Random Number Generation and Quasi-Monte Carlo *

Pierre L'Ecuyer
Université de Montréal, Canada, and Inria Rennes, France

November 2014

Keywords: *random number generator, pseudorandom numbers, linear generator, multiple recursive generator, tests of uniformity, random variate generation, inversion, rejection, simulation, Monte Carlo, quasi-Monte Carlo, low-discrepancy*

Abstract

Abstract: Probability theory defines random variables and stochastic processes in terms of probability spaces, an abstract notion whose concrete and exact realization on a computer is far from obvious. (Pseudo) random number generators (RNGs) implemented on computers are actually deterministic programs which imitate, to some extent, independent random variables uniformly distributed over the interval $[0, 1]$ (i.i.d. $U[0, 1]$, for short). RNGs are a key ingredient for Monte Carlo simulations, probabilistic algorithms, computer games, cryptography, casino machines, and so on. In this article, we outline the main principles underlying the design and testing of RNGs for statistical computing and simulation. Then we indicate how $U(0, 1)$ random numbers can be transformed to generate random variates from other distributions. Finally, we summarize the main ideas on quasi-random points, which are more evenly distributed than independent random point and permit one to estimate integrals more accurately for the same number of function evaluations.

Introduction

Probability theory defines **random variables** and **stochastic processes** in terms of *probability spaces*, a purely abstract notion whose concrete and exact realization on a computer is far from obvious. *(Pseudo) random number generators* (RNGs) implemented on computers are actually deterministic programs that *imitate*, to some extent, independent random variables **uniformly distributed** over the interval [0, 1] (i.i.d. $U[0, 1]$, for short) [13, 15, 23]. RNGs are a key ingredient for **Monte Carlo simulation** , probabilistic algorithms, computer games, etc. In the section 'Uniform Random Number Generators', we discuss

*An older version of this article was originally published online in 2006 in Encyclopedia of Actuarial Science, © John Wiley & Sons, Ltd. This version, largely rewritten, is submitted for Wiley StatsRef: Statistics Reference Online, 2014.

the main ideas underlying the design and testing of RNGs for computational statistics and simulation. For other applications such as cryptography and lotteries, for example, there are different (stronger) requirements [40, 42].

Random variates from nonuniform distributions and stochastic objects of all sorts are simulated by applying appropriate transformations to these fake i.i.d. $U[0,1]$ [4, 12]. Conceptually, the easiest way of generating a random variate X with **cumulative distribution function** F (i.e., such that $F(x) = \mathbb{P}[X \le x]$ for all $x \in \mathbb{R}$) is to apply the inverse of F to a $U[0,1]$ random variate U:

$$X = F^{-1}(U) \stackrel{\text{def}}{=} \min\{x | F(x) \ge U\}. \tag{1}$$

Then, $\mathbb{P}[X \le x] = \mathbb{P}[F^{-1}(U) \le x] = \mathbb{P}[U \le F(x)] = F(x)$, so X has the desired distribution. This is the *inversion* method (see). If X has a **discrete distribution** with $\mathbb{P}[X = x_i] = p_i$, inversion can be implemented by storing the pairs $(x_i, F(x_i))$ in a table and using binary search or index search to find the value of X that satisfies (1). In the cases in which F^{-1} is hard or expensive to compute, other methods are sometimes more advantageous, as explained in the section 'Nonuniform Random Variate Generation'.

One major use of simulation is to estimate the mathematical expectation of a function of several random variables, which can usually be expressed (sometimes implicitly) in the form

$$\mu = \int_{[0,1]^s} f(\boldsymbol{u}) \mathrm{d}\boldsymbol{u}, \tag{2}$$

where f is a real-valued function defined over the unit hypercube $[0,1]^s$, $\boldsymbol{u} = (u_1, \ldots, u_s) \in [0,1]^s$, and s is the number of calls to the RNG required by the simulation.

A simple way of approximating μ is to select a point set $P_n = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\} \subset [0,1]^s$ and take the average

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{u}_i) \tag{3}$$

as an approximation. The **Monte Carlo method** (MC) chooses the $\boldsymbol{u}_i$'s as n i.i.d. uniformly distributed random vectors over $[0,\ 1]^s$. Then, $\hat{\mu}_n$ is an **unbiased estimator** of μ. If f is also square integrable, then $\hat{\mu}_n$ obeys a **central limit theorem** , which permits one to compute an asymptotically valid **confidence interval** for μ, and the error $|\hat{\mu}_n - \mu|$ converges to zero as $\mathcal{O}(n^{-1/2})$ in probability, in the sense that the width of the confidence interval converges to 0 at that rate.

The idea of *quasi-Monte Carlo* (QMC) is to select the point set P_n more evenly distributed over $[0,1]^s$ than a typical set of random points, with the aim of reducing the error compared with MC [6, 22, 43]. Important issues are: How should we measure the uniformity of P_n? How can we construct such highly uniform point sets? Under what conditions is the error (or variance) effectively smaller? and How much smaller? These questions are discussed briefly in the section 'Quasi-Monte Carlo Methods'.

Uniform Random Number Generators

Following [15], a uniform RNG can be defined as a structure $(\mathcal{S}, \mu, f, \mathcal{U}, g)$, where $\mathcal{S}$ is a finite set of *states*, μ is a probability distribution on $\mathcal{S}$ used to select the *initial state* s_0 (the *seed*), $f : \mathcal{S} \to \mathcal{S}$ is the *transition function*, $\mathcal{U}$ is the *output set*, and $g : \mathcal{S} \to \mathcal{U}$ is the *output function*. In what follows, we assume that $\mathcal{U} = [0, 1]$. The state evolves according to the recurrence $s_i = f(s_{i-1})$, for $i \geq 1$, and the *output* at step i is $u_i = g(s_i) \in \mathcal{U}$. These u_i are the *random numbers* produced by the RNG. Because $\mathcal{S}$ is finite, one must have $s_{l+j} = s_l$ for some $l \geq 0$ and $j > 0$. Then, $s_{i+j} = s_i$ and $u_{i+j} = u_i$ for all $i \geq l$; that is, the output sequence is eventually periodic. The smallest positive j for which this happens is the *period length* ρ. Of course, ρ cannot exceed $|\mathcal{S}|$, the cardinality of $\mathcal{S}$. So $\rho \leq 2^b$ if the state is represented over b bits. Good RNGs are designed so that their period length is close to that upper bound.

'Truly random' generators based on physical devices such as noise diodes, quantum computing devices, and so on, are also available, but for simulation and computational statistics, they are much less convenient than RNGs based on deterministic recurrences. A major advantage of the latter is their ability to repeat exactly the same sequence of numbers: This is very handy for program verification and replay, and for implementing variance reduction methods in simulation (e.g. for comparing similar systems with common random numbers) [1, 34, 21]. True randomness may nevertheless be used for selecting the seed s_0. Then, the RNG can be viewed as an *extensor* of randomness, stretching a short random seed into a long sequence of random-looking numbers.

What quality criteria should we consider in RNG design? One obvious requirement is an extremely long period, to make sure that no wraparound over the cycle can occur in practice. The RNG must also be *efficient* (run fast and use little memory), *repeatable* (able to reproduce the same sequence), and *portable* (work the same way in different software/hardware environments). The availability of efficient jumping-ahead methods, that is, to quickly compute $s_{i+\nu}$ given s_i, for any large ν, is also an important asset, because it permits one to partition the sequence into long disjoint streams and substreams for constructing *virtual generators* from a single backbone RNG [21, 23, 31, 38]. Such multiple streams of random numbers are very convenient when simulating slightly different systems with common random numbers to compare their performance. They facilitate the task of synchronizing the use of the random numbers across the different systems [1, 14, 21, 24, 34]. Multiple streams of random numbers are also essential for parallel simulation on multiple processors, which is becoming increasingly important, in view of the fact that computational power increases nowadays no longer by increasing the clock speeds, but by increasing the number of computing elements [31]. On some popular parallel computing devices, such as *graphical processing units* (GPUs), the size of fast-access memory for each processing element is quite small, so the state of the RNG (and of each stream) should remain small (say at most a few dozen bytes). This is also true for other types of small computing devices.

A long period does not suffice for the u_i's to behave as uniform and independent. Ideally, we would like the vector $(u_0, \ldots, u_{s-1})$ to be uniformly distributed over $[0, 1]^s$ for each $s > 0$. This cannot be exactly true, because these vectors always take their values only from the

finite set $\Psi_s = \{(u_0, \ldots, u_{s-1}) : s_0 \in \mathcal{S}\}$, whose cardinality cannot exceed $|\mathcal{S}|$. If s_0 is random, Ψ_s can be viewed as the **sample space** from which vectors of successive output values are taken randomly. Producing s-dimensional vectors by taking nonoverlapping blocks of s output values of an RNG can be viewed in a way as picking points at random from Ψ_s, without replacement. It seems natural, then, to require that Ψ_s be very evenly distributed over the unit cube, so that the uniform distribution over Ψ_s is a good approximation to that over $[0,1]^s$, at least for moderate values of s. For this, the cardinality of $\mathcal{S}$ must be huge, so that it is possible for Ψ_s to fill the unit hypercube densely enough. This is in fact a more important reason for having a large state space than just the fear of wrapping around the cycle.

The uniformity of Ψ_s is usually assessed by *figures of merit* measuring the *discrepancy* between the empirical distribution of its points and the uniform distribution over $[0,1]^s$ [13, 33, 23, 43]. Several such measures can be defined and they correspond to **goodness-of-fit test statistics** for the uniform distribution over $[0,1]^s$. An important criterion in choosing a specific measure is the ability to compute it efficiently without generating the points explicitly, and this depends on the mathematical structure of Ψ_s. This is why different figures of merit are used in practice for analyzing different classes of RNGs. The selected figure of merit is usually computed for a range of dimensions, for example, for $s \leq s_1$ for some arbitrary integer s_1. Examples of such figures of merit include the (normalized) *spectral test* in the case of multiple recursive generators (MRGs) and linear congruential generators (LCGs) [13, 8, 25, 18, 37] and measures of equidistribution for generators based on linear recurrences modulo 2 [16, 33, 51]. These RNGs are defined below.

More generally, one can compute a discrepancy measure for sets of the form $\Psi_s(I) = \{(u_{i_1}, \ldots, u_{i_s}) : s_0 \in \mathcal{S}\}$, where $I = \{i_1, i_2, \ldots, i_s\}$ is a fixed set of nonnegative integers. Do this for all I in a given class $\mathcal{I}$, and take either the worst case or some type of average (after appropriate normalization) as a figure of merit for the RNG [30, 33]. The choice of $\mathcal{I}$ is left open. Typically, $\mathcal{I}$ would contain sets I such that s and $i_s - i_1$ are both relatively small.

After an RNG has been designed based on sound mathematical analysis, it is good practice to submit it to a battery of **empirical statistical tests** that try to detect empirical evidence against the hypothesis $\mathcal{H}_0$ that the u_i are i.i.d. $U[0,1]$. A test can be defined by any function T of a finite set of u_i's, and whose distribution under $\mathcal{H}_0$ is known or can be closely approximated. There is an unlimited number of such tests. No finite set of tests can guarantee, when passed, that a given generator is fully reliable for all kinds of simulations. Passing large batteries of tests cannot prove that the RNG is foolproof, but it certainly improves one's confidence in the RNG. In reality, no RNG can pass all possible statistical tests. Roughly speaking, bad RNGs are those that fail simple tests, whereas good ones fail only complicated tests that are very hard to find and run. Ideally, the statistical tests should be selected in close relation with the target application, that is, T should mimic the random variable of interest, but this is rarely practical, especially for general purpose RNGs. Large collections of statistical tests for uniform RNGs are proposed and implemented in [2, 13, 35, 36] and other references given there.

The most widely used RNGs are based on linear recurrences of the form

$$x_i = (a_1 x_{i-1} + \ldots + a_k x_{i-k}) \bmod m, \tag{4}$$

for positive integers m and k, and coefficients a_l in $\{0, 1, \ldots, m-1\}$. The state at step i is $s_i = (x_{i-k+1}, \ldots, x_i)$. If m is a prime number, one can choose the coefficients a_l's so that the period length reaches $\rho = m^k - 1$, which is the largest possible value [13].

A *multiple recursive generator* (MRG) uses (4) with a large value of m and defines the output as $u_i = x_i/m$. For $k = 1$, this is the classical LCG. Implementation techniques and concrete examples are given, for example, in [8, 18, 23, 39] and the references given there.

A different approach takes $m = 2$, which allows fast implementations by exploiting the binary nature of computers. By defining the output as $u_i = \sum_{j=1}^{L} x_{is+j-1} 2^{-j}$ for some positive integers s and L, one obtains a *linear feedback shift register* (LFSR) generator [19, 43, 51]. This can be generalized to a linear recurrence of the form $\boldsymbol{x}_i = \boldsymbol{A}\boldsymbol{x}_{i-1} \bmod 2$, where the k-bit vector $\boldsymbol{x}_i = (x_{i,0}, \ldots, x_{i,k-1})^\mathsf{T}$ is the state at step i and $\boldsymbol{A}$ is a $k \times k$ binary matrix. To construct the output, we define the w-bit vector $\boldsymbol{y}_i = (y_{i,0}, \ldots, y_{i,w-1})^\mathsf{T} = \boldsymbol{B}\boldsymbol{x}_i \bmod 2$, where $\boldsymbol{B}$ is a $w \times k$ binary matrix, and put $u_i = y_{i,0}/2 + y_{i,1}/2^2 + \cdots + y_{i,w-1}/2^w$. The sequence of states $\boldsymbol{x}_i$ has maximal period $2^k - 1$ if and only if the characteristic polynomial of $\boldsymbol{A}$ is primitive modulo 2, and for each j, the binary sequence $\{y_{i,j},\, i \geq 0\}$ obeys the recurrence (4) where the a_l are the coefficients of this characteristic polynomial [33]. This implies that all these RNGs fail statistical tests based on the linear complexity of this binary sequence, but many $\mathbb{F}_2$-linear RNGs are nevertheless useful and reliable for most applications, and they are very fast. This setup encompasses several types of generators, including the Tausworthe, polynomial LCG, generalized feedback shift register (GFSR), twisted GFSR, Mersenne twister, WELL, xorshift, and combinations of these [32, 33, 41, 47].

Some of the best RNGs currently available are *combined generators*, constructed by combining the outputs of two or more RNGs having a simple structure. The idea is to keep the components simple so that they run fast, and to select them carefully so that their combination has a more complicated structure and highly-uniform sets $\Psi_s(I)$ for the values of s and sets I deemed important. Such recommendable combinations are proposed in [19, 18, 32, 39], for example. By combining MRGs with $\mathbb{F}_2$-linear RNGs, the structure becomes nonlinear and it becomes more difficult to measure the uniformity, but useful bounds on uniformity measures can still be computed [26], and the resulting RNGs are very robust to statistical testing. In general, nonlinearity can be introduced by making either f or g nonlinear. In some cases, the transition function f does very little and much of the work is left to g. *Counter-based RNGs* push this to the limit: f just increments a counter, so $f(i) = i+1$, and g implements a complicated transformation such as a block cipher encryption algorithm [11, 36, 48, 49]. One advantage is that the output u_i at any step i can be generated directly and efficiently without generating the previous values, so the u_i's can be generated in any order. One drawback is that the good ones are typically slower than linear RNGs.

Other types of generators, including nonlinear ones, are discussed, for example, in [7, 13, 15, 17, 23, 51], and some perform quite well in statistical tests [36]. Plenty of very bad and unreliable RNGs abound in both commercial and open-source software. Convincing

examples can be found in [17, 20, 36], for example. In particular, we think that all LCGs with period length less than 2^{100}, say, should be discarded.

Nonuniform Random Variate Generation

For most applications, inversion is the method of choice for generating nonuniform random variates. The fact that it transforms U monotonously into X (X is a nondecreasing function of U) makes it compatible with major variance reductions techniques [1, 14]. For certain types of distributions (the **normal** and **chi-square** , for example), there is no close form expression for F^{-1} but good numerical approximations are available [4, 9, 12]. Methods to construct accurate approximations of F^{-1} are discussed in [12]. If an efficient algorithm is available to compute F, one can use it to approximate $F^{-1}(U)$ by binary search, or by Newton-Raphson if the density is also known [3, 4].

For discrete distributions over integers i, inversion can be implemented by first tabulating the pairs $(x_i, F(x_i))$, then using binary search to find $I = \min\{i \mid F(x_i) \geq U\}$ [3]. If there are too many possible values of i (e.g., an infinite number), one can store only the main portion of the table and compute other values when needed [3]. Faster algorithms use an index: partition $(0, 1)$ into c subintervals of size $1/c$ and tabulate $L_j = F^{-1}(j/c)$ for $j = 0, \ldots, c-1$. When $U \in [j/c, (j+1)/c)$, search for X in $L_j, \ldots, L_{j+1}$. When c is large, this is very fast, as fast as the *alias method* [12].

There are situations where speed is important and where noninversion methods are appropriate. 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). Simplicity should generally not be sacrificed for small speed gains. In what follows, we outline some important special cases of noninversion methods.

Suppose we want to generate X from a complicated density f. Select another density r such that $f(x) \leq t(x) \stackrel{\text{def}}{=} ar(x)$ for all x for some constant $a \geq 1$, and such that generating variates Y from the density r is easy. To generate X, repeat the following: generate Y from the density r and an independent $U[0, 1]$ variate U, until $Ut(Y) \leq f(Y)$. Then, return $X = Y$. This is the *rejection* method [4, 12]. The number R of turns into the 'repeat' loop is one plus a geometric random variable with parameter $1/a$, so $\mathbb{E}[R] = a$, and we want a to be as small (close to 1) as possible. There is usually a compromise between having a close to 1 and keeping r simple. When f is a bit expensive to compute, one can also use a *squeeze function* q which is faster to evaluate and such that $q(x) \leq f(x)$ for all x. To verify the condition $U\,t(Y) \leq f(Y)$, first check if $U\,t(Y) \leq q(Y)$, in which case we accept immediately and there is no need to compute $f(Y)$.

The rejection method is often applied after a change of variable that transforms 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. 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 [4, 12].

Changes of variable are also used commonly to standardize stochastic processes by changing their time scale. For example, to generate arrivals from a Poisson process with piecewise-constant rate, one can make a change of variable to obtain a Poisson process with rate 1 (with inter-arrival times that are independent and exponential with mean 1), generate the arrivals for that standard process, and transform the arrival times back to the original time scale. More generally, the change of variable can be random, driven by a subordinator process (e.g., a gamma process), and this applies to various kinds of Lévy processes, including Brownian motion and geometric Brownian motion [1].

Besides the general methods, several specialized and fancy techniques have been designed for commonly used distributions like the Poisson, normal, and so on. Details can be found in [1, 4, 9, 12], for example.

Quasi-Monte Carlo Methods

The primary ingredient for QMC is a *highly uniform* (or *low-discrepancy*) point set P_n to be used in (3). The two main classes of approaches for constructing such point sets are *lattice rules* and *digital nets* [5, 6, 22, 43, 50]. For these constructions, the point sets P_n have the same structures as Ψ_s for LCGs and for $\mathbb{F}_2$-linear generators, respectively, and their uniformity (or discrepancy) can be measured in a similar way. A *low-discrepancy sequence* or **Quasi-Random Sequence** is an infinite sequence of points P_∞ such that the set P_n comprised of the first n points of the sequence have low discrepancy for all n, as $n \to \infty$ [6, 43]. With such a sequence, one does not have to fix n in advance when evaluating (3); sequential sampling becomes possible. Applications of QMC in different areas are discussed, for example, in [6, 10, 22, 27].

QMC methods are typically justified by worst-case error bounds of the form

$$|\hat{\mu}_n - \mu| \le \|f - \mu\| \cdot D(P_n) \tag{5}$$

for all f in some Banach space $\mathcal{F}$ with norm $\|\cdot\|$. Here, $\|f - \mu\|$ measures the *variability* of f, while $D(P_n)$ measures the *discrepancy* of P_n and its definition depends on how the norm is defined in $\mathcal{F}$. These inequalities are typically derived by applying a version of Holder's inequality in the selected space. A popular special case of (5) is the **Koksma-Hlawka inequality** , in which the norm is the variation of f in the sense of Hardy and Krause and $D(P_n)$ is the (rectangular) *star discrepancy* [43]. The latter considers all rectangular boxes aligned with the axes and with a corner at the origin in $[0, 1]^s$, computes the absolute difference between the volume of the box, and the fraction of P_n falling in it, and takes the worst case over all such boxes. This is a multidimensional version of the **Kolmogorov-Smirnov goodness-of-fit test statistic** . Several other versions of (5) are discussed in [6, 22, 44] and other references therein. Some of the corresponding discrepancy measures are much easier to compute than the star discrepancy. Like for RNGs, the discrepancy measures used to construct and choose point sets in practice depend on the structure of the construction, and are selected so they can be computed effectively [6, 30].

Specific sequences P_∞ have been constructed with star discrepancy $D(P_n) = \mathcal{O}(n^{-1}(\ln n)^s)$ when $n \to \infty$ for fixed s, yielding a convergence rate of $\mathcal{O}(n^{-1}(\ln n)^s)$ for the worst-case error if $\|f-\mu\| < \infty$ [6, 43, 51]. This is *asymptotically* better than MC. There are other discrepancy measures for which point sets P_n can be constructed with discrepancy $D(P_n) = \mathcal{O}(n^{-\alpha+\epsilon})$ for $\epsilon > 0$ arbitrarily close to 0 and for an arbitrary $\alpha > 0$, and we know how to construct such points [6, 5, 22, 30]. The larger α, the more restricted is the class of functions f for which (5) holds (the functions must be smoother in some sense). Also, the hidden constant in the $\mathcal{O}(n^{-\alpha+\epsilon})$ expression depends on s and α and may be huge for large α or s, so the bound is sometimes meaningless unless n is astronomically large. Thus, even though QMC performs much better than MC in practice in some situations (sometimes by huge factors), the worst-case bound (5) does not always explain why. In fact, the actual error is often much smaller than the bound.

One explanation is that even if s is large, f can sometimes be well approximated by a sum of orthogonal functions defined over some low-dimensional subspaces of $[0,1]^s$, and the bound (5) applies to the integration error over each of those subspaces. If the projections of P_n over the important subspaces have low discrepancy, which is much easier to achieve than getting a small overall $D(P_n)$ for large s, then the overall integration error can be small because the error over each important subspace is small. This has motivated the introduction of discrepancy measures that give more weight to projections of P_n over selected (small) subsets of coordinates [5, 27, 22, 30]. One can choose a discrepancy measure and select the weights for one's specific problem. Software is now available to construct point sets with small weighted discrepancy for the selected weights [29].

One limitation of QMC with a deterministic point set P_n is that no statistical error estimate is available. *Randomized* QMC addresses this issue by randomizing P_n so that (a) each point of the randomized set is uniformly distributed over $[0,1]^s$ and (b) the high uniformity of P_n is preserved. For example, one way to achieve this is to shift the entire point set P_n randomly, by adding a single uniform random variable U modulo 1, for each coordinate. Another way is to take an exclusive-or of each point with the same random U (a digital random shift). By doing this, (3) becomes an unbiased estimator of μ and, by taking a small number of independent randomizations of P_n, one can also obtain an unbiased estimator of $\mathrm{Var}[\hat{\mu}_n]$ and eventually compute a confidence interval for μ [22, 28, 46]. This turns QMC into a variance reduction method [27, 28].

Bounds and expressions for the variance have been developed (to replace (5)) for certain classes of functions and randomized point sets [22, 46]. Under appropriate assumptions on f, the variance converges faster than the MC rate of $\mathcal{O}(1/n)$. For example, a *scrambling* method introduced by Owen for a class of digital nets gives $\mathrm{Var}[\hat{\mu}_n] = \mathcal{O}(n^{-3}(\ln n)^s)$ if the mixed partial derivatives of f are Lipschitz [45]. More generally, convergence results for the variance can be obtained easily from the fact that the variance converges at least as fast as the square of the worst-case error.

Acknowledgements

This work has been supported by the Natural Sciences and Engineering Research Council of Canada Grant No. ODGP0110050, a Canada Research Chair, and an Inria International Chair to the author.

References

[1] S. Asmussen and P. W. Glynn. *Stochastic Simulation*. Springer-Verlag, New York, 2007.

[2] Lawrence E. Bassham III, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST special publication 800-22, revision 1a, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2010.

[3] R. C. H. Cheng. Random variate generation. In Jerry Banks, editor, *Handbook of Simulation*, pages 139–172. Wiley, 1998. chapter 5.

[4] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, NY, 1986.

[5] J. Dick, F. Y. Kuo, and I. H. Sloan. High dimensional integration—the quasi-Monte Carlo way. *Acta Numerica*, 22:133–288, 2013.

[6] J. Dick and F. Pillichshammer. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration*. Cambridge University Press, Cambridge, U.K., 2010.

[7] J. Eichenauer-Herrmann. Pseudorandom number generation by nonlinear methods. *International Statistical Reviews*, 63:247–255, 1995.

[8] G. S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer Series in Operations Research. Springer-Verlag, New York, NY, 1996.

[9] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer, New York, NY, second edition, 2003.

[10] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York, 2004.

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

[12] W. Hörmann, J. Leydold, and G. Derflinger. *Automatic Nonuniform Random Variate Generation*. Springer-Verlag, Berlin, 2004.

[13] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, third edition, 1998.

[14] A. M. Law. *Simulation Modeling and Analysis*. McGraw-Hill, New York, fifth edition, 2014.

[15] P. L'Ecuyer. Uniform random number generation. *Annals of Operations Research*, 53:77–120, 1994.

[16] P. L'Ecuyer. Maximally equidistributed combined Tausworthe generators. *Mathematics of Computation*, 65(213):203–213, 1996.

[17] P. L'Ecuyer. Bad lattice structures for vectors of non-successive values produced by some linear recurrences. *INFORMS Journal on Computing*, 9(1):57–60, 1997.

[18] P. L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1):159–164, 1999.

[19] P. L'Ecuyer. Tables of maximally equidistributed combined LFSR generators. *Mathematics of Computation*, 68(225):261–269, 1999.

[20] P. L'Ecuyer. Software for uniform random number generation: Distinguishing the good and the bad. In *Proceedings of the 2001 Winter Simulation Conference*, pages 95–105, Piscataway, NJ, 2001. IEEE Press.

[21] P. L'Ecuyer. *SSJ: A Java Library for Stochastic Simulation*, 2008. Software user's guide, available at `http://www.iro.umontreal.ca/~lecuyer`.

[22] P. L'Ecuyer. Quasi-Monte Carlo methods with applications in finance. *Finance and Stochastics*, 13(3):307–349, 2009.

[23] P. L'Ecuyer. Random number generation. In J. E. Gentle, W. Haerdle, and Y. Mori, editors, *Handbook of Computational Statistics*, pages 35–71. Springer-Verlag, Berlin, second edition, 2012.

[24] P. L'Ecuyer and E. Buist. Variance reduction in the simulation of call centers. In *Proceedings of the 2006 Winter Simulation Conference*, pages 604–613. IEEE Press, 2006.

[25] P. L'Ecuyer and R. Couture. An implementation of the lattice and spectral tests for multiple recursive linear random number generators. *INFORMS Journal on Computing*, 9(2):206–217, 1997.

[26] P. L'Ecuyer and J. Granger-Piché. Combined generators with components from different families. *Mathematics and Computers in Simulation*, 62:395–404, 2003.

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

[28] P. L'Ecuyer and C. Lemieux. Recent advances in randomized quasi-Monte Carlo methods. In M. Dror, P. L'Ecuyer, and F. Szidarovszky, editors, *Modeling Uncertainty: An Examination of Stochastic Theory, Methods, and Applications*, pages 419–474. Kluwer Academic, Boston, 2002.

[29] P. L'Ecuyer and D. Munger. Lattice builder: A general software tool for constructing rank-1 lattice rules. Submitted, see `http://www.iro.umontreal.ca/~lecuyer/papers.html`, 2012.

[30] P. L'Ecuyer and D. Munger. On figures of merit for randomly-shifted lattice rules. In H. Wozniakowski and L. Plaskota, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2010*, pages 133–159, Berlin, 2012. Springer-Verlag.

[31] P. L'Ecuyer, B. Oreshkin, and R. Simard. Random numbers for parallel computers: Requirements and methods, 2014. `http://www.iro.umontreal.ca/~lecuyer/myftp/papers/parallel-rng-imacs.pdf`.

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

[33] P. L'Ecuyer and F. Panneton. $\mathbf{F}_2$-linear random number generators. In C. Alexopoulos, D. Goldsman, and J. R. Wilson, editors, *Advancing the Frontiers of Simulation: A Festschrift in Honor of George Samuel Fishman*, pages 169–193. Springer-Verlag, New York, 2009.

[34] P. L'Ecuyer and G. Perron. On the convergence rates of IPA and FDC derivative estimators. *Operations Research*, 42(4):643–656, 1994.

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

[36] P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4):Article 22, August 2007.

[37] P. L'Ecuyer and R. Simard. On the lattice structure of a special class of multiple recursive random number generators. *INFORMS Journal on Computing*, 26(2):449–460, 2014.

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

[39] P. L'Ecuyer and R. Touzin. 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*, pages 683–689, Piscataway, NJ, 2000. IEEE Press.

[40] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, Princeton, 1996.

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

[42] A. J. Menezes, S. A. Vanstone, and P. C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, 1996.

[43] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, PA, 1992.

[44] Dirk Nuyens. The construction of good lattice rules and polynomial lattice rules. In Peter Kritzer, Harald Niederreiter, Friedrich Pillichshammer, and Arne Winterhof, editors, *Radon Series on Computational and Applied Mathematics*. De Gruyter, 2014. to appear.

[45] A. B. Owen. Scrambled net variance for integrals of smooth functions. *Annals of Statistics*, 25(4):1541–1562, 1997.

[46] A. B. Owen. Latin supercube sampling for very high-dimensional simulations. *ACM Transactions on Modeling and Computer Simulation*, 8(1):71–102, 1998.

[47] F. Panneton, P. L'Ecuyer, and M. Matsumoto. Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software*, 32(1):1–16, 2006.

[48] C. L. Phillips, J. A. Anderson, and S. C. Glotzer. Pseudo-random number generation for brownian dynamics and dissipative particle dynamics simulations on GPU devices. *Journal of Computational Physics*, 230(19):7191–7201, 2011.

[49] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw. Parallel random numbers: as easy as 1, 2, 3. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 16:1–16:12, New York, 2011. ACM.

[50] I. H. Sloan and S. Joe. *Lattice Methods for Multiple Integration*. Clarendon Press, Oxford, 1994.

[51] S. Tezuka. *Uniform Random Numbers: Theory and Practice*. Kluwer Academic Publishers, Norwell, MA, 1995.