Generating Uniform Random Numbers

Christos Alexopoulos and Dave Goldsman

Georgia Institute of Technology, Atlanta, GA, USA

June 7, 2009

Outline

- Introduction
- Some Generators We Won't Use
- Linear Congruential Generators
- Tausworthe Generator
- Generalizations of LCGs
- 6 Choosing a Good Generator Some Theory
- Choosing a Good Generator Statistical Tests
 - χ² Goodness-of-Fit Test
 - Runs Tests for Independence

Introduction

Uniform(0,1) random numbers are the key to random variate generation in simulation.

Goal: Give an algorithm that produces a sequence of *pseudo-random numbers* (*PRN*'s) $R_1, R_2, ...$ that "appear" to be iid Unif(0,1).

Desired properties of algorithm

- output appears to be iid Unif(0,1)
- very fast
- ability to reproduce any sequence it generates

References: Banks, Carson, Nelson, and Nicol (2005); Bratley, Fox, and Schrage (1987); Knuth (2) (1981); Law (2007).

Classes of Unif(0,1) Generators

- output of random device
- table of random numbers
- midsquare (not very useful)
- Fibonacci (not very useful)
- linear congruential (most commonly used in practice)
- Tausworthe (linear recursion mod 2)
- hybrid

Outline

- Introduction
- Some Generators We Won't Use
- Linear Congruential Generators
- Tausworthe Generator
- Generalizations of LCGs
- 6 Choosing a Good Generator Some Theory
- Choosing a Good Generator Statistical Tests
 - χ^2 Goodness-of-Fit Test
 - Runs Tests for Independence

Some Generators We Won't Use

a. Random Devices

Nice randomness properties. However, Unif(0,1) sequence storage difficult, so it's tough to repeat experiment.

Examples:

- flip a coin
- particle count by Geiger counter
- least significant digits of atomic clock

b. Random Number Tables

List of digits supplied in tables.

Cumbersome and slow — not very useful.

Once tabled no longer random.

c. Mid-Square Method (J. von Neumann)

Idea: Take the middle part of the square of the previous random number. John von Neumann was a brilliant and fun-loving guy, but method is lousy!

Example: Take $R_i = X_i/10000$, $\forall i$, where the X_i 's are positive integers < 10000.

Set seed
$$X_0 = 6632$$
; then $6632^2 \rightarrow 43983424$;
So $X_1 = 9834$; then $9834^2 \rightarrow 96707556$;
So $X_2 = 7075$, etc,...

Unfortunately, positive serial correlation in R_i 's.

Also, occasionally degenerates; e.g., consider $X_i = 0003$.

d. Fibonacci and Additive Congruential Generators

These methods are also no good!!

Take

$$X_i = (X_{i-1} + X_{i-2}) \mod m, \quad i = 1, 2, \dots,$$

where $R_i = X_i/m$, m is the modulus, X_0, X_1 are seeds, and $a = b \mod m$ iff a is the remainder of b/m, e.g., $6 = 13 \mod 7$.

Problem: Small numbers follow small numbers.

Also, it's not possible to get $X_{i-1} < X_{i+1} < X_i$ or $X_i < X_{i+1} < X_{i-1}$ (which should occur w.p. 1/3).

Outline

- Introduction
- Some Generators We Won't Use
- Linear Congruential Generators
- Tausworthe Generator
- Generalizations of LCGs
- 6 Choosing a Good Generator Some Theory
- Choosing a Good Generator Statistical Tests
 - χ^2 Goodness-of-Fit Test
 - Runs Tests for Independence

Linear Congruential Generators

LCG's are the most widely used generators. These are pretty good when implemented properly.

$$X_i = (aX_{i-1} + c) \mod m$$
, where X_0 is the seed.

$$R_i = X_i/m, i = 1, 2, \dots$$

Choose a, c, m carefully to get good statistical quality and long *period* or *cycle length*, i.e., time until LCG starts to repeat itself.

If c = 0, LCG is called a *multiplicative* generator.

Trivial Example: For purposes of illustration, consider the LCG

$$X_i = (5X_{i-1} + 3) \bmod 8$$

If $X_0 = 0$, we have $X_1 = (5X_0 + 3) \mod 8 = 3$; continuing,

	i	0	1	2	3	4	5	6	7	8	9
				2							
_	R_i	0	$\frac{3}{8}$	$\frac{2}{8}$	$\frac{5}{8}$	$\frac{4}{8}$	$\frac{7}{8}$	<u>6</u> 8	$\frac{1}{8}$	0	$\frac{3}{8}$

so that the sequence starts repeating with $X_8 = 0$.

This is a *full-period generator*, since it has cycle length m=8. Generally speaking, full-period is a good thing. \Box

Better Example: Here's a portable FORTRAN implementation (Bratley, Fox, and Schrage 1987) that we've seen before. It works fine, is fast, and is full-period with cycle length > 2 billion,

$$X_i = 16807 X_{i-1} \mod (2^{31} - 1).$$

Input: IX is an integer seed between 1 and $2^{31} - 1$.

FUNCTION UNIF(IX)

K1 = IX/127773 (integer division leaves no remainder)

IX = 16807*(IX - K1*127773) - K1*2836

IF (IX.LT.0) IX = IX + 2147483647

UNIF = IX*4.656612875E-10

RETURN

END

Output: UNIF is real-valued in (0,1).

Stay tuned for various ways to assess the quality of PRN generators.

So what can go wrong with LCG's?

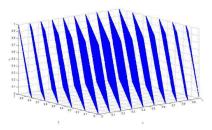
- a. Something like $X_i = (4X_{i-1} + 2) \mod 8$ is *not* full-period, since it only produces even integers.
- b. Something like $X_i = (X_{i-1} + 1) \mod 8$ is full-period, but it produces very non-random output: $X_1 = 1$, $X_2 = 2$, $X_3 = 3$, etc.
- c. In any case, if m is small, you'll get quick cycling whether or not the generator is full period. "Small" could mean anything less than 2 billion or so!
- d. And just because m is big, you still have to be careful. In addition to a. and b. above, some subtle problems can arise. Take a look at RANDU....

Example: The infamous RANDU generator,

$$X_i = 65539 X_{i-1} \bmod 2^{31},$$

was popular during the 1960's.

Here's what (R_{i-2}, R_{i-1}, R_i) look like if you plot them in 3-D (stolen from Wikipedia). If they were truly iid Unif(0,1), you'd see dots randomly dispersed in the unit cube. But instead, the random numbers fall entirely on 15 hyperplanes (not good).



Outline

- Introduction
- Some Generators We Won't Use
- Linear Congruential Generators
- Tausworthe Generator
- Generalizations of LCGs
- 6 Choosing a Good Generator Some Theory
- Choosing a Good Generator Statistical Tests
 - χ^2 Goodness-of-Fit Test
 - Runs Tests for Independence

Tausworthe Generator

Define a sequence of binary digits B_1, B_2, \ldots , by

$$B_i = \left(\sum_{j=1}^q c_j B_{i-j}\right) \bmod 2,$$

where $c_i = 0$ or 1.

Looks a bit like a generalization of LCG's.

Usual implementation (saves computational effort):

$$B_i = (B_{i-r} + B_{i-q}) \mod 2 \quad (0 < r < q).$$

Obtain

$$B_i = 0$$
, if $B_{i-r} = B_{i-q}$ or $B_i = 1$, if $B_{i-r} \neq B_{i-q}$.

To initialize the B_i sequence, specify B_1, B_2, \ldots, B_q .

Example (Law 2007):

$$r=3, q=5; B_1=\cdots=B_5=1$$

 $B_i=(B_{i-3}+B_{i-5}) \ \text{mod} \ \mathbf{2}=B_{i-3} \ \text{XOR} \ B_{i-5}, \quad i>5$
 $B_6=(B_3 \ \text{XOR} \ B_1)=0, B_7=(B_4 \ \text{XOR} \ B_2)=0, \ \text{etc.}$

Turns out period of 0-1 bits is $2^q - 1 = 31$.

How do we go from B_i 's to Unif(0,1)'s?

Easy way: Use (ℓ -bit binary integers)/ 2^{ℓ} .

Example: Set $\ell = 4$ in previous example and get:

$$\frac{15}{16}, \frac{8}{16}, \ldots \to 1111, 1000, \ldots$$

Lots of potential for Tausworthe generators. Nice properties, including long periods, fast calculation. Theoretical issues still being investigated.

Outline

- Introduction
- Some Generators We Won't Use
- Linear Congruential Generators
- Tausworthe Generator
- Generalizations of LCGs
- 6 Choosing a Good Generator Some Theory
- Choosing a Good Generator Statistical Tests
 - χ^2 Goodness-of-Fit Test
 - Runs Tests for Independence

Generalizations of LCGs

A Simple Generalization:

 $X_i = \left(\sum_{j=1}^q a_i X_{i-j}\right) \mod m$, where the a_i 's are constants.

Extremely large periods possible (up to m^q-1 if parameters are chosen properly). But watch out! — Fibonacci is a special case.

Combinations of Generators:

Can combine two generators X_1, X_2, \ldots and Y_1, Y_2, \ldots to construct Z_1, Z_2, \ldots Some suggestions:

- Set $Z_i = (X_i + Y_i) \mod m$
- Shuffling
- Set $Z_i = X_i$ or $Z_i = Y_i$

Sometimes desired properties are improved (difficult to prove).

A Really Good Combined Generator due to L'Ecuyer (1999) (and discussed in Law 2007).

Initialize $X_{1,0}$, $X_{1,1}$, $X_{1,2}$, $X_{2,0}$, $X_{2,1}$, $X_{2,2}$. For $i \geq 3$, set

$$X_{1,i} = (1,403,580 X_{1,i-2} - 810,728 X_{1,i-3}) \operatorname{mod}(2^{32} - 209)$$

$$X_{2,i} = (527,612 X_{2,i-1} - 1,370,589 X_{2,i-3}) \operatorname{mod}(2^{32} - 22,853)$$

$$Y_i = (X_{1,i} - X_{2,i}) \operatorname{mod}(2^{32} - 209)$$

$$U_i = Y_i/(2^{32} - 209)$$

As crazy as this generator looks, it's actually pretty simple, works well, and has an amazing cycle length of about 2^{191} !

It is of interest to note that Matsumoto and Nishimura have developed the "Mersenne Twister" generator, which has period of $2^{19937}-1$ (yes, that's a prime number).

Outline

- Introduction
- Some Generators We Won't Use
- Linear Congruential Generators
- Tausworthe Generator
- Generalizations of LCGs
- 6 Choosing a Good Generator Some Theory
- Choosing a Good Generator Statistical Tests
 - χ^2 Goodness-of-Fit Test
 - Runs Tests for Independence

Choosing a Good Generator — Some Theory

Here are some miscellaneous results due to Knuth and others that are helpful in determining the quality of a PRN generator.

Theorem: The generator $X_{i+1} = aX_i \mod 2^n \ (n > 3)$ can have cycle length of at most 2^{n-2} . This is achieved when X_0 is odd and a = 8k + 3 or a = 8k + 5 for some k.

Theorem: $X_{i+1} = (aX_i + c) \mod m, c > 0$ has full cycle if (i) c and m are relatively prime; (ii) a-1 is a multiple of every prime which divides m; and (iii) a-1 is a multiple of 4 if 4 divides m.

Corollary: $X_{i+1} = (aX_i + c) \mod 2^n \ (c, n > 1)$ has full cycle if c is odd and a = 4k + 1 for some k.

Lots of cycle length results like these.

Example (Banks et al.): $X_i = 13X_{i-1} \mod(64)$.

i	X_i	X_i	X_i	X_i
0	1	2	3	4
1	13	26	39	52
2	41	18	56	36
3	21	42	63	20
4	17	34	51	4
:	:	:	:	:
8	33	2	35	
:	:	:	:	
16	1		3	

The minimum period = 4... terrible random numbers! Why does cycling occur so soon? See first theorem.

And here's a theorem that gives a condition for multiplicative generators to be full period.

Theorem: The multiplicative generator $X_i = aX_{i-1} \mod m$, with prime m has full period (m-1) if and only if

- (a) m divides $a^{m-1} 1$.
- (b) For all integers i < m-1, m does not divide a^i-1 .

How many such multipliers exist?

For $m=2^{31}-1$, it can be shown that 534,600,000 multipliers yield full period.

Remark: The "best" multiplier with $m=2^{31}-1$ is a=950,706,376 (Fishman and Moore 1986).

Geometric Considerations

Theorem: The k-tuples (R_i, \ldots, R_{i+k-1}) , $i \ge 1$, from multiplicative generators lie on parallel hyperplanes in $[0,1]^k$.

The following geometric quantities are of interest.

- Minimum number of hyperplanes (in all directions). Find the multiplier that maximizes this number.
- Maximum distance between parallel hyperplanes. Find the multiplier that minimizes this number.
- Minimum Euclidean distance between adjacent *k*-tuples. Find the multiplier that maximizes this number.

Remark: The RANDU generator is particularly bad since it lies on only 15 hyperplanes.

Can also look at one-step serial correlation.

Serial Correlation of LCG's (Greenberger 1961):

$$Corr(R_1, R_2) \le \frac{1}{a} \left(1 - \frac{6c}{m} + 6(\frac{c}{m})^2 \right) + \frac{a+6}{m}$$

This upper bound is very small for m in the range of 2 billion and, say, a=16807.

Lots of other theory considerations that can be used to evaluate the performance of a particular PRN generator.

Outline

- Introduction
- Some Generators We Won't Use
- Linear Congruential Generators
- Tausworthe Generator
- Generalizations of LCGs
- 6 Choosing a Good Generator Some Theory
- Choosing a Good Generator Statistical Tests
 - χ^2 Goodness-of-Fit Test
 - Runs Tests for Independence

Choosing a Good Generator — Statistical Tests

We'll look at two classes of tests:

Goodness-of-fit tests — are the PRN's approximately Unif(0,1)?

Independence tests — are the PRN's approximately independent?

If a particular generator passes both types of tests (in addition to other tests that we won't tell you about), we'll be happy to use the PRN's it generates.

All tests are of the form H_0 (our null hypothesis) vs. H_1 (the alternative hypothesis).

We regard H_0 as the status quo, so we'll only reject H_0 if we have "ample" evidence against it.

In fact, we want to avoid incorrect rejections of the null hypothesis. Thus, when we design the test, we'll set the *level of significance*

$$\alpha \equiv P(\text{Reject } H_0 | H_0 \text{ true}) = P(\text{Type I error})$$

(typically, α = 0.05 or 0.1). We won't worry about Type II error at this point.

χ^2 Goodness-of-Fit Test

Test $H_0: R_1, R_2, \dots R_n \sim \text{Unif(0,1)}$.

Divide the unit interval into k cells (subintervals). If you choose equi-probable cells $[0,\frac{1}{k}),[\frac{1}{k},\frac{2}{k}),\dots,[\frac{k-1}{k},1]$, then a particular observation R_j will fall in a particular cell with prob 1/k.

Tally how many of the n observations fall into the k cells. If $O_i \equiv \#$ of R_j 's in cell i, then (since the R_j 's are iid), we can easily see that $O_i \sim \text{Bin}(n,\frac{1}{k})$, $i=1,2,\ldots,k$.

Thus, the expected number of R_j 's to fall in cell i will be $E_i \equiv \mathrm{E}[O_i] = n/k, i = 1, 2, \dots, k.$

We'll reject the null hypothesis H_0 if the O_i 's don't match well with the E_i 's.

The χ^2 goodness-of-fit statistic is

$$\chi_0^2 \equiv \sum_{i=1}^k \frac{(O_i - E_i)^2}{E_i}.$$

A large value of this statistic indicates a bad fit.

In fact, we *reject* the null hypothesis H_0 (that the observations are uniform) if $\chi_0^2 > \chi_{\alpha,k-1}^2$, where $\chi_{\alpha,k-1}^2$ is the appropriate $(1-\alpha)$ quantile from a χ^2 table, i.e., $P(\chi_{k-1}^2 < \chi_{\alpha,k-1}^2) = 1-\alpha$.

If $\chi_0^2 \leq \chi_{\alpha,k-1}^2$, we fail to reject H_0 .

Usual recommendation from baby stats class: For the χ^2 g-o-f test to work, pick k,n such that $E_i \geq 5$ and n at least 30. But...

Unlike what you learned in baby stats class, when we test PRN generators, we usually have a *huge* number of observations n (at least millions) with a large number of cells k. When k is large, we can use the approximation

$$\chi^2_{\alpha,k-1} \approx (k-1) \left[1 - \frac{2}{9(k-1)} + z_\alpha \sqrt{\frac{2}{9(k-1)}} \right]^3,$$

where z_{α} is the appropriate standard normal quantile.

Remarks: (1) 16807 PRN generator usually passes the g-o-f test just fine. (2) We'll show how to do g-o-f tests for other distributions later on — just doing uniform PRN's for now. (3) Other g-o-f tests: Kolmogorov-Smirnov test, Anderson-Darling test, etc.

Illustrative Example (Banks et al.): n=100 observations, k=10 intervals. Thus, $E_i=10$ for $i=1,2,\ldots,10$. Further, suppose that $O_1=13,\,O_2=8,\,\ldots,\,O_{10}=11$. (In other words, 13 observations fell in the cell [0,0.1), etc.)

Turns out that

$$\chi_0^2 \equiv \sum_{i=1}^k \frac{(O_i - E_i)^2}{E_i} = 3.4.$$

Let's take $\alpha = 0.05$. Then from χ^2 tables, we have

$$\chi^2_{\alpha,k-1} = \chi^2_{0.05,9} = 16.9.$$

Since $\chi_0^2 < \chi_{\alpha,k-1}^2$, we fail to reject H_0 , and so we'll assume that the observations are approximately uniform. \Box

Runs Tests for Independence

Now we consider $H_0: R_1, R_2, \dots, R_n$ are independent.

First look at Runs Tests.

Consider some examples of coin tossing:

A run is a series of similar observations.

In A above, the runs are: "H", "T", "H", "T",.... (many runs)

In B, the runs are: "HHHHHH", "TTTTT", (very few runs)

In C: "HHH", "TT", "H", "TT",.... (medium number of runs)

A runs test will reject the null hypothesis of independence if there are "too many" or "too few" runs, whatever that means.

Runs Test "Up and Down". Consider the following sequence of uniforms.

If the uniform increases, put a +; if it decreases, put a - (like H's and T's). Get the sequence

Here are the associated runs:

So do we have too many or two few runs?

Let A denote the total number of runs "up and down" out of n observations. (A=6 in the above example.)

Amazing Fact: If n is large (at least 20) and the R_j 's are actually independent, then

$$A \approx \mathsf{Nor}\bigg(\frac{2n-1}{3}, \, \frac{16n-29}{90}\bigg).$$

So if n = 100, we would expect around 67 runs!

We'll reject the null hypothesis if A is too big or small. The standardized test statistic is

$$Z_0 = \frac{A - \mathrm{E}[A]}{\sqrt{\mathrm{Var}(A)}}.$$

Thus, we reject H_0 if $|Z_0|>z_{\alpha/2}.$ E.g., if $\alpha=0.05,$ we reject if $|Z_0|>1.96.$

Illustrative Example: Suppose that n = 100. Then

$$A \approx Nor(66.7, 17.5).$$

So we could expect to see $66.7 \pm 1.96\sqrt{17.5} \approx [58.5, 74.9]$ runs.

If we see anything out of that range, we'll reject.

Lots of other tests available for independence: Other runs tests, correlation tests, gap test, poker test, birthday test, etc.