What is a Linear Congruential Random Number Generator?

Many computer applications rely on random number generation. For example, if you want to write a program to simulate a poker game, you don't want each player to get the same cards every hand. Since some programs require a large number of random numbers, we can greatly speed up the program by using a faster, more efficient random number generator. The method of this random number generation by *linear congruential method*, works by computing each successive random number from the previous. Starting with a seed, Xo, the linear congruential method uses the following formula:

Xi+1 = (A*Xi + C) mod M

In his book, *The Art of Computer Programming*, Donald Knuth presents several rules for maximizing the length of time before the random number generator comes up with the same value as the seed. This is desirable because once the random number generator comes up with the initial seed, it will start to repeat the same sequence of random numbers (which will not be so random since the second time around since we can predict what they will be). According to Knuth's rules, if M is prime, we can let C be 0 and he suggests that this variant of the line

THEOREM : (By Greenberger in 1961)

The LCG defined above has full period . if and only if the following conditions are satisfied

a) m and c are relatively prime

b) If q is a prime number that divides m , then q divides a-1

c) If 4 divides m, then 4 divides a-1

The LCG tend to behave differently for c>0 and c=0

A linear congruential generator *lnc( )* generates a sequences of integers

$$U_0, U_1, \text{����.}, U_k$$

that are restricted to the range *0* to m. On each call to *lnc(*), you must

give it as argument the previous number in the sequence. It returns the following by

$$U_{k+1} = (a\,U_k + b) \bmod (m+1).$$

where a, *b,* and *m* can be chosen to be any positive integers. (The operation mod *n p* is the integer remainder when the integer *n* is divided by the integer *p.* This remainder must be between 0 and *p-1,* inclusive.) Once you�ve chosen these three constants and the starting value 0 *U* you�ve completely defined the sequence that *Inc( )* will produce. If you set a, *b,* and *m* to reasonably large values (there are rules of thumb to follow in choosing these values so as the maximize the apparent disorder in the sequence: see Knuth, 1971), the resulting sequence looks satisfyingly random. Of course, you want numbers between 0 and 1, not integers between 0 and *m.* But you need only divide 1 *k U* + by *m+2* to map the output of *Inc* into the desired range.

It�s easier to see what *Inc( )* is doing if we pick small values *a*=5, *b=1,* and *m=7.* If 0 *U* is set to 2, the resulting sequence is

2, 3, 0, 1, 6, 7, 4, 5, 2, 3, 0, 1 �.

and we see that it repeats after the eighth term in the sequence. Of course it *must*repeat after the eighth term or earlier since there are only eight possible values in the series (0,1,2,�,7). Once the series repeats it must continue to cycle endlessly since the next and later terms in the series are completely determined by the current term. Note also that the sequence produced is a little too uniform. There is exactly one occurrence of each possible value in each sequence of 8 values and so, after 16, 24, 32, etc draws we will discover that the number of 0�s, 1�s, etc. are all the same. Morever, notice that the sequence 2, 2 will never occur (and the sequence 2, 3 will occur in every cycle of 8). If we choose poor values of *a, b,* and *m,* then the sequence may cycle much more frequently than every *m+1* terms. If we choose, *a=2, b=2, m*=7, for example, and start with 2, we get

2, 6, 6, 6, 6, �..

a most boring outcome.

Tests of Randomness

Linear congruential RNGs were once considered to be a good method of generating pseudo-random numbers (1960�s) but are now obsolete. As the example above suggests, they did tend to produce (after division by m+2) a series of numbers that took on values in the unit interval that were remarkably uniformly-distributed. They tended to fail in other important respects, notably in the distribution of

sequences of numbers. Modern pseudo-random number generators pass tests of sequential randomness that any linear congruential generator would flunk. They will also cycle if you wait long enough but the length of a cycle is so great that you need never worry about repeating (given the expected life span of the universe).

The point of this example is that we can characterize any pseudo-random number generator in terms of the tests of randomness that it can pass. The linear congruential generator did produce a very uniform distribution but it flunked a simple sequential test. Note that in the sequence we computed, that ODD and EVEN alternate! The tests that a pseudo-random generator can pass determine what it is good for. If, for example, your application requires only a source of uniformly-distributed number, a linear-congruential generator wouldn�t be so bad. But if you were simulating a coin toss sequence by generating random integers and choosing H (heads) when the integer is odd, T (tails) when it is even, you might be very surprised to get HTHTHTHTHTH�.. The pseudo-random number generators available in modern computer languages like *matlab* have passed very many tests of sequential randomness and it is very likely you�ll never be able to tell then from a truly random sequence. But beware. *Never* use a home-grown random number generator. It�s like doing your own dental work. From this point on in the course, we drop the �pseudo� in �pseudo-random generator and, for example, refer to the output of *rand(n,m)* as realizations of a uniform(0,1) random variable.

Generating Non-Uniform Random Variables

Standard *matlab* (and the student edition) provide only two functions that generate random numbers. We�ve see the first, *rand(n,m)*, and the other, *randn(n,m)* generates Normally-distributed (Gaussian-distributed) random variables. The question that confronts us at the moment is, how can we generate realizations of other kinds of random variables such as the exponential distribution we encountered in the first example? The key concept we need is the definition of the *cumulative distribution function (cdf)* of a random variable *X* with pdf

***f(x)* :**

$$F(x) = \int_{-\infty}^{x} f(t)\,dt$$

This is just the probability that the realization of *X* ends up in the interval ( ] ***x***
-�, **. Note the asymmetric brackets here. They�re intended to remind you of Eq. 1.4. Some basic facts about probability guarantee that *F(x)* is non-decreasing and that it goes from 0 to 1.**

With a little bit of geometric reasoning we can show that the cdf of a *U(0,1)* variable is just,

$$\begin{aligned} F_U(x) &= 0 \quad , \quad x \le 0 \\ &= x \quad , \quad 0 < x \le 1 \\ &= 1 \quad , \quad x > 1 \end{aligned}$$

which is plotted in Figure 3.

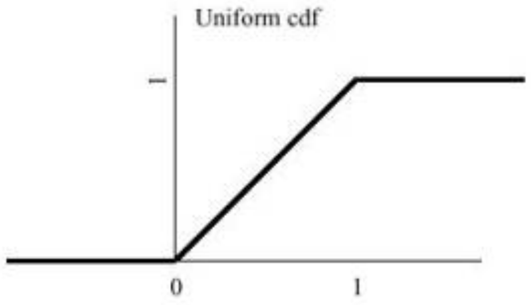

Fig. 3

A little bit of calculus gives us the cdf of the exponential distribution,

$$\begin{aligned} F_E(x) &= 1 - e^{-\alpha x} \quad , \quad x \ge 0 \\ & 0 \qquad\qquad , \quad x < 0 \end{aligned}$$

It turns out that knowledge of the cdf completely determines a random variable just as knowledge of the pdf did. We need to go over some parts of the definitions carefully before we can be certain about that, but it will turn out to be true.

But, back to the problem at hand. We want to generate an exponentially-distributed variable given a U(0,1) variable and the key to that will be the inverse function of the cdf over the interval (0,1), which I�ll denote as the *icdf.* It�s also known as the quantile function for reasons that will become obvious later on. The icdf for the exponential is,

$$F_E^{-1}(u) = -\alpha^{-1} \ln(1-u) \quad , \quad 0 < u < 1$$

Note the suggestive choice of dummy variable, *u. What would happen if we applied the icdf of the exponential to the realization of the uniform(0,1) random variable, U?*

Let Y= $F_E^{-1}(u) = - a^{-1} \ln(1-u)$

Then we want to compute the cdf of *Y:*

$$\begin{aligned} P[Y \le x] &= P[-\alpha^{-1}\ln(1-U) \le x] \\ &= P\left[U \le 1-e^{-\alpha x}\right] = 1-e^{-\alpha x} \end{aligned}$$

when 0 *x* � and it is 0 otherwise. But this is precisely the cdf of the exponentialrandom variable. So we have the following recipe for generating exponential random variables with rate parameter a: generate a U(0,1) random variable and transform it by the icdf of the exponential.

This recipe works for *any* random variable if we substitute the icdf of the random variable in question.

So we have a general recipe for computing random variables with any distribution whose cdf can be written down in a simple closed form

Transformation of random variable distributions :

Since the computer is capable of generating only uniform distributed random variables , other random variables with different distributions have to be generated from uniform distribution using appropriate transformation functions

a) Generation of exponential distribution :

Uniform random variables can be converted into exponentially distributed random variables by using the transformation

$$Y = -\left(1/m\right) * \ln(X)$$

Proof :

The probability density function (pdf) of the uniform variable X is

$$f(x) = 1 \text{ , } 0<X<1$$

From the transformation equation , we get $X = e^{-mY}$

Since Y is a monotonically decreasing function of X , we have

$g(y) = f(x) * | dx/dy |$

� $g(y) = 1 * | -me^{-my} | = me^{-my}$, $Y>0$ which is the pdf of an exponentially distributed random variable with mean m .

b) Generation of Rayleigh distribution from exponential :

This can be accomplished using the transformation

$Y =$ � X

where X is an exponentially distributed random variable .

Proof :

The pdf of an exponential random variable is $f(x) = me^{-mx}$, $X>0$

From the transformation equation , we get $X = Y^2$

Since Y is a monotonically decreasing function of X , we have

$g(y) = f(x) * | dx/dy |$

� $g(y) = m\exp(-my^2) * 2y = (y / s^2) * \exp(- y^2 / 2s^2)$ where $s = 1/2m$

which is the pdf of a Rayleigh distribution with standard deviation s .

c) Generation of Normal distribution from Rayleigh and Uniform random variables :

This transformation equation for this conversion is

$Z = X * \cos Y$ (or) $Z = X * \sin Y$

where X is a Rayleigh distributed random variable and Y is a uniformly distributed random variable in (0,2p).

Proof :

The pdf of a Rayleigh distribution is $f(x) = (x / s^2) * \exp (-x^2 / 2s^2)$, $X>0$

The pdf of the uniform random variable is $g(y) = (1/2p)$, $0 < Y < 2p$

The random variables are independent of each other .

The joint pdf is given as $f(x,y) = f(x)*g(y) = (x/2ps^2)* \exp(-x^2/2s^2)$

Let $U = X*\cos Y$ and $V = X*\sin Y$ � $X =$ � $(U^2 + V^2)$ and $Y = \tan^{-1}(V/U)$

� $d(x,y) / d(u,v)$ = Jacobian of (x,y) w.r.t. $(u,v) = 1 /$ �$(U^2 + V^2)$

The joint pdf of (u,v) is given as $g(u,v) = f(x,y) * | d(x,y) / d(u,v) |$

� $g(u,v) = (1 / 2ps^2) \exp [-(u^2 + v^2) / 2s^2]$

The marginal pdf of U gives $f(u) = f(x*\cos y)$ = � $g(u,v)\, dv$

$f(u) = (1 / s$� $2p)* \exp(-u^2/2s^2)$ since $(1/s$� $2p)$ � $\exp(-v^2/2s^2)dv = 1$

which is the pdf of a normally distributed random variable .

d) Generation of Normal distribution from Uniform distribution:

There are two methods in which this transformation can be achieved .

i) Box-Muller method :

In this method , we use the formula

$Y =$ �$(-2\ln U_1) * \cos(2pU_2)$ (or) $Y =$ �$(-2\ln U_1) * \sin(2pU_2)$

where U_1 and U_2 are uniform random variables .

This formula has been derived from the three transformations given above .

ii) Central Limit Theorem :

According to the Central limit theorem , when a large number of independent identically distributed uniform random variables are added , the resultant sum tends towards a normal distribution . For all practical purposes , the number of random variables to be added in order to get a normal distribution is taken to be 12.
Hence if X_1,X_2,� are independent identically distributed uniform random variables
then (�Xi - N/2) / �(N/12) where N is the number of variables added .

e) Generation of Rayleigh distribution from normal distributions

The formula used for this transformation is

$$Y = \text{�} \left(X_1^2 + X_2^2 \right)$$

where X_1 and X_2 are normal distributions .