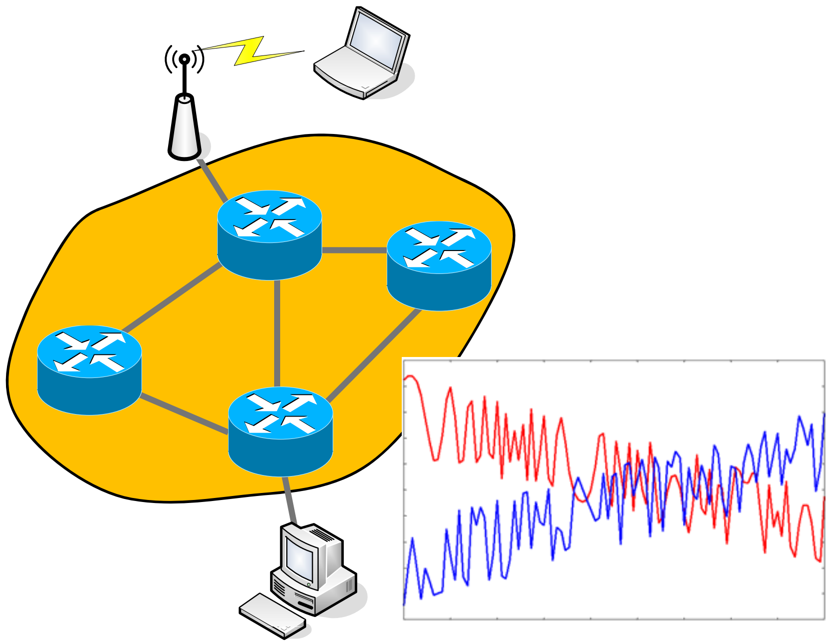
Chapter 6

Random-Number Generation



Contents

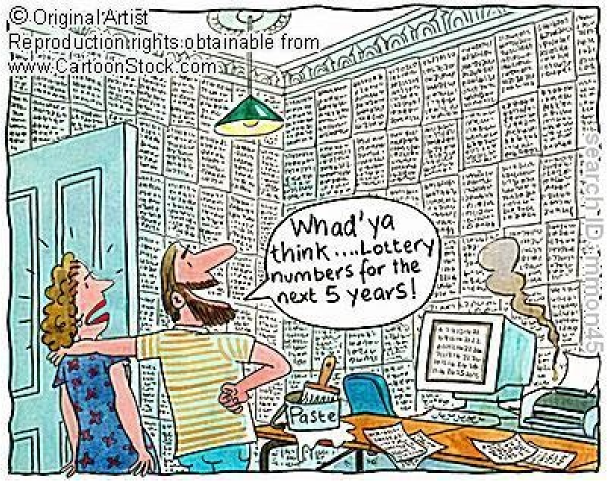
- Properties of Random Numbers
- Pseudo-Random Numbers
- Generating Random Numbers
  - Linear Congruential Method
  - Combined Linear Congruential Method
- Tests for Random Numbers
- Real Random Numbers

Overview

- Discuss characteristics and the generation of random numbers.
- Subsequently, introduce tests for randomness:
  - Frequency test
  - Autocorrelation test

Overview

- Historically
  - Throw dices
  - Deal out cards
  - Draw numbered balls
  - Use digits of π
  - Mechanical devices (spinning disc, etc.)
  - Electric circuits
    - Electronic Random Number Indicator (ERNIE)
  - Counting gamma rays
- In combination with a computer
  - Hook up an electronic device to the computer
  - Read-in a table of random numbers



Pseudo-Random Numbers

Pseudo-Random Numbers

- Approach: Arithmetically generation (calculation) of random numbers
- “Pseudo”, because generating numbers using a known method removes the potential for true randomness.

> *Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number — there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.*
>
> *John von Neumann, 1951*

Pseudo-Random Numbers

> *... probably ... can not be justified, but should merely be judged by their results. Some statistical study of the digits generated by a given recipe should be made, but exhaustive tests are impractical. If the digits work well on one problem, they seem usually to be successful with others of the same type.*
>
> *John von Neumann, 1951*

- Goal: To produce a sequence of numbers in [0,1] that simulates, or imitates, the ideal properties of random numbers (RN).

Pseudo-Random Numbers

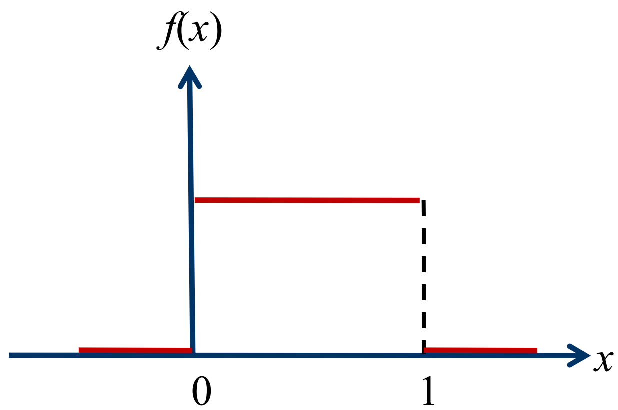
- Important properties of good random number routines:
  - Fast
  - Portable to different computers
  - Have sufficiently long cycle
  - Replicable
    - Verification and debugging
    - Use identical stream of random numbers for different systems
  - Closely approximate the ideal statistical properties of
    - uniformity and
    - independence

Pseudo-Random Numbers: Properties

- Two important statistical properties:
  - Uniformity
  - Independence
- Random number R_i must be independently drawn from a uniform distribution with PDF:

$$f(x) = \begin{cases} 1, & 0 \le x \le 1 \\ 0, & \text{otherwise} \end{cases}$$

$$E(R) = \int_0^1 x\,dx = \left.\frac{x^2}{2}\right|_0^1 = \frac{1}{2}$$



PDF for random numbers

Pseudo-Random Numbers

- Problems when generating pseudo-random numbers
  - The generated numbers might not be uniformly distributed
  - The generated numbers might be discrete-valued instead of continuous-valued
  - The mean of the generated numbers might be too high or too low
  - The variance of the generated numbers might be too high or too low

- There might be dependence:
  - Autocorrelation between numbers
  - Numbers successively higher or lower than adjacent numbers
  - Several numbers above the mean followed by several numbers below the mean

Generating Random Numbers

Generating Random Numbers

- Midsquare method
- Linear Congruential Method (LCM)
- Combined Linear Congruential Generators (CLCG)
- Random-Number Streams

Generating Random Numbers

Midsquare method

Midsquare method

- First arithmetic generator: Midsquare method
  - von Neumann and Metropolis in 1940s
- The Midsquare method:
  - Start with a four-digit positive integer Z_0
  - Compute: $Z_0^2 = Z_0 \times Z_0$ to obtain an integer with up to eight digits
  - Take the middle four digits for the next four-digit number

i	Z_i	U_i	$Z_i \times Z_i$
0	7182	-	51581124
1	5811	0.5811	33767721
2	7677	0.7677	58936329
3	9363	0.9363	87665769
…			

Midsquare method

- Problem: Generated numbers tend to 0

i	Z_i	U_i	$Z_i \times Z_i$
0	7182	-	51581124
1	5811	0,5811	33767721
2	7677	0,7677	58936329
3	9363	0,9363	87665769
4	6657	0,6657	44315649
5	3156	0,3156	09960336
6	9603	0,9603	92217609
7	2176	0,2176	04734976
8	7349	0,7349	54007801
9	78	0,0078	00006084
10	60	0,006	00003600
11	36	0,0036	00001296
12	12	0,0012	00000144
13	1	0,0001	00000001
14	0	0	00000000
15	0	0	00000000

... random numbers should not be generated with a method chosen at random. Some theory should be used.

Donald E. Knuth, The Art of Computer Programming, Vol. 2

Generating Random Numbers

Linear Congruential Method

Linear Congruential Method

- To produce a sequence of integers $X_1, X_2, \ldots$ between 0 and m-1 by following a recursive relationship:

$$X_{i+1} = (\underbrace{a}_{\text{The multiplier}} X_i + \underbrace{c}_{\text{The increment}}) \bmod \underbrace{m}_{\text{The modulus}}, \quad i = 0,1,2,\ldots$$

- Assumption: $0 < m \; and \; 0 \leq a, c, X_0 < m$
- The selection of the values for a, c, m, and X_0 drastically affects the statistical properties and the cycle length
- The random integers X_i are being generated in $[0, m\text{-}1]$

Linear Congruential Method

- Convert the integers X_i to random numbers

$$R_i = \frac{X_i}{m}, \quad i = 1,2,...$$

- Note:
  - $X_i \in \{0, 1, ..., m\text{-}1\}$
  - $R_i \in [0, (m\text{-}1)/m]$

Linear Congruential Method: Example

- Use $X_0 = 27, a = 17, c = 43$, and $m = 100$.
- The X_i and R_i values are:

$$
\begin{array}{lll}
X_1 = (17 \times 27 + 43) \bmod 100 = 502 \bmod 100 = 2 & \Rightarrow & R_1 = 0.02 \\
X_2 = (17 \times 2 \ \ + 43) \bmod 100 = 77 & \Rightarrow & R_2 = 0.77 \\
X_3 = (17 \times 77 + 43) \bmod 100 = 52 & \Rightarrow & R_3 = 0.52 \\
X_4 = \ (17 \times 52 + 43) \bmod 100 = 27 & \Rightarrow & R_3 = 0.27 \\
\ldots & &
\end{array}
$$

Linear Congruential Method: Example

- Use $a = 13, c = 0$, and $m = 64$
- The period of the generator is very low
- Seed X_0 influences the sequence

i	X_i $X_0=1$	X_i $X_0=2$	X_i $X_0=3$	X_i $X_0=4$
0	1	2	3	4
1	13	26	39	52
2	41	18	59	36
3	21	42	63	20
4	17	34	51	4
5	29	58	23	
6	57	50	43	
7	37	10	47	
8	33	2	35	
9	45		7	
10	9		27	
11	53		31	
12	49		19	
13	61		55	
14	25		11	
15	5		15	
16	1		3	

Linear Congruential Method:
Characteristics of a good Generator

- Maximum Density
  - The values assumed by R_i, $i=1,2,\ldots$ leave no large gaps on [0,1]
  - Problem: Instead of continuous, each R_i is discrete
  - Solution: a very large integer for modulus m
    - Approximation appears to be of little consequence

- Maximum Period
  - To achieve maximum density and avoid cycling
  - Achieved by proper choice of a, c, m, and X_0

- Most digital computers use a binary representation of numbers
  - Speed and efficiency are aided by a modulus, m, to be (or close to) a power of 2.

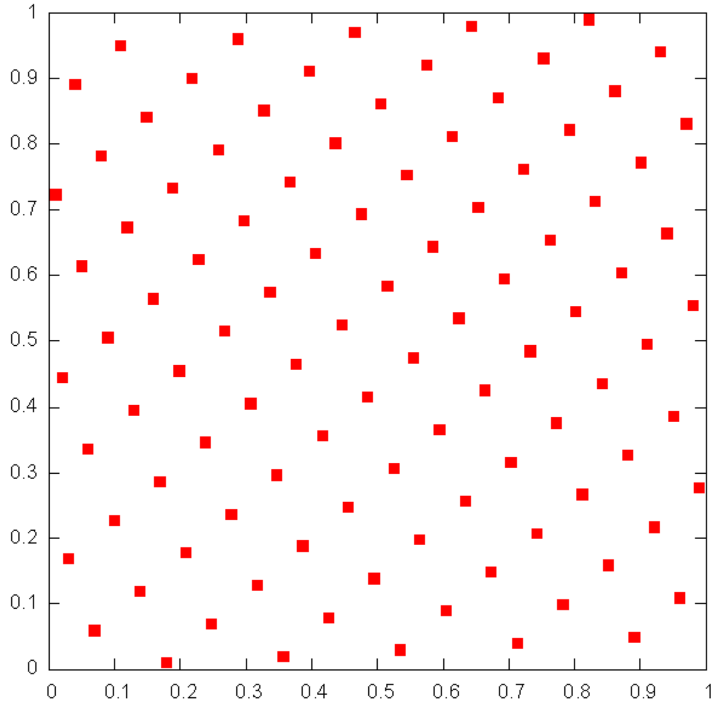
Linear Congruential Method:
Characteristics of a good Generator

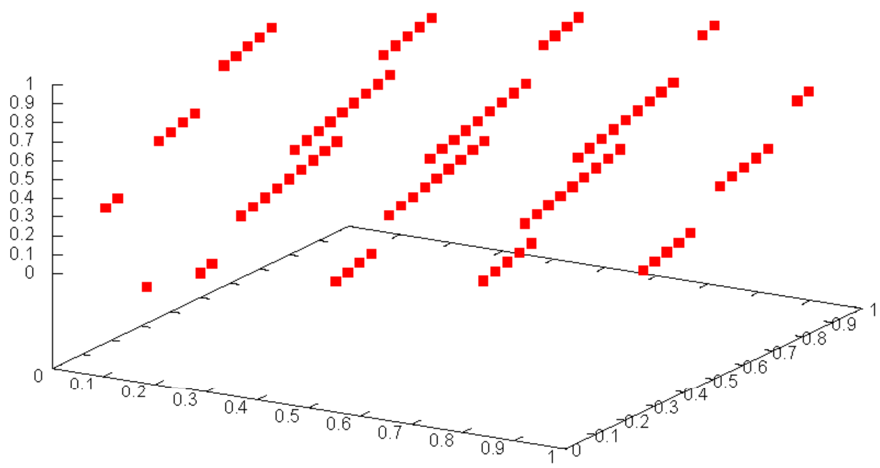
- The LCG has full period if and only if the following three conditions hold (Hull and Dobell, 1962):
  1. The only positive integer that (exactly) divides both m and c is 1
  2. If q is a prime number that divides m, then q divides a-1
  3. If 4 divides m, then 4 divides a-1

Linear Congruential Method:
Proper choice of parameters

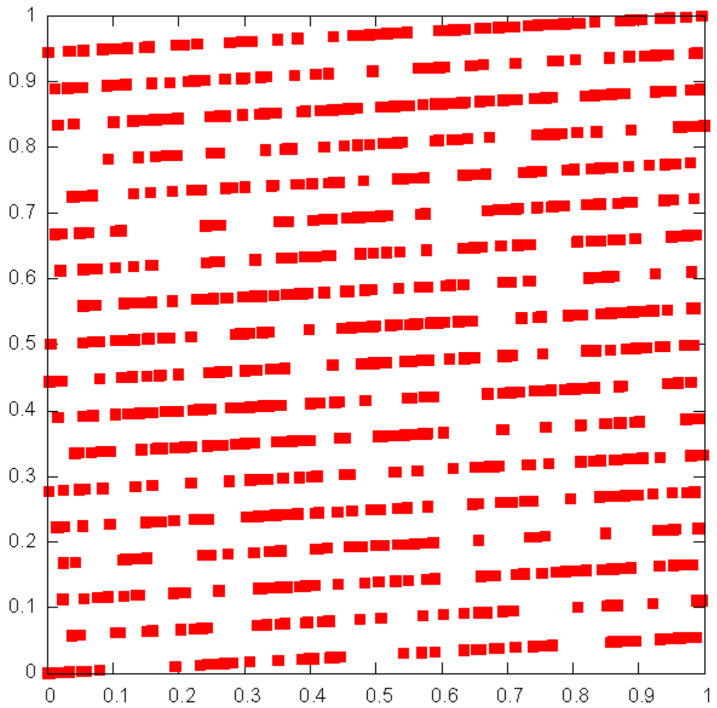
- For m a power 2, $m=2^b$, and $c \neq 0$
  - Longest possible period $P=m=2^b$ is achieved
    if c is relative prime to m and $a=1+4k$, where k is an integer
- For m a power 2, $m=2^b$, and $c=0$
  - Longest possible period $P=m/4=2^{b-2}$ is achieved
    if the seed X_0 is odd and $a=3+8k$ or $a=5+8k$, for $k=0,1,...$
- For m a prime and $c=0$
  - Longest possible period $P=m-1$ is achieved
    if the multiplier a has property that smallest integer k such that a^k-1 is divisible by m is $k = \mathrm{m}-1$

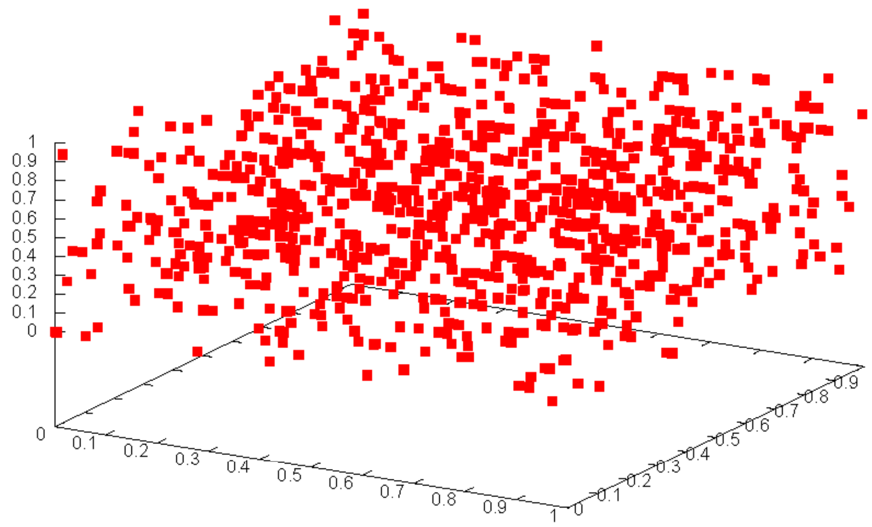
Characteristics of a Good Generator





Characteristics of a Good Generator





Random-Numbers in Java

- Defined in java.util.Random

```
private final static long multiplier = 0x5DEECE66DL; // 25214903917
private final static long addend = 0xBL;             // 11
private final static long mask = (1L << 48) - 1;     // 2^48-1 = 281474976710655

protected int next(int bits) {
    long oldseed, nextseed;
    ...
    oldseed = seed.get();
    nextseed = (oldseed * multiplier + addend) & mask;
    ...
    return (int)(nextseed >>> (48 - bits));    // >>> Unsigned right shift
}
```

General Congruential Generators

- Linear Congruential Generators are a special case of generators defined by:

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

- where $g()$ is a function of previous X_i's
  - $X_i \in [0, m\text{-}1]$, $R_i = X_i/m$

- Quadratic congruential generator
  - Defined by: $g(X_i, X_{i-1}) = aX_i^2 + bX_{i-1} + c$
- Multiple recursive generators
  - Defined by: $g(X_i, X_{i-1}, \ldots) = a_1 X_i + a_2 X_{i-1} + \cdots + a_k X_{i-k}$
- Fibonacci generator
  - Defined by: $g(X_i, X_{i-1}) = X_i + X_{i-1}$

Combined Linear Congruential Generators

- Reason: Longer period generator is needed because of the increasing complexity of simulated systems.
- Approach: Combine two or more multiplicative congruential generators.

- Let $X_{i,1}$, $X_{i,2}$, ..., $X_{i,k}$ be the i-th output from k different multiplicative congruential generators.
  - The j-th generator $X_{\bullet,j}$:

$$X_{i+1,j} = (a_j X_i + c_j) \bmod m_j$$

    - has prime modulus m_j, multiplier a_j, and period m_j-1
    - produces integers $X_{i,j}$ approx ~ Uniform on $[0, m_j - 1]$
    - $W_{i,j} = X_{i,j}$ - 1 is approx ~ Uniform on integers on $[0, m_j$ - $2]$

Combined Linear Congruential Generators

- Suggested form:

$$X_i = \left(\sum_{j=1}^{k} (-1)^{j-1} X_{i,j} \right) \bmod m_1 - 1 \qquad \text{Hence, } R_i = \begin{cases} \dfrac{X_i}{m_1}, & X_i > 0 \\ \dfrac{m_1 - 1}{m_1}, & X_i = 0 \end{cases}$$

- The maximum possible period is: $P = \dfrac{(m_1 - 1)(m_2 - 1)...(m_k - 1)}{2^{k-1}}$

Combined Linear Congruential Generators

- Example: For 32-bit computers, combining $k = 2$ generators with $m_1 = 2147483563$, $a_1 = 40014$, $m_2 = 2147483399$ and $a_2 = 40692$.
  The algorithm becomes:

  Step 1: Select seeds

  $X_{0,1}$ in the range $[1, 2147483562]$ for the 1st generator

  $X_{0,2}$ in the range $[1, 2147483398]$ for the 2nd generator

  Step 2: For each individual generator,

  $$X_{i+1,1} = 40014 \times X_{i,1} \bmod 2147483563$$

  $$X_{i+1,2} = 40692 \times X_{i,2} \bmod 2147483399$$

  Step 3: $X_{i+1} = (X_{i+1,1} - X_{i+1,2}) \bmod 2147483562$

  Step 4: Return

  $$R_{i+1} = \begin{cases} \dfrac{X_{i+1}}{2147483563}, & X_{i+1} > 0 \\[2ex] \dfrac{2147483562}{2147483563}, & X_{i+1} = 0 \end{cases}$$

  Step 5: Set $i = i+1$, go back to step 2.

- Combined generator has period: $(m_1 - 1)(m_2 - 1)/2 \sim 2 \text{ x } 10^{18}$

Random-Numbers in Excel 2003

- In Excel 2003 and 2007 new Random Number Generator

$$X, Y, Z \in \{1,...,30000\}$$

$$X = X \cdot 171 \bmod 30269$$

$$Y = Y \cdot 172 \bmod 30307$$

$$Z = Z \cdot 170 \bmod 30323$$

$$R = \left(\frac{X}{30269} + \frac{Y}{30307} + \frac{Z}{30323} \right) \bmod 1.0$$

- It is stated that this method produces more than 10^{13} numbers
- For more info: http://support.microsoft.com/kb/828795

Random-Numbers Streams

- The seed for a linear congruential random-number generator:
  - Is the integer value X_0 that initializes the random-number sequence
  - Any value in the sequence $(X_0, X_1, \ldots, X_P)$ can be used to "seed" the generator
- A random-number stream:
  - Refers to a starting seed taken from the sequence $(X_0, X_1, \ldots, X_P)$.
  - If the streams are b values apart, then stream i is defined by starting seed:

$$S_i = X_{b(i-1)} \qquad i = 1,2,\ldots,\left\lfloor \frac{P}{b} \right\rfloor$$

  - Older generators: $b = 10^5$
  - Newer generators: $b = 10^{37}$
- A single random-number generator with k streams can act like k distinct virtual random-number generators
- To compare two or more alternative systems.
  - Advantageous to dedicate portions of the pseudo-random number sequence to the same purpose in each of the simulated systems.

Random Numbers in OMNeT++

- OMNeT++ releases prior to 3.0 used a linear congruential generator (LCG) with a cycle length of $2^{31}-2$.
- By default, OMNeT++ uses the Mersenne Twister RNG (MT) by M. Matsumoto and T. Nishimura.
- MT has a period of $2^{19937}-1$, and 623-dimensional equidistribution property is assured.
- This RNG can be selected from omnetpp.ini
- OMNeT++ allows plugging in your own RNGs as well. This mechanism, based on the cRNG interface.

Tests for Random Numbers

Tests for Random Numbers

- Two categories:
  - Testing for **uniformity**:

$$H_0: \quad R_i \sim U[0,1]$$
$$H_1: \quad R_i \nsim U[0,1]$$

    - Failure to reject the null hypothesis, H_0, means that evidence of non-uniformity has not been detected.
  - Testing for **independence**:

$$H_0: \quad R_i \sim \text{independent}$$
$$H_1: \quad R_i \nsim \text{independent}$$

    - Failure to reject the null hypothesis, H_0, means that evidence of dependence has not been detected.

- Level of significance α, the probability of rejecting H_0 when it is true:

$$\alpha = P(\text{reject } H_0 \mid H_0 \text{ is true})$$

Tests for Random Numbers

- When to use these tests:
  - If a well-known simulation language or random-number generator is used, it is probably unnecessary to test
  - If the generator is not explicitly known or documented, e.g., spreadsheet programs, symbolic/numerical calculators, tests should be applied to many sample numbers.

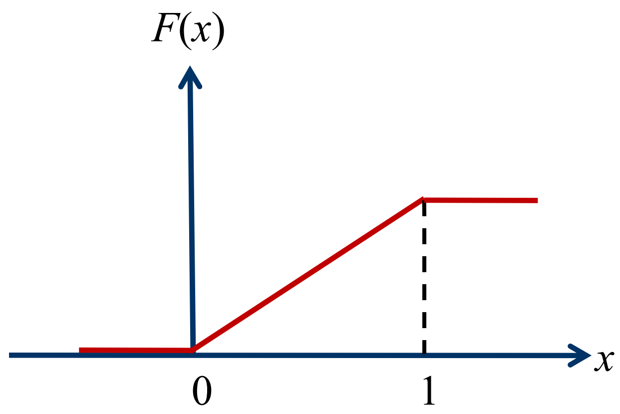
- Types of tests:
  - Theoretical tests: evaluate the choices of m, a, and c without actually generating any numbers
  - Empirical tests: applied to actual sequences of numbers produced.
    - Our emphasis.

Tests for Random Numbers

Frequency tests: Kolmogorov-Smirnov Test

Kolmogorov-Smirnov Test

- Compares the continuous CDF, $F(x)$, of the uniform distribution with the empirical CDF, $S_N(x)$, of the N sample observations.
  - We know: $F(x) = x, \quad 0 \le x \le 1$



  - If the sample from the RNG is $R_1, R_2, \ldots, R_N$, then the empirical CDF, $S_N(x)$ is:

$$S_N(x) = \frac{\text{Number of } R_i \text{ where } R_i \le x}{N}$$

- Based on the statistic: $D = max \mid F(x) - S_N(x)\mid$
  - Sampling distribution of D is known

Kolmogorov-Smirnov Test

- The test consists of the following steps
  - **Step 1:** Rank the data from smallest to largest
    $R_{(1)} \leq R_{(2)} \leq \ldots \leq R_{(N)}$
  - **Step 2:** Compute
    $$D^{+} = \max_{1 \leq i \leq N} \left\{ \frac{i}{N} - R_{(i)} \right\}$$
    $$D^{-} = \max_{1 \leq i \leq N} \left\{ R_{(i)} - \frac{i-1}{N} \right\}$$
  - **Step 3:** Compute $D = \max(D^{+}, D^{-})$
  - **Step 4:** Get D_{α} for the significance level α
  - **Step 5:** If $D \leq D_{\alpha}$ accept, otherwise reject H_0

Kolmogorov-Smirnov Critical Values

Degrees of Freedom (N)	$D_{0.10}$	$D_{0.05}$	$D_{0.01}$
1	0.950	0.975	0.995
2	0.776	0.842	0.929
3	0.642	0.708	0.828
4	0.564	0.624	0.733
5	0.510	$\boxed{0.565}$	0.669
6	0.470	0.521	0.618
7	0.438	0.486	0.577
8	0.411	0.457	0.543
9	0.388	0.432	0.514
10	0.368	0.410	0.490
11	0.352	0.391	0.468
12	0.338	0.375	0.450
13	0.325	0.361	0.433
14	0.314	0.349	0.418
15	0.304	0.338	0.404
16	0.295	0.328	0.392
17	0.286	0.318	0.381
18	0.278	0.309	0.371
19	0.272	0.301	0.363
20	0.264	0.294	0.356
25	0.24	0.27	0.32
30	0.22	0.24	0.29
35	0.21	0.23	0.27
Over 35	$\frac{1.22}{\sqrt{N}}$	$\frac{1.36}{\sqrt{N}}$	$\frac{1.63}{\sqrt{N}}$

Kolmogorov-Smirnov Test

- Example: Suppose N=5 numbers: 0.44, 0.81, 0.14, 0.05, 0.93.

Step 1: Arrange $R_{(i)}$ from smallest to largest

Step 2: $D^+ = max\{i/N - R_{(i)}\}$ and $D^- = max\{R_{(i)} - (i-1)/N\}$

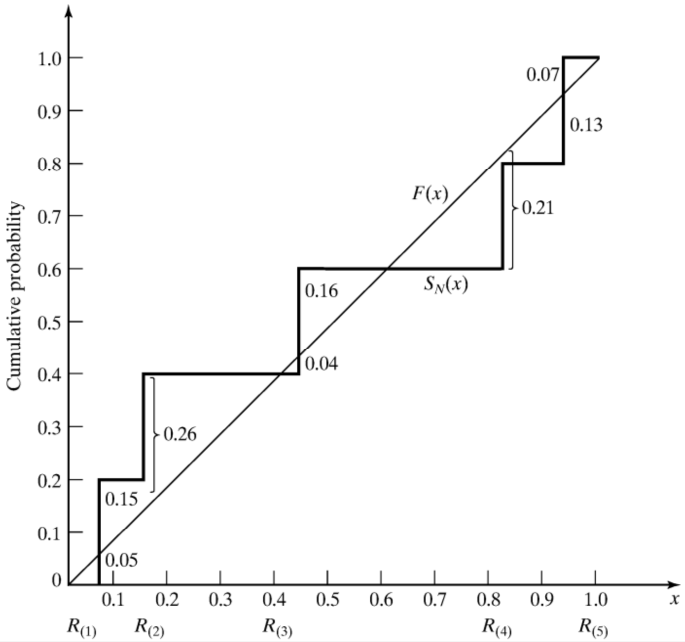
i	1	2	3	4	5
$R_{(i)}$	0.05	0.14	0.44	0.81	0.93
i/N	0.20	0.40	0.60	0.80	1.00
$i/N - R_{(i)}$	0.15	0.26	0.16	-	0.07
$R_{(i)} - (i-1)/N$	0.05	-	0.04	0.21	0.13

Step 3: $D = \max(D^+, D^-) = 0.26$

Step 4: For $\alpha = 0.05$,

$D_\alpha = 0.565 > D = 0.26$

Hence, H_0 is not rejected.



Tests for Random Numbers

Frequency tests: Chi-square Test

Chi-square Test

- Chi-square test uses the sample statistic:

n is the # of classes

O_i is the observed # in the i-th class

E_i is the expected # in the i-th class

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

  - Approximately the chi-square distribution with n-1 degrees of freedom
  - For the uniform distribution, E_i, the expected number in each class is:

$$E_i = \frac{N}{n}, \quad \text{where N is the total number of observations}$$

- Valid only for large samples, e.g., $N \geq 50$

Chi-square Test: Example

- Example with 100 numbers from [0,1], $\alpha=0.05$
- 10 intervals
- $\chi^2_{0.05,9} = 16.9$
- Accept, since
  - $X^2_0 = 11.2 < \chi^2_{0.05,9}$

Interval	Upper Limit	O_i	E_i	O_i-E_i	(O_i-E_i)^2	(O_i-E_i)^2/E_i
1	0.1	10	10	0	0	0
2	0.2	9	10	-1	1	0.1
3	0.3	5	10	-5	25	2.5
4	0.4	6	10	-4	16	1.6
5	0.5	16	10	6	36	3.6
6	0.6	13	10	3	9	0.9
7	0.7	10	10	0	0	0
8	0.8	7	10	-3	9	0.9
9	0.9	10	10	0	0	0
10	1.0	14	10	4	16	1.6
Sum		**100**	**100**	**0**	**0**	**11.2**

X^2_0=11.2

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

Tests for Random Numbers

Tests for autocorrelation

Tests for Autocorrelation

- Autocorrelation is concerned with dependence between numbers in a sequence
- Example:

0.12	0.01	0.23	0.28	0.89	0.31	0.64	0.28	0.83	0.93
0.99	0.15	0.33	0.35	0.91	0.41	0.60	0.27	0.75	0.88
0.68	0.49	0.05	0.43	0.95	0.58	0.19	0.36	0.69	0.87

- Numbers at 5-th, 10-th, 15-th, ... are very similar
- Numbers can be
  - Low
  - High
  - Alternating

Tests for Autocorrelation

- Testing the autocorrelation between every m numbers (m is a.k.a. the lag), starting with the i-th number
  - The autocorrelation $\rho_{i,m}$ between numbers: $R_i, R_{i+m}, R_{i+2m}, R_{i+(M+1)m}$
  - M is the largest integer such that $i+(M+1)m \leq N$

- Hypothesis:

$$H_0: \quad \rho_{i,m} = 0, \quad \text{if numbers are independent}$$
$$H_1: \quad \rho_{i,m} \neq 0, \quad \text{if numbers are dependent}$$

- If the values are uncorrelated:
  - For large values of M, the distribution of the estimator of $\rho_{i,m}$, denoted $\hat{\rho}_{i,m}$ is approximately normal.

Tests for Autocorrelation

- Correlation at lag j

$$\rho_j = \frac{C_j}{C_0}$$

$$C_j = Cov(X_i, X_{i+j}) = E(X_i X_{i+j}) - E(X_i)E(X_{i+j})$$

$$C_0 = Cov(X_i, X_i) = E(X_i X_i) - E(X_i)E(X_i) = E(X_i^2) - [E(X_i)]^2 = Var(X_i)$$

$$\Rightarrow \rho_j = \frac{E(X_i X_{i+j}) - E(X_i)E(X_{i+j})}{Var(X_i)}$$

- Assume $X_i = U_i$

$$E(U_i) = \frac{1}{2} \quad \text{and} \quad Var(U_i) = \frac{1}{12}$$

$$\rho_j = \frac{E(U_i U_{i+j}) - \frac{1}{4}}{\frac{1}{12}} = 12E(U_i U_{i+j}) - 3$$

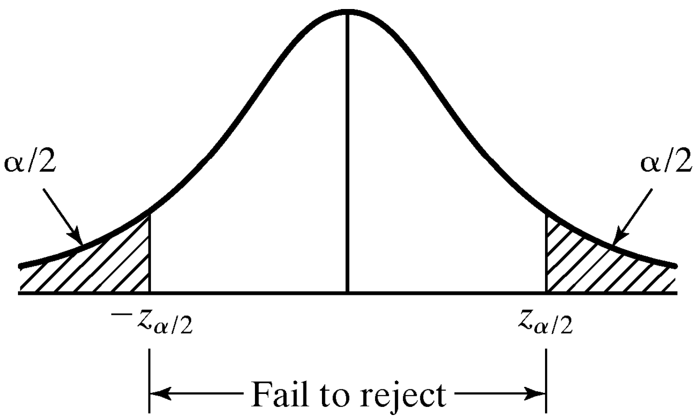
Tests for Autocorrelation

- Test statistics is: $$Z_0 = \frac{\hat{\rho}_{i,m}}{\hat{\sigma}_{\hat{\rho}_{i,m}}}$$
  - Z_0 is distributed normally with mean $= 0$ and variance $= 1$, and:

$$\hat{\rho}_{i,m} = \frac{1}{M+1}\left[\sum_{k=0}^{M} R_{i+km} \times R_{i+(k+1)m}\right] - 0.25$$

$$\hat{\sigma}_{\rho_{i,m}} = \frac{\sqrt{13M+7}}{12(M+1)}$$

- After computing Z_0 do not reject the hypothesis of independence if $-z_{\alpha/2} \leq Z_0 \leq -z_{\alpha/2}$



Tests for Autocorrelation

- If $\rho_{i,m} > 0$, the subsequence has positive autocorrelation
  - High random numbers tend to be followed by high ones, and vice versa.

- If $\rho_{i,m} < 0$, the subsequence has negative autocorrelation
  - Low random numbers tend to be followed by high ones, and vice versa.

Example

- Test whether the *3rd*, *8th*, *13th*, and so on, for the numbers on Slide 38.
  - Hence, $\alpha = 0.05, i = 3, m = 5, N = 30,$ and $M = 4$

$$\hat{\rho}_{35} = \frac{1}{4+1}\begin{bmatrix}(0.23)(0.28)+(0.28)(0.33)+(0.33)(0.27) \\ +(0.27)(0.05)+(0.05)(0.36)\end{bmatrix} - 0.25$$
$$= -0.1945$$
$$\sigma_{\hat{\rho}_{35}} = \frac{\sqrt{13(4)+7}}{12(4+1)} = 0.128$$
$$Z_0 = -\frac{0.1945}{0.1280} = -1.516$$

  - $z_{0.025} = 1.96$
  - Since $-1.96 \le Z_0 = -1.516 \le 1.96$, the hypothesis is not rejected.

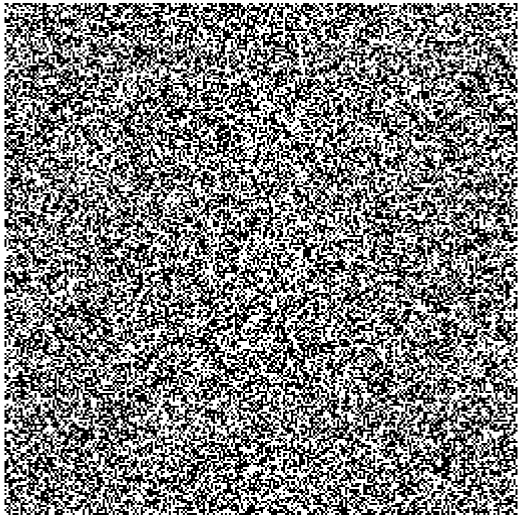
Shortcomings

- The test is not very sensitive for small values of M, particularly when the numbers being tested are on the low side.
- Problem when "fishing" for autocorrelation by performing numerous tests:
  - If $\alpha = 0.05$, there is a probability of 0.05 of rejecting a true hypothesis.
  - If 10 independence sequences are examined:
    - The probability of finding no significant autocorrelation, by chance alone, is $0.95^{10} = 0.60$.
    - Hence, the probability of detecting significant autocorrelation when it does not exist = 40%

Real Random Numbers

Real Random Numbers

- There are also sources for real random numbers in the Internet
- www.random.org
  „RANDOM.ORG offers *true* random numbers to anyone on the Internet. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs. People use the numbers to run lotteries, draws and sweepstakes and for their games and gambling sites."



http://www.random.org/analysis/

Real Random Numbers

- http://www.randomnumbers.info/
  „It offers the possibility to download true random numbers generated using a quantum random number generator upon demand. "

Real Random Numbers

- Hardware based generation of random numbers
- http://www.comscire.com

Summary

- In this chapter, we described:
  - Generation of random numbers
  - Testing for uniformity and independence
  - Sources of real random numbers

- Caution:
  - Even with generators that have been used for years, some of which still in use, are found to be inadequate.
  - This chapter provides only the basics
  - Also, even if generated numbers pass all the tests, some underlying pattern might have gone undetected.