Random thoughts

10^{435} years is a very long time

by Cleve Moler

"Nothing in nature is random... A thing appears random only through the incompleteness of our knowledge."—Spinoza

Do you recognize this number?

0.21895918632809

If you're an avid MATLAB user, you've probably seen this number before, but don't remember it. It's the first number produced by the MATLAB random number generator with its default settings. Start up a fresh MATLAB session, set `format long`, type `rand`, and that's the number you get.

So, if all MATLAB users, all around the world, on all different computers, keep getting this same number, is it really "random"? No, it isn't. Computers are (in principle) deterministic machines and should not exhibit random behavior. If your computer doesn't access some external device, such as a gamma ray counter or a clock, then it must really be computing *pseudorandom* numbers. My favorite definition was given in 1951 by Berkeley Professor D. H. Lehmer, a pioneer in computing and, especially, computational number theory:

> *A random sequence is a vague notion...in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests traditional with statisticians...*

Lehmer also invented the *multiplicative congruential* algorithm, which is the basis for many of the random number generators in use today. Lehmer's generators involve three integer parameters, a, c, and m, and an initial value, x_1, called the *seed*. A sequence of integers is defined by

$$x_{k+1} = (a\, x_k + c) \bmod m$$

(The operation "mod m" means take the remainder after division by m.) For example, with $a = 13$, $c = 0$, $m = 31$, and $x_1 = 1$, the sequence begins with

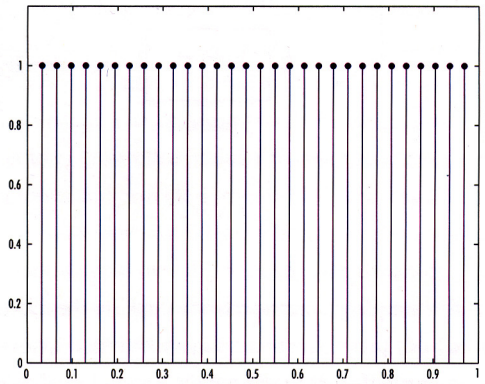
1 13 14 27 10 6 16 22 7 29 5 3

What's the next value? Well, it looks pretty unpredictable, but you've been initiated. So you can compute 13·3 mod 31, which is 8. The first 30 terms in the sequence are a permutation of the integers from 1 to 30 and then the sequence repeats itself. It has a *period* equal to m-1.

A pseudorandom integer sequence with values between 0 and m can be scaled by dividing by m to give floating-point numbers uniformly distributed in the interval [0, 1]. Our simple example begins with

0.0323 0.4194 0.4516 0.8710 0.3226 0.1935 0.5161...

The histogram of this sequence is:

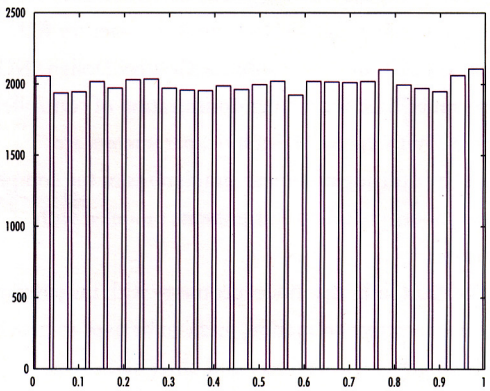


There are only a finite number of values—30 in this case. The smallest value is 1/31; the largest is 30/31. Each number is equally probable in a long run of the sequence.

The uniform random number function, `rand`, in the current release of MATLAB, has similar behavior. It is a multiplicative congruential generator with parameters

$a = 7^5 = 16807$
$c = 0$
$m = 2^{31}\text{-}1 = 2147483647$

These values were recommended in a 1988 paper by S. K. Park and K. W. Miller, "Random number generators: Good ones are hard to find" (*Comm ACM*, vol. 32). Here is a histogram of 50,000 values produced by rand.



Each of the 25 bins contains roughly 2,000 numbers. We would see a similar picture for any other reasonable number of bins. Our generator satisfactorily passes this histogram test, which is the admission exam for uniform generators.

Like our toy generator, `rand` generates all real numbers of the form k/m for $k = 1 \ldots m\text{-}1$. The smallest and largest are 0.00000000046566 and 0.99999999953434. It repeats itself after generating m-1 values, which is a little over two billion numbers. A few years ago that was regarded as plenty. It probably still is today, but it's getting a little skimpy. On a 75 MHz Pentium laptop, we can exhaust the period in fewer than four hours. Of course, to do anything useful with two billion numbers takes more time, but we would still like to have a longer period.

We've developed a replacement for our current `rand`. It will be part of MATLAB version 5. The new algorithm is based on advice from George Marsaglia, a professor at Florida State University, and author of the classic analysis of random number generators, "Random numbers fall mainly in the planes," (*Proc. Nat. Acad. Sci.*, vol 61, 1968).

Marsaglia's new generator does not use Lehmer's multiplicative congruential scheme. In fact, there are no multiplications or divisions at all. It is specifically designed to produce floating-point values. The results are not just scaled integers. And, it is *fast*. We get close to a "megarand"—a million random numbers per second—on our laptop.

In place of a single seed, the new generator has 35 words of internal memory or *state*. Thirty two of these words form a cache of floating-point numbers, z, between 0 and 1. The remaining three words contain an integer index i, which varies between 1 and 32, a single random integer j, and a "borrow" flag b. This entire state vector is built up a bit at a time during an initialization phase. Different initial states can be triggered by specifying different values of j.

The generation of the i-th floating-point number in the sequence involves a "subtract with borrow" step, where one number in the cache is replaced by the difference of two others.

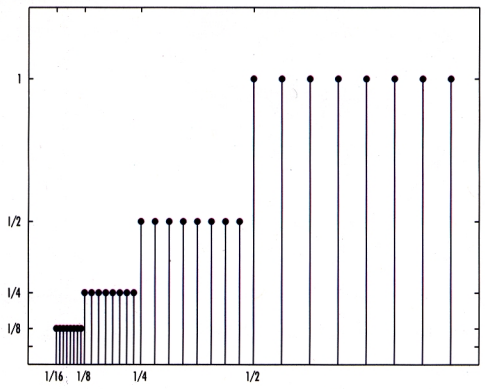
$$z_i = z_{i+20} - z_{i+5} - b$$

The three indices, i, i+20, and i+5 are all interpreted mod 32 (by using just their last five bits). The quantity b is left over from the previous step; it is either zero or a small positive value. If the computed z_i is positive, b is set to zero for the next step. But if the computed z_i would be negative, it is made positive by adding 1.0 before it is saved and b is set to 2^{-53} for the next step. The quantity 2^{-53}, which is half of MATLAB's built-in constant `eps`, is called one *ulp* because it is one unit in the last place for floating-point numbers slightly less than 1.

By itself, this generator would be almost completely satisfactory. Marsaglia has shown that it has a huge period—almost 2^{1430} values would be generated before it would repeat itself. But it has one slight defect. All the numbers are the result of floating-point additions and subtractions of numbers in the initial cache, so they are all integer multiples of 2^{-53}. Consequently, many of the floating-point numbers in [0,1] are not represented.

The floating-point numbers between 1/2 and 1 are equally spaced with a spacing of one ulp, and our subtract-with-borrow generator will eventually generate all of them. But numbers less than 1/2 are more closely spaced and the generator would miss most of them. It would generate only half of the possible numbers in the interval [1/4,1/2], only a quarter of the numbers in [1/8,1/4], and so on. This is where the quantity j in the state vector comes in. It is the result of a separate, independent, random number generator based on bitwise logical operations. The floating-point fraction of each z_i is xored with j to produce the result returned by the generator. This breaks up the even spacing of the numbers less than 1/2. It is now theoretically possible to generate *all* of the floating-point numbers between 2^{-53} and $1\text{-}2^{-53}$. We're not sure whether all of them are actually generated, but we don't know of any that can't be.

This graph illustrates what we're trying to accomplish, with one ulp equal to 2^{-4} instead of 2^{-53}.



The graph depicts the relative frequency of each of the floating-point numbers. A total of 32 floating-point numbers are shown. Eight of them are between 1/2 and 1 and they are all equally likely to occur. There are also eight numbers between 1/4 and 1/2, but since this interval is only half as wide, each of them should occur only half as often. As we move to the left, each subinterval is half as wide as the previous one, but it still contains the same number of floating-point numbers, so their relative frequencies must be cut in half. Imagine this picture with 2^{53} numbers in each of 2^{52} successively smaller intervals and you will see what our new random number generator is doing.

With this additional bit fiddling, the period becomes something like 2^{1492}. Maybe we should call it the Christopher Columbus generator. In any case, it will be a *very* long time before it repeats itself. At one million per second, it will take more than 10^{435} years. ■

Cleve Moler is Chairman and co-founder of The MathWorks. His e-mail address is moler@mathworks.com.