

EEC 686/785  
Modeling & Performance Evaluation of  
Computer Systems

## Lecture 17

Wenbing Zhao

Department of Electrical and Computer Engineering  
Cleveland State University

wenbing@ieee.org

(based on Dr. Raj Jain's lecture notes)



## Outline

- Review of lecture 16
- Testing random-number generators

6 November 2005

EEC686/785

Wenbing Zhao



## Random-Number Generation

- A key step in developing a simulation: have a routine to generate random values for variables with a specified random distribution
  - **Random number generation:** a sequence of random numbers distributed uniformly between 0 and 1
  - **Random variate generation:** the sequence is transformed to produce random values obeying the desired distribution

6 November 2005

EEC686/785

Wenbing Zhao



## Terminology

- **Seed:**  $x_0$
- **Pseudo-random:** deterministic yet would pass randomness tests
- **Fully random:** not repeatable
- **Cycle length, Tail, Period:**

6 November 2005

EEC686/785

Wenbing Zhao

## 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

$$x_n = a^n \bmod m$$

Equivalently,

$$x_n = a x_{n-1} \bmod m$$

$a$  = multiplier,  $m$  = modulus

## Tausworthe Generators

- Generate random sequence of binary digits (0 or 1). Divide the sequence into strings of desired length
- Proposed by Tausworthe (1965):  

$$b_n = c_{q-1} b_{n-1} \oplus c_{q-2} b_{n-2} \oplus c_{q-3} b_{n-3} \oplus \dots \oplus c_0 b_{n-q}$$
 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 => autoregressive sequence of order  $q$  or AR( $q$ )
- An AR( $q$ ) generator can have a maximum period of  $2^q - 1$

## Tausworthe Generators

- Characteristic polynomial  
 $x^q + c_{q-1}x^{q-1} + c_{q-2}x^{q-2} + \dots + c_0$
- The period is the smallest positive integer  $n$  for which  $x^n - 1$  is divisible by the characteristic polynomial
- The maximum possible period which a polynomial of order  $q$  is  $2^q - 1$ . The polynomials that give this period are called **primitive polynomials**

## 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:

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

Or equivalently:  $x_n = \sum_{j=1}^l 2^{-j} b_{sn+j-1}$

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 primeness guarantees a full period  $2^q - 1$  for  $x_n$

## Example 26.5

- $b_n = b_{n-2} \oplus b_{n-5}$
- The period  $2^7 - 1 = 127$
- $l = 8, s = 8$ :  
 $x_0 = 0.11111110_2 = 0.99219_{10}$   
 $x_1 = 0.00011101_2 = 0.11328_{10}$   
 $x_2 = 0.11100101_2 = 0.89453_{10}$   
 $x_3 = 0.10010010_2 = 0.29688_{10}$   
 $x_4 = 0.00000100_2 = 0.36328_{10}$   
 $x_5 = 0.01001100_2 = 0.42188_{10}$   
 ...

## Properties of Tausworthe Generators

- The  $l$ -bit numbers have the following property:
  - The mean of the sequence is one-half:  $E[x_n] \approx 1/2$
  - The variance of the sequence is one-twelfth:  $\text{Var}[x_n] \approx 1/12$
  - The serial correlation is zero:  
 $\text{Corr}[x_n, x_{n+2}] = 0$  for  $0 < |s| < (2^q - 1 - l)/l$
  - The sequence is  $k$ -distributed for all  $k$ 's up to  $\lfloor q/l \rfloor$ .  
 This means that 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

## Primitive Trinomials

- Only three non-zero terms  $\Rightarrow$  generation of each new bit requires just one exclusive-or operation

$$b_n = b_{n-q+r} \oplus b_{n-q}$$

$2r \leq q \Rightarrow$  successive  $q$ -bits can be generated using shift and an exclusive-or sequence:

- 1. The individual bits in a word are read from the right. For example, the seed is  $b_{q-1} b_{q-2} \dots b_0$
- 2. Start with a  $q$ -bit seed  $Y_1$

## Primitive Trinomials

- 3. Right-shift  $Y_1$  by  $r$  bits, filling with zeros on the left. Call the result  $Y_2$
- 4. Exclusive-or  $Y_1$  and  $Y_2$ . Call the result  $Y_3$ . This completes the computation of the right  $q - r$  bits
- 5. Left-shift  $Y_3$  by  $q - r$  bits, filling with zeros on the right. Call the result  $Y_4$
- 6. Exclusive-or  $Y_3$  and  $Y_4$ . The result  $Y_5$  is the new  $q$ -bit seed

## Example 26.4

- $x^7 + x^3 + 1$
- $r=3$ ,  $q=7$ , and  $q - r = 4 \Rightarrow$  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 26.4

- 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

$x^2+x+1$	$x^3+x+1$	$x^4+x+1$	$x^5+x^2+1$
$x^6+x+1$	$x^7+x+1$	$x^7+x^3+1$	$x^9+x^4+1$
$x^{10}+x^3+1$	$x^{11}+x^2+1$	$x^{15}+x+1$	$x^{15}+x^4+1$
$x^{15}+x^7+1$	$x^{17}+x^3+1$	$x^{17}+x^5+1$	$x^{17}+x^6+1$
$x^{18}+x^7+1$	$x^{20}+x^3+1$	$x^{21}+x^2+1$	$x^{22}+x+1$
$x^{23}+x^5+1$	$x^{23}+x^9+1$	$x^{25}+x^3+1$	$x^{25}+x^7+1$
$x^{28}+x^3+1$	$x^{28}+x^9+1$	$x^{28}+x^{13}+1$	$x^{29}+x^2+1$
$x^{31}+x^3+1$	$x^{31}+x^6+1$	$x^{31}+x^7+1$	$x^{31}+x^{13}+1$

- 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

- $l$  bits sequence  $x_n$  is generated as follows:

$$x_n = 0.b_n b_{n+s} b_{n+2s} \dots b_{n+(l-1)s}$$

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:  
 $x_n = x_{n-1} + x_{n-2} \bmod m$   
=> High serial correlation  
=> Not good randomness properties
- Combine the fifth and seventeenth most recent values:  
 $x_n = x_{n-5} + x_{n-17} \bmod 2^k$   
This generator passes most statistical tests and recommends its implementation as follows using 17 storage locations  $L[1], \dots, L[17]$

## Extended Fibonacci Generators

- Initialization: fill the locations with 17 integers, not all even, and set two pointers  $i$  and  $j$  to 17 and 5, respectively
- On each successive call:
  - $x := L[i] + L[j];$
  - $L[i] := x;$
  - $i := i - 1; \text{ IF } i = 0 \text{ THEN } i := 17;$
  - $j := j - 1; \text{ IF } j = 0 \text{ THEN } j := 17;$
  - Return  $x;$

## Extended Fibonacci Generators

- The period of the generator is  $2^k(2^{17} - 1)$
- For  $k=8, 16,$  and  $32,$  this period is  $1.6 \times 10^7, 4.3 \times 10^9,$  and  $2.8 \times 10^{14},$  respectively
- The period is considerably longer than that possible with LCGs

## Combined Generators

- Adding random numbers obtained by two or more generators

$$w_n = (x_n + y_n) \bmod m$$

- For example, L'Ecuyer (1986):

$$x_n = 40014 x_{n-1} \bmod 2147483563$$

$$y_n = 40692 y_{n-1} \bmod 2147483399$$

This would produce:

$$w_n = (x_n - y_n) \bmod 2147483562$$

$$\text{Period} = 2.3 \times 10^{18}$$

## Combined Generators

- For 16-bit computers:

$$w_n = 157w_{n-1} \bmod 32363$$

$$x_n = 146x_{n-1} \bmod 31727$$

$$y_n = 142y_{n-1} \bmod 31657$$

This would produce:

$$v_n = (w_n - x_n + y_n) \bmod 32362$$

This generator has a period of  $8.1 \times 10^{12}$

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

## Combined Generators

- Shuffle. Use one sequence as an index to decide which of several numbers generated by the second sequence should be returned
- Algorithm M:
  - Fill an array of size, say, 100
  - Generate a new  $y_n$  (between 0 and  $m - 1$ )
  - Index  $i = 1 + 100 y_n / m$
  - $i$ th element of the array is returned as the next random number
  - A new value of  $x_n$  is generated and stored in the  $i$ th location
- Disadvantage: can't skip a long subsequence as is usually required in multiple stream simulations

## Seed Selection

- **Multistream simulations:** need more than one random stream. Single queue = two streams. Random arrival and random service times
- **Do not use zero.** Fine for mixed LCGs, but multiplicative LCG or a Tausworthe LCG will stick at zero
- **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

## Seed Selection

- **Do not subdivide one stream:** using a single stream for all variables is a common mistake.
  - $u_1$  to generate inter-arrival times,  $u_2$  to generate service time => strong correlation
- **Use non-overlapping streams.**
  - Overlap => correlation. E.g., same seed => same stream
- **Reuse seeds in successive replications**
- **Do not use random seeds such as the time of day**
  - Can't reproduce. Can't guarantee non-overlap

## Myths About Random-Number Generation

- A complex set of operations leads to random results
  - It is better to use simple operations that can be analytically evaluated for randomness
- A single test, such as the chi-square test, is sufficient to test the goodness of a random-number generator
  - The sequence  $0, 1, 2, \dots, m-1$  will pass the chi-square test with a perfect score, but will fail the run test => use as many tests as possible
- Random numbers are unpredictable
  - Easy to compute the parameters,  $a$ ,  $c$ , and  $m$  from a few numbers => LCGs are unsuitable for cryptographic applications

## Myths About Random-Number Generation

29

- Some seeds are better than others
  - $x_n = (9806x_{n-1} + 1) \bmod (2^{17} - 1)$  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

6 November 2005

EEC686/785

Wenbing Zhao

## Myths About Random-Number Generation

30

- Accurate implementation is not important
  - Must be implemented without any overflow or truncation
- 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

6 November 2005

EEC686/785

Wenbing Zhao

## Example 26.7

31

- $x_n = (25173x_{n-1} + 13849) \bmod 2^{16}$
- Starting with a seed of  $x_0$
- Notice that:
  - Bit 1 (the least significant bit) is always 1
  - Bit 2 is always 0
  - Bit 3 alternates between 1 and 0, thus, it has a cycle of length 2
  - Bit 4 follows a cycle (0110) of length 4
  - Bit 5 follows a cycle (11010010) of length 8

$n$	$x_n$	
	Decimal	Binary
1	25173	01100010 01010101
2	12345	00110000 00111001
3	54509	11010100 11101101
4	27825	01101100 10110001
5	55493	11011000 11000101
6	25449	01100011 01101001
7	13277	00110011 11011101
8	53857	11010010 01100001
9	64565	11111100 00110101
10	1945	00000111 10011001
11	6093	00010111 11001101
12	24849	01100001 00010001
13	48293	10111100 10100101
14	52425	11001100 11001001
15	61629	11110000 10111101
16	18625	01001000 11000001
17	2581	00001010 00010101
18	25337	01100010 11111001
19	11949	00101110 10101101
20	47473	10111001 01110001

6 November 2005

EEC686/785

Wenbing Zhao

## Example 26.7

32

- In general,  $l$ th bit follows a cycle of length  $2^{l-2}$  for  $l \geq 2$
- True for all LCGs with modulus  $m = 2^k$  both mixed and multiplicative
  - For all multiplicative LCGs with  $m = 2^k$ :
    - ✦ 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

6 November 2005

EEC686/785

Wenbing Zhao

## Testing Random-Number Generators

- Goal: to ensure that the random number generator produces a random stream
  - Plot histograms
  - Plot quantile-quantile plot
  - Use other tests
  - Passing a test is necessary but not sufficient
  - Pass  $\neq$  good; fail  $\Rightarrow$  bad

## Chi-Square Test

- Most commonly used test. Can be used for any distribution
- Prepare a histogram of the observed data
- Compare observed frequencies with theoretical
  - $k$  = number of cells
  - $o_i$  = observed frequency for  $i$ th cell
  - $e_i$  = expected frequency

$$D = \sum_{i=1}^k \frac{(o_i - e_i)^2}{e_i}$$

## Chi-Square Test

- $D=0 \Rightarrow$  exact fit
- $D$  has a chi-square distribution with  $k - 1$  degrees of freedom
  - $\Rightarrow$  compare  $D$  with  $\chi_{[1-\alpha; k-1]}^2$
  - Pass with confidence  $\alpha$  if  $D$  is less

## Example 27.1

- 1000 random numbers with  $x_0 = 1$ :

$$x_n = (125 x_{n-1} + 1) \bmod 2^{12}$$

- $\chi_{[0.9; 9]}^2 = 14.68$
- Observed difference = 10.380  
Less  $\Rightarrow$  Accept IID  
 $U(0, 1)$

Cell	Observed	Expected	$\frac{(\text{Observed}-\text{Expected})^2}{\text{Expected}}$
1	100	100.0	0.000
2	96	100.0	0.160
3	98	100.0	0.040
4	85	100.0	2.250
5	105	100.0	0.250
6	93	100.0	0.490
7	97	100.0	0.090
8	125	100.0	6.250
9	107	100.0	0.490
10	94	100.0	0.360
Total	1000	1000.0	10.380

## Chi-Square for Other Distributions

- Errors in cells with a small  $e_i$  affect the chi-square statistic more
- Best when  $e_i$ 's are equal  $\Rightarrow$  use an equiprobable histogram with variable cell sizes
- Combine adjoining cells so that the new cell probabilities are approximately equal
- Designed for discrete distributions and for large sample sizes only

## Kolmogorov-Smirnov Test

- Developed by A. N. Kolmogorov and N. V. Smirnov
- Designed for continuous distributions
- Difference between the observed CDF (cumulative distribution function)  $F_o(x)$  and the expected cdf  $F_e(x)$  should be small
- $K^+$  = maximum observed deviation above the expected cdf  
 $K^-$  = maximum observed deviation below the expected cdf

$$K^+ = \sqrt{n} \max_x (F_o(x) - F_e(x))$$

$$K^- = \sqrt{n} \max_x (F_e(x) - F_o(x))$$

## Kolmogorov-Smirnov Test

- $K^+ < K_{[1-\alpha;n]}$  and  $K^- < K_{[1-\alpha;n]}$   $\Rightarrow$  pass at  $\alpha$  level of significance
- Use  $F_e(x_{i+1}) - F_o(x_i)$  for  $K^+$
- For  $U(0, 1)$ :  $F_e(x) = x$ ,  $F_o(x) = j/n$ , where  $x > x_1, x_2, \dots, x_{j-1}$

$$K^+ = \sqrt{n} \max_j \left( \frac{j}{n} - x_j \right)$$

$$K^- = \sqrt{n} \max_j \left( x_j - \frac{j-1}{n} \right)$$

## Example 27.2

- 30 random numbers using a seed of  $x_0 = 15$ :  
 $x_n = 3x_{n-1} \bmod 31$
- The numbers are: 14, 11, 2, 6, 18, 23, 7, 31, 1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15
- The normalized numbers obtained by dividing the sequence by 31 are: 0.45161, 0.35484, ...

$j$	$x_j$	$\frac{j}{n} - x_j$	$x_j - \frac{j-1}{n}$
1	0.03226	0.00108	0.03226
2	0.06452	0.00215	0.03118
3	0.09677	0.00323	0.03011
4	0.12903	0.00430	0.02903
5	0.16129	0.00538	0.02796
6	0.19355	0.00645	0.02688
7	0.22581	0.00753	0.02581
8	0.25806	0.00860	0.02473
9	0.29032	0.00968	0.02366
10	0.32258	0.01075	0.02258
12	0.38710	0.01290	0.02043
13	0.41935	0.01398	0.01935
14	0.45161	0.01505	0.01828
15	0.48387	0.01613	0.01720
16	0.51613	0.01720	0.01613
17	0.54839	0.01828	0.01505
18	0.58065	0.01935	0.01398
19	0.61290	0.02043	0.01290
20	0.64516	0.02151	0.01183
21	0.67742	0.02258	0.01075
22	0.70968	0.02366	0.00968
23	0.74194	0.02473	0.00860
24	0.77419	0.02581	0.00753
25	0.80645	0.02688	0.00645
26	0.83871	0.02796	0.00538
27	0.87097	0.02903	0.00430
28	0.90323	0.03011	0.00323
29	0.93548	0.03118	0.00215
30	0.96774	0.03226	0.00108
Max		0.03226	0.03226

## Example 27.2

$$K^+ = \sqrt{n} \max_j \left( \frac{j}{n} - x_j \right) = \sqrt{30} \times 0.03226 = 0.1767$$

$$K^- = \sqrt{n} \max_j \left( x_j - \frac{j-1}{n} \right) = \sqrt{30} \times 0.03226 = 0.1767$$

$K_{[0.9;n]}$  value for  $n = 30$  and  $\alpha = 0.1$  is 1.0424  $\Rightarrow$  pass

## Chi-Square vs. K-S Test

K-S Test	Chi-Square Test
Small samples	Large sample
Continuous distributions	Discrete distributions
Differences between observed and expected CDFs	Differences between observed and hypothesized probabilities (pdfs or pmfs)
Use each observation in the sample without any grouping $\Rightarrow$ makes a better use of the data	Groups observations into a small number of cells
Cell size is not a problem	Cell sizes affect the conclusion but no firm guidelines
Exact	Approximate

## Serial-Correlation Test

- Nonzero covariance  $\Rightarrow$  dependence
  - The inverse is not true
- $R_k$  = autocovariance at lag  $k = \text{Cov}[x_n, x_{n+k}]$

$$R_k = \frac{1}{n-k} \sum_{i=1}^{n-k} \left( U_i - \frac{1}{2} \right) \left( U_{i+k} - \frac{1}{2} \right)$$

- For large  $n$ ,  $R_k$  is normal distributed with a mean of zero and a variance of  $1/[144(n-k)]$

## Serial-Correlation Test

- 100(1- $\alpha$ )% confidence interval for the autocovariance is

$$R_k \mp z_{1-\alpha/2} / (12\sqrt{n-k})$$

- Check if CI includes zero
- For  $k=0$ ,  $R_0$  = variance of the sequence  
Expected to be 1/12 for IID  $U(0,1)$

## Example 27.3

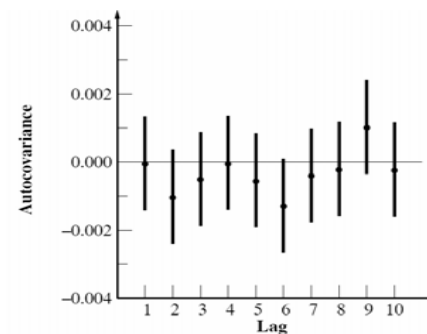
- 10,000 random numbers with  $x_0 = 1$ :

$$x_n = 7^5 x_{n-1} \bmod (2^{31} - 1)$$

Lag	Autocovariance	St. Dev.	90% Confidence Interval	
$k$	$R_k$	of $R_k$	Lower Limit	Upper Limit
1	-0.000038	0.000833	-0.001409	0.001333
2	-0.001017	0.000833	-0.002388	0.000354
3	-0.000489	0.000833	-0.001860	0.000882
4	-0.000033	0.000834	-0.001404	0.001339
5	-0.000531	0.000834	-0.001902	0.000840
6	-0.001277	0.000834	-0.002648	0.000095
7	-0.000385	0.000834	-0.001757	0.000986
8	-0.000207	0.000834	-0.001579	0.001164
9	0.001031	0.000834	-0.000340	0.002403
10	-0.000224	0.000834	-0.001595	0.001148

## Example 27.3

- All confidence intervals include zero => pass



## Two-Level Tests

- If the sample size is too small, the test results may apply locally, but not globally to the complete cycle
- Similarly, global test may not apply locally
- Use two-level tests => use Chi-square test on  $n$  samples of size  $k$  each and then use a Chi-square test on the set of  $n$  Chi-square statistics so obtained => Chi-square on Chi-square test
- Similarly, K-S on K-S
- Can also use this to find a “nonrandom” segment of an otherwise random sequence

## $k$ -Dimensional Uniformity or $k$ -Distribution

- Chi-square => uniformity in one dimension  
=> given two real numbers  $a_1$  and  $b_1$  between 0 and 1 such that  $b_1 > a_1$ :

$$P(a_1 \leq u_n < b_1) = b_1 - a_1 \quad \forall b_1 > a_1$$

This is known as **1-distributivity property** of  $u_n$

- The **2-distributivity** is a generalization of this property in two dimensions:

$$P(a_1 \leq u_{n-1} < b_1 \text{ and } a_2 \leq u_n < b_2) = (b_1 - a_1)(b_2 - a_2)$$

For all choices of  $a_1, b_1, a_2, b_2$  in  $[0, 1]$ ,  $b_1 > a_1$  and  $b_2 > a_2$

## $k$ -Dimensional Uniformity or $k$ -Distribution

49

- **$k$ -distributed** if:

$$P(a_1 \leq u_{n-1} < b_1, \dots, a_k \leq u_{n+k-1} < b_k) = (b_1 - a_1) \cdots (b_k - a_k)$$

For all choices of  $a_i, b_i$  in  $[0, 1)$ , with  $b_i > a_i, i=1, 2, \dots, k$ .

- $k$ -distributed sequence is always  $(k-1)$ -distributed. The inverse is not true
- Two tests:
  - Serial test
  - Spectral test
- Visual test for 2-dimensions: plot successive overlapping pairs of numbers

6 November 2005

EEEC686/785

Wenbing Zhao

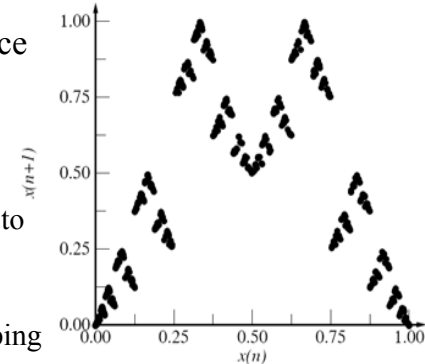
## Example 27.4

50

- Tausworthe sequence generated by:

$$x^{15} + x + 1$$

- The sequence is  $k$ -distributed for  $k$  up to  $\lceil q/l \rceil$ , that is,  $k=1$
- In two dimensions: successive overlapping pairs  $(x_n, x_{n+1})$



6 November 2005

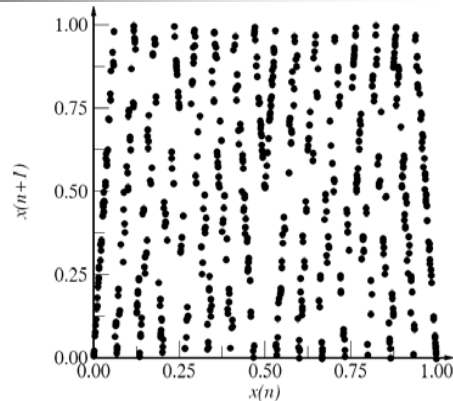
EEEC686/785

Wenbing Zhao

## Example 27.5

51

- Consider the polynomial:  $x^{15} + x^4 + 1$



6 November 2005

EEEC686/785

Wenbing Zhao

## Serial Test

52

- Goal: to test for uniformity in two dimensions or higher
- In two dimensions, divide the space between 0 and 1 into  $K^2$  cells of equal area
- Given  $\{x_1, x_2, \dots, x_n\}$ , use  $n/2$  nonoverlapping pairs  $(x_1, x_2), (x_3, x_4), \dots$  and count the points in each of the  $K^2$  cells

6 November 2005

EEEC686/785

Wenbing Zhao

## Serial Test

- Expected =  $n/(2K^2)$  points in each cell
- Use Chi-square test to find the deviation of the actual counts from the expected counts
- The degrees of freedom in this case area  $K^2 - 1$
- For  $k$ -dimensions: use  $k$ -tuples of nonoverlapping values

## Serial Test

- $k$ -tuples must be nonoverlapping
  - Overlapping  $\Rightarrow$  number of points in the cells are not independent. Chi-square test cannot be used
- In other visual checks, such as spectral test, overlapping tuples are used
- Given  $n$  numbers, there are  $n - 1$  overlapping pairs,  $n/2$  nonoverlapping pairs

## Spectral Test

- Goal:** to determine how densely the  $k$ -tuples  $\{x_1, x_2, \dots, x_n\}$  can fill up the  $k$ -dimensional hyperspace
- The  $k$ -tuples from an LCG all on a finite number of parallel hyperplanes
- Successive pairs would lie on a finite number of lines
- In three dimensions, successive triplets lie on a finite number of planes

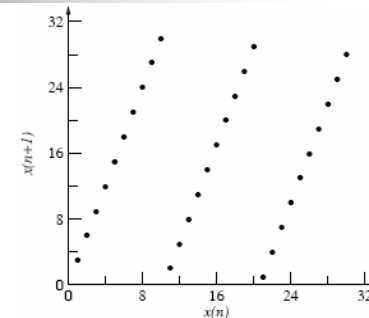
## Example 27.6

- $x_n = 3x_{n-1} \bmod 31$
- Plot of overlapping pairs
- All points lie on three straight lines
 
$$x_n = 3x_{n-1}$$

$$x_n = 3x_{n-1} - 31$$

$$x_n = 3x_{n-1} - 62$$

$$\Rightarrow x_n = 3x_{n-1} - 31k \quad k = 0,1,2 \quad (27.2)$$



## Example 27.6

- In three dimensions, the points for the above generator would lie on five planes given by:

$$x_n = 2x_{n-1} + 3x_{n-2} - 31k \quad k = 0,1,\dots,4$$

Obtained by adding the following to equation (27.2):

$$x_{n-1} = 3x_{n-2} - 31k_1 \quad k_1 = 0,1,2$$

Note that  $k+k_1$  will be an integer between 0 and 4

## Spectral Test (Continued)

- Marsaglia (1968): successive  $k$ -tuples obtained from an LCG fall on, at most,  $(k!m)^{1/k}$  parallel hyperplanes, where  $m$  is the modulus used in the LCG
- Example:  $m = 2^{32}$ , fewer than 2,953 hyperplanes will contain all 3-tuples, fewer than 566 hyperplanes will contain all 4-tuples, and fewer than 41 hyperplanes will contain all 10-tuples. Thus, this is a weakness of LCGs
- Spectral test**: determine the max distance between adjacent hyperplanes
  - > Larger distance => worse generator
- In some cases, it can be done by complete enumeration

## Example 27.7

- Compare the following two generators:

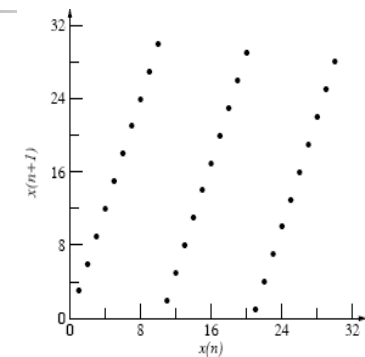
$$\begin{aligned} x_n &= 3x_{n-1} \bmod 31 \\ x_n &= 13x_{n-1} \bmod 31 \end{aligned}$$

- Using a seed of  $x_0 = 15$ , first generator: 14, 11, 2, 6, 18, 23, 7, 31, 1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14
- Using the same seed in the second generator: 9, 24, 2, 26, 28, 23, 20, 12, 1, 13, 14, 27, 10, 6, 16, 22, 7, 29, 5, 3, 8, 11, 19, 30, 18, 17, 4, 21, 25, 15, 9
- Every number between 1 and 30 occurs once and only once => both sequences will pass the Chi-square test for uniformity

## Example 27.7

- Three straight lines of positive slope or ten lines of negative slope
- Since the distance between the lines of positive slope is more, consider only the lines with positive slope:

$$\begin{aligned} x_n &= 3x_{n-1} \\ x_n &= 3x_{n-1} - 31 \\ x_n &= 3x_{n-1} - 62 \end{aligned}$$



## Example 27.7

- Distance between two parallel lines  $y = ax + c_1$  and  $y = ax + c_2$  is given by  $|c_2 - c_1| / \sqrt{1 + a^2}$
- The distance between the above lines is  $31/\sqrt{10}$  or 9.80

## Example 27.7

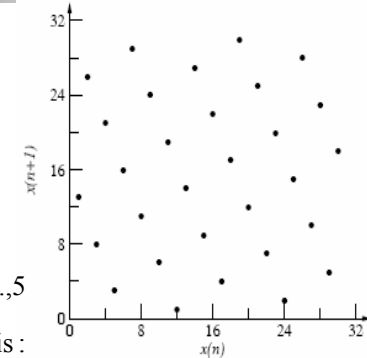
- All points fall on seven straight lines of positive slope or six straight lines of negative slope

- Considering lines with negative slopes:  

$$x_n = -\frac{5}{2}x_{n-1} + k\frac{31}{2}, k = 0, 1, \dots, 5$$

The distance between lines is:

$$(31/2) / \sqrt{1 + (5/2)^2} \text{ or } 5.76$$



## Example 27.7

- The second generator has a smaller maximum distance and, hence, the second generator has a better 2-distributivity
- The set with a larger distance may not always be the set with fewer lines
- Either overlapping or nonoverlapping  $k$ -tuples can be used
  - With overlapping  $k$ -tuples, we have  $k$  times as many points, which makes the graph visually more complete
  - The number of hyperplanes and the distance between them are the same with either choice

## Example 27.7

- With serial test, only nonoverlapping  $k$ -tuples should be used
- For generators with a large  $m$  and for higher dimensions, finding the maximum distance becomes quite complex