Scientific Computing: An Introductory Survey

Chapter 13 – Random Numbers and Stochastic Simulation

Prof. Michael T. Heath

Department of Computer Science
University of Illinois at Urbana-Champaign

Stochastic Simulation

- *Stochastic simulation* mimics or replicates behavior of system by exploiting randomness to obtain statistical sample of possible outcomes
- Because of randomness involved, simulation methods are also known as *Monte Carlo* methods
- Such methods are useful for studying
  - Nondeterministic (stochastic) processes
  - Deterministic systems that are too complicated to model analytically
  - Deterministic problems whose high dimensionality makes standard discretizations infeasible (e.g., Monte Carlo integration)

< interactive example >   < interactive example >

Stochastic Simulation, continued

- Two main requirements for using stochastic simulation methods are
  - Knowledge of relevant probability distributions
  - Supply of random numbers for making random choices
- Knowledge of relevant probability distributions depends on theoretical or empirical information about physical system being simulated
- By simulating large number of trials, probability distribution of overall results can be approximated, with accuracy attained increasing with number of trials

< interactive example >   < interactive example >

Randomness

- *Randomness* is somewhat difficult to define, but we usually associate randomness with unpredictability
- One definition is that sequence of numbers is *random* if it has no shorter description than itself
- Physical processes, such as flipping coin or tossing dice, are deterministic if enough is known about equations governing their motion and appropriate initial conditions
- Even for deterministic systems, extreme sensitivity to initial conditions can make their chaotic behavior unpredictable in practice
- Wheter deterministic or not, highly complicated systems are often tractable only by stochastic simulation methods

Repeatability

- In addition to unpredictability, another distinguishing characteristic of true randomness is lack of *repeatability*
- However, lack of repeatability could make testing algorithms or debugging computer programs difficult, if not impossible
- Repeatability is desirable in this sense, but care must taken to ensure independence among trials

Pseudorandom Numbers

- Although random numbers were once supplied by physical processes or tables, they are now produced by computers
- Computer algorithms for generating random numbers are in fact deterministic, although sequence generated may *appear* random in that it exhibits no apparent pattern
- Such sequences of numbers are more accurately called *pseudorandom*
- Although pseudorandom sequence may appear random, it is in fact quite predictable and reproducible, which is important for debugging and verifying results
- Because only finite number of numbers can be represented in computer, any sequence must eventually repeat

Random Number Generators

Properties of good random number generator as possible

- *Random pattern*: passes statistical tests of randomness
- *Long period*: goes as long as possible before repeating
- *Efficiency*: executes rapidly and requires little storage
- *Repeatability*: produces same sequence if started with same initial conditions
- *Portability*: runs on different kinds of computers and is capable of producing same sequence on each

Random Number Generators, continued

- Early attempts at producing random number generators on computers often relied on complicated procedures whose very complexity was presumed to ensure randomness
- Example is “midsquare” method, which squares each member of sequence and takes middle portion of result as next member of sequence
- Lack of theoretical understanding of such methods proved disastrous, and it was soon recognized that simple methods with well-understood theoretical basis are far preferable

Congruential Generators

- *Congruential* random number generators have form

$$x_k = (a x_{k-1} + c) \pmod{M}$$

  where a and c are given integers

- Starting integer x_0 is called *seed*
- Integer M is approximately (often equal to) largest integer representable on machine
- Quality of such generator depends on choices of a and c, and in any case its period cannot exceed M

Congruential Generators, continued

- It is possible to obtain reasonably good random number generator using this method, but values of a and c must be chosen *very* carefully
- Random number generators supplied with many computer systems are of congruential type, and some are notoriously poor
- Congruential generator produces random integers between 0 and M
- To produce random floating-point numbers, say uniformly distributed on interval $[0, 1)$, random integers must be divided by M (*not* integer division!)

< interactive example >

Fibonacci Generators

- *Fibonacci* generators produce floating-point random numbers on interval $[0, 1)$ directly as difference, sum, or product of previous values
- Typical example is subtractive generator

$$x_k = x_{k-17} - x_{k-5}$$

- This generator is said to have *lags* of 17 and 5
- Lags must be chosen carefully to produce good subtractive generator
- Such formula may produce negative result, in which case remedy is to add 1 to get back into interval $[0, 1)$

Fibonacci Generators, continued

- Fibonacci generators require more storage than congruential generator, and also require special procedure to get started
- Fibonacci generators require no division to produce floating-point results
- Well-designed Fibonacci generators have very good statistical properties
- Fibonacci generators can have much longer period than congruential generators, since repetition of one member of sequence does not entail that all subsequent members will also repeat in same order

Sampling on Other Intervals

- If we need uniform distribution on some other interval $[a, b)$, then we can modify values x_k generated on $[0, 1)$ by transformation

$$(b - a)x_k + a$$

  to obtain random numbers that are uniformly distributed on desired interval

Nonuniform Distributions

- Sampling from nonuniform distributions is more difficult
- If cumulative distribution function of desired probability density function is easily invertible, then we can generate random samples with desired distribution by generating uniform random numbers and inverting them
- For example, to sample from exponential distribution

  $$f(t) = \lambda e^{-\lambda t}, \quad t > 0$$

  we can take

  $$x_k = -\log(1 - y_k)/\lambda$$

  where y_k is uniform on $[0, 1)$
- Unfortunately, many important distributions are not easily invertible, and special methods must be employed to generate random numbers efficiently for these distributions

Normal Distribution

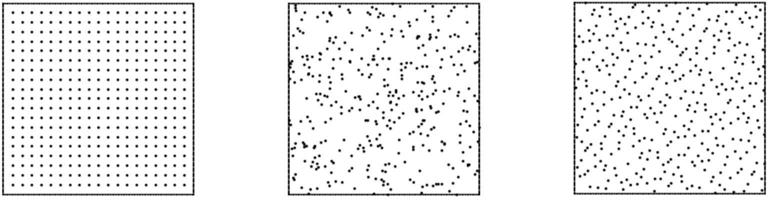
- Important example is generation of random numbers that are normally distributed with given mean and variance
- Available routines often assume mean 0 and variance 1
- If some other mean μ and variance σ^2 are desired, then each value x_k produced by routine can be modified by transformation $\sigma x_k + \mu$ to achieve desired normal distribution

Quasi-Random Sequences

- For some applications, achieving reasonably uniform coverage of sampled volume can be more important than whether sample points are truly random
- Truly random sequences tend to exhibit random clumping, leading to uneven coverage of sampled volume for given number of points
- Perfectly uniform coverage can be achieved by using regular grid of sample points, but this approach does not scale well to higher dimensions
- Compromise between these extremes of coverage and randomness is provided by *quasi-random* sequences

Quasi-Random Sequences, continued

- Quasi-random sequences are not random at all, but are carefully constructed to give uniform coverage of sampled volume while maintaining reasonably random appearance
- By design, points tend to avoid each other, so clumping associated with true randomness is eliminated



Grid Random Quasi-random

< interactive example >