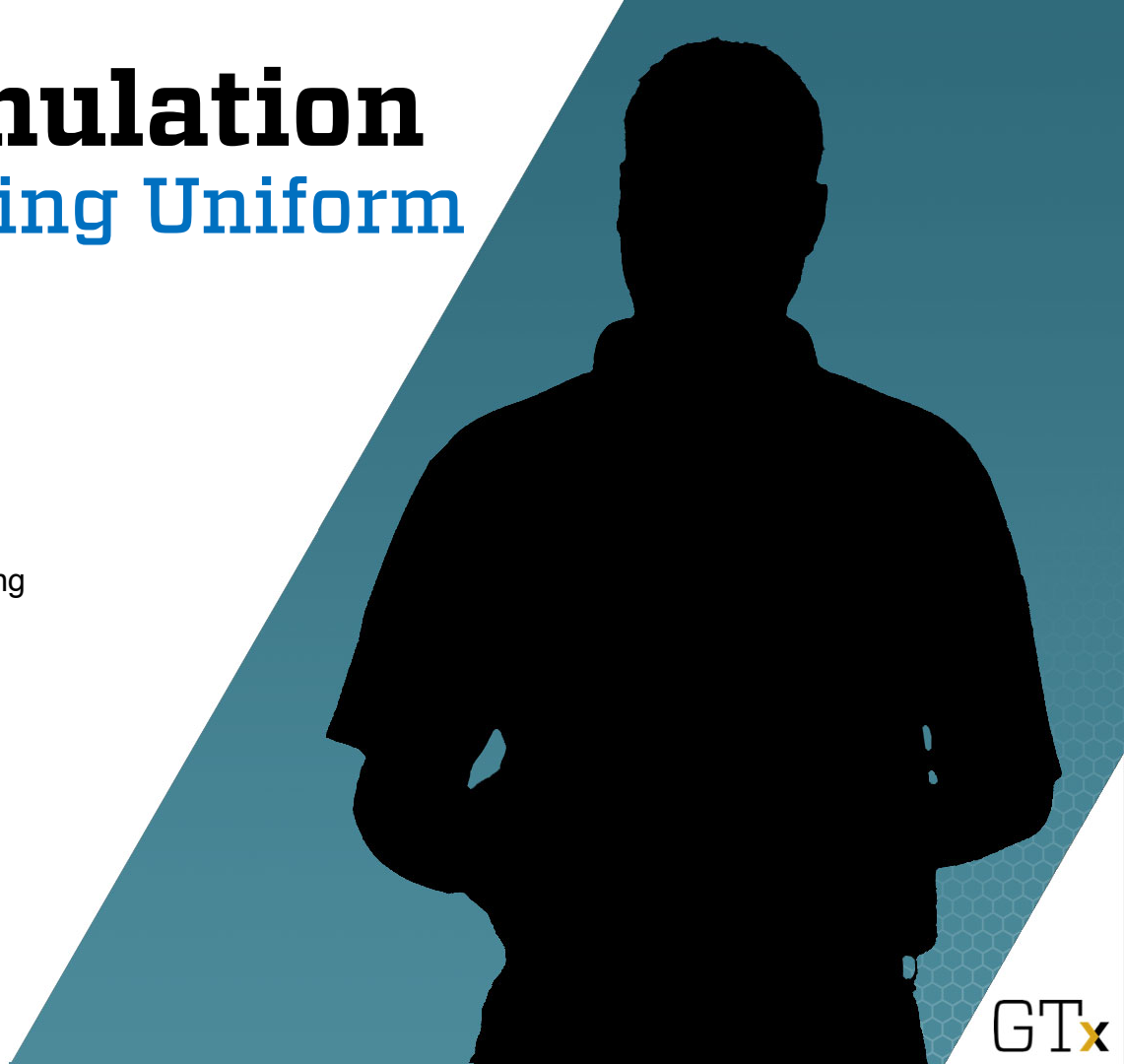
Computer Simulation

Module 6: Generating Uniform Random Numbers

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Introduction

GTx

Module Overview

Last Module: We covered basic (and advanced) Arena concepts.

This Module: We'll do a deep dive to investigate how to generate Unif(0,1) pseudo-random numbers.

Idea: It's a deterministic cookbook!

To Serve Man:
www.youtube.com/watch?v=dk01eeKMD_I

GTx

Module Overview

1. Introduction ← This lesson
2. Some Lousy Generators
3. Linear Congruential Generators
4. Tausworthe Generators
5. Generalizations of LCGs
6. Choosing a Good Generator — Some Theory
7. Choosing a Good Generator — Statistical Tests

GTx

Introduction

Uniform(0,1) random numbers are the key to random variate generation in simulation — you transform uniforms to get other RVs.

Goal: Give an algorithm that produces a sequence of *pseudo-random numbers (PRNs)* $R_1, R_2, \ldots$ that "appear" to be i.i.d. Unif(0,1).

Desired properties of algorithm

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

Intro (cont'd)

Classes of Unif(0,1) Generators

- Some lousy generators
  - output of random device
  - table of random numbers
  - midsquare
  - Fibonacci
- Linear congruential (most commonly used in practice)
- Tausworthe (linear recursion mod 2)
- Hybrid

Summary

This Time: Discussed what's coming up in this module on Unif(0,1) random number generation.

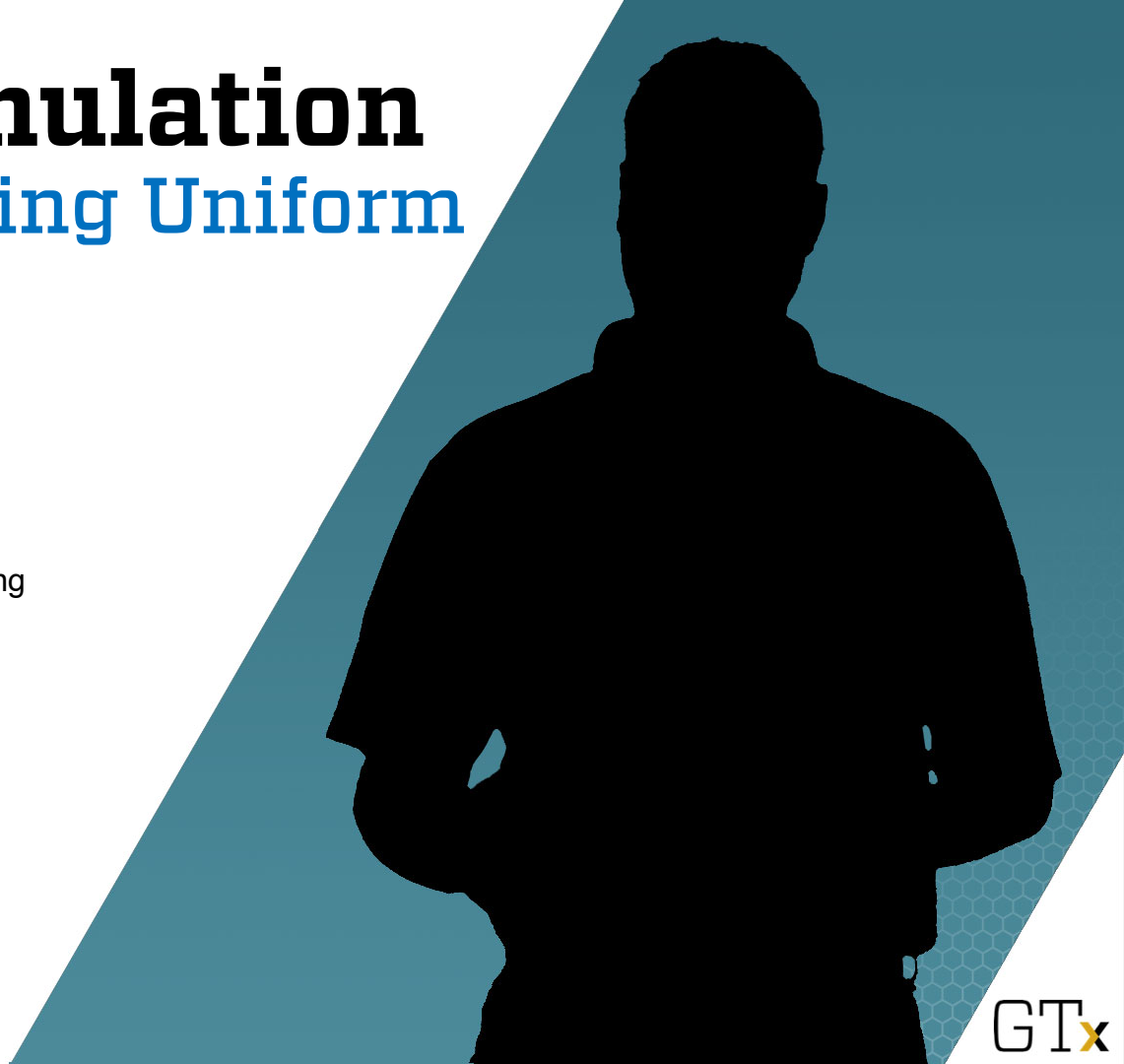
Next Time: We'll actually spend a little time looking at *poor* generators!

GTx

Computer Simulation

Module 6: Generating Uniform Random Numbers

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Some Lousy Generators

GTx

Lesson Overview

Last Lesson: We introduced the topic of Unif(0,1) generation. It's the key to all knowledge!

This Lesson: We'll spend some time talking about *poor* generators.

Idea: What could possibly go wrong? ☹

GTx

Some Lousy Generators

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

More Lousy Generators

b. Random Number Tables
List of digits supplied in tables.

- *A Million Random Digits with 100,000 Normal Deviates*
  www.rand.org/content/dam/rand/pubs/monograph_reports/MR1418/MR1418.digits.pdf

Cumbersome, slow, tables too small — not very useful.
Once tabled no longer random.

Even More!

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

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

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

Set *seed* $X_0 = 6632$; then $6632^2 \rightarrow 43\mathbf{9834}24$;
So $X_1 = 9834$; then $9834^2 \rightarrow 96\mathbf{7075}56$;
So $X_2 = 7075$, etc,...

Unfortunately, positive serial correlation in R_i's.

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

Still More Bad Boys

d. Fibonacci and Additive Congruential Generators

These methods are also no good!!

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

where $R_i = X_i/m$, m is the *modulus*, X_{-1}, X_0 are *seeds*, and $a = b \bmod m$ iff a is the remainder of b/m, e.g., $6 = 13 \bmod 7$.

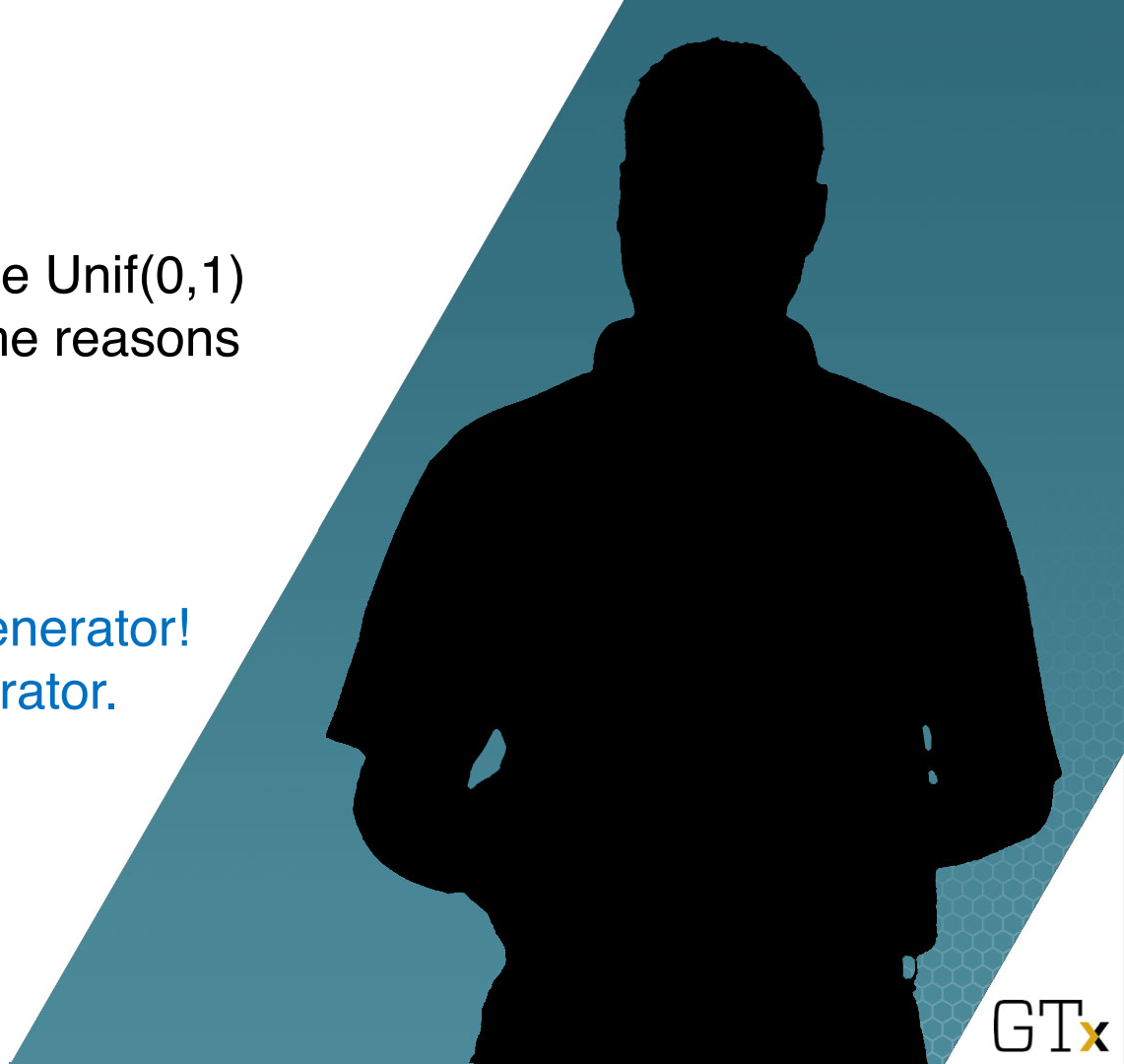
Problem: Small numbers follow small numbers.

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).

Summary

This Time: Talked about some Unif(0,1) rotten apples, as well as some reasons for the worms.

Next Time: Finally, a good generator! The linear congruential generator.

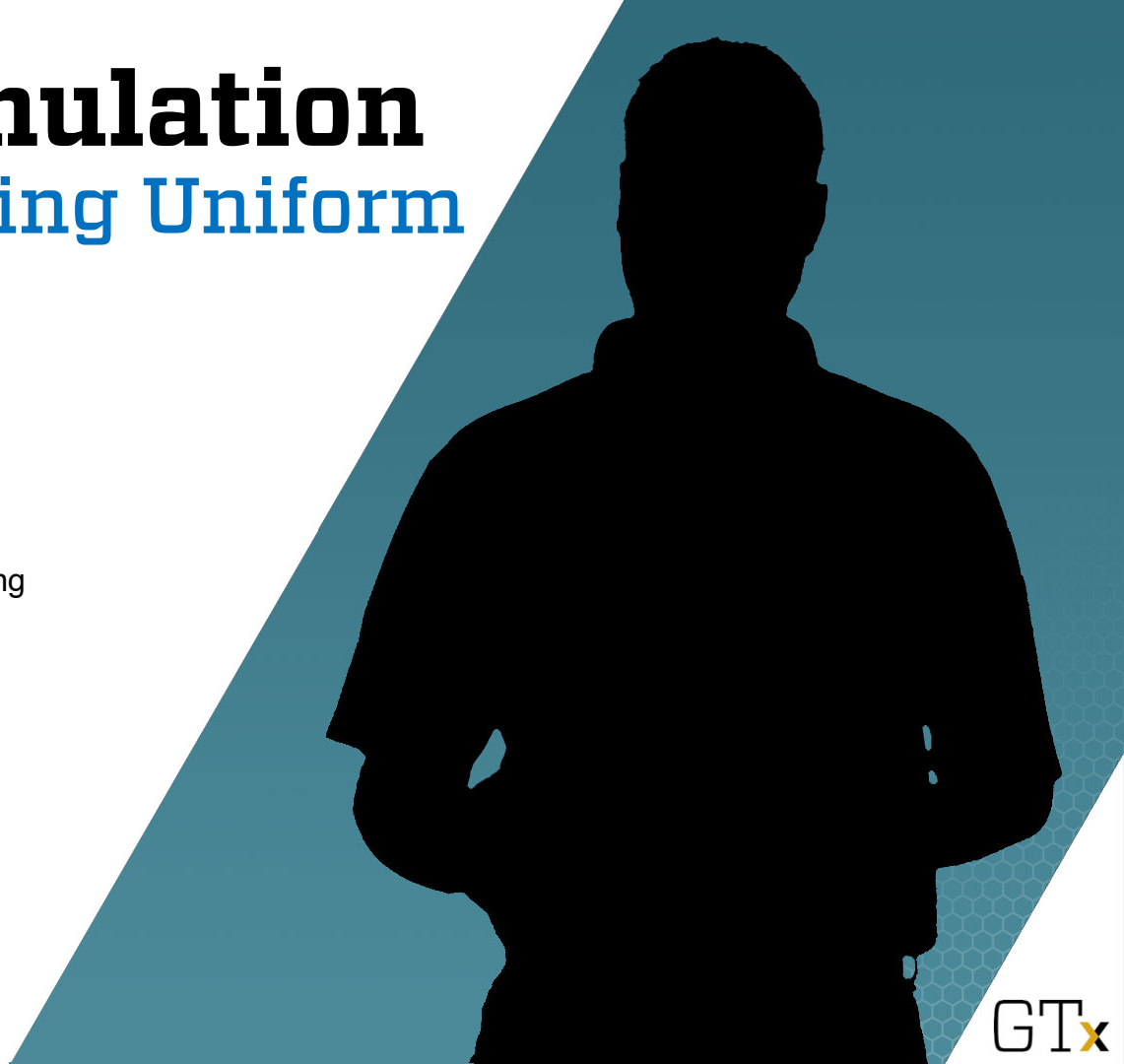


GTx

Computer Simulation

Module 6: Generating Uniform Random Numbers

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Linear Congruential Generators

GTx

Lesson Overview

Last Time: Discussed some awful Unif(0,1) generators.

This Time: We'll look at a really good one – the *linear congruential generator*.

Variations of the LCG are the most commonly used generators in practice.

GTx

LCGs

LCGs are the most widely used generators. These are pretty good when implemented properly.

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

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

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

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

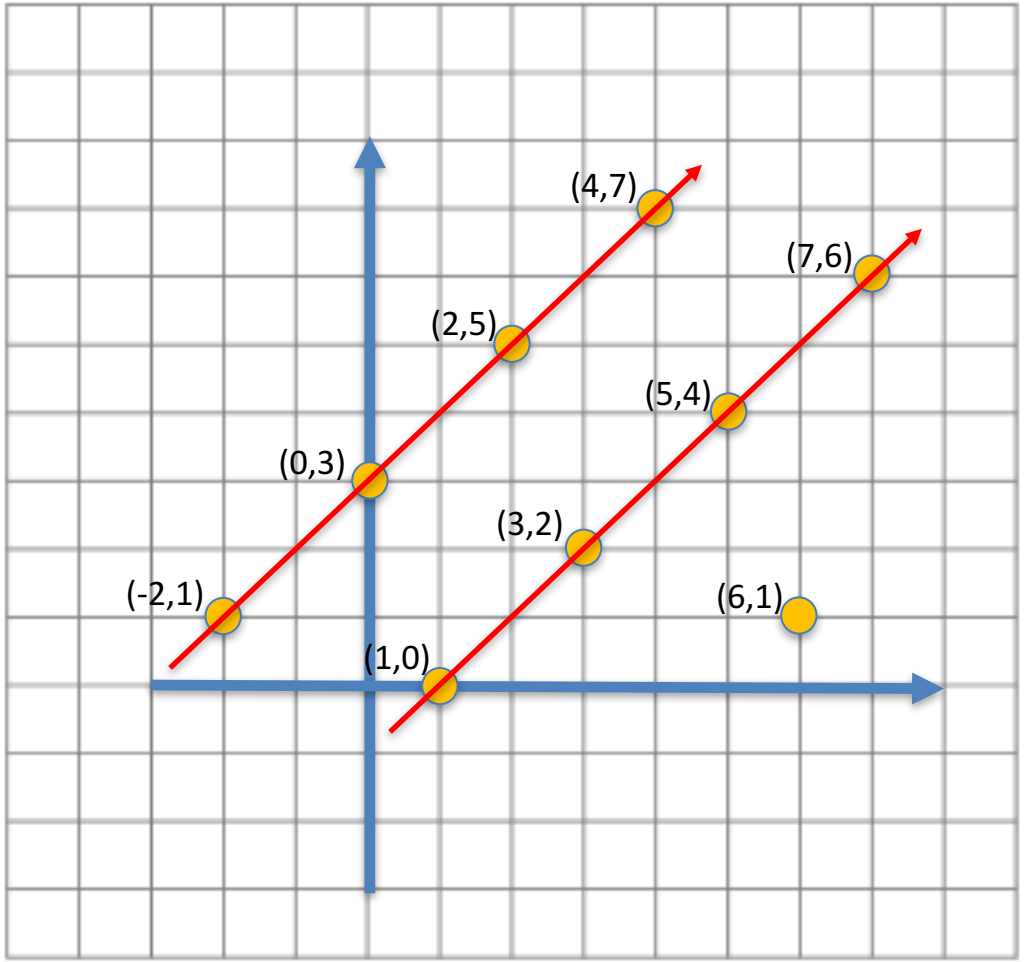
If $X_0 = 0$, we have $X_1 = (5X_0 + 3) \bmod 8 = 3$; continuing,

i	0	1	2	3	4	5	6	7	8	9
X_i	0	3	2	5	4	7	6	1	0	3
R_i	0	$\frac{3}{8}$	$\frac{2}{8}$	$\frac{5}{8}$	$\frac{4}{8}$	$\frac{7}{8}$	$\frac{6}{8}$	$\frac{1}{8}$	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. □

Plot of (X_{i-1}, X_i)



The random numbers fall mainly on the planes

Easy Exercise

Consider the generator

$$X_i = (5\ X_{i-1} + 2) \bmod 8$$

(which is very similar to the previous generator).

Does this generator achieve full cycle? Explain.

Now that's odd. No it's not! ☺

GTx

Better Example (desert island generator): Here's our old 16807 implementation (BFS 1987), which I've translated from FORTRAN. It works fine, is fast, and is full-period with cycle length > 2 billion,

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

Algorithm: Let X_0 be an integer seed between 1 and $2^{31} - 1$.

For $i = 1, 2, \ldots,$

$K \leftarrow \lfloor X_{i-1} \,/\, 127773 \rfloor$ (integer division leaves no remainder)
$X_i \leftarrow 16807(X_{i-1} - 127773K) - 2836K$
if $X_i < 0$, then set $X_i \leftarrow X_i + 2147483647$
$R_i \leftarrow X_i * 4.656612875\text{E-}10$

Example: If $X_0 = 12345678$, then $K = 96$ and

$$X_1 \leftarrow 16807\big[12345678 - 127773(96)\big] - 2836(96) = 1335380034,$$

so that $R_1 = 0.621835$. $\square$

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

First of all, what can go wrong with LCGs?

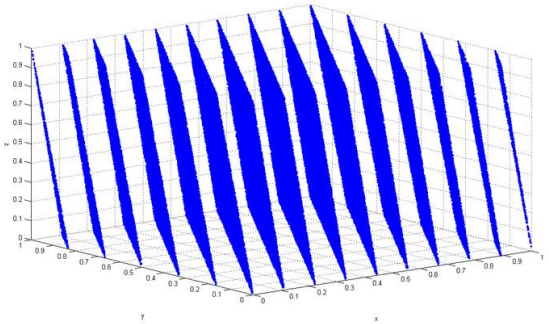
- Something like $X_i = (4X_{i-1} + 2) \bmod 8$ is *not* full-period, since it only produces even integers.
- Something like $X_i = (X_{i-1} + 1) \bmod 8$ is full-period, but it produces very non-random output: $X_1 = 1$, $X_2 = 2$, $X_3 = 3$, etc.
- In any case, if m is small, you'll get quick cycling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



Exercises:

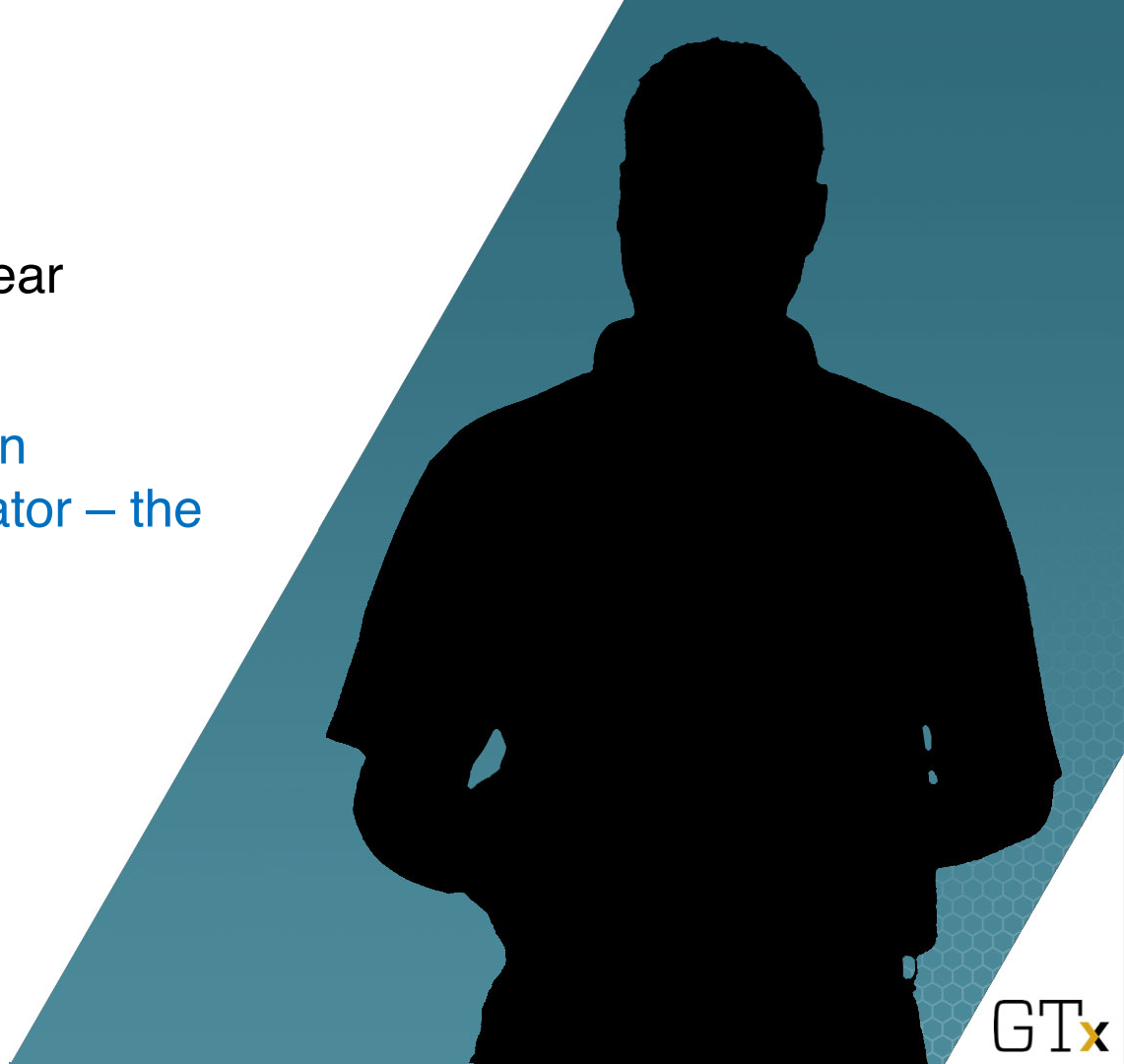
1. Implement RANDU and see how it does. That is, plot R_{i-1} vs. R_i for $i = 1, 2, \ldots, 1000$ and see what happens. Try a few different seeds and maybe you'll see some hyperplanes. You'll certainly see them if you plot (R_{i-2}, R_{i-1}, R_i) in 3-D.

2. Now do the same thing with the 16807 generator. You probably won't be able to see any hyperplanes here.

Summary

This Time: Discussed the linear congruential generator.

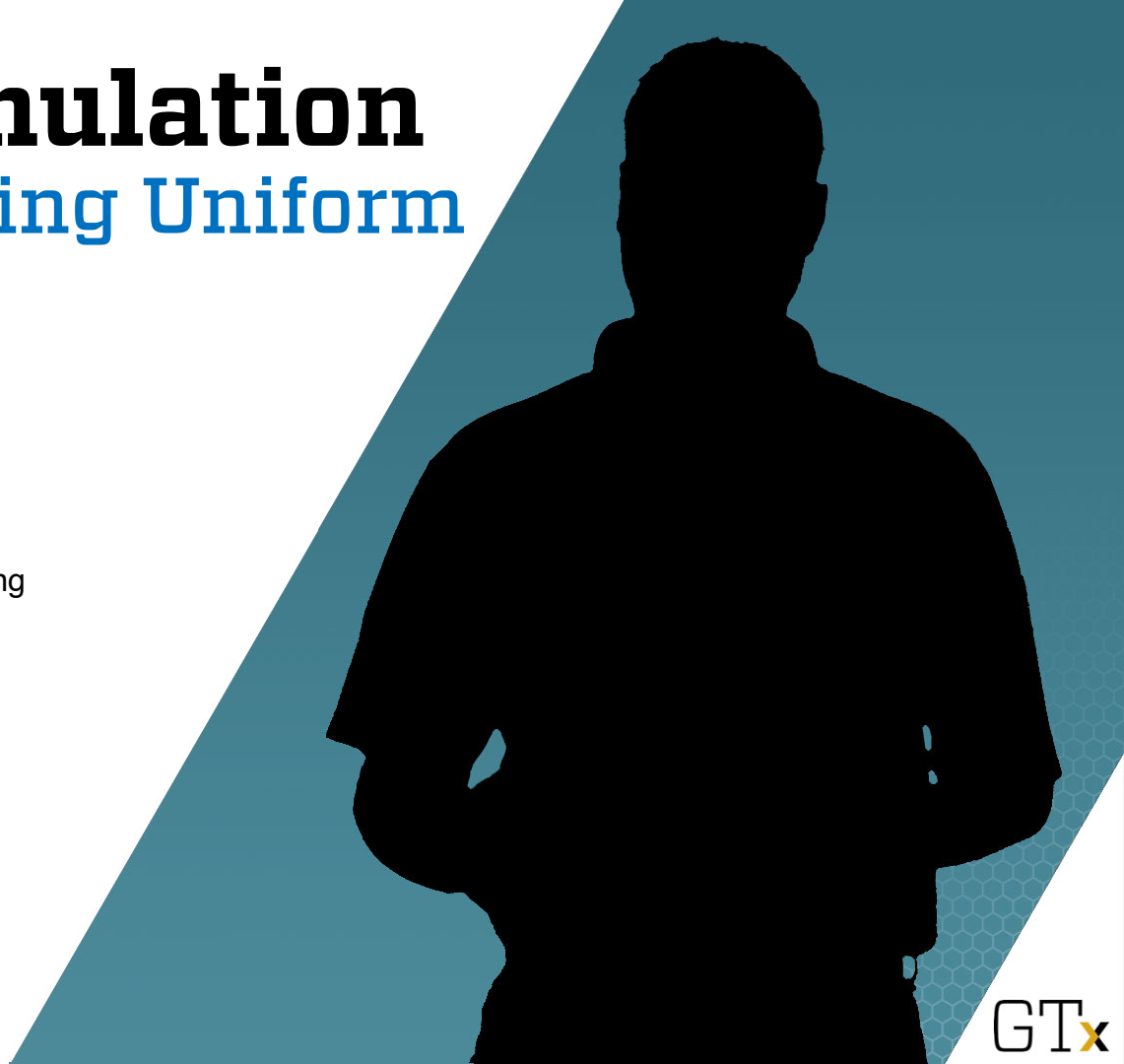
Next Time: We'll talk about an alternative good PRN generator – the Tausworthe generator.



Computer Simulation

Module 6: Generating Uniform Random Numbers

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Tausworthe Generators

GTx

Lesson Overview

Last Time: Discussed LCG PRNs, OMG! QED, BTW! LOL! ☺

This Time: We'll look at a "competing" good generator – the *Tausworthe generator*.

This class of generators seems to be more popular with computer science folks.

GTx

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 $c_j = 0$ or 1. Looks a bit like a generalization of LCGs.

Usual implementation (saves computational effort):

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

Obtain

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

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

Example (Law 2015): $r = 3, q = 5; B_1 = \cdots = B_5 = 1$. Obtain

$B_i = (B_{i-3} + B_{i-5}) \bmod 2 = B_{i-3}$ XOR $B_{i-5}, \quad i > 5$

$B_6 = (B_3 \text{ XOR } B_1) = 0, B_7 = (B_4 \text{ XOR } B_2) = 0$, etc.

1111 1000 1101 1101 0100 0010 0101 1001 1111 □

The period of 0-1 bits is always $2^q - 1 = 31$.

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

Easy way: Use (ℓ-bits in base 2)/2^ℓ and convert to base 10.

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

$$1111_2, 1000_2, 1101_2, 1101_2, \ldots \to \frac{15}{16}, \frac{8}{16}, \frac{13}{16}, \frac{13}{16}, \ldots. \quad \square$$

Lots of potential for Tausworthe generators. Nice properties, including long periods, fast calculation.

Summary

This Time: Discussed Tausworthe generators, which are alternatives to LCGs.

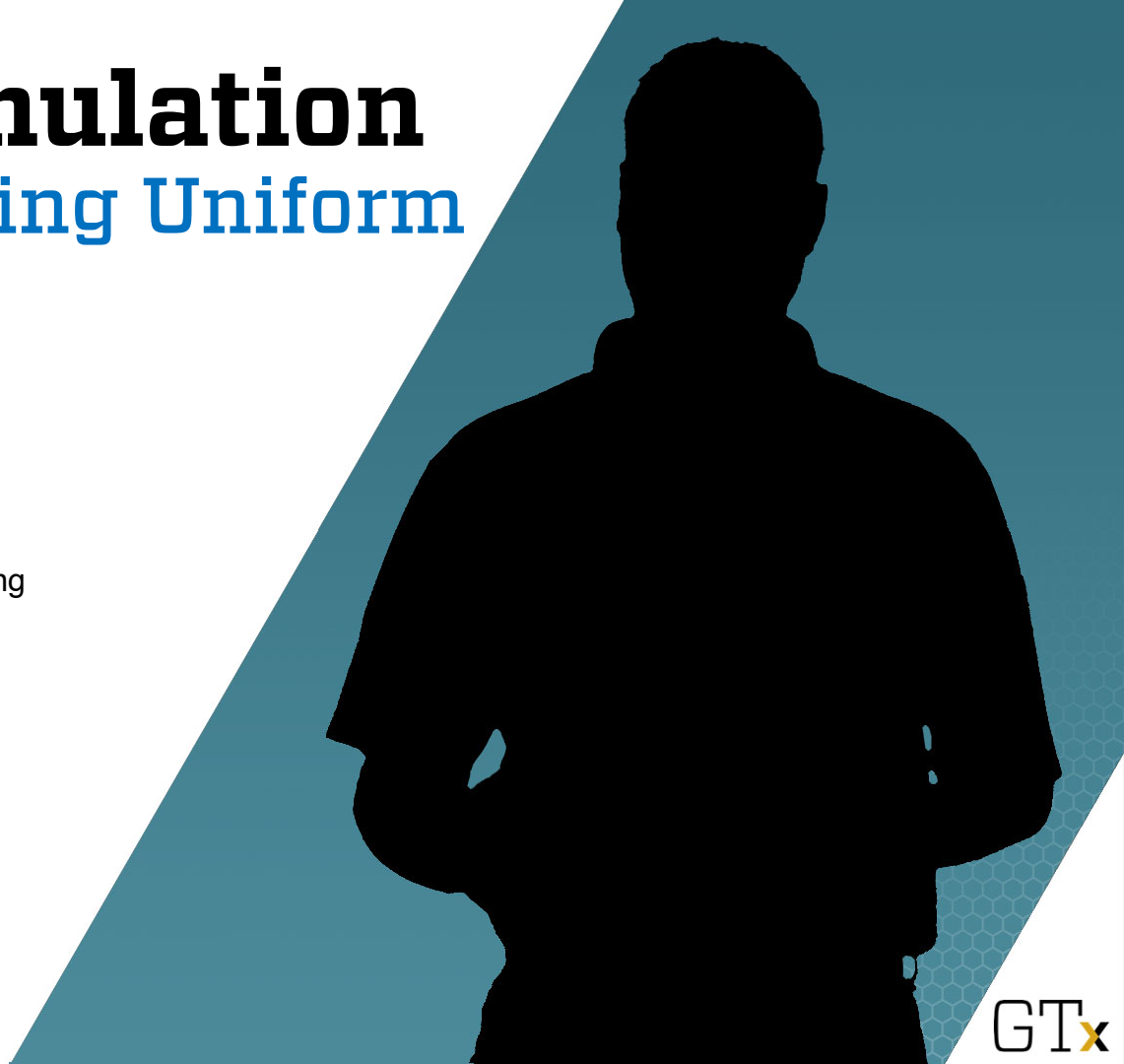
Next Time: We'll look at generalizations of LCGs, some of which have *incredible* properties.

GTx

Computer Simulation

Module 6: Generating Uniform Random Numbers

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Generalizations of LCGs

GTx

Lesson Overview

Last Time: We studied the Tausworthe PRN generator, which actually looked a bit like an LCG.

This Time: We'll look at generalizations of LCGs.

Some of these have *incredible* properties.

GTx

A Simple Generalization:

$X_i = \left(\sum_{j=1}^{q} a_i X_{i-j}\right) \bmod m$, 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 $Z_i = (X_i + Y_i) \bmod m$
- Shuffling
- Set $Z_i = X_i$ or $Z_i = Y_i$

Danger Will Robinson! Properties are difficult to prove!

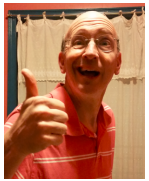
GTx

A Really Good Combined Generator due to L'Ecuyer (1999) (and discussed in Law 2015).

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

$$
\begin{aligned}
X_{1,i} &= (1{,}403{,}580\, X_{1,i-2} - 810{,}728\, X_{1,i-3}) \bmod (2^{32} - 209) \\
X_{2,i} &= (527{,}612\, X_{2,i-1} - 1{,}370{,}589\, X_{2,i-3}) \bmod (2^{32} - 22{,}853) \\
Y_i &= (X_{1,i} - X_{2,i}) \bmod (2^{32} - 209) \\
R_i &= Y_i / (2^{32} - 209)
\end{aligned}
$$

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

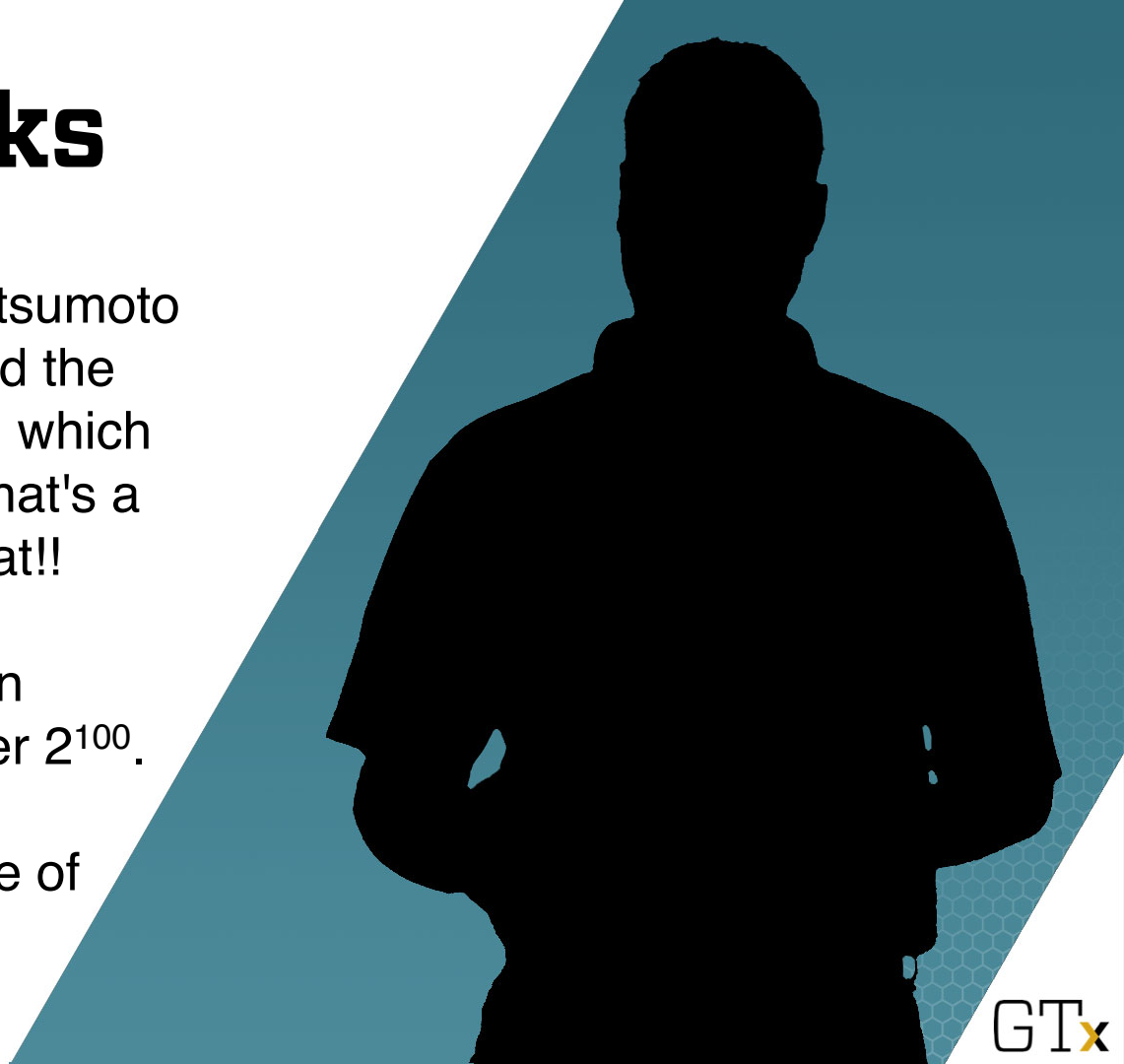


Some Remarks

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) and works great!!

You’ll often need several billion PRN’s, but never anything over 2^{100}.

All standard packages use one of these good generators.



GTx

Summary

This Time: Discussed generalizations of certain PRN generators, and gave a couple *awesome* examples.

Next Time: We'll go over some theoretical considerations when deciding on a good generator to use.

GTx