Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Linear congruential generators of order K>1

Pierre L'Ecuyer and François Blouin
Département d'informatique
Université Laval
Ste-Foy, Qué., Canada, G1K 7P4.

ABSTRACT

We present a class of random number generators defined by a linear congruential recursion of order $k > 1$, modulo m, and whose period can attain $m^k - 1$. The spectral test can be extended to this class of generators, and permits the computation of the distance between the successive t-dimensional hyperplanes in which lie all the t-tuples of successive values. In terms of the spectral test, these generators could be almost as good as the best regular (order 1) linear congruential generators with modulus close to $m^k - 1$. We discuss the implementation of computer programs to search for maximal period generators of this kind and to apply the (generalized) spectral test. We also present the results of a search to find good vectors of multipliers, for some values of the modulus m, and give an example of a portable implementation.

1. INTRODUCTION

So called "random number" generators on computers are in fact simple deterministic functions that produce a periodic sequence of numbers (see Knuth (1981) or Bratley, Fox and Schrage (1987)). The most widely used form is the *linear congruential generator* (LCG), whose state is a non-negative integer that evolves according to the recursion

$$X_n := (aX_{n-1} + c) \text{ MOD } m. \tag{1}$$

where the *modulus* m and the *multiplier* $a < m$ are positive integers, and the constant $c < m$ is a non-negative integer. One often takes $c = 0$, which gives the (common) special case called the *multiplicative linear congruential generator* (MLCG), whose *state space* is $\{1, 2, \ldots, m-1\}$. The value of the state X_n is usually transformed into a real value between 0 and 1 by

$$U_n := X_n/m, \tag{2}$$

and one hopes that $\{U_n, n = 0, 1, 2, \ldots\}$ behaves somewhat like a sequence of *i.i.d.* $U(0,1)$ random variables.

The period of a generator is the smallest positive integer p such that

$$X_{n+p} = X_n$$

for all $n > \nu$, for some integer $\nu > 0$. The maximal period that can be attained by a MLCG is $p = m - 1$. It is in fact attained when m is prime and a is a primitive element modulo m (see Knuth (1981), page 19). Finding maximal period MLCGs is quite easy. For $m = 2^{31} - 1$, for instance, about 25% of the values of a between 1 and m are primitive elements modulo m.

It is also well known that all the tuples of successive values produced by a given LCG have a special lattice structure. Marsaglia (1968) pointed out this geometric property, derived from previously known results in the field of the geometry of numbers. More precisely, for any integer $t > 0$, all t-tuples $(U_{n+1}, \ldots, U_{n+t})$ of successive values generated by a LCG lie on a set of at most $(t!m)^{1/t}$ equidistant parallel hyperplanes in the t-dimensional hypercube $(0,1)^t$. This is usually viewed as a major weakness of LCGs (see Bratley, Fox and Schrage (1987)). When m is too small, it is obviously a strong limitation to the t-dimensional uniformity. Many microcomputer generators, for instance, present obvious linear patterns when we use them to place points in the unit square, or place points in the unit cube and display the projection of a thin slice of that cube (see L'Ecuyer (1987)). The structure is much less apparent when a and m are well chosen. There is an algorithm, called the *spectral test* (Knuth (1981)), that permits to compute the actual distance between successive hyperplanes, in the t-dimensional hypercube, for a given LCG. Fishman and Moore (1986) and L'Ecuyer ((1986) and (1988)) have performed extensive computer searches to find the "best" MLCGs according to a criterion based on the spectral test in dimensions 2 to 6, for a given set of prime moduli m (all less than 2^{31}).

If m is very large (say $m > 2^{100}$, for instance) and a is well chosen, the distance between successive hyperplanes could be so small that the lattice structure cannot produce any bad effect, for all practical purposes. The lattice structure could then be viewed as a desirable property, since it gives us a theoretical insurance that at least over its entire period, the generator has no bad correlation structure, and leaves no holes in the t-dimensional hypercube. One problem

is that generators with such large values of m are difficult to implement efficiently on contemporary 32-bit (or less) machines.

L'Ecuyer ((1986) and (1988)) has proposed a simple and efficient method for combining two or more MLCGs, to obtain a generator whose period is the least common multiple of the individual periods. The only mathematically-demonstrated improvement of the combined generators over their components is a (much) longer period. Beyond this, the combination is an intuitively appealing heuristic supported by empirical tests and by the fact that no structure is apparent in the geometric behavior of the proposed combined generators, even if the lattice structure of their individual components can be made visible.

In this paper, we consider a more general form of MLCGs, where the value X_n is computed as a linear combination of the k previous values, modulo m. Hence (1) is replaced by

$$X_n := (a_1 X_{n-1} + \cdots + a_k X_{n-k}) \text{ MOD } m \qquad (3)$$

where the a_i are integers between $-m$ and m (strictly), and $a_k \neq 0$. We call these generators MLCGs of order k. As mentioned in Knuth (1981), page 28, when m is prime, it is possible to find multipliers $a_1, \ldots, a_k$ such that the generator defined by (3) has period $m^k - 1$ (full period). Furthermore, the proportion of vectors $(a_1, \ldots, a_k)$ giving rise to full period generators is usually quite large. For example, if $m = 2^{31} - 1$ and $k = 2$, that proportion is about 12.45%. Sufficient conditions for a generator defined by (3) to have a period of length $m^k - 1$ are recalled in section 2 of this paper. Implementing computer programs to verify these conditions is not an easy task; we explain how it can be done, and describe an actual implementation.

A seemingly different class of generators, the (well studied) Generalized Feedback Shift Register (GFSR) generators (Fushimi and Tesuka (1983), Fushimi (1988)), are in fact based on MLCGs generators of large order k, with modulus $m = 2$. To produce b-bits integers, they generate b streams of bits in parallel, using the same basic recursion, but starting with different seeds. Usually, the basic MLCG has only two a_i equal to 1, the others beeing zero, and the recursion becomes

$$X_n := X_{n-j} \text{ XOR } X_{n-k} \qquad (4)$$

where XOR represents an addition modulo 2. These generators are fast and have excellent (proven) theoretical properties, provided k and j are well chosen. However, they use a substantial amount of space to keep the state (which is a b by k array of bits), and the choice of a proper seed (i.e. initial state) is not trivial and somewhat time consuming. This can be a serious drawback for simulation applications, especially when many different generators must be used in parallel (see Bratley et al. (1987)). In this paper, instead of a small m and large k, we consider large values of m and small values of k. The proposed generators are a good compromise between speed and space efficiency. They are trivial to initialize, and it is relatively easy to split their sequence into a number of disjoints subsequences, or to jump ahead into their sequence.

The lattice structure property mentioned above also applies to MLCGs of order $k > 1$, and it is possible to generalize the spectral test to this class of generators. The upper bound on the number of parallel hyperplanes in the t-dimensional hypercube, for $t > k$, is now $(t!(m^k-1))^{1/t}$ instead of $(t!m)^{1/t}$ (for $t \leq k$, it is obviously m). Hence, with a MLCG of order k and modulus m, one can obtain the good theoretical properties of the best order 1 MLCGs based on a prime modulus approximately equal to $m^k - 1$ in dimensions higher than k, and equidistribution in dimensions $\leq k$, while avoiding the implementation problems associated with the MLCGs that use large moduli.

To the best of our knowledge, this paper presents the first systematic search to find good MLCGs of order $k > 1$. In section 3, we describe an efficient way to perform such a computer search, and present the results of a search to find good MLCGs of order 2 to 7, for some prime values of m. In section 4, we give an example of a portable implementation.

2. FINDING MAXIMAL PERIOD GENERATORS

The maximal possible period for a generator defined by the recursion (3) is the size of the state space: $p = m^k - 1$. Let $f(x)$ be the characteristic polynomial of the recursion (3):

$$f(x) = x^k - \sum_{i=1}^{k} a_i x^{k-i}. \tag{5}$$

Theorem 1 below gives sufficient conditions for this maximal period to be attained. For the remainder of this paper, we assume that m is prime, and let

$$r = \frac{m^k - 1}{m - 1} = m^{k-1} + m^{k-2} + \cdots + m + 1. \tag{6}$$

THEOREM 1.
If m is prime and if the following conditions are satisfied, then the generator defined by (3) has period $p = m^k - 1$:

(a) $((-1)^{k+1} a_k)^{(m-1)/q} \bmod m \neq 1$ for each prime factor q of $m - 1$;

(b) $((x^r \bmod f(x)) \bmod m) = ((-1)^{k+1} a_k) \bmod m$;

(c) $((x^{r/q} \bmod f(x)) \bmod m)$ has degree > 0 for each prime factor q of r. ∎

Unfortunately, these conditions are not so easy to verify in practice. The major difficulty is the factorization of r: factoring large numbers is (under the present state of the art) a very hard problem, for which no polynomial-time algorithm is known (Brassard and Bratley (1987)). Take for instance $m = 2^{31} - 1$ and $k = 7$; then r has 56 decimal digits and its factorization can take hours of CPU time on a mainframe, even with the currently most sophisticated algorithms (Montgomery (1987), Silverman (1987)). When k is even, $m + 1$ is always a factor of r, and one also has $r = (m^{k/2} + 1)(m^{k/2} - 1)/(m - 1)$. When k is odd, it might happen that r is prime, which is a very desirable situation, since besides eliminating the factorization problem, this simplifies the verification of condition (c) in Theorem 1. Thus, it could be a really good idea to select m and k such that r is prime. The difficulties don't stop there: primality testing is also a problem. The best currently known exact algorithms (see, for instance, Cohen and Lenstra (1987)) can test numbers of up to about 200 decimal digits, but are quite involved and difficult to implement. For the results presented below, we have used a *probabilistic* primality test proposed by Rabin (see Brassard and Bratley (1987) or Rabin (1980)). This test consists of j iterations of a loop, and at each iteration, one tries (randomly) to find a "proof", based on Fermat's theorem, that the number n to be checked is *not* prime. If n is prime, then the test always says so, while if n is odd and composite, the test can also say (erroneously) that n is prime, but only with probability smaller than 4^{-j}. The conditional probability that n is effectively prime when the test says so depends on the (constant) *a priori* probability that n is prime, through the theorem of Bayes. It can be made arbitrarily small by increasing j. In practice, one can take say $j = 1000$, and the probability of a bad answer by the test will probably be smaller than the probability of an error by the computer! This is what we did.

3. THE SPECTRAL TEST AND RESULTS OF A COMPUTER SEARCH

We have implemented a computer program to verify the conditions of Theorem 1, and a generalization of the spectral test (Knuth (1981)) to MLCGs of order k. Such a generalization has been discussed previously by Grube (1973). The programs are written in Modula-2, and they use another Modula-2 package (L'Ecuyer, Perron and Blouin (1988)) for doing arithmetic with arbitrary large integers. They work for any values of m and $k \leq 7$, but they assume that r has already been factorized. We also performed a computer search to find good generators of orders 2 to 7, for 16-bit and 32-bit computers. This work was done on a VAX-11/780 and a Microvax II computers.

For the spectral test, we use a criterion similar to those in Fishman and Moore (1986) and L'Ecuyer (1988). For a given order k, modulo m and vector of multipliers $a = (a_1, \ldots, a_k)$, let $d_t(k, m, a)$ be the maximal distance between adjacent parallel hyperplanes in dimension t, the maximum being taken over all families of parallel hyperplanes that cover all the points $(U_{n+1}, \ldots, U_{n+t})$, $n \geq 0$, where U_n is defined by (2). The smaller that maximal distance, the better it is, since this implies smaller empty "slices" in the hypercube. If the generator has maximal period, then $d_t(k, m, a) = 1/m$ for $1 \leq t \leq k$, since its sequence is k-distributed (every k-tuple $(X_{n+1}, \ldots, X_{n+k})$ except the zero tuple appears exactly once over the full period). A theoretical lower bound on $d_t(k, m, a)$ is given by:

$$d_t(k, m, a) \geq d_t^*(k, m) \stackrel{\text{def}}{=} \begin{cases} m^{-k/t}/\gamma_t & \text{if } t > k; \\ 1/m & \text{if } t \leq k. \end{cases} \tag{7}$$

where

$$\gamma_t = \begin{cases} (4/3)^{1/4} & \text{if } t = 2; \\ 2^{1/6} & \text{if } t = 3; \\ 2^{1/4} & \text{if } t = 4; \\ 2^{3/10} & \text{if } t = 5; \\ (64/3)^{1/12} & \text{if } t = 6; \\ 2^{3/7} & \text{if } t = 7; \\ 2^{1/2} & \text{if } t = 8. \end{cases} \tag{8}$$

Normalizing $d_t(k, m, a)$, we obtain the figure of merit

$$S_t(k, m, a) = \frac{d_t^*(k, m)}{d_t(k, m, a)} \tag{9}$$

which lies between 0 and 1. The higher the value, the better it is. In this paper, we decided (somewhat arbitrarily) to rate the maximal period generators according to their worst case measure

$$M_8(k, m, a) \stackrel{\text{def}}{=} \min_{k<t\leq 8} S_t(k, m, a). \tag{10}$$

To be selected, a generator must perform well in every dimension from 1 to 8.

For large m and $k > 1$, the number of possible generators is astronomical. An exhaustive search for the best vector a is out of the question and, after some empirical investigations, it appeared to us that the best way to find good generators is through a (Monte-Carlo) random search. Our general method of search was as follows:

1. Select m and k; factorize $m-1$ and r.

2. For $i = 1, \ldots, k$, choose b_i, c_i and h_i such that $-m < b_i \leq c_i < m$ and $0 \leq h_i \leq c_i - b_i$.

3. Select n and repeat n times:

   3a. For $i = 1, \ldots, k$, generate α_i from a discrete uniform distribution over $\{b_i, \ldots, c_i - h_i\}$.

   3b. Examine each vector of multipliers $a = (a_1, \ldots, a_k)$ such that $\alpha_i \leq a_i \leq \alpha_i + h_i$ for all i. To examine a vector, we verify the conditions of Theorem 1, and if the generator has maximal period, we apply the spectral test. The best ones according to M_8 are kept in the final results.

In step 3b, we take care to verify condition (a) of Theorem 1 only once for each distinct value of a_k. This is one of the main reasons why for each randomly generated point, we examine all the vectors contained in some hyperrectangle containing that point: many of them have the same value of a_k. In practice, one usually chooses smaller values of h_i as k increases. In our experiments, for $k = 7$, most h_i were set to zero, while for $k = 2$ we often had $h_i = 10$. To force one specific a_i to be zero, one selects $b_i = c_i = h_i = 0$, while to force $a_i^2 \leq m$, one selects $c_i = \lfloor\sqrt{m}\rfloor$ and $b_i = -\lfloor\sqrt{m}\rfloor$. The value of n would depend on the CPU time one is willing to spend for the search. For the results given here, we spent in total more than 500 hours of CPU time on a VAX-780 and a Microvax II. For some entries in tables 1–2, we examined around 3000 generators, while for others, we examined many more. For example, for the fourth entry of table 2, we examined about 10^7 generators.

Ease of implementation and speed are also important factors in the selection of a generator. In our case, things are better if some of the a_i are zero, so we looked for good generators with a few zero multipliers, too. In fact, it is usually possible to obtain maximal period generators with only a_k and one other a_i different from zero. Generators are also easier to implement when each a_i^2 is less than m, and m is representable as a regular integer on the target computer (L'Ecuyer (1988)). For this reason, we also looked for good generators for which every a_i^2 is smaller than m, some of them being equal to zero.

Partial results of our computer search appear in tables 1–2. For each entry, the first line gives the nonzero a_i, while the second and third give the values of $d_t(k, m, a)$ and $S_t(k, m, a)$, $t = 2, \ldots, 8$, respectively. For $k > 1$, each entry is the best (highest $M_8(k, m, a)$) that we found so far in its class, except for third and sixth entries in table 2 which were proposed by Grube (1973). In some cases, a class is defined by the constraints discussed in the previous paragraph. For $k = 1$, multipliers 219 in table 1 and 39373 in table 2 were analysed in previous work (L'Ecuyer (1988)). The first entry in table 2 shows the multiplier 742938285 suggested by Fishman and Moore (1986). For $k = 7$ we were not able to perform the search with $m = 2^{31} - 1$ because we don't know the factorization of r.

Table 1. Some generators with $m = 2^{15} - 19 = 32749$.
For the missing entries, $d_t = 1/m = .0000305$.

	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$
	$(a_1 = 219)$						
d_t	.00553	.03510	.08610	.141	.180	.229	.267
S_t	.9299	.7930	.7263	.7180	.7628	.7334	.7214
	$(a_1 = 32385, a_2 = -29316)$						
d_t		.00104	.00601	.0167	.0291	.0516	.0697
S_t		.8339	.7729	.7605	.8334	.7378	.7544
	$(a_1 = 180, a_2 = -176)$						
d_t		.00397	.00713	.0351	.0356	.0928	.0928
S_t		.2191	.6513	.3615	.6808	.4104	.5661
	$(a_1 = 25129, a_2 = 15046, a_3 = 28484)$						
d_t			.00047	.00213	.0058	.0112	.0198
S_t			.7403	.7440	.7384	.7720	.7224
	$(a_1 = 25716, a_3 = 931)$						
d_t			.00089	.00257	.00575	.0118	.0243
S_t			.3878	.6184	.7442	.7290	.5892
	$(a_1 = 15696, a_2 = 22006, a_3 = 24592, a_4 = 4283)$						
d_t				.000254	.00098	.00257	.0051
S_t				.7817	.7693	.7609	.7696
	$(a_1 = 538, a_4 = 16201)$						
d_t				.000905	.00250	.00361	.0062
S_t				.2193	.3024	.5408	.6316
	$(a_1 = 31939, a_5 = 24837)$						
d_t					.000926	.00246	.00248
S_t					.1445	.1798	.4294
	$(a_1 = 28779, a_6 = 28742)$						
d_t						.000916	.00267
S_t						.1093	.1087
	$(a_1 = 15707, a_7 = 30363)$						
d_t							.00090
S_t							.0875

Table 2. Some generators with $m = 2^{31} - 1 = 2147483647$.
For the missing entries, $d_t = 1/m = .000000000465 = 4.65\text{E-}10$.

	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$
	$(a_1 = 742938285)$(*Fishman and Moore (1986)*)						
d_t	.000023	.000802	.00453	.0133	.0259	.0553	.0682
S_t	.8673	.8607	.8627	.8319	.8341	.6239	.7067
	$(a_1 = 39373)$						
d_t	.000025	.000915	.00497	.0146	.0286	.0443	.0861
S_t	.7907	.7549	.7866	.7580	.7545	.7792	.5600
	$(a_1 = 337190270, a_2 = 268152554)$(*Grube (1973)*)						
d_t		.0000009	.0000268	.000259	.000839	.00245	.00492
S_t		.6127	.6766	.5792	.7155	.6550	.6674
	$(a_1 = 268152228, a_2 = -337190548)$						
d_t		.0000007	.0000212	.000192	.000782	.00209	.00445
S_t		.7410	.8543	.7843	.7683	.7654	.7381
	$(a_1 = 46339, a_2 = -46336)$						
d_t		.0000153	.0000252	.00310	.00310	.00310	.00504
S_t		.0351	.7211	.0484	.1935	.5161	.6522
	$(a_1 = 518175991, a_2 = 510332243, a_3 = 71324449)$(*Grube (1973)*)						
d_t			.0000001	.0000031	.0000286	.000101	.000456
S_t			.8182	.6528	.5843	.7369	.4906
	$(a_1 = 928528895, a_2 = 664504896, a_3 = 714296896)$						
d_t			.0000001	.0000026	.0000233	.000098	.000307
S_t			.8114	.7824	.7171	.7560	.7287
	$(a_2 = 518621249, a_3 = 666838593)$						
d_t			.0000006	.0000038	.0000266	.000122	.000337
S_t			.1509	.5345	.6292	.6095	.6648
	$(a_1 = 43825, a_2 = 45465, a_3 = 44940)$						
d_t			.0000129	.0000159	.0000222	.000124	.000434
S_t			.0065	.1284	.7529	.6008	.5155
	$(a_1 = 45187, a_3 = 45777)$						
d_t			.0000155	.0000155	.0000254	.000100	.000308
S_t			.0054	.1315	.6593	.7406	.7263
	$(a_1 = 1734821887, a_4 = 510316546)$						
d_t				.0000006	.0000030	.0000053	.0000257
S_t				.0488	.1542	.6583	.5941
	$(a_1 = 46310, a_4 = 41976)$						
d_t				.0000160	.0000160	.0000160	.0000292
S_t				.0017	.0291	.2160	.5233
	$(a_1 = 43102, a_5 = 46092)$						
d_t					.0000158	.0000158	.0000158
S_t					.0008	.0101	.0656
	$(a_1 = -45137, a_6 = 41275)$						
d_t						.0000163	.0000163
S_t						.0005	.0043

4. A PORTABLE IMPLEMENTATION

As an example, we give in figure 1 a *Pascal* implementation of a fifth order generator, with $m = 2^{31} - 1$, $k = 5$, $a_1 = 43102$, $a_5 = 46092$ and $a_2 = a_3 = a_4 = 0$. It is a maximal period generator and it appears as the next to last entry of table 2. The product modulo m is computed using the technique described in Bratley et al. (1987) and in section 3 of L'Ecuyer (1988). One has $m = 49823 \times a_1 + 12701 = 46591 \times a_5 + 11275$. The following code will work correctly provided all integers in the range $[-2^{31}, 2^{31} - 1]$ are well represented. The integer variables x1 to x5 are global and hold the current generator state. They must be initialized, before the first call, to values in the range $[0, 2^{31} - 2]$, with at least one $x_i \neq 0$. These five initial values constitute the *seed*. The function `Random` will return an integer in the range $[0, 2^{31} - 2]$. `DOUBLE` denotes the double precision type. Using this type, `Uniform01` will never return 0.0 or 1.0. However, it might return 1.0 if the result is returned as a single precision variable with 23-bit mantissa (the standard case for 32-bit machines). Specific seeds, spaced say 2^d values apart in the sequence for some integer d, can be computed along the lines of L'Ecuyer and Côté (1987).

```
VAR
   x1, x2, x3, x4, x5 : INTEGER;

FUNCTION Random : INTEGER;
   VAR
      h, p : INTEGER;
   BEGIN
   h := x5 DIV 46591;
   p := 46092 * (x5 - h * 46591) - h * 11275;
   IF p < 0 THEN p := p + 2147483647;

   x5 := x4;   x4 := x3;   x3 := x2;   x2 := x1;

   h := x1 DIV 49823;
   x1 := 43102 * (x1 - h * 49823) - h * 12701;
   IF x1 <= 0 THEN x1 := x1 + p
              ELSE x1 := x1 + (p - 2147483647);
   IF x1 < 0 THEN x1 := x1 + 2147483647;

   Random := x1
   END;

FUNCTION Uniform01 : DOUBLE;
   VAR
      Z : INTEGER;
   BEGIN
   Z := Random;
   IF Z = 0 THEN Z := 2147483647;
   Uniform01 := Z * 4.65661273077393D-10
   END;
```

Figure 1. A portable generator for 32-bit computers.

The generator given here has not been tested statiscally yet. It is presented only to illustrate our suggested implementation technique, and should not be viewed (yet) as a "recommended" generator.

There are alternative ways of obtaining a $U(0, 1)$ value from the state of the generator. This issue is discussed in Monahan (1985). The technique suggested here is rather simple. As suggested by Bruce Schmeiser (private communication), one can also transform x1—x5 into floating point numbers between zero and one before adding them modulo one to obtain the output of `Uniform01`. This additional "scrambling" will increase the number of possible output values, and might make the values look more random.

ACKNOWLEDGMENTS

This work has been supported by NSERC-Canada grant # A5463 and FCAR-Quebec grant # EQ2831 to the first author. We wish to thank Peter L. Montgomery, from Unisys, who provided us with a list of factorizations of $(m^k - 1)/(m - 1)$ for prime values of m near 2^{31}.

REFERENCES

Brassard, G. and Bratley P. (1987). *Algorithmique, conception et analyse* (in french), Masson, Paris, and Les Presses de l'Université de Montréal.

Bratley, P., Fox, B. L. and Schrage, L. E. (1987). *A Guide to Simulation*, second edition. Springer-Verlag, New York.

Cohen, H. and Lenstra, A. K. (1987). Implementation of a New Primality Test. *Mathematics of Computation*, **48**, 177, 103–121.

Fishman, G. S. and Moore III, L. S. (1986). An Exhaustive Analysis of Multiplicative Congruential Random Number Generators with Modulus $2^{31} - 1$. *SIAM J. on Scientific and Statistical Computing* **7**, 1, 24–45.

Fushimi, M and Tezuka, S. (1983). The k-Distribution of Generalized Feedback Shift Register Pseudorandom Numbers. *Communications of the ACM*, **26**, 7, 516–523.

Fushimi, M. (1988). Designing a Uniform Random Number Generator Whose Subsequences Are k-Distributed. *SIAM J. on Computing*, **17**, 1, 89–99.

Grube, A. (1973). Mehrfach rekursiv-erzeugte Pseudo-Zufallszahlen (in german), *Zeitschrift für angewandte Math. und Mechanik* **53**, T223–T225.

Knuth, D. E. (1981). *The Art of Computer Programming : Seminumerical Algorithms*, vol. 2, second edition. Addison-Wesley.

L'Ecuyer, P. (1986). Efficient and Portable 32-Bit Random Variate Generators. *Proceedings of the 1986 Winter Simulation Conference*, 275–277.

L'Ecuyer, P. (1987). A Portable Random Number Generator for 16-Bit Computers. *Modeling and Simulation on Microcomputers: 1987*, The Society for Computer Simulation, 45–49.

L'Ecuyer, P. (1988). Efficient and Portable Combined Random Number Generators. *Communications of the ACM*, **31**, 6, 742–749 and 774.

L'Ecuyer, P. and Côté, S. (1987). A Random Number Package with Splitting Facilities. Report no. DIUL-RR-8705, dépt. d'informatique, Univ. Laval (submitted to *ACM Trans. on Math. Software*).

L'Ecuyer, P., Perron, G. and Blouin, F. (1988). SENTIERS: Un logiciel Modula-2 pour l'arithmétique sur les grands entiers. Report no. DIUL-RT-8802, dépt. d'informatique, Univ. Laval.

Marsaglia, G. (1968). Random Numbers Fall Mainly in the Planes. *Proceedings of the National Academy of Sciences of the United States of America* 60, 25–28.

Monahan, J. F. (1985). Accuracy in Random Number Generation. *Mathematics of Computation*, 45, 172, 559–568.

Montgomery, P. L. (1987). Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48, 177, 243–264.

Rabin, M. O. (1980). Probabilistic Algorithms for Primality Testing. *J. Number Theory*, **12**, 128–138.

Silverman, R. D. (1987). The Multiple Polynomial Quadratic Sieve. *Mathematics of Computation*, 48, 177, 329–339.

AUTHOR'S BIOGRAPHIES

PIERRE L'ECUYER is an associate professor in the Computer Science Department at Laval University, Ste-Foy, Québec, Canada. He received the B.Sc. degree in mathematics in 1972, and was a college teacher in mathematics from 1973 to 1978. He then received the M.Sc. degree in operations research and the Ph.D. degree in computer science, in 1980 and 1983 respectively, both from the University of Montreal. From 1980 to 1983, he was also a research assistant at l'Ecole des Hautes Etudes Commerciales, in Montreal. His research interests are in Markov renewal decision processes, approximation methods in dynamic programming, optimization in stochastic processes, random number generation, and discrete-event simulation software. He is a member of ACM, IEEE, ORSA and SCS.

Pierre L'Ecuyer
Département d'informatique
Pavillon Pouliot
Université Laval
Ste-Foy, Qué., Canada
G1K 7P4
(418) 656-3226

FRANÇOIS BLOUIN is a master's student in computer science at Laval University. He received a B.Sc. in computer science from Laval University in 1987, and worked there as a research assistant. He is a member of ACM.

François Blouin
862 Moreau
Ste-Foy Qué., Canada
G1V 3B4
(418) 659-4045