Random-Number Generation

Raj Jain
Washington University
Saint Louis, MO 63131
Jain@cse.wustl.edu

These slides are available on-line at:
http://www.cse.wustl.edu/~jain/cse574-06/
Overview

- Desired properties of a good generator
- Linear-congruential generators
- Tausworthe generators
- Survey of random number generators
- Seed selection
- Myths about random number generation
Random-Number Generation

- Random Number = Uniform (0, 1)
- Random Variate = Other distributions
  = Function(Random number)
A Sample Generator

\[ x_n = f(x_{n-1}, x_{n-2}, \ldots) \]

- For example,
  \[ x_n = 5x_{n-1} + 1 \mod 16 \]

- Starting with \( x_0 = 5 \):
  \[ x_1 = 5(5) + 1 \mod 16 = 26 \mod 16 = 10 \]

- The first 32 numbers obtained by the above procedure 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5.

- By dividing x's by 16:
  0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375, 0.2500, 0.3125, 0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375, 0.2500, 0.3125.
Terminology

- **Seed** = $x_0$
- **Pseudo-Random**: Deterministic yet would pass randomness tests
- Fully Random: Not repeatable
- **Cycle length, Tail, Period**
Desired Properties of a Good Generator

- It should be efficiently computable.
- The period should be large.
- The successive values should be independent and uniformly distributed.
Types of Random-number Generators

- Linear congruential generators
- Tausworthe generators
- Extended Fibonacci generators
- Combined generators
Linear-Congruential Generators

- Discovered by D. H. Lehmer in 1951
- The residues of successive powers of a number have good randomness properties.

\[ x_n = a^n \mod m \]

Equivalently,

\[ x_n = ax_{n-1} \mod m \]

\( a = \) multiplier
\( m = \) modulus
Linear-Congruential Generators (Cont)

- Lehmer's choices: \( a = 23 \) and \( m = 10^8+1 \)
- Good for ENIAC, an 8-digit decimal machine.
- Generalization:
  \[ x_n = a x_{n-1} + b \mod m \]

- Can be analyzed easily using the theory of congruences
  \( \Rightarrow \) Mixed Linear-Congruential Generators or Linear-Congruential Generators (LCG)
- Mixed = both multiplication by \( a \) and addition of \( b \)
Selection of LCG Parameters

- \( a, b, \) and \( m \) affect the period and autocorrelation.
- The modulus \( m \) should be large.
- The period can never be more than \( m \).
- For mod \( m \) computation to be efficient, \( m \) should be a power of 2 \( \Rightarrow \) Mod \( m \) can be obtained by truncation.
Selection of LCG Parameters (Cont)

- If \( b \) is nonzero, the maximum possible period \( m \) is obtained if and only if:
  - Integers \( m \) and \( b \) are relatively prime, that is, have no common factors other than 1.
  - Every prime number that is a factor of \( m \) is also a factor of \( a-1 \).
  - If integer \( m \) is a multiple of 4, \( a-1 \) should be a multiple of 4.
  - Notice that all of these conditions are met if \( m=2^k \), \( a = 4c + 1 \), and \( b \) is odd. Here, \( c \), \( b \), and \( k \) are positive integers.
Period vs. Autocorrelation

- A generator that has the maximum possible period is called a full-period generator.

\[ x_n = (2^{34} + 1)x_{n-1} + 1 \mod 2^{35} \]

\[ x_n = (2^{18} + 1)x_{n-1} + 1 \mod 2^{35} \]

- Lower autocorrelations between successive numbers are preferable.

- Both generators have the same full period, but the first one has a correlation of 0.25 between \( x_{n-1} \) and \( x_n \), whereas the second one has a negligible correlation of less than \( 2^{-18} \).
Multiplicative LCG

- Multiplicative LCG: \( b=0 \)

\[ x_n = ax_{n-1} \mod m \]

- Two types:
  \[ m = 2^k \]
  \[ m \neq 2^k \]
Multiplicative LCG with $m=2^k$

- $m = 2^k \Rightarrow$ trivial division
  - $\Rightarrow$ Maximum possible period $2^{k-2}$
- Period achieved if multiplier $a$ is of the form $8i \pm 3$, and the initial seed is an odd integer
- One-fourth the maximum possible may not be too small
- Low order bits of random numbers obtained using multiplicative LCG's with $m=2^k$ have a cyclic pattern
Example 26.1a

\[ x_n = 5x_{n-1} \mod 2^5 \]

- Using a seed of \( x_0 = 1 \):
  5, 25, 29, 17, 21, 9, 13, 1, 5, ...
  Period = 8 = 32/4

- With \( x_0 = 2 \), the sequence is: 10, 18, 26, 2, 10, ...
  Here, the period is only 4.
Example 26.1b

- Multiplier not of the form $8i \pm 3$:

\[ x_n = 7x_{n-1} \mod 2^5 \]

- Using a seed of $x_0 = 1$, we get the sequence:
  7, 17, 23, 1, 7,…

- The period is only 4
Multiplicative LCG with \( m \neq 2^k \)

- Modulus \( m = \) prime number
  
  With a proper multiplier \( a \), period = \( m-1 \)
  
  Maximum possible period = \( m \)

- If and only if the multiplier \( a \) is a *primitive root* of the modulus \( m \)

- \( a \) is a primitive root of \( m \) if and only if \( a^n \mod m \neq 1 \) for \( n = 1, 2, ..., m-2 \).
Example 26.2

\[ x_n = 3x_{n-1} \mod 31 \]

- Starting with a seed of \( x_0 = 1 \):
  
  1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1, …

  The period is 30

  \( \Rightarrow \) 3 is a primitive root of 31

- With a multiplier of \( a = 5 \): 1, 5, 25, 1,…

  The period is only 3 \( \Rightarrow \) 5 is not a primitive root of 31

  \[ 5^3 \mod 31 = 125 \mod 31 = 1 \]

- Primitive roots of 31 = 3, 11, 12, 13, 17, 21, 22, and 24.
Schrage's Method

- PRN computation assumes:
  - No round-off errors, integer arithmetic and no overflows
    ⇒ Can't do it in BASIC
  - Product \( a x_{n-1} > \text{Largest integer} \) ⇒ Overflow
- Identity: \( ax \mod m = g(x) + m h(x) \)

Where: \( g(x) = a (x \mod q) - r (x \div q) \)

And: \( h(x) = (x \div q) - (ax \div m) \)

Here, \( q = m \div a, r = m \mod a \)

  `A div B' = dividing A by B and truncating the result.

- For all x's in the range 1, 2, ..., m-1, computing g(x) involves numbers less than m-1.
- If \( r < q \), \( h(x) \) is either 0 or 1, and it can be inferred from g(x); h(x) is 1 if and only if g(x) is negative.
Example 26.3

\[ x_n = 7^5 x_{n-1} \mod (2^{31} - 1) \]

- \( 2^{31} - 1 = 2147483647 \) is a prime number

- \( 7^5 = 16807 \) is one of its 534,600,000 primitive roots

- The product \( a x_{n-1} \) can be as large as \( 16807 \times 2147483647 \approx 1.03 \times 2^{45} \)

- Need 46-bit integers
  - \( a = 16807 \)
  - \( m = 2147483647 \)
  - \( q = m \div a = 2147483647 \div 16807 = 12773 \)
  - \( r = m \mod a = 2147483647 \mod 16807 = 2836 \)

- For a correct implementation, \( x_0 = 1 \implies x_{10000} = 1,043,618,065 \).
Generator Using Integer Arithmetic

FUNCTION Random(VAR x:INTEGER) : REAL;

CONST
   a = 16807;   (* Multiplier *)
   m = 2147483647; (* Modulus *)
   q = 127773;   (* m div a *)
   r = 2836;     (* m mod a *)

VAR
   x_div_q, x_mod_q, x_new: INTEGER;

BEGIN
   x_div_q := x DIV q;
   x_mod_q := x MOD q;
   x_new := a*x_mod_q - r*x_div_q;
   IF x_new > 0 THEN x := x_new ELSE x := x_new + m;
   Random := x/m;
END;
Generator Using Real Arithmetic

FUNCTION Random(VAR x:DOUBLE) : DOUBLE;

CONST
  a = 16807.0D0;  (* Multiplier *)
  m = 2147483647.0D0; (* Modulus *)
  q = 127773.0D0;  (* m div a *)
  r = 2836.0D0;    (* m mod a *)

VAR
  x_div_q, x_mod_q, x_new: DOUBLE;

BEGIN
  x_div_q := TRUNC(x/q);
  x_mod_q := x-q*x_div_q;
  x_new := a*x_mod_q - r*x_div_q;
  IF x_new > 0.0D0 THEN x := x_new ELSE x := x_new + m;
  Random := x/m;
END;
Tausworthe Generators

- Need long random numbers for cryptographic applications
- Generate random sequence of binary digits (0 or 1)
- Divide the sequence into strings of desired length
- Proposed by Tausworthe (1965)

\[ b_n = c_{q-1} b_{n-1} \oplus c_{q-2} b_{n-2} \oplus c_{q-3} b_{n-3} \oplus \cdots \oplus c_0 b_{n-q} \]

Where \( c_i \) and \( b_i \) are binary variables with values of 0 or 1, and \( \oplus \) is the exclusive-or (mod 2 addition) operation.
- Uses the last \( q \) bits of the sequence
  \[ \Rightarrow \text{autoregressive sequence of order } q \text{ or } \text{AR}(q). \]
- An AR\((q)\) generator can have a maximum period of \( 2^q - 1 \).
Tausworthe Generators (Cont)

- \( D = \) delay operator such that \( Db(n) = b(n + 1) \)
  
  \[
  D^q b(i - q) = c_{q-1} D^{q-1} b(i - q) + c_{q-2} D^{q-2} b(i - q) + \cdots + c_0 b(i - q) \mod 2
  \]
  
  \[
  D^q - c_{q-1} D^{q-1} - c_{q-2} D^{q-2} - \cdots - c_0 = 0 \mod 2
  \]
  
  \[
  D^q + c_{q-1} D^{q-1} + c_{q-2} D^{q-2} + \cdots + c_0 = 0 \mod 2
  \]

- **Characteristic polynomial:**
  
  \[
  x^q + c_{q-1} x^{q-1} + c_{q-2} x^{q-2} + \cdots + c_0
  \]

- The period is the smallest positive integer \( n \) for which \( x^n - 1 \) is divisible by the characteristic polynomial.

- The maximum possible period with a polynomial of order \( q \) is \( 2^q - 1 \). The polynomials that give this period are called **primitive polynomials**.
Example 26.4

\[ x^7 + x^3 + 1 \]

- Using \( D \) operator in place of \( x \):
  \[ D^7 b(n) + D^3 b(n) + b(n) = 0 \mod 2 \]
  Or:
  \[ b_{n+7} + b_{n+3} + b_n = 0 \mod 2 \quad n = 0, 1, 2, \ldots \]

- Using the exclusive-or operator
  \[ b_{n+7} \oplus b_{n+4} \oplus b_n = 0 \quad n = 0, 1, 2, \ldots \]
  Or:
  \[ b_{n+7} = b_{n+4} \oplus b_n \quad n = 0, 1, 2, \ldots \]

- Substituting \( n-7 \) for \( n \):
  \[ b_n = b_{n-4} \oplus b_{n-7} \quad n = 7, 8, 9, \ldots \]
Example 26.4 (Cont)

- Starting with $b_0 = b_1 = \cdots = b_6 = 1$:
  \[ b_7 = b_3 \oplus b_0 = 1 \oplus 1 = 0 \]
  \[ b_8 = b_4 \oplus b_1 = 1 \oplus 1 = 0 \]
  \[ b_9 = b_5 \oplus b_2 = 1 \oplus 1 = 0 \]
  \[ b_{10} = b_6 \oplus b_3 = 1 \oplus 1 = 0 \]
  \[ b_{11} = b_7 \oplus b_4 = 0 \oplus 1 = 1 \]

- The complete sequence is:
  1111111 0000111 0111100 1011001 0010000 0010001 0011000 1011101 0110110 0000110 0110101 0011100 1111011 0100001 0101011 1110100 1010001 1011100 0111111 1000011 1000000.

- Period = 127 or $2^7-1$ bits
  \[ \Rightarrow \] The polynomial $x^7 + x^3 + 1$ is a primitive polynomial.
Linear Feedback Shift Register

\[ x^5 + x^3 + 1 \Rightarrow b_n = b_{n-2} \oplus b_{n-5} \]

- This can be easily implemented using shift registers:

- In general:
  
  \[ b_n = c_{q-1} b_{n-1} \oplus c_{q-2} b_{n-2} \oplus c_{q-3} b_{n-3} \oplus \cdots \oplus c_0 b_{n-q} \]

AND gates are not required if \( c_i \)'s are known.
Generating U(0,1)

- Divide the sequence into successive groups of $s$ bits and use the first $l$ bits of each group as a binary fraction:

$$x_n = 0.b_{sn}b_{sn+1}b_{sn+2}b_{sn+3} \cdots b_{sn+l-1}$$

Or equivalently:

$$x_n = \sum_{j=1}^{s} 2^{-j} b_{sn+j-1}$$

Here, $s$ is a constant greater than or equal to $l$ and is relatively prime to $2^q-1$.

$s \geq l \Rightarrow x_n$ and $x_j$ for $n \neq j$ have no bits in common

- Relative prime-ness guarantees a full period $2^q-1$ for $x_n$. 
Example 26.5

\[ b_n = b_{n-4} \oplus b_{n-7} \]

- The period \(2^7-1=127\)
- \(l=8, s=8:\)
  \[ x_0 = 0.11111110_2 = 0.99219_{10} \]
  \[ x_1 = 0.00011101_2 = 0.11328_{10} \]
  \[ x_2 = 0.11100101_2 = 0.89453_{10} \]
  \[ x_3 = 0.10010010_2 = 0.29688_{10} \]
  \[ x_4 = 0.00000100_2 = 0.36328_{10} \]
  \[ x_5 = 0.01001100_2 = 0.42188_{10} \]
  \[
  \cdots
  \]
Properties of Tausworthe Generators

The \( l \)-bit numbers have the following property:

1. The mean of the sequence is one-half:
   \[
   E[x_n] \approx \frac{1}{2}
   \]

2. The variance of the sequence is one-twelfth:
   \[
   \text{Var}[x_n] \approx \frac{1}{12}
   \]

3. The serial correlation is zero:
   \[
   \text{Corr}[x_n, x_{n+s}] = 0 \quad \text{for } 0 < |s| < \frac{(2^q - 1 - l)}{l}
   \]
Properties of Tausworthe Generators (Cont)

4. The sequence is $k$-distributed for all $k$'s up to $\lfloor q/l \rfloor$.

$\Rightarrow$ Every $k$-tuple of $l$-bit numbers appears $2^{q-kl}$ times over the full period except the all-zero tuple, which appears one time less. The bit sequence contains $2^{q-1}$ ones and $2^{q-1}-1$ zeros.

If a window of length $q$ slides along the sequence, each of the $2^{q-1}$ nonzero $k$-tuples appears exactly once in a complete period.
Primitive Trinomials

- Trinomials: Only three non-zero terms
  \[ b_n = b_{n-q+r} \oplus b_{n-q} \]

- \( 2r \leq q \) \( \Rightarrow \) Successive \( q \)-bits can be generated using shift and an exclusive-or sequence:

- The individual bits in a word are read from the right.
For example, the seed is $b_{q-1} b_{q-2} \cdots b_0$.

1. Start with a $q$-bit seed $Y_1$.
2. Right-shift $Y_1$ by $r$ bits, filling with zeros on the left. Call the result $Y_2$.
3. Exclusive-or $Y_1$ and $Y_2$. Call the result $Y_3$. This completes the computation of the right $q-r$ bits.

4. Left-shift $Y_3$ by $q-r$ bits, filling with zeros on the right. Call the result $Y_4$.

5. Exclusive-or $Y_3$ and $Y_4$. The result $Y_5$ is the new $q$-bit seed.
Example 26.6

\[ x^7 + x^3 + 1 \]

- \( r = 3, \ q = 7, \) and \( q - r = 4 \)
  - Need a 3-bit right-shift and a 4-bit left-shift

- Seed \( X = 1111111:\)
  - Step 1: Copy seed \( Y_1 = X = 1111111 \)
  - Step 2: Right-shift by 3 \( Y_2 = 0001111 \)
  - Step 3: Exclusive-or \( Y_3 = Y_1 \oplus Y_2 = 1110000 \)
  - Step 4: Left-shift by 4 \( Y_4 = 0000000 \)
  - Step 5: Exclusive-or \( Y_5 = Y_3 \oplus Y_4 = 1110000 \)
Example 26.6 (Cont)

- The next 7 bits (read from the right) are 0000111. The process can then be repeated:
  
  **Step 1:** Copy seed \( Y_1 = X = 1110000 \)  
  **Step 2:** Right-shift by 3 \( Y_2 = 0001110 \)  
  **Step 3:** Exclusive-or \( Y_3 = Y_1 \oplus Y_2 = 1111110 \)  
  **Step 4:** Left-shift by 4 \( Y_4 = 1100000 \)  
  **Step 5:** Exclusive-or \( Y_5 = Y_3 \oplus Y_4 = 0011110 \)

- The next 7 bits (read from the right) are 0111100
List of Primitive Trinomials.

<table>
<thead>
<tr>
<th>$x^2 + x + 1$</th>
<th>$x^3 + x + 1$</th>
<th>$x^4 + x + 1$</th>
<th>$x^5 + x^2 + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^6 + x + 1$</td>
<td>$x^7 + x + 1$</td>
<td>$x^7 + x^3 + 1$</td>
<td>$x^9 + x^4 + 1$</td>
</tr>
<tr>
<td>$x^{10} + x^3 + 1$</td>
<td>$x^{11} + x^2 + 1$</td>
<td>$x^{15} + x + 1$</td>
<td>$x^{15} + x^4 + 1$</td>
</tr>
<tr>
<td>$x^{15} + x^7 + 1$</td>
<td>$x^{17} + x^3 + 1$</td>
<td>$x^{17} + x^5 + 1$</td>
<td>$x^{17} + x^6 + 1$</td>
</tr>
<tr>
<td>$x^{18} + x^7 + 1$</td>
<td>$x^{20} + x^3 + 1$</td>
<td>$x^{21} + x^2 + 1$</td>
<td>$x^{22} + x + 1$</td>
</tr>
<tr>
<td>$x^{23} + x^5 + 1$</td>
<td>$x^{23} + x^9 + 1$</td>
<td>$x^{25} + x^3 + 1$</td>
<td>$x^{25} + x^7 + 1$</td>
</tr>
<tr>
<td>$x^{28} + x^3 + 1$</td>
<td>$x^{28} + x^9 + 1$</td>
<td>$x^{28} + x^{13} + 1$</td>
<td>$x^{29} + x^2 + 1$</td>
</tr>
<tr>
<td>$x^{31} + x^3 + 1$</td>
<td>$x^{31} + x^6 + 1$</td>
<td>$x^{31} + x^7 + 1$</td>
<td>$x^{31} + x^{13} + 1$</td>
</tr>
</tbody>
</table>

If $x^q + x^r + 1$ is listed then $x^q + x^{q-r} + 1$ is also primitive.
Disadvantages of Tausworthe Generators

- The sequence may produce good test results over the complete cycle, it may not have satisfactory local behavior.
- It performed negatively on the runs up and down test.
- Although the first-order serial correlation is almost zero, it is suspected that some primitive polynomials may give poor high-order correlations.
- Not all primitive polynomials are equally good.
Generalized Feedback Shift Register (GFSR)

- l-bit sequence $x_n$ is generated as follows:
  
  $x_n = 0.b_n b_{n+s} b_{n+2s} \cdots b_{n+(l-1)s}$

  Here, $s$ is a "carefully selected delay."

- The sequence $x_n$ can be generated very efficiently using word-wide shift and exclusive-or instructions.

- Need to store an array of numbers and carefully initialize the array.
Extended Fibonacci Generators

- Fibonacci sequence: \( x_n = x_{n-1} + x_{n-2} \)
- Random numbers: \( x_n = x_{n-1} + x_{n-2} \mod m \)

- High serial correlation
- Not good randomness properties
- Combine the fifth and seventeenth most recent values:
  \( x_n = x_{n-5} + x_{n-17} \mod 2^k \)

This generator passes most statistical tests.

- It can be implemented as follows using 17 storage locations
  \( L[1], \ldots, L[17] \)

- Initialization: fill the locations with 17 integers, not all even, and set two pointers \( i \) and \( j \) to 17 and 5, respectively.
Extended Fibonacci Generators (Cont)

- On each successive call:
  \[
  x := L[i] + L[j]; \\
  L[i] := x; \\
  i := i - 1; \text{ IF } i = 0 \text{ THEN } i := 17; \\
  j := j - 1; \text{ IF } j = 0 \text{ THEN } j := 17; \\
  \text{Return } x;
  \]

- The add operation in the first line is automatically modulo \(2^k\) in \(k\)-bit machines with 2's complement arithmetic.

- The period of the generator is \(2^k(2^{17}-1)\).

- For \(k = 8, 16, \text{ and } 32\), this period is \(1.6 \times 10^7, 4.3 \times 10^9, \text{ and } 2.8 \times 10^{14}\), respectively.

- The period is considerably longer than that possible with LCGs.
Combined Generators

1. Adding random numbers obtained by two or more generators.

\[ w_n = (x_n + y_n) \mod m \]

For example, L'Ecuyer (1986):

\[ x_n = 40014x_{n-1} \mod 2147483563 \]
\[ y_n = 40692y_{n-1} \mod 2147483399 \]

This would produce:

\[ w_n = (x_n - y_n) \mod 2147483562 \]

Period = \(2.3 \times 10^{18}\)
Another Example: For 16-bit computers:

\[ w_n = 157w_{n-1} \mod 32363 \]

\[ x_n = 146x_{n-1} \mod 31727 \]

\[ y_n = 142y_{n-1} \mod 31657 \]

Use:

\[ v_n = (w_n - x_n + y_n) \mod 32362 \]

This generator has a period of \(8.1 \times 10^{12}\).
Combined Generators (Cont)

2. Exclusive-or random numbers obtained by two or more generators.

3. Shuffle. Use one sequence as an index to decide which of several numbers generated by the second sequence should be returned.
Algorithm M:

a) Fill an array of size, say, 100.
b) Generate a new \( y_n \) (between 0 and \( m-1 \))
c) Index \( i = l + 100 \frac{y_n}{m} \)
d) \( i \)th element of the array is returned as the next random number

e) A new value of \( x_n \) is generated and stored in the \( i \)th location
A currently popular multiplicative LCG is:
\[ x_n = 7^5 x_{n-1} \mod (2^{31} - 1) \]

- Used in:
  - SIMPL/I system (IBM 1972),
  - APL system from IBM (Katzan 1971),
  - PRIMOS operating system from Prime Computer (1984), and
  - Scientific library from IMSL (1980)
- \( 2^{31}-1 \) is a prime number and \( 7^5 \) is a primitive root of it
  \( \Rightarrow \) Full period of \( 2^{31}-2 \).
- This generator has been extensively analyzed and shown to be good.
- Its low-order bits are uniformly distributed.
Survey of RNG’s (Cont)

- Fishman and Moore (1986)'s exhaustive search of $m=2^{31}-1$:
  
  \[ x_n = 48271x_{n-1} \mod (2^{31} - 1) \]

  \[ x_n = 69621x_{n-1} \mod (2^{31} - 1) \]

- SIMSCRIPT II.5 and in DEC-20 FORTRAN:
  
  \[ x_n = 630360016x_{n-1} \mod (2^{31} - 1) \]
Survey of RNG’s (Cont)

``RANDU'' (IBM 1968): Very popular in the 1960s:

\[ x_n = (2^{16} + 3)x_{n-1} \mod 2^{31} \]

- Modulus and the multiplier were selected primarily to facilitate easy computation.
- Multiplication by \(2^{16}+3=65539\) can be easily accomplished by a few shift and add instructions.
- Does not have a full period and has been shown to be flawed in many respects.
- Does not have good randomness properties (Knuth, p 173).
- Triplets lie on a total of 15 planes
  \(\Rightarrow\) Unsatisfactory three-distributivity
- Like all LCGs with \(m=2^k\), the lower order bits of this generator have a small period. RANDU is no longer used.
Survey of RNG’s (Cont)

- Analog of RANDU for 16-bit microprocessors:
  \[ x_n = (2^8 + 3)x_{n-1} \mod (2^{15}) \]
  - This generator shares all known problems of RANDU
  - Period = only a few thousand numbers
    ⇒ not suitable for any serious simulation study

- University of Sheffield Pascal system for Prime Computers:
  \[ x_n = 16807x_{n-1} \mod 2^{31} \]
  - 16807 ≠ 8i± 3 ⇒ Does not have the maximum possible period of \( 2^{31}-2 \).
  - Used with a shuffle technique in the subroutine UNIFORM of the SAS statistical package
SIMULA on UNIVAC uses the following generator:

$$x_n = 5^{13} x_{n-1} \mod 2^{35}$$

- Has maximum possible period of $2^{33}$, Park and Miller (1988) claim that it does not have good randomness properties.

The UNIX operating system:

$$x_n = (1103515245 x_{n-1} + 12345) \mod 2^{32}$$

- Like all LCGs with $m=2^k$, the binary representation of $x_n$'s has a cyclic bit pattern
Seed Selection

- **Multi-stream simulations**: Need more than one random stream
  - Single queue ⇒ Two streams
    = Random arrival and random service times

1. **Do not use zero**. Fine for mixed LCGs. But multiplicative LCG or a Tausworthe LCG will stick at zero.

2. **Avoid even values**. For multiplicative LCG with modulus $m=2^k$, the seed should be odd. Better to avoid generators that have too many conditions on seed values or whose performance (period and randomness) depends upon the seed value.

3. **Do not subdivide one stream**.
Seed Selection (Cont)

4. Do not generate successive seeds: \( u_1 \) to generate inter-arrival times, \( u_2 \) to generate service time \( \Rightarrow \) Strong correlation

5. Use non-overlapping streams.
Overlap \( \Rightarrow \) Correlation, e.g., Same seed \( \Rightarrow \) same stream

6. Reuse seeds in successive replications.

7. Do not use random seeds: Such as the time of day.
Can't reproduce. Can't guaranteed non-overlap.

8. Select \( \{u_0, u_{100,000}, u_{200,000}, \ldots\} \)

\[
x_n = a^n x_0 + \frac{c(a^n - 1)}{a - 1} \mod m
\]
Table of Seeds

\[ x_n = 7^5 x_{n-1} \mod (2^{31} - 1) \]

<table>
<thead>
<tr>
<th>( x_{100000} )</th>
<th>( x_{100000}(i+1) )</th>
<th>( x_{100000}(i+2) )</th>
<th>( x_{100000}(i+3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>46,831,694</td>
<td>1,841,581,359</td>
<td>1,193,163,244</td>
</tr>
<tr>
<td>727,633,698</td>
<td>933,588,178</td>
<td>804,159,733</td>
<td>1,671,059,989</td>
</tr>
<tr>
<td>1,061,288,424</td>
<td>1,961,692,154</td>
<td>1,227,283,347</td>
<td>1,171,034,773</td>
</tr>
<tr>
<td>276,090,261</td>
<td>1,066,728,069</td>
<td>209,208,115</td>
<td>554,590,007</td>
</tr>
<tr>
<td>721,958,466</td>
<td>1,371,272,478</td>
<td>675,466,456</td>
<td>1,095,462,486</td>
</tr>
<tr>
<td>1,808,217,256</td>
<td>2,095,021,727</td>
<td>1,769,349,045</td>
<td>904,914,315</td>
</tr>
<tr>
<td>373,135,028</td>
<td>717,419,739</td>
<td>881,155,353</td>
<td>1,489,529,863</td>
</tr>
<tr>
<td>1,521,138,112</td>
<td>298,370,230</td>
<td>1,140,279,430</td>
<td>1,335,826,707</td>
</tr>
<tr>
<td>706,178,559</td>
<td>110,356,601</td>
<td>884,434,366</td>
<td>962,338,209</td>
</tr>
<tr>
<td>1,341,315,363</td>
<td>709,314,158</td>
<td>591,449,447</td>
<td>431,918,286</td>
</tr>
<tr>
<td>851,767,375</td>
<td>606,179,079</td>
<td>1,500,869,201</td>
<td>1,434,868,289</td>
</tr>
<tr>
<td>263,032,577</td>
<td>753,643,799</td>
<td>202,794,285</td>
<td>715,851,524</td>
</tr>
</tbody>
</table>
Myths About Random-Number Generation

1. *A complex set of operations leads to random results.* It is better to use simple operations that can be analytically evaluated for randomness.

2. *A single test, such as the chi-square test, is sufficient to test the goodness of a random-number generator.* The sequence \(0,1,2,\ldots,m-1\) will pass the chi-square test with a perfect score, but will fail the run test \(\Rightarrow\) Use as many tests as possible.

3. *Random numbers are unpredictable.* Easy to compute the parameters, \(a, c,\) and \(m\) from a few numbers \(\Rightarrow\) LCGs are unsuitable for cryptographic applications
4. Some seeds are better than others. May be true for some.

\[ x_n = (9806x_{n-1} + 1) \mod (2^{17} - 1) \]

- Works correctly for all seeds except \( x_0 = 37911 \)
- Stuck at \( x_n = 37911 \) forever
- Such generators should be avoided.
- Any nonzero seed in the valid range should produce an equally good sequence.
- For some, the seed should be odd.
- Generators whose period or randomness depends upon the seed should not be used, since an unsuspecting user may not remember to follow all the guidelines.
5. **Accurate implementation is not important.**
   - RNGs must be implemented without any overflow or truncation. For example,
     \[ x_n = \text{1103515245} x_{n-1} + 12345 \mod 2^{31} \]
   - In FORTRAN:
     \[ x_n = (\text{1103515245} x_{n-1} + 12345).\text{AND}.'7FFFFFFF' \]
   - The AND operation is used to clear the sign bit.
   - Straightforward multiplication above will produce overflow.

6. **Bits of successive words generated by a random-number generator are equally randomly distributed.**
   - If an algorithm produces \( l \)-bit wide random numbers, the randomness is guaranteed only when all \( l \) bits are used to form successive random numbers.
Example 26.7

\[ x_n = (25173x_{n-1} + 13849) \mod 2^{16} \]

Notice that:

a) Bit 1 (the least significant bit) is always 1.

b) Bit 2 is always 0.

c) Bit 3 alternates between 1 and 0, thus, it has a cycle of length 2.

d) Bit 4 follows a cycle (0110) of length 4.

e) Bit 5 follows a cycle (11010010) of length 8.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n )</th>
<th>( x_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Decimal</td>
<td>Binary</td>
</tr>
<tr>
<td>1</td>
<td>25,173</td>
<td>01100010 01010101</td>
</tr>
<tr>
<td>2</td>
<td>12,345</td>
<td>00110000 00111001</td>
</tr>
<tr>
<td>3</td>
<td>54,509</td>
<td>11010100 11101101</td>
</tr>
<tr>
<td>4</td>
<td>27,825</td>
<td>01101100 10110001</td>
</tr>
<tr>
<td>5</td>
<td>55,493</td>
<td>11011000 11000101</td>
</tr>
<tr>
<td>6</td>
<td>25,449</td>
<td>01100011 01101001</td>
</tr>
<tr>
<td>7</td>
<td>13,277</td>
<td>00110011 11011101</td>
</tr>
<tr>
<td>8</td>
<td>53,857</td>
<td>11010010 01100001</td>
</tr>
<tr>
<td>9</td>
<td>64,565</td>
<td>11111100 00110101</td>
</tr>
<tr>
<td>10</td>
<td>1945</td>
<td>00000111 10011001</td>
</tr>
<tr>
<td>11</td>
<td>6093</td>
<td>00010111 11001101</td>
</tr>
<tr>
<td>12</td>
<td>24,849</td>
<td>01100001 00010001</td>
</tr>
<tr>
<td>13</td>
<td>48,293</td>
<td>10111100 10100101</td>
</tr>
</tbody>
</table>
Example 26.7 (Cont)

- The least significant bit is either always 0 or always 1.
- The $l$th bit has a period at most $2^l$. ($l=1$ is the least significant bit)
- For all mixed LCGs with $m=2^k$:
  - The $l$th bit has a period at most $2^l$.
  - In general, the high-order bits are more randomly distributed than the low-order bits.
    ⇒ Better to take the high-order $l$ bits than the low-order $l$ bits.
Pseudo-random numbers are used in simulation for repeatability, non-overlapping sequences, long cycle.

It is important to implement PRNGs in integer arithmetic without overflow => Schrage’s method.

For multi-stream simulations, it is important to select seeds that result in non-overlapping sequences.

Two or more generators can be combined for longer cycles.

Bits of random numbers may not be random.
Homework

Submit answer to Exercise 26.5. Submit code and report $x_{20,000}$
Exercise 26.1

What is the maximum period obtainable from the following generator:

\[ x_n = a x_{n-1} \mod 2^4 \]

What should be the value of \( a \)?
What restrictions are required on the seed?
Exercise 26.2

Determine $24^n \mod 31$ for $n = 1, \ldots, 30$.
Find the smallest $n$ for which the mod operation's result is 1.
Is 24 is a primitive root of 31?
Exercise 26.3

Determine all primitive roots of 11.
Exercise 26.4

- Compute the period of the following generator:

\[ x_n = 13x_{n-1} \mod 2311 \]
Exercise 26.5

Implement the following LCG using Schrage's method to avoid overflow:

\[ x_n = 40014x_{n-1} \mod 2147483563 \]

Using a seed of \( x_0 = 1 \), determine \( x_{10000} \).
Exercise 26.6

Implement the following LCG with and without the Schrage's method:

\[ x_n = 11x_{n-1} \mod 31 \]

Are the sequences generated the same? If not, explain what is the problem.
Exercise 26.7

- Can you implement the following LCG using Schrage's method?

\[ x_n = 24x_{n-1} \mod 31 \]

Write programs to implement the generator with and without the Schrage's method and justify your answer.
Exercise 26.8

Determine which of the following polynomials is a primitive polynomial:

1. \( x^2 + x + 1 \)
2. \( x^3 + x^2 + 1 \)
3. \( x^4 + x^2 + 1 \)
4. \( x^5 + x^2 + 1 \)
Exercise 26.9

Determine the period of the Tausworthe sequence generated using each of the following characteristic polynomials:

1. $x^5 + x + 1$
2. $x^6 + x^3 + 1$
3. $x^8 + x + 1$
4. $x^{30} + x^{15} + 1$
Exercise 26.10

Generate five 6-bit numbers using the Tausworthe method for the following characteristic polynomial starting with a seed of $x_0=0.111111_2$:

$$x^6 + x + 1$$
Exercise 26.11

What is wrong with the parameters of the following two generators?

\[ x_n = (40x_{n-1} + 3641) \mod 729 \]

\[ x_n = (61x_{n-1} + 2323) \mod 500 \]
Exercise 26.12

Generate 48 random numbers using a seed of $x_0 = 1$ in the following mixed LCG:

$$x_n = 13x_{n-1} + 11 \mod 2^{16}$$

Find the period of $l$th bit from the right for $l=1,2\ldots,5$. 