Monte-Carlo Techniques

Roger Crawfis

Monte-Carlo Integration

- Overview
  1. Generating Psuedo-Random Numbers
  2. Multidimensional Integration
     a) Handling complex boundaries.
     b) Handling complex integrands.

Pseudo-Random Numbers

- Definition of random from Merriam-Webster:
- Main Entry: **random**
  Function: *adjective*
  Date: 1565
  1 a : lacking a definite plan, purpose, or pattern **b :** made, done, or chosen at random <read *random* passages from the book>
  2 a : relating to, having, or being elements or events with definite probability of occurrence <*random* processes> **b :** being or relating to a set or to an element of a set each of whose elements has equal probability of occurrence <a *random* sample>; *also* **:** characterized by procedures designed to obtain such sets or elements <*random* sampling>

Random Computer Calculations?

- Compare this to the definition of an algorithm (dictionary.com):
  - **algorithm**
    - n : a precise rule (or set of rules) specifying how to solve some problem.

Random Number

- What is random number ? Is 3 ?
  - There is no such thing as single random number
- Random number
  - A set of numbers that have nothing to do with the other numbers in the sequence
- In a uniform distribution of random numbers in the range [0,1] , every number has the same chance of turning up.
  - 0.00001 is just as likely as 0.5000

Random v. Pseudo-random

- **Random numbers** have no defined sequence or formulation. Thus, for any *n* random numbers, each appears with equal probability.
- If we restrict ourselves to the set of 32-bit integers, then our numbers will start to repeat after some very large *n*. The numbers thus clump within this range and around these integers.
- Due to this limitation, computer algorithms are restricted to generating what we call **pseudo-random numbers**.

Monte-Carlo Methods

- 1953, Nicolaus Metropolis
- Monte Carlo method refers to any method that makes use of random numbers
  - Simulation of natural phenomena
  - Simulation of experimental apparatus
  - Numerical analysis

How to generate random numbers ?

- Use some chaotic system (Balls in a barrel – Lotto)
- Use a process that is inherently random
  - Radioactive decay
  - Thermal noise
  - Cosmic ray arrival
- Tables of a few million random numbers
- Hooking up a random machine to a computer.

Pseudo Random number generators

- The closest random number generator that can be obtained by computer algorithm.
- Usually a uniform distribution in the range [0,1]
- Most pseudo random number generators have two things in common
  - The use of large prime numbers
  - The use of modulo arithmetic
- Algorithm generates integers between 0 and M, map to zero and one.

$$X_n = I_n / M$$

An early example (John Von Neumann,1946)

- To generate 10 digits of integer
  - Start with one of 10 digits integers
  - Square it and take middle 10 digits from answer
  - Example: $5772156649^2 = 33317\underline{7923805949}09291$
- The sequence appears to be random, but each number is determined from the previous → not random.
- Serious problem : Small numbers (0 or 1) are lumped together, it can get itself to a short loop. For example:
  - $6100^2 = 37\underline{2100}00$
  - $2100^2 = 04\underline{4100}00$
  - $4100^2 = 16\underline{8100}00$
  - $5100^2 = 65\underline{6100}00$

Linear Congruential Method

- Lehmer, 1948
- Most typical so-called random number generator
- Algorithm : $I_{n+1} = (aI_n + c)\,\mathrm{mod}(m)$
  - *$a,c >= 0$, $m > I_0, a, c$*
- *Advantage :*
  - *Very fast*
- *Problem :*
  - *Poor choice of the constants can lead to very poor sequence*
  - *The relationship will repeat with a period no greater than m (around m/4)*
    - *C complier RAND_MAX : m = 32767*

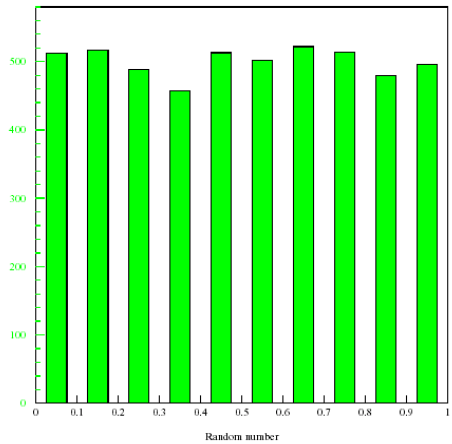
RANDU Generator

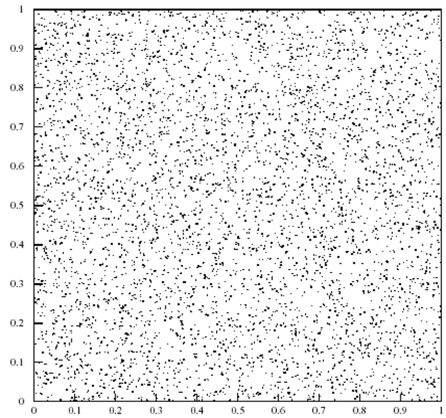
- 1960's IBM
- Algorithm

$$I_{n+1} = (65539 \times I_n)\,\mathrm{mod}(2^{31})$$

- This generator was later found to have a serious problem

1D and 2D Distribution of RANDU





Random Number Algorithms

- The class of multiplicative congruential random-number generators has the form: . The choice of the coefficients is critical. Example in book:

$$x_n = \frac{l_n}{2^{31}-1}$$
$$l_{n+1} = \left(7^5 l_n\right) \bmod \left(2^{31}-1\right)$$
$$l_0 = 1$$
$$l_1 = 7^5 \quad \Rightarrow x_1 = 0.0000078263692594256108903445354152213\text{e-6}$$
$$l_2 = 7^{10} \bmod \left(2^{31}-1\right) = 7^{10} \quad \Rightarrow x_2 = 0.13153778814316624223402060672362$$
$$l_3 = 7^{15} \bmod \left(2^{31}-1\right) = 1622650073 \quad \Rightarrow x_3 = 0.7556053221950332271843372039424$$
$$l_4 = \left(7^5 * 1622650073\right) \bmod \left(2^{31}-1\right) = 984943658 \quad \Rightarrow x_4 = 0.45865013192344928715538665985474$$
$$x_5 = 0.53276723741216922058359217857178$$

Use of Prime Numbers

- The number $2^{31} - 1$ is a prime number, so the remainder when a number is divided by a prime is rather, well random.
- Notes on the previous algorithm:
  - The l's can reach a maximum value of the prime number.
  - Dividing by this number maps the integers into reals within the open interval (0,1.0).
    - Why open interval?
  - ℓ_0 is called the *seed* of the random process. We can use anything here.

Other Algorithms

- Multiply by a large prime and take the lower-order bits.
- Here, we use higher-bit integers to generate 48-bit random numbers.
- Drand48()

$$x_{n+1} = (2736731631558x_n + 138) \bmod 2^{48}$$

$$x_0 = 1$$
$$x_1 = 2736731631696$$
$$x_2 = 216915228954218$$
$$x_3 = 44664858844294$$
$$x_4 = 123276424030766$$
$$x_5 = 162415264731678$$

29961701459390

51892741493630

251715685692926

37108576904446

163500647628542

Other Algorithms

- Many more such algorithms.

$$u_{n+1} = (8t-3)u_n$$

$$x_n = \frac{u_n}{2^q}$$

t is any large number

What is this operation?

- Some do not use integers. Integers were just more efficient on old computers.

$$x_{n+1} = (\pi + x_n)^5 \bmod 1$$

Other Algorithms

- One way to improve the behavior of random number generator

$$I_n = (a \times I_{n-1} + b \times I_{n-2}) \bmod(m)$$

→ Has two initial seed and can have a period greater than m

The RANMAR generator

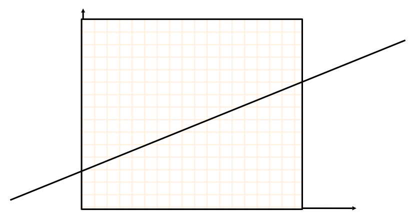
- Available in the CERN Library
  - Requires 103 initial seed
  - Period : about 10^{43}
  - This seems to be the ultimate random number generator

Properties of Pseudo-Random Numbers

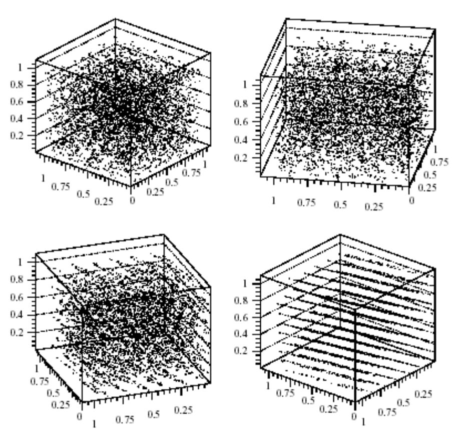
- Three key properties that you should remember:
  1. These algorithms generate periodic sequences (hence not random). To see this, consider what happens when a random number is generated that matches our initial seed.

Properties of Pseudo-Random Numbers

1. The restriction to quantized numbers (a finite-set), leads to problems in high-dimensional space. Many points end up to be co-planar. For ten-dimensions, and 32-bit random numbers, this leads to only 126 hyper-planes in 10-dimensional space.



3D Distribution from RANDU



Problems seen when observed at the right angle

The Marsaglia effect

- 1968, Marsaglia
- Randon numbers fall mainly in the planes

- The replacement of the multiplier from 65539 to 69069 improves performance significantly

Properties of Pseudo-Random Numbers

1. The individual digits in the random number may not be independent. There may be a higher probability that a 3 will follow a 5.

Available functions

- Standard C Library
  - Type in “man rand” on your CIS Unix environment.
    - Rather poor pseudo-random number generator.
    - Only results in 16-bit integers.
    - Has a periodicity of 2**31 though.
  - Type in “man random” on your CIS Unix environment.
    - Slightly better pseudo-random number generator.
    - Results in 32-bit integers.
    - Used rand() to build an initial table.
    - Has a periodicity of around 2**69.
  - #include <stdlib.h>

Available functions

- Drand48() – returns a pseudo-random number in the range from zero to one, using double precision.
  - Pretty good routine.
  - May not be as portable.

Initializing with Seeds

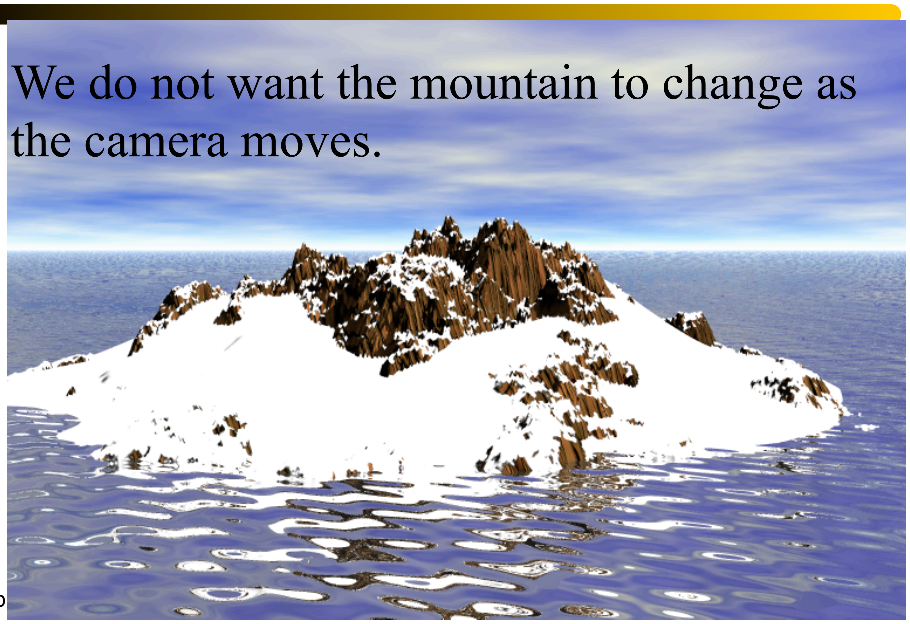
- Most of the algorithms have some state that can be initialized. Many times this is the last generated number (not thread safe).
- You can set this state using the routines initialization methods (srand, srandom or srand48).
  - Why would you want to do this?

Initializing with Seeds

- Two reasons to initialize the seed:
  1. The default state always generates the same sequence of random numbers. Not really random at all, particularly for a small set of calls. Solution: Call the seed method with the lower-order bits of the system clock.
  2. You need a deterministic process that is repeatable.

Initializing with Seeds

- We do not want the mountain to change as the camera moves.



Mapping random numbers

- Most computer library support for random numbers only provides random numbers over a fixed range.
- You need to map this to your desired range.
- Two common cases:
  - Random integers from zero to some maximum.
  - Random floating-point or double-precision numbers mapped to the range zero to one.

Non-rectangular Areas

- In 2D, we may want *points* randomly distributed over some region.
  - Square – independently determine x and y.
  - Rectangle - ???
  - Circle - ???
    - *Wrong way* – independently determine r and θ.

Monte-Carlo Techniques

- **Problem**: What is the probability that 10 dice throws add up exactly to 32?
- **Exact Way.** Calculate this exactly by counting all possible ways of making 32 from 10 dice.
- **Approximate (Lazy) Way.** Simulate throwing the dice (say 500 times), count the number of times the results add up to 32, and divide this by 500.
- **Lazy Way can get quite close to the correct answer quite quickly.**

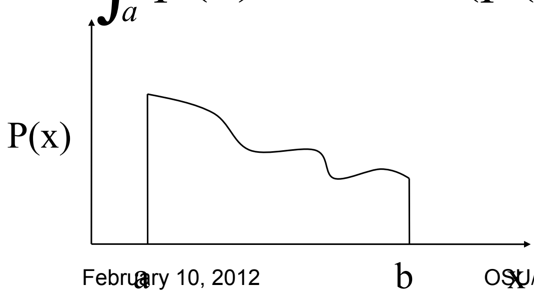
Monte-Carlo Techniques

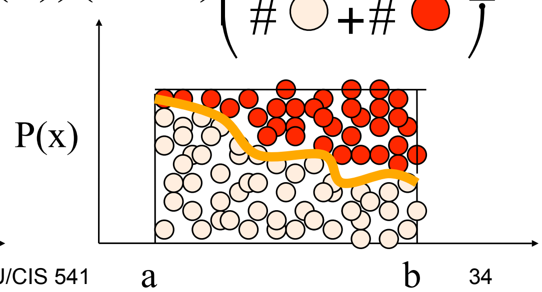
- Sample Applications
  - Integration
  - System simulation
  - Computer graphics - Rendering.
  - Physical phenomena - radiation transport
  - Simulation of Bingo game
  - Communications - bit error rates
  - VLSI designs - tolerance analysis

Simple Example: $\int_a^b p(x)dx$

- Method 1: Analytical Integration
- Method 2: Quadrature
- Method 3: MC -- random sampling the area enclosed by a<x<b and 0<y<max (p(x))

$$\int_a^b p(x)dx \approx \max(p(x))(b-a)\left(\frac{\#\bigcirc}{\#\bigcirc + \#\bullet}\right)$$





Simple Example: $\int_a^b p(x)dx$

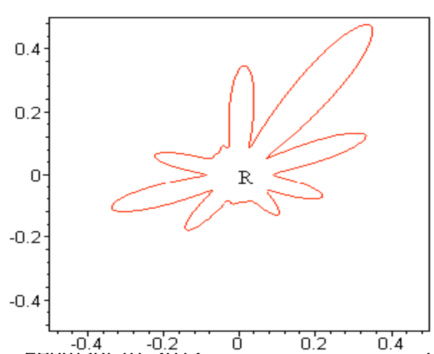
- Intuitively:

$$\int_a^b p(x)dx \approx \max(p(x))(b-a)\left(\frac{\#\bigcirc}{\#\bigcirc + \#\bullet}\right)$$

$$\Rightarrow Area_{Box} \bullet \text{Probability}\left\{\bar{y} \le f\left(\bar{x}\right)\right\}$$

Shape of High Dimensional Region

- Higher dimensional shapes can be complex.
- How to construct weighted points in a grid that covers the region R ?

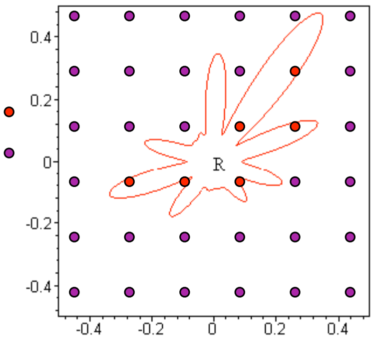


Problem :
mean-square distance from the origin

$$< r^2 > = \frac{\iint (x^2 + y^2)dxdy}{\iint dxdy}$$

Integration over simple shape ?

$$s = \begin{cases} 1 & \text{inside R} \\ 0 & \text{outside R} \end{cases}$$



$$\langle r^2 \rangle = \frac{\int_{-0.5}^{+0.5} dx \int_{-0.5}^{+0.5} dy (x^2 + y^2) s(x,y)}{\int_{-0.5}^{+0.5} dx \int_{-0.5}^{+0.5} dy s(x,y)}$$

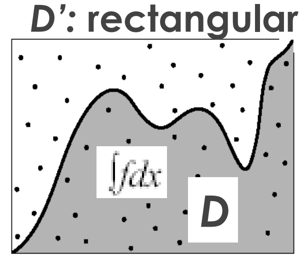
→ Grid must be fine enough !

Monte-Carlo Integration

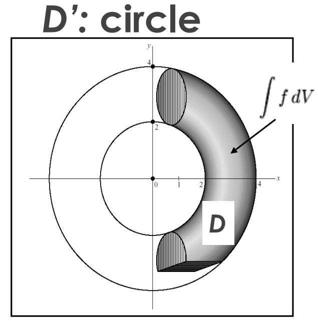
- **Integrate a function over a complicated domain**
  - **D: complicated domain.**
  - **D': Simple domain, superset of D.**
- **Pick random points over D':**
- **Counting: N: points over D**
- **N': points over D'**

$$\frac{Volume_D}{Volume_{D'}} \approx \frac{N}{N'}$$

D': rectangular



D': circle



Estimating π using Monte Carlo

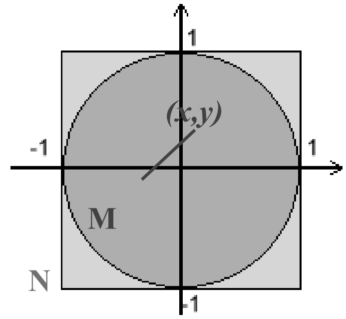
- The probability of a random point lying inside the unit circle:

$$\mathbf{P}\left(x^2+y^2<1\right)=\frac{A_{circle}}{A_{square}}=\frac{\pi}{4}$$

- If pick a random point *N* times and *M* of those times the point lies inside the unit circle:

$$\mathbf{P}^{\circ}\left(x^2+y^2<1\right)\underset{\longrightarrow}{=}\frac{M}{N}$$

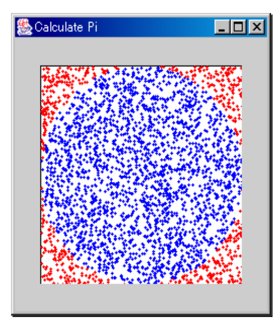
- If *N* becomes very large, P=P⁰



$$\pi=\frac{4\cdot M}{N}$$

Estimating π using Monte Carlo

- Results:
  - N = 10,000 Pi= 3.104385
  - N = 100,000 Pi= 3.139545
  - N = 1,000,000 Pi= 3.139668
  - N = 10,000,000 Pi= 3.141774
  - …



Estimating π using Monte Carlo

```
double x, y, pi;
const long m_nMaxSamples = 100000000;
long count=0;
for (long k=0; k<m_nMaxSamples; k++) {
    x=2.0*drand48() – 1.0;  // Map to the range [-1,1]
    y=2.0*drand48() – 1.0;
    if (x*x+y*y<=1.0) count++;
 }
 pi=4.0 * (double)count / (double)m_nMaxSamples;
```


Standard Quadrature

- We can find numerical value of a definite integral by the definition:

$$\int_a^b f(x)dx = \lim_{\Delta x \to \infty} \sum_{i=1}^{N} f(x_i)\Delta x$$

where points x_i are uniformly spaced.

Error in Quadrature

- Consider integral in *d* dimensions:

$$\int_{\Omega} f(X)\,dx_1 dx_2 \cdots dx_d \approx \sum f(X_i)\Delta x^d$$

- The error with *N* sampling points is

$$\left| \int f(X)\,dX - \sum f(X)\Delta x^d \right| \propto N^{-1/d}$$

Monte Carlo Error

- From probability theory one can show that the Monte Carlo error decreases with sample size *N* as

$$\varepsilon \propto \frac{1}{\sqrt{N}}$$

independent of dimension *d*.

General Monte Carlo

- If the samples are not drawn uniformly but with some probability distribution P(*X*), we can compute by Monte Carlo:

$$\int f(X)P(X)dX = \frac{1}{N}\sum_{i=1}^{N} f(X_i)$$

Where P(X) is normalized, $\int P(X)dX = 1$