;login:

THE MAGAZINE OF USENIX & SAGE
February 2001 • volume 26 • number 1

inside:

RAY SWARTZ:
A NEW TWIST ON RANDOM
NUMBER GENERATORS

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

a new twist on random number generators

by Ray Swartz

Ray Swartz has been fascinated with computer simulation since learning about it in graduate school. Since then, Ray has created computer models of copper mine development, ink-jet printers, financial planning, and betting strategies.

<ray@trainingonline.net>

It is surprising how frequently I need to use random numbers in my programs. First, I like to write games of chance involving dice or cards. Second, I am often hired to write Monte Carlo simulation models, which make heavy use of random numbers to select values from probability distributions. What's more, I often find random numbers useful for testing my code.

When I first started using random numbers, I wondered, how could a number generated by a deterministic computer program be considered "random" in any sense of the word? Wouldn't numbers generated by a computer program show some obvious pattern?

Well, yes and no.

Some Random History

In 1946, John Von Neumann suggested generating random numbers by squaring the previous number (usually called the "seed") and extracting its middle digits. After some research, it was discovered that this "middle-square method" cycled (produced the same run of numbers) fairly quickly and that the longest period (the length of values between repeated numbers) was 142 [Knuth, *The Art of Computer Programming*, vol. 2, p. 4].

Knuth [p. 5] demonstrates that "random" algorithms don't necessarily generate random numbers. A 13-step "Super-random" generator invented by Knuth cycled when it hit a number that was "magically" transformed into itself by the algorithm.

Knuth concludes, "random numbers should not be generated with a method chosen at random."

Linear Congruential Generators

The most common way to generate random numbers today is by using a linear congruential generator (LCG). LCGs are of the form

*(a * seed + c) % modulus*

where *seed* refers to the last number generated. If the *modulus* is the word size of the machine and the calculation is done as an unsigned integer, we can reduce this to

*a * seed + c*

A particular LCG produces a set sequence of numbers. That is, a specific pair of *a* and *c* values generates the same set of numbers in the same order every time. What makes this useful is that if you choose *a* and *c* carefully, the sequence is long, exhibits "random" characteristics, and is repeatable (something a truly random phenomenon isn't!). LCGs also generate their results quickly.

In summary, the rules for selecting *a* and *c* as to maximize the LCG's period are:

1. if the modulus is a power of 2, pick *a* so that *a % 8 = 5*;
2. *a* should be between *0.01 * modulus* and *0.99 * modulus*;
3. the value of *c* must have no factor in common with the modulus. [Knuth, p. 184]

Note that LCGs with a maximum period are not necessarily "random." Consider the sequence: 1, 2, 3, 4, 5, ... (e.g., $a = 1$ and $c = 1$). This sequence has a maximum period but exhibits very little randomness!

A Few Random Comments

In choosing values for a and c, how can we distinguish a "good" (i.e., random) LCG from a "bad" one (i.e., 1, 2, 3, 4, 5, ...)? Therein lies the rub! What makes a sequence of numbers random enough?

Over time, several tests have been devised to check the randomness of the generated numbers. Many tests are statistical in nature like the chi-squared test (which determines the likelihood of an observed result compared to an expected result) and the mean test (checking the mean of generated numbers between 0 and 1 – it should be very close to 0.5).

The most important randomness test for LCGs is the spectral test, which tests sequences of numbers for patterns in n-dimensional space. Knuth values the spectral test: "Not only do all good generators pass this test, all generators now known to be bad actually fail it" [p. 93]. He also provides a table of results for the spectral test of 29 LCGs [p. 106]. A rigorous suite of tests, known as the "Die Hard" tests, was written by Professor Marsaglia of Florida State University. Professor Marsaglia can be reached at <*geo@stat.fsu.edu*>.

Why should you care about how random numbers are generated? Why not just use whatever the rand() function gives you? The answers are: (1) system random number generators have proven to be unreliable in the past and (2) supplied random number generators may be slow or have too small a period for the task at hand.

My first experience with computer simulation was using the random function built into a PDP-11. For 1,000 coin flips, my program reported that heads came up 65% of the time! More coin flips didn't change the outcome. Since then, I have always written my own (well, actually, it was one recommended by Knuth) LCGs to generate random numbers for my programs.

This is not to say that all random number generators supplied by rand() functions are bad, only that you should be wary of using any random number generator for serious work without first learning its pedigree and effectiveness.

Generating Limits

One problem that arises when using LCGs is that even the best ones cycle, this being the very nature of LCGs. Knuth recommends pulling no more than modulus/1,000 values from an LCG. This isn't a problem if you are testing a program that requires only a few thousand random data points. However, a large, complex simulation might require a million random numbers or more to produce meaningful results.

Recently, I faced precisely this problem. I was hired to create a detailed model for a piece of computer hardware – a model with a huge appetite for random numbers. What's more, the client wanted to be able to run the model on standard PCs (whose word sizes are 32 bits).

If your modulus is 32 bits long (as is the case with most PC and UNIX random number generators), then modulus/1,000 is only about 4 million random numbers, not nearly enough! Not only would my model run up against this limit, but after a few

You should be wary of using any random number generator for serious work without first learning its pedigree and effectiveness.

If you have a need for a reliable random number generator, I highly recommend you check out the Mersenne Twister.

runs, I might have to trash my trusted LCG and find another one. Picking good LCGs is not all that simple and a bad one often produces meaningless results.

The newer random() generators from Earl T. Cohen (which are not LCGs), with their dramatically larger state space and non-linear additive feedback methods, can give a long enough sequence (2**69), but they are only available on UNIX systems.

What was I going to do? Moving the model to a machine with a bigger word size was not an option, in this case.

Random Numbers with a Twist

I discovered a completely different solution: a Web search pointed me to the home page of the Mersenne Twister (MT), a random number generator developed by Makoto Matsumoto and Takuji Nishimura of Keio University in Japan.

The basic idea is quite simple. Instead of generating numbers by manipulating a single seed, the MT generates numbers by "twisting" the bits in 623 seeds.

Here is how it works: first, using a traditional LCG, generate 623 values. When these have been used up, create a new set of 623 values by mixing the bits of two consecutive numbers. When this new set has been used, repeat the mixing procedure to get another 623.

Here is a simplified example in base 10 using three numbers. First, we generate three random numbers:

123
221
332

To create a new set of values in this example, take the first digit of one value and combine it with the last two digits of the next value. This results in:

121
232
323

where "next" for the final value means "wrap back to the first number."

Matsumoto and Nishimura have proven that the period of the MT is $2^{(19937)}$-1, which is around $10^{6,000}$ (The generator is named for the length of its cycle, which is a Mersenne prime.) The period of MT is so large that it can't be fully generated by today's computers (there are approximately 2^{80} microseconds in 10 billion years)!

The MT is relatively new (1998) but has been in general use since it was unveiled. The MT has been extensively tested and passed all current tests, including the spectral test and the Die Hard suite. For more information, see the MT home page (<*http://www.math.keio.ac.jp/~matumoto/emt.html*>), which contains links to MT implementations in many programming languages, scientific papers, and other news. The MT code is freely available. Dr. Matsumoto only asks that you send him email if you choose to use the MT.

I've been using the MT for all my random number needs for the past six months and can say that it runs fast, passes every test I've tried on it, and frees me from worrying about the number of values I pull from it. If you have a need for a reliable random number generator, I highly recommend you check out the Mersenne Twister.