Random-Number Generation

Rationale

- Simulators need generators of random variables with specific distributions
- In this way we can generate
  - Synthetic traces for the input processes of the simulator (e.g., arrival process, service times, …)
  - Aggregate behavior for part of the system that are not simulated in details (e.g., the channel error probability, propagation delay through sections of the network, …)

Rationale

- Generators should
  - Generate sequences of numbers whose statistical properties approximate with enough accuracy the theoretical ones
  - Make the experiments *repeatable*
  - Generate a very large number of different and uncorrelated instances
  - Be efficient
  - Be portable on different computers and programming languages

Rationale

- The actual procedure is:

1. We generate a sequence X of integer “random” numbers
2. Using X, we generate a sequence Z of instances of a random variable uniform in (0,1)
3. Using Z, we generate instances of the chosen random variable

Pseudo-random numbers

- Generators for *pseudo*-random sequences
  - The generation algorithm is deterministic
  - Knowing the algorithm parameters, we can regenerate the same sequence, so it is possible to repeat an experiment
  - The generated sequences *mimic* the random ones, since they have the same statistical properties of a sequence of instances of a uniform r.v.

Linear congruential generators

- They are the most commonly used
- Proposed by Lehmer in the 50s
- Based on the *module* operator
- They generate a sequence of integer numbers $X=\{X_i\}$ in the range $[0, m-1]$ through the iterative relation

$$X_{i+1} = (aX_i + c) \bmod m$$

Linear congruential generators

$$X_{i+1} = (aX_i + c) \bmod m$$

- X_0 : is called *seed*
- *a: multiplier*
- *c: increment*
  - $c = 0$ in *multiplicative congruential* generators
  - $c \neq 0$ in *mixed congruential* generators
- *m: module*

Sequences in [0,1]

- Given a generator of sequences X of integer numbers in [*0,m-1*], a sequence Z of values in [0,1) or in [0,1] is generated from

$$Z_i = X_i / m \quad \text{or} \quad Z_i = X_i / (m-1)$$

- Depending on the normalization used, the extremes, 0 and 1, can either be included or not

Linear congruential generators

- **Sequences are not random**

  The algorithm is deterministic; however, the generated sequences *look* random, like a sequence of instances uniformly and independently distributed

- **The sequence cannot assume all the values in (0,1)**

  Values different from i/m are not possible, but for m large enough the sequence is very dense

Linear congruential generators

- The choice of the generator parameters (X_0,a,c,m) strongly influences the properties of the generated sequence
- Different generators have different stochastic characteristics making their behavior more or less similar to the one of uniform variables
  - A generator with module m should generate integers uniformly distributed in [0,m-1]

Sequence period

- The length of the generated sequence is called *period*
- A generator with module m has *maximum period* equal to
  - m if $c \neq 0$
  - m-1 if $c = 0$ (the value 0 is not possible)
- Not all the generators can achieve a maximum period

Linear congruential generators: Example 1

- $a=7$, $c=0$, $m=11$, $X_0=1$
- Generated sequence:

  1, 7, 5, 2, 3, 10, 4, 6, 9, 8, 1, 7, 5

- The sequence contains all the possible integer numbers in [1,10]: it has the maximum period, equal to m-1
- As soon as the seed value is generated again, the sequence repeats

Linear congruential generators: Example 2

- **a=5**, c=0, m=11, X_0=1
- Generated sequence:

  1, 5, 3, 4, 9, 1, 5, 3

- The length of the sequence is 5, less than m-1=10
- Some numbers never appear

Linear congruential generators: Example 3

- a=5, c=0, m=11, $\mathbf{X_0=6}$
- Generated sequence:

6, 8, 7, 2, 10, 6, 8,

- Starting from a seed not contained in the previous sequence, we obtain numbers not generated in the previous sequence

Characteristics of the generated sequences

- Given *m*, in previous examples we could expect the average value of the sequence being

$$\frac{1}{m-1}\sum_{i=1}^{m-1} X_i \rightarrow \frac{m(m-1)}{2(m-1)} = \frac{m}{2} = 5.5$$

  - Example 1: mean = 5.5 (maximum period)
  - Example 2: mean = 4.4
  - Example 3: mean = 6.6

Criteria for the parameters selection

Case $m = 2^b$, $c \neq 0$

- the maximum period is m if
  - c and m are reciprocal prime (their only common factor is 1)
  - $a = 1 + 4k$, with k integer

Case $m = 2^b$, $c = 0$

- the maximum period is $m/4 = 2^{b-2}$ if
  - X_0 is odd
  - $a = 3 + 8k$ or $a = 5 + 8k$, with k integer

Criteria for the parameters selection

Case m prime and $c = 0$

- the maximum period is m-1 if
  - the smallest integer k such that a^k-1 is divisible by m is $k = m - 1$

Characteristics of the generated sequences

- In previous examples, since *m* is prime and *c=0*, the maximum period is
  - Example 1: *m*-1=10, because for *a* =7 the smallest *k* such that a^k -1 is divisible by *m*=11 is *k*=10, equal to *m*-1
  - Examples 2 e 3: period is not the maximum because for *a*=5 the smallest *k* such that a^k -1 is divisible by *m*=11 is *k*=5, smaller than *m*-1

Computational issues

- Linear congruential generators may suffer from the following issues
  - The used programming language must perform the operations with integer math (no rounding)
  - The product aX_i may overflow, since its representation might exceed the internal used integer representation of a and X_i alone

Other congruential generators

- A few techniques permit to increase the period of the sequence
- General congruential generator are in the form

$$X_i = g(X_{i-1}, X_{i-2}, \cdots) \bmod m$$

where g() is a generic function

Other congruential generators

- Example 1: *quadratic* generator

$$X_i = (a_2 X_{i-1}^2 + a_1 X_{i-1} + c) \bmod m$$

with a_1=a_2=1, c=0, m=2^b it has good proprieties

- Example 2: *Fibonacci* generator

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

period larger than m but bad stochastic characteristics

“Good” generators

- C Language
  - $a = 7^5 = 16{,}807$ -- $c = 0$
    $m = 2^{31} - 1 = 2{,}147{,}483{,}647$ (prime)
- JAVA java.util.Random
  - $a=25{,}214{,}903{,}917$ -- $c=11$ -- $m=2^{48}$
  - 32-bit values are returned after math operations on the series
- Visual Basic
  - $a=1{,}140{,}671{,}485$ -- $c=12{,}820{,}163$ -- $m=2^{24}$

Good generators

- Python uses the Mersenne Twister as the core generator
  - It is based on a series of bitwise (very easy to compute) XOR, AND, and shifting operations that are mathematically equivalent to operations on a twist matrix
  - It produces sequences with a period of $2^{19937}-1$ – extremely long period

Good generators

- The Mersenne Twister is one of the most extensively tested random number generators in existence
- It has better properties, and is nearly as efficient to compute

Tests of RNGs

- Each number in the sequence Z_i should be an instance of a r.v. with uniform distribution in (0,1)
- The sequence should have two properties:
  - **Uniformity**
  - **Indipendence**
- There are several tests used to verify these properties
- After several tests, some RNGs are considered better than others or weaknesses are identified

Wrap-up

- Simulations need instances of random variables
- To derive them we need a random number generator with
  - Good properties
  - Long sequences
  - Efficient to generate instances