*Behavior Research Methods, Instruments, & Computers*
*1993, 25 (3), 384-395*

# A theoretical and empirical comparison of mainframe, microcomputer, and pocket calculator pseudorandom number generators

PATRICK ONGHENA
*Katholieke Universiteit Leuven, Leuven, Belgium*

This article presents an extensive theoretical and empirical analysis of the pseudorandom number generators provided by subroutine libraries (NAG, CERN, IMSL, and ESSL), statistical and simulation software packages (GLIM, SAS, SPSS, DATASIM, ESSP, and LLRANDOMII), built-in functions of programming languages (APL, Turbo Pascal, Advanced BASIC, GW-BASIC, and QBASIC), and autoimplemented algorithms (Fishman & Moore, 1986; Wichmann & Hill, 1982; Van Es, Gill, & Van Putten, 1983). On the basis of these analyses, it is concluded that most of the built-in functions of the software packages that were tested can be used safely. In addition, it is concluded that the Wichmann and Hill algorithm is a good choice if only single-precision arithmetic is available, but that a prime-modulus multiplicative congruential generator with modulus $2^{31}-1$ and multiplier 16,807 is a better choice if double-precision arithmetic is available, and that the same generator with multiplier 62,089,911 or 742,938,285 is the best choice if extended-precision arithmetic is available. A Turbo Pascal and a VS FORTRAN program for the latter are given in the Appendixes.

In the social and behavioral sciences, random numbers may be needed in a variety of situations. They may be needed for the design of experiments (random sampling and random assignment of subjects to the experimental conditions), for data analysis (Monte Carlo testing, bootstrap and jackknife techniques), or for more theoretical work (building and testing stochastic models of psychological processes) (Bradley, 1989a, 1989b; Diaconis & Efron, 1983; Heth, 1984; Whicker & Sigelman, 1991).

In former times, these numbers were generated by physical devices each time anew or through reference to a prefab table, but nowadays they are almost exclusively generated by computer algorithms (Dudewicz & Ralley, 1981; Hull & Dobell, 1962; Ripley, 1987). There is a growing concern, however, about the "randomness" of the numbers generated by using the built-in computer algorithms of software packages, and more and more, researchers have been encouraged to implement an alternative random-number program (Brysbaert, 1991; L'Ecuyer, 1990; Lordahl, 1988; Ripley, 1983, 1988).

This article represents an attempt to determine to what extent such a concern is justified, and whether or not researchers should indeed make the effort to implement the alternative algorithms. For this purpose, the statistical properties of the "random" numbers generated by prewritten programs in software libraries, built-in functions of commonly used software packages and compilers, and some alternative autoimplemented algorithms for mainframes, personal computers, and pocket calculators were examined. More specifically, the generators in the NAG FORTRAN Library (Numerical Algorithms Group, 1990); the CERN Program Library (CERN, 1989); the GLIM System (Payne, 1987); the Statistical Analysis System (SAS, 1989, 1991); the DATASIM package (Bradley, 1988); the Turbo Pascal (Borland, 1990), Advanced BASIC (IBM, 1986), GW-BASIC (Microsoft, 1987), and QBASIC (Microsoft, 1991) programming languages; the algorithms of Wichmann and Hill (1982) and Fishman and Moore (1986); and two pocket calculator generators (Van Es et al., 1983) were compared. Reference is also made to IMSL (1987), ESSL (1990), SPSS (1983), ESSP (Lewis, Orav, & Uribe, 1988), LLRANDOMII (Lewis & Uribe, 1988), and APL (Katzen, 1970).

A review of the theoretical properties will be given, and some empirical results will be added. On the basis of these properties and results, and taking into account the speed of the algorithms and the possible computational environments, some recommendations concerning the most appropriate generators will also be made.

## PSEUDORANDOM NUMBER GENERATORS

The generators discussed below are special cases of Lehmer's congruential iteration (Knuth, 1981)[1]

The author wishes to thank Drake Bradley, Marc Brysbaert, John Castellan, Stef Decoene, Luc Delbeke, Ignace Hanoulle, Rianne Janssen, Daniel Lordahl, Gert Storms, and one anonymous reviewer for their helpful comments on an earlier draft of the article, and Anne-Marie De Meyer, Magda Vuylsteke, Paul De Boeck and the Research Group on Quantitative Methods for their support. The author is Research Assistant of the National Fund for Scientific Research, Belgium. Correspondence concerning this article should be addressed to P. Onghena, K. U. Leuven, Department of Psychology, Center for Mathematical Psychology and Psychological Methodology, Tiensestraat 102, B-3000 Leuven, Belgium (e-mail: fpaag02@blekul11.earn).

Copyright 1993 Psychonomic Society, Inc.

$$x_{i+1} = (ax_i + c) \bmod m, \quad (1)$$

where $a$, $c$, and $m$ are nonnegative integers ($a$ and $c$ less than $m$, and $a$ different from zero), and mod is the modulus operator (which gives the remainder of an integer division). If an initial value $x_0$ is chosen from the set $\{0, 1, \ldots m-1\}$, then the formula produces a sequence $x_1$, $x_2$, ... of numbers in the same set, called *pseudorandom numbers*. These integers are then divided by $m$ to obtain a sequence of real numbers between 0 and 1.

The generators vary in their choice of $a$, $c$, $m$, and possible $x_0$s. The integer $a$ is called the *multiplier*, $c$ is called the *increment*, $m$ is called the *modulus*, and $x_0$ is called the *seed*. The terms *mixed congruential generator* and *multiplicative congruential generator* are used by many authors to denote linear congruential algorithms with $c \neq 0$ and $c = 0$, respectively. If $c = 0$ and $m$ is a prime integer, the term *prime-modulus multiplicative congruential generator* is used. It is obvious from Equation 1 that, for all multiplicative congruential generators, the seed should not be zero.

### The NAG Pseudorandom Number Generator for Mainframes

The NAG FORTRAN Library (Numerical Algorithms Group, 1990) is a collection of mathematical subroutines coded in FORTRAN that can be called from within any other program. The mainframe subroutines that generate pseudorandom numbers use a multiplicative congruential algorithm with multiplier $13^{13}$ and modulus $2^{59}$:

$$x_{i+1} = (13^{13}x_i) \bmod 2^{59};$$

that is, multiply the base ($x_i$) by $13^{13}$, divide by $2^{59}$, and take the remainder to get the next number ($x_{i+1}$).

The seed ($x_0$) is set by default to $123{,}456{,}789 = (2^{32}+1)$, but it can be changed to any other odd number.

### The CERNLIB Pseudorandom Number Generator for Mainframes

The CERN Program Library (CERN, 1989) is a large collection of general-purpose programs maintained and offered in both source and object code form on the CERN central computers. The pseudorandom number generator uses a multiplicative congruential algorithm with multiplier 44,485,709,377,909 and modulus $2^{48}$. The seed can be any odd number.

### The GLIM System Pseudorandom Number Generator for Mainframes

The Generalized Linear Interactive Modelling System (Payne, 1987) is a flexible, interactive program for statistical analysis, developed under the auspices of the Royal Statistical Society. It provides a framework for statistical analysis through the fitting of generalized linear models to the data. The pseudorandom number generator uses a mixed congruential algorithm with multiplier 8,404,997, increment 1, and modulus $2^{35}$. The seed can be any nonnegative integer less than the modulus.

### The SAS Pseudorandom Number Generator

The Statistical Analysis System (SAS, 1989, 1991) is an integrated software package for data analysis. For both the mainframe- and the PC-version, the built-in RANUNI function uses a prime-modulus multiplicative congruential generator with multiplier 397,204,094 and modulus $2^{31}-1$ (a generator proposed by Learmonth & Lewis, 1974). The seed must be a numeric constant less than $2^{31}-1$. If the seed is 0, SAS uses a reading of the time of day from the computer's clock to generate the first number. Otherwise, the specified constant is used directly.[2]

### The DATASIM Pseudorandom Number Generator

Bradley's (1988, 1989a, 1989b) data simulator DATASIM is a general-purpose program for generating, analyzing, and graphing simulated data for experimental, multivariate, and contingency table designs. It uses as a default a prime-modulus multiplicative congruential generator with multiplier 16,807 and modulus $2^{31}-1$ (a generator due to Lewis, Goodman, & Miller, 1969), but, with the RMULT and the RMOD commands, it is possible to change these two parameters. The seed is either selected at random (on the basis of digits obtained from the system clock), or explicitly set by the user (a positive integer less than $2^{31}-1$).

The same generator is used by ESSL (1990), SPSS (1983), ESSP (Lewis et al., 1988), and APL (Katzen, 1970).

### The Turbo Pascal Pseudorandom Number Generator

The Turbo Pascal (Borland, 1990) programming language provides the user with a built-in pseudorandom generator in the Random function. This function is initialized by making a call to the Randomize procedure to obtain a seed based on the system clock, or by explicitly assigning a value (in the range from $-2^{31}$ to $2^{31}-1$) to the predeclared variable RandSeed. Although the generating algorithm is undocumented in the manual, one letter to the Technical Department of Borland Inc. was enough to obtain the required information: from Version 4.0 on (up to Version 6.0, the version that was current at the time of writing the letter), Turbo Pascal uses a mixed congruential pseudorandom number generator with multiplier 134,775,813, increment 1, and modulus $2^{32}$.

### The BASIC Pseudorandom Number Generator

The Advanced BASIC (IBM, 1986), GW-BASIC (Microsoft, 1987), and QBASIC (Microsoft, 1991) programming languages provide the function RND, which is initialized by the RANDOMIZE command together with a number between $-2^{15}$ and $2^{15}-1$, and which returns a pseudorandom single-precision number between 0 and 1. As with Turbo Pascal, the generating algorithm is undocumented in the manual. Our letters to Microsoft Inc. about the algorithm remain unanswered.

### The Wichmann and Hill Algorithm

Wichmann and Hill (1982) presented FORTRAN code for three coupled prime-modulus multiplicative generators with multipliers 171, 172, 170 and moduli 30,269, 30,307, and 30,323. Because three multiplicative congruential generators are involved, three seeds are needed (larger than zero and smaller than the moduli 30,269, 30,307, and 30,323, respectively). Ready-made BASIC and Pascal versions are also available (Brysbaert, 1991; Wichmann & Hill, 1987).

It can be shown (using the Chinese remainder theorem; see, e.g., Knuth, 1981; Zeisel, 1986) that the resulting generator is equivalent to a simple multiplicative congruential generator with multiplier 16,555,425,264,690 and modulus 27,817,185,604,309. However, the coupled algorithm requires arithmetic only up to 30,323 and can therefore be implemented easily on a 16-bit microprocessor.

### The Fishman and Moore Multipliers

After an exhaustive theoretical and empirical analysis of prime-modulus multiplicative congruential generators with modulus $2^{31}-1$, Fishman and Moore (1986) found 62,089,911, 742,938,285, 950,706,376, 1,226,874,159, and 1,343,714,438 to be the optimal multipliers among the more than 534 million candidates. Between these five best multipliers there were no consistent differences. Notice that neither 16,807 (as in ESSL, DATASIM, SPSS, ESSP, and APL) nor 397,204,094 (as in SAS) is one of the five. However, the DATASIM package gives the user the opportunity to change the default multiplier with the RMULT command (but see below).

Following the Fishman and Moore (1986) study, IMSL (1987) gives the choice between 16,807, 397,204,094, and 950,706,376 as multipliers and LLRANDOMII (Lewis & Uribe, 1988) between 16,807, 397,204,094, and all five Fishman and Moore multipliers. A Turbo Pascal PC implementation for the two smallest Fishman and Moore multipliers is given in Appendix A and a VS FORTRAN mainframe implementation for all five Fishman and Moore multipliers is given in Appendix B.

### The Pocket Calculator Pseudorandom Number Generators

Van Es et al. (1983) suggested two pseudorandom number generators for pocket calculators (using decimal arithmetic) with a 10-figure display and 10-figure accuracy. Pocket I has modulus $10^5$, multiplier 31,481, and increment 21,139 and makes use of the equivalence of Equation 1 with

$$u_{i+1} = (au_i+b) \bmod 1 = \text{Frac}(au_i+b), \qquad (2)$$

where $u_i = x_i/m$, $b = c/m$, and Frac($x$) is the fractional part of a real number $x$, or Frac($x$) = $x$ − Int($x$)—that is, the original number minus the integer part.

Pocket II has modulus $10^9$, multiplier 314,159,221, and increment 211,324,863 and makes use of the equivalence of Equation 2 with

$$u_{i+1} = \text{Frac}[\text{Frac}(a^{(1)}u_i^{(2)}) + \text{Frac}(a^{(2)}u_i^{(1)}) + a^{(2)}u_i^{(2)} + b], \qquad (3)$$

where $u_i^{(1)} = \text{Int}(10^5u_i)$, $u_i^{(2)} = \text{Frac}(10^5u_i)$, $a^{(1)} = \text{Int}(10^{-5}a)$, and $a^{(2)} = \text{Frac}(10^{-5}a)$. Although this notation is cumbersome, the splitting up of the numbers $a$ and $u_i$ of Equation 2 into the five most significant and the five least significant digits is necessary to avoid rounding errors in the multiplications.

Both are mixed congruential generators, so they can be seeded with any nonnegative integer less than the modulus.

## A THEORETICAL COMPARISON

### The Period of the Generator

A simple index for the quality of a generator is its *period*. Since any pseudorandom number depends only on the previous one, once a value has been repeated, the entire sequence after it must be repeated. The period is the length of such a repeating sequence. It is obvious that generators with large periods are to be preferred.

Knuth (1981) has shown that (1) mixed congruential generators have the maximal period $m$ if and only if the increment and the modulus have no common divisor other than 1, and the multiplier $a$ is chosen such that ($a$ mod $p$) = 1 for each prime factor $p$ of the modulus and ($a$ mod 4) = 1 if 4 is a factor of the modulus; (2) multiplicative congruential generators with modulus $2^k$ ($k>2$) have maximal period $2^{k-2}$ if and only if the multiplier is a primitive root modulo $m$—that is, ($a^{(m-1)/p}$ mod $m$) $\neq$ 1 for each prime factor $p$ of $m-1$—and the seed is odd; and (3) prime-modulus multiplicative congruential generators have period $m-1$ if and only if the multiplier is a primitive root modulo $m$. This gives for some of the generators described above the periods of Table 1. For the BASIC built-in generators, no theoretical results are available because of the lack of information.

In a large-scale empirical study, Modianos, Scott, and Cornwell (1984, 1987) found the period of IBM PC BASIC and IBM PC Extended BASIC generators to be

**Table 1**
**Periods of Some Pseudorandom Number Generators and Their Relation to the Moduli**

| Generator | Period | Relation to Modulus $m$ |
|---|---|---|
| NAG | $2^{57} = 1.44\times10^{17}$ | $m/4$ |
| CERN | $2^{46} = 7.04\times10^{13}$ | $m/4$ |
| GLIM | $2^{35} = 3.44\times10^{10}$ | $m$ |
| SAS | $2^{31}-2 = 2.15\times10^{9}$ | $m-1$ |
| DATASIM | $2^{31}-2 = 2.15\times10^{9}$ | $m-1$ |
| Turbo Pascal | $2^{32} = 4.29\times10^{9}$ | $m$ |
| Wichmann and Hill | $6.95\times10^{12}$ | $(m_1-1)(m_2-1)(m_3-1)/4$ |
| Fishman and Moore | $2^{31}-2 = 2.15\times10^{9}$ | $m-1$ |
| Pocket I | $10^5$ | $m$ |
| Pocket II | $10^9$ | $m$ |

Note—Software versions are: NAG Mark 14, CERN 1989 release, GLIM Release 3.77, SAS Version 6.04, DATASIM Version 1.1, and Turbo Pascal Version 6.0.

$2^{16}$, which is much too short for most applications. Whitney (1984) came to the same result with IBM PC Advanced BASIC, and he also observed systematic, short, wave-like subcycles.

In an attempt to replicate these findings, Version A3.21 of Advanced BASIC, Version 3.2 of GW-BASIC, and Version 1.0 of QBASIC were tested. The results of Modianos et al. (1984, 1987) could not be confirmed, and a replication of Whitney's (1984) random-walk analysis did not reveal any systematic subcycles for any of the BASIC versions. More than a week of PC time showed that the periods of the BASIC generators had to be more than $2^{32}$.

It should also be noted that Wichmann and Hill's algorithm does not have the maximal period $2.78 \times 10^{13}$. Dependencies between the three generators reduce the period of the combined generator to $(m_1-1)(m_2-1)(m_3-1)/4$, or about $6.95 \times 10^{12}$ (Wichmann & Hill, 1984, 1987).

### The Dimensional Structure of the Generator

With the use of theoretical tests, it is possible to assess the global randomness of a generator over the full period. The most famous test in this context is the spectral test proposed by Conveyou and MacPherson (1967) and developed by Knuth (1969). The spectral test is based on the fact that the pseudorandom numbers $(x_i, x_{i+1}, \ldots, x_{i+t-1})$, $i = 0, \ldots m-1$, lie in the hypercube $[0,m]^t$ on various sets of parallel equidistant hyperplanes (see also Marsaglia, 1968). The numbers have the highest global randomness if the distance between consecutive hyperplanes, for the set of hyperplanes that makes this distance maximal, is as small as possible. The spectral test consists in computing these maximal distances $v_t^{-1}$ for a number of values of $t$; the transformation $\mu_t = \pi^{t/2} v_t^t / [(t/2)!m]$ yields an index of quality (the *merit*), which is roughly comparable over different values of $t$ and $m$. A generator passes the test if $\mu_t \geq 0.1$ for $2 \leq t \leq 6$, and it passes "with flying colors" if $\mu_t \geq 1$ for all these $t$ (Knuth, 1981).

Algorithms for performing the spectral test are given by Golder (1976a, 1976b), Hoaglin and King (1978), and Ripley (1987). Table 2 gives the results for the generators described above (except for the BASIC generators).

**Table 2**
**Spectral Test Merits $\mu_t$ for $2 \leq t \leq 6$**
**for Some Pseudorandom Number Generators**

| | | | | | |
|---|---|---|---|---|---|
| NAG | 2.56 | 0.72 | 1.96 | 0.96 | 1.56 |
| CERN | 0.62 | 0.61 | 0.06 | 2.11 | 1.00 |
| GLIM | 1.12 | 1.67 | 0.07 | 3.13 | 1.26 |
| SAS | 1.12 | 1.13 | 1.96 | 3.97 | 1.06 |
| DATASIM | 0.41 | 0.51 | 1.08 | 3.22 | 1.73 |
| Turbo Pascal | 0.70 | 1.32 | 0.90 | 2.83 | 3.06 |
| Wichmann and Hill* | 2.01 | 1.74 | 2.06 | 4.91 | 2.90 |
| 62,089,911† | 2.14 | 4.34 | 4.23 | 4.77 | 7.99 |
| 742,938,285† | 2.73 | 3.78 | 5.47 | 5.94 | 8.04 |
| 950,706,376† | 2.67 | 4.30 | 5.63 | 6.00 | 7.66 |
| 1,226,874,159† | 2.57 | 4.02 | 4.58 | 6.15 | 8.63 |
| 1,343,714,438† | 2.46 | 3.42 | 4.56 | 5.73 | 7.55 |
| Pocket I | 0.11 | 1.52 | 0.91 | 1.24 | 0.21 |
| Pocket II | 0.81 | 2.15 | 0.56 | 2.21 | 3.43 |

Note—Tested versions are: NAG Mark 14, CERN 1989 release, GLIM Release 3.77, SAS Version 6.04, DATASIM Version 1.1, and Turbo Pascal Version 6.0. *Modified spectral test (MacLaren, 1989). †For the Fishman and Moore algorithm, the five best multipliers are given.

Only the CERN and GLIM generators do not pass the test. Both have merits below 0.1 in the fourth dimension. The SAS, Wichmann and Hill, and Fishman and Moore generators pass "with flying colors." In all dimensions, the Fishman and Moore algorithms pass the spectral test with superior merits in comparison with the SAS function and the Wichmann and Hill algorithm. Moreover, for the Wichmann and Hill algorithm, it remains unclear whether the merits of the modified spectral test are completely comparable with the merits of the conventional spectral test (MacLaren, 1989). As Ripley (1988) argues:

> The "better the unknown than the devil we know" attitude still surfaces. For example, Wichmann and Hill advocate combining three simple congruential generators. We know very little about such combination generators. The authors (and their referees) even stated the wrong period. This may well be an excellent generator, but to my knowledge none can prove so. The history of the subject has shown that empirical tests are not sufficiently comprehensive; theoretical calculations are required. (p. 55)

This argument holds a forteriori for the undocumented BASIC generators.

## AN EMPIRICAL COMPARISON

### The Precision of the Generator

Before the results of the empirical tests are reported, it should be mentioned that in implementing a generator one should take account of the word size (and consequently the precision of the representable values in a floating point system) that the computer can handle. Otherwise, it is very possible to destroy the optimal properties of a theoretically sound generator. Bradley, Senko, and Stewart (1990) properly remarked that with respect to the Fishman and Moore multipliers in DATASIM, the user should verify that no loss in precision occurs. With an IBM PC/AT with an 80287 floating-point coprocessor using 64-bit double-precision arithmetic, all five Fishman and Moore multipliers are indeed too large for the multiplications to be evaluated correctly. The PC gives

$$(62,089,911)(2^{31}-2) = 133,337,068,454,095,504$$
$$(742,938,285)(2^{31}-2) = 1,595,447,817,024,787,200$$
$$(950,706,376)(2^{31}-2) = 2,041,626,394,607,926,784$$
$$(1,226,874,159)(2^{31}-2) = 2,634,692,192,152,503,808$$
$$(1,343,714,438)(2^{31}-2) = 2,885,604,780,499,080,704$$

instead of the correct (hand-calculated) values:

$$(62,089,911)(2^{31}-2) = 133,337,068,454,095,506$$
$$(742,938,285)(2^{31}-2) = 1,595,447,817,024,787,110$$
$$(950,706,376)(2^{31}-2) = 2,041,626,394,607,926,896$$
$$(1,226,874,159)(2^{31}-2) = 2,634,692,192,152,503,714$$
$$(1,343,714,438)(2^{31}-2) = 2,885,604,780,499,080,948.$$

The same problem would arise if the SAS multiplier was used in a straightforward implementation, using double-precision arithmetic. The PC gives

$$(397{,}204{,}094)(2^{31}-2) = 852{,}989{,}295{,}989{,}246{,}720$$

instead of the correct (hand-calculated) value:

$$(397{,}204{,}094)(2^{31}-2) = 852{,}989{,}295{,}989{,}246{,}724.$$

This inaccuracy is caused by the internal representation of floating-point variables and operations (Lewis & Orav, 1989; Thisted, 1987). Notice that the multipliers are smaller than $2^{31}$ and the products smaller than $2^{62}$, so that one could mistakenly expect to have a correct implementation by using 64-bit arithmetic. However, 64-bit arithmetic is not 64-bit precision. In Turbo Pascal double-precision, for example, 1 bit is used for the sign and 11 bits for the exponent, leaving only 52 bits for the significand (or mantissa). The smallest Fishman and Moore multiplier already gives a product larger than $2^{57}$. For the generator, the rounding-off errors result in a deviation from the theoretical sequence of the congruential algorithm, which could produce serious dependencies in the pseudorandom numbers (see below).

The problem can be avoided on an IBM PC/AT with an 80287 floating-point coprocessor for the smallest two Fishman and Moore multipliers by using 80-bit extended-precision arithmetic (as in Appendix A). In Turbo Pascal extended-precision, for example, 63 bits are used for the significand. The SAS and the built-in Turbo Pascal generator can also be implemented by using 80-bit extended-precision arithmetic. A simple algorithm for a generator with one of the largest three Fishman and Moore multipliers is possible with the use of the 128-bit quadruple-precision arithmetic of VS FORTRAN Version 2 on mainframe (as in Appendix B). For the DATASIM default multiplier 16,807 there is no problem, since a straightforward implementation requires only 64-bit double-precision arithmetic. It is even possible to implement this generator by using only 32-bit single-precision arithmetic (Bratley, Fox, & Schrage, 1983; Park & Miller, 1988; Schrage, 1979).

**Statistical Tests**

In order to check the local randomness of subsequences of moderate length and to assess the effect of implementation inaccuracy, a battery of statistical tests was performed on the pseudorandom number generators described above. The Fishman and Moore generators were tested twice—first, by using the RMULT command of DATASIM, and second, by using the algorithms of Appendixes A and B. In addition, the disreputable RANDU was subjected to the same battery of tests as a control. RANDU is a multiplicative congruential generator with multiplier 65,539 and modulus $2^{31}$, which has been used very widely on IBM 360/370 and PDP-11 machines but which has been shown to be fatally flawed in the third dimension (Fishman & Moore, 1982; Marsaglia, 1972).

The battery of statistical tests consisted of (1) the Kolmogorov-Smirnov distribution test; (2) the chi-square goodness-of-fit test for uniformity (9 *df*); (3) the gaps test ($r_l = 0.4$; $r_u = 0.6$; 9 *df*); (4) the runs-above-the-mean test (9 *df*); (5) the runs-below-the-mean test (9 *df*); (6) the runs-up test (6 *df*); (7) the runs-down test (6 *df*); (8) the pairs test (99 *df*); (9) the triplets test (124 *df*); and (10) the autocorrelation test (10 *df*). Appendix C gives references and descriptions of these tests. Following Fishman and Moore (1982) and L'Ecuyer (1988), 100 consecutive sequences of 200,000 pseudorandom numbers for each of the generators (10,000 for the BASIC generators) were produced for each test. Consequently, for each test, 100 test statistics and 100 corresponding $p$ values were obtained. Next, the Kolmogorov-Smirnov test was used again, but this time on a metalevel to check whether the 100 statistics and $p$ values of the first-level tests were plausible under the expected theoretical distribution (see Appendix C). Furthermore, this metalevel Kolmogorov-Smirnov test was used on the 1,000 $p$ values of all first-level tests for each generator to have an overall measure of goodness-of-fit to the uniform distribution. The tests were programmed in VS FORTRAN Version 2 (IBM, 1987), using the NAG Library subroutines G08CBF (Test 1), G08CGF (Test 2), G08EDF (Tests 3, 4, and 5), G08EAF (Tests 6 and 7), G08EBF (Test 8), G08ECF (Test 9), and G13ABF (Test 10), and were carried out on an IBM 3090/600e VF mainframe under the VM/XA operating system (the programs are available from the author). The testing took more than 300 h of CPU time.

In Table 3, the $p$ values for the metalevel Kolmogorov-Smirnov tests are presented for each statistical test and each generator. Although the DATASIM default generator performed very well on all tests, DATASIM, using the RMULT command to get the Fishman and Moore multipliers, gave $p$ values smaller than .0001 on all the tests, indicating strong deviations from randomness as a result of the rounding-off errors. Also, for the first generator of Van Es et al. (1983) and for RANDU, the $p$ value of the overall metalevel Kolmogorov-Smirnov test is smaller than .0001, which gives evidence for the inappropriateness of these generators.

In order to get a more differentiated picture, the $p$ values for each test smaller than .05 were marked "suspect," and the statistical tests for the corresponding generator were repeated. The triplets test on GLIM, the Kolmogorov-Smirnov test on the Fishman and Moore algorithm with multiplier 62,089,911, the runs-above-the-mean and the triplets tests on the Fishman and Moore algorithm with multiplier 950,706,376, and the autocorrelation test on the Fishman and Moore algorithm with multiplier 1,226,874,159 gave $p$ values smaller than .05 on the first trial, but $p$ values larger than .05 on the second trial. Of course, among the many $p$ values of the first trial, one would expect some $p$ values smaller than .05, even if the null hypothesis of randomness were true. The $p$ value of a metalevel Kolmogorov-Smirnov test on the 15,000 $p$ values

**Table 3**

**First- and Second-trial $p$ values for the Metalevel Kolmogorov-Smirnov Test on 100 Statistics From the Kolmogorov-Smirnov Distribution Test (K-S), the Chi-Square Goodness-of-Fit Test (CHI), the Gaps ($r_l = 0.4$; $r_u = 0.6$) Test (GAP), the Runs-Above-the-Mean Test (RUNSABOVE), the Runs-Below-the-Mean Test (RUNSBELOW), the Runs-Up Test (RUNSUP), the Runs-Down Test (RUNSDOWN), the Pairs Test (PAIRS), the Triplets Test (TRIPLET), and the Autocorrelation Test (AUTOCORR), Calculated from 100 Consecutive Sequences of 200,000 Pseudorandom Numbers for Each Generator Described in the Text**

| | NAG | CERN | GLIM | SAS | DATASIM† | Turbo Pascal | GW-BASIC | QBASIC | Wichmann & Hill | Appendix 62,... | Appendix 742,... | Appendix 950,... | Appendix 1,2... | Appendix 1,3... | Pocket II‡ | RANDU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K-S | .28330 | .28316 | .33010 | .50636 | .54588 | .13520 | .20309 | .72066 | .14597 | .03257* (.35497) | .45010 | .81352 | .72040 | .28330 | .82740 | .51620 |
| CHI | .82816 | >.9999 | .18028 | >.9999 | .94380 | >.9999 | .24543 | .34607 | .85684 | .58870 | .40748 | .92727 | .90416 | .60981 | .73870 | >.9999 |
| GAPS | .78543 | .94720 | .51000 | .19437 | >.9999 | .88510 | .76480 | .57655 | .45872 | .42476 | .50410 | .77543 | .47004 | .37373 | >.9999 | .02061* (.10854) |
| RUNSABOVE | >.9999 | .87095 | .93912 | .26217 | .66150 | .88286 | .37960 | .64957 | .40238 | .59660 | .81169 | .04110* (.90021) | .24953 | >.9999 | .46878 | >.9999 |
| RUNSBELOW | .88785 | .75171 | .59814 | .83126 | .07170 | .61019 | .65453 | .28312 | .45878 | .97810 | .54072 | .81140 | .50054 | .75233 | >.9999 | .52786 |
| RUNSUP | >.9999 | .71244 | .41006 | .12051 | .52970 | .13250 | .56702 | .72330 | .32257 | .60098 | >.9999 | .80254 | .96699 | .68946 | .58225 | .00071* (<.0001*) |
| RUNSDOWN | .60068 | .91708 | .85378 | .92103 | .94572 | .28781 | .59687 | .62101 | .75320 | .07202 | >.9999 | .19205 | .27250 | >.9999 | .33412 | .00060* (<.0001*) |
| PAIRS | .38135 | .25266 | .11157 | .76829 | .61209 | .84027 | .49843 | .46108 | .99577 | >.9999 | .99720 | .28880 | .52686 | .41444 | .52672 | .01259* (.20777) |
| TRIPLETS | .95751 | .81780 | .01324* (.10640) | .57264 | >.9999 | >.9999 | .46046 | .53917 | .88100 | .66004 | .64541 | .00197* (>.9999) | .99852 | .92792 | .12284 | <.0001* (<.0001*) |
| AUTOCORR | .41883 | >.9999 | .78253 | .44863 | .25053 | .87560 | .95518 | .43316 | .11992 | .43074 | >.9999 | .77543 | .00424* (.22080) | .52099 | .44395 | .08463 |
| OVERALL | .50028 | .24770 | .85345 | .91379 | .56613 | .68238 | .74174 | .33225 | .08651 | .40973 | .29411 | .77814 | .77563 | .18971 | .47822 | <.0001 |

* $p < .05$, second-trial $p$ values are given in brackets below the first-trial $p$ values if the latter are smaller than .05. †In the column DATASIM, the results for the default generator with multiplier 16,807 are given. Using the DATASIM RMULT command to get the Fishman and Moore multipliers resulted in $p < .0001$ on both trials. ‡The first generator of Van Es et al. (1983) had $p < .0001$ on both trials. Note—Tested versions are: NAG Mark 14, CERN 1989 release, GLIM Release 3.77, SAS Version 6.04, DATASIM Version 1.1, Turbo Pascal Version 6.0, GW-BASIC Version 3.2, and QBASIC 1.0. The row labeled OVERALL gives the $p$ value for the metalevel Kolmogorov-Smirnov Test Based on the 1,000 statistics from the ten tests.

of the 15 appropriate generators gave a $p$ value of .433, which shows a good fit of the $p$ values to the uniform distribution. Furthermore, because the second-trial $p$ values were larger than .05, it is probable that the suspect $p$ values on the first trial were the result of mere chance fluctuations. The $p$ values smaller than .0001 for the second-trial runs-up, runs-down, and triplets tests on RANDU confirm the inappropriateness of this generator.

### The Speed of the Generator

The time needed to produce one pseudorandom number with the software packages, the built-in functions of the compilers, and the algorithms with or without calling the subroutine libraries described above were monitored by generating 200,000 numbers on an IBM 3090/600e VF mainframe under the VM/XA operating system or on an IBM PC/80486 running at 50 MHz under DOS 5.0, or 10,000 numbers on an IBM PC/AT 80286 with an 80287 floating-point coprocessor running at 8 MHz under DOS 3.2. Clock speeds were verified with the Landmark System Speed Test (Landmark Research, 1990). Timing the pocket calculator generators was not considered relevant, because the time needed to produce pseudorandom numbers is mainly a function of the ability to use the pocket calculator interactively.

Table 4 shows that it takes less than .01 msec to generate a pseudorandom number on a mainframe, except when one uses the Fishman and Moore algorithms. The quadruple-precision arithmetic in the latter causes the generator to be more than 100 times slower than the Lewis et al. (1969) generator involving only double-precision arithmetic.

The SAS generator is about 12 times slower on an 80486 than on a mainframe, but the programs coded in Turbo Pascal with double-precision arithmetic on an 80486 are only about 3 times slower than the programs coded in VS FORTRAN on a mainframe. By making use of the extended-precision arithmetic of Turbo Pascal, one can even have the algorithm of Fishman and Moore with one of the two smallest multipliers run about 32 times faster than on the mainframe. Note also the speed of the built-in Turbo Pascal Random function. This is sheer "mainframe" performance. The built-in QBASIC RND function is disappointing in this respect.

For Turbo Pascal, the algorithms run about 40 to 50 times slower on an 80286 than on an 80486. For the statistical packages, it is about 20 times slower, and for BASIC, about 7.5 times slower. It seems that QBASIC does not make optimal use of the 80486 speed.

It should be mentioned that the relative slowness of the statistical software packages in generating pseudorandom numbers is the price to pay for the additional features they offer for using the pseudorandom numbers. For example, in the case of DATASIM, the uniform random numbers are stored in an array for subsequent use, which requires the evaluation of two subscripts on each invocation, and the ability to alter the values of the multiplier or modulus means that these are represented internally as variables rather than constants. Consequently, the speed performance of DATASIM is not the speed performance of the congruential algorithm with multiplier 16,807 and modulus $2^{31}-1$ as such. The latter is given in the row labeled Lewis in Table 4. The same applies to SAS. An indicator of the speed performance of its algorithm with multiplier 397,204,094 and modulus $2^{31}-1$ as such is given in the row labeled Fishman and Moore 1 in Table 4.

**Table 4**
**Number of Milliseconds Needed to Produce One Pseudorandom Number With the Software Packages, the Built-In Functions of the Compilers, and the Algorithms With or Without Calling the Subroutine Libraries on an IBM 3090/600e VF Mainframe Under the VM/XA Operating System, on an IBM PC/80486 Running at 50 MHz Under DOS 5.0, or on an IBM PC/AT 80286 With an 80287 Floating-Point Coprocessor Running at 8 Mhz Under DOS 3.2**

| Generator | Mainframe | 80486 | 80286 |
|---|---|---|---|
| NAG | 0.0023 | | |
| CERN | 0.0035 | | |
| GLIM | 0.0088 | | |
| SAS | 0.0093 | 0.1150 | 1.9000 |
| DATASIM | | 0.0500 | 1.0900 |
| Turbo Pascal | | 0.0030[a] | 0.1210[a] |
| BASIC | | 0.0870[b] | 0.6530[c] |
| Wichmann and Hill | 0.0045[d] | 0.0140[a] | 0.5210[a] |
| Fishman and Moore 1[e] | 0.2400[f] | 0.0074[g] | 0.3790[g] |
| Fishman and Moore 2[h] | 0.2400[f] | | |
| Lewis[i] | 0.0022[d] | 0.0066[a] | 0.3290[a] |

[a]Using Turbo Pascal Version 6.0, double-precision. [b]QBASIC Version 1.0. [c]GW-BASIC Version 3.2. [d]Using VS FORTRAN Version 2, double-precision. [e]The two smallest Fishman and Moore multipliers (Appendix A). [f]Using VS FORTRAN Version 2, quadruple-precision. [g]Using Turbo Pascal Version 6.0, extended-precision. [h]The three largest Fishman and Moore multipliers (Appendix B). [i]The algorithm of Lewis, Goodman, and Miller (1969) used by, for example, DATASIM Version 1.1.

## DISCUSSION

On the basis of the theoretical and empirical results, one can conclude that the available pseudorandom number generators in statistical packages such as SAS, SPSS, ESSP, and DATASIM; in compilers such as Turbo Pascal and APL; and in mathematical software libraries such as NAG, ESSL, and IMSL are good enough to justify their use. This should be reassuring, because recently some doubt has been raised about the reliability of these generators and because programming and debugging alternative algorithms could be a very time-consuming activity, especially if one wanted to generate random variates according to a specific distribution (see, e.g., Brysbaert, 1991; L'Ecuyer, 1990; Lordahl, 1988; Ripley, 1983, 1988).

However, the empirical results indicate that it is inappropriate to change the DATASIM default multiplier to one of the Fishman and Moore multipliers, because the inaccuracy in the multiplications causes serious depen-

dencies in the pseudorandom numbers. It is also not possible to recommend routine scientific use of the BASIC built-in RND function, because the generator is undocumented. Finally, the CERN and GLIM generators do not seem preferable, because they fail the spectral test in the fourth dimension.

It should also be remarked that the pseudorandom number generators in statistical packages are usually relatively slow. Consequently, if speed is important and one merely needs pseudorandom numbers, one should implement the algorithms of Appendix A or B, or use the built-in function of a compiler such as Turbo Pascal.

If one needs pseudorandom numbers, the use of existing commercial software is not required at all. In fact, the easy-to-implement algorithms (see the Appendixes) are superior to most of the available built-in functions. The appropriateness of the algorithms depends on the precision that one's computer or compiler can handle. If only single-precision arithmetic is available, the Wichmann and Hill generator seems to be a good choice (Brysbaert, 1991; Wichmann & Hill, 1987). However, if double-precision arithmetic is available, a prime-modulus multiplicative congruential generator with modulus $2^{31}-1$ and multiplier 16,807 is more appropriate for large applications, because of its simplicity and the resulting speed advantage. If extended-precision arithmetic is available, the same generator with multiplier 62,089,911 or 742,938,285 (Appendix A) has an additional theoretical lead. On the basis of the statistical tests concerning uniformity and independence, the use of the first generator of Van Es et al. (1983) is not recommended. If only a pocket calculator is available, the second generator of Van Es et al. would seem to be a good choice.

Finally, it should be remarked that the quest for the optimal pseudorandom number generator is not over. As computer power gets progressively cheaper, more robust generators with longer periods will be needed, and as hardware configurations become increasingly sophisticated, the tools will be available to provide them.

## A Word of Caution

The results of the present study can be misrepresented in three important ways, all of which have to do with generalizability. Therefore, it seems appropriate to state explicitly what the study does *not* show.

1. The good results with the tested implementations do not generalize to all other implementations. For example, Afflerbach (1985) and Aldridge (1987) have reported specific problems for the Apple II series of computers, not dealt with in this review. McLeod (1985) even found problems in implementing the Wichmann and Hill algorithm on Prime-400 computers, because on these machines only 23 bits are used for the representation of the fractional part of a real variable.

2. The good results with the tested congruential generators do not generalize to all other generators. For example, Ferrenberg, Landau, and Wong (1992) have shown how fast implementations of four noncongruential generators severely biased their Monte Carlo simulations of the behavior of atoms in a magnetic crystal. However, the only congruential algorithm that they tested (the generator of Lewis et al., 1969, the default in DATASIM) gave good results, even when alternative algorithms were used to increase the speed.

3. The good results with the algorithms to generate uniform pseudorandom numbers do not generalize immediately to any complete application. It can be that the interaction with or the position in the complete algorithm is problematic. For example, Brysbaert (1991) and Castellan (1992) have shown that the random permutation algorithm proposed by Nilsson (1978) does not generate permutations that are equally likely and that the deviations are extreme and systematic, even if one has a good implementation of a good pseudorandom number generator. As another example, Brysbaert and Cavegn (1993) have shown how multiple reseeding of the pseudorandom number generator in the same program may lead to severe departures from randomness.

Consequently, the results of this study are by no means a safeguard for the correct application of algorithms for randomness. This study merely compares the tools for starting to build these applications and presents the best ones. For each application, it is the responsibility of the researcher to scrutinize the final product.

## REFERENCES


AFFLERBACH, L. (1985). The pseudo-random number generators in Commodore and Apple microcomputers. *Statistische Hefte*, **26**, 321-333.

ALDRIDGE, J. W. (1987). Cautions regarding random number generation on the Apple II. *Behavior Research Methods, Instruments, & Computers*, **19**, 397-399.

BORLAND INTERNATIONAL (1990). *Turbo Pascal, Version 6.0* [Computer program]. Scotts Valley, CA: Author.

BOX, G. E. P., & JENKINS, G. M. (1976). *Time series analysis* (rev. ed.). San Francisco: Holden-Day.

BOX, G. E. P., & PIERCE, D. A. (1970). Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association*, **64**, 1509.

BRADLEY, D. R. (1988). *DATASIM*. Lewiston, ME: Desktop Press.

BRADLEY, D. R. (1989a). Computer simulation with DATASIM. *Behavior Research Methods, Instruments, & Computers*, **21**, 99-112.

BRADLEY, D. R. (1989b). A general purpose simulation program for statistics and research methods. In G. Garson & S. Nagel (Eds.), *Advances in social science and computers* (Vol. 1, pp. 145-186). Greenwich, CT: JAI Press.

BRADLEY, D. R., SENKO, M. W., & STEWART, F. A. (1990). Statistical simulation on microcomputers. *Behavior Research Methods, Instruments, & Computers*, **22**, 236-246.

BRATLEY, P., FOX, B. L., & SCHRAGE, E. L. (1983). *A guide to simulation*. New York: Springer-Verlag.

BRYSBAERT, M. (1991). Algorithms for randomness in the behavioral sciences: A tutorial. *Behavior Research Methods, Instruments, & Computers*, **23**, 45-60.

BRYSBAERT, M., & CAVEGN, D. (1993). *A note on reseeding pseudorandom number generators*. Manuscript submitted for publication.

CASTELLAN, N. J., JR. (1992). Shuffling arrays: Appearances may be deceiving. *Behavior Research Methods, Instruments, & Computers*, **24**, 72-77.

CERN (1989). *CERN program library* [Computer program]. Geneva, Switzerland: European Organization for Nuclear Research.

CONVEYOU, R. R., & MACPHERSON, R. D. (1967). Fourier analysis of uniform random number generators. *Journal of the Association for Computing Machinery*, **14**, 100-119.

DAVIES, N., & NEWBOLD, P. (1979). Some power studies of a port-

manteau test of time series model specification. *Biometrika*, **66**, 153-155.

DIACONIS, P., & EFRON, B. (1983). Computer-intensive methods in statistics. *Scientific American*, **247**(5), 96-129.

DUDEWICZ, E. J., & RALLEY, T. G. (1981). *The handbook of random number generation and testing with TESTRAND computer code*. Columbus, OH: American Sciences.

EDGELL, S. E. (1979). A statistical check of the DECsystem-10 FORTRAN pseudorandom number generator. *Behavior Research Methods & Instrumentation*, **11**, 529-530.

EDGINGTON, E. S. (1961). Probability table for number of runs of signs of first differences in ordered series. *Journal of the American Statistical Association*, **56**, 156-159.

ESSL (1990). *Engineering and Scientific Subroutine Library: Guide and reference, Release 4* (5th ed.). New York: IBM.

FERRENBERG, A. M., LANDAU, D. P., & WONG, Y. J. (1992). Monte Carlo simulations: Hidden errors from "good" random number generators. *Physical Review Letters*, **69**, 3382-3384.

FISHMAN, G. S., & MOORE, L. R. (1982). A statistical evaluation of multiplicative congruential random number generators with modulus $2^{31}-1$. *Journal of the American Statistical Association*, **77**, 129-136.

FISHMAN, G. S., & MOORE, L. R., III (1986). An exhaustive analysis of multiplicative congruential generators with modulus $2^{31}$-1. *Society for Industrial & Applied Mathematics Journal of Scientific & Statistical Computing*, **7**, 24-45.

GOLDER, E. R. (1976a). Algorithm AS98: The spectral test for the evaluation of congruential pseudo-random generators. *Applied Statistics*, **25**, 173-180.

GOLDER, E. R. (1976b). Remark ASR18: The spectral test for the evaluation of congruential pseudo-random generators. *Applied Statistics*, **25**, 324.

HETH, C. D. (1984). A Pascal version of a pseudorandom number generator. *Behavior Research Methods, Instruments, & Computers*, **16**, 548-550.

HOAGLIN, D. C., & KING, M. L. (1978). Remark ASR24: A remark on algorithm AS98: The spectral test for the evaluation of congruential pseudorandom generators. *Applied Statistics*, **27**, 375-377.

HULL, T. E., & DOBELL, A. R. (1962). Random number generators. *Society for Industrial & Applied Mathematics Review*, **4**, 230-254.

IBM CORPORATION (1986). *The IBM Personal Computer BASIC, Version A3.21* [Computer program]. Portsmouth, NH: Author.

IBM CORPORATION (1987). *VS FORTRAN, Version 2* [Computer program]. San Jose, CA: Author.

IMSL (1987). *Integrated Mathematical and Statistical Library: User's manual, Version 1.0*. Houston: Author.

KATZEN, H., JR. (1970). *APL programming and computer techniques*. New York: Van Nostrand.

KNUTH, D. E. (1969). *The art of computer programming: Vol. 2. Seminumerical algorithms*. Reading, MA: Addison-Wesley.

KNUTH, D. E. (1981). *The art of computer programming: Vol. 2. Seminumerical algorithms* (2nd ed.). Reading, MA: Addison-Wesley.

LANDMARK RESEARCH INTERNATIONAL (1990). *Landmark System Speed Test, Version 2.0* [Computer program]. Clearwater, FL: Author.

LEARMONTH, G. P., & LEWIS, P. A. W. (1974). Statistical tests of some widely used and recently proposed uniform random number generators. In W. J. Kennedy (Ed.), *Proceedings of the Seventh Conference on the Computer Science and Statistics Interface*. Ames, IA: Iowa State University Press.

L'ECUYER, P. (1988). Efficient and portable combined random number generators. *Communications of the Association for Computing Machinery*, **31**, 742-749, 774.

L'ECUYER, P. (1990). Random numbers for simulation. *Communications of the Association for Computing Machinery*, **33**(10), 86-97.

LEWIS, P. A. W., GOODMAN, A. S., & MILLER, J. M. (1969). A pseudo-random number generation system for the System 360. *IBM Systems Journal*, **8**, 136-146.

LEWIS, P. A. W., & ORAV, E. J. (1989). *Simulation methodology for statisticians, operations analysts, and engineers* (Vol. 1). Pacific Grove, CA: Wadsworth.

LEWIS, P. A. W., ORAV, E. J., & URIBE, L. C. (1988). *Enhanced Simulation and Statistics Package*. Pacific Grove, CA: Wadsworth.

LEWIS, P. A. W., & URIBE, L. C. (1988). *The new Naval Postgraduate School random number generator package LLRANDOMII* [Computer program]. Monterey, CA: Naval Postgraduate School.

LORDAHL, D. S. (1988). Repairing the Microsoft BASIC RND function. *Behavior Research Methods, Instruments, & Computers*, **20**, 221-223.

MACLAREN, N. M. (1989). The generation of multiple independent sequences of pseudorandom numbers. *Applied Statistics*, **38**, 351-359.

MARSAGLIA, G. (1968). Random numbers fall mainly in the planes. *Proceedings of the National Academy of Science*, **61**, 25-28.

MARSAGLIA, G. (1972). The structure of linear congruential sequences. In S. K. Zaremba (Ed.), *Applications of number theory to numerical analysis*. New York: Academic Press.

MARSAGLIA, G. (1976). Random number generation. In A. Ralston (Ed.), *Encyclopedia of computer science* (pp. 103-152). New York: Van Nostrand Reinhold.

MARSAGLIA, G. (1985). A current view of random number generators. In L. Billard (Ed.), *Computer science and statistics: Proceedings of the Sixteenth Symposium on the Interface* (pp. 3-10). Amsterdam: North-Holland.

MCLEOD, A. I. (1985). A remark on Algorithm AS183: An efficient and portable pseudo-random number generator. *Applied Statistics*, **34**, 198-200.

MICROSOFT CORPORATION (1987). *Graphic & Window-BASIC, Version 3.2* [Computer program]. Bellevue, WA: Author.

MICROSOFT CORPORATION (1991). *QBASIC, Version 1.0* [Computer program]. Bellevue, WA: Author.

MODIANOS, D. T., SCOTT, R. C., & CORNWELL, L. W. (1984). Random-number generation on microcomputers. *Interfaces*, **4**, 81-87.

MODIANOS, D. T., SCOTT, R. C., & CORNWELL, L. W. (1987, January). Testing intrinsic random-number generators. *Byte*, pp. 175-178.

MOOD, A. M., GRAYBILL, F. A., & BOES, D. C. (1974). *Introduction to the theory of statistics* (3rd ed.). Singapore: McGraw-Hill.

NILSSON, T. H. (1978). Randomization without replacement using replacement without losing your place. *Behavior Research Methods & Instrumentation*, **10**, 419.

NUMERICAL ALGORITHMS GROUP (1990). *The NAG Fortran Library* (Mark 14) [Computer program]. Oxford, U.K.: Author.

PARK, S. K., & MILLER, K. W. (1988). Random number generators: Good ones are hard to find. *Communications of the Association for Computing Machinery*, **31**, 1192-1201.

PAYNE, C. D. (Ed.) (1987). *The GLIM System, Release 3.77* (2nd ed.). Oxford, U.K.: NAG and the Royal Statistical Society.

RASMUSSEN, J. L. (1984). A FORTRAN program for statistical evaluation of pseudorandom number generators. *Behavior Research Methods, Instruments, & Computers*, **16**, 63-64.

RIPLEY, B. D. (1983). Computer generation of random variables: A tutorial. *International Statistical Review*, **51**, 301-319.

RIPLEY, B. D. (1987). *Stochastic simulation*. New York: Wiley.

RIPLEY, B. D. (1988). Uses and abuses of statistical simulation. *Mathematical Programming*, **42**, 53-68.

SAS INSTITUTE INC. (1988). *SAS language guide, Release 6.03*. Cary, NC: Author.

SAS INSTITUTE INC. (1989). *Statistical Analysis System for Personal Computers, Release 6.04* [Computer program]. Cary, NC: Author.

SAS INSTITUTE INC. (1991). *Statistical Analysis System for VM/CMS, Release 6.07* [Computer program]. Cary, NC: Author.

SCHRAGE, L. (1979). A more portable FORTRAN random number generator. *Association for Computing Machinery Transactions on Mathematical Software*, **5**, 132-138.

SIEGEL, S., & CASTELLAN, N. J., JR. (1988). *Nonparametric statistics for the behavioral sciences* (2nd ed.). New York: McGraw-Hill.

SPSS INC. (1983). *Statistical Package for the Social Sciences*. Chicago: Author.

STRUBE, M. J. (1983). Tests of randomness for pseudorandom number generators. *Behavior Research Methods & Instrumentation*, **15**, 536-537.

THISTED, R. A. (1987). *Elements of statistical computing*. New York: Chapman & Hall.

VAN ES, A. J., GILL, R. D., & VAN PUTTEN, C. (1983). Random number generators for a pocket calculator. *Statistica Neerlandica*, **37**, 95-102.

WHICKER, M. L., & SIGELMAN, L. (1991). *Computer simulation applications*. Newbury Park: Sage.
WHITNEY, C. A. (1984, October). Generating and testing pseudorandom numbers. *Byte*, pp. 128-129, 442-464.
WICHMANN, B. A., & HILL, J. D. (1982). Algorithm AS183: An efficient and portable pseudo random number generator. *Applied Statistics*, **31**, 188-190.
WICHMANN, B. A., & HILL, J. D. (1984). An efficient and portable pseudo random number generator: Correction. *Applied Statistics*, **33**, 123.
WICHMANN, B. A., & HILL, J. D. (1987, March). Building a random-number generator. *Byte*, pp. 127-128.
ZEISEL, H. (1986). A remark on Algorithm AS183: An efficient and portable pseudo-random number generator. *Applied Statistics*, **35**, 89.

## NOTES

1. We will not deal with the Fibonacci and the Tausworthe generators because they are rarely used, and only little theory is available (see Marsaglia, 1976, 1985).

2. In former SAS versions,there was a separate UNIFORM function using a multiplicative congruential generator with multiplier 16,807, modulus $2^{31}$, and a 64-value shuffle table. The seed had to be either 0 or a five-, six-, or seven-digit odd integer (SAS, 1988). From Version 6.04 on, the UNIFORM function produces the same streams of numbers as the RANUNI function.

## APPENDIX A
## A Turbo Pascal Function for the Prime-Modulus Multiplicative Congruential Generator Proposed by Fishman and Moore (1986) Using 80-Bit Extended-Precision Arithmetic

```
function fishman : extended;

{Fishman is a function to generate one pseudorandom real
between zero and one following Fishman and Moore (1986).
An extended-precision variable with the name 'seed' should be
initialized in the main program with a positive integer. A mathe-
matical coprocessor is required.}

const a : extended = 742938285; {the multiplier}
      m : extended = 2147483647; {the modulus}

begin
     seed:=(a*seed) - (int((a*seed)/m))*m;
     fishman:=seed/m;
end;
```

{The multiplier *a* can be changed to 62089911. If the multiplier is changed to 397204094, the generator proposed by Learmonth and Lewis (1974) is obtained. If it is changed to 16807, the generator proposed by Lewis, Goodman, and Miller (1969) is obtained. In the latter case only double-precision arithmetic is required.

The correctness of the implementation can be checked with:

```
program testfish;

uses crt;

var i : longint;
    u, seed : extended;

begin
     clrscr;
     seed:=2147483646;
     for i:=1 to 10 do
        begin
        u:=fishman;
        writeln(u:10:10);
        end;
     readln
end.
```

This should give .6540424017, .2032902977, .1634123433, .0948051145, .1617738056, .6769099178, .4410270808, .0819611824, .3259203002, and .9101976547.}

## APPENDIX B
## A VS FORTRAN Version 2 Subroutine for the Prime Modulus Multiplicative Congruential Generator Proposed by Fishman and Moore (1986), Using 128-Bit Quadruple-Precision Arithmetic

```
      SUBROUTINE FISHMAN(QSEED,U,N)
C
      INTEGER N,I
      REAL*16 QSEED,U(N),QMULT,Q31M1
C
C     FISHMAN is a subroutine to generate pseudorandom reals between zero and one following Fish-
C     man and Moore (1986). On entry, QSEED and N need to be specified. QSEED is a quadruple-
C     precision constant between 1Q0 and 2147483646Q0 and N is an integer constant specifying the
C     number of pseudorandom reals. The output vector U gives N quadruple-precision pseudorandom
C     reals between zero and one.
C
      QMULT=1343714438Q0
      Q31M1=2147483647Q0
      DO 7 I=1,N
      QSEED=QMOD(QMULT*QSEED,Q31M1)
      U(I)=QSEED/Q31M1
    7 CONTINUE
      RETURN
      END
```

```
C
C     QMULT can be changed to 1226874159Q0, 950706376Q0, 742938285Q0, or 62089911Q0. If
C     QMULT is changed to 397204094Q0, the generator proposed by Learmonth and Lewis (1974) is
C     obtained. If QMULT is changed to 16807D0, the generator proposed by Lewis, Goodman, and
C     Miller (1969) is obtained. In the latter case only double-precision arithmetic is required.
C
C     The correctness of the implementation can be checked with:
C
C     INTEGER J
C     REAL*16 V(10)
C     CALL FISHMAN(2147483646Q0,V,10)
C     DO 6 J=1,10
C     WRITE (*,100) V(J)
C   6 CONTINUE
C 100 FORMAT (F12.10)
C     STOP
C     END
C
C     This should give .3742842047, .8185105211, .8821909571, .1886723238, .5398265391,
C     .6456288102, .8941928232, .8355328761, .0669999332, and .6502664646.
```

## APPENDIX C
## Description of the Battery of Statistical Tests Used in the Empirical Analysis of Pseudorandom Number Generators

The Kolmogorov-Smirnov distribution test and the chi-square goodness-of-fit test are well-known tests for uniformity (see, e.g., Siegel & Castellan, 1988). The other tests are less known and are presented below. This has already been done for this journal by Edgell (1979), Strube (1983), and Rasmussen (1984), but here some further details are given.

### Gaps, Runs-Above-the-Mean, and Runs-Below-the-Mean Tests

Gaps tests are used to test for cyclical trends in a series of $n$ observations. The term *gap* is used to describe the distance in the series between two numbers that lie in the interval $(r_l, r_u)$. A gap ends at $x_j$ if $r_l \leq x_j \leq r_u$ and the next gap begins at $x_{j+1}$.

To perform the gaps test, the number of gaps of different lengths is counted. Let $o_i$ denote the number of gaps of length $i$, for $i = 1, 2, \ldots k-1$, and let $o_k$ denote the number of gaps of length $k$ or greater. An unfinished gap at the end of a sequence is not counted.

Under the null hypothesis of randomness, the expected frequencies for gaps of length $i$, $e_i$, is

$$e_i = p(1-p)^{i-1} \sum_{t=1}^{n} o_t$$

if $i < k$, and

$$e_i = (1-p)^{i-1} \sum_{t=1}^{n} o_t$$

if $i = k$, with

$$p = \frac{r_u - r_l}{r_{\max} - r_{\min}}$$

where $r_{\max} - r_{\min}$ is the length of the interval of all possible numbers. The usual $\chi^2$ statistic with $k-1$ degrees of freedom,

$$\chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i},$$

is used to test this null hypothesis.

A runs-above-the-mean test can be understood as a gaps test with $r_l$ equal to the lowest possible $x_j$ and $r_u$ equal to the expected value of $x_j$; a runs-below-the-mean test as a gaps test with $r_l$ equal to the expected value of $x_j$ and $r_u$ equal to the highest possible $x_j$.

### Runs-Up and Runs-Down Tests

Runs tests are used to test whether the frequencies of ascending or descending sequences of certain lengths in a series of observations deviate from the frequencies that might be expected if the observations were random numbers. A run up is a sequence of numbers in increasing order. A run up ends at $x_j$ when $x_{j+1} < x_j$ and the new run begins at $x_{j+1}$.

To perform the runs up test, the number of runs up of different lengths is counted. Let $o_i$ denote the number of runs of length $i$, for $i = 1, 2, \ldots k-1$, and let $o_k$ denote the number of runs of length $k$ or greater. An unfinished run at the end of a sequence is not counted.

An approximate $\chi^2$ statistic with $k$ degrees of freedom,

$$\chi^2 = (\mathbf{o}-\mathbf{e})^{\mathrm{T}} \Sigma^{-1} (\mathbf{o}-\mathbf{e}),$$

is used, where $\mathbf{o}$ is the vector of observed frequencies, $o_i$, for $i = 1, 2, \ldots k$, $\mathbf{e}$ is the vector of expected frequencies under the null hypothesis of randomness, $e_i$, for $i = 1, 2, \ldots k$, and $\Sigma$ is the matrix of covariances of the frequencies under the null hypothesis. For the derivation of the expected values and covariances of the frequencies, the reader is referred to Knuth (1981).

A run down is a sequence of numbers in decreasing order. A runs-down test can be performed by multiplying each observation by $-1$ and performing a runs-up test. Other runs tests

are described by Edgington (1961), Rasmussen (1984), and Ripley (1987).

### Pairs and Triplets Tests

Pairs and triplets tests are used to test whether the frequencies of certain pairs or triplets of adjacent observations in a series of $n$ observations deviate from the frequencies that might be expected if the observations were random numbers. For the pairs test, an $m \times m$ matrix $\mathbf{O}$ of observed frequencies is formed containing the elements $o_{jk}$, which are the number of pairs $(x_i, x_{i+1})$ such that

$$\frac{j-1}{m} < x_i < \frac{j}{m}$$

and

$$\frac{k-1}{m} < x_{i+1} < \frac{k}{m},$$

where i = 1, 3, 5, ... $n-1$, and $m$ is chosen such that none of the cells has an expected frequency of less than 5 (Siegel & Castellan, 1988).

Under the null hypothesis that the sequence is random, the expected number of pairs for each class (i.e., each element of the frequency matrix) is the same; that is, the pairs should be uniformly distributed over the unit square $[0,1]^2$. Thus the expected frequency for each class, $e_{jk}$, is just the total number of pairs divided by the number of classes:

$$e_{jk} = e = \frac{\sum_{j=1}^{m}\sum_{k=1}^{m} o_{jk}}{m^2}.$$

A $\chi^2$ statistic with $m^2-1$ degrees of freedom,

$$\chi^2 = \frac{\sum_{j=1}^{m}\sum_{k=1}^{m} (o_{jk}-e)^2}{e},$$

is used to test the hypothesis of randomness.

For the triplets test, an $m \times m$ tensor is formed in a similar way. The null hypothesis, then, is that the triplets are uniformly distributed over the unit cube $[0,1]^3$. A $\chi^2$ statistic with $m^3-1$ degrees of freedom is used to test the hypothesis of randomness.

### The Autocorrelation Test

An autocorrelation test can be used to check whether there is no correlation between adjacent observations. To perform the autocorrelation test for a series of $n$ observations, the sample autocorrelation coefficients of lags $k = 1, 2, \ldots K$, are computed ($K < n$, $n > 1$). The autocorrelation coefficient of lag $k$ is defined as

$$r_k = \frac{\sum_{i=1}^{n-k} (x_i-\bar{x})\,(x_{i+k}-\bar{x})}{\sum_{i=1}^{n} (x_i-\bar{x})^2}.$$

A $\chi^2$ statistic with $K$ degrees of freedom, defined as

$$\chi^2 = n\sum_{k=1}^{K} r_k^2,$$

can be used to test the hypothesis of a zero autocorrelation function (Box & Jenkins, 1976, pp. 290-293; Box & Pierce, 1970; Davies & Newbold, 1979).

### The Metalevel Kolmogorov-Smirnov Test

The results of first-level statistical tests for randomness can be used again as input for a metalevel Kolmogorov-Smirnov test to increase power (L'Ecuyer, 1988; Ripley, 1987). For, under the null hypothesis of randomness, $N$ independent repetitions of first-level statistical tests result in $N$ test statistics from a specific theoretical distribution of test statistics. Consequently, the empirical distribution of test statistics can be compared with this theoretical distribution by using the Kolmogorov-Smirnov criterion to test the null hypothesis of randomness. Furthermore, following the probability integral transform theorem, testing whether $N$ statistics are plausible under the expected theoretical distribution is the same as testing whether the $N$ first-level $p$ values are distributed uniformly in the [0,1] interval (Mood, Graybill, & Boes, 1974, pp. 202-203). With the Kolmogorov-Smirnov distribution test, the $N$ first-level $p$ values are condensed in one metalevel $p$ value. This is the $p$ value that is shown in Table 3.

For example, if 100 gaps tests were performed on 100 consecutive sequences of 200,000 pseudorandom numbers, all of which result in very small $p$ values, then the null hypothesis of randomness would be rejected if one were using the metalevel Kolmogorov-Smirnov test. The same would be true if all $p$ values were very large, or, more generally, if all $p$ values fell within the same small range. So the metalevel Kolmogorov-Smirnov test does not take into account the magnitude of the $p$ values as such, but rather the distribution of the $p$ values between 0 and 1. Only if the $p$ values are distributed evenly over the [0,1] interval will the metalevel $p$ value be high and the null hypothesis of randomness be not rejected.

(Manuscript received March 23, 1992;
revision accepted for publication April 20, 1993.)