# Random Number Generators With Period Divisible by a Mersenne Prime

Richard P. Brent[1] and Paul Zimmermann[2]

[1] Oxford University Computing Laboratory,
Wolfson Building, Parks Road,
Oxford OX1 3QD, UK

[2] LORIA/INRIA Lorraine
615 rue du jardin botanique
BP 101, F-54602 Villers-lès-Nancy
France

**Abstract.** Pseudo-random numbers with long periods and good statistical properties are often required for applications in computational finance. We consider the requirements for good uniform random number generators, and describe a class of generators whose period is a Mersenne prime or a small multiple of a Mersenne prime. These generators are based on "almost primitive" trinomials, that is trinomials having a large primitive factor. They enable very fast vector/parallel implementations with excellent statistical properties.

## 1 Introduction

Pseudo-random numbers have been used in Monte Carlo calculations since the pioneering days of Von Neumann [39]. Monte Carlo and quasi-Monte Carlo methods are of great importance in computational finance [18, 24, 30], numerical integration [40], physics [12, 16, 25, 32], simulation [51], etc. With the increasing speed of vector processors and parallel computers, considerable attention must be paid to the statistical quality of random number generators. A program running on a supercomputer might use $10^8$ random numbers per second over a period of many hours (or months in the case of QCD calculations), so $10^{12}$ or more random numbers might contribute to the result. Small correlations or other deficiencies in the random number generator could easily lead to spurious effects and invalidate the results of the computation, see e.g. [16, 46].

Applications require random numbers with various distributions (e.g. normal, exponential, Poisson, ...) but the algorithms used to generate these random numbers almost invariably require a good uniform random number generator –

Revision of invited paper to appear in *Proc. ICCSA* (Montreal, May 2003).
This work was supported in part by the Oxford Centre for Computational Finance, the Oxford Supercomputing Centre, and EPSRC grant GR/N35366.
 rpb211

see for example [7, 14, 26, 42]. In this paper we consider only the generation of uniformly distributed numbers. Usually we are concerned with *real* numbers $u_n$ that are intended to be uniformly distributed on the interval $[0,1)$. Sometimes it is convenient to consider *integers* $U_n$ in some range $0 \le U_n < m$. In this case we require $u_n = U_n/m$ to be (approximately) uniformly distributed.[3]

Pseudo-random numbers generated in a deterministic fashion on a digital computer can not be truly random. What is required is that finite segments of the sequence $u_0, u_1, \cdots$ behave in a manner indistinguishable from a truly random sequence. In practice, this means that they pass all statistical tests that are relevant to the problem at hand. Since the problems to which a library routine will be applied are not known in advance, random number generators in subroutine libraries should pass a number of stringent statistical tests (and not fail any) before being released for general use.

A sequence $u_0, u_1, \cdots$ depending on a finite state must eventually be periodic, i.e. there is a positive integer $\rho$ such that $u_{n+\rho} = u_n$ for all sufficiently large $n$. The minimal such $\rho$ is called the *period.*

In §2 we consider desiderata for random number generators. Then, in §3, we outline some popular classes of random number generators, and consider to what extent they pass or fail our requirements. In §4 we consider generators whose period is (a small multiple of) a Mersenne prime, and describe some new generators based on "almost primitive" trinomials over GF(2). Finally, in §5 we briefly describe an implementation that meets the requirements of §2.

## 2 Requirements for good random number generators

The important requirements for a good pseudo-random number generator and its implementation in a subroutine library have been discussed in many surveys, e.g. [1, 5, 7, 11, 15, 25, 26, 33, 41]. Here we summarize them –

- *Uniformity.* The sequence of random numbers should pass statistical tests for uniformity of distribution.
- *Independence.* Subsequences of the full sequence $u_0, u_1, \cdots$ should be independent. Random numbers are often used to sample a $d$-dimensional space, so the sequence of $d$-tuples $(u_{dn}, u_{dn+1}, \ldots, u_{dn+d-1})$ should be uniformly distributed in the $d$-dimensional cube $[0,1]^d$ for all "small" values of $d$ (certainly for all $d \le 6$). For random number generators on parallel machines, the sequences generated on each processor should be independent.
- *Long period.* As mentioned above, a simulation might use $10^{12}$ random numbers. In such a case the period $\rho$ must exceed $10^{12}$. For many generators there are strong correlations between $u_0, u_1, \cdots$ and $u_m, u_{m+1}, \cdots$, where $m = \rho/2$ (and similarly for other simple fractions of the period). Thus, in practice the period should be *much* larger than the number of random numbers that will ever be used. A good rule of thumb is to use at most $\sqrt{\rho}$ numbers.

[3] For clarity we adopt the convention that upper case subscripted variables such as $U_n$ denote integers, and the corresponding lower case variables such as $u_n$ denote real numbers.

- *Repeatability.* For testing and development it is useful to be able to repeat a run with *exactly* the same sequence of random numbers as was used in an earlier run. This is usually easy if the sequence is restarted from the beginning ($u_0$). It may not be so easy if the sequence is to be restarted from some other value, say $u_m$ for a large integer $m$, because this requires saving the state information associated with the random number generator.
- *Portability.* Again, for testing and development purposes, it is useful to be able to generate *exactly* the same sequence of random numbers on two different machines, possibly with different wordlengths.
- *Skipping over terms.* If a simulation is to be run on a machine with several processors, or if a large simulation is to be performed on several independent machines, it is essential to ensure that the sequences of random numbers used by each processor are disjoint. Two methods of subdivision are commonly used. Suppose, for example, that we require 4 disjoint subsequences for a machine with 4 processors. One processor could use the subsequence $(u_0, u_4, u_8, \cdots)$, another the subsequence $(u_1, u_5, u_9, \cdots)$, etc. For efficiency each processor should be able to "skip over" the terms that it does not require. Alternatively, processor $j$ could use the subsequence $(u_{m_j}, u_{m_j+1}, \cdots)$, where the indices $m_0, m_1, m_2, m_3$ are sufficiently widely separated that the (finite) subsequences do not overlap. This requires some efficient method of generating $u_m$ for large $m$ without generating all the intermediate values $u_1, \ldots, u_{m-1}$.
- *Proper initialization.* The initialization of random number generators, especially those with a large amount of state information, is an important and often neglected topic. In some applications only a short sequence of random numbers is used after each initialization of the generator, so it is important that short sequences produced with different seeds are uncorrelated. Many current generators fail this test [19, 27].
- *Efficiency.* It should be possible to implement the method efficiently so that only a few arithmetic operations are required to generate each random number, all vector/parallel capabilities of the machine are used, and overheads such as those for subroutine calls are minimal.

## 3 Generalized Fibonacci generators

The Fibonacci numbers satisfy the recurrence $F_n = F_{n-1} + F_{n-2}$. However, it is easy to see that the corresponding recurrence $u_n = u_{n-1} + u_{n-2} \bmod 1$ does not give a satisfactory sequence of pseudo-random numbers [26, ex. 3.2.2.2].

We can generalize the Fibonacci recurrence to obtain "generalized Fibonacci" or "lagged Fibonacci" random number generators [22, 26, 44]. Marsaglia [33] considers generators $F(r, s, \theta)$ that satisfy

$$U_n = U_{n-r} \,\theta\, U_{n-s} \mod m$$

for fixed "lags" $r$ and $s$ ($r > s > 0$) and $n \geq r$. Here $m$ is a modulus (typically $2^w$ if $w$ is the wordlength in bits), and $\theta$ is some binary operator, e.g. addition, subtraction, multiplication or "exclusive or". We abbreviate these operators by $+, -, *$ and $\oplus$ respectively. Generators using $\oplus$ are also called "shift register" generators or "Tausworthe" generators [21, 48].

If $\theta$ is $+$ or $-$ then theoretical results can be deduced from the generating function

$$G(x) = \sum_{n=0}^{\infty} U_n \, x^n$$

that is given by $G(x) = P(x)/Q(x) \bmod m$, where $Q(x) = 1 - (x^r \; \theta \; x^s)$ and $P(x)$ is a polynomial of degree at most $r-1$, determined by the initial values $U_0, \ldots, U_{r-1}$. For example, if $m = 2$ and the initial values are not all zero, then the sequence has maximal period $2^r - 1$ if and only if $Q(x)$ is a primitive polynomial (mod 2).

If $m = 2^w$ and the lags $r$ and $s$ are chosen correctly, it is possible to obtain period

$$\rho = \begin{cases} 2^r - 1 & \text{if } \theta = \oplus, \\ 2^{w-1}(2^r - 1) & \text{if } \theta = \pm, \\ 2^{w-3}(2^r - 1) & \text{if } \theta = * \,. \end{cases}$$

The initial values must be odd for $\theta = *$, not all even for $\theta = \pm$, and not all zero for $\theta = \oplus$. For precise conditions, see [6, 35]. We see one advantage of the generalized Fibonacci generators over most other classes of generators – the period can be made very large by choosing $r$ large.

Marsaglia [33] reports the results of statistical tests on the generators $F(17,5,\theta)$, $F(31,13,\theta)$ and $F(55,24,\theta)$. The results for $\theta = \oplus$ are poor – several tests are failed. All tests are passed for the generators $F(607,273,\theta)$ and $F(1279,418,\theta)$, so the conclusion is that $\oplus$ generators should only be used if the lag $r$ is large.

Marsaglia's results for $\theta = \pm$ are good with one exception – the generators with $r \le 55$ fail the "Birthday Spacings" test. Our conclusion is that these generators are probably acceptable if $r$ and $s$ are sufficiently large (not necessarily as large as for the $\oplus$ generators).

With the generalized Fibonacci recurrence

$$U_n = U_{n-r} \pm U_{n-s} \mod m \; ,$$

only four of the six orderings of $(U_n, U_{n-r}, U_{n-s})$ can occur. A statistical test based on this fact will "fail" any $F(r,s,\pm)$ generator. Even if $r$ and $s$ are not assumed to be known, the test can check all possible $r$ and $s$ satisfying $0 < s < r < B$ say, where $B$ is a prescribed bound. We call such a test a "Generalized Triple" test. Clearly the existence of such tests is one reason for choosing $r$ to be large.

Marsaglia's results indicate that generalized Fibonacci generators with $\theta = * \bmod m$ are acceptable. Unfortunately, it is more difficult to implement multiplication (mod $m$) than addition/subtraction (mod $m$) because of the requirement for a double-length product, unless $m$ is small enough for $m^2$ to be representable in single-precision.

There are several ways to improve the performance of generalized Fibonacci generators on the Birthday Spacings and Generalized Triple tests. The simplest is to include small odd integer multipliers $\alpha$ and $\beta$ in the generalized Fibonacci recurrence, e.g.

$$U_n = \alpha U_{n-r} + \beta U_{n-s} \mod m \ .$$

The theory goes through with minor modifications. Note that, because $\alpha$ and $\beta$ are odd, the values of $U_n$ mod 2 are unchanged. By [6, Theorem 3], the period is $2^{w-1}(2^r-1)$ if $m = 2^w$, provided the trinomial $x^r + x^s + 1$ is primitive (mod 2) and $U_0, \ldots, U_{r-1}$ are not all even. Other ways to improve statistical properties (at the expense of speed) are to include more terms in the linear recurrence [28, 29], to discard some members of the sequence [32], or to combine two or three generators in various ways [13, 26, 33, 38, 49].

In the following we consider additive generalized Fibonacci generators whose period $\rho$ can be proved to be a small multiple of a large prime $p$. For simplicity, we consider only the (period of the) least-significant bit, so all arithmetic can be considered over the finite field GF(2) of two elements. Because carries propagate from the least-significant bit into higher-order bits, the overall period will be $2^k\rho$ for some integer $k$ depending on the wordlength $w$ (usually $k = w - 1$).

## 4 Generators with Mersenne prime period

We recall some definitions relating to polynomials over finite fields. A polynomial $P(x)$ is *reducible* if it has nontrivial factors; otherwise it is *irreducible*. A polynomial $P(x)$ of degree $r > 1$ is *primitive* if $P(x)$ is irreducible and $x^j \neq 1 \bmod P(x)$ for $0 < j < 2^r - 1$. For other definitions and algorithmic details see [9, 31, 38].

Known algorithms for verifying, in time polynomial in $r$, that the period of a generator is (a multiple of) $2^r - 1$, require knowledge of the prime factorization of $2^r - 1$. Complete factorizations are unknown for most $r > 700$ (see [52] for small $r$). However, this problem disappears if $r$ is the exponent of a Mersenne prime (i.e. $2^r - 1$ is prime), because then the prime factorization is trivial. In this case a polynomial is primitive if and only if it is irreducible, and irreducibility can be tested in time $O(r^2)$.

We restrict our attention to generators whose period $\rho$ is a Mersenne prime $M_r = 2^r - 1$ or a small multiple of $M_r$. The integer $r$ is called a *Mersenne exponent*, and is itself prime. All Mersenne exponents less than $10^7$ are known [20].

### 4.1 Implications of Swan's theorem

Generators with Mersenne prime period have been discussed by several authors, e.g. [5, 23, 36, 37]. However, not all of the known large Mersenne primes have been considered, because Swan's Theorem[4] rules out about half of the candidates if we wish to use a generator based on a primitive trinomial.

[4] Swan's Theorem is a rediscovery of earlier results – see Swan [47, p. 1099] and von zur Gathen [17]. An elementary proof is given by Berlekamp [2, p. 159].

**Theorem 1.** Swan [47, Corollary 5]. *Suppose* $r > s > 0$, $r - s$ *odd. Then* $x^r + x^s + 1$ *has an even number of irreducible factors over* GF(2) *if and only if one of the following holds:*

*a)* $r$ *even,* $r \neq 2s$, $rs/2 \bmod 4 \in \{0, 1\}$*;*
*b)* $2r \neq 0 \bmod s$, $r = \pm 3 \bmod 8$*;*
*c)* $2r = 0 \bmod s$, $r = \pm 1 \bmod 8$.

If both $r$ and $s$ are odd, we can replace $s$ by $r - s$ (leaving the number of irreducible factors unchanged) and apply Swan's theorem. Since a polynomial that has an even number of irreducible factors is reducible, we have:

**Corollary 1.** *If* $r$ *is prime,* $r = \pm 3 \bmod 8$, $s \neq 2$, $s \neq r - 2$, *then* $x^r + x^s + 1$ *is reducible over* GF(2).

Corollary 1 shows that there are no irreducible trinomials with $r = \pm 3 \bmod 8$ (except possibly for $s = 2$ or $r - 2$, but the only known cases are $r = 3, 5$). This appears to prevent us from finding generators (based on trinomials) whose periods are (small multiples of) the corresponding Mersenne primes. One solution is to use pentanomials instead of trinomials [28, 29], but this incurs a significant speed penalty.

### 4.2 Almost primitive trinomials

Fortunately, there is a way around Swan's theorem without paying a large speed penalty. Blake, Gao and Lambert [4], following Tromp, Zhang and Zhao [50], suggest searching for trinomials of slightly larger degree (say degree $r + \delta$) that have a primitive factor of degree $r$, and give examples for $r \leq 500$. For most purposes, such trinomials are almost as good as primitive trinomials of degree $r$. We say that a polynomial $P(x)$ of degree $r + \delta$ is *almost primitive (almost irreducible)* if $P(0) \neq 0$ and $P(x)$ has a primitive (irreducible) factor of degree $r$, where $0 \leq \delta < r$.

Inspired by Blake *et al.*, we have conducted a search for almost primitive trinomials with $r$ a Mersenne exponent. The search is now complete for all Mersenne exponents $r < 2976221$. In all cases of Mersenne exponent $r = \pm 3 \bmod 8$ with $5 < r < 10^7$, we have found trinomials of degree $r + \delta$ that have a primitive polynomial factor of degree $r$, for some $\delta$ such that $2 \leq \delta \leq 12$. The results are summarized in Table 1, which also includes the "traditional" case $\delta = 0$, using data from [9, Table 3 and update] and the results of previous authors [23, 28, 45, 53].

The entries for $\delta > 0$ in Table 1 were computed using an extension of the algorithm for $\delta = 0$ that is described in [9]. Details of the extension are given in [10, §4].

For example, in the case $r = 13$, a primitive trinomial of degree 13 does not exist, but the trinomial $x^{16} + x^3 + 1$ has a primitive factor of degree 13. In fact

$$x^{16} + x^3 + 1 = (x^3 + x^2 + 1)(x^{13} + x^{12} + x^{11} + x^9 + x^6 + x^5 + x^4 + x^2 + 1) ,$$

| $r$ | $\delta$ | $s$ | $f$ | $r$ | $\delta$ | $s$ | $f$ |
|---|---|---|---|---|---|---|---|
| 2 | 0 | 1 | 1 | 3 | 0 | 1 | 1 |
| 5 | 0 | 2 | 1 | 7 | 0 | 3 | 1 |
| 13 | 3 | 3 | 7 | 17 | 0 | 6 | 1 |
| 19 | 3 | 3 | 7 | 31 | 0 | 13 | 1 |
| 61 | 5 | 17 | 31 | 89 | 0 | 38 | 1 |
| 107 | 2 | 17 | 3 | 127 | 0 | 63 | 1 |
| 521 | 0 | 168 | 1 | 607 | 0 | 273 | 1 |
| 1279 | 0 | 418 | 1 | 2203 | 3 | 355 | 7 |
| 2281 | 0 | 1029 | 1 | 3217 | 0 | 576 | 1 |
| 4253 | 8 | 1806 | 255 | 4423 | 0 | 2098 | 1 |
| 9689 | 0 | 4187 | 1 | 9941 | 3 | 1077 | 7 |
| 11213 | 6 | 227 | 63 | 19937 | 0 | 9842 | 1 |
| 21701 | 3 | 7587 | 7 | 23209 | 0 | 9739 | 1 |
| 44497 | 0 | 21034 | 1 | 86243 | 2 | 2288 | 3 |
| 110503 | 0 | 53719 | 1 | 132049 | 0 | 54454 | 1 |
| 216091 | 12 | 42930 | 3937 | 756839 | 0 | 279695 | 1 |
| 859433 | 0 | 288477 | 1 | 1257787 | 3 | 74343 | 7 |
| 1398269 | 5 | 417719 | 21 | 2976221 | 8 | 1193004 | 85 |
| 3021377 | 0 | 1010202 | 1 | 6972593 | 0 | 3037958 | 1 |

**Table 1.** Some primitive and almost primitive trinomials over GF(2): $x^{r+\delta} + x^s + 1$ has a primitive factor of degree $r$; all irreducible factors are primitive; the period $\rho = f(2^r - 1)$.

where both factors are primitive over GF(2), with relatively prime periods $2^3-1$ and $2^{13}-1$. Thus, a random bit generator $U_n = U_{n-16} \oplus U_{n-3}$ will have period $(2^{13}-1)(2^3-1)$, provided the initial vector $(U_0, \ldots, U_{15})$ is in "general" position. The period will be only $2^3-1$ if the initial vector lies in a subspace defined by $U_n \oplus U_{n-2} \oplus U_{n-3} = 0$ for $3 \le n \le 15$, because then the simpler recurrence $U_n = U_{n-3} \oplus U_{n-2}$, corresponding to the factor $x^3 + x^2 + 1$, is satisfied for all $n \ge 3$. We can rule this out very easily by taking $U_0 = U_1 = U_2 = U_3 = 1$, or in general $U_0 = \cdots = U_\delta = 1$.

A larger example is $r = 216091$, $\delta = 12$. We have

$$x^{216103} + x^{42930} + 1 = (x^{12} + x^{11} + x^5 + x^3 + 1)P(x) \ ,$$

where $P(x)$ is a primitive polynomial of degree 216091. The factor of degree 12 splits into a product of two primitive polynomials, $x^5 + x^4 + x^3 + x + 1$ and $x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$. The contribution to the period from these factors is $(2^5-1)(2^7-1) = 3937$.

For all but one of the entries in Table 1, the period $\rho$ of the corresponding recurrence satisfies

$$\rho = f(2^r - 1) > 2^{r+\delta-1}$$

provided the initial vector is in general position (this is easy to verify). Thus, the period is greater than what can be obtained for any polynomial of degree less than $r + \delta$. The exception is the entry for $r = 2976221$, where the factor of degree $\delta$ has period $85 = 255/3$.

Table 1 includes all Mersenne exponents $r < 10^7$, see [20]. For the largest known Mersenne exponent, $r = 13466917$, we have not yet started an extensive computation, but we have shown in [10] that $\delta \geq 3$ and $\delta \neq 4$.

## 5 Implementation

Many random number generators based on primitive trinomials have been documented in the literature, see e.g. [23, 26, 36, 43], but the implementations are usually for a fixed trinomial. The choice of this trinomial involves a tradeoff. Larger values of $r$ give generators with better statistical properties, but the space requirement is proportional to $r$, and the time required for initialization also increases with $r$. Thus, the optimal choice of a trinomial depends on the particular application and computing resources available.

The first author has implemented an open-source uniform pseudo-random number generator `ranut` [8] that automatically selects a primitive or almost primitive trinomial whose degree depends on the size of the working space allocated by the user, and then implements a generalized Fibonacci generator based on that trinomial. In the current implementation $127 \leq r \leq 3021377$; additional trinomials can be added very easily. We believe that `ranut` satisfies all the requirements of §2. It has been tested with Marsaglia's *Diehard* package [34] and no anomalous results have been observed.

## References

1. S. L. Anderson, Random number generators on vector supercomputers and other advanced architectures, *SIAM Review* **32** (1990), 221–251.
2. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
3. I. F. Blake, S. Gao and R. J. Lambert, Constructive problems for irreducible polynomials over finite fields, in *Information Theory and Applications*, *Lecture Notes in Computer Science* **793**, Springer-Verlag, Berlin, 1994, 1–23.
4. I. F. Blake, S. Gao and R. Lambert, Construction and distribution problems for irreducible trinomials over finite fields, in *Applications of Finite Fields* (D. Gollmann, ed.), Oxford, Clarendon Press, 1996, 19–32. `www.math.clemson.edu/~sgao/pub.html`
5. R. P. Brent, Uniform random number generators for supercomputers, *Proc. Fifth Australian Supercomputer Conference*, Melbourne, December 1992, 95–104. `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub132.html`
6. R. P. Brent, On the periods of generalized Fibonacci recurrences, *Math. Comp.* **63** (1994), 389–401. `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub133.html`
7. R. P. Brent, Random number generation and simulation on vector and parallel computers, *Lecture Notes in Computer Science* **1470**, Springer-Verlag, Berlin, 1998, 1–20. `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub185.html`

8. R. P. Brent, *Some uniform and normal random number generators,* version 1.03 (January 2002). Available from `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/random.html`
9. R. P. Brent, S. Larvala and P. Zimmermann, A fast algorithm for testing reducibility of trinomials mod 2 and some new primitive trinomials of degree 3021377, *Math. Comp.* **72** (2003), 1417–1441. Preprint and update available at `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub199.html`
10. R. P. Brent and P. Zimmermann, Algorithms for finding almost irreducible and almost primitive trinomials, *Proc. Conference in Honour of Professor H. C. Williams*, Banff, Canada (May 2003), The Fields Institute, Toronto, to appear. Preprint available at `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub212.html`
11. P. D. Coddington, Random number generators for parallel computers, *The NHSE Review* **2** (1996). `http://nhse.cs.rice.edu/NHSEreview/RNG/PRNGreview.ps`
12. A. Compagner and A. Hoogland, Maximum-length sequences, cellular automata, and random numbers, *J. Computational Physics* **71** (1987), 391–428.
13. D. Coppersmith, H. Krawczyk and Y. Mansour, The shrinking generator, *Lecture Notes in Computer Science* **773** (1994), Springer-Verlag, Berlin, 22–39.
14. L. Devroye, *Non-Uniform Random Variate Generation,* Springer-Verlag, New York, 1986.
15. P. L'Ecuyer, Random numbers for simulation, *Comm. ACM* **33**, 10 (1990), 85–97.
16. A. M. Ferrenberg, D. P. Landau and Y. J. Wong, Monte Carlo simulations: hidden errors from "good" random number generators, *Phys. Review Letters* **69** (1992), 3382–3384.
17. J. von zur Gathen, Irreducible trinomials over finite fields, *Math. Comp.* **71** (2002), 1699–1734.
18. J. E. Gentle, *Random Number Generation and Monte Carlo Methods*, Springer-Verlag, Berlin, 1998.
19. P. Gimeno, *Problem with* `ran_array`, personal communication, 10 Sept. 2001.
20. GIMPS, *The Great Internet Mersenne Prime Search*, `http://www.mersenne.org/`
21. S. W. Golomb, *Shift register sequences,* Aegean Park Press, revised edition, 1982.
22. B. F. Green, J. E. K. Smith and L. Klem, Empirical tests of an additive random number generator, *J. ACM* **6** (1959), 527–537.
23. J. R. Heringa, H. W. J. Blöte and A. Compagner, New primitive trinomials of Mersenne-exponent degrees for random-number generation, *International J. of Modern Physics C* **3** (1992), 561–564.
24. P. Jaeckel, *Monte Carlo Methods in Finance*, John Wiley and Sons, 2002.
25. F. James, A review of pseudorandom number generators, *Computer Physics Communications* **60** (1990), 329–344.
26. D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* (third edition), Addison-Wesley, Menlo Park, CA, 1998.
27. D. E. Knuth, *A better random number generator*, January 2002, `http://www-cs-faculty.stanford.edu/~knuth/news02.html`
28. T. Kumada, H. Leeb, Y. Kurita and M. Matsumoto, New primitive $t$-nomials ($t = 3$, 5) over GF(2) whose degree is a Mersenne exponent, *Math. Comp.* **69** (2000), 811–814. Corrigenda: *ibid* **71** (2002), 1337–1338.
29. Y. Kurita and M. Matsumoto, Primitive $t$-nomials ($t = 3, 5$) over GF(2) whose degree is a Mersenne exponent $\leq 44497$, *Math. Comp.* **56** (1991), 817–821.
30. C. Lemieux and P. L'Ecuyer, On the use of quasi-Monte Carlo methods in computational finance, *Lecture Notes in Computer Science* **2073**, Springer-Verlag, Berlin, 2001, 607–616.

31. R. Lidl and H. Niederreiter, *Introduction to Finite Fields and their Applications*, Cambridge Univ. Press, Cambridge, second edition, 1994.
32. M. Lüscher, A portable high-quality random number generator for lattice field theory simulations, *Computer Physics Communications* **79** (1994), 100–110.
33. G. Marsaglia, A current view of random number generators, in *Computer Science and Statistics: The Interface*, Elsevier Science Publishers B. V.,1985, 3–10.
34. G. Marsaglia, *Diehard*, 1995. Available from `http://stat.fsu.edu/~geo/`
35. G. Marsaglia and L. H. Tsay, Matrices and the structure of random number sequences, *Linear Algebra Appl.* **67** (1985), 147–156.
36. M. Mascagni, M. L. Robinson, D. V. Pryor and S. A. Cuccaro, Parallel pseudorandom number generation using additive lagged-Fibonacci recursions, *Lecture Notes in Statistics* **106**, Springer-Verlag, Berlin, 1995, 263–277.
37. M. Matsumoto and T. Nishimura, Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, *ACM Transactions on Modeling and Computer Simulations* **8**, 1998, 3–30. Also `http://www.math.keio.ac.jp/~matumoto/emt.html`
38. A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997. `http://cacr.math.uwaterloo.ca/hac/`
39. J. von Neumann, Various techniques used in connection with random digits, *The Monte Carlo Method*, National Bureau of Standards (USA), Applied Mathematics Series **12** (1951), 36.
40. H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, CBMS-NSF Regional Conference Series in Applied Mathematics **63**, SIAM, Philadelphia, 1992.
41. S. K. Park and K. W. Miller, Random number generators: good ones are hard to find, *Comm. ACM* **31** (1988), 1192–1201.
42. W. P. Petersen, Some vectorized random number generators for uniform, normal, and Poisson distributions for CRAY X-MP, *J. Supercomputing* **1** (1988), 327–335.
43. W. P. Petersen, Lagged Fibonacci series random number generators for the NEC SX-3, *Internat. J. High Speed Computing* **6** (1994), 387–398.
44. J. F. Reiser, *Analysis of Additive Random Number Generators*, Ph. D. thesis and Technical Report STAN-CS-77-601, Stanford University, 1977.
45. E. R. Rodemich and H. Rumsey, Jr., Primitive trinomials of high degree, *Math. Comp.* **22** (1968), 863–865.
46. L. N. Shchur, J. R. Heringa and H. W. J. Blöte, Simulation of a directed random-walk model: the effect of pseudo-random-number correlations, *Physica A* **241** (1997), 579.
47. R. G. Swan, Factorization of polynomials over finite fields, *Pacific J. Mathematics* **12** (1962), 1099–1106.
48. R. C. Tausworthe, Random numbers generated by linear recurrence modulo two, *Math. Comp.* **19** (1965), 201–209.
49. S. Tezuka, Efficient and portable combined Tausworthe random number generators, *ACM Trans. on Modeling and Computer Simulation* **1** (1991), 99–112.
50. J. Tromp, L. Zhang and Y. Zhao, Small weight bases for Hamming codes, *Theoretical Computer Science* **181**(2), 1997, 337–345.
51. I. Vattulainen, T. Ala-Nissila and K. Kankaala, Physical tests for random numbers in simulations, *Phys. Review Letters* **73** (1994), 2513–2516.
52. S. S. Wagstaff *et al.*, *The Cunningham Project*, `http://www.cerias.purdue.edu/homes/ssw/cun/`
53. N. Zierler, Primitive trinomials whose degree is a Mersenne exponent, *Information and Control* **15** (1969), 67–69.