Random Number Generation and Quasi-Monte Carlo

PIERRE L'ECUYER

Volume 3, pp. 1363–1369

In

Encyclopedia Of Actuarial Science
(ISBN 0-470-84676-3)

Edited by

Jozef L. Teugels and Bjørn Sundt

Random Number Generation and Quasi-Monte Carlo

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, 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 the section 'Uniform Random Number Generators', we discuss the main ideas underlying their design and testing.

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

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

Then, $P[X \le x] = P[F^{-1}(U) \le x] = P[U \le F(x)] = F(x)$, so X has the desired distribution. This is the *inversion* method. If X has a **discrete distribution** with $P[X = x_i] = p_i$, inversion can be implemented by storing the values of $F(x_i)$ in a table and using binary 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 in **estimating** mathematical expectations of functions of several random variables, that is, multidimensional integrals. To apply simulation, the integrand is expressed (sometimes implicitly) as a function f of s i.i.d. $U[0, 1]$ random variables, where s can be viewed as the number of calls to the RNG required by the simulation (s is allowed to be infinite or random, but here we assume it is a constant, for simplicity). In other words, the goal is to estimate

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

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

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

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 $O_p(n^{-1/2})$.

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. 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 in the section 'Quasi-Monte Carlo Methods'.

Uniform Random Number Generators

Following [11], 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 $i \ge 1$, and the *output* at step i is $u_i = g(s_i) \in U$. 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 $l \ge 0$ and $j > 0$. Then, $s_{i+j} = s_i$ and $u_{i+j} = u_i$ for all $i \ge l$; 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.

Attempts have been made to construct 'truly random' generators based on physical devices such as noise diodes, gamma ray counters, and so on, but these remain largely impractical and not always reliable. A major advantage of RNGs based on deterministic recurrences is the ability to repeat exactly the same sequence of numbers – this is very handy for program verification and for implementing variance reduction methods in simulation (e.g. for comparing similar systems with common random numbers) [1, 5, 10]. True randomness can 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 [10, 22].

A long period does not suffice for the u_i's to appear 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 formally 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 quite 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, to make sure that Ψ_s can fill up the unit hypercube densely enough. The latter 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 $[0, 1]^s$ [7, 19, 29]. 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) [5, 9, 15] and measures of equidistribution for generators based on linear recurrences modulo 2 [12, 20, 34]. (These RNGs are defined below.)

More generally, one can compute a discrepancy measure for sets of the form $\Psi_s(I) = \{(u_{i_1}, \ldots, u_{i_s}) \mid s_0 \in \mathcal{S}\}$, 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 kind of average (after appropriate normalization) as a figure of merit for the RNG [18, 20]. The choice of $\mathcal{I}$ is arbitrary. Typically, it would contain sets I such that s and $i_s - i_1$ are rather 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 many tests may improve one's confidence in the RNG, but it never proves that the RNG is foolproof. In fact, no RNG can pass all statistical tests. Roughly, *bad* RNGs are those that fail *simple* tests, whereas *good* ones fail only complicated tests that are very hard to find and run. Whenever possible, 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. Specific tests for RNGs are proposed and implemented in [9, 21, 27] and other references given there.

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

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

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 [9].

A multiple recursive generator 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 *linear congruential generator*. Implementation techniques and concrete examples are given, for example, in [5, 15, 24] and the references given there.

A different approach takes $m = 2$, which allows fast implementations by exploiting the binary nature of computers. The output can be defined as $u_i = \sum_{j=1}^{L} x_{is+j-1} m^{-j}$ for some positive integers s and L, yielding a *linear feedback shift register* (LFSR) or *Tausworthe* generator [16, 29, 34]. This can be generalized to a recurrence of the form $\mathbf{x}_i = A\mathbf{x}_{i-1} \bmod 2$, $\mathbf{y}_i = B\mathbf{x}_i \bmod 2$, and $u_i = y_{i,0}/2 + y_{i,1}/2^2 + \cdots + y_{i,w-1}/2^w$, where k and w are positive integers, $\mathbf{x}_i$ is a k-dimensional vector of bits, $\mathbf{y}_i = (y_{i,0}, \ldots, y_{i,w-1})^T$ is a w-dimensional vector of bits, and A and B are binary matrices. Then, for each j, the bit sequence $\{y_{i,j},\ i \geq 0\}$ obeys the recurrence (4) where the a_l are the coefficients of the characteristic polynomial of A [20]. This setup encompasses several types of generators, including the Tausworthe, polynomial LCG, generalized feedback shift register (GFSR), twisted GFSR, Mersenne twister, and combinations of these [20, 28].

Some of the best currently available RNGs 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 combinations are proposed in [15, 16, 20], for example.

Other types of generators, including nonlinear ones, are discussed, for example, in [3, 9, 11, 13, 22, 34]. Plenty of very bad and unreliable RNGs abound in software products, regardless of how much they cost. Convincing examples can be found in [13, 17, 23], for example. In particular, all LCGs with period length less than 2^{100}, say, should be discarded in my opinion.

On the other hand, the following RNGs have fairly good theoretical support, have been extensively tested, and are easy to use: the Mersenne twister of [28], the combined MRGs of [15], and the combined Tausworthe generators of [16]. A convenient software package with multiple streams and substreams of random numbers, available in several programming languages, is described in [22]. Further discussion and up-to-date developments on RNGs can be found in [9, 14] and from the web pages: http://www.iro.umontreal.ca/∼lecuyer, http://random.mat.sbg.ac.at, and http://cgm.cs.mcgill.ca/∼luc/.

Nonuniform Random Variate Generation

For most applications, inversion should be 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, 10]. For certain types of distributions (the normal, student, and chi-square, for example), there is no close form expression for F^{-1} but good numerical approximations are available [1, 6]. There are also 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.

The *alias method* is a very fast way of generating a variate X from the discrete distribution over the finite set $\{x_1, \ldots, x_N\}$, with $p_i = P[X = x_i]$ for each i, when N is large. It is not monotone, but generates random variates in $O(1)$ time per call, after a table setup, which takes $O(N)$ time. Consider a bar diagram of the distribution, where each index i has a bar of height p_i. The idea is to 'level' the

bars so that each has a height $1/N$, by cutting off bar pieces and transferring them to different indices. This is done in a way that in the new diagram, each bar i contains one piece of size q_i (say) from the original bar i and one piece of size $1/N - q_i$ from another bar whose index j, denoted by $A(i)$, is called the *alias* value of i. The setup procedure initializes two tables A and R where $A(i)$ is the alias value of i and $R(i) = (i-1)/N + q_i$; see [2, 10] for the details. To generate X, take a $U[0,1]$ variate U, let $i = \lceil N \cdot U \rceil$, and return $X = x_i$ if $U < R(i)$; $X = x_{A(i)}$ otherwise. There is a version of the alias method for continuous distributions; it is called the *acceptance–complement* method [2].

Now suppose we want to generate X from a complicated density f. Select another density r such that $f(x) \le t(x) \stackrel{\text{def}}{=} ar(x)$ for all x for some constant a, and such that generating variates Y from the density r is easy. Clearly, one must have $a \ge 1$. To generate X, repeat the following: generate Y from the density r and an independent $U[0,1]$ variate U, until $Ut(Y) \le f(Y)$. Then, return $X = Y$. This is the *acceptance–rejection* method [2]. The number R of turns into the 'repeat' loop is one plus a geometric random variable with parameter $1/a$, so $E[R] = a$. Thus, we want a to be as small (close to 1) as possible, that is, we want to minimize the area between f and the *hat function* t. In practice, there is usually a compromise between bringing 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 $q(x) \le f(x)$ for all x. To verify the condition $Ut(Y) \le f(Y)$, first check if $Ut(Y) \le q(Y)$, in which case there is no need to compute $f(Y)$.

The acceptance–rejection method is often applied after transforming 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 [2, 26].

A variant of acceptance–rejection, called *thinning*, is often used for generating events from a nonhomogeneous **Poisson process**. Suppose the process has rate $\lambda(t)$ at time t, with $\lambda(t) \le \overline{\lambda}$ for all t, where $\overline{\lambda}$ is a finite constant. One can generate Poisson pseudoarrivals at constant rate $\overline{\lambda}$ by generating interarrival times as i.i.d. exponentials of mean $1/\overline{\lambda}$. Then, a pseudoarrival at time t is accepted (becomes an arrival) with probability $\lambda(t)/\overline{\lambda}$ (i.e. if $U \le \lambda(t)/\overline{\lambda}$, where U is an independent $U[0,1]$), and rejected with probability $1 - \lambda(t)/\overline{\lambda}$. Nonhomogeneous Poisson processes can also be generated by inversion [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, 2, 6]. Recently, there has been an effort in developing *automatic* or *black box* algorithms for generating variates from an arbitrary (known) density, based on acceptance–rejection with a transformed density [8, 25, 26]. Another important class of general **nonparametric methods** is those that sample directly from a smoothened version of the **empirical distribution** of a given data set [8, 10]. These methods shortcut the fitting of a specific type of distribution to the data. Perhaps the best currently available software for nonuniform variate generation is [25].

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* [4, 7, 19, 29, 30, 33]. The issue of measuring the uniformity (or discrepancy) of P_n is the same as for the set Ψ_s of a RNG. Moreover, RNG construction methods such as LCGs and linear recurrences modulo 2 with linear output transformations can be used as well for QMC: It suffices to take $P_n = \Psi_s$ in s dimensions, with $n = |\Psi_s|$ much smaller than for RNGs. The same methods can be used for measuring the discrepancy of P_n for QMC and of Ψ_s for RNGs. Point sets constructed via an LCG and a Tausworthe generator are actually special cases of lattice rules and digital nets, respectively. A *low-discrepancy 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 (high uniformity) for all n, as $n \to \infty$ [7, 29, 30]. 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 [4, 18, 30].

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

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

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 $\mathcal{F}$. 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*. 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 [4, 7, 30] and other references therein. Some of the corresponding discrepancy measures are much easier to compute than the rectangular star discrepancy. Like for RNGs, different discrepancy measures are often used in practice for different types of point set constructions, for computational efficiency considerations (e.g. variants of the spectral test for lattice rules and measures of equidistribution for digital nets).

Specific sequences P_∞ have been constructed with rectangular star discrepancy $D(P_n) = O(n^{-1}(\ln n)^s)$ when $n \to \infty$ for fixed s, yielding a convergence rate of $O(n^{-1}(\ln n)^s)$ for the worst-case error if $\|f - \mu\| < \infty$ [29, 34]. This is *asymptotically* better than MC. But for reasonable values of n, the QMC bound is smaller only for small s (because of the $(\ln n)^s$ factor), so other justifications are needed to explain why QMC methods really work (they do!) for some high-dimensional real-life applications. The bound (5) is only for the worst case – the actual error can be much smaller for certain functions. In particular, even if s is large, f can sometimes be well approximated by a sum of functions defined over some low-dimensional subspaces of $[0, 1)^s$. It suffices, then, that the projections of P_n over these subspaces has low discrepancy, which is much easier to achieve than getting a small $D(P_n)$ for large s. This has motivated the introduction of discrepancy measures that give more weight to projections of P_n over selected (small) subsets of coordinates [18, 19, 30].

One drawback of QMC with a deterministic point set P_n is that no statistical error estimate is available (and error bounds of the form (5) are practically useless). *Randomized* QMC methods have been introduced for this reason. The idea is to randomize P_n in a way 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 (e.g. one can simply shift the entire point set P_n randomly, modulo 1, for each coordinate). Then, (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 compute a confidence interval for μ [19, 32]. This turns QMC into a variance reduction method [18, 19]. Bounds and expressions for the variance have been developed (to replace (5)) for certain classes of functions and randomized point sets [19, 32]. In some cases, under appropriate assumptions on f, the variance has been shown to converge faster than for MC, that is, faster than $O(1/n)$. For example, a *scrambling* method introduced by Owen for a class of digital nets gives $\mathrm{Var}[\hat{\mu}_n] = O(n^{-3}(\ln n)^s)$ if the mixed partial derivatives of f are Lipschitz [31]. Simplified (and less costly) versions of this scrambling have also been proposed [19].

Acknowledgements

This work has been supported by the Natural Sciences and Engineering Research Council of Canada Grant No. ODGP0110050 and NATEQ-Québec grant No. 02ER3218. R. Simard and C. Lemieux helped in improving the manuscript.

References

[1] Bratley, P., Fox, B.L. & Schrage, L.E. (1987). *A Guide to Simulation*, 2nd Edition, Springer-Verlag, New York.

[2] Devroye, L. (1986). *Non-Uniform Random Variate Generation*, Springer-Verlag, New York.

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

[4] Fang, K.-T., Hickernell, F.J. & Niederreiter, H., eds (2002). *Monte Carlo and Quasi-Monte Carlo Methods 2000*, Springer-Verlag, Berlin.

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

[6] Gentle, J.E. (2003). *Random Number Generation and Monte Carlo Methods*, 2nd Edition, Springer, New York.

[7] Hellekalek, P. & Larcher, G., eds (1998). *Random and Quasi-Random Point Sets*, Volume 138 of Lecture Notes in Statistics, Springer, New York.

[8] Hörmann, W. & Leydold, J. (2000). Automatic random variate generation for simulation input, in *Proceedings of the 2000 Winter Simulation Conference*, J.A. Joines, R.R. Barton, K. Kang, & P.A. Fishwick, eds, IEEE Press, Pistacaway, NJ, pp. 675–682.

[9] Knuth, D.E. (1998). *The Art of Computer Programming*, 3rd Edition, *Volume 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA.

[10] Law, A.M. & Kelton, W.D. (2000). *Simulation Modeling and Analysis*, 3rd Edition, McGraw-Hill, New York.

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

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

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

[14] L'Ecuyer, P. (1998). Random number generation, in *Handbook of Simulation*, J. Banks, ed., Wiley, New York, pp. 93–137, Chapter 4.

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

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

[17] L'Ecuyer, P. (2001). Software for uniform random number generation: Distinguishing the good and the bad, *Proceedings of the 2001 Winter Simulation Conference*, IEEE Press, Pistacaway, NJ, pp. 95–105.

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

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

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

[21] L'Ecuyer, P. & Simard, R. (2002). TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators, Software User's Guide.

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

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

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

[25] Leydold, J. & Hörmann, W. (2002). UNURAN–A Library for Universal Non-Uniform Random Number Generators, Available at http://statistik.wu-wien.ac.at/unuran.

[26] Leydold, J., Janka, E. & Hörmann, W. (2002). Variants of transformed density rejection and correlation induction, in *Monte Carlo and Quasi-Monte Carlo Methods 2000*, K.-T. Fang, F.J. Hickernell, & H. Niederreiter, eds, Springer-Verlag, Berlin, pp. 345–356.

[27] Marsaglia, G. (1996). DIEHARD: a Battery of Tests of Randomness, See http://stat.fsu.edu/geo/diehard.html.

[28] 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.

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

[30] Niederreiter, H. & Spanier, J., eds (2000). *Monte Carlo and Quasi-Monte Carlo Methods 1998*, Springer, Berlin.

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

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

[33] Sloan, I.H. & Joe, S. (1994). *Lattice Methods for Multiple Integration*, Clarendon Press, Oxford.

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

(*See also* **Affine Models of the Term Structure of Interest Rates**; **Derivative Pricing, Numerical Methods**; **Derivative Securities**; **Frailty**; **Graduation**; **Risk-based Capital Allocation**; **Simulation Methods for Stochastic Differential Equations**)

PIERRE L'ECUYER