The following is based on the Keynote Address, Computer Science and Statistics: 16th Symposium on the Interface, Atlanta, 1984. It appeared in *The Proceedings* of the Conference, published by Elsevier Press.

A Current View of Random Number Generators

George Marsaglia

The ability to generate satisfactory sequences of random numbers is one of the key links between Computer Science and Statistics. Standard methods may no longer be suitable for increasingly sophisticated uses, such as in precision Monte Carlo studies, testing for primes, combinatorics or public encryption schemes. This article describes stringent new tests for which standard random number generators: congruential, shift-register and lagged-Fibonacci, give poor results, and describes new methods that pass the stringent tests and seem more suitable for precision Monte Carlo use.

1. INTRODUCTION

Most computer systems have random number generators available, and for most purposes they work remarkably well. Indeed, a random number generator is much like sex: when its good its wonderful, and when its bad its still pretty good.

But many of the standard random number generators (RNG's) are not good enough for increasingly sophisticated Monte Carlo uses, such as in geometric probability, combinatorics, estimating distribution functions, comparing statistical procedures, generating and testing for large primes for use in encryption schemes and the like. At least, that is my current view, and I will give reasons for it, by describing new, more stringent tests that standard RNG's fail yet new kinds of generators pass.

The more stringent tests are still reasonable, related to the kinds of applications of RNG's that cheap and fast computing power have made feasible, often calling for samples of hundreds of thousands or millions. The kinds of generators that pass the more stringent tests are those that combine simple, standard generators in various ways.

Most of the standard RNG 's produce a sequence of elements by means of a linear transformation on some algebraic structure. Three methods dominate: congruential, shift-register and lagged-Fibonacci. These will be called *simple* RNG's; details and properties will be summarized in sections below. Most examples of simple generators pass standard tests for randomness, such as those enumerated in Knuth's Volume 2 [6], but may fail spectacularly on one or more of the new, stringent tests described below. Methods that combine two of the simple generators do much better on the stringent tests.

After summary descriptions of the three most common simple RNG's : congruential, shift-register and lagged-Fibonacci (with more detail on the latter two, as, in spite of being considered by various researchers for the past 25 years, they are not as widely known as congruential RNG's), I will give some theoretical justification for combining simple generators, then describe new, stringent tests that simple generators fail but combination generators pass.

2. SIMPLE GENERATORS: CONGRUENTIAL

These generators use a linear transformation on the ring of reduced residues of some modulus m, to produce a sequence of integers:

$$x_1, x_2, x_3, \ldots \text{ with } x_n = ax_{n-1} + b \bmod m.$$

They are the most widely used RNG's, and they work remarkably well for most purposes. But for some purposes they are not satisfactory; points in n-space produced by congruential RNG's fall on a lattice with a huge unit cell volume, m^{n-1}, compared to the unit cell volume of 1 that would be expected from truly random integers. Details are in [9,10]. Congruential RNG's perform well on many of the stringent tests described below, but not on all of them.

3. SIMPLE GENERATORS: SHIFT-REGISTER

These are based on viewing the bits of a computer word as the elements of a binary vector, then using iterates of a linear transformation to generate a sequence of binary vectors, and hence computer words, that may be interpreted as a sequence of uniform random integers. In terms of vectors and matrices, the sequence e is $\beta, \beta T, \beta T^2, \beta T^3, \ldots$ with β a $1 \times n$ binary vector, T an $n \times n$ binary matrix of 0's and 1's, all arithmetic modulo 2 and addition of binary vectors the exclusive-or operation (designated $\oplus$) of the two corresponding computer words. Shift-register generators are sometimes called Tausworthe generators. The binary matrix T is usually chosen so that the product βT may be produced with simple computer operations. A good choice is $T=(I+R^s)(I+L^t)$, where R is the matrix that transforms every vector $\beta=(b_1,b_2,\ldots,b_{n-1},b_n)$ into $\beta(b_2,b_3,\ldots,b_n,0)$. Thus R is all 0's except for 1's on the principal super-diagonal; L is all 0's except for 1's on the principal subdiagonal.

Then $\beta T=\beta(I+R^s)(I+L^t)$ may be easily produced by forming $\beta\leftarrow\beta\oplus\beta R^s$, with a logical right-shift-s and a $\oplus$, followed by $\beta\leftarrow\beta\oplus\beta L^t$ with a logical left-shift-t and another $\oplus$. The maximum possible period of a shift-register generator is 2^n-1, the number of non-null $1\times n$ binary vectors. For such a generator, any non-null initial (seed) vector may be used, a desirable property for generators that allow setting the seed value, which to the user becomes any non-zero integer. Matrix theory provides an easy means to characterize shift-register sequences of maximal period; for an easy proof, see [11].

THEOREM 1. *Let T be a nonsingular $n\times n$ binary matrix. In order that the sequence $\beta,\beta T,\beta T^2,\ldots$ have period 2^n-1 for every non-null binary seed vector β, it is necessary and sufficient that the matrix T have order 2^n-1 in the group of nonsingular $n\times n$ binary matrices.*

4. SIMPLE GENERATORS: LAGGED-FIBONACCI

These generators use an initial set of elements $x_1,x_2,\ldots,x_r$ and two "lags" r and s with $r>s$. Successive elements are generated by the recursion, for $n>r$: $x_n=x_{n-r}\diamond x_{n-s}$, where $\diamond$ is some binary operation. The initial (seed) elements are computer words and the binary operation might be $+,-,*$ or $\oplus$ (exclusive-or). For $+$ or $-$, the x's might be integers mod 2^k or single- or double-precision reals mod 1. For $*$, odd integers mod 2^k. We designate such a generator loosely as F(r,s,$\diamond$), although each lagged-Fibonacci generator depends on details of the particular binary operation and the finite set of elements it operates on.

Examples of generators of maximal period are F(17,5,+) or F(17,5,$-$) on integers mod 2^k, period $(2^{17}-1)2^{k-1}$; F(17,5,$-$) on 32-bit reals with 24 bit fractions, subtraction modulo 1, period $(2^{17}-1)2^{23}$ or double precision reals modulo 1, period $(2^{17}-1)2^{55}$; F(17,5,$*$) on odd integers mod 2^{32}, period $(2^{17}-1)2^{29}$; F(17,5,$\oplus$), period $2^{17}-1=131071$. Other good choices for (r,s) are (31,13),(55,24),(68,33),(97,33),(607,273),(1279,418) plus many others, some of which are listed in Knuth [6, p28].

A lagged-Fibonacci generator is easily programmed, using a circular list and two pointers. For example, a procedure for an F(17,5,$\diamond$) generator uses 17 memory locations $L()$, initially filled as $L(1)=x_{17},L(2)=x_{16},\ldots,L(17)=x_1$ and pointers $I=17$, $J=5$. Then each call to the procedure executes these instructions:

$L(I)\leftarrow L(I)\diamond L(J)$
output $L(I)$
$I\leftarrow I-1$; If $I=0$, $I\leftarrow 17$
$J\leftarrow J-1$; If $J=0$, $J\leftarrow 17$

(Decrementing the pointers and testing on 0 is usually faster than incrementing and testing on 17.)

Characterization of maximal period F(r,s,-) generators may be based on the theory of linear recursive sequences of integers:

$$x_n=a_1x_{n-1}+a_2x_{n-2}+\cdots+a_kx_{n-k} \bmod m.$$

There is extensive literature on this problem, much more extensive than necessary for our purposes, for it is possible to develop a brief self-contained treatment for characterizing lagged-Fibonacci generators of maximal period for the most important modulus, 2^n, using only elementary matrix theory.

For lagged-Fibonacci generators, the integer recursion reduces to $x_n=x_{n-r}\pm x_{n-s}$, but the theory for the general integer recursion mod 2^n is just as easily established, by considering an initial vector of integers, $\alpha=(x_1,x_2,...,x_r)$ and successive vectors $\alpha T,\alpha T^2,\alpha T^3,\ldots$ generated by a matrix T of integers. Here is the result; for a proof see [11].

THEOREM 2. *Let r be an $r\times r$ matrix of integers, with odd determinant. In order that the sequence of vectors*

$$\alpha,\alpha T,\alpha T^2,\alpha T^3,\ldots \mod 2^n$$

have period $(2^r-1)2^{n-1}$ *for every* $n \geq 1$ *and every initial vector of integers* $\alpha = (x_1, \ldots, x_r)$ *not all even, it is necessary and sufficient that* T *have order* $j = 2^r - 1$ *in the group of non-singular matrices for mod 2, and order 2j for mod 4 and order 4j for mod 8.*

To verify that a particular F(r,s,+) or F(r,s,-) generator has maximal period $j = (2^r-1)2^{n-1}$ for integers mod 2^n, one need only call a matrix-squaring routine (mod 8) a few more than r times to verify that T has order j ,2j ,4j for mod 2,4,8. *This will be true only if* $H = T^{j+1}$ *is* T *mod 2, not* T *mod 4 and* H^2 *not* T mod 8.

The matrix T is a companion matrix, with 0's everywhere except for 1's on the principal sub-diagonal and two 1's in the appropriate positions of the last column. For example, the F(3,1,+) generator on integers mod 2^n has $T = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$.

A few calls to a matrix-squaring routine verifies that T^8 is T mod 2, not T mod 4 and T^{16} is not T mod 8. Since T is nonsingular and $j = 2^3 - 1$ is prime, the order of T for modulus 2 is j. When j is composite, say $j = 2^{55} - 1$, it takes a little more work to verify the order of T. Successive squaring will verify that T^{j+1} is T for mod 2 and not T for mod 4 etc., but one must also verify that T^k is not I mod 2 for each $k = j/p$, with p ranging over the prime divisors of j. Even then, a simple computer program will serve to establish—or refute—that a proposed F(r,s,+) or F(r,s,-) generator has maximal period.

To find the period of an F(r,s,$*$) generator under multiplication of residues relatively prime to a modulus m, one need only express the Abelian group of those residues as a direct product of cyclic groups, then consider the F(r,s,+) generator on the exponents of the generators of the cyclic groups. For example, every odd integer mod 2^n has a unique representation as a product $(-1)^i 3^j$ with $i \in \{0,1\}$ and $j \in \{1, 2, \ldots, 2^{n-2}\}$. Thus the period of F(r,s,$*$) for odd integers mod 2^n is the period of the F(r,s,+) generator on integers mod 2^{n-1}:
If the F(r,s,+) generator has maximal period, $(2^r-1)2^{n-1}$, *for integers mod* 2^n *then the F(r,s,$*$) generator on odd integers mod* 2^n *has period* $(2^r-1)2^{n-3}$.

The F(r,s,$-$) generators on reals are particularly suitable for random number subroutines that return UNI or VNI, continuous random variables on [0,1) or (-1,1). Ordinarily, such subroutines generate a random integer from some set, then divide by the largest integer in the set to get the required UNI or VNI. If the initial reals $x_1, x_2, \ldots, x_r$ are all binary fractions of the form $k/2^{24}$, then the binary operation $x \diamond y = \{\text{if } x \geq y \text{ then } x-y \text{ else } x-y+1\}$ in the F(r,s,$\diamond$) generator will produce a sequence of reals on [0,1), each a binary fraction $k/2^{24}$ with numerator the integer that would be produced by the corresponding F(r,s,$-$) generator on integers mod 2^{24}.

Thus the F(r,s,$\diamond$) generator using that above $\diamond$ on 32-bit reals (having 24-bit fractions) produces real UNI's on [0,1) with period $(2^r-1)2^{23}$ directly, avoiding the division operation necessary in conventional generators. The same method will directly produce 64-bit double precision reals with period $(2^r-1)2^{55}$.

Lagged-Fibonacci Generators with $\oplus$

The F(r,s,$\oplus$) sequence starts with an initial set of computer words $x_1, x_2, \ldots, x_r$, then generates successive words by the recursion $x_n = x_{n-r} \oplus x_{n-s}$. The tests described below show generators based on this sequence to be among the worst of all generators, unless the lag r is some 600 or more. In addition, the maximum possible period is $2^r - 1$, whatever the word size, far short of the attainable $(2^r-1)2^{n-1}$ for F(r,s,+), F(r,s,$-$) or $(2^r-1)2^{n-3}$ for F(r,s,$*$) with words of n bits. The exclusive-or operation, $\oplus$, is no faster than $+$ or $-$ in most computers, and not available in many high-level languages. So, taken with the poor statistical performance and far shorter periods, one wonders why F(r,s,$\oplus$) generators have ever been given serious consideration. But they have. Called "generalized feedback shift register generators" (GFSR's), they have been the subject of several papers [1,2,4,7].

5. COMBINATION GENERATORS

Empirical studies suggest that combining two or more simple generators, by means of a convenient computer operation such as $+,-,*$ or $\oplus$ (exclusive-or), provides a composite with better randomness than either of the components. There is interesting theoretical support for such an observation.

Let x be a random variable taking values in a finite set S. To fix ideas, let $S = \{1, 2, 3\}$; the theory is easily extended to any finite set. Let the probability vector for x be (a, b, c), that is,

$\Pr(x=1)=a,\ \Pr(x=2)=b,\ \Pr(x=3)=c.$

We seek the uniform distributions on S, with probability vector $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$. Let $\delta(x)$ be the distance between the distribution of x and the uniform distribution, defined as the distance between the corresponding probability vectors:

$$\delta(x)=||(a,b,c)-(\frac{1}{3},\frac{1}{3},\frac{1}{3})||,$$

where $||$ is the encloser for any vector norm: L_1, L_2, L_∞ or any L_p. Let y be some other random variable on S, independent of x, with probability vector (r,s,t). Furthermore, suppose we have a binary operation $\bullet$ on S, with operation table forming a Latin square, say

$\bullet$	1	2	3
1	3	1	2
2	1	2	3
3	2	3	1

Then we may use the binary operation $\bullet$ to form a new random variable, $z=x\bullet y$, and the distribution of z will be "closer" to uniform than that of either x or y:

THEOREM 3. *$\delta(x\bullet y)\le\delta(x)$ and $\delta(x\bullet y)\le\delta(y)$.*

The proof is simple for the above S and $\bullet$, and it generalizes in an obvious way.

$$\begin{aligned}\Pr(x\bullet y=1)&=as+br+ct\\ \Pr(x\bullet y=2)&=at+bs+cr\\ \Pr(x\bullet y=3)&=ar+bt+cs.\end{aligned}$$

Thus the probability vector for $z=x\bullet y$ is

$$(a,b,c)\begin{pmatrix} s & t & r\\ r & s & t\\ t & r & s\end{pmatrix}=(a,b,c)M,\ \text{say}.$$

Then, since $(\frac{1}{3},\frac{1}{3},\frac{1}{3})M=(\frac{1}{3},\frac{1}{3},\frac{1}{3})$,

$$\delta(x\bullet y)=||(a,b,c)M-(\frac{1}{3},\frac{1}{3},\frac{1}{3})||=||[(a,b,c)-(\frac{1}{3},\frac{1}{3},\frac{1}{3})]M||.$$

It follows that $\delta(x\bullet y)\le\delta(x)$ because for any (f,g,h),

$$||(f,g,h)M||\le||(f,g,h)||\,||M||\le||(f,g,h)||$$

as long as $||M||\le 1$. Here $||M||$ is the matrix norm induced by the vector p-norm $||$. This is certainly true for the matrix M arising from the Latin square operation table for $\bullet$, since the rows and columns of M are probability vectors, non-negative summing to 1. The L_1 norm of M is 1, the max column sum, the L_∞ norm of M is 1, the max row sum, and the L_2 (Euclidean) norm of M is the square root of the dominant eigenvalue of MM', at most 1.

This theoretical result suggests that if two RNG's produce sequences $x_1,x_2,x_3,\ldots$ and $y_1,y_2,y_3,\ldots$ on some finite set S on which we have a binary operation $\bullet$, then the combination generator, producing $x_1\bullet y_1, x_2\bullet y_2, x_3\bullet y_3,\ldots$ should be better, or at least no worse than, either of the component RNG's. Applications might include $+$ or $-$ for residues of some modulus m, exclusive-or, $\oplus$, on computer words, or multiplication on the set S of residues relatively prime to modulus m, such as odd integers mod 2^n. The binary operation need only have the property that its operation table forms a Latin square; it need be neither commutative nor associative.

Another attractive feature of combination RNG's is that they tend not only to make the results more uniform—they make them more independent. Given $x_1,x_2,x_3,\ldots$ we hope that x_1 is uniform on S, that (x_1, x_2) is uniform on $S\times S$, that (x_1, x_2, x_3) is uniform on $S\times S\times S$, and so on. If we combine $x_1, x_2, x_3, \ldots$ and $y_1, y_2, y_3, \ldots$ then (x_1, x_2, x_3) has a certain distribution on the finite set $S\times S\times S$, as does (y_1, y_2, y_3), and thus $(x_1 \bullet y_1, x_2 \bullet y_2, x_3 \bullet y_3)$ has a distribution at least as uniform on $S \times S \times S$. But that means the combined vector tends to have independent coordinates, since uniformity on a product set $S\times S\times S$ implies independence of the coordinates.

These remarks apply to any two random k-tuples $(x_1, \ldots, x_k)$ and $(y_1, \ldots, y_k)$:
on the product set S^k, the composite vector $(x_1 \bullet y_1, \ldots, x_k \bullet y_k)$ is likely to be closer to the ideal of having independent, uniformly distributed elements than either of the contributing vectors.

What is the period of a combination generator? The reader may wish to convince himself of the following: If $x_1, x_2, x_3, \ldots$ and $y_1, y_2, y_3, \ldots$ are strictly periodic sequences having periods rd and sd with r and s relatively prime, then the period of $x_1 \bullet y_1, x_2 \bullet y_2, x_3 \bullet y_3, \ldots$ is rst, where t is a divisor of d. Thus the shortest possible period is rs, the longest rsd.

References [8] and [12l seem to have first urged that simple generators be combined; Brown and Solomon [3] provided theoretical support for such combinations. They gave an elaborate proof that $x + y \bmod m$ was more uniform than x or y on residues mod m, relying heavily on the techniques of majorization. Marshall and Olkin [13 p. 383] made the result more general in their elegant book on inequalities and majorization. To those not familiar with the powerful techniques of majorization, the above development, using only elementary properties of vector norms, may be preferable.

6. TESTS OF RNG's

A random number generator is supposed to produce a sequence of independent, identically distributed (iid) random variables $x_1, x_2, x_3, \ldots$. Any function of elements of that sequence may serve as a test, if its distribution is known, or even if its distribution is merely compared with that of other RNG's. In spite of the ease with which tests of RNG's may be created, there are surprisingly few reported in the literature. The same simple, easily passed tests are reported again and again. Such is the power of the printed word.

A few tests were suggested in the early days of making tables of random digits [5], and M. D. MacLaren and I suggested several more [8]. These, and a few others, have become a *de facto* standard set of tests, enumerated in Knuth's V2, [6]. Knuth's books are such marvels that they sometimes discourage initiative—so well done that many readers take them as gospel, the definitive word on the particular subject treated. And so they are, most of the time.

But not, I think, for testing random number generators. Anyone with a knowledge of probability theory should be able to create his own tests. If the RNG is to be used for a particular problem, one should try to create a test based on a similar problem for which the underlying distributions are known, or, lacking that, at least compared with results produced by widely different RNG's. I will describe several tests that I use. They are called **stringent** tests, because they seem more difficult to pass than the mild tests that have become standard. Most of them were developed as analogues to particular Monte Carlo problems, but with conditions that make finding the underlying distributions possible.

7. OVERLAPPING M-TUPLE TESTS

We illustrate with an example. Consider a sequence of n values such as

5,4,5,0,4,2,7,3,4,1,...,5,3,6,2

Suppose these are base-8 digits produced by the first 3 bits of n calls to a RNG. A standard test would use Pearson's chi-square,$\sum(\mathrm{OBS} - \mathrm{EXP})^2/\mathrm{EXP}$, on the individual digits, on non-overlapping pairs, triples, etc. to see if the observed were satisfactorily close to the expected frequencies. Such are typical tests of RNG's: simple and easily passed.

As a means to test independence as well as uniformity, suppose we first make the sequence circular by adjoining the first two elements to the end, getting

5,4,5,0,4,2,7,3,4,1,...,5,3,6,2,5,4

(this has an asymptotically negligible effect, but it makes deriving the subsequent covariance matrix much simpler).

Now consider the n triples formed by successive elements of that sequence:

(5,4,5),(4,5,0),...,(3,6,2),(6,2,5),(2,5,4).

Let w_{ijk} be the number of times the triple (i, j, k) appears in this sequence of n triples. If n is large, the 512 random variables w_{ijk} should be jointly normal with means $\mu_{ijk} = n/512$ and a certain covariance matrix C. If C^- is any weak inverse of C, $CC^-C = C$, then the quadratic form:

$$\sum(w_{ijk} - \mu_{ijk})c^-_{ijk,rst}(w_{rst} - \mu_{rst})$$

is invariant under choice of weak inverse C^- and has (asymptotically) a chi-square distribution with degrees of freedom the asymptotic rank of C. This provides a test for both uniformity and independence of successive values produced by the RNG. Of course, any set of three bits other than the first three could serve for the test, or sets of four bits, etc.

The above quadratic form turns out to be remarkably easy to evaluate. Here is the rule: Compute Q_3, the quadratic form one would use if naively applying Pearson's test to the 3-tuples:

$$Q_3 = \sum_{i,j,k}(w_{ijk} - \mu_{ijk})^2/\mu_{ijk}$$

Then compute Q_2, the naive quadratic form for testing pairs:

$$Q_2 = \sum_{i,j}(w_{ij} - \mu_{ij})^2/\mu_{ij},$$

where w_{ij} is the number of occurrences of the overlapping pair (i, j) in the original sequence. Then the general quadratic form in the weak inverse C^-, above, reduces to $Q_3 - Q_2$ and is asymptotically chi-square with $8^3 - 8^2$ degrees of freedom.

More generally, this is the overlapping 3-tuple test for iid sequences: Let $v_1, v_2, \ldots$ be a sequence of independent random variables taking values $1, 2, \ldots, b$ with probabilities $p_1, p_2, \ldots, p_b$. If w_{ijk} and w_{ij} are, resp., the number of times the triple (i, j, k) and the double (i, j) appear in a circular sequence of n v's, (v_1, v_2 adjoined to get $v_1, v_2, v_3, \ldots, v_n, v_1, v_2$) then

$$\sum_{i,j,k}(w_{ijk} - np_ip_jp_k)^2/(np_ip_jp_k) - \sum_{i,j}(w_{ij} - np_ip_j)^2/(np_ip_j)$$

is asymptotically chisquare with $b^3 - b^2$ degrees of freedom and should be used to test for both uniformity and independence for the sequence $v_1, v_2, \ldots$.

A similar result holds for testing, say, overlapping 4-tuples: the quadratic form $Q_4 - Q_3$ should be chi-square with $b^4 - b^3$ degrees of freedom, where Q_4 is the $\sum(\mathrm{OBS} - \mathrm{EXP})^2/\mathrm{EXP}$ one would get by naive application of Pearson's test to the overlapping 4-tuple counts, and Q_3 etc.

The test results MTUPLE, mentioned below, are based on overlapping 3-tuples for bits 1,2,3 then for bits 2,3,4 then 3,4,5 and so on, for each successive three bits of the computer words produced by the RNG being tested.

Another example of the overlapping m-tuple test is based on overlapping pairs in the circular list $v_1, v_2, \ldots, v_n, v_1, v_2$, with the v's the number of 1's in the first six bits of computer words produced by a RNG. Then the v's take values 0 to 6 with corresponding binomial probabilities. The quadratic form $Q_2 - Q_1$ should be chi-square distributed with $49 - 7 = 42$ degrees of freedom. F(r,s,$\oplus$) generators fail such tests unless r is quite large.

8. OVERLAPPING-PERMUTATION TESTS

These tests are examples of overlapping m-tuple tests for which elements of the overlapping m-tuples are not independent, or even successive states of a Markov chain: Let $u_1, u_2, u_3, \ldots$ be uniform variates produced by a RNG. Each of the overlapping 3-tuples $(u_1, u_2, u_3), (u_2, u_3, u_4), (u_3, u_4, u_5), \ldots$ is in one of six possible states:

S_1: $x < y < z$; S_2: $x < z < y$; S_3: $y < x < z$
S_4: $y < z < x$; S_5: $z < x < y$; S_6: $z < y < x$.

Thus overlapping triples of u's lead to a sequence of states such as

$3, 3, 2, 5, 1, 4, 3, ..., 3, 2, 5, \ldots$

If w_{ijk} represents the number of times that the successive states i, j, k appear in the state sequence, then

$$\sum(w_{ijk} - \mu_{ijk})c^{-}_{ijk,rst}(w_{rst} - \mu_{rst})$$

will have, asymptotically, a chi-square distribution. The means and covariance matrix C, and any weak inverse C^{-} must be found. I have found these for 3-,4- and 5-permutations. These tests, designated OPERM below, are not very stringent; most generators seem to pass them, except for F(r,s,$\oplus$).

9. THE OPSO TEST

OPSO means overlapping-pairs-sparse-occupancy, a test devised to overcome the problem of the huge samples necessary for the overlapping-pairs tests, when the number of possible pairs is very large. For example, if we use the first 11 bits of successive words produced by a RNG to give a sequence of integers in the range 0 to 2047, and want to use the overlapping 2-tuple test, we would need some 2^{26} calls to the RNG in order that the counts w_{ij} be satisfactorily close to jointly normal with the derived covariance matrix. Instead, suppose we envision 2^{22} cells, each associated with a particular pair i, j and consider this generalized occupancy problem: for each successive pair i, j we place a ball in the appropriate cell. If x_0, x_1, x_2 are, resp., the number of cells with 0,1,2 balls then, with very large n, we expect that x_0, x_1, x_2 will be jointly normal with means and covariance matrix to be calculated. With my student, L. H. Tsay, I found the means and covariances some years ago. It turned out, however, that using just x_0, the number of empty cells, served as a good test for RNG's. Here are a few examples of the resulting OPSO test:

The first 10 bits from each of 2^{21} calls to the Super-Duper RNG produced a circular list of 2^{21} integers in the range 0 to 1023, and "placing a ball" in cell(i, j) for each overlapping pair i, j yielded 141,711 empty cells. Subtracting the mean, $\mu = 141909$, and dividing by $\sigma = 290.46$ produced what should have been a standard normal variate, value -.682. Performing the test four times yielded the values -.682, .32, -1.09, -.84. Quite satisfactory.

But four similar values from the congruential RNG $x_n = 62605x_{n-1} + 113218009 \bmod 2^{29}$ (the Berkeley Unix Pascal RNG) produced 1.81, 1.92, 2.15, 3.36. Not so good. This congruential generator fails the OPSO test, but not spectacularly. The congruential RNG $x_n = 69069x_{n-1} \bmod 2^{32}$ produced the four values 4.611, 4.682, 4.114, 5.591—a failure bordering on the spectacular for four supposedly standard normal variates. For something really spectacular: the F(17,5,$\oplus$) generator produced values 2895.9 standard deviations from the mean.

Here is a short table of the number of bits, the sample size n, the mean μ and standard deviation σ for similar OPSO tests:

Bits	n	μ	σ
10	2^{21}	141,909	290.26
11	2^{22}	1,542,998	638.75
11	2^{23}	567,639	580.80

10. PARKING LOT, LATTICE AND RELATED TESTS

These tests are designed to assess the uniformity of points in m-space, where coordinates of the points come from successive calls to a RNG. Congruential generators use iterates of a linear transformation on the ring of residues of some modulus, and as a consequence, m-tuples of points produced by the generator fall on a lattice with a huge unit cell volume [9,10]. Since shift-register generators also use iterates of a linear transformation, there might be some sort of regularity analogous to that for congruential generators, and there is. But the binary vectors produced by a shift-register generator are viewed as base-2 representations of integers in subsequent use, and the regularities get folded over and distorted, much as the original sedimentary layers in the earth are folded and distorted in geological formations viewed eons later. Figure 1 gives an example of such regularities in shift-register generators.

Two figures should be inserted here, but I no longer have the originals. You will have to go to the *Proceedings* volume to see them.

On the top is a set of 16000 random points produced by a good random number generator, one that combines two standard generators (Super-Duper). The bottom shows 16000 points produced by the very shift-register generator proposed by Whittlesey [14] as a replacement for congruential generators, after discovery of their lattice structure [9]. The proposed shift-register generator uses 31 bit binary vectors and $T = (I + R^{28})(I + L^3)$.

Lagged-Fibonacci generators also use iterates of a linear transformation, but lags such as in F(17,5,-), F(31,13,-) or F(55,24,-) generators seem long enough that no obvious regularities appear in dimensions up to 10 or so.

Visual tests such as in figure 1 are striking, but not feasible in higher dimensions. A quantitative version of the parking lot test goes as follows: Let each point in m-space be the center of a cubic or spherical "car", of specified size, and suppose we park "by ear" (as many people do). If $c_1, c_2, c_3 \ldots$ are non-overlapping cars, already parked, we try to park randomly until we succeed with a car that does not hit any of those already parked, then add the new car to the list. Out of n tries, we will have a list of k cars successfully parked. I do not know the distribution of, say, $k(2000)$, but simulation with a good RNG gives the mean and variance accurately enough for comparison with other RNG's. Figure 2 shows the difference in curves $k(n)$, the number of cars parked after n tries, for a combination RNG (top curves) and a shift-register RNG (bottom curves). Clearly the two are different; extensive tries with various combination RNG's suggest the upper curves are those to be expected with a perfect RNG and thus shift-register (and also F(r,s,$\oplus$)) generators fail parking lot tests.

The quantitative parking lot tests can be performed in any dimension; one need only specify the size and shape of "cars", then try m-dimensional parking-by-ear to compare the performance of different RNG's. The random parking of cubes and spheres is an important, unsolved probability problem, and RNG's used for simulating such problems should be beyond reproach.

11. THE BIRTHDAY-SPACINGS TEST

This is the discrete version of what I call the Iterated Spacings Test, which goes like this: Let $S_1, S_2, \ldots, S_n$ be the spacings induced by $n-1$ uniform random variables on [0,1). If the S's are sorted, to get $S_{(1)} \le S_{(2)} \le \cdots \le S_{(n)}$, then the weighted differences:

$$nS_{(1)}, (n-1)[S_{(2)} - S_{(1)}], \ldots, 2[S_{(n-1)} - S_{(n-2)}], 1[S_{(n)} - S_{(n-1)}]$$

form a new set of uniform spacings, to which the KS test may be applied, then the weighted differences of those sorted spacings produce a new set of spacings, and so on, forever: Each initial set of uniform spacings leads to an infinite sequence of sets of uniform spacings.

In practice, the procedure breaks down after 3 or 4 to perhaps 10 iterations, because of the finite representation of uniform variates in the computer. Degeneracy arises from equal values of spacings, and thus the discrete version of this procedure leads to the *Birthday-Spacings Test*:

Let the RNG produce integers $I_1, I_2, I_3, \ldots, I_m$ in the range 1 to n. The I's are m birthdays in a year of n days. The famous problem of Von-Mises and Feller on duplicate birthdays is not stringent enough for a test here, but the problem of duplicate spacings is: Sort the I's to get $I_{(1)} \le I_{(2)} \le \cdots \le I_{(m)}$. Let Y be the number of values which appear more than once among the spacings

$$I_{(1)}, I_{(2)} - I_{(1)}, I_{(3)} - I_{(2)}, \ldots, I_{(m)} - I_{(m-1)}$$

Thus Y is m minus the number of distinct birthday spacings, and Y is asymptotically Poisson with parameter $\lambda = m^3/(4n)$.

A proof, by Janos Komlos, and detailed discussion of the test will appear elsewhere. The birthday spacings test addresses only the set of integers produced by a RNG, not the order in which they are produced. Congruential generators generally pass the test. Shift-register generators fail it, as do lagged-Fibonacci generators using $+, -$ or $\oplus$. Of the simple generators, only congruential and lagged Fibonacci using multiplication pass the birthday spacings test; most combination generators pass it.

12. RANKS OF RANDOM BINARY MATRICES.

Many Monte Carlo studies, particularly in combinatorics and graph theory, call for random incidence matrices, elements 0 or 1 to represent the absence or presence of some property. It is natural to let the rows of such a random matrix be formed by successive computer words, or portions of words, produced by a random number generator.

Shift-register or F(r,s,$\oplus$) generators are not suitable for such use. In order that the sequence of $1 \times n$ binary vectors $\beta, \beta T, \beta T^2, \ldots$ have a long period, it is necessary that the first n vectors in the sequence be linearly independent. Thus a binary matrix with rows formed by n or fewer successive vectors produced by a shift-register generator will always have full rank, while a truly random $m \times n$ binary matrix will have rank m with probability $(1-2^{-n})(1-2^{1-n})\cdots(1-2^{m-1-n})$, about .30 when $m = n \geq 10$ or so.

More specifically, the rank of a random $m \times n$ binary matrix takes the value $r = 1, 2, \ldots, \min(m, n)$ with probability

$$2^{r(n+m-r)-mn} \prod_{i=0}^{r-1} \frac{(1-2^{i-n})(1-2^{i-m})}{(1-2^{i-r})}.$$

If the rows of the $m \times n$ binary matrix are m consecutive computer words, or portions thereof, produced by a random number generator, then the rank of the matrix should have the distribution given by the probabilities above. Shift-register generators fail such tests, as do F(r,s,$\oplus$) generators. Congruential and F(r,s,$\pm$) usually pass.

13. CONCLUSIONS

The table below indicates how various kinds of RNG's perform in DIEHARD, a battery of stringent tests designed to test RNG's more thoroughly than standard tests do. It uses various overlapping m-tuple and OPSO tests to test for independence as well as uniformity, and tests all parts of a computer word, not only the most significant bits. Choice of starting values seems to make no difference in the tests, and results marked FAIL are spectacular failures—the observed distribution produced by the RNG is nowhere near that called for by probability theory for truly iid sequences.

The tests show that some of the worst RNG's are those that use exclusive-or: shift-register and F(r,s,$\oplus$), perhaps not surprising when one considers that a RNG is supposed to scramble the bits of a current computer word, or words, to get a new one, and $\oplus$, a no-carry add, does little to scramble bits, compared to $+, -$ or $*$.

The tests on lagged-Fibonacci generators used F(17,5,$\diamond$), F(31,13,$\diamond$) and F(55,24,$\diamond$). Using $\oplus$ gave almost uniformly bad results. The binary operations $+, -$ and $*$ gave good results, except on the birthday-spacings test. Note, however, that F(r,s,$\diamond$) generators with a very long lag r, say F(607,273,$\diamond$) or F(1279,418,$\diamond$), pass all tests for every choice of binary operation: $+, -, *$ and even $\oplus$. But also recall the gist of my previous comment: It is difficult to understand why anyone would use any F(r,s,$\oplus$) when the corresponding generator using $+$ or $-$ has at least as good statistical behavior and a period billions of times as long. Implementations have exactly the same programming structure, except that $\oplus$ is replaced by $-$ or $+$.

Combination generators do best in stringent tests. The generator COMBO returns $x_i - y_i \bmod 2^{32}$, with $x_n = x_{n-1} * x_{n-2} \bmod 2^{32}$, an F(2,1,$*$) generator on odd integers and $y_i = y_{i-3} - y_{i-1} \bmod 2^{30} - 35$, an F(3,1,$-$) generator. It passed all tests, as did NCOMBO, a similar combination except that $y_i = y_{i-3} - y_{i-1} \bmod 2^{32} - 5$.

These are two combination generators that use only a few computer instructions and have no arrays to access. Thus they are fast. Many other examples are, of course, possible. One should not get too carried away with the promise of combination generators provided by the above theory, however. In combining generators, it seems prudent to use sequences with incompatible algebraic structures, and with the combining binary operation on an incompatible structure as well. The generator Super-Duper, part of the McGill Random Number Package widely used at several hundred locations, combines the congruential generator $x_n = 69069x_{n-1} \bmod 2^{32}$ via $\oplus$ with the shift-register generator $\beta \leftarrow \beta(I + R^{15})(I + L^{17})$. It fails the MTUPLE test on substrings of low order bits, probably because both of the constituent parts do, and $\oplus$ is a poor binary operation for scrambling bits. Those who use Super-Duper may want to modify the assembly language instructions: replace $\oplus$ by $-$ in the step that combines the two simple generators.

Congruential generators with a prime modulus seem to do better on stringent tests than do those with modulus 2^n. While the latter are easier and faster for computer implementation, they give unsatisfactory results on substrings of low order bits. Multiplication modulo a prime p is difficult to implement in integer arithmetic. One of the best ways is to represent integers as double precision reals and use the DMOD function. In that case, the prime modulus $p = 2^{31} - 1$, used in a popular IBM generator, seems a poor choice. It gives only 31, rather than 32 bits, and $p - 1$ has too many factors. Better choices are, for example, $p = 2^{33} - 209$, which gives a full 32 bits, or $p = 2^{37} - 45$ which, with a multiplier of 19 or fewer bits, allows full and exact exploitation of double precision arithmetic. For the two latter choices, $(p - 1)/2$ is also prime, so that half of the residues of p are primitive elements and may be used as multipliers giving the full period.

Based on the above discussion, my current view of RNG's may be summarized with the following bottom line: Combination generators seem best; congruential generators are liked, but not well-liked; shift-register and lagged-Fibonacci generators using $\oplus$ are no good; avoid $\oplus$; lagged-Fibonacci generators using $+$ or $-$ pass most of the stringent tests except birthday spacings, and even those if the lag is long enough, say 1279; Lagged-Fibonacci generators using multiplication on odd integers mod 2^{32} pass all the tests; combination generators seem best—if the numbers are not random, they are at least higgledy piggledy.

A SUMMARY OF SOME TEST RESULTS

	LATTICE	PARKLOT	MTUPLE	OPSO	BDAY	OPERM	RUNS	RANK
Congruential	FAIL	pass	FAIL	FAIL	pass	pass	pass	pass
Shift-Register	pass	FAIL	FAIL	FAIL	FAIL	pass	FAIL	FAIL
Lagged-Fibonacci using $\oplus$	pass	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL
Lagged-Fibonacci using $+$ or$-$	pass	pass	pass	pass	FAIL	pass	pass	pass
Lagged-Fibonacci using $*$	pass	pass	pass	pass	pass	pass	pass	pass
Super-Duper	pass	pass	FAIL	pass	pass	pass	pass	pass
COMBO	pass	pass	pass	pass	pass	pass	pass	pass
NCOMBO	pass	pass	pass	pass	pass	pass	pass	pass

REFERENCES

[1] Arvillias, A.C. and Maritsas, A.E. Partitioning the period of m-sequences and applications to pseudorandom number generation. *J. ACM* **25**, 675-686, (1978).

[2] Bright, H. J. and Enison, R. L. Quasi-random number sequences from a long period TLP generator with remarks on applications to cryptography. *Computing Surveys* **11**, 357-370, (1979).

[3] Brown,M. and Solomon, H. On combining pseudorandom number generators, Technical Report No. 233, Dept. of Statistics, Stanford University. (1976).

[4] Fushimi, M. and Tezuka, 5. The k-distribution of generalized feedback shift register pseudorandom numbers. *Communications ACM* **26**, 516-523, (1983).

[5] Kendall, D. G. and Babington-Smith, B. Randomness and random sampling numbers, *J. Royal Statist. Soc.* **101** 146- 166,(1938). See also *J. Royal Statist. Soc. Supplement* **6** 51-36, (1939).

[6] Knuth, D. E. *The Art of Computer Programming, V2: Semi-numerical Algorithms*, 2nd Edition, Addison-Wesley, Reading, Mass., (1981).

[7] Lewis,T. G. and Payne, W. H. Generalized feedback shift register pseudorandom number algorithms. *Journal ACM* **20**, 456-468, (1973).

[8] MacLaren, M. D. and Marsaglia, G. Uniform random number generators, *J. ACM* **12**, 83-89 (1965).

[9] Marsaglia, G. Random numbers fall mainly in the planes. *Proceedings National Academy Science* **61**, 25-28 (1968).

[10] Marsaglia, G. The structure of linear congruential sequences, *Applications of Number Theory to Numerical Analysis.* Z. K. Zaremba, Ed., Academic Press, New York, (1972).

[11] Marsaglia. G. and L. H. Tsay, Matrices and the structure of random number sequences. *Linear Algebra and Its Applications* **67** 147-156, (1985)

[12] Marsaglia, G. and Bray, T. A. One-line random number generators and their use in combination, *Communications ACM* **11**, 757-759, (1968).

[13] Marshall, A. W. and Olkin, I. *Inequalities: Theory of Majorization and its Applications*, Academic Press, New York, (1979).

[14] Whittlesey, J. R. B. On the multidimensional uniformity of pseudorandom number generators. *Comm. ACM* **12** 247, (1969).