Introduction to Monte Carlo methods

Matteo Negrini
INFN-Bologna

INFN

Outline

- Monte Carlo methods
- Random and pseudo-random number generators
- Sampling techniques
- Examples

The aim is to introduce the main concepts in an intuitive and user-oriented way.

The rigorous formalism is sometimes avoided.

The idea in short

“The spirit of Monte Carlo is best conveyed by the example discussed in von Neumann's letter to Richtmyer.

Consider a spherical core of fissionable material surrounded by a shell of tamper material.

Assume some initial distribution of neutrons in space and in velocity but ignore radiative and hydrodynamic effects. The idea is to now follow the development of a large number of individual neutron chains as a consequence of scattering, absorption, fission, and escape.

At each stage a sequence of decisions has to be made based on statistical probabilities appropriate to the physical and geometric factors. The first two decisions occur at time t = 0, when a neutron is selected to have a certain velocity and a certain spatial position. The next decisions are the position of the first collision and the nature of that collision. If it is determined that a fission occurs, the number of emerging neutrons must be decided upon, and each of these neutrons is eventually followed in the same fashion as the first.

If the collision is decreed to be a scattering, appropriate statistics are invoked to determine the new momentum of the neutron. When the neutron crosses a material boundary, the parameters and characteristics of the new medium are taken into account. Thus, a genealogical history of an individual neutron is developed.

The process is repeated for other neutrons until a statistically valid picture is generated.”

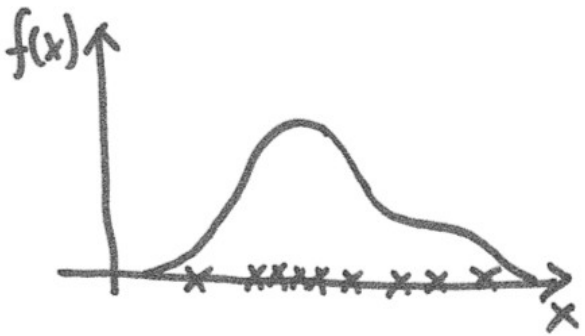
N. Metropolis, The Beginning of the Monte Carlo Method, Los Alamos Science 15, Special Issue 1987

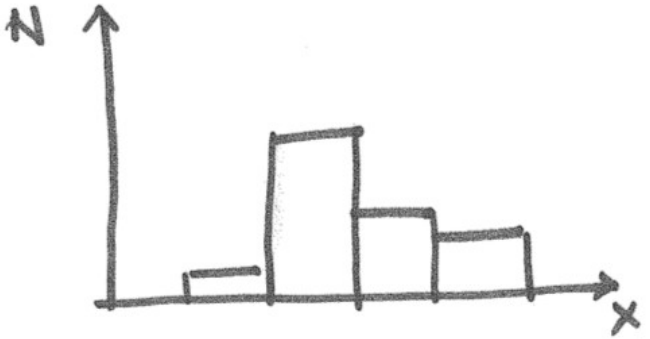
Monte Carlo methods

- Monte Carlo methods use random number generators in scientific computing
- Can be used to compute physics quantities when the analytical solution is impossible or impractical to obtain
- Typical procedure:
  - choose one initial configuration within the allowed ranges for the input variables
  - compute the output for this configuration
  - repeat the previous steps N times, changing the initial configuration
  - statistically analyze the results (uncertainty scales as $1/\sqrt{N}$)
- Possible applications:
  - Compute theoretical quantities
  - Design of an experiment

Random number

- A random number is:
  - a number chosen by chance from some specified distribution
  - a large set of these numbers should reproduce the underlying distribution
  - in a sequence of random numbers there is no correlation between two successive elements





Pseudo-random number generators

- Pseudo-random number generators are computer algorithms that:
  - can produce long sequences of numbers determined by the initial value (seed)
  - these sequences are indistinguishable from a “real” sequence of random numbers
- By running several times a generator using the same seed will produce exactly the same sequence
- In order to produce independent samples the seed should be changed
- The sequence will repeat after some point (period, bound by the total number of possible outcomes but can be shorter)

Pseudo-random generator algorithms

Possibility to choose among different algorithms:

- **C++:** use the <random> header to access several generators
- **ROOT:** generators accessed through TRandom class, see list here: https://root.cern.ch/doc/master/classTRandom.html
- **Geant4:** generators accessed through the choice of the engine in CLHEP::HepRandom class, see list here: http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/Fundamentals/global.html#the-heprandom-module-in-clhep

- Different performance in terms of speed, period, pass randomness tests

Example: rand()

- The rand() function of stdlib can be used to extract a sequence of pseudo-random numbers
- Each time it is called it returns an integer in the range [0,RAND_MAX] (defined in stdlib depending on the implementation of the algorithm)
- srand(seed) can be used to set the random seed

Find the correct way to extract a series of 100 numbers uniformly distributed in [0,1] using rand()

```
srand(12345);
for (int i=0; i<100; i++) {
  float val=rand()/(float)RAND_MAX;
}
```

Correct

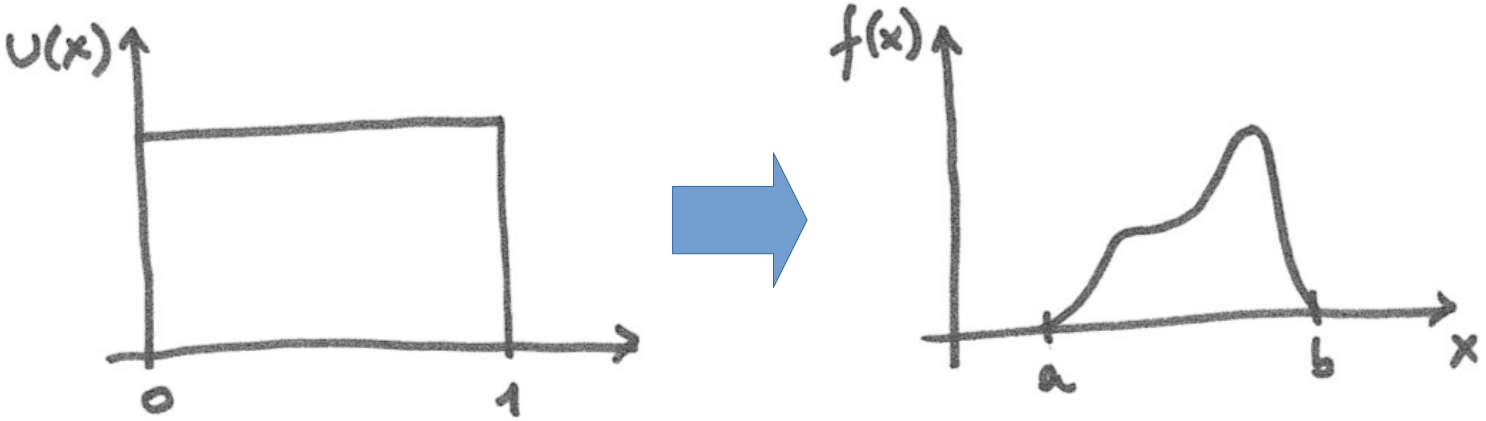
I set the seed only once before the loop

```
for (int i=0; i<100; i++) {
  srand(12345);
  float val=rand()/(float)RAND_MAX;
}
```

rand() extracts an int but I need a float so I need the (float), otherwise I'd always get 0 because rand()/RAND_MAX goes from 0 to 1. At least one of the two numbers must be float.

Sampling techniques

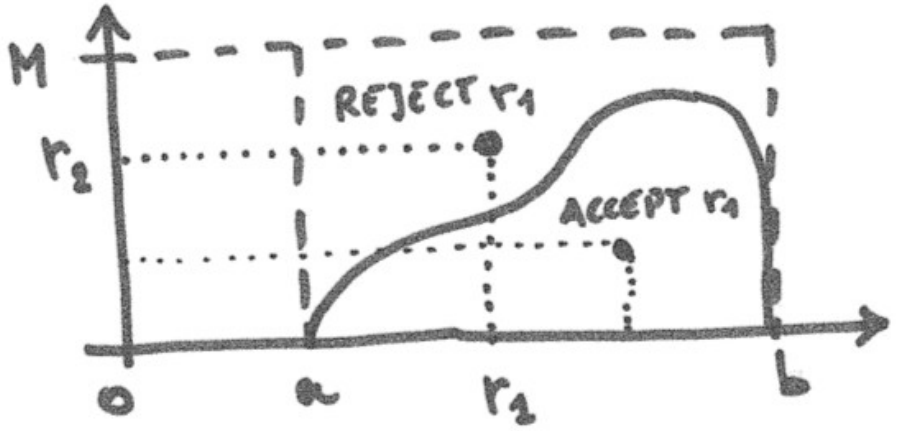
- Produce a sample of a random variable X following the desired probability distribution function f(x) from Uniform from 0 to 1
- Several “common” distributions are probably already available for your favorite code (root, matlab, R, etc.)
  - gaussian, poisson, exponential, binomial, ...
- Otherwise a technique should be applied to generate X starting from a uniform distribution



1 Acceptance - rejection method

- Produce a sequence of random numbers in an interval [a,b], following the desired probability density function f(x), with M≥max(f(x))
- Start from uniform generator:
  - extract one random number r_1 uniform in [a,b]
  - extract a random number r_2 uniform in [0,M]
  - if $r_2<f(r_1)$ accept r_1, otherwise repeat the procedure
- Disadvantages:
  - needs at least 2 extraction for each number to be used
  - may need to loop several times if f(x)<<M over wide regions in [a,b] (so smarter solutions should be implemented)

Points in the tail will be rejected more times the more the tail is long. Due to a high rejection rate increases also the computation time



2 Inverse of the CDF

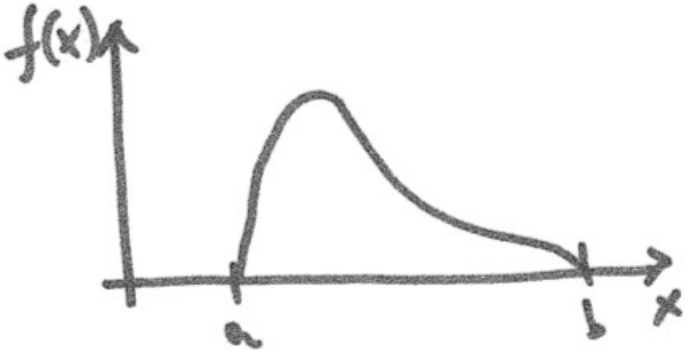
CDF = Cumulative Density Function

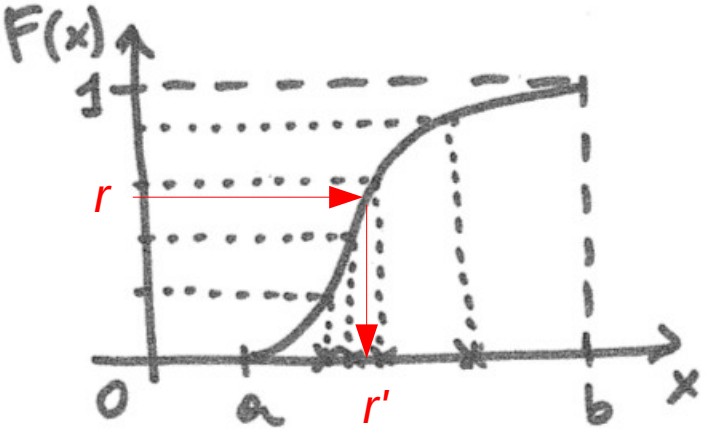
- Given a PDF f(x) defined in [a,b], the cumulative distribution function (CDF) is defined as

$$F(x) = \int_a^x f(x')\,dx'$$

- By construction $0 \le F(x) \le 1$
- The procedure is to extract a sequence of numbers r uniformly distributed in [0,1] and to compute $r' = F^{-1}(r)$
  - assuming F(x) can be computed and inverted
- r' are distributed as f(x)

More efficient than hit or miss but I must know the CDF and be able to invert it





Markov Chain Monte Carlo

- Algorithm proposed by Metropolis in 1953
- A set of random numbers is a Markov Chain if: $P(x_{i+1}=x \mid x_0, x_1, \ldots, x_i) = P(x_{i+1}=x \mid x_i)$ (ex. Random walk)
- Build a Markov Chain whose invariant distribution is f(x), defined in the domain Ω. Samples from f(x) will be approximated by a Markov Chain in Ω
- Allows to sample complex and multi-variate probability distributions

1. choose an appropriate PDF g(x) defined in Ω
2. choose an initial value x_0 in Ω
3. select a candidate step r_i from g(x) so that the candidate $\bar{x}= x_i+r_i$
4. if $f(\bar{x})>f(x_i)$ accept $x_{i+1}=\bar{x}$, otherwise accept it with probability $f(\bar{x})/f(x_i)$ or reject it
5. repeat from step 3
6. the initial values (burning phase), strongly influenced by the choice of x_0, should be rejected

Example: compute π

```
srand(123456);
double naccept=0;

for (int i=0; i<10000; i++) {
  double randX = rand() / (double)RAND_MAX;
  double randY = rand() / (double)RAND_MAX;
  if (randX*randX+randY*randY<1) naccept++; // sqrt(1)=1

  outfile << 4*naccept/(i+1) << std::endl;

}
```

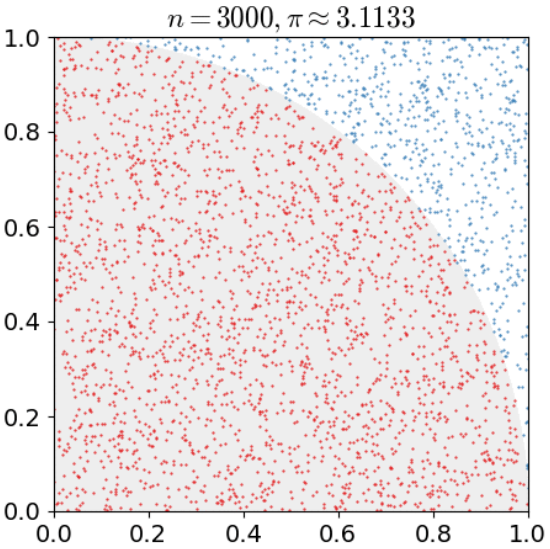
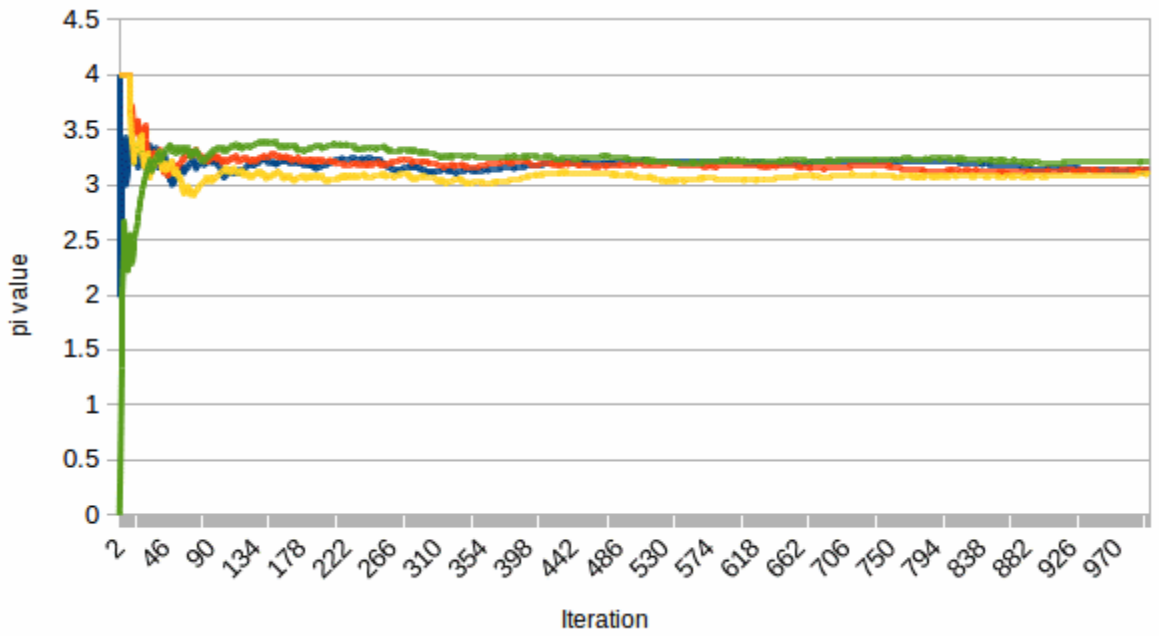


Figure from Wikipedia



The 4 colors refer to different random seeds used in the calculation

Example: energy resolution of a detector

- 511 keV photon in a scintillator
- Energy deposition following photoelectric or Compton interactions can be simulated using tools like Geant4
- Possible to add a simple energy resolution model to compute the detector response, for example:

$$E_i^{meas} = E_i^{dep} + \Delta E_i$$

$$\Delta E_i = Gaus() \times \sigma\left(E_i^{dep}\right)$$

where the Gaus() generator provides normally distributed pseudo-random samples