Random Number Generation for the New Century

Lih-Yuan DENG and Dennis K. J. LIN

Use of empirical studies based on computer-generated random numbers has become a common practice in the development of statistical methods, particularly when the analytical study of a statistical procedure becomes intractable. The quality of any simulation study depends heavily on the quality of the random number generators. Classical uniform random number generators have some major defects—such as the (relatively) short period length and the lack of higher-dimension uniformity. Two recent uniform pseudo-random number generators (MRG and MCG) are reviewed. They are compared with the classical generator LCG. It is shown that MRG/MCG are much better random number generators than the popular LCG. Special forms of MRG/MCG are introduced and recommended as the random number generators for the new century. A step-by-step procedure for constructing such random number generators is also provided.

KEY WORDS: Linear congruential generator (LCG); Matrix congruential generator (MCG); Multiple recursive generator (MRG); Portable and efficient generator.

1. INTRODUCTION

A tremendous amount of computer software has been developed to support statistical computing requirements in the past decade. Simulation study has become a common practice in the development of statistical methods. This is particularly true when the analytical study is intractable. Generation of random variates with a specified distribution is a key issue of almost any simulation study. Given the distribution, often there are several generating methods that can be used to produce a random number sequence. These methods are mainly based on the generation of independent variates from the uniform distribution, $U(0,1)$. Thus, the uniform random number generation is the foundation for all computer simulation studies. Because of this, we will focus on $U(0,1)$ random number generators in this article.

Note that random numbers generated by any specific algorithm are "systematic" and therefore are neither truly independent, nor purely random. A set of criteria is thus desirable for selecting/comparing random number generators. It is common to consider five major issues for the comparison: (1) period length; (2) computing efficiency; (3) portability; (4) theoretical justification on randomness; and (5) empirical performance.

Many random number generators have been proposed in the literature. The most popular generator, the *linear congruential generator* (LCG), was proposed by Lehmer (1951) and is widely used today. It is generated from a simple linear recursive relation with a modulus p, which is commonly chosen as a large prime number or of the form 2^w to fit in the computer word size (say, 32-bits). Once pseudo random integers between 0 and p are generated, they are transformed into the interval $[0,1]$ by a scale of $1/p$. One of the major limitations of the LCG is that its period is limited by the modulus p. Most of the software uses $p = 2^{31} - 1$. This may have been sufficiently large for most simulations done in the past; however, with better and better computer facilities now available, the scale of simulation study is getting larger and larger. For a large scale simulation study, such a life period may not be enough. This is particularly true for parallel simulation systems.

Two recent uniform number generators are discussed in Section 2: *multiple recursive generator* (MRG) and *matrix congruential generator* (MCG). MRG is generated from a linear combination of the past k random numbers. MCG can be considered as a k-dimensional extension of the classical LCG. MCG and MRG are closely related (see, e.g., Deng, Rousseau, and Yuan 1992). The maximum period of MRG/MCG is $p^k - 1$ (as compared to p for LCG). The major drawback of using MRG/MCG in their general forms is that computing time is about k times more than that of the LCG. Such a drawback, however, can be managed. Section 3 proposes special forms of MRG/MCG with periods of $p^k - 1$ and computing times the same as that of LCG. In addition, sample listings of MRG/MCG's for $k = 2, 3, 4$ are given. A step-by-step procedure for constructing such a random number generator is also provided. Section 4 compares the performance of LCG with MRG/MCG. It is shown that, in terms of several popular criteria, MRG/MCG perform much better than LCG. It is our hope that the software authors, especially for statistical applications, can update their random number generator to MRG/MCG to prepare the simulation needs for the new century.

2. CLASSICAL AND RECENT RANDOM NUMBER GENERATORS

This section reviews briefly three random number generators: *linear congruential generator* (LCG), *multiple recursive generator* (MRG), and *matix congruential generator* (MCG). For a recent review on random number generation, see Deng (1998).

2.1 Linear Congruential Generator (LCG)

The congruential method, proposed by Lehmer (1951), is the most commonly used pseudo-random number generator.

Lih-Yuan Deng is Professor, Department of Mathematical Sciences, The University of Memphis, Memphis, TN 38152 (E-mail: DENGL@msci.memphis.edu). Dennis K. J. Lin is Professor, Department of Management Science and Information Systems, Pennsylvania State University, University Park, PA 16802 (E-mail: DKL5@psu.edu).

A sequence of random numbers is obtained by setting

$$X_i = (BX_{i-1} + A) \bmod m, \quad i \geq 1, \tag{1}$$

where X_i, B, A, and m are non-negative integers. The quality of the generator is determined by the choice of the increment A, multiplier B, initial seed X_0, and modulus m. If A is not zero, it is possible to achieve the full period m (Knuth 1981, p. 16). If $A = 0$, it is called a *multiplicative linear congruential generator* (MLCG), in which case it becomes

$$X_i = BX_{i-1} \bmod m, \quad i \geq 1. \tag{2}$$

The maximum period of the sequence $\{X_0, X_1, X_2, \ldots\}$ generated depends on the choice of the modulus m (Knuth 1981, p. 20).

Marsaglia (1968) was the first to show that successive overlapping sequences of k random numbers fall on at most $(k!m)^{1/k}$ planes, where m is the modulus chosen. This shortcoming may yield grossly wrong results for certain applications, such as the Monte Carlo multiple-integration method. It is also known that LCG cannot generate all lattice points in two- or higher dimensional space, even though it is capable to generate uniformity in one-dimensional space.

2.2 Multiple Recursive Generator (MRG)

MRG is a natural extension of LCG. Instead of computing the next random number from the one just computed, MRG computes a linear combination of the past k random numbers generated. The maximum period of MRG can be generated from a degree k primitive polynomial (Knuth 1981, p. 28–29)

$$f(x) = x^k - \alpha_1 x^{k-1} - \cdots - \alpha_k, \tag{3}$$

with period $p^k - 1$ by

$$X_i = (\alpha_1 X_{i-1} + \cdots + \alpha_k X_{i-k}) \bmod p, \quad i \geq k \tag{4}$$

for any initial nonzero vector $(X_0, \ldots, X_{k-1})$, where p is a large prime number. A polynomial of degree k is said to be a "primitive polynomial modulo p" if this polynomial has a root that is a primitive element of the field with p^k elements. Please see Knuth (1981), Zierler (1959), Golomb (1967), and Lidl and Niederreiter (1986) for more about the primitive polynomial and the MRG. Clearly, when $k = 1$, MRG is reduced to LCG.

2.3 Matrix Congruential Generator (MCG).

The matrix generator, considered by Franklin (1964), Grothe (1987), and Niederreiter (1986), is defined by

$$\mathbf{X}_i = \mathbf{B}\mathbf{X}_{i-1} \bmod p, \quad i \geq 1, \tag{5}$$

where $\mathbf{X}_i$'s are k-dimensional vectors, $\mathbf{X}_0 \neq \mathbf{0}$ is an initial nonzero vector, $\mathbf{B}$ is a $k \times k$ matrix, and p is usually chosen as a large prime number. The maximum period of the MCG is $p^k - 1$. A brief review of the matrix generator was given by L'Ecuyer (1990). The procedure proposed by Grothe (1987) of finding the matrix multiplier B with the maximum period depends on the availability of the primitive polynomial of degree k. Deng, Rousseau, and Yuan (1992) described a more efficient and flexible procedure to find MRG/MCG with the maximum period.

3. EFFICIENT AND PORTABLE MRG/MCG

3.1 Efficient MRG

As we can see in Section 2, the MRG has a much longer period than the period of the LCG. One of the drawbacks is its computing time, which is about k times longer than the computing time of LCG. To improve the efficiency of the MRG, we can make use of Watson's (1962) idea to set as many terms of α_i in MRG to be 0 and/or ± 1 as possible. In particular, we can find some MRG's with $\alpha_1 = \pm 1$ and $\alpha_i = 0$ for $2 \leq i \leq k-1$, $\alpha_k = B$, where B is an integer properly chosen so that the MRG has the maximum period $p^k - 1$. Here, we propose a special form of the MRG, *fast MRG* (FMRG), as

$$X_i = (BX_{i-k} - X_{i-1}) \bmod p, \quad i \geq k \tag{6}$$

Clearly, the FMRG defined in (6) is very similar to the LCG defined in (1). The difference between the LCG in (1) and the FMRG in (6) is that we add a constant increment A in LCG whereas we add/subtract a variable increment X_{i-1} in the FMRG. Obviously, we do not expect any difference between the LCG and the FMRG in terms of their computing time. L'Ecuyer (1990) suggested consideration of two nonzero terms, α_j and α_k $(1 \leq j < k)$, of the MRG in (4). Therefore, their computing time is about double the computing time of the FMRG proposed here. It is well known that the subtraction of two positive integers represented in a computer word will not cause an overflow. If the result is negative, then add p to the final result. On the other hand, the addition of two positive integers may cause an overflow. Thus, using $BX_{i-k} - X_{i-1}$ in (6) is better than using $BX_{i-k} + X_{i-1}$.

3.2 Portable MRG

In many applications, we prefer a random number generator which is portable. A portable generator can be implemented in high-level programming language (e.g., FORTRAN, Pascal, C/C++) and will produce the same random sequence on any machine with a sufficient word length.

To program a portable generator, care must be taken to avoid a possible loss of lower order bits when multiplying. Most high-level languages have a data type like DOUBLE PRECISION in FORTRAN, but the number of bits stored is less than the double word. The reason is that a number of bits are used to store the exponent in the floating point representation. For example, an IBM 370 carries only 53–56 bits of accuracy, commonly called the mantissa, in double-precision. The remaining bits are used for the sign bit and its exponent. The IEEE microprocessor floating-point format standard for double-precision is 52 bits for its mantissa and 14 bits for its sign and exponent. The multiplication of two 31-bit numbers may yield a number 62-bits long, which cannot be stored exactly in double-precision. There are sev-

Table 1. Listing of B in (6), $p = 2^{31} - 1$

			(a)		
$k = 2$	26403	27149	29812	30229	31332
	33236	33986	34601	36098	36181
	36673	36848	37097	37877	39613
	40851	40961	42174	42457	43199
	43693	44314	44530	45670	46338
Period = 4, 611, 686, 014, 132, 420, 608					
			(b)		
$k = 3$	21960	23990	24683	28676	29234
	29935	30173	30994	31139	31373
	32226	33069	34577	35216	35712
	35849	36572	39211	39683	42085
	42293	43586	44656	45148	46273
Period = 9, 903, 520, 300, 447, 984, 150, 353, 281, 022					
			(c)		
$k = 4$	22093	22141	23234	23584	23761
	28097	33356	33986	34074	34217
	34736	35592	36098	36848	37886
	39188	39532	40214	41440	41863
	44530	44762	45221	46071	46135
Period = 212676478929445727369988602696879308 80					

eral solutions to this problem. The first is to implement the generator in an assembly language, which is consequentially more efficient. Of course, the disadvantage is that it will not be portable across all machines. The second choice is to restrict the multiplier B in (6) to be a certain limit (say 15 bits or $B \leq \sqrt{p}$) so that the multiplication can not exceed 52 bits. A statement like $X_i := DMOD(BX_{i-k} - X_{i-1}, p)$ will produce the exact result.

Another reason for restricting $B < \sqrt{p}$ is due to a clever method given by Payne, Rabung, and Bogyo (1969). Specifically, to compute $Y = B \cdot X \bmod p$, we first find the quotient $A = [p/B]$ and the remainder $C = p - A \cdot B$, where $[z]$ is the largest integer $\leq z$. Therefore, we have

$$p = A \cdot B + C, \quad 0 \leq C < B. \tag{7}$$

For the value of X given, we find find the quotient $Q = [X/A]$ and the remainder $R = X - A \cdot Q$. That is, we have

$$X = A \cdot Q + R, \quad 0 \leq R < A. \tag{8}$$

Using the equations above, we can see that

$$\begin{aligned} Y &= B \cdot X \bmod p \\ &= B \cdot (A \cdot Q + R) \bmod p \\ &= ((B \cdot A \cdot Q) + B \cdot R) \bmod p \\ &= (-C \cdot Q + B \cdot R) \bmod p \end{aligned} \tag{9}$$

Payne, Rabung, and Bogyo (1969) showed that the computation always stays strictly between $-p$ and p, if $B \leq \sqrt{p}$ (see also L'Ecuyer 1988). Using the algorithm described by Deng, Rousseau, and Yuan (1992), we give a sample list of $B \leq \sqrt{p}$ for $p = 2^{31} - 1$ in Table 1.

3.3 Efficient and Portable MCG

Similarly, to improve the efficiency of MCG, we can set as many terms of B_{ij} in MCG to be 0 or ± 1 as possible. In particular, we can find MCG with the maximum period

Table 2. Listing of B_i in (10), $p = 2^{31} - 1$

				(a)		
$k = 2$	B_1	41546	32840	45670	13489	34601
	B_2	39606	35496	1853	22921	32207
Period = 4, 611, 686, 014, 132, 420, 608						
				(b)		
$k = 3$	B_1	24101	28876	21199	34577	4572
	B_2	13872	44515	34942	25100	25580
	B_3	11269	794	34546	20127	32253
Period = 9, 903, 520, 300, 447, 984, 150, 353, 281, 022						
				(c)		
$k = 4$	B_1	36421	18331	2995	19875	18799
	B_2	42276	32944	72	35787	24874
	B_3	28478	24787	5121	18825	25217
	B_4	42247	45231	18677	25443	24181
Period = 212676478929445727369988602696879308 80						

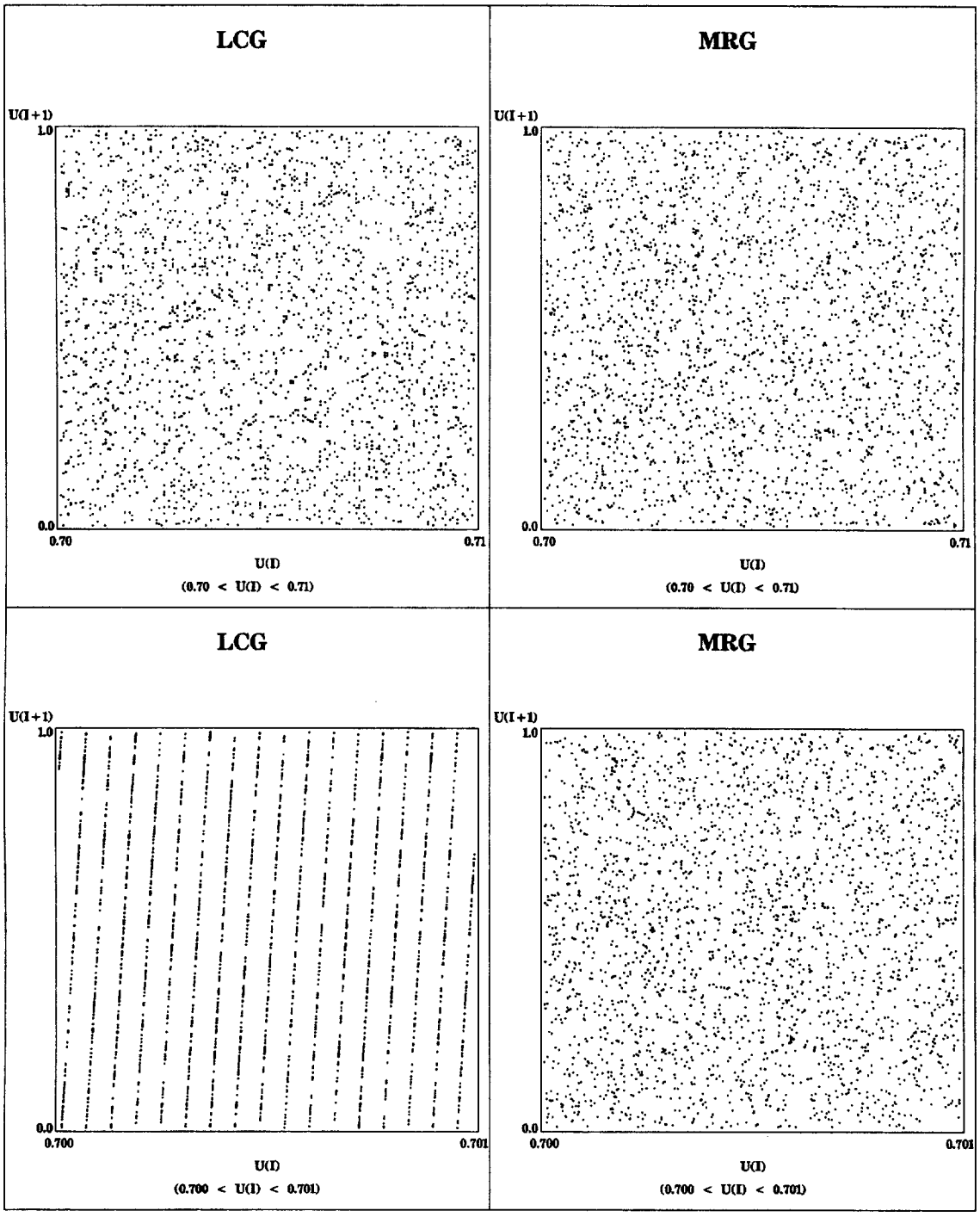

Figure 1. Comparing LCG and MRG. LCG: X(I) = 16807 X(I − 1) mod 2147483647. MRG: X(I) = 39613 X(I − 2) − X(I − 1) mod 2147483647.

$p^k - 1$ of the following form:

$$\mathbf{B} = \begin{pmatrix} B_1 & -1 & 0 & \dots & 0 \\ 0 & B_2 & -1 & \dots & 0 \\ 0 & 0 & . & \dots & 0 \\ . & . & . & \dots & . \\ . & . & . & \dots & . \\ 0 & 0 & 0 & \dots & -1 \\ -1 & 0 & 0 & \dots & B_k \end{pmatrix}, \qquad (10)$$

where $B_i (1 \leq i \leq k)$ are suitably chosen integers. In this case, the matrix in the MCG the recursive formula becomes

$$\begin{pmatrix} X_{\text{new},1} \\ X_{\text{new},2} \\ . \\ . \\ X_{\text{new},k} \end{pmatrix} = \begin{pmatrix} B_1 X_1 - X_2 \\ B_2 X_2 - X_3 \\ . \\ . \\ B_k X_k - X_1 \end{pmatrix} \quad \text{mod } p. \qquad (11)$$

Following the naming convention of FMRG, we will call the special form of MCG in (10) or (11) the *Fast MCG* (FMCG).

Clearly, each component of the FMCG defined above is very similar to the LCG defined in (1). The difference between the LCG and each component of the FMCG is that we add a constant increment A in LCG whereas we subtract a variable increment in FMCG. Obviously, we do not expect any difference between the LCG in (1) and the special form of the FMCG in (11) in their computing time. In fact, one can argue that FMCG can be even slightly faster than LCG: in order to compute k random numbers, one needs a simple loop computing FMCG whereas one needs to call the LCG routine k times. The overheads of calling the function/subroutine may cause the LCG to be slower than the special form of the FMCG in (11). Like the FMRG, the FMCG has a much longer period than that of LCG. Similar to the discussion of portable MRG, we will restrict $B_i \leq \sqrt{p}$ for the FMCG. Using the algorithm described by Deng, Rousseau, and Yuan (1992), we give a sample list of $B_i \leq \sqrt{p}, i = 1, 2, \ldots, k$ for $p = 2^{31} - 1$ in Table 2.

3.4 Construction of FMRG/FMCG

To demonstrate the use of FMRG/FMCG, we summarize the steps as follows. Since the steps described are very similar for FMRG and FMCG, only the steps for implementing portable FMRG are stated here.

1. Pick a k in FMRG. As we see, the period length increases by about p fold whenever k is increased by 1. On the other hand, a larger k for the FMRG will be somewhat slower than a smaller k. We need to keep track of the values of past k values to compute the next random number. Once this is done, we need to shift and update the k values.
2. Pick a B in FMRG. Once k is decided, we can pick a B from Table 1. For example, in the next section, we pick a FMRG with $k = 2$ and $B = 39613$ from Table 1(a). The corresponding values of A and C in (7) are $A = 54211$ and $C = 23304$.
3. Pick any initial seed. For $k = 2$, any nonzero two-dimensional vector (X_0, X_1) can be used as initial seed.
4. Portable FMRG. In the specific case mentioned, for $i \geq 2$, using formula (9) and $p = 2147483647$, we can compute the FMRG iteratively as

$$\begin{aligned} X_i &= (39613X_{i-2} - X_{i-1}) \bmod p \\ &= (39613 \cdot R - 23304 \cdot Q - X_{i-1}) \bmod p, \quad (12) \end{aligned}$$

where $Q = [X_{i-2}/54211]$, and $R = \bmod (X_{i-2}, 54211)$.

4. EMPIRICAL AND STATISTICAL JUSTIFICATIONS

This section empirically compares performances of the LCG and the FMRG/FMCG. As previously mentioned, FMRG and FMCG are closely related to each other and their performances are also similar. Therefore, we only report the empirical results for FMRG here. The LCG with $B = 16807$ in (2) has been used in almost all computer systems and packages (e.g., IMSL and SAS). The modulus is $p = 2^{31} - 1 = 2147483647$. In this empirical study, we choose the same modulus p as in LCG and $k = 2$ for FMRG. The FMRG considered here is given in (12).

4.1 Period Length Comparison

From previous sections, we know the period comparison about these two generators: For LCG: $X_i = 16807X_{i-1} \bmod 2147483647$, with a period $= 2,147,483,646$; while for FMRG: $(k = 2)$ $X_i = 39613X_{i-2} - X_{i-1} \bmod 2147483647$, with a period $= 4,611,686,014,132,420,608$. The period of the LCG is bounded by the word size of the computer, while other generators have an extremely long period which is independent of the computer word size. For example, for FMRG with $k = 4$ and $p = 2^{31} - 1$, the period is about 2.1×10^{37}, which will take today's fastest computer at least a trillion years to complete the whole cycle.

4.2 Efficiency and Portability

The FMRG in (6) and LCG in (1) have a similar form except the increment term. Therefore, they can be implemented easily and efficiently. As described in Section 4, both LCG and FMRG can be implemented in a high-level programming language (e.g., FORTRAN, Pascal, C/C++) and it will produce the same random sequence on any machine with a sufficient word length.

4.3 Graphical Comparison

Once X_i has been generated, we can scale it to $(0, 1)$ by $U_i = X_i/p$ for LCG and FMRG. Figure 1 shows the plots of (U_i, U_{i+1}) for the first 2,500 pairs generated: Figure 1(a) zooms in the range $.70 < U_i < .71$; while Figure 1(b) zooms in the range $.700 < U_i < .701$ Clearly, the plot shows that the FMRG is a much better generator than LCG. As expected, the plot of (U_i, U_{i+1}) using the LCG displays a distinct linear (nonrandom) pattern over a smaller range of U_i as in $.700 < U_i < .701$. On the other hand, the FMRG does not display any linear pattern.

4.4 Theoretical or Statistical Justification

It is well known that the LCG is one-distributed. The sequence of random numbers by a random number generator is said to have "k-distribution property" if every k-tuple of numbers appears exactly the same number of times, with the exception of the all-zero tuple which appears one time less. (See, e.g., Tootill, Robinson, and Eagle 1973.) Marsaglia (1968) was the first to show that consecutive k points generated by the LCG will lie in a relatively small number of parallel planes. The successive *overlapping* of k-tuples of sequence generated by the FMRG will generate all points in k-dimension lattice points (except 0). (See, e.g., Golomb 1967.) The FMRG can be thought of as a combination generator similar to that considered by Wichmann and Hill (1982). Deng and George (1990) and Deng, Lin, Wang, and Yuan (1997) gave some statistical justification for the asymptotic uniformity and asymptotic independence of vectors generated by the combination generators.

5. CONCLUDING REMARKS

This article proposes a portable FMRG/FMCG. We have shown that FMRG/FMCG are much better random number generators than the classical LCG. FMRG/FMCG are

as fast as the classical LCG, while their period lengths are much longer than that of LCG. The empirical performances of FMRG/FMCG are shown to be better than that of the LCG. Finally, FMRG/FMCG have a nicer theoretical support than the LCG. We truly believe that it is about time to replace the *old* generator like LCG with *new* and *improved* generators like FMRG/FMCG for the new century.

[Received June 1997. Revised November 1998.]

REFERENCES

Deng, L. Y. (1998), "Uniform Random Numbers," in *Encyclopedia of Biostatistics* (vol. 5), eds. P. Armitage and T. Colton, New York: Wiley, pp. 4651–4656.

Deng, L. Y., and George, E. O. (1990), "Generation of Uniform Variates from Several Nearly Uniformly Distributed Variables," *Communications in Statistics*, B19, No 1, 145–154.

Deng, L. Y., Lin, D. K. J., Wang, J., and Yuan, Y. (1997), "Statistical Justification of Combination Generators," *Statistica Sinica*, 7, 993–1003.

Deng, L. Y., Rousseau, C., and Yuan, Y. (1992), "Generalized Lehmer-Tausworthe Random Number Generators," in *Proceedings of the 30th Annual ACM Southeast Regional Conference*, pp. 108–115.

Franklin, J. N. (1964), "Equidistribution of Matrix-Power Residues Modulo One," *Mathematics of Computation*, 18, 560–568.

Golomb, S. W. (1967), *Shift Register Sequence*, San Francisco: Holden-Day.

Grothe, H. (1987), "Matrix Generators for Pseudo-Random Vector Generation," *Statistical Papers*, 28, 233–238.

Knuth, D. E. (1981), *The Art of Computer Programming* (vol. 2), Reading, MA: Addison-Wesley.

L'Ecuyer, P. (1988), "Efficient and Portable Combined Random Number Generators," *Communications of the Association for Computing Machinery*, 31, 742–748, 774.

——— (1990), "Random Numbers for Simulation," *Communications of the Association for Computing Machinery*, 33, 85–97.

Lehmer, D. H. (1951), "Mathematical Methods in Large-Scale Computing Units," in *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery*, Cambridge, MA: Harvard University Press, pp. 141–146.

Lidl, R., and Niederreiter, H. (1986), *Introduction to Finite Fields and Their Applications*, Cambridge, MA: Cambridge University Press.

Marsaglia, G. (1968), "Random Numbers Fall Mainly in Planes," in *Proceedings of the National Academy of Sciences*, 61, pp. 25–28.

Niederreiter, H. (1986), "A Pseudorandom Vector Generator Based on Finite Field Arithmetic," *Mathematica Japonica*, 31, 759–774.

Payne, W. H., Rabung, J. R., and Bogyo, T. (1969), "Coding the Lehmer Pseudo Number Generator," *Communications of the Association for Computing Machinery*, 12, 85–86.

Tootill, J. P. R., Robinson, W. D., and Eagle, D. J. (1973), "An Asymptotically Random Tausworthe Sequence," *Journal of the Association for Computing Machinery*, 18, 381–399.

Watson, E. J. (1962), "Primitive Polynomials (mod 2)," *Mathematics of Computation*, 16, 368–369.

Wichmann, B. A., and Hill, I. D. (1982), "An Efficient and Portable Pseudo-Random Number Generator," *Applied Statistics*, 31, 188–190.

Zierler, N. (1959), "Linear Recurring Sequences," *Journal of the Society for Industrial and Applied Mathematics*, 7, 31–48.