S.G. Henderson and B.L. Nelson (Eds.), *Handbook in OR & MS, Vol. 13*

DOI: 10.1016/S0927-0507(06)13003-0

# Chapter 3
# Uniform Random Number Generation

*Pierre L'Ecuyer*

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

**Abstract**

This chapter covers the basic design principles and methods for uniform random number generators used in simulation. We also briefly mention the connections between these methods and those used to construct highly-uniform point sets for quasi-Monte Carlo integration. The emphasis is on the methods based on linear recurrences modulo a large integer, or modulo 2. This reflects the fact that their mathematical structure is much better understood than other types of generators, and that most generators used in simulation have that form. We discuss the main requirements for a good generator, theoretical figures of merit for certain classes of linear-type generators, implementation issues, nonlinear generators, and statistical testing.

## 1 Introduction

A reliable *(pseudo)random number generator* (RNG) is a basic and essential ingredient for any stochastic simulation. The mathematical theory underlying simulation methods is built over the elegant concepts of probability space and random variable. However, since the exact implementation of these concepts on conventional computers seems impossible, random variables and other random objects are *simulated* by *deterministic algorithms*. The aim of these algorithms is to produce sequences of numbers or objects whose behavior is hard to distinguish from that of their "truly random" counterparts, at least for the application of interest. The details of these requirements may differ depending on the context. For the (Monte Carlo) simulation methods discussed in this handbook, the main goal is to reproduce the statistical properties on which these methods are based, so that the estimators of interest behave as expected. For gaming 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.

Random variate generation for simulation can be decomposed 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.

Step (2) is examined in Chapters 4 and 5. This chapter is devoted to algorithms used for Step (1).

In Section 2 we give a definition and the main requirements of a good uniform RNG. In Section 3 we cover RNGs defined by a linear recurrence modulo a large integer $m$. We examine their lattice structure, quality criteria, and implementation techniques. In Section 4 we provide a similar coverage for RNGs based on linear recurrences modulo 2 and examine the relationships between these two types of constructions. One example is given in each of these two sections. Nonlinear RNGs are briefly mentioned in Section 5. In Section 6 we discuss empirical statistical testing of RNGs. Additional textbooks and tutorial-like references on uniform RNGs include Knuth (1998), L'Ecuyer (1994, 1998), Niederreiter (1992) and Tezuka (1995).

## 2 Uniform random number generators

### 2.1 Generators based on a deterministic recurrence

RNGs used for simulation are almost always based on deterministic algorithms that fit the following framework (L'Ecuyer, 1994): an 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*) $s_0$, $f : \mathcal{S} \to \mathcal{S}$ is the *transition function*, $\mathcal{U}$ is the *output space* and $g : \mathcal{S} \to \mathcal{U}$ is the *output function*. In what follows, we assume that $\mathcal{U} = (0, 1)$. The state of the RNG evolves according to the recurrence $s_i = f(s_{i-1})$, for $i \geqslant 1$, and the *output* at step $i$ is $u_i = g(s_i) \in \mathcal{U}$. The output values $u_0, u_1, u_2, \ldots$ are the so-called *random numbers* produced by the RNG.

Because the state space $\mathcal{S}$ is finite, there are necessarily finite integers $l \geqslant 0$ and $j > 0$ such that $s_{l+j} = s_l$. Then, for all $i \geqslant l$, one has $s_{i+j} = s_i$ and $u_{i+j} = u_i$, because both $f$ and $g$ are deterministic. This means that the state and output sequences are eventually periodic. The smallest positive $j$ for which this happens is called the *period length* of the RNG, and is denoted by $\rho$. When $l = 0$, the sequence is said to be *purely periodic*. Obviously, the period length $\rho$ cannot exceed $|\mathcal{S}|$, the cardinality of the state space. Good RNGs are designed so that their period length $\rho$ is not far from that upper bound. For general recurrences, $\rho$ may depend on the seed $s_0$, but good RNGs are normally designed so that $\rho$ is the same for all admissible seeds.

In practice, it is important that the output be *strictly* between 0 and 1, because the transformations that generate nonuniform variates sometimes take infinite values when $U$ is 0 or 1. For example, an exponential random variate $X$ with mean 1 is usually generated via $X = -\ln(1 - U)$ and this gives $X = \infty$ when $U = 1$. All good implementations never produce 0 or 1. 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 a significant difference in the mathematical structure of the generator.

## 2.2 Quality criteria

What are the most important quality criteria to be considered when designing an RNG? An extremely *long period* is essential, to make sure that no wrap-around over the cycle can occur. The length of the period must be guaranteed by a mathematical proof. The RNG must also be *efficient* (run fast and use little 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., 2002a). These virtual generators can be used on parallel processors or to support different sources of randomness in a large simulation model, for example (see Chapter 7 for further discussion).

It is important to realize, however, that these elementary properties are *far from sufficient*. As a simple illustration, consider an RNG with state space $\mathcal{S} = \{0, \ldots, 2^{10000} - 1\}$, transition function $s_{i+1} = f(s_i) = (s_i + 1) \bmod 2^{10000}$ and $u_i = g(s_i) = s_i/2^{10000}$. This RNG has the huge period length $2^{10000}$ and enjoys all the nice properties described in the preceding paragraph, but it is certainly *not* imitating "randomness". The analysis outlined in the following paragraphs, although admittedly heuristic, goes a little deeper.

A sequence of real-valued random variables $u_0, u_1, u_2, \ldots$ are i.i.d. U(0, 1) if and only if for all integers $i \geqslant 0$ and $t > 0$, the vector $\mathbf{u}_{i,t} = (u_i, \ldots, u_{i+t-1})$ 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

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

which is the *finite set* of all vectors of $t$ successive output values that can be produced, from all possible initial states. 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 that the seed $s_0$ is selected randomly, uniformly over $\mathcal{S}$. 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 $s_0$, $\mathbf{u}_{i,t} = (u_i, \ldots, u_{i+t-1})$ is also uniformly distributed over $\Psi_t$ for all $i \geqslant 0$. Since the goal is to approximate the uniform distribution over the unit hypercube $(0, 1)^t$, it becomes clear that the set $\Psi_t$ should provide a good uniform coverage of this hypercube. In other words, $\Psi_t$ acts as an *approximation* of the theoretical *sample space* $(0, 1)^t$ from which the vectors of successive output values are supposed to be drawn randomly. The vectors are generated uniformly over $\Psi_t$ rather than over $(0, 1)^t$. To design good-quality RNGs, we must therefore have practical and effective methods for measuring the uniformity of the corresponding sets $\Psi_t$, even when they have huge cardinalities. It is true that to have a long period, we need a large state space $\mathcal{S}$. However, a much more important reason for requesting a large number of states is to have a larger set $\Psi_t$, which can be used to obtain a better coverage of the unit hypercube $[0, 1)^t$.

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

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

where $I = \{i_1, \ldots, i_t\}$ is a fixed set of nonnegative integers such that $0 \leqslant i_1 < \cdots < i_t$. As a special case, for $I = \{0, \ldots, t-1\}$, we recover $\Psi_t = \Psi_I$.

The uniformity of $\Psi_I$ is typically assessed by measuring the *discrepancy* between the empirical distribution of its points and the uniform distribution over $(0, 1)^t$ (Hellekalek and Larcher, 1998; L'Ecuyer and Lemieux, 2002; Niederreiter, 1992). Discrepancy measures are equivalent to goodness-of-fit test statistics for the multivariate uniform distribution. They can be defined in many ways. The choice of a specific definition typically depends on the mathematical structure of the RNG (different measures are used for different types of RNGs), the reason being that we must be able to compute these uniformity measures quickly even when $\mathcal{S}$ is very large. This 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 some 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. Concrete examples of figures of merit are given later on, for specific types of RNGs.

Generating a random $s_0$ uniformly over $\mathcal{S}$ can be implemented (approximately) by using a physical device. However, for most practical simulation applications and robust RNGs, just picking an arbitrary $s_0$ would suffice.

### *2.3 Links with highly-uniform point sets for quasi-Monte Carlo integration*

Point sets $\Psi_t$ that are highly uniform in the $t$-dimensional unit hypercube are also used for purposes other than imitating randomness. Another major

application is *quasi-Monte Carlo* (QMC) integration, where the integral of a function $f$ over $[0, 1)^t$ is approximated by the average of $f$ over the full point set $\Psi_t$ (see Chapter 12). Usually, the point set $\Psi_t$ is randomized in a way that (1) each individual point has the uniform distribution over $[0, 1)^t$, so the value of $f$ at that (random) point is an unbiased estimator of the integral, and (2) the high uniformity of the point set as a whole is preserved. Under certain conditions, this reduces the variance of the average (which is an unbiased estimator of the integral) by inducing a negative correlation between the different evaluations of $f$.

The point sets used for QMC can actually be constructed in pretty much the same way as RNGs. Their uniformity can also be assessed by the same criteria. In fact, many of the criteria mentioned near the end of the previous subsection were originally introduced for QMC point sets. One general class of construction methods simply consists in selecting an RNG based on a recurrence over a small set of states $\mathcal{S}$ and adopting the corresponding point set $\Psi_t$. We call it a *recurrence-based* point set. In principle, any of the RNG construction methods discussed in the forthcoming sections could be used to define the recurrence. However, since the size of $\mathcal{S}$ must be kept small, all the techniques whose aim is to obtain efficient implementations of long-period generators become irrelevant. So recurrence-based point sets are usually defined via quite simple linear recurrences using modular arithmetic. Two primary examples are the Korobov lattice rules and the recurrences defined via linear feedback shift registers (see Sections 3 and 4). These methods turn out to be special cases of the two main classes of QMC point set construction techniques: *lattice rules* and *digital nets* (see Chapter 12).

### 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 $u_0, u_1, \ldots$ 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.

It is impossible to build an RNG that *passes all statistical tests*. Consider, for example, the class of all tests that examine the first (most significant) $b$ bits of $n$ successive output values, $u_0, \ldots, u_{n-1}$, 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 tests in this class that return $X = 1$ for *exactly* $\alpha 2^{nb}$ of the $2^{nb}$ possible output sequences. The sequence is said to *fail* the test when $X = 1$. The set $\mathcal{T}_{n,b,\alpha}$ is the set of all statistical tests of (exact) level $\alpha$. The number of tests in this set 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, every output sequence fails exactly the same number of tests! This result, pointed out by Leeb (1995), should not be surprising. 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 leads to a dead end. For statistical testing of RNG sequences to be meaningful, all tests should not be considered on equal footing. So which ones are more important? Any answer is certainly tainted with its share of 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, whenever possible, to apply 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). Further discussion of statistical testing for RNGs is given in Section 6.

## 3 Linear recurrences modulo $m$

### 3.1 The multiple recursive generator

The most widely used class of RNGs is based on the general linear recurrence

$$x_i = (a_1 x_{i-1} + \cdots + a_k x_{i-k}) \bmod m, \tag{1}$$

where $m$ and $k$ are positive integers called the *modulus* and the *order*, and the *coefficients* $a_1, \ldots, a_k$ are in $\mathbb{Z}_m$, interpreted as the set $\{0, \ldots, m-1\}$ on which all operations are performed with reduction modulo $m$. The *state* at step $i$ is $s_i = \mathbf{x}_i = (x_{i-k+1}, \ldots, x_i)^\mathsf{T}$ (where "T" means "transposed"). 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 length reaches $\rho = m^k - 1$, the largest possible value (Knuth, 1998). This maximal period length is achieved if and only if the

characteristic polynomial of the recurrence, $P(z) = z^k - a_1 z^{k-1} - \cdots - a_k$, is a primitive polynomial over $\mathbb{Z}_m$, i.e., if and only if the smallest positive integer $\nu$ such that $(z^\nu \bmod P(z)) \bmod m = 1$ (interpreted as a constant polynomial) 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

$$x_n = (a_r x_{n-r} + a_k x_{n-k}) \bmod m. \tag{2}$$

A *multiple recursive generator* (MRG) uses (1) 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), which was the standard in most simulation software products and textbooks until a few years ago, usually with $m = 2^{31} - 1$. These LCGs have too short a period ($\rho$ is at most $m - 1$) and too coarse a structure of their point set $\Psi_t$ to be used as reliable RNGs (see Section 6). They should simply be discarded. On the other hand, small LCGs can be used to construct QMC point sets which are a special case of lattice rules (see Chapter 12).

In practice, the output function of MRGs 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. To simplify the theoretical analysis, here we follow the usual 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 $x_{i,0}, x_{i,1}, x_{i,2}, \ldots$ the values of $x_0, x_1, x_2, \ldots$ produced by the recurrence (1) when the initial state $\mathbf{x}_0$ is $\mathbf{e}_i$. An arbitrary initial state $\mathbf{x}_0 = (z_1, \ldots, z_k)^\mathsf{T}$ can be written as the linear combination $z_1\mathbf{e}_1 + \cdots + z_k\mathbf{e}_k$ and the corresponding sequence is a linear combination of the sequences $(x_{i,0}, x_{i,1}, \ldots)$, 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 $t \geqslant 1$, $\Psi_t = L_t \cap [0, 1)^t$, where

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

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

$$\mathbf{v}_1 = \frac{(1, 0, \ldots, 0, x_{1,k}, \ldots, x_{1,t-1})^\mathsf{T}}{m},$$

$$\vdots$$

$$\mathbf{v}_k = \frac{(0, 0, \ldots, 1, x_{k,k}, \ldots, x_{k,t-1})^\mathsf{T}}{m},$$

$$\mathbf{v}_{k+1} = (0, 0, \ldots, 0, 1, \ldots, 0)^{\mathsf{T}},$$
$$\vdots$$
$$\mathbf{v}_t = (0, 0, \ldots, 0, 0, \ldots, 1)^{\mathsf{T}}.$$

For $t \leqslant k$, $L_t$ 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 regular pattern, in equidistant parallel hyperplanes. Graphical illustrations of this lattice structure can be found in several papers and books; e.g., Gentle (2003), Knuth (1998), Law and Kelton (2000) and L'Ecuyer (1998). Define the *dual lattice* to $L_t$ as

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

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 (where $\|\cdot\|_2$ denotes the Euclidean norm), 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 nonzero 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 that 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* (Fishman, 1996; Knuth, 1998).

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 $I = \{i_1, \ldots, i_t\}$. 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 L'Ecuyer and Couture (1997). Finding the shortest nonzero vector in a lattice with basis $\mathbf{v}_1, \ldots, \mathbf{v}_t$ can be formulated as an integer programming problem with a quadratic objective function

$$\text{Minimize } \|\mathbf{v}\|_2^2 = \sum_{i=1}^{t}\sum_{j=1}^{t} z_i \mathbf{v}_i^{\mathsf{T}}\mathbf{v}_j z_j$$

subject to $z_1, \ldots, z_t$ 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

$$M_{\mathcal{J}} = \min_{I \in \mathcal{J}} \frac{\ell_I}{\ell^*_{|I|}(m^k)}. \tag{3}$$

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 $I = \{1, \ldots, t\}$ for $t \leqslant t_1$, 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 $\mathcal{J} = \{\{0, 1, \ldots, i\}\colon i < t_1\} \cup \{\{i_1, i_2\}\colon 0 = i_1 < i_2 < t_2\} \cup \cdots \cup \{\{i_1, \ldots, i_d\}\colon 0 = i_1 < \cdots < i_d < t_d\}$ for some positive integers $d, t_1, \ldots, t_d$. 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 $\mathbf{h} = (-a_k, \ldots, -a_1, 1, 0, \ldots, 0)^{\mathsf{T}}$ always belongs 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 (1), which implies that $(-a_k, \ldots, -a_1, 1, 0, \ldots, 0)\mathbf{v}$ must be an integer. Therefore, one always has $\ell_t^2 \leqslant \|\mathbf{h}\|_2^2 = 1 + a_1^2 + \cdots + a_k^2$. Likewise, if $I$ contains 0 and all indices $j$ such that $a_{k-j} \neq 0$, then $\ell_I^2 \leqslant 1 + a_1^2 + \cdots + a_k^2$ (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 (1) 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 $\ell_I^2 \leqslant 3$, 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.

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

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

with $a^2 < m$. If $a$ is small, the bound $\ell_I^2 \leqslant 1 + a^2$ 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

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

In this case, for $I = \{0, k - j_{t-1}, \ldots, k - j_2, k\}$, the vectors $(1, a, \ldots, a)$ and $(a^*, 1, \ldots, 1)$ 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 skipping 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.

We saw that the point set $\Psi_t$ of an LCG or MRG is the intersection of some special lattice $L_t$ with the unit hypercube, where $L_t$ contains all corners of the hypercube. A *lattice rule* is a QMC integration method defined by selecting an arbitrary lattice $L_t$ with this property, and using its intersection with the unit hypercube as a QMC point set. The uniformity of lattice rules can be measured by the spectral test in the same way as MRGs (see Chapter 12).

### *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-hand side of (1) 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 Côté (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 $|a_j|(m-1) \leqslant 2^{53}$, 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

$$y = |a_j|x_{i-j}, \qquad z_j = y - m\left\lfloor \frac{y}{m} \right\rfloor.$$

Similarly, if $(|a_1| + \cdots + |a_k|)(m-1) \leqslant 2^{53}$, $a_1 x_{i-1} + \cdots + a_k x_{i-k}$ 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

$$y = \left\lfloor \frac{x_{i-j}}{q_j} \right\rfloor, \qquad z = |a_j|(x_{i-j} - yq_j) - yr_j,$$
$$\text{if } z < 0 \text{ then } z_j = z + m \text{ else } z_j = z.$$

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 (L'Ecuyer and Simard, 1999; L'Ecuyer and Touzin, 2000; Wu, 1997). 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

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

Suppose now that

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

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). It has been pointed out by L'Ecuyer and Simard (1999) 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 MRGs can be obtained via the power-of-two decomposition method, as explained in L'Ecuyer and Touzin (2000).

Another interesting idea for improving efficiency is to take all nonzero coefficients $a_j$ equal to the same constant $a$ (Deng and Xu, 2003; Marsaglia, 1996). Then, computing the right-hand side of (1) requires a single multiplication. Deng and Xu (2003) provide specific parameter sets and concrete implementations for MRGs of this type, for prime $m$ near $2^{31}$, and $k = 102, 120$ and 1511.

One may be tempted to take $m$ equal to a power of two, say $m = 2^e$, because computing the products and sums modulo $m$ is then much easier: it suffices to keep the $e$ least significant bits of the results. However, taking a power-of-two modulus has very important 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 length of the recurrence (1) cannot exceed $(2^k - 1)2^{e-1}$ if $k > 1$, and $2^{e-2}$ if $k = 1$ and $e \geqslant 4$. The maximal period length achievable with $k = 7$ and $m = 2^{31}$, for example, is more than $2^{180}$ times

smaller than the maximal period length 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) are generally in conflict with those required for having a good lattice structure and statistical robustness. *Combined MRGs* provide one solution to this problem. Consider $J$ distinct MRGs evolving in parallel, based on the recurrences

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

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

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

and

$$w_i = \left(\frac{\delta_1 x_{1,i}}{m_1} + \cdots + \frac{\delta_J x_{J,i}}{m_J}\right) \bmod 1. \tag{8}$$

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

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 length $\rho_j$. Let $m = m_1 \cdots m_J$ and let $\rho$ be the least common multiple of $\rho_1, \ldots, \rho_J$. Under these conditions, the following results have been proved by L'Ecuyer and Tezuka (1991) and L'Ecuyer (1996a): (a) the sequence (8) is exactly equivalent to the output sequence of an MRG with (composite) modulus $m$ and coefficients $a_j$ that can be computed explicitly as explained in L'Ecuyer (1996a); (b) the two sequences in (7) and (8) have period length $\rho$; and (c) if both sequences have the same initial state, then $u_i = w_i + \varepsilon_i$ where $\max_{i \geqslant 0} |\varepsilon_i|$ can be bounded explicitly by a constant $\varepsilon$ 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 length $\rho_j = m_j^k - 1$, 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 (1) 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

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

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 $\mathrm{O}(\log_2 \nu)$ matrix multiplications via a standard divide-and-conquer algorithm (Knuth, 1998):

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

### 3.6 Linear recurrences with carry

The basic idea here is to add a *carry* to the linear recurrence (1). The general form of this RNG, called *multiply-with-carry* (MWC), can be written as

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

$$c_i = \left\lfloor \frac{a_0 x_i + a_1 x_{i-1} + \cdots + a_k x_{i-k} + c_{i-1}}{b} \right\rfloor, \tag{10}$$

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

where $b$ is a positive integer (e.g., a power of two), $a_0, \ldots, a_k$ 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 $s_i = (x_{i-k+1}, \ldots, x_i, c_i)^\mathsf{T}$. 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. These types of recurrences were introduced by Marsaglia and Zaman (1991) to obtain a large period even when $m$ is a power of two (this may allow a faster implementation). They were studied and generalized by Couture and L'Ecuyer (1994, 1997), Goresky and Klapper (2003) and Tezuka et al. (1994).

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 Couture and L'Ecuyer (1997), Goresky and Klapper (2003) and Tezuka et al. (1994) is that if the initial states agree, the output sequence $\{u_i, i \geqslant 0\}$ 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 an LCG with very large modulus. It has been shown by Couture and L'Ecuyer (1997) that the value of $\ell_t$ for this LCG satisfies $\ell_t^2 \leqslant a_0^2 + \cdots + a_k^2$ for $t \geqslant k$, 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 $\ell_t^2 \leqslant 3$ for $t \geqslant k$ 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 length 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 length of $\rho = m - 1$ can be achieved by allowing a more general $a_0$. They provided specific parameters that give a maximal period for $b$ ranging from $2^{21}$ to $2^{35}$ and $\rho$ up to approximately $2^{2521}$.

### *3.7 Computer searches for good parameters and an example*

When searching for specific instances of MRGs with good parameters, one would usually impose constraints on the parameters $k$, $m$ and $a_j$'s (for each component in the case of a combined generator). These constraints are based on implementation efficiency considerations. One of the constraints might be that the MRG (or each component) has maximal period length. The constraints determine a set of feasible solutions in parameter space. A figure of merit measuring the uniformity of the MRG point set, such as $M_{\mathcal{J}}$ in (3) for some set $\mathcal{J}$, is also selected. Then, an "intelligent" random search algorithm (that usually employs several heuristics) is used to find a feasible solution with the largest possible figure of merit. Such computer searches can take days of CPU time, because checking for maximal period and computing the figure of merit is generally very demanding computationally. Nevertheless, unless the constraints are too restrictive, it is typically easy to find good parameter sets by random search, because there is usually a large number of nearly optimal solutions.

As a concrete illustration, consider the combined generator `MRG32k3a` proposed in L'Ecuyer (1999a). It has $J = 2$ components of order $k = 3$ defined as in (6), with $m_1 = 2^{32} - 209$, $a_{11} = 0$, $a_{12} = 1403580$, $a_{13} = -810728$, $m_2 = 2^{32} - 22853$, $a_{21} = 527612$, $a_{22} = 0$, $a_{23} = -1370589$. The combination is defined by $z_i = (x_{1,i} - x_{2,i}) \bmod m_1$ and the MRG that corresponds to this combination has order $k = 3$, modulus $m = m_1 m_2 = 18446645023178547541$ and multipliers $a_1 = 18169668471252892557$, $a_2 = 3186860506199273833$ and $a_3 = 8738613264398222622$. Its period length is $(m_1^3 - 1)(m_2^3 - 1)/2 \approx 2^{191}$.

This generator was found by a computer search with a computing budget of a few days of CPU time, as follows. The values of $J$, $k$, $m_1$ and $m_2$ were fixed. These values of $m_1$ and $m_2$ have special properties explained in L'Ecuyer

(1999a), which make the maximal period length conditions easier to verify. The constraints $a_{11} = a_{22} = 0$ and $(|a_{j,0}| + |a_{j,1}| + |a_{j,2}|)(m_j - 1) < 2^{53}$ were also imposed to make sure that the recurrence of each component was easily implementable in floating-point arithmetic. The figure of merit (to maximize) was $M_{\mathcal{J}}$ with $\mathcal{J} = \{\{0, \ldots, i\}: i < 32\}$ (i.e., the worst-case standardized spectral test value in up to 32 dimensions). The retained generator (given above) has $M_{\mathcal{J}} = 0.6336$. We later verified that $M_{\mathcal{J}} = 0.6225$ if we go up to 45 dimensions instead of 32. Computer codes that implement this particular generator in several languages (such as C, C++, Java) are available from the author's web page. It is also implemented in many commercial simulation packages such as Arena, Automod, Witness, etc.

## 4 Generators based on recurrences modulo 2

### *4.1 A general framework*

It is certainly a good idea to exploit the fact that computers work in binary arithmetic by designing RNGs defined directly in terms of bit strings and sequences. This is the idea underlying the following framework. 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 the following matrix linear recurrence over $\mathbb{F}_2$:

$$\mathbf{x}_i = \mathbf{A}\mathbf{x}_{i-1}, \tag{12}$$

$$\mathbf{y}_i = \mathbf{B}\mathbf{x}_i, \tag{13}$$

$$u_i = \sum_{\ell=1}^{w} y_{i,\ell-1} 2^{-\ell} = y_{i,0} y_{i,1} y_{i,2} \cdots, \tag{14}$$

where $\mathbf{x}_i = (x_{i,0}, \ldots, x_{i,k-1})^{\mathsf{T}} \in \mathbb{F}_2^k$ is the $k$-bit *state vector* at step $i$, $\mathbf{y}_i = (y_{i,0}, \ldots, y_{i,w-1})^{\mathsf{T}} \in \mathbb{F}_2^w$ 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 $\mathbb{F}_2$.

Let

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

be the characteristic polynomial of $\mathbf{A}$, where $\mathbf{I}$ is the identity matrix and each $\alpha_j$ is in $\mathbb{F}_2$. For any sequence of $\mathbf{x}_i$'s that satisfies (12), for each $j$, the sequence $\{x_{i,j}, i \geqslant 0\}$ obeys the linear recurrence

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

(L'Ecuyer, 1994; Niederreiter, 1992). The sequences $\{y_{i,j}, i \geqslant 0\}$, for $0 \leqslant j < w$, also obey the same recurrence (although they may also follow recurrences of

shorter order in certain situations, depending on **B**). We assume that $\alpha_k = 1$, so that the recurrence (15) has *order* $k$ and is purely periodic. Its period length is $2^k - 1$ (i.e., maximal) if and only if $P(z)$ is a primitive polynomial over $\mathbb{F}_2$ (Knuth, 1998; Niederreiter, 1992).

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. However, this multiplication could becomes unacceptably time consuming when $k$ exceeds a few hundreds.

Several popular classes of RNGs fit this framework as special cases, by appropriate choices of the matrices **A** and **B**. This includes the Tausworthe or LFSR, polynomial LCG, GFSR, twisted GFSR, Mersenne twister, multiple recursive matrix generators, and combinations of these (L'Ecuyer and Panneton, 2002; Matsumoto and Nishimura, 1998; Niederreiter, 1995; Tezuka, 1995). We detail some of them after discussing how to measure their uniformity. The point sets $\Psi_t$ produced by these RNGs generally contain $2^k$ points and turn out to be instances of *digital nets*, which form a large class of construction methods for QMC point sets (Chapter 12). This means that any of these RNG implementation methods can be employed to construct recurrence-based QMC point sets, by taking a small value of $k$.

### 4.2 Measures of equidistribution

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, 2004a; L'Ecuyer and Panneton, 2002; Tezuka, 1995). For an arbitrary vector $\mathbf{q} = (q_1, \ldots, q_t)$ of nonnegative 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 $2^{q_1+\cdots+q_t}$ rectangular boxes of the same size and shape. We call this partition the **q**-equidissection of the unit hypercube.

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

The **q**-equidistribution of $\Psi_I$ depends only on the first $q_j$ bits of $x_{i_j}$ for $1 \leqslant j \leqslant t$, for the points $(x_{i_1}, \ldots, x_{i_t})$ that belong to $\Psi_I$. The vector of these $q_1 + \cdots + q_t = k - q$ bits can always be expressed as a linear function of the $k$ bits of the initial state $\mathbf{x}_0$, i.e., as $\mathbf{M_q x}_0$ for some $(k - q) \times k$ binary matrix $\mathbf{M_q}$, and it is easily seen that $\Psi_I$ is **q**-equidistributed if and only if $\mathbf{M_q}$ has full rank $k - q$. This provides a simple way of checking equidistribution (Fushimi, 1983; L'Ecuyer, 1996b; Tezuka, 1995).

If $\Psi_I$ is $(\ell, \ldots, \ell)$-equidistributed for some $\ell \geqslant 1$, 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 QMC methods, where a different notation is used (Niederreiter, 1992), if it is $(q_1, \ldots, q_t)$-equidistributed in base 2 for all nonnegative integers $q_1, \ldots, q_t$ 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 $t$. This measure is frequently used for QMC point sets, for which $k$ is small (Hellekalek and Larcher, 1998; Niederreiter, 1992). However, when $k - q$ is large, i.e., for long-period generators having good equidistribution, it is extremely difficult to compute because there are too many vectors $\mathbf{q}$ for which equidistribution needs to be checked. In practice, for RNGs, one must settle for figures of merit that involve a smaller number of equidissections.

If $\delta_I = 0$ for all sets $I$ of the form $I = \{0, \ldots, t-1\}$, for $1 \leqslant t \leqslant k$, 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 $\ell \leqslant w$ 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) and Panneton and L'Ecuyer (2004), 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 (2004a), 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

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

where $\mathbb{F}_2[z]$ is the ring of polynomials with coefficients in $\mathbb{F}_2$, and the basis vectors $\mathbf{v}_j(z)$ are in $\mathbb{L}_2^t$. The elements of the dual lattice $\mathcal{L}_t^*$ are the vectors $\mathbf{h}(z)$ in $\mathbb{L}_2^t$ whose scalar product with any vector of $\mathcal{L}_t$ is a polynomial in $\mathbb{F}_2[z]$.

In one setting that applies for instance to LFSR generators, we define the mapping $\varphi : \mathbb{L}_2 \to \mathbb{R}$ by

$$\varphi\left(\sum_{\ell=\omega}^{\infty} x_\ell z^{-\ell}\right) = \sum_{\ell=\omega}^{\infty} x_\ell 2^{-\ell}$$

and it turns out that the point set $\Psi_t$ produced by the generator is equal to $\varphi(\mathcal{L}_t) \cap [0,1)^t$ for some lattice $\mathcal{L}_t$. The general case is covered by defining the lattice in a different way (adopting the *resolutionwise* lattice) as explained in Couture and L'Ecuyer (2000). Moreover, the equidistribution properties examined in Section 4.2 can be expressed in terms of lengths of shortest vectors in the dual lattice, with appropriate definitions of the length (or norm). Much of the theory and algorithms developed for lattices in the real space can be adapted to these new types of lattices (Couture and L'Ecuyer, 2000; L'Ecuyer, 2004a; Lemieux and L'Ecuyer, 2003; Panneton, 2004; Tezuka, 1995).

### 4.4 The LFSR generator

*Linear feedback shift register* (LFSR) (or *Tausworthe*) generators (L'Ecuyer, 1996b; Tausworthe, 1965; Tezuka, 1995) are a special case of (12)–(14) with $\mathbf{A} = \mathbf{A}_0^s$ (in $\mathbb{F}_2$) for some positive integer $s$, where

$$\mathbf{A}_0 = \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ a_k & a_{k-1} & \dots & a_1 \end{pmatrix}, \tag{16}$$

$a_1, \dots, a_k$ are in $\mathbb{F}_2$, $a_k = 1$, and all blank entries in the matrix are zeros. We take $w \leqslant k$ and the matrix $\mathbf{B}$ contains the first $w$ lines of the $k \times k$ identity matrix. The RNG thus obtained can be defined equivalently by

$$x_i = a_1 x_{i-1} + \cdots + a_k x_{i-k} \bmod 2, \tag{17}$$

$$u_i = \sum_{\ell=1}^{w} x_{is+\ell-1} 2^{-\ell}, \tag{18}$$

where $x_{is+\ell-1} = x_{i,\ell-1}$. Here, $P(z)$ is not the characteristic polynomial of the recurrence (17), but the characteristic polynomial of the matrix $\mathbf{A}_0^s$. The choice of $s$ has an important impact on the quality of the generator. A common special case is when a single $a_j$ is nonzero in addition to $a_k$; then $P(z)$ is a trinomial and we say that we have a *trinomial-based* LFSR generator. These generators are known to have important statistical deficiencies (Matsumoto and Kurita, 1996; Tezuka, 1995) but they can be used as components of combined RNGs (see Section 4.6).

LFSR generators can be expressed as LCGs in a space of polynomials (L'Ecuyer, 1994; Tezuka and L'Ecuyer, 1991; Tezuka, 1995). With this representation, their lattice structure as discussed in Section 4.3 follows immediately.

### 4.5 The GFSR and twisted GFSR

Here we take $\mathbf{A}$ as the $pq \times pq$ matrix

$$\mathbf{A} = \begin{pmatrix} & & \mathbf{I}_p & & \mathbf{S} \\ \mathbf{I}_p & & & & \\ & \mathbf{I}_p & & & \\ & & \ddots & & \\ & & & \mathbf{I}_p & \end{pmatrix}$$

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 length cannot exceed $2^q - 1$, because each bit of $\mathbf{x}_i$ follows the same binary recurrence of order $k = q$, with characteristic polynomial $P(z) = z^q - z^{q-r} - 1$.

More generally, we can define $\mathbf{x}_i$ as the bitwise exclusive-or of $\mathbf{x}_{i-r_1}$, $\mathbf{x}_{i-r_2}, \ldots, \mathbf{x}_{i-r_d}$, 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 length is still bounded by $2^q - 1$, whereas considering the $pq$-bit state, we should rather expect a period length 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 $P_S(z) = z^p - a_p z^{p-1} - \cdots - a_1$ is the characteristic polynomial of $\mathbf{S}$, 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 length $2^k - 1$. Tezuka (1994) pointed out important weaknesses of the original TGFSR and Matsumoto and Kurita (1994) 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 proposed by Matsumoto and Nishimura (1998) is fast, robust, has the huge period length of $2^{19937} - 1$, and has become quite popular.

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 output 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 $j = 1, \ldots, J$. The output of the combined generator at step $i$ is defined by

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

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

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

The TGFSR and Mersenne twister generators proposed in Matsumoto and Kurita (1994), Matsumoto and Nishimura (1998), and Nishimura (2000) cannot be maximally equidistributed. L'Ecuyer and Panneton (2002), on the other hand, have constructed concrete examples of maximally equidistributed combined TGFSR generators, with period lengths near $2^{466}$ and $2^{1250}$. 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.

### 4.7 An example

Consider the combined generator with $J = 4$ LFSR components whose recurrences (17) have the following characteristic polynomials: $P_1(z) = z^{31} - z^6 - 1$, $P_2(z) = z^{29} - z^2 - 1$, $P_3(z) = z^{28} - z^{13} - 1$, and $P_4(z) = z^{25} - z^3 - 1$, and whose values of $s$ in (18) are $s_1 = 18$, $s_2 = 2$, $s_3 = 7$ and $s_4 = 13$, respectively. The corresponding combined LFSR generator has a characteristic polynomial $P(z) = P_1(z)P_2(z)P_3(z)P_4(z)$ of degree 113, with 58 coefficients equal to 0 and 55 equal to 1, and its period length is

$(2^{31}-1)(2^{29}-1)(2^{28}-1)(2^{25}-1) \approx 2^{113}$. This combined generator is also maximally equidistributed (as defined in Section 4.2). An implementation in C is given in L'Ecuyer (1999c), under the name of `lfsr113`. This generator is faster than `MRG32k3a`: it needs approximately 30 seconds to produce $10^9$ (one billion) uniform random numbers on a 2.8 GHz Athlon-based computer, compared to approximately 100 seconds for `MRG32k3a`.

Its parameters were selected as follows. The degrees of the characteristic polynomials $P_j(z)$ were fixed at $k_1 = 31$, $k_2 = 29$, $k_3 = 28$ and $k_4 = 25$, and these polynomials were required to be primitive trinomials of the form $P_j(z) = z^{k_j} - z^{q_j} - 1$ with $0 < 2q_j < k_j$ and with step size $s_j$ satisfying $0 < s_j \leqslant k_j - q_j < k_j \leqslant w = 32$ and $\gcd(s_j, 2^{k_j} - 1) = 1$. Components that satisfy these conditions have maximal period length $2^{k_j} - 1$ and can be implemented efficiently as described in L'Ecuyer (1996b). These values of $k_j$ were selected so that the period lengths $2^{k_j} - 1$ of the components have no common factor, which implies that the period length of the combined generator is their product. An exhaustive search was performed to find all parameter values that satisfy these conditions; there are approximately 3.28 million. Among them, there are 4744 for which the combined generator is maximally equidistributed and also collision-free (which means that when the number of points does not exceed the number of boxes in the equidissection, there is never more than one point in a box). The `lfsr113` generator given above is one of them.

## 5 Nonlinear RNGs

The linear RNGs discussed so far have point sets $\Psi_t$ with a very regular structure. To get away from this regularity, one can either use a nonlinear transition function $f$, or keep the transition function linear but use a nonlinear output function $g$. Several types of nonlinear RNGs have been proposed over the years; see, e.g., Blum et al. (1986), Eichenauer-Herrmann (1995), Eichenauer-Herrmann et al. (1998), Hellekalek and Wegenkittl (2003), Knuth (1998), L'Ecuyer (1994), L'Ecuyer and Granger-Piché (2003) and Niederreiter and Shparlinski (2002). 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 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. In most cases, on the other hand, their statistical properties can be analyzed only empirically or via asymptotic theoretical results. They are generally slower than the linear ones.

Various ways of *combining* RNGs also give rise to nonlinear RNGs whose output sequence shows less regularity; see, e.g., Fishman (1996), Gentle (2003), Knuth (1998), Law and Kelton (2000), L'Ecuyer (1994) and 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. It is important to understand that to assess the quality of the combined generator, one must analyze the mathematical structure of the combined generator itself rather than the structure of its components (L'Ecuyer, 1996b, 1996a; L'Ecuyer and Granger-Piché, 2003; Tezuka, 1995). Otherwise, these combination techniques are heuristics which often improve the uniformity (empirically), but can also make it worse.

## 6 Empirical statistical tests

A statistical test for RNGs can be defined by any 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

$$p_{\mathrm{R}} = \mathrm{P}[X \geqslant x|\mathcal{H}_0] \quad \text{and} \quad p_{\mathrm{L}} = \mathrm{P}[X \leqslant x|\mathcal{H}_0].$$

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.

Statistical tests are defined by partitioning the possible realizations of $(u_0, \ldots, u_{\tau-1})$ 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 simple and natural way of doing that is to take $\tau = t$ (a constant) and cut the interval $[0, 1)$ into $d$ equal segments for some positive integer $d$, to partition the hypercube $[0, 1)^t$ into $k = d^t$ subcubes of volume $1/k$. We then generate $n$ points $\mathbf{u}_i = (u_{ti}, \ldots, u_{ti+t-1}) \in [0, 1)^t$, for $i = 0, \ldots, n-1$, and count the number $N_j$ of points falling in subcube $j$, for $j = 0, \ldots, k-1$. 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., 2002b). They can be *sparse* (if $n/k \ll 1$), or *dense* (if $n/k \gg 1$), or something in between. There are also *overlapping* versions, where the points are defined by $\mathbf{u}_i = (u_i, \ldots, u_{i+t-1})$ for $i = 0, \ldots, n-1$.

For further details, specific instances of serial tests, and other empirical tests commonly applied to RNGs (based, e.g., on close pairs of points among in the space, random walks on the real line or over the integers, the linear complexity of a binary output sequence, the simulation of dice or poker hands, etc.), we refer the reader to (Knuth, 1998; L'Ecuyer and Hellekalek, 1998;

L'Ecuyer and Simard, 2001; L'Ecuyer, 2001; L'Ecuyer et al., 2002b; L'Ecuyer and Simard, 2002; Marsaglia, 1985; 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 $\mathcal{H}_0$. *Test suites* for RNGs include a selection of tests, with predetermined parameters and sample sizes. The best known are DIEHARD (Marsaglia, 1996) and the NIST test suite (Rukhin et al., 2001). The library *TestU01* (L'Ecuyer and Simard, 2002) implements a large selection of tests in the C language and provides a variety of test suites, some designed for U(0, 1) output sequences and others for strings of bits.

## 7 Conclusion, future work and open issues

The ultimate goal of RNG design is to obtain a fast algorithm or device whose output cannot be distinguished in any way from a realization of an infinite sequence of i.i.d. uniform random variables. This requirement is equivalent to passing all possible statistical tests of uniformity and independence. It seems that this can only be achieved through a physical device based on quantum physics. Using this type of device for simulation has several drawbacks, one of them being that the sequence cannot be reproduced without storing it.

RNGs based on linear recurrences and output transformations, on the other hand, are known to fail statistical tests of linear complexity (for obvious reasons), even when their period length is huge. This seems to have no impact for the great majority of relevant discrete-event simulation applications, but it would nevertheless be good to have efficient alternative nonlinear RNGs that also pass these linear complexity tests. Work in that direction has been initiated in L'Ecuyer and Granger-Piché (2003), for instance. In fact, what is needed is a collection of RNGs having different types of structures, different sizes of their state space, for both 32-bit and 64-bit computers, perhaps some faster and some slower but more robust, and where each RNG can provide multiple streams of random numbers as in L'Ecuyer et al. (2002a) (see also Chapter 7). It should also be easy and simple to replace the pseudorandom numbers by (possibly randomized) quasirandom numbers in a simulation.

Work is currently in progress to develop generators with huge period lengths (e.g., near $2^{20000}$ or more) as well as faster generators based on linear recurrences modulo 2 and good equidistribution properties. The huge-period generators are not necessarily the way to go because they require a large amount of memory and managing multiple streams involves much more overhead than for the smaller generators. Their huge periods may also hide rather long bad subsequences, due to the fact that the transition function typically modifies only a small part of their state at each step. For example Panneton and L'Ecuyer (2006) have shown that if the Mersenne twister proposed by Matsumoto and Nishimura (1998) is initialized to a state that contains almost only zeros, then

the fraction of zeros in the state tends to remain very large for several thousand steps. These issues require further study.

Poor (or plain bad) generators can still be found in popular commercial statistical and simulation software, spreadsheets, etc. Do not trust the default RNGs available in these products: many of them are quite unreliable (L'Ecuyer, 2001). Vendors should be pressured to change this state of affairs. Each year, several new RNGs are proposed in the scientific literature or over the Internet. Many of them are based on very little theoretical analysis. An important task of RNG experts is to study these proposals carefully to shed light on their potential weaknesses. This is an area where negative results are often as important to publish as the positive ones.

## Acknowledgements

This text is an adaptation of a chapter entitled "Random Number Generation", written for the *Handbook of Computational Statistics*, published by Springer-Verlag (L'Ecuyer, 2004b). This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Grant No. ODGP0110050, NATEQ–Québec Grant No. 02ER3218, and a Canada Research Chair to the author. François Panneton, Richard Simard, Shane Henderson, Barry Nelson and Christiane Lemieux made helpful comments and corrections on an earlier draft.

## References

Blum, L., Blum, M., Schub, M. (1986). A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing* 15 (2), 364–383.

Conway, J.H., Sloane, N.J.A. (1999). *Sphere Packings, Lattices and Groups*, 3rd edition. *Grundlehren der Mathematischen Wissenschaften*, vol. 290. Springer-Verlag, New York.

Couture, R., L'Ecuyer, P. (1994). On the lattice structure of certain linear congruential sequences related to AWC/SWB generators. *Mathematics of Computation* 62 (206), 798–808.

Couture, R., L'Ecuyer, P. (1997). Distribution properties of multiply-with-carry random number generators. *Mathematics of Computation* 66 (218), 591–607.

Couture, R., L'Ecuyer, P. (2000). Lattice computations for random numbers. *Mathematics of Computation* 69 (230), 757–765.

Deng, L.-Y., Lin, D.K.J. (2000). Random number generation for the new century. *The American Statistician* 54 (2), 145–150.

Deng, L.-Y., Xu, H. (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.

Eichenauer-Herrmann, J. (1995). Pseudorandom number generation by nonlinear methods. *International Statistical Reviews* 63, 247–255.

Eichenauer-Herrmann, J., Herrmann, E., Wegenkittl, S. (1998). A survey of quadratic and inversive congruential pseudorandom numbers. In: Hellekalek, P., Larcher, G., Niederreiter, H., Zinterhof, P. (Eds.), *Monte Carlo and Quasi-Monte Carlo Methods 1996*. *Lecture Notes in Statistics*, vol. 127. Springer-Verlag, New York, pp. 66–97.

Fincke, U., Pohst, M. (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*. Springer-Verlag, New York.

Fushimi, M. (1983). Increasing the orders of equidistribution of the leading bits of the Tausworthe sequence. *Information Processing Letters* 16, 189–192.

Gentle, J.E. (2003). *Random Number Generation and Monte Carlo Methods*, 2nd edition. Springer-Verlag, New York.

Goresky, M., Klapper, A. (2003). Efficient multiply-with-carry random number generators with maximal period. *ACM Transactions on Modeling and Computer Simulation* 13 (4), 310–321.

Hellekalek, P., Larcher, G. (Eds.) (1998). *Random and Quasi-Random Point Sets*. *Lecture Notes in Statistics*, vol. 138. Springer-Verlag, New York.

Hellekalek, P., Wegenkittl, S. (2003). Empirical evidence concerning AES. *ACM Transactions on Modeling and Computer Simulation* 13 (4), 322–333.

Knuth, D.E. (1998). *Seminumerical Algorithms*, 3rd edition. *The Art of Computer Programming*, vol. 2. Addison-Wesley, Reading, MA.

Law, A.M., Kelton, W.D. (2000). *Simulation Modeling and Analysis*, 3rd edition. McGraw-Hill, New York.

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: Banks, J. (Ed.), *Principles, Methodology, Advances, Applications, and Practice*. *Handbook of Simulation*. Wiley, New York, pp. 93–137, Chapter 2.

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*. IEEE Press, Piscataway, NJ, pp. 95–105.

L'Ecuyer, P. (2004a). Polynomial integration lattices. In: Niederreiter, H. (Ed.), *Monte Carlo and Quasi-Monte Carlo Methods 2002*. Springer-Verlag, Berlin, pp. 73–98.

L'Ecuyer, P. (2004b). Random number generation. In: Gentle, J.E., Haerdle, W., Mori, Y. (Eds.), *Concepts and Methods*. *Handbook of Computational Statistics*. Springer-Verlag, Berlin, pp. 35–70, Chapter II.2.

L'Ecuyer, P., Andres, T.H. (1997). A random number generator based on the combination of four LCGs. *Mathematics and Computers in Simulation* 44, 99–107.

L'Ecuyer, P., Côté, S. (1991). Implementing a random number package with splitting facilities. *ACM Transactions on Mathematical Software* 17 (1), 98–111.

L'Ecuyer, P., Couture, R. (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., Granger-Piché, J. (2003). Combined generators with components from different families. *Mathematics and Computers in Simulation* 62, 395–404.

L'Ecuyer, P., Hellekalek, P. (1998). Random number generators: Selection criteria and testing. In: Hellekalek, P., Larcher, G. (Eds.), *Random and Quasi-Random Point Sets*. *Lecture Notes in Statistics*, vol. 138. Springer-Verlag, New York, pp. 223–265.

L'Ecuyer, P., Lemieux, C. (2000). Variance reduction via lattice rules. *Management Science* 46 (9), 1214–1235.

L'Ecuyer, P., Lemieux, C. (2002). Recent advances in randomized quasi-Monte Carlo methods. In: Dror, M., L'Ecuyer, P., Szidarovszki, F. (Eds.), *Modeling Uncertainty: An Examination of Stochastic Theory, Methods and Applications*. Kluwer Academic Publishers, Boston, pp. 419–474.

L'Ecuyer, P., Panneton, F. (2002). Construction of equidistributed generators based on linear recurrences modulo 2. In: Fang, K.-T., Hickernell, F.J., Niederreiter, H. (Eds.), *Monte Carlo and Quasi-Monte Carlo Methods 2000*. Springer-Verlag, Berlin, pp. 318–330.

L'Ecuyer, P., Simard, R. (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., Simard, R. (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., Simard, R. (2002). TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators. Software user's guide. Available at *http://www.iro.umontreal.ca/~lecuyer*.

L'Ecuyer, P., Tezuka, S. (1991). Structural properties for two classes of combined random number generators. *Mathematics of Computation* 57 (196), 735–746.

L'Ecuyer, P., Touzin, R. (2000). Fast combined multiple recursive generators with multipliers of the form $a = \pm 2^q \pm 2^r$. In: Joines, J.A., Barton, R.R., Kang, K., Fishwick, P.A. (Eds.), *Proceedings of the 2000 Winter Simulation Conference*. IEEE Press, Piscataway, NJ, pp. 683–689.

L'Ecuyer, P., Touzin, R. (2004). On the Deng–Lin random number generators and related methods. *Statistics and Computing* 14, 5–9.

L'Ecuyer, P., Blouin, F., Couture, R. (1993). A search for good multiple recursive random number generators. *ACM Transactions on Modeling and Computer Simulation* 3 (2), 87–98.

L'Ecuyer, P., Simard, R., Chen, E.J., Kelton, W.D. (2002a). An object-oriented random-number package with many long streams and substreams. *Operations Research* 50 (6), 1073–1075.

L'Ecuyer, P., Simard, R., Wegenkittl, S. (2002b). Sparse serial tests of uniformity for random number generators. *SIAM Journal on Scientific Computing* 24 (2), 652–668.

Leeb, H. (1995). Random numbers for computer simulation. Master's thesis, University of Salzburg.

Lemieux, C., L'Ecuyer, P. (2003). Randomized polynomial lattice rules for multivariate integration and simulation. *SIAM Journal on Scientific Computing* 24 (5), 1768–1789.

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*. Elsevier (North-Holland), Amsterdam, pp. 3–10.

Marsaglia, G. (1996). The Marsaglia random number CDROM including the DIEHARD battery of tests of randomness. Available at *http://stat.fsu.edu/pub/diehard*.

Marsaglia, G., Zaman, A. (1991). A new class of random number generators. *The Annals of Applied Probability* 1, 462–480.

Matsumoto, M., Kurita, Y. (1992). Twisted GFSR generators. *ACM Transactions on Modeling and Computer Simulation* 2 (3), 179–194.

Matsumoto, M., Kurita, Y. (1994). Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation* 4 (3), 254–266.

Matsumoto, M., Kurita, Y. (1996). Strong deviations from randomness in $m$-sequences based on trinomials. *ACM Transactions on Modeling and Computer Simulation* 6 (2), 99–106.

Matsumoto, M., Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* 8 (1), 3–30.

Niederreiter, H. (1992). *Random Number Generation and Quasi-Monte Carlo Methods*. *SIAM CBMS–NSF Regional Conference Series in Applied Mathematics*, vol. 63. SIAM, Philadelphia.

Niederreiter, H. (1995). The multiple-recursive matrix method for pseudorandom number generation. *Finite Fields and Their Applications* 1, 3–30.

Niederreiter, H., Shparlinski, I.E. (2002). Recent advances in the theory of nonlinear pseudorandom number generators. In: Fang, K.-T., Hickernell, F.J., Niederreiter, H. (Eds.), *Monte Carlo and Quasi-Monte Carlo Methods 2000*. Springer-Verlag, Berlin, pp. 86–102.

Nishimura, T. (2000). Tables of 64-bit Mersenne twisters. *ACM Transactions on Modeling and Computer Simulation* 10 (4), 348–357.

Panneton, F. (2004). Construction d'ensembles de points basée sur des récurrences linéaires dans un corps fini de caractéristique 2 pour la simulation Monte Carlo et l'intégration quasi-Monte Carlo. PhD thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada, August 2004.

Panneton, F., L'Ecuyer, P. (2004). Random number generators based on linear recurrences in $F_{2w}$. In: Niederreiter, H. (Ed.), *Monte Carlo and Quasi-Monte Carlo Methods 2002*. Springer-Verlag, Berlin, pp. 367–378.

Panneton, F., L'Ecuyer, P. (2006). Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software* 32 (1), in press.

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S. (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. Available at *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. (1994). A unified view of large-period random number generators. *Journal of the Operations Research Society of Japan* 37, 211–227.

Tezuka, S. (1995). *Uniform Random Numbers: Theory and Practice*. Kluwer Academic Publishers, Norwell, MA.

Tezuka, S., L'Ecuyer, P. (1991). Efficient and portable combined Tausworthe random number generators. *ACM Transactions on Modeling and Computer Simulation* 1 (2), 99–112.

Tezuka, S., L'Ecuyer, P., Couture, R. (1994). On the add-with-carry and subtract-with-borrow random number generators. *ACM Transactions on Modeling and Computer Simulation* 3 (4), 315–331.

Tootill, J.P.R., Robinson, W.D., Eagle, D.J. (1973). An asymptotically random Tausworthe sequence. *Journal of the ACM* 20, 469–481.

Vattulainen, I., Ala-Nissila, T., Kankaala, K. (1995). Physical models as tests of randomness. *Physical Review E* 52 (3), 3205–3213.

Wang, D., Compagner, A. (1993). On the use of reducible polynomials as random number generators. *Mathematics of Computation* 60, 363–374.

Wegenkittl, S., Matsumoto, M. (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.