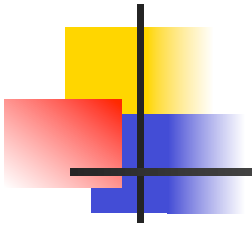
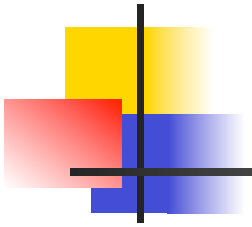


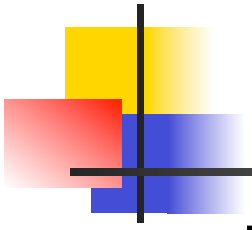
Computational Methods

Randomness and Monte Carlo Methods



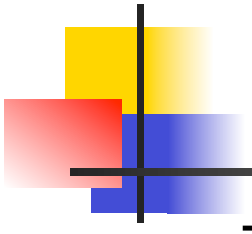
Randomness and Monte Carlo Methods

- Introducing randomness in an algorithm can lead to improved efficiencies
  - Random sampling can provide probabilistically good results with relatively few samples
- Many random algorithms use stochastic simulation as part of their computation – Monte Carlo Methods
  - Exploit randomness to obtain statistical sample of outcomes
- Monte Carlo methods are particularly useful to study
  - Nondeterministic systems
  - Deterministic systems that are too complicated to model
  - Deterministic problems too high dimensional for discretization



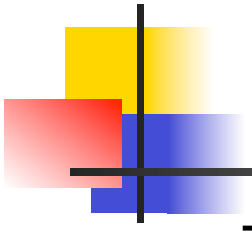
Monte Carlo Methods

- Monte Carlo methods randomly draw samples from a distribution and determining values for each sample
  - Monte Carlo for expected value problems
    - Sample from the distribution and average the function values at the samples to get the expected value over the given distribution
  - Monte Carlo for ratio problems
    - Sample from a distribution and determine the ratio of valid vs. invalid samples to compute the desired ratio
- Monte Carlo methods provide increasingly precise solutions as the number of samples increases but require
  - Knowledge of relevant probability distribution and function
  - Access to random numbers



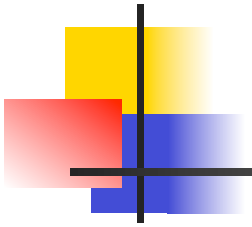
Randomness

- True randomness is difficult to determine
  - Many processes we consider random are not random but rather too complicated for us to model (or measure) accurately
  - Sometimes modeling processes that we know are deterministic as stochastic can increase the efficiency of representing or solving them (sometimes making solving possible)
- Randomness is often defined in terms of
  - Uncompressability
    - The random sequence is the shortest description of itself
  - Unpredictability
    - The next random number is not predictable from the previous ones
  - Not repeatable
    - Random sequences do not repeat (might not always be desirable)



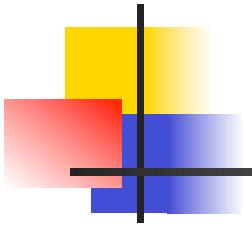
Random Number Generators

- To be used, the computer needs access to random numbers
  - True random number generators
    - To generate true random numbers, physical processes can be used
      - Radioactive decay
    - Tables with true random sequences can be (and have been) used
  - Pseudo-random number generators
    - Random numbers are generated using a deterministic algorithm
    - Sequence of numbers appears random without knowledge of the algorithm
      - Pseudo-random numbers are predictable if the algorithm is known
      - Pseudo-random numbers are repeatable and reproducible
      - Pseudo-random number sequences will eventually repeat
  - Quasi-random number generators
    - Quasi-random numbers sacrifice randomness of points and focuses on the uniformity of the sample sequence



Random Number Generators

- Random number generators should ideally fulfill a number of requirements
  - Pattern of random numbers should appear random
    - Pass statistical tests of randomness
  - Random numbers should not repeat for a long time
    - Long period in generation algorithm
  - Next random number should be available fast
    - Highly efficient algorithm (and implementation)
  - Random number sequence should be repeatable
    - Production of the same sequence form same initial conditions allows for better comparisons and evaluations
  - Generator should be portable across systems



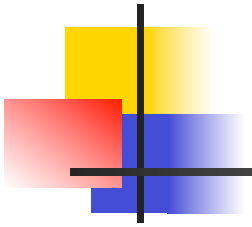
Pseudo-Random Numbers

- A range of pseudo-random number generators are used, including
  - Congruential random number generator
    - Use a very simple equation to calculate the next pseudo-random number (as a Natural number) based on the previous pseudo-random number

$$x_{k+1} = (ax_k + c) \mod m \quad , \quad u_{k+1} = x_{k+1}/m$$

      - Once a number repeats, the entire sequence repeats
  - Fibonacci generator
    - Next pseudo-random number is generated directly as a real number based on two previous pseudo-random numbers (as product, sum, difference, ...)

$$x_{k+1} = \begin{cases} x_{k-l_1} - x_{k-l_2} + 1 & if \quad x_{k-l_1} - x_{k-l_2} < 0 \\ x_{k-l_1} - x_{k-l_2} - 1 & if \quad x_{k-l_1} - x_{k-l_2} > 1 \\ x_{k-l_1} - x_{k-l_2} + 1 & otherwise \end{cases}$$

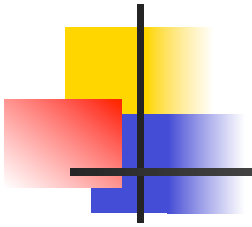


Congruential Generator

- Congruential random number generators are a very common type of generator.

$$x_{k+1} = (ax_k + c) \mod m \quad , \quad u_{k+1} = x_{k+1}/m$$

  - Performance depends on the choice of parameters a, c, and m
    - m determines the range of numbers that the random number generator can generate
    - Non-careful choice of a, b, and m can lead to statistically biased random number sequences
      - One example of this is the randu generator used in early IBM computers: a=65539, b=0, $m=2^{31}$
  - m is often chosen as the maximum representable number
    - Minimizes repetition



Fibonacci Generator

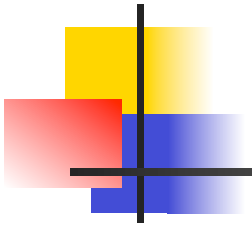
- Fibonacci generators and their variations are replacing congruential pseudo-random number generators.
  - Fibonacci generators can directly generate floating point numbers as difference, sum, or product of previous numbers

$$x_k = (x_{k-i} \circ x_{k-j}) \ \ MOD \ \ m$$

$$e.g.: \quad x_k = (x_{k-i} - x_{k-j}) \ \ MOD \ \ 1$$

$$x_k = (x_{k-i} * x_{k-j}) \ \ MOD \ \ k$$

    - MOD operation ensures that numbers stay within the required range
  - Performance depends on the choice of parameters i, j, m
    - Common choice for a subtractive generator are i=17, j=5, m=1
  - Performance also depends on choice of initial elements



Nonuniform Distributions

- Using a random number generator for uniform random numbers, a number of other distributions can be obtained
  - Shifted uniform distribution: To generate a uniform distribution in interval [a,b) we can simply transform a uniform random number in the interval [0,1)

$$x = (b-a)u + a$$

  - To achieve another distribution p(x) we can use its cumulative distribution P(x) and a uniform random number such that u=P(x), i.e. $x = P^{-1}(u)$
  - Exponential distribution: $p(x) = \lambda e^{-\lambda x}$

$$x = \frac{-\log(1-u)}{\lambda}$$

Convergence Rates of Monte Carlo Methods

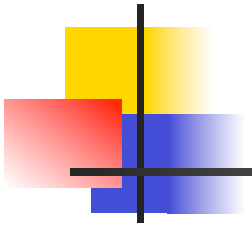
- Monte Carlo Simulations with pseudo-random numbers converge with the inverse of the square root of the number of samples

$$Error \propto 1/\sqrt{n}$$

  - This is the result of the way the expected value of the variance changes

$$E\left[\left(\frac{x_1 + \ldots + x_n}{n} - \hat{x}\right)^2\right] = \frac{1}{n^2}\sum E\left[\left(x_i - \hat{x}\right)^2\right] = \frac{1}{n^2} n\sigma^2 = \frac{\sigma^2}{n}$$

$$\sqrt{E\left[\left(\frac{x_1 + \ldots + x_n}{n} - \hat{x}\right)^2\right]} \propto \frac{1}{\sqrt{n}}$$



Quasi-Random Numbers

- Quasi-Random number generators generate numbers that uniformly cover the space but do not individually appear random
  - Consecutive numbers are not unbiased
- Base-p low-discrepancy sequence
  - Generate quasi-random numbers by inverting the digit sequence of consecutive base-p natural numbers and adding them after the decimal point of a base-p fixed point number with 0 before the decimal point
    - E.g. base 2: 0.0 , 0.01 , 0.11, 0.001, 0.101, 0.011, 0.111, …
    - Base 3 is also called Halton sequence

Convergence Rates of Monte Carlo Methods

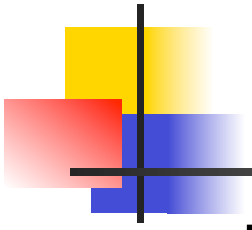
- Monte Carlo Simulations with quasi-random numbers converge at a different rate that depends on the dimensionality of the random number, d
  - Monte Carlo for expected value problems:

$$Error \propto (\ln n)^d / n$$

  - Monte Carlo for ratio problems:

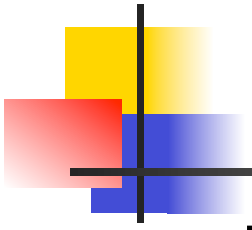
$$Error \propto 1 / n^{\frac{1}{2}+\frac{1}{2d}}$$

- Monte Carlo methods converge faster with quasi-random numbers than with pseudo-random numbers



Pseudo-Random vs. Quasi-Random Numbers

- Pseudo-Random number generators are usually designed such that the sequence of numbers are uncorrelated and pass a number of statistical independence tests. Quasi-Random numbers will usually fail these sequence tests and show high sequence correlations.
- Quasi-Random numbers lead to faster convergence in Monte Carlo methods where only the uniformity of the distribution matters but not the randomness of the actual sequence.
  - E.g. Monte-Carlo integration
- Pseudo-Random numbers perform much better in situations where the randomness of the sequence of numbers matters
  - E.g. Monte-Carlo simulation of random processes



Randomness

- Randomization in sample-based problem solutions can increase the performance in very complex problems
  - Random sampling has statistical properties that deterministic samples can not guarantee
- To use randomization it is necessary to generate random numbers that have particular properties
  - Pseudo-Random numbers are aimed at generating random sequences where the sequential samples appear as if they are independently generated from a particular random distribution
  - Quasi-Random numbers are aimed at generating samples that overall follow a particular distribution but might be correlated as a sequence