}_i$  to  $\mathbf{x}_{i+\nu}$ , for an arbitrary integer  $\nu$ , it suffices to exploit the relationship

$$\mathbf{x}_{i+\nu} = \mathbf{A}^\nu \mathbf{x}_i \bmod m = (\mathbf{A}^\nu \bmod m) \mathbf{x}_i \bmod m.$$

If this is to be done several times for the same  $\nu$ , the matrix  $\mathbf{A}^\nu \bmod m$  can be precomputed once for all. For a large  $\nu$ , this can be done in  $O(\log_2 \nu)$  matrix multiplications via a standard divide-and-conquer algorithm (Knuth, 1998):

$$\mathbf{A}^\nu \bmod m = \begin{cases} (\mathbf{A}^{\nu/2} \bmod m)(\mathbf{A}^{\nu/2} \bmod m) \bmod m & \text{if } \nu \text{ is even;} \\ \mathbf{A}(\mathbf{A}^{\nu-1} \bmod m) \bmod m & \text{if } \nu \text{ is odd.} \end{cases}$$

### 3.6 Linear Recurrences With Carry

These types of recurrences were introduced by Marsaglia and Zaman (1991) to obtain a large period even when  $m$  is a power of two (in which case the implementation may be faster). They were studied and generalized by Tezuka et al. (1994), Couture and L'Ecuyer (1994, 1997), and Goresky and Klapper (2003). The basic idea is to add a *carry* to the linear recurrence (2). The general form of this RNG, called *multiply-with-carry* (MWC), can be written as

$$x_i = (a_1 x_{i-1} + \cdots + a_k x_{i-k} + c_{i-1})d \bmod b, \quad (9)$$

$$c_i = \lfloor (a_0 x_i + a_1 x_{i-1} + \cdots + a_k x_{i-k} + c_{i-1})/b \rfloor, \quad (10)$$

$$u_i = \sum_{\ell=1}^{\infty} x_{i-\ell+1} b^{-\ell}, \quad (11)$$

where  $b$  is a positive integer (e.g., a power of two),  $a_0, \dots, a_k$  are arbitrary integers such that  $a_0$  is relatively prime to  $b$ , and  $d$  is the multiplicative inverse of  $-a_0$  modulo  $b$ . The state at step  $i$  is  $s_i = (x_{i-k+1}, \dots, x_i, c_i)^\top$ . In practice, the sum in (11) is truncated to a few terms (it could be a single term if  $b$  is large), but the theoretical analysis is much easier for the infinite sum.

Define  $m = \sum_{\ell=0}^k a_\ell b^\ell$  and let  $a$  be the inverse of  $b$  in arithmetic modulo  $m$ , assuming for now that  $m > 0$ . A major result proved in Tezuka et al. (1994), Couture and L'Ecuyer (1997), and Goresky and Klapper (2003) is that if the initial states agree, the output sequence  $\{u_i, i \geq 0\}$  is exactly the same as that produced by the LCG with modulus  $m$  and multiplier  $a$ . Therefore, the MWC can be seen as a clever way of implementing a LCG with very large modulus. Couture and L'Ecuyer (1997) have shown that the value of  $\ell_t$  for this LCG

satisfies  $\ell_t^2 \leq a_0^2 + \dots + a_k^2$  for  $t \geq k$ , which means that the lattice structure will be bad unless the sum of squares of coefficients  $a_j$  is large.

In the original proposals of Marsaglia and Zaman (1991), called *add-with-carry* and *subtract-with-borrow*, one has  $-a_0 = \pm a_r = \pm a_k = 1$  for some  $r < k$  and the other coefficients  $a_j$  are zero, so  $\ell_t^2 \leq 3$  for  $t \geq k$  and the generator has essentially the same structural defect as the additive lagged-Fibonacci generator. In the version studied by Couture and L'Ecuyer (1997), it was assumed that  $-a_0 = d = 1$ . Then, the period cannot exceed  $(m-1)/2$  if  $b$  is a power of two. A concrete implementation was given in that paper. Goresky and Klapper (2003) pointed out that the maximal period of  $\rho = m-1$  can be achieved by allowing a more general  $a_0$ . They provided specific parameters that give the maximal period for  $b$  ranging from  $2^{21}$  to  $2^{35}$  and  $\rho$  up to approximately  $2^{2521}$ .

## 4 Generators Based on Recurrences Modulo 2

### 4.1 A General Framework

It seems natural to exploit the fact that computers work in binary arithmetic and to design RNGs defined directly in terms of bit strings and sequences. We do this under the following framework, taken from L'Ecuyer and Panneton (2002) and L'Ecuyer and Panneton (2009). Let  $\mathbb{F}_2$  denote the finite field with two elements, 0 and 1, in which the operations are equivalent to addition and multiplication modulo 2. Consider the RNG defined by a matrix linear recurrence over  $\mathbb{F}_2$ , as follows:

$$\mathbf{x}_i = \mathbf{A}\mathbf{x}_{i-1}, \quad (12)$$

$$\mathbf{y}_i = \mathbf{B}\mathbf{x}_i, \quad (13)$$

$$u_i = \sum_{\ell=1}^w y_{i,\ell-1} 2^{-\ell} = .y_{i,0} y_{i,1} y_{i,2} \dots, \quad (14)$$

where  $\mathbf{x}_i = (x_{i,0}, \dots, x_{i,k-1})^\top \in \mathbb{F}_2^k$  is the  $k$ -bit *state vector* at step  $i$ ,  $\mathbf{y}_i = (y_{i,0}, \dots, y_{i,w-1})^\top \in \mathbb{F}_2^w$  is the  $w$ -bit *output vector* at step  $i$ ,  $k$  and  $w$  are positive integers,  $\mathbf{A}$  is a  $k \times k$  *transition matrix* with elements in  $\mathbb{F}_2$ ,  $\mathbf{B}$  is a  $w \times k$  *output transformation matrix* with elements in  $\mathbb{F}_2$ , and  $u_i \in [0, 1)$  is the *output* at step  $i$ . All operations in (12) and (13) are performed in  $\mathbb{F}_2$ .

It is well-known (Niederreiter, 1992, L'Ecuyer, 1994) that when the  $\mathbf{x}_i$ 's obey (12), for each  $j$ , the sequence  $\{x_{i,j}, i \geq 0\}$  follows the linear recurrence

$$x_{i,j} = (\alpha_1 x_{i-1,j} + \dots + \alpha_k x_{i-k,j}) \bmod 2, \quad (15)$$

whose *characteristic polynomial*  $P(z)$  is the characteristic polynomial of  $\mathbf{A}$ , i.e.,

$$P(z) = \det(\mathbf{A} - z\mathbf{I}) = z^k - \alpha_1 z^{k-1} - \dots - \alpha_{k-1} z - \alpha_k,$$

where  $\mathbf{I}$  is the identity matrix and each  $\alpha_j$  is in  $\mathbb{F}_2$ . The sequences  $\{y_{i,j}, i \geq 0\}$ , for  $0 \leq j < w$ , also obey the same recurrence (although some of them may follow recurrences of shorter order as well, depending on  $\mathbf{B}$ ). We assume that  $\alpha_k = 1$ , so that the recurrence (15) has *order*  $k$  and is purely periodic. Its period is  $2^k - 1$  (i.e., maximal) if and only if  $P(z)$  is a primitive polynomial over  $\mathbb{F}_2$  (Niederreiter, 1992, Knuth, 1998).

To jump ahead directly from  $\mathbf{x}_i$  to  $\mathbf{x}_{i+\nu}$  with this type of generator, it suffices to precompute the matrix  $\mathbf{A}^\nu$  (in  $\mathbb{F}_2$ ) and then multiply  $\mathbf{x}_i$  by this matrix.

Several popular classes of RNGs fit this framework as special cases, by appropriate choices of the matrices  $\mathbf{A}$  and  $\mathbf{B}$ . This includes the Tausworthe or LFSR, polynomial LCG, GFSR, twisted GFSR, Mersenne twister, **WELL**, **xorshift**, multiple recursive matrix generators, and combinations of these (L'Ecuyer and Panneton, 2009, Matsumoto and Nishimura, 1998, Niederreiter, 1995, Panneton and L'Ecuyer, 2005, Panneton et al., 2006, Tezuka, 1995). We detail some of them after discussing measures of uniformity.

## 4.2 Measures of Uniformity

The uniformity of point sets  $\Psi_I$  produced by RNGs based on linear recurrences over  $\mathbb{F}_2$  is usually assessed by measures of equidistribution defined as follows (L'Ecuyer, 1996b, L'Ecuyer and Panneton, 2002, L'Ecuyer, 2004, L'Ecuyer and Panneton, 2009, Tezuka, 1995). For an arbitrary vector  $\mathbf{q} = (q_1, \dots, q_t)$  of non-negative integers, partition the unit hypercube  $[0, 1)^t$  into  $2^{q_j}$  intervals of the same length along axis  $j$ , for each  $j$ . This determines a partition of  $[0, 1)^t$  into  $2^{q_1 + \dots + q_t}$  rectangular boxes of the same size and shape. We call this partition the  $\mathbf{q}$ -equidissection of the unit hypercube.

For some index set  $I = \{i_1, \dots, i_t\}$ , if  $\Psi_I$  has  $2^k$  points, we say that  $\Psi_I$  is  $\mathbf{q}$ -*equidistributed in base 2* if there are exactly  $2^q$  points in each box of the  $\mathbf{q}$ -equidissection, where  $k - q = q_1 + \dots + q_t$ . This means that among the  $2^k$  points  $(x_{j_1}, \dots, x_{j_t})$  of  $\Psi_I$ , if we consider the first  $q_1$  bits of  $x_{j_1}$ , the first  $q_2$  bits of  $x_{j_2}, \dots$ , and the first  $q_t$  bits of  $x_{j_t}$ , each of the  $2^{k-q}$  possibilities occurs exactly the same number of times. This is possible only if  $q \leq k$ .

The  $\mathbf{q}$ -equidistribution of  $\Psi_I$  depends only on the first  $q_j$  bits of  $x_{i_j}$  for  $1 \leq j \leq t$ , for the points  $(x_{i_1}, \dots, x_{i_t})$  that belong to  $\Psi_I$ . The vector of these  $q_1 + \dots + q_t = k - q$  bits can always be expressed as a linear function of the  $k$  bits of the initial state  $\mathbf{x}_0$ , i.e., as  $\mathbf{M}_{\mathbf{q}}\mathbf{x}_0$  for some  $(k - q) \times k$  binary matrix  $\mathbf{M}_{\mathbf{q}}$ , and it is easily seen that  $\Psi_I$  is  $\mathbf{q}$ -equidistributed if and only if  $\mathbf{M}_{\mathbf{q}}$  has full rank  $k - q$ . This provides an easy way of checking equidistribution (L'Ecuyer, 1996b, L'Ecuyer and Panneton, 2009, Tezuka, 1995).

If  $\Psi_I$  is  $(\ell, \dots, \ell)$ -equidistributed for some  $\ell \geq 1$ , it is called *t-distributed with  $\ell$  bits of accuracy*, or *(t,  $\ell$ )-equidistributed* (L'Ecuyer, 1996b). The largest value of  $\ell$  for which this holds is called the *resolution* of the set  $\Psi_I$  and is denoted by  $\ell_I$ . This value has the upper bound  $\ell_t^* = \min(\lfloor k/t \rfloor, w)$ . The *resolution gap* of  $\Psi_I$  is defined as  $\delta_I = \ell_t^* - \ell_I$ . In the same vein as for MRGs, a worst-case figure of merit can be defined here by

$$\Delta_{\mathcal{J}} = \max_{I \in \mathcal{J}} \delta_I,$$

where  $\mathcal{J}$  is a preselected class of index sets  $I$ .

The point set  $\Psi_I$  is a  $(q, k, t)$ -*net in base 2* (often called a  $(t, m, s)$ -net in the context of quasi-Monte Carlo methods, where a different notation is used (Niederreiter, 1992)), if it is  $(q_1, \dots, q_t)$ -equidistributed in base 2 for all non-negative integers  $q_1, \dots, q_t$  summing to  $k - q$ . We call the smallest such  $q$  the *q-value* of  $\Psi_I$ . The smaller it is, the better. One candidate for a figure of merit could be the *q-value* of  $\Psi_t$  for some large  $t$ . **Although widely used to construct and evaluate low-discrepancy point sets for quasi-Monte Carlo methods**, a major drawback of this measure is that it is **too costly** to compute for good long-period generators (for which  $k - q$  is large), because there are too many vectors  $\mathbf{q}$  for which equidistribution needs to be checked. In practice, one must settle for figures of merit that involve a smaller number of equidissections.

When  $\delta_I = 0$  for all sets  $I$  of the form  $I = \{0, \dots, t - 1\}$ , for  $1 \leq t \leq k$ , the RNG is said to be *maximally equidistributed* or *asymptotically random* for the word size  $w$  (L'Ecuyer, 1996b, Tezuka, 1995, Tootill et al., 1973). This property ensures perfect equidistribution of all sets  $\Psi_t$ , for any partition of the unit hypercube into subcubes of equal sizes, as long as  $\ell \leq w$  and the number of subcubes does not exceed the number of points in  $\Psi_t$ . Large-period maximally equidistributed generators, together with their implementations, can be found in L'Ecuyer (1999c), L'Ecuyer and Panneton (2002), Panneton and L'Ecuyer (2004), and Panneton et al. (2006), for example.

## 4.3 Lattice Structure in Spaces of Polynomials and Formal Series

The RNGs defined via (12)–(14) do not have a lattice structure in the real space like MRGs, but they do have a lattice structure in a space of formal series, as explained in Couture and L'Ecuyer (2000), L'Ecuyer (2004), L'Ecuyer and Panneton (2009), Lemieux and L'Ecuyer (2003), and Tezuka (1995). The real space  $\mathbb{R}$  is replaced by the space  $\mathbb{L}_2$  of formal power series with coefficients in  $\mathbb{F}_2$ , of the form  $\sum_{\ell=\omega}^{\infty} x_{\ell} z^{-\ell}$  for some integer  $\omega$ . In that setting, the lattices have the form

$$\mathcal{L}_t = \left\{ \mathbf{v}(z) = \sum_{j=1}^t h_j(z) \mathbf{v}_j(z) \text{ such that each } h_j(z) \in \mathbb{F}_2[z] \right\},$$



for some positive integers  $p$  and  $q$ , where  $\mathbf{I}_p$  is the  $p \times p$  identity matrix,  $\mathbf{S}$  is a  $p \times p$  matrix, and the matrix  $\mathbf{I}_p$  on the first line is in columns  $(r-1)p+1$  to  $rp$  for some positive integer  $r$ . Often,  $w = p$  and  $\mathbf{B}$  contains the first  $w$  lines of the  $pq \times pq$  identity matrix. If  $\mathbf{S}$  is also the identity matrix, the generator thus obtained is the trinomial-based *generalized feedback shift register* (GFSR), for which  $\mathbf{x}_i$  is obtained by a bitwise exclusive-or of  $\mathbf{x}_{i-r}$  and  $\mathbf{x}_{i-q}$ . This gives a very fast RNG, but its period cannot exceed  $2^q - 1$ , because each bit of  $\mathbf{x}_i$  follows the same binary recurrence of order  $k = q$ , with characteristic polynomial  $P(z) = z^q - z^{q-r} - 1$ . **It also fails several simple empirical tests (L'Ecuyer and Simard, 2007).**

More generally, we can define  $\mathbf{x}_i$  as the bitwise exclusive-or of  $\mathbf{x}_{i-r_1}, \mathbf{x}_{i-r_2}, \dots, \mathbf{x}_{i-r_d}$  where  $r_d = q$ , so that each bit of  $\mathbf{x}_i$  follows a recurrence in  $\mathbb{F}_2$  whose characteristic polynomial  $P(z)$  has  $d+1$  nonzero terms. However, the period is still bounded by  $2^q - 1$ , whereas considering the  $pq$ -bit state, we should rather expect a period close to  $2^{pq}$ . This was the main motivation for the *twisted GFSR* (TGFSR) generator. In the original version introduced by Matsumoto and Kurita (1992),  $w = p$  and the matrix  $\mathbf{S}$  is defined as the transpose of  $\mathbf{A}_0$  in (16), with  $k$  replaced by  $p$ . The characteristic polynomial of  $\mathbf{A}$  is then  $P(z) = P_S(z^q + z^m)$ , where  $P_S(z) = z^p - a_p z^{p-1} - \dots - a_1$  is the characteristic polynomial of  $S$ , and its degree is  $k = pq$ . If the parameters are selected so that  $P(z)$  is primitive over  $\mathbb{F}_2$ , then the TGFSR has period  $2^k - 1$ . Matsumoto and Kurita (1994) pointed out important weaknesses of the original TGFSR and proposed an improved version that uses a well-chosen matrix  $\mathbf{B}$  whose lines differ from those of the identity. The operations implemented by this matrix are called *tempering* and their purpose is to improve the uniformity of the points produced by the RNG.

The *Mersenne twister* (Matsumoto and Nishimura, 1998, Nishimura, 2000) is a variant of the TGFSR where  $k$  is slightly less than  $pq$  and can be a prime number. **A specific instance named MT19937, proposed by Matsumoto and Nishimura (1998), has become quite popular; it runs very fast and has the huge period of  $2^{19937} - 1$ . However, its state  $\mathbf{x}_i$  occupies a large amount of memory (19937 bits) and changes very slowly as a function of  $i$ . Panneton et al. (2006) showed that as a consequence of this slow change, if the generator starts in a state with very few bits equal to 1, then the average output values over the next few thousand steps is likely to be much less than 1/2. In particular, if the initial state has a single bit at 1, say randomly selected, then we need about 3/4 million steps before the average output value gets close to 1/2. Likewise, if two initial states differ by a single bit, it takes the same number of steps before the corresponding outputs differ by about half of their bits. This problem is related to the fact that the characteristic polynomial  $P(z)$  has too few nonzero coefficients, namely 135 out of 19938.**

Panneton et al. (2006) went on to develop a class of  $\mathbb{F}_2$ -linear generators called *well-equidistributed long-period linear* (WELL), which run almost as fast as MT19937, but whose state changes faster and whose polynomial  $P(z)$  contains nearly 50% nonzero coefficients. They propose specific instances with periods ranging from  $2^{512} - 1$  to  $2^{44497} - 1$ , which are all almost (or exactly) maximally equidistributed.

In the *multiple recursive matrix method* of Niederreiter (1995), the first row of  $p \times p$  matrices in  $\mathbf{A}$  contains arbitrary matrices. However, a fast implementation is possible only when these matrices are sparse and have a special structure.

## 4.6 Combined Linear Generators Over $\mathbb{F}_2$

Many of the best generators based on linear recurrences over  $\mathbb{F}_2$  are constructed by combining the outputs of two or more RNGs having a simple structure. The idea is the same as for MRGs: select simple components that can run fast but such that their combination has a more complicated structure and highly-uniform sets  $\Psi_I$  for the sets  $I$  considered important.

Consider  $J$  distinct recurrences of the form (12)–(13), where the  $j$ th recurrence has parameters  $(k, w, \mathbf{A}, \mathbf{B}) = (k_j, w, \mathbf{A}_j, \mathbf{B}_j)$  and state  $\mathbf{x}_{j,i}$  at step  $i$ , for  $j = 1, \dots, J$ . The output of the combined generator at step  $i$  is defined by

$$\begin{aligned} \mathbf{y}_i &= \mathbf{B}_1 \mathbf{x}_{1,i} \oplus \dots \oplus \mathbf{B}_J \mathbf{x}_{J,i}, \\ u_i &= \sum_{\ell=1}^w y_{i,\ell-1} 2^{-\ell}, \end{aligned}$$

where  $\oplus$  denotes the bitwise exclusive-or operation. One can show (Tezuka, 1995) that the period  $\rho$  of this combined generator is the least common multiple of the periods  $\rho_j$  of its components. Moreover, this combined generator is equivalent to the generator (12)–(14) with  $k = k_1 + \dots + k_J$ ,  $\mathbf{A} = \text{diag}(\mathbf{A}_1, \dots, \mathbf{A}_J)$ , and  $\mathbf{B} = (\mathbf{B}_1, \dots, \mathbf{B}_J)$ .

With this method, by selecting the parameters carefully, the combination of LFSR generators with characteristic polynomials  $P_1(z), \dots, P_J(z)$  gives yet another LFSR with characteristic polynomial  $P(z) = P_1(z) \cdots P_J(z)$  and period equal to the product of the periods of the components (Tezuka and L'Ecuyer, 1991, Wang and Compagner, 1993, L'Ecuyer, 1996b, Tezuka, 1995). Tables and fast implementations of maximally equidistributed combined LFSR generators are given in L'Ecuyer (1999c).

The TGFSR and Mersenne twister generators cannot be maximally equidistributed. However, concrete examples of maximally equidistributed combined TGFSR generators with periods near  $2^{466}$  and  $2^{1250}$  can be found in L'Ecuyer and Panneton (2002). These generators have the additional property that the resolution gaps  $\delta_I$  are zero for a class of small sets  $I$  with indices not too far apart.

## 5 Nonlinear RNGs

All RNGs discussed so far are based on linear recurrences and their structure may be deemed too regular. **For example, we saw earlier that the output binary sequence  $\{y_{i,j}, i \geq 0\}$  of any  $\mathbb{F}_2$ -linear generator obeys the linear recurrence (15). This can be detected easily by applying statistical tests that measure the linear complexity of this output sequence, or that construct “random” binary matrices from this sequence and compute their ranks (L'Ecuyer and Simard, 2007). Because of the linear dependences between the bits, the linear complexity and the matrix ranks will be smaller than what they should be on average. For the great majority of Monte Carlo applications, this linearity is not a problem, because the random numbers are transformed nonlinearly by the simulation algorithm. But for the rare situations where it may matter, we need alternatives.**

There are several ways of getting rid of the regular linear structure, including: (1) use a nonlinear transition function  $f$ ; (2) keep the transition function linear but use a nonlinear output function  $g$ ; (3) combine two linear RNGs of different types, such as an MRG with an  $\mathbb{F}_2$ -linear generator; (4) shuffle (randomly permute) the output values using another generator. Several types of genuinely nonlinear RNGs have been proposed over the years; see for example Blum et al. (1986), Eichenauer-Herrmann (1995), Eichenauer-Herrmann et al. (1998), Hellekalek and Wegenkittl (2003), Knuth (1998), L'Ecuyer and Proulx (1989), L'Ecuyer (1994), L'Ecuyer and Simard (2007), Niederreiter and Shparlinski (2002), and Tezuka (1995). Their nonlinear mappings are defined in various ways by multiplicative inversion in a finite field, quadratic and cubic functions in the finite ring of integers modulo  $m$ , and other more complicated transformations. Many of them have output sequences that tend to behave much like i.i.d.  $U(0, 1)$  sequences even over their entire period length, in contrast with “good” linear RNGs, whose point sets  $\Psi_t$  are much more regular than typical random points (Eichenauer-Herrmann et al., 1998, L'Ecuyer and Hellekalek, 1998, L'Ecuyer and Granger-Piché, 2003, Niederreiter and Shparlinski, 2002). On the other hand, their statistical properties have been analyzed only empirically or via asymptotic theoretical results. For specific nonlinear RNGs, the uniformity of the point sets  $\Psi_t$  is very difficult to measure theoretically. Moreover, the nonlinear RNGs are generally significantly slower than the linear ones. The RNGs recommended for cryptology are all nonlinear.

An interesting idea for adding nonlinearity without incurring an excessive speed penalty is to combine a small nonlinear generator with a fast long-period linear one (Aiello et al., 1998). L'Ecuyer and Granger-Piché (2003) show how to do this while ensuring theoretically the good uniformity properties of  $\Psi_t$  for the combined generator. A fast implementation can be achieved by using precomputed tables for the nonlinear component. Empirical studies suggest that mixed linear-nonlinear combined generators are more robust than the linear ones with respect to statistical tests, because of their less regular structure.

Several authors have proposed various ways of combining RNGs to produce streams of random numbers with less regularity and better “randomness” properties; see, e.g., Collings (1987), Knuth (1998), Gentle (2003), Law and Kelton (2000), L'Ecuyer (1994), Fishman (1996), Marsaglia (1985), and other references given there. This includes *shuffling* the output sequence of one generator using another one (or the same one), alternating between several streams, or just adding them in different ways. Most of these techniques are heuristics. They usually improve the uniformity (empirically), but they can also make it worse. For *random variables* in the mathematical sense, certain types of combinations (e.g., addition modulo 1) can *provably*

improve the uniformity, and some authors have used this fact to argue that combined RNGs are provably better than their components alone (Brown and Solomon, 1979, Deng and George, 1990, Marsaglia, 1985, Gentle, 2003), but this argument is faulty because the output sequences of RNGs are deterministic, not sequences of independent random variables. To assess the quality of a combined generator, one must analyze the mathematical structure of the combined generator itself rather than the structure of its components (L'Ecuyer, 1996b,a, 1998, L'Ecuyer and Granger-Piché, 2003, Tezuka, 1995).

## 6 Empirical Statistical Tests

As mentioned earlier, a statistical test for RNGs is defined by a random variable  $X$  whose distribution under  $\mathcal{H}_0$  can be well approximated. When  $X$  takes the value  $x$ , we define the right and left  $p$ -values of the test by

$$p_R = P[X \geq x \mid \mathcal{H}_0] \quad \text{and} \quad p_L = P[X \leq x \mid \mathcal{H}_0].$$

When testing RNGs, there is no need to prespecify the level of the test. If either of the right or left  $p$ -value is extremely close to zero, e.g., less than  $10^{-15}$ , then it is clear that  $\mathcal{H}_0$  (and the RNG) must be rejected. When a *suspicious*  $p$ -value is obtained, e.g., near  $10^{-2}$  or  $10^{-3}$ , one can just repeat this particular test a few more times, perhaps with a larger sample size. Almost always, things will then clarify.

Most tests are defined by partitioning the possible realizations of  $(u_0, \dots, u_{\tau-1})$  into a finite number of subsets (where the integer  $\tau$  can be random or deterministic), computing the probability  $p_j$  of each subset  $j$  under  $\mathcal{H}_0$ , and measuring the discrepancy between these probabilities and empirical frequencies from realizations simulated by the RNG.

A special case that immediately comes to mind is to take  $\tau = t$  (a constant) and cut the interval  $[0, 1)$  into  $d$  equal segments for some positive integer  $d$ , in order to partition the hypercube  $[0, 1)^t$  into  $k = d^t$  subcubes of volume  $1/k$ . We then generate  $n$  points  $\mathbf{u}_i = (u_{ti}, \dots, u_{ti+t-1}) \in [0, 1)^t$ , for  $i = 0, \dots, n-1$ , and count the number  $N_j$  of points falling in subcube  $j$ , for  $j = 0, \dots, k-1$ . Any measure of distance (or divergence) between the numbers  $N_j$  and their expectations  $n/k$  can define a test statistic  $X$ . The tests thus defined are generally called *serial tests* of uniformity (Knuth, 1998, L'Ecuyer et al., 2002). They can be *sparse* (if  $n/k \ll 1$ ), or *dense* (if  $n/k \gg 1$ ), or somewhere in between. There are also *overlapping* versions, where the points are defined by  $\mathbf{u}_i = (u_i, \dots, u_{i+t-1})$  for  $i = 0, \dots, n-1$  (they have overlapping coordinates).

Special instances for which the distribution under  $\mathcal{H}_0$  is well-known are the chi-square, the (negative) empirical entropy, and the number of collisions (L'Ecuyer and Hellekalek, 1998, L'Ecuyer et al., 2002, Read and Cressie, 1988). For the latter, the test statistic  $X$  is the number of times a point falls in a subcube that already had a point in it. Its distribution under  $\mathcal{H}_0$  is approximately Poisson with mean  $\lambda_1 = n^2/(2k)$ , if  $n$  is large and  $\lambda_1$  not too large.

A variant is the *birthday spacings* test, defined as follows (Marsaglia, 1985, Knuth, 1998, L'Ecuyer and Simard, 2001). Let  $I_{(1)} \leq \dots \leq I_{(n)}$  be the numbers of the subcubes that contain the points, sorted by increasing order. Define the *spacings*  $S_j = I_{(j+1)} - I_{(j)}$ , for  $j = 1, \dots, n-1$ , and let  $X$  be the number of collisions between these spacings. Under  $\mathcal{H}_0$ ,  $X$  is approximately Poisson with mean  $\lambda_2 = n^3/(4k)$ , if  $n$  is large and  $\lambda_2$  not too large.

Consider now a MRG, for which  $\Psi_t$  has a regular lattice structure. Because of this regularity the points of  $\Psi_t$  will tend to be more evenly distributed among the subcubes than random points. For a well-chosen  $k$  and large enough  $n$ , we expect the collision test to detect this: it is likely that there will be too few collisions. In fact, the same applies to any RNG whose set  $\Psi_t$  is very evenly distributed. When a birthday spacings test with a very large  $k$  is applied to a MRG, the numbers of the subcubes that contain one point of  $\Psi_t$  tend to be too evenly spaced and the test detects this by finding too many collisions.

These specific interactions between the test and the structure of the RNG lead to systematic patterns in the  $p$ -values of the tests. To illustrate this, suppose that we take  $k$  slightly larger than the cardinality of  $\Psi_t$  (so  $k \approx \rho$ ) and that due to the excessive regularity, no collision is observed in the collision test. The left  $p$ -value will then be  $p_L \approx P[X \leq 0 \mid X \sim \text{Poisson}(\lambda_1)] = \exp[-n^2/(2k)]$ . For this  $p$ -value to be smaller than a given  $\epsilon$ , we need a sample size  $n$  proportional to the square root of the period  $\rho$ . And after that,  $p_L$  decreases exponentially fast in  $n^2$ .



Extensive experiments with LCGs, MRGs, and LFSR generators confirms that this is actually what happens with these RNGs (L'Ecuyer and Hellekalek, 1998, L'Ecuyer, 2001, L'Ecuyer et al., 2002). For example, if we take  $\epsilon = 10^{-15}$  and define  $n_0$  as the minimal sample size  $n$  for which  $p_L < \epsilon$ , we find that  $n_0 \approx 16\rho^{1/2}$  (plus some noise) for LCGs that behave well in the spectral test as well as for LFSR generators. For the birthday spacings test, the rule for LCGs is  $n_0 \approx 16\rho^{1/3}$  instead (L'Ecuyer and Simard, 2001). So to be safe with respect to these tests, the period  $\rho$  must be so large that generating more than  $\rho^{1/3}$  numbers is practically unfeasible. This certainly disqualifies all LCGs with modulus smaller than  $2^{100}$  or so.

Other types of tests for RNGs include tests based on the closest pairs of points among  $n$  points generated in the hypercube, tests based on random walks on the real line or over the integers, tests based on the linear complexity of a binary sequence, tests based on the simulation of dice or poker hands, and many others (Gentle, 2003, Knuth, 1998, L'Ecuyer and Simard, 2007, Marsaglia, 1996, Rukhin et al., 2001, Vattulainen et al., 1995).

When testing RNGs, there is no specific alternative hypothesis to  $\mathcal{H}_0$ . Different tests are needed to detect different types of departures from  $\mathcal{H}_0$ . The *TestU01* library of L'Ecuyer and Simard (2007) implements a large collection of tests in the C language, and also provides specific test suites with preselected parameters and sample sizes. Some of these suites are designed for i.i.d.  $U(0, 1)$  output sequences and others for strings of bits. Other (smaller) test suites for RNGs are DIEHARD (Marsaglia, 1996) and the NIST suite (Rukhin et al., 2001).

## 7 Available Software and Recommendations

Applying standard statistical test suites to RNGs found in popular software (statistical and simulation software, spreadsheets, system libraries, etc.) reveals that many of them are surprisingly poor and fail the tests spectacularly (L'Ecuyer, 2001, L'Ecuyer and Simard, 2007). There is no good reason to use these poor RNGs, because several good ones are available that are fast, portable, and pass these test suites with flying colors.

The RNG I use most of the time is the combined MRG *MRG32k3a* from L'Ecuyer (1999a). A convenient object-oriented software package with multiple streams and substreams of random numbers, based on this generator, is described in L'Ecuyer et al. (2002) and is available in Java, C, and C++, at <http://www.iro.umontreal.ca/~lecuyer>. This tool has been included recently in several software products, including MATLAB, SAS, R, Arena, Automod, ns3, and many more. *MRG32k3a* is not the fastest RNG available, but it is very robust and reliable. A faster alternative is *MGR31k3p* from L'Ecuyer and Touzin (2000). Other good combined MRGs, some for 64-bit computers, are available in L'Ecuyer (1999a). Even faster ones are the combined LFSRs, Mersenne twisters, and WELL generators proposed in L'Ecuyer (1999c), L'Ecuyer and Panneton (2002), Matsumoto and Nishimura (1998), Nishimura (2000), and Panneton et al. (2006). When speed is a concern, I personally use *LFSR113* or *LFSR258* from L'Ecuyer (1999c). Software tools that provide multiple streams and substreams with most of these generators (except the ones with very large state) are available in the *SSJ* library (L'Ecuyer, 2008).

## 8 Non-Uniform Random Variate Generation

Like for the uniform case, non-uniform variate generation often involves approximations and compromises. The first requirement is, of course, *correctness*. This does not mean that the generated random variate  $X$  must always have *exactly* the required distribution, because this would sometimes be much too costly or even impossible. But we must have a *good approximation* and, preferably, some understanding of the quality of that approximation. *Robustness* is also important: when the accuracy depends on the parameters of the distribution, it must be good *uniformly* over the entire range of parameter values that we are interested in.

The method must also be *efficient* both in terms of speed and memory usage. Often, it is possible to increase the speed by using more memory (e.g. for larger precomputed tables) or by relaxing the accuracy requirements. Some methods need a one-time setup to compute constants and construct tables. The setup time can be significant but may be well worth spending if it is amortized by a large number of subsequent calls to the generator. For example, it makes sense to invest in a more extensive setup if we plan to make a

million calls to a given generator than if we expect to make only a few calls, assuming that this investment can improve the speed of the generator sufficiently.

In general, compromises must be made between simplicity of the algorithm, quality of the approximation, robustness with respect to the distribution parameters, and efficiency (generation speed, memory requirements, and setup time).

In many situations, compatibility with variance reduction techniques is another important issue (Asmussen and Glynn, 2007, Bratley et al., 1987, Law and Kelton, 2000). We may be willing to sacrifice the speed of the generator to preserve inversion, because the gain in efficiency obtained via the variance reduction methods may more than compensate (sometimes by orders of magnitude) for the slightly slower generator.

## 8.1 Inversion

The inversion method, defined in the introduction, should be the method of choice for generating non-uniform random variates in a majority of situations. The fact that  $X = F^{-1}(U)$  is a monotone (non-decreasing) function of  $U$  makes this method compatible with important variance reductions techniques such as common random numbers, antithetic variates, Latin hypercube sampling, and randomized quasi-Monte Carlo methods (Bratley et al., 1987, Law and Kelton, 2000, L'Ecuyer and Lemieux, 2000, L'Ecuyer et al., 2009).

For some distributions, an analytic expression can be obtained for the inverse distribution function  $F^{-1}$  and inversion can be easily implemented. As an example, consider the *Weibull* distribution function with parameters  $\alpha > 0$  and  $\beta > 0$ , defined by  $F(x) = 1 - \exp[-(x/\beta)^\alpha]$  for  $x > 0$ . It is easy to see that  $F^{-1}(U) = \beta[-\ln(1 - U)]^{1/\alpha}$ . For  $\alpha = 1$ , we have the special case of the exponential distribution with mean  $\beta$ .

For an example of a simple discrete distribution, suppose that  $P[X = i] = p_i$  where  $p_0 = 0.6$ ,  $p_1 = 0.3$ ,  $p_2 = 0.1$ , and  $p_i = 0$  elsewhere. The inversion method in this case will return 0 if  $U < 0.6$ , 1 if  $0.6 \leq U < 0.9$ , and 2 if  $U \geq 0.9$ . For the discrete uniform distribution over  $\{0, \dots, k-1\}$ , return  $X = \lfloor kU \rfloor$ . As another example, let  $X$  have the *geometric* distribution with parameter  $p$ , so  $P[X = x] = p(1-p)^x$  for  $x = 0, 1, 2, \dots$ , where  $0 < p < 1$ . Then,  $F(x) = 1 - (1-p)^{\lfloor x+1 \rfloor}$  for  $x \geq 0$  and one can show that  $X = F^{-1}(U) = \lceil \ln(1 - U) / \ln(1 - p) \rceil - 1$ .

For other distributions (e.g., the normal, Student, chi-square) there is no closed-form expression for  $F^{-1}$  but good numerical approximations are available (Bratley et al., 1987, Gentle, 2003, Hörmann et al., 2004, Marsaglia et al., 1994). When the distribution has only scale and location parameters, we need to approximate  $F^{-1}$  only for a standardized version of the distribution. For the normal distribution, for example, it suffices to have an efficient method for evaluating the inverse distribution function of a  $N(0, 1)$  random variable  $Z$ , since a normal with mean  $\mu$  and variance  $\sigma^2$  can be generated by  $X = \sigma Z + \mu$ . When shape parameters are involved (e.g., the gamma and beta distributions), things are more complicated because  $F^{-1}$  then depends on the parameters in a more fundamental manner.

Hörmann and Leydold (2003) propose a general adaptive and automatic method that constructs a highly accurate Hermite interpolation method of  $F^{-1}$ . In a one-time setup, their method produces tables for the interpolation points and coefficients. Random variate generation using these tables is then quite fast.

A less efficient but simpler way of implementing inversion when a method is available for computing  $F$  is via binary search (Cheng, 1998). If the density is also available and if it is unimodal with known mode, a Newton-Raphson iteration method can advantageously replace the binary search (Cheng, 1998, Devroye, 1986).

To implement inversion for general discrete distributions, sequential search and binary search with look-up tables are the standard methods (Bratley et al., 1987, Cheng, 1998). For a discrete distribution over the values  $x_1 < \dots < x_k$ , one first tabulates the pairs  $(x_i, F(x_i))$ , where  $F(x_i) = P[X \leq x_i]$  for  $i = 1, \dots, k$ . To generate  $X$ , it then suffices to generate  $U \sim U(0, 1)$ , find  $I = \min\{i \mid F(x_i) \geq U\}$ , and return  $X = x_I$ . The following algorithms do that.

**Sequential search** (needs  $O(k)$  iterations in the worst case);

```
generate  $U \sim U(0, 1)$ ;   let  $i = 1$ ;
while  $F(x_i) < U$  do  $i = i + 1$ ;
return  $x_i$ .
```

**Binary search** (needs  $O(\log k)$  iterations in the worst case);

```

generate  $U \sim U(0, 1)$ ;   let  $L = 0$  and  $R = k$ ;
while  $L < R - 1$  do
   $m = \lfloor (L + R)/2 \rfloor$ ;
  if  $F(x_m) < U$  then  $L = m$  else  $R = m$ ;
  /* Invariant: at this stage, the index  $I$  is in  $\{L + 1, \dots, R\}$ . */
return  $x_R$ .

```

These algorithms can be modified in many different ways. For example, if  $k = \infty$ , in the binary search, one can start with an arbitrary value of  $R$ , double it until  $F(x_R) \geq U$ , and start the algorithm with this  $R$  and  $L = R/2$ . Of course, only a finite portion of the table (a portion that contains most of the probability mass) would be precomputed in this case, the other values can be computed only when needed. This can also be done if  $k$  is finite but large.

Another class of techniques use indexing or buckets to speed up the search (Bratley et al., 1987, Chen and Asau, 1974, Devroye, 1986). For example, one can partition the interval  $(0, 1)$  into  $c$  subintervals of equal sizes and use (pre-tabulated) initial values of  $(L, R)$  that depend on the subinterval in which  $U$  falls. For the subinterval  $[j/c, (j + 1)/c)$  the values of  $L$  and  $R$  would be  $L_j = F^{-1}(j/c)$  and  $R_j = F^{-1}((j + 1)/c)$ , for  $j = 0, \dots, c - 1$ . The subinterval number that corresponds to a given  $U$  is simply  $J = \lfloor cU \rfloor$ . Once we know that subinterval, we can search it by linear or binary search. With a larger value of  $c$  the search is faster (on the average) but the setup is more costly and a larger amount of memory is needed. So a compromise must be made depending on the situation (e.g., the value of  $k$ , the number of variates we expect to generate, etc.). For  $c = 1$ , we recover the basic sequential and binary search algorithms given above. A well-implemented indexed search with a large enough  $c$  is competitive with the alias method (described in the next paragraph). A combined indexed/binary search algorithm is given below. An easy adaptation gives the combined indexed/sequential search, which is generally preferable when  $k/c$  is small, because it has smaller overhead.

```

Indexed search (combined with binary search);
generate  $U \sim U(0, 1)$ ;   let  $J = \lfloor cU \rfloor$ ,  $L = L_J$ , and  $R = R_J$ ;
while  $L < R - 1$  do
   $m = \lfloor (L + R)/2 \rfloor$ ;
  if  $F(x_m) < U$  then  $L = m$  else  $R = m$ ;
return  $x_R$ .

```

These search methods are also useful for piecewise-linear (or piecewise-polynomial) distribution functions. Essentially, it suffices to add an interpolation step at the end of the algorithm, after the appropriate linear (or polynomial) piece has been determined (Bratley et al., 1987).

Finally, the stochastic model itself can sometimes be selected in a way that makes inversion easier. For example, one can fit a parametric, highly-flexible, and easily computable inverse distribution function  $F^{-1}$  to the data, directly or indirectly (Nelson and Yamnitsky, 1998).

There are situations where *speed* is important and where non-inversion methods are appropriate. In forthcoming subsections, we outline the main non-inversion methods.

## 8.2 The Alias Method

Sequential and binary search require  $O(k)$  and  $O(\log k)$  time, respectively, in the worst case, to generate a random variate  $X$  by inversion over the finite set  $\{x_1, \dots, x_k\}$ . The *alias method* (Walker, 1977) can generate such a  $X$  in  $O(1)$  time per variate, after a table setup that takes  $O(k)$  time and space. On the other hand, it does not implement inversion, i.e., the transformation from  $U$  to  $X$  is not monotone.

To explain the idea, consider a bar diagram of the distribution, where each index  $i$  has a bar of height  $p_i = P[X = x_i]$ . The idea is to “equalize” the bars so that they all have height  $1/k$ , by cutting-off bar pieces and transferring them to other bars. This is done in a way that in the new diagram, each bar  $i$  contains one piece of size  $q_i$  (say) from the original bar  $i$  and one piece of size  $1/k - q_i$  from another bar whose index  $j$ , denoted  $A(i)$ , is called the *alias* value of  $i$ . The setup procedure initializes two tables,  $A$  and  $R$ , where  $A(i)$  is the alias value of  $i$  and  $R(i) = (i - 1)/k + q_i$ . See Devroye (1986) and Law and Kelton (2000) for the details. To generate  $X$ , we generate  $U \sim U[0, 1]$ , define  $I = \lceil kU \rceil$ , and return  $X = x_I$  if  $U < R(I)$  and  $X = x_{A(I)}$  otherwise.

There is a version of the alias method for continuous distributions, called the *acceptance-complement* method (Kronmal and Peterson, 1984, Devroye, 1986, Gentle, 2003). The idea is to decompose the density  $f$  of the target distribution as the convex combination of two densities  $f_1$  and  $f_2$ ,  $f = wf_1 + (1 - w)f_2$  for some real number  $w \in (0, 1)$ , in a way that  $wf_1 \leq g$  for some other density  $g$  and so that it is easy to generate from  $g$  and  $f_2$ . The algorithm works as follows: Generate  $X$  from density  $g$  and  $U \sim U(0, 1)$ ; if  $Ug(X) \leq wf_1(X)$  return  $X$ , otherwise generate a new  $X$  from density  $f_2$  and return it.

### 8.3 Kernel Density Estimation and Generation

Instead of selecting a parametric distribution that is hard to invert and estimating the parameters, one can estimate the density via a *kernel density estimation* method for which random variate generation is very easy (Devroye, 1986, Hörmann et al., 2004). In the case of a Gaussian kernel, for example, one can generate variates simply by selecting one observation at random from the data and adding random noise generated from a normal distribution with mean zero. However, this method *is not* equivalent to inversion. Because of the added noise, selecting a larger observation does not necessarily guarantee a larger value for the generated variate.

### 8.4 The Rejection Method

Suppose we want to generate  $X$  from a complicated density  $f$ . In fact  $f$  may be known only up to a multiplicative constant  $\kappa > 0$ , i.e., we know only  $\kappa f$ . If we know  $f$ , we may just take  $\kappa = 1$ . We select another density  $r$  such that  $\kappa f(x) \leq t(x) \stackrel{\text{def}}{=} ar(x)$  for all  $x$  for some constant  $a$ , and such that generating variates  $Y$  from the density  $r$  is easy. The function  $t$  is called a *hat function* or *majorizing function*. By integrating this inequality with respect to  $x$  on both sides, we find that  $\kappa \leq a$ . The following *rejection* method generates  $X$  with density  $f$  (von Neumann, 1951, Devroye, 1986, Evans and Swartz, 2000):

**Rejection method;**

```
repeat
  generate  $Y$  from the density  $r$  and  $U \sim U(0, 1)$ , independent;
until  $Ut(Y) \leq \kappa f(Y)$ ;
return  $X = Y$ .
```

The number  $R$  of turns into the “repeat” loop is one plus a geometric random variable with parameter  $\kappa/a$ , so  $E[R] = a/\kappa$ . Thus, we want  $a/\kappa \geq 1$  to be as small as possible, i.e., we want to minimize the area between  $\kappa f$  and  $t$ . There is generally a compromise between bringing  $a/\kappa$  close to 1 and keeping  $r$  simple.

When  $\kappa f$  is expensive to compute, we can also use *squeeze functions*  $q_1$  and  $q_2$  that are faster to evaluate and such that  $q_1(x) \leq \kappa f(x) \leq q_2(x) \leq t(x)$  for all  $x$ . To verify the condition  $Ut(Y) \leq \kappa f(Y)$ , we first check if  $Ut(Y) \leq q_1(Y)$ , in which case we accept  $Y$  immediately, otherwise we check if  $Ut(Y) \geq q_2(Y)$ , in which case we reject  $Y$  immediately. The value of  $\kappa f(Y)$  must be computed only when  $Ut(Y)$  falls between the two squeezes. Sequences of embedded squeezes can also be used, where the primary ones are the least expensive to compute, the secondary ones are a little more expensive but closer to  $\kappa f$ , etc.

It is common practice to transform the density  $f$  by a smooth increasing function  $T$  (e.g.,  $T(x) = \log x$  or  $T(x) = -x^{-1/2}$ ) selected so that it is easier to construct good hat and squeeze functions (often piecewise linear) for the transformed density  $T(f(\cdot))$ . By transforming back to the original scale, we get hat and squeeze functions for  $f$ . This is the *transformed density rejection* method, which has several variants and extensions (Devroye, 1986, Evans and Swartz, 2000, Hörmann et al., 2004).

The rejection method works for discrete distributions as well; it suffices to replace densities by probability mass functions.

## 8.5 Thinning for Point Processes with Time-Varying Rates

*Thinning* is a cousin of acceptance-rejection, often used for generating events from a non-homogeneous Poisson process. Suppose the process has rate  $\lambda(t)$  at time  $t$ , with  $\lambda(t) \leq \bar{\lambda}$  for all  $t$ , where  $\bar{\lambda}$  is a finite constant. One can generate Poisson *pseudo-arrivals* at constant rate  $\bar{\lambda}$  by generating interarrival times that are i.i.d. exponentials of mean  $1/\bar{\lambda}$ . Then, a pseudo-arrival at time  $t$  is accepted (becomes an arrival) with probability  $\lambda(t)/\bar{\lambda}$  (i.e., if  $U \leq \lambda(t)/\bar{\lambda}$ , where  $U$  is an independent  $U[0, 1]$ ), and rejected with probability  $1 - \lambda(t)/\bar{\lambda}$ . Non-homogeneous Poisson processes can also be generated by inversion (Bratley et al., 1987). The idea is to apply a nonlinear transformation to the time scale to make the process homogeneous with rate 1 in the new time scale. Arrival times are generated in this new time scale (which is easy), and then transformed back to the original time scale. The method can be adapted to other types of point processes with time-varying rates.

## 8.6 The Ratio-of-Uniforms Method

If  $f$  is a density over the real-line,  $\kappa$  an arbitrary positive constant, and the pair  $(U, V)$  has the uniform distribution over the set

$$\mathcal{C} = \left\{ (u, v) \in \mathbb{R}^2 \text{ such that } 0 \leq u \leq \sqrt{\kappa f(v/u)} \right\},$$

then  $V/U$  has density  $f$  (Kinderman and Monahan, 1977, Devroye, 1986, Gentle, 2003). This interesting property can be exploited to generate  $X$  with density  $f$ : generate  $(U, V)$  uniformly over  $\mathcal{C}$  and return  $X = V/U$ . This is the *ratio-of-uniforms* method. The key issue is how do we generate a point uniformly over  $\mathcal{C}$ . In the cases where this can be done efficiently, we immediately have an efficient way of generating  $X$ .

The most frequent approach for generating  $(U, V)$  uniformly over  $\mathcal{C}$  is the rejection method: Define a region  $\mathcal{C}_2$  that contains  $\mathcal{C}$  and in which it is easy to generate a point uniformly (for example, a rectangular box or a polygonal region). To generate  $X$ , repeat: generate  $(U, V)$  uniformly over  $\mathcal{C}_2$ , until it belongs to  $\mathcal{C}$ . Then return  $X = V/U$ . If there is another region  $\mathcal{C}_1$  contained in  $\mathcal{C}$  and for which it is very fast to check if a point  $(U, V)$  is in  $\mathcal{C}_1$ , this  $\mathcal{C}_1$  can also be used as a squeeze to accelerate the verification that the point belongs to  $\mathcal{C}$ . Several special cases and refinements are described in Devroye (1986), Gentle (2003), Leydold (2000), and other references given there.

## 8.7 Composition and Convolution

Suppose  $F$  is a convex combination of several distributions, i.e.,  $F(x) = \sum_j p_j F_j(x)$ , or more generally  $F(x) = \int F_y(x) dH(y)$ . To generate from  $F$ , one can generate  $J = j$  with probability  $p_j$  (or  $Y$  from  $H$ ), then generate  $X$  from  $F_J$  (or  $F_Y$ ). This method, called the *composition algorithm*, is useful for generating from *compound* distributions such as the hyperexponential or from compound Poisson processes. It is also frequently used to design specialized algorithms for generating from complicated densities. The idea is to partition the area under the complicated density into pieces, where piece  $j$  has surface  $p_j$ . To generate  $X$ , first select a piece (choose piece  $j$  with probability  $p_j$ ), then draw a random point uniformly over that piece and project it to the horizontal axis. If the partition is defined so that it is fast and easy to generate from the large pieces, then  $X$  will be returned very quickly most of the time. The rejection method with a squeeze is often used to generate from some of the pieces.

A dual method to composition is the *convolution method*, which can be used when  $X = Y_1 + Y_2 + \dots + Y_n$ , where the  $Y_i$ 's are independent with specified distributions. With this method, one just generates the  $Y_i$ 's and sum up. This requires at least  $n$  uniforms. Examples of random variables that can be expressed as sums like this include the hypoexponential, Erlang, and binomial distributions.

## 8.8 Other Special Techniques

Specialized and sometimes very elegant techniques have been designed for commonly used distributions such as the Poisson distribution with small mean, the normal (e.g., the Box-Muller and the polar methods), for generating points uniformly on a  $k$ -dimensional sphere, for generating random permutations, and so on. Details can be found, e.g., in Bratley et al. (1987), Cheng (1998), Devroye (1986), Fishman (1996), Gentle (2003). Many of those methods are based on a clever multivariate change of variables, defined so that the random variates or random vectors in the new coordinates are much easier to generate. In the Box-Muller and Polar methods, for example, a pair of independent standard normals is generated in polar coordinates, and then transformed back into rectangular coordinates.

Recently, there has been an effort in developing *automatic* or *black box* algorithms for generating variates from an arbitrary (known) density, and reliable software that implements these methods (Hörmann and Leydold, 2000, Hörmann et al., 2004, Leydold, 2009).

## 8.9 Multivariate Distributions

Inversion does not directly apply to generate a  $d$ -dimensional random vector  $\mathbf{X} = (X_1, \dots, X_d)^\top$ , because the inverse of its distribution function is not well defined. In some cases, one can generate the first coordinate  $X_1$  by inversion from its marginal distribution, then generate  $X_2$  by inversion from its marginal distribution conditional on  $X_1$ , then generate  $X_3$  by inversion from its marginal distribution conditional on  $(X_1, X_2)$ , and so on. But this is not always possible and convenient.

Simple and elegant methods are available for certain classes of distributions. For example, if  $\mathbf{X}$  has a *multinormal distribution* with mean vector  $\boldsymbol{\mu}$  and covariance matrix  $\boldsymbol{\Sigma}$ , then one can decompose  $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^\top$  for some matrix  $\mathbf{A}$ , generate a vector  $\mathbf{Z}$  of  $d$  independent standard normal random variable (with mean 0 and variance 1), usually by inversion, and return  $\mathbf{X} = \boldsymbol{\mu} + \mathbf{A}\mathbf{Z}$ . One way to decompose  $\boldsymbol{\Sigma}$  is via the Cholesky decomposition, for which  $\mathbf{A}$  is lower triangular, but there are many other possibilities, including the eigendecomposition as in principal component analysis. The choice of decomposition can have a large impact on the variance reduction in the context of randomized quasi-Monte Carlo integration, by concentrating more of the variance on the first few uniform random numbers that are generated L'Ecuyer (2009).

Multivariate normals are often used to generate vectors from other distributions. For example, to generate a random point on a  $d$ -dimensional sphere of radius  $r$  centered at zero, one can generate a vector  $\mathbf{Z}$  of independent standard normals (this amounts to generating a random direction), then normalize its length to  $r$ . More generally, by putting  $\mathbf{X} = R\mathbf{Z}/\|\mathbf{Z}\|$  where  $R$  has an arbitrary distribution over  $(0, \infty)$ , one generates a vector with a *radially symmetric* distribution. As a special case, if  $R$  has the Student distribution,  $\mathbf{X}$  is multivariate Student. As a further generalization, let  $\mathbf{X} = \boldsymbol{\mu} + R\mathbf{A}\mathbf{Z}/\|\mathbf{Z}\|$  where  $\mathbf{Z}$  is multinormal in  $k$  dimensions and  $\mathbf{A}$  is a  $d \times k$  matrix. This  $\mathbf{X}$  has an *elliptic multivariate* distribution.

A richer class of multivariate distributions are defined via *copula methods* (Asmussen and Glynn, 2007, Hörmann et al., 2004, Nelsen, 1999). Start with an arbitrary  $d$ -dimensional distribution function  $G$  with continuous marginals  $G_j$ , generate  $\mathbf{Y} = (Y_1, \dots, Y_d)^\top$  from  $G$ , and let  $\mathbf{U} = (U_1, \dots, U_d) = (G_1(Y_1), \dots, G_d(Y_d))^\top$ . These  $U_j$  have the uniform distribution over  $(0, 1)$ , but they are not independent in general. The distribution function of  $\mathbf{U}$  is the *copula associated with  $G$*  and it specifies the dependence structure of the vector  $\mathbf{U}$ . In fact, any multivariate distribution function over  $(0, 1)^d$  with uniform marginals is a copula. To generate  $\mathbf{X} = (X_1, \dots, X_d)^\top$  with arbitrary marginal distribution functions  $F_j$  and dependence structure specified by this copula, put  $X_j = F_j^{-1}(U_j)$  for each  $j$ . A popular choice for  $G$  is the multinormal distribution with standard normal marginals; then  $\mathbf{Y}$  and  $\mathbf{U}$  are easy to generate, and one can select the correlation matrix of  $\mathbf{Y}$  to approximate a target correlation (or rank correlation) matrix for  $\mathbf{X}$ . It can usually match the correlations pretty well. But to approximate the whole dependence structure in general, a much richer variety of copulas is required Asmussen and Glynn (2007), Hörmann et al. (2004), Nelsen (1999).

The rejection method extends easily to the multivariate case. For a target  $d$ -dimensional density  $f$  known up to the multiplicative constant  $\kappa$ , pick a  $d$ -dimensional density  $r$  such that  $\kappa f(\mathbf{x}) \leq ar(\mathbf{x})$  for all  $\mathbf{x}$  and some constant  $a$ , and such that sampling random vectors  $\mathbf{Y}$  from  $r$  is easy. To generate  $\mathbf{X}$  with density  $f$ , generate  $\mathbf{Y}$  from  $r$  and  $U$  uniform over  $(0, 1)$  independent of  $\mathbf{Y}$ , until  $Uar(\mathbf{Y}) \leq \kappa f(\mathbf{Y})$ , and return  $\mathbf{X} = \mathbf{Y}$ .

There are many situations where one wishes to generate random vectors  $\mathbf{X}$  from quite complicated distributions and no efficient method is available to do it exactly. One very important approach that often permit one to generate  $\mathbf{X}$  approximately from the target distribution is the *Markov chain Monte Carlo* (MCMC) method. In a nutshell, it constructs an artificial Markov chain whose steady-state distribution is the target distribution of  $\mathbf{X}$ , and runs the Markov chain for a large number of steps, until it is deemed sufficiently close to steady-state. Then the state of the chain has a distribution close to the target one. MCMC is covered elsewhere in this handbook.

## Acknowledgements

This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and a Canada Research Chair to the author. Wolfgang Hörmann, Josef Leydold, François Panneton, and Richard Simard made helpful comments and corrections on an earlier draft. The author has been asked to write chapters on Random Number Generation for several handbooks and encyclopedia over the years. Inevitably, there is a large amount of duplication between these chapters.

## References

- Aiello, W., S. Rajagopalan, and R. Venkatesan. 1998. Design of practical and provably good random number generators. *Journal of Algorithms* 29 (2): 358–389.
- Asmussen, S., and P. W. Glynn. 2007. *Stochastic simulation*. New York: Springer-Verlag.
- Blum, L., M. Blum, and M. Schub. 1986. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing* 15 (2): 364–383.
- Bratley, P., B. L. Fox, and L. E. Schrage. 1987. *A guide to simulation*. Second ed. New York, NY: Springer-Verlag.
- Brown, M., and H. Solomon. 1979. On combining pseudorandom number generators. *Annals of Statistics* 1:691–695.
- Chen, H. C., and Y. Asau. 1974. On generating random variates from an empirical distribution. *AIEE Transactions* 6:163–166.
- Cheng, R. C. H. 1998. Random variate generation. In *Handbook of Simulation*, ed. J. Banks, 139–172. Wiley, chapter 5.
- Collings, B. J. 1987. Compound random number generators. *Journal of the American Statistical Association* 82 (398): 525–527.
- Conway, J. H., and N. J. A. Sloane. 1999. *Sphere packings, lattices and groups*. 3rd ed. Grundlehren der Mathematischen Wissenschaften 290. New York: Springer-Verlag.
- Couture, R., and P. L'Ecuyer. 1994. On the lattice structure of certain linear congruential sequences related to AWC/SWB generators. *Mathematics of Computation* 62 (206): 798–808.
- Couture, R., and P. L'Ecuyer. 1996. Orbits and lattices for linear random number generators with composite moduli. *Mathematics of Computation* 65 (213): 189–201.
- Couture, R., and P. L'Ecuyer. 1997. Distribution properties of multiply-with-carry random number generators. *Mathematics of Computation* 66 (218): 591–607.
- Couture, R., and P. L'Ecuyer. 2000. Lattice computations for random numbers. *Mathematics of Computation* 69 (230): 757–765.
- Deng, L.-Y. 2005. Efficient and portable multiple recursive generators of large order. *ACM Transactions on Modeling and Computer Simulation* 15 (1): 1–13.
- Deng, L.-Y., and E. O. George. 1990. Generation of uniform variates from several nearly uniformly distributed variables. *Communications in Statistics* B19 (1): 145–154.
- Deng, L.-Y., and D. K. J. Lin. 2000. Random number generation for the new century. *The American Statistician* 54 (2): 145–150.
- Deng, L.-Y., and H. Xu. 2003. A system of high-dimensional, efficient, long-cycle and portable uniform random number generators. *ACM Transactions on Modeling and Computer Simulation* 13 (4): 299–309.



- Devroye, L. 1986. *Non-uniform random variate generation*. New York, NY: Springer-Verlag.
- Devroye, L. 2006. Nonuniform random variate generation. In *Simulation*, ed. S. G. Henderson and B. L. Nelson, Handbooks in Operations Research and Management Science, 83–121. Amsterdam, The Netherlands: Elsevier. Chapter 4.
- Eichenauer-Herrmann, J. 1995. Pseudorandom number generation by nonlinear methods. *International Statistical Reviews* 63:247–255.
- Eichenauer-Herrmann, J., E. Herrmann, and S. Wegenkittl. 1998. A survey of quadratic and inversive congruential pseudorandom numbers. In *Monte Carlo and Quasi-Monte Carlo Methods 1996*, ed. P. Hellekalek, G. Larcher, H. Niederreiter, and P. Zinterhof, Volume 127 of *Lecture Notes in Statistics*, 66–97. New York, NY: Springer.
- Evans, M., and T. Swartz. 2000. *Approximating integrals via Monte Carlo and deterministic methods*. Oxford, UK: Oxford University Press.
- Fincke, U., and M. Pohst. 1985. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation* 44:463–471.
- Fishman, G. S. 1996. *Monte Carlo: Concepts, algorithms, and applications*. Springer Series in Operations Research. New York, NY: Springer-Verlag.
- Gentle, J. E. 2003. *Random number generation and Monte Carlo methods*. second ed. New York, NY: Springer.
- Goresky, M., and A. Klapper. 2003. Efficient multiply-with-carry random number generators with maximal period. *ACM Transactions on Modeling and Computer Simulation* 13 (4): 310–321.
- Hellekalek, P., and S. Wegenkittl. 2003. Empirical evidence concerning AES. *ACM Transactions on Modeling and Computer Simulation* 13 (4): 322–333.
- Hörmann, W., and J. Leydold. 2000, Dec. Automatic random variate generation for simulation input. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 675–682. Piscataway, NJ: IEEE Press.
- Hörmann, W., and J. Leydold. 2003. Continuous random variate generation by fast numerical inversion. *ACM Transactions on Modeling and Computer Simulation* 13 (4): 347–362.
- Hörmann, W., J. Leydold, and G. Derflinger. 2004. *Automatic nonuniform random variate generation*. Berlin: Springer-Verlag.
- Kinderman, A. J., and J. F. Monahan. 1977. Computer generation of random variables using the ratio of uniform deviates. *ACM Transactions on Mathematical Software* 3:257–260.
- Knuth, D. E. 1998. *The art of computer programming, volume 2: Seminumerical algorithms*. Third ed. Reading, MA: Addison-Wesley.
- Kronmal, R. A., and A. V. Peterson. 1984. An acceptance-complement analogue of the mixture-plus-acceptance-rejection method for generating random variables. *ACM Transactions on Mathematical Software* 10:271–281.
- Lagarias, J. C. 1993. Pseudorandom numbers. *Statistical Science* 8 (1): 31–39.
- Law, A. M., and W. D. Kelton. 2000. *Simulation modeling and analysis*. Third ed. New York, NY: McGraw-Hill.
- L’Ecuyer, P. 1990. Random numbers for simulation. *Communications of the ACM* 33 (10): 85–97.
- L’Ecuyer, P. 1994. Uniform random number generation. *Annals of Operations Research* 53:77–120.
- L’Ecuyer, P. 1996a. Combined multiple recursive random number generators. *Operations Research* 44 (5): 816–822.
- L’Ecuyer, P. 1996b. Maximally equidistributed combined Tausworthe generators. *Mathematics of Computation* 65 (213): 203–213.
- L’Ecuyer, P. 1997. Bad lattice structures for vectors of non-successive values produced by some linear recurrences. *INFORMS Journal on Computing* 9 (1): 57–60.
- L’Ecuyer, P. 1998. Random number generation. In *Handbook of Simulation*, ed. J. Banks, 93–137. Wiley. chapter 4.
- L’Ecuyer, P. 1999a. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research* 47 (1): 159–164.
- L’Ecuyer, P. 1999b. Tables of linear congruential generators of different sizes and good lattice structure. *Mathematics of Computation* 68 (225): 249–260.
- L’Ecuyer, P. 1999c. Tables of maximally equidistributed combined LFSR generators. *Mathematics of Computation* 68 (225): 261–269.

- L'Ecuyer, P. 2001. Software for uniform random number generation: Distinguishing the good and the bad. In *Proceedings of the 2001 Winter Simulation Conference*, 95–105. Piscataway, NJ: IEEE Press.
- L'Ecuyer, P. 2004. Polynomial integration lattices. In *Monte Carlo and Quasi-Monte Carlo Methods 2002*, ed. H. Niederreiter, 73–98. Berlin: Springer-Verlag.
- L'Ecuyer, P. 2008. *SSJ: A Java library for stochastic simulation*. Software user's guide, available at <http://www.iro.umontreal.ca/~lecuyer>.
- L'Ecuyer, P. 2009. Quasi-Monte Carlo methods with applications in finance. *Finance and Stochastics* 13 (3): 307–349.
- L'Ecuyer, P., and T. H. Andres. 1997. A random number generator based on the combination of four LCGs. *Mathematics and Computers in Simulation* 44:99–107.
- L'Ecuyer, P., F. Blouin, and R. Couture. 1993. A search for good multiple recursive random number generators. *ACM Transactions on Modeling and Computer Simulation* 3 (2): 87–98.
- L'Ecuyer, P., and S. Côté. 1991. Implementing a random number package with splitting facilities. *ACM Transactions on Mathematical Software* 17 (1): 98–111.
- L'Ecuyer, P., and R. Couture. 1997. An implementation of the lattice and spectral tests for multiple recursive linear random number generators. *INFORMS Journal on Computing* 9 (2): 206–217.
- L'Ecuyer, P., and J. Granger-Piché. 2003. Combined generators with components from different families. *Mathematics and Computers in Simulation* 62:395–404.
- L'Ecuyer, P., and P. Hellekalek. 1998. Random number generators: Selection criteria and testing. In *Random and Quasi-Random Point Sets*, ed. P. Hellekalek and G. Larcher, Volume 138 of *Lecture Notes in Statistics*, 223–265. New York, NY: Springer-Verlag.
- L'Ecuyer, P., and C. Lemieux. 2000. Variance reduction via lattice rules. *Management Science* 46 (9): 1214–1235.
- L'Ecuyer, P., and C. Lemieux. 2002. Recent advances in randomized quasi-Monte Carlo methods. In *Modeling Uncertainty: An Examination of Stochastic Theory, Methods, and Applications*, ed. M. Dror, P. L'Ecuyer, and F. Szidarovszky, 419–474. Boston: Kluwer Academic.
- L'Ecuyer, P., M. Mandjes, and B. Tuffin. 2009. Importance sampling and rare event simulation. In *Rare Event Simulation Using Monte Carlo Methods*, ed. G. Rubino and B. Tuffin, 17–38. Wiley. Chapter 2.
- L'Ecuyer, P., and F. Panneton. 2002. Construction of equidistributed generators based on linear recurrences modulo 2. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, ed. K.-T. Fang, F. J. Hickernell, and H. Niederreiter, 318–330. Berlin: Springer-Verlag.
- L'Ecuyer, P., and F. Panneton. 2009.  $\mathbf{F}_2$ -linear random number generators. In *Advancing the Frontiers of Simulation: A Festschrift in Honor of George Samuel Fishman*, ed. C. Alexopoulos, D. Goldsman, and J. R. Wilson, 169–193. New York: Springer-Verlag.
- L'Ecuyer, P., and R. Proulx. 1989, Dec. About polynomial-time “unpredictable” generators. In *Proceedings of the 1989 Winter Simulation Conference*, 467–476: IEEE Press.
- L'Ecuyer, P., and R. Simard. 1999. Beware of linear congruential generators with multipliers of the form  $a = \pm 2^q \pm 2^r$ . *ACM Transactions on Mathematical Software* 25 (3): 367–374.
- L'Ecuyer, P., and R. Simard. 2001. On the performance of birthday spacings tests for certain families of random number generators. *Mathematics and Computers in Simulation* 55 (1–3): 131–137.
- L'Ecuyer, P., and R. Simard. 2007, August. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software* 33 (4): Article 22.
- L'Ecuyer, P., R. Simard, E. J. Chen, and W. D. Kelton. 2002. An object-oriented random-number package with many long streams and substreams. *Operations Research* 50 (6): 1073–1075.
- L'Ecuyer, P., R. Simard, and S. Wegenkittl. 2002. Sparse serial tests of uniformity for random number generators. *SIAM Journal on Scientific Computing* 24 (2): 652–668.
- L'Ecuyer, P., and S. Tezuka. 1991. Structural properties for two classes of combined random number generators. *Mathematics of Computation* 57 (196): 735–746.
- L'Ecuyer, P., and R. Touzin. 2000. Fast combined multiple recursive generators with multipliers of the form  $a = \pm 2^q \pm 2^r$ . In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 683–689. Piscataway, NJ: IEEE Press.
- L'Ecuyer, P., and R. Touzin. 2004. On the Deng-Lin random number generators and related methods. *Statistics and Computing* 14:5–9.
- Leeb, H. 1995. Random numbers for computer simulation. Master's thesis, University of Salzburg.

- Lemieux, C., and P. L'Ecuyer. 2003. Randomized polynomial lattice rules for multivariate integration and simulation. *SIAM Journal on Scientific Computing* 24 (5): 1768–1789.
- Leydold, J. 2000. Automatic sampling with the ratio-of-uniform method. *ACM Transactions on Mathematical Software* 26 (1): 78–98.
- Leydold, J. 2009. *UNU.RAN—universal non-uniform random number generators*. Available at <http://statmath.wu.ac.at/unuran/>.
- Luby, M. 1996. *Pseudorandomness and cryptographic applications*. Princeton: Princeton University Press.
- Lüscher, M. 1994. A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Communications* 79:100–110.
- Marsaglia, G. 1985. A current view of random number generators. In *Computer Science and Statistics, Sixteenth Symposium on the Interface*, 3–10. North-Holland, Amsterdam: Elsevier Science Publishers.
- Marsaglia, G. 1996. The Marsaglia random number CDROM including the DIEHARD battery of tests of randomness. See <http://stat.fsu.edu/pub/diehard>.
- Marsaglia, G., and A. Zaman. 1991. A new class of random number generators. *The Annals of Applied Probability* 1:462–480.
- Marsaglia, G., A. Zaman, and J. C. W. Marsaglia. 1994. Rapid evaluation of the inverse normal distribution function. *Statistics and Probability Letters* 19:259–266.
- Matsumoto, M., and Y. Kurita. 1992. Twisted GFSR generators. *ACM Transactions on Modeling and Computer Simulation* 2 (3): 179–194.
- Matsumoto, M., and Y. Kurita. 1994. Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation* 4 (3): 254–266.
- Matsumoto, M., and Y. Kurita. 1996. Strong deviations from randomness in  $m$ -sequences based on trinomials. *ACM Transactions on Modeling and Computer Simulation* 6 (2): 99–106.
- Matsumoto, M., and T. Nishimura. 1998. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* 8 (1): 3–30.
- Nelsen, R. B. 1999. *An introduction to copulas*, Volume 139 of *Lecture Notes in Statistics*. New York, NY: Springer-Verlag.
- Nelson, B. L., and M. Yamnitsky. 1998. Input modeling tools for complex problems. In *Proceedings of the 1998 Winter Simulation Conference*, 105–112. Piscataway, NJ: IEEE Press.
- Niederreiter, H. 1992. *Random number generation and quasi-Monte Carlo methods*, Volume 63 of *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*. Philadelphia, PA: SIAM.
- Niederreiter, H. 1995. The multiple-recursive matrix method for pseudorandom number generation. *Finite Fields and their Applications* 1:3–30.
- Niederreiter, H., and I. E. Shparlinski. 2002. Recent advances in the theory of nonlinear pseudorandom number generators. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, ed. K.-T. Fang, F. J. Hickernell, and H. Niederreiter, 86–102. Berlin: Springer-Verlag.
- Nishimura, T. 2000. Tables of 64-bit Mersenne twisters. *ACM Transactions on Modeling and Computer Simulation* 10 (4): 348–357.
- Panneton, F., and P. L'Ecuyer. 2004. Random number generators based on linear recurrences in  $F_{2^w}$ . In *Monte Carlo and Quasi-Monte Carlo Methods 2002*, ed. H. Niederreiter, 367–378. Berlin: Springer-Verlag.
- Panneton, F., and P. L'Ecuyer. 2005. On the xorshift random number generators. *ACM Transactions on Modeling and Computer Simulation* 15 (4): 346–361.
- Panneton, F., P. L'Ecuyer, and M. Matsumoto. 2006. Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software* 32 (1): 1–16.
- Read, T. R. C., and N. A. C. Cressie. 1988. *Goodness-of-fit statistics for discrete multivariate data*. Springer Series in Statistics. New York, NY: Springer-Verlag.
- Rukhin, A., J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. 2001. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST special publication 800-22, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA. See <http://csrc.nist.gov/rng/>.
- Tausworthe, R. C. 1965. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation* 19:201–209.
- Tezuka, S. 1995. *Uniform random numbers: Theory and practice*. Norwell, MA: Kluwer Academic Publishers.
- Tezuka, S., and P. L'Ecuyer. 1991. Efficient and portable combined Tausworthe random number generators. *ACM Transactions on Modeling and Computer Simulation* 1 (2): 99–112.

- Tezuka, S., P. L'Ecuyer, and R. Couture. 1994. On the add-with-carry and subtract-with-borrow random number generators. *ACM Transactions of Modeling and Computer Simulation* 3 (4): 315–331.
- Tootill, J. P. R., W. D. Robinson, and D. J. Eagle. 1973. An asymptotically random Tausworthe sequence. *Journal of the ACM* 20:469–481.
- Vattulainen, I., T. Ala-Nissila, and K. Kankaala. 1995. Physical models as tests of randomness. *Physical Review E* 52 (3): 3205–3213.
- von Neumann, J. 1951. Various techniques used in connection with random digits. In *The Monte Carlo Method*, ed. A. S. H. et al., Volume 12, 36–38. National Bureau of Standards, Applied Mathematics Series.
- Walker, A. J. 1977. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software* 3:253–256.
- Wang, D., and A. Compagner. 1993. On the use of reducible polynomials as random number generators. *Mathematics of Computation* 60:363–374.
- Wegenkittl, S., and M. Matsumoto. 1999. Getting rid of correlations among pseudorandom numbers: Discarding versus tempering. *ACM Transactions on Modeling and Computer Simulation* 9 (3): 282–294.
- Wu, P.-C. 1997. Multiplicative, congruential random number generators with multiplier  $\pm 2^{k_1} \pm 2^{k_2}$  and modulus  $2^p - 1$ . *ACM Transactions on Mathematical Software* 23 (2): 255–265.

# Index

- $(t, m, s)$ -net, 11
- $p$ -value, 15
- acceptance-complement method, 19
- add-with-carry, 10
- alias method, 18
- asymptotically random, 11
- automatic methods, 17, 21
- binary search, 17
- birthday spacings test, 15
- characteristic polynomial, 10, 12–14
- collision test, 15
- combined generators, 8, 13
- composition method, 20
- convolution method, 20
- copula, 21
- discrepancy, 3
- dual lattice, 6, 12
- entropy, 15
- equidissection, 11
- equidistribution, 11
- generalized feedback shift register (GFSR), 10, 13
- goodness-of-fit, 3
- hat function, 19
- indexed search, 18
- inversion method, 1, 17
- kernel density estimation, 19
- lagged-Fibonacci generator, 7
- lattice, 5, 11, 15
- linear complexity, 14
- linear congruential generator (LCG), 5, 9, 15
- linear feedback shift register (LFSR), 10, 12, 14, 16
- linear recurrence, 5
- linear recurrence modulo 2, 10
- linear recurrence with carry, 9
- Markov chain Monte Carlo, 21
- matrix linear recurrence, 10
- maximally equidistributed, 11, 14
- Mersenne twister, 10, 13, 14, 16
- Monte Carlo methods, 1
- multinormal distribution, 21
- multiple recursive generator (MRG), 5
- multiple recursive matrix generator, 13
- multiply-with-carry generator, 9
- multivariate distribution, 21
- Poisson distribution, 15
- Poisson process, 20
- polynomial lattice, 12
- polynomial LCG, 12
- primitive polynomial, 5, 10
- pseudorandom number generator, 1
- random number generator, 1
  - approximate factoring, 7
  - combined generators, 8, 13, 14, 16
  - definition, 2
  - figure of merit, 6, 11
  - floating-point implementation, 7
  - implementation, 7, 14
  - jumping ahead, 3, 9, 10
  - non-uniform, 16
  - nonlinear, 14
  - period, 2, 10, 13
  - physical device, 2
  - power-of-two modulus, 8
  - powers-of-two-decomposition, 8
  - purely periodic, 2
  - quality criteria, 3, 16
  - seed, 2
  - state, 2
  - statistical tests, 4, 15
  - streams and substreams, 3, 16
- ratio-of-uniforms method, 20
- rejection method, 19–21
- serial test, 15
- spectral test, 6
- squeeze function, 19
- subtract-with-borrow, 10
- Tausworthe generator, 10, 12
- tempering, 13
- thinning, 20
- transformed density rejection, 19
- twisted GFSR, 10, 13, 14
- uniform distribution, 1
- uniformity measure, 3, 11
- unpredictability, 5
- variance reduction, 17
- WELL generator, 10, 16
- well generator, 13
- xorshift generator, 10