very next free flight are accumulated after each collision.

The Monte Carlo method has proven to be a powerful and useful tool. In :act, "solitaire games" now range from .he neutron- and photon-transport codes :hrough the evaluation of multi-dimen-sional integrals. the exploration of the properties of high-temperature plasmas, and into the quantum mechanics of sys-:ems too complex for other methods.

Quite a handful. ■

# Random-Number Generators

*by* ***Tony Warnock***

Random numbers have applications in many areas: simulation, game-playing, cryptography, statistical sampling, evaluation of multiple integrals, particle-transport calculations, and computations in statistical physics, to name a few. Since each application involves slightly different criteria for judging the "worthiness" of the random numbers generated, a variety of generators have been developed, each with its own set of advantages and disadvantages.

Depending on the application, three types of number sequences might prove adequate as the "random numbers." From a purist point of view, of course, a series of numbers generated by a truly random process is most desirable. This type of sequence is called a *random-number sequence,* and one of the key problems is deciding whether or not the generating process is, in fact, random. A more practical sequence is the *pseudo-random sequence,* a series of numbers generated by a deterministic process that is intended merely to imitate a random sequence but which, of course, does not rigorously obey such things as the laws of large numbers (see page 69). Finally, a *quasi-random sequence* is a series of numbers that makes no pretense at being random but that has important predefine statistical properties shared with random sequences.

## Physical Random-Number Generators

Games of chance are the classic examples of random processes, and the first inclination would be to use traditional gambling devices as random-number generators. Unfortunately, these devices are rather slow, especially since the typical computer application may require millions of numbers per second. Also, the numbers obtained from such devices are not always truly random: cards may be imperfectly shuffled, dice may not be true, wheels may not be balanced, and so forth. However, in the early 1950s the Rand Corporation constructed a million-digit table of random numbers using an electrical "roulette wheel." (The device had 32 slots, of which 12 were ignored; the others were numbered from O to 9 twice.)

Classical gambling devices appear random only because of our ignorance of initial conditions; in principle, these devices follow deterministic Newtonian physics. Another possibility for generating truly random numbers is to take advantage of the Heisenberg uncertainty principle and quantum effects, say by counting decays of a radioactive source or by tapping into electrical noise. Both of these methods have been used to generate random numbers for computers, but both suffer the defects of slowness and ill-defined distributions (however, on a different but better order of magnitude than gambling devices).

For instance, although each decay in a radioactive source may occur randomly and independently of other decays, it is not necessarily true that successive counts in the detector are independent of each other. The time it takes to reset the counter, for example, might depend on the previous count. Furthermore, the source itself constantly changes in time as the number of remaining radioactive particles decreases exponentially. Also, voltage drifts can introduce bias into the noise of electrical devices.

There are, of course, various tricks to overcome some of these disadvantages. One can partially compensate for the counter-reset problem by replacing the string of bits that represents a given count with a new number in which all of the original 1-1 and O-O pairs have been discarded and all of the original O-1 and 1-0 pairs have been changed to O and 1, respectively. This trick reduces the bias caused when the probability of a O is different from that of a 1 but does not completely eliminate nonindependence of successive counts.

A shortcoming of ***any*** physical generator is the lack of reproducibility. Reproducibility is needed for debugging codes that use the random numbers and for making correlated or anti-correlated computations. Of course, if one wants random numbers for a cryptographic one-time pad, reproducibility is the last attribute desired, and time can be traded for security. A radioactive source used with the bias-removal technique described above is probably sufficient.

## Arithmetical Pseudo-Random Generators

The most common method of generating pseudo-random numbers on the computer uses a recursive technique called the linear-congruential, or Lehmer, generator. The sequence is defined on the set of integers by the recursion formula

$$x_{n+1} = Ax_n + C \pmod{M},$$

where $x_n$ is the $n$th member of the sequence, and $A$, $C$, and $M$ are parameters that can be adjusted for convenience and to ensure the pseudo-random nature of the sequence. For example, $M$, the modulus, is frequently taken to be the word size on the computer, and $A$, the multiplier, is chosen to yield both a long period for the sequence and good statistical properties.

When $M$ is a power of 2, it has been shown that a suitable sequence can be generated if, among other things, $C$ is odd and $A$ satisfies $A = 5 \pmod 8$ (that is, $A - 5$ is a multiple of 8). A simple example of the generation of a 5-bit number sequence using these conditions would be to set $M = 32$ (5 bits), $A = 21$, $C = 1$, and $x_0 = 13$. This yields the sequence

$$13, 18, 27, 24, 25, 14, 7, 20, 5, 10 \ldots,$$

or, in binary,

$$01101, 10010, 11011, 11000, 11001, 01110, 00111, 10100, 00101, 01010, \ldots. \quad (1)$$

This type of generator has the interesting (or useful, or disastrous) property, illustrated by Seq. 1, that the least significant bit always has the alternating pattern 101010.... Further, the next bit has a pattern with period 4 (0110 above), the third bit has period 8, and so forth. Ultimately, the most significant bit has period $M$, which becomes the period of the sequence itself. Our example uses a short 5-bit word, which generates a sequence with a period of only 32. It is not unusual in many computer applications, however, to use many more bits (for example, to use a 32-bit word to generate a sequence with period $M = 2^{32}$).

One must be careful not to use such sequences in a problem with structures having powers of 2 in their dimensions. For example, a sequence with period $2^{32}$ would be a poor choice if the problem involved, say, a 3-dimensional lattice with sides of 128 (= $2^7$) because the structure of the sequence can then interact unfavorably with the structure of the problem. Furthermore, there would be only $2^{32}/(2^7)^3 = 2048$ possible states. The usual assumption in Monte Carlo computations is that one has used a "representative" sample of the total number of possible computations—a condition that is certainly not true for this example.

One method of improving a pseudo-random-number generator is to combine two or more unrelated generators. The length of the hybrid will be the least common multiple of the lengths of the constituent sequences. For example, we can use the theory of *normal numbers* to construct a sequence that has all the statistical features of a "truly random" sequence and then combine it with a linear-congruential sequence. This technique yields a hybrid possessing the strengths of both sequences—for example, one that retains the statistical features of the normal-number sequence.

We first construct a normal number, that is, a number in base $b$ for which each block of $K$ digits has limiting frequency $(1/b)^K$. A simple example in base 2 can be constructed by concatenating the sequence of integers

$$1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, \ldots$$

to form the normal number

$$11011100101110111100010011010101111001101111101111 \ldots .$$

If the number is blocked into 5-digit sets

$$11011, 10010, 11101, 11100, 01001, 10101, 01111, 00110, 11110, 11111, \ldots, \qquad (2)$$

it becomes a sequence of numbers in base 2 that satisfy all linear statistical conditions for randomness. For example, the frequency of a specific 5-bit number is $(1/2)^5$.

Sequences of this type do not "appear" random when examined; it is easy for a person to guess the rule of formation. However, we can further disguise the sequence by combining it with the linear-congruence sequence generated earlier (Seq. 1). We do this by performing an *exclusive-or* (XOR) operation on the two sequences:

$$01101, 10010, 11011, 11000, 11001, 01110, 00111, 10100, 00101, 01010, \ldots \quad (1)$$

and

$$11011, 10010, 11101, 11100, 01001, 10101, 01111, 00110, 11110, 11111, \ldots \quad (2)$$

yield

$$10110, 00000, 00110, 00100, 10000, 11011, 01000, 10010, 11011, 10101, \ldots. \quad (3)$$

Of course, if Seq. 3 is carried out to many places, a pattern in it will also become apparent. To eliminate the new pattern, the sequence can be XOR'ed with a third pseudo-random sequence of another type, and so on.

This type of hybrid sequence is easy to generate on a binary computer. Although for most computations one does not have to go to such pains, the technique is especially attractive for constructing "canonical" generators of apparently random numbers.

A key idea here is to take the notion of randomness to mean simply that the sequence can pass a given set of statistical tests. In a sequence based on normal numbers, each term will depend nonlinearly on the previous terms. As a result, there are nonlinear statistical tests that can show the sequence not to be random. In particular, a test based on the transformations used to construct the sequence itself will fail. But, the sequence will pass all ***linear*** statistical tests, and, on that level, it can be considered to be random.

What types of linear statistical tests are applied to pseudo-random numbers? Traditionally, sequences are tested for uniformity of distribution of single elements, pairs, triples, and so forth. Other tests may be performed depending on the type of problem for which the sequence will be used. For example, just as the correlation between two sequences can be tested, the auto-correlation of a single sequence can be tested after displacing the original sequence by various amounts. Or the number of different types of "runs" can be checked against the known statistics for runs. An increasing run, for example, consists of a sequential string of increasing numbers from the generator (such as, 0.08, 0.21, 0.55, 0.58, 0.73, . . .). The waiting times for various events (such as the generation of a number in each of the five intervals (0, 0,2), (0.2, 0,4), ..., (0.8, 1)) may be tallied and, again, checked against the known statistics for random-number sequences.

If a generator of pseudo-random numbers passes these tests, it is deemed to be a "good" generator, otherwise it is "bad." Calling these criteria "tests of randomness" is misleading because one is testing a hypothesis known to be false. The usefulness of the tests lies in their similarity to the problems that need to be solved using the stream of pseudo-random numbers. If the generator fails one of the simple tests, it will surely not perform reliably for the real problem. (Passing all such tests may not, however, be enough to make a generator work for a given problem, but it makes the programmers setting up the generator feel better.)

## Quasi-Random Numbers

For some applications, such as evaluating integrals numerically, the use of quasi-random sequences is much more efficient than the use of either random or pseudo-random sequences. Although quasi-random sequences do not necessarily mimic a random sequence, they can be tailored to satisfy the equi-distribution criteria needed for the integration. By this I mean, roughly speaking, that the numbers are spread throughout the region of interest in a much more uniform manner than a random or pseudo-random sequence.

For example, say one needs to find the average of the quantity $f(x)$ over the set of coordinates $x$, knowing the distribution of coordinate values $\rho(x)$ for the system being considered. Ordinarily, the average is given by the expression

$$\langle f \rangle = \frac{\int \rho(x) f(x)\, dx}{\int \rho(x)\, dx}.$$

Rather than evaluating this integral, however, one can evaluate $f(x)$ at a series of random points. If the probability of picking a particular point $x$ is proportional to the statistical weight $\rho(x)$, then $\langle f \rangle$ is given by the expression

$$\langle f \rangle = \sum_{i=1}^{N} f(x_i)/N,$$

where $N$ is the total number of points chosen. This idea is the basis of the Metropolis technique of evaluating integrals by the Monte Carlo method,

Now if the points are taken from a random or a psuedo-random sequence, the statistical uncertainty will be proportional to 1/@. However, if a quasi-random sequence is **used,** the points will occupy the coordinate space with the correct distribution but in a more uniform manner, and the statistical uncertainty will be proportional to l/N. In other words, the uncertainty will decrease much faster with a quasi-random sequence than with a random or pseudo-random sequence.

How are quasi-random sequences generated? One type of sequence with a very uniform distribution is based on the radical-inverse function. The radical-inverse function $\phi(N,b)$ of a number $N$ with base $b$ is constructed by

1. writing the number in base $b$ (for example, 14 in base 3 is 112);
2. reversing the digits (112 becomes 211); and
3. writing the result as a fraction less than 1 in base $b$ (211 becomes 211/1000 in base 3 and, thus, $\phi(14,3) = .211$).

A sequence based on the radical-inverse function is generated by choosing a prime number as the base $b$ and finding $\phi(1,b), \phi(2,b), \phi(3,b), \phi(4,b), \ldots.$ For a problem with $k$ dimensions, the first $k$ primes are used, and $(\phi(N,b_1), \phi(N,b_2), \ldots \phi(N,b_k))$ becomes the $N$th point of the $k$-dimensional sequence. This sequence has a very uniform distribution and is useful in mutiple integration or multi-dimensional sampling.

There are many other types of random, pseudo-random, or quasi-random sequences than the ones I have discussed here, and there is much research aimed at generating sequences with the properties appropriate to the desired application. However, the examples I have discussed should illustrate both the approaches being taken and the obstacles that must be overcome in the quest of suitable “random” numbers. ■