Jagannath Pisharath Newer Math

Fall 2003

Linear Congruential Number Generators

A useful, if not important, ability of modern computers is random number generation. Without this ability, if you wanted to, for example, draw a lot of a thousand numbers, you'd have to erect a rather large wheel with a thousand partitions and spin, spin, spin. Thus, computerized random number generators (RNG) make our lives easier; or so one would think. It is easier said than done to create an RNG that performs satisfactorily. This is because computerized random number generators do not generate random numbers at all. In fact, the numbers created are called pseudo-random in that they can be predicted as long as certain attributes and parameters are known.

This discussion will focus on a particular type of RNG, namely the *Linear Congruential Number Generator* (LCG). The LCG is perhaps the most commonly used RNG in modern computer applications, but strangely, it was invented by D.H. Lehmer in a time when his concept had almost no practical use, as there were essentially no computers around [1]. It was only later on, when programmers required a fast way to generate a large stream of seemingly random numbers that Lehmer's LCG method was used. These early programmers were more interested in speed than statistical randomness, and thus many, if not all, of the early LCGs were horrendously flawed.

Lehmer's LCG involves modulus math. Since all calculations done within a modulus limit the output by the size of the modulus, modulus math is generally faster than its traditional counterpart. Furthermore, a modulus allows there to be a cycle of

numbers, which is essential in generating random numbers because it provides a fixed range of output. This known range of output allows random output to be ‘uniformized’ into numbers between zero and one. For example, generating random numbers with a modulus of fifteen means that any digit that is outputted can be divided by fifteen to convert it to a uniformed number; that is, dividing by a modulus allows an easy conversion to the accepted format of expressing random numbers.

The basic form of an LCG is [2]:

$$I_n = (a \cdot I_{n-1} + c) \bmod m$$

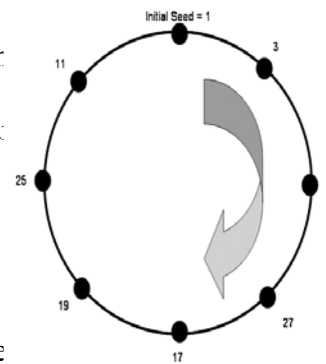
where A, M, and C are pre-selected constants. A is the *multiplier*; it multiplies the previous value of the equation, I_{n-1}, times itself. C is called the *increment*, and typically this is set to zero. Cases where $C \neq 0$ will be discussed later. Finally M, the *modulus* is arguably the most important number of the three. As said before, M dictates the upper limit of output from this recursive function.

The process of generating random numbers with an LCG is initiated by another pre-selected constant, called the *seed*. The *seed* is in fact I_1, the first number in the LCG output stream. LCGs and their parameters can be written in function notation such as LCG (A,C,M,*seed*). For example, the LCG (3,0,10,1) would be initiated by performing the calculation $I_1 = (3 \cdot 1 + 0) \bmod 10$, which equals 3. Then the next iteration is performed, with I_{n-1} being 3 instead of 1. Thus the LCG (3,0,10,1) would yield an output of 1,3,9,7… Regarding the seed, S.L Anderson recommends using another pseudo-RNG based upon the current year, month, day, hour, minute, and second to ensure maximum randomness, but it should be noted that it is sometimes useful to use the same seed multiple times for testing purposes [3].

The selection of an LCG's parameters A, C, and M can make or break the generator in terms of its randomness and by extension, usability. Since the use of modulus math in LCGs creates a fixed range of output, it is very easy to use up and exhaust the numbers within the period and end up with an LCG stream that repeats. For example, the LCG below has a definite problem:

LCG (3, 0, 32, 1) = 1, 3, 9, 27, 17, 19, 25, 11, 1, 3, 9…

The numbers start to repeat after the eighth iteration. In LCG jargon, this is called a *period* [1]. This LCG has a period of eight, and if called upon to deliver any more than this amount of numbers, it would start to simply repeat these eight numbers over and over again.



Some LCGs have what's known as a *full period* [1]. That is, the LCG output stream visits every number within the modulus before repeating. If the above example had visited every number from 1 to 31, it would be a full period LCG. Thus, a full period LCG has a period of m-1. The key to a successful LCG is to employ a full period that is very large (at least a few million) so that it is not exhausted by a reasonable request for random numbers.

Apart from worrying about the period, when creating an LCG, one has to be mindful that the modulus and multiplier (M and A) are relative primes (that is, their greatest common denominator is one). This is a crucial requirement. Take for example the below LCG. A glimpse at the output stream of the LCG shows a fatal flaw:

LCG (2, 0, 4, 1) = 1, 2, 0, 0, 0…

The reason for this collapse is the fact that the modulus is divisible by its multiplier and thus, they are not relatively prime. The easiest way to ensure that the two numbers are indeed relatively prime is to just set the modulus as a prime number [3].

As mentioned before, the first LCGs were optimized for speed, not randomness. Thus, their parameters were often numbers which computers could process quickly. Many of the early LCGs used moduli that were a power of two (for example, 2^{12}). Calculations with these numbers can be done efficiently by computers, but alas, they are non-prime integers, and thus prone to problems. Regarding the ubiquity of flawed LCGs, Donald Knuth advises "…look at the [random number] subroutine library of each computer installation in your organization… Try to avoid being too shocked at what you find. [1]"

Now that the modulus (M) and multiplier (A) have been analyzed, the increment (C) can be discussed. As said before, the increment is almost always set to zero. However, LCGs where $C \neq 0$ are called *Mixed Linear Congruential Number Generators* (MLCG). The advantage of MLCGs is that they can be employed to satisfy the condition when $I_{n-1} = 0$, $I_n \neq 0$. This means that the LCG stream can have an output of zero without collapse, and consequently, a full period MLCG has a period that is a length of M rather than M-1. No real advantage has been found to using MLCGs over LCGs, and thus, they are not very prominently used today [4].

Knuth's aforementioned statement accurately reflects the magnitude of the 'badness' of the first LCGs, which were ostensibly used for scientific and professional use. But what constitutes a bad LCG? Park and Miller suggest using a three-test system to ascertain the quality of an LCG [1]:

Test 1: Does the LCG have a full period?

A LCG without a full period would likely exhaust its period within a few iterations, no matter how big the modulus is.

Test 2: Is the output random?

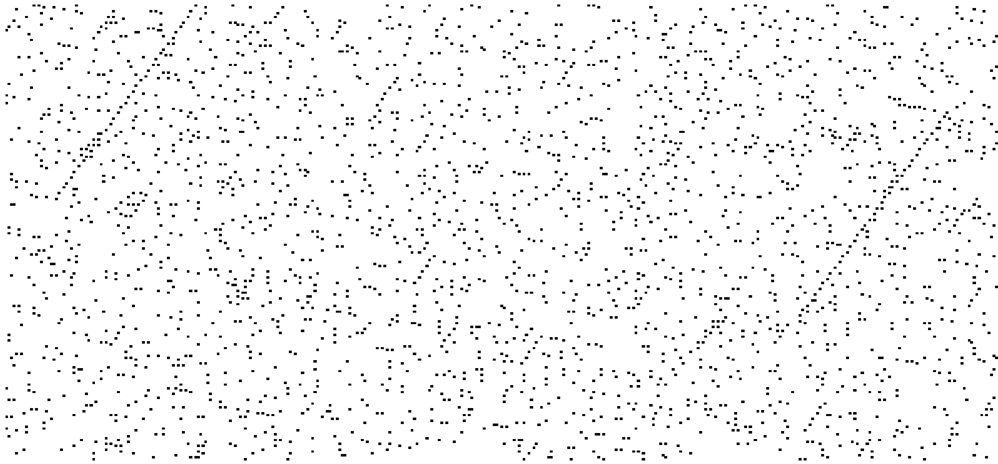
This requisite is rather reasonable, if not obvious.

Test 3: Can the LCG be implemented efficiently with a 32-bit computing architecture

This particular test, though important when it is time to actually implement an LCG, is not too relevant to the scope of this discussion.

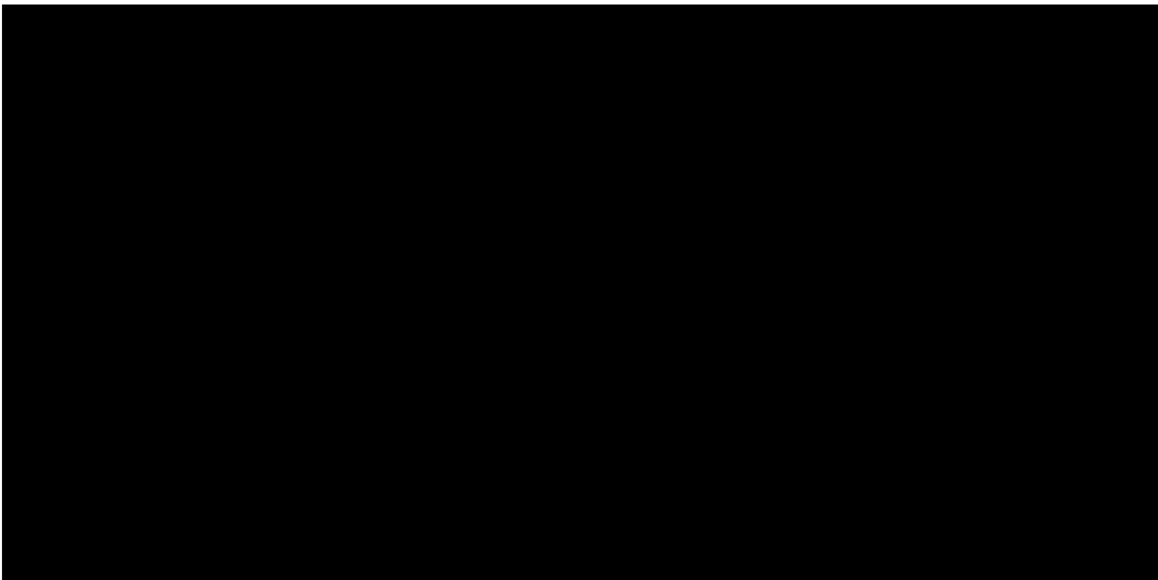
Out of these three tests, test two is the one that most of the early LCGs have failed. There is a gamut of statistical trials that can be performed, like spectral and chi-squared tests, to assess an output stream's randomness. Here, we will focus only on the subjective two and three dimensional dot-plot tests of LCG output to determine randomness. (A one dimensional dot-plot test is omitted because I have not found it very useful in giving significant feedback regarding the quality of a given LCG stream.) These dot-plots are constructed using successive pairs of numbers derived from the LCG being tested. That is, the ordered pairs of a 2-d dot-plot of a hypothetical LCG would consist of the points $(I_1,I_2),(I_3,I_4)$..., while the same LCG represented in a 3-d dot-plot would consist of the points $(I_1,I_2,I_3),(I_4,I_5,I_6)$...

Here is an example LCG (1277, 0, 131072, 1) to which the 2-d dot-plot test has been applied to with 5,000 trials [5]:



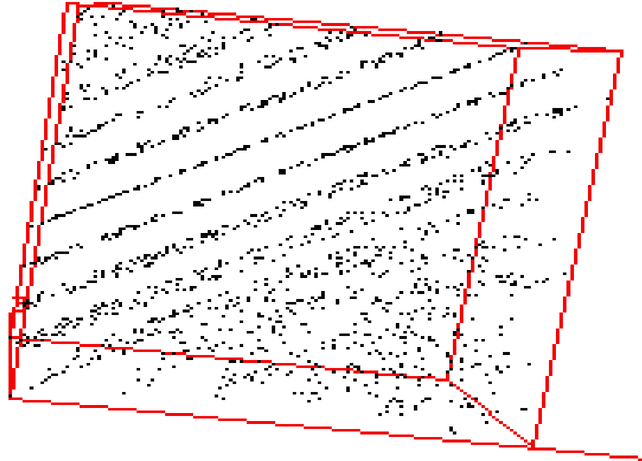
It is easy to see the emerging linear bands in this graph. The modulus of this LCG is even, and the period is only 32,769 (that is, it is not a full period). Trying to derive more than 32,769 random numbers with this LCG will result in repetition.

The next example is the infamous IBM RANDU LCG (65539, 0, 2147483648, 123456789), which was made by the company for its 360 System computers in the '60's [6]. A 2-d test with 5,000 trials shows no apparent problems [5]:



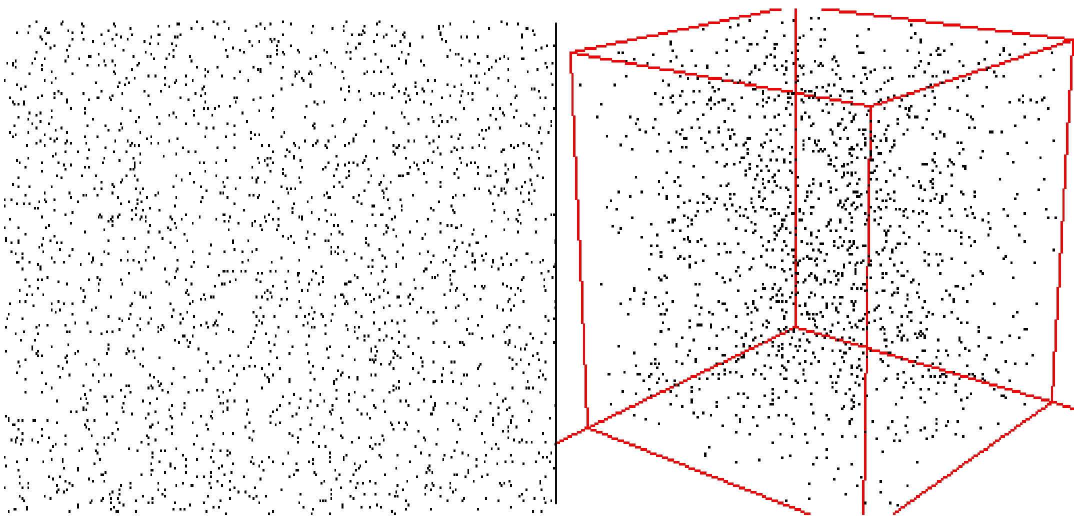
And indeed, this is perhaps why RANDU was put into use on a wide scale basis. It is

only until a 3-d plot is made of the LCG at 5,000 trials that problems become visible [5]:



The modulus of RANDU was a power of two (2^{31}), which in hindsight was a mistake.

In response to these flawed LCGs, Park and Miller have devised a better one, the Minimal Standard LCG. Their message was that this LCG should be used as a benchmark in terms of randomness. That is, if one were to create an LCG, it should only be used if it can be proven to be 'more random' than the Minimal Standard LCG. The LCG uses a prime modulus of 2^{31}-1 and a multiplier of 16807. A 2-d test with 5,000 trials and a 3-d test with the same number of trials show no apparent randomness:



Minimal Standard LCG (16807, 0, 231-1, 1 1) [5]

As a side note, here is the LCG that Maple uses:

LCG(427,419,669,081 999,999,999,989 0) [6]

One would hope that since Maple is a relatively new program, that it would only use its own LCG if it were better than the Minimal Standard one. Another LCG that has been developed lately is the Mersenne Twister LCG algorithm, which was invented in 1997 by Matsumoto and Nishimura. It has a period of 2^{19937}-1, and is “proven to be equidistributed in 623 dimensions (for 32-bit values), and runs faster than all but the least statistically desirable generators. It is now becoming increasingly accepted as the random number generator of choice for all statistical simulations and generative modeling. [7]”

LCGs, especially newer versions such as the Minimal Standard, can produce sets of numbers that can pass any statistical test of randomness, but it should be kept in mind that there is nothing random about LCG output. Von Neumann observes, “Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. [7]" Indeed, given the parameters of the algorithm, any LCG sequence can be repeated. Furthermore, one has to keep in mind the period of the LCG before implementing it. For example, the Minimal Standard LCG, with its period of about two billion, would be ill-equipped to handle a simulation where every person on earth is assigned a random number. And even if a simulation would not exhaust the period of an LCG, there is no guarantee that any output from an LCG, even the Minimal Standard, is statistically random. That is, there may be unknown flaws in them. Thus it is probably unwise to rely upon LCGs in certain situations where randomness is a top priority.

If LCG output can indeed be repeated, and thus predicted, one has to wonder about cases where the unpredictability of the RNG is paramount. Case in point: It would

be disconcerting to know that slot-machine manufacturers could supply the information of when their machines would pay out to casino owners. In fact, it turns out that slot-machines do indeed use what seems to be an LCG to determine when a player has hit the jackpot. When the slot-machine starts to spin, its internal LCG spits out thousands of numbers. Then, when the player pulls the handle, the number that was in memory at the precise time of the handle-pull is selected to determine whether or not the player has won. Thus, slot-machines use a mechanical-LCG that is more or less unpredictable.

Even with their deficiencies, LCGs are a good way to get large streams of pseudo-random numbers for most situations. As long as one is aware of an LCG's shortcomings and capabilities, they can be used with confidence.

References:

1 Stephen K. Park and Keith W. Miller
Random Number Generators: Good Ones Are Hard To Find
Communications of the ACM, **31**(10):1192-1201, 1988.

2 Linear Congruential Number Generators
http://www.taygeta.com/rwalks/node1.html

3 Random Number Generators
http://csep1.phy.ornl.gov/rn/rn.html

4 Generating Random Numbers
http://www.brpreiss.com/books/opus4/html/page472.html

5 Random Number Generator [2-d and 3-d dot-plots]
http://www.cs.pitt.edu/~kirk/cs1501/animations/Random.html

6 [Strangely, the site is inaccessible, but it was working fine a month ago!]
http://crypto.mat.sbg.ac.at/results/karl/server/server.html

7 Wikipedia: Pseudorandom Number Generator
http://en.wikipedia.org/wiki/Pseudorandom_number_generator

8 Howstuffworks “How Slot Machines Work”
http://howstuffworks.lycoszone.com/slot-machine3.htm