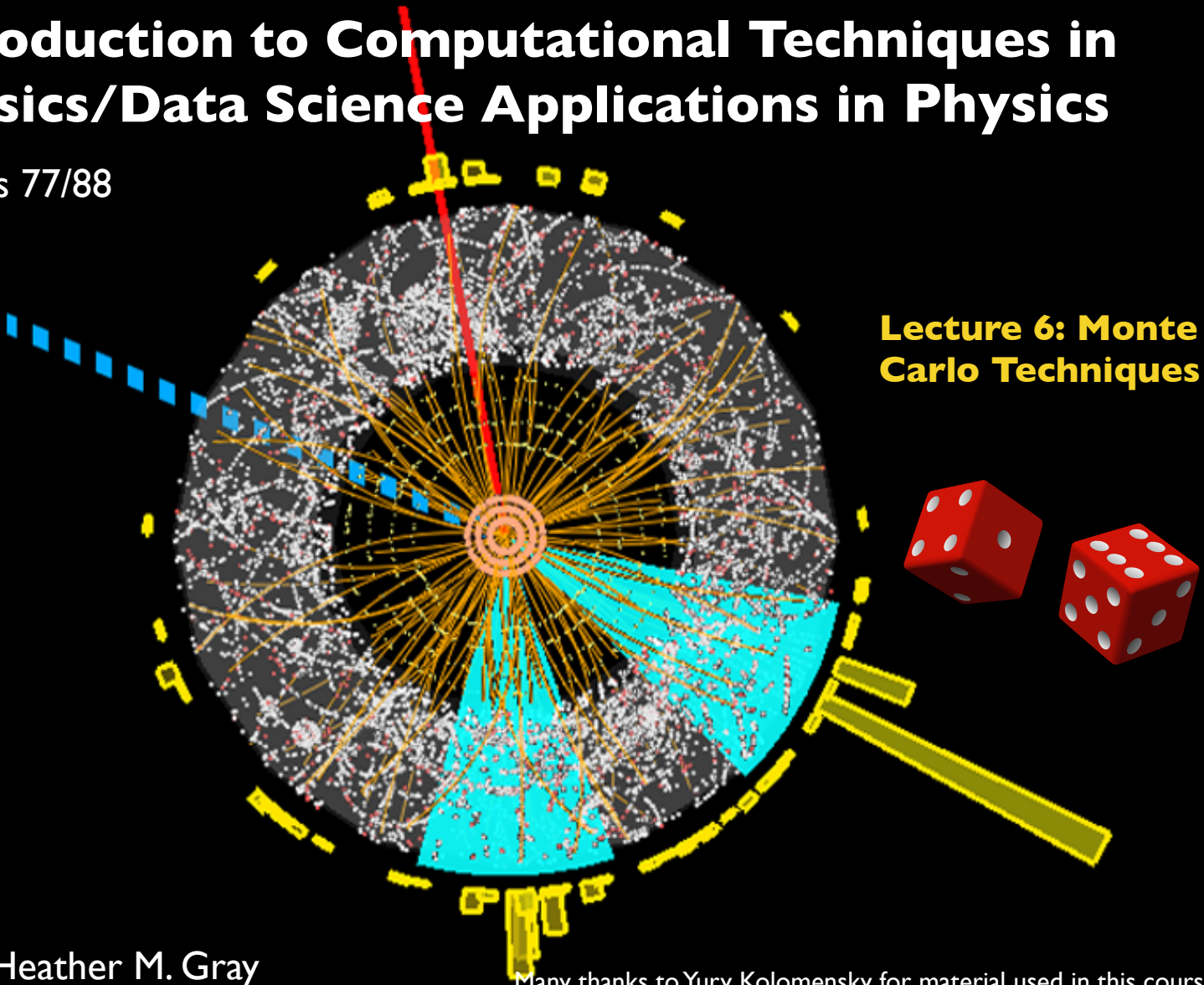
Introduction to Computational Techniques in Physics/Data Science Applications in Physics

Physics 77/88



Lecture 6: Monte Carlo Techniques

Prof. Heather M. Gray

Many thanks to Yury Kolomensky for material used in this course

Monte Carlo Techniques: Outline

- Introduction
- Random number generators
- Random number distributions
- Applications: integration

Monte Carlo Simulation

- Numerical analysis technique
  - Applicable when an analytical solution to a problem is too difficult or impossible
  - Applicable when the problem has probabilistic nature
    - e.g. an outcome of a measurement
  - Involves repeated random sampling of the parameter space to obtain approximate (usually statistically-limited) result
- Applications
  - Most fields of physics
  - Engineering, finance: risk assessment and analysis
  - Computational biology
  - And of course, gaming

Examples

- Let's start with some examples in the jupyter notebook
  - flip a coin
  - roll a dice
  - pick a card from a deck

From: https://en.wikipedia.org/wiki/Coin_flipping#/media/File:Coin_Toss_(3635981474).jpg

From: https://en.wikipedia.org/wiki/Dice

From: https://en.wikipedia.org/wiki/File:AcetoFive.JPG

More examples

- Professional packages
  - Particle physics:
    - JETSET
    - Pythia
    - GEANT

  } particle production and decay

  detector simulation : interaction of particle of matter
  - Many ray-tracing/optics packages
  - First (documented) use: Manhattan project
- Possible Final Projects
  - Tournament simulator (e.g. March Madness, NHL playoffs, etc)
  - Program a game (craps, blackjack, poker)
    - Incorporate some statistical analysis
      - e.g. betting odds, strategy

The Monte Carlo Technique

- What Monte Carlo simulation can not do:
  - Predict outcome of a particular event (measurement, process, …) or sequence of events
- What Monte Carlo simulation can do:
  - Compute probability for a particular outcome
  - Determine probability distributions, correlations between observables, … that resemble those observed in the real process
- Precision limited by:
  - Random sampling (statistics, CPU time)
  - Modeling of the underlying process
    - e.g. approximations in describing probability distribution (systematics)

Overview of the technique

- Split the process into elementary steps, such that probability density functions (PDFs) for each ~~could be~~ step could be computed
- For each step:
  - Identify random variables and their correlations
  - Generate random number
  - Evaluate (conditional) outcome of the step
- Repeat for the next step
- At the end, evaluate result
- Repeat many times, build up a PDF of the outcomes

Example

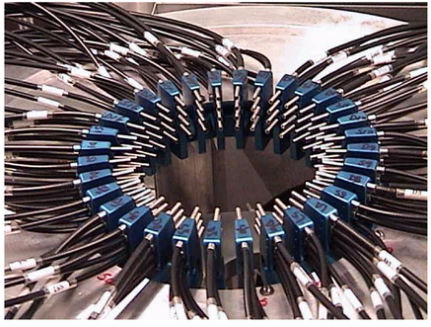
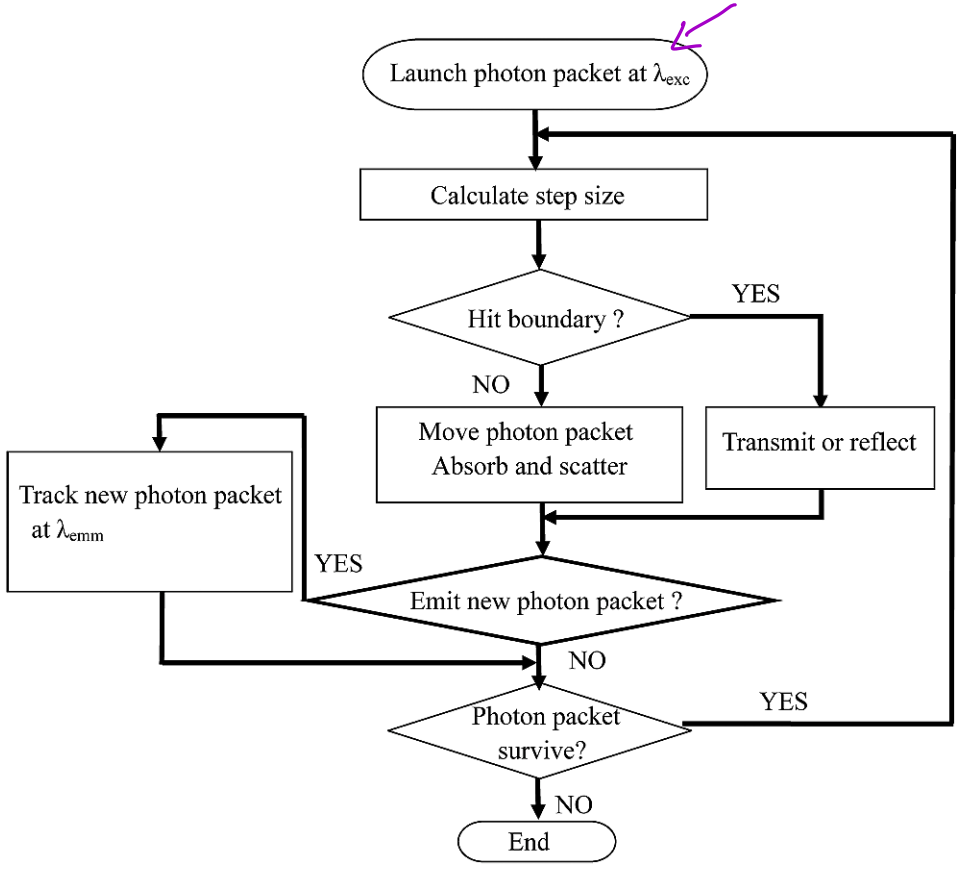


Image Credit

fiber-optic array for diffuse optical tomography for breast cancer detection



C. Zhu and Q. Liu, "Review of Monte Carlo modeling of light transport in tissues" J. Biomed. Opt. 18(5), 050902 (May 10, 2013). doi:10.1117/1.JBO.18.5.050902

Random Number Generators

- All MC simulations rely on a pseudo random number generator
  - Computer algorithm, so not truly random
    - Although "quantum" generators are possible (e.g. Johnson's noise generator — a cute 111A project, random.org)
- The most basic distribution is uniform
  - All distributions can be generated by a transform of a distribution (see below)
- Requirements:
  - Generate sequences with a very long period
  - Small correlation within the sequence
  - "Seeded" to produce repeatable sequences
  - speed

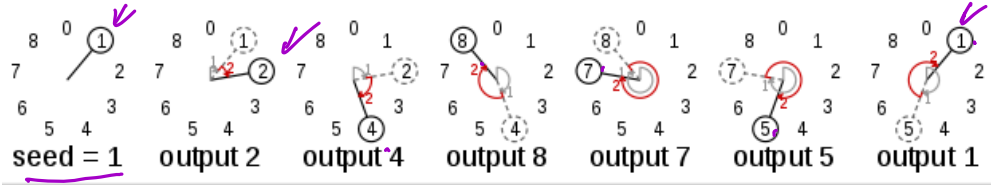
Simplest: Linear Congruent Generator

- $x_{n+1} = (a x_n + b) \bmod c$
  - Algorithm fixed by parameters: a, b and c
  - Sequence depends on initial seed: x_s
  - Period at most c
- So make c big, a & b prime
- E.g. RANDU: a= 65539 b= 0, c= 2^{31}
  - Period of 2^{31} numbers
  - Produces random numbers which are correlated within the sequence
- Modern generator: Mersenne Twister
  - Implemented in Python, ROOT (TRandom3)
    - Period of $2^{19937}-1$

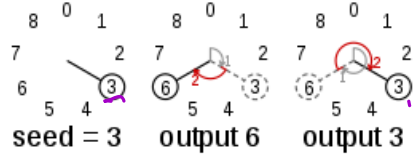
Parameters for LCG

- As we saw, certain sets of parameters can lead to periodic behavior
  - $x_{n+1} = (ax_n + b) \bmod c$

a=2
b=0
c=9



a=2
b=0
c=9



period 2

a=4
b=1
c=9

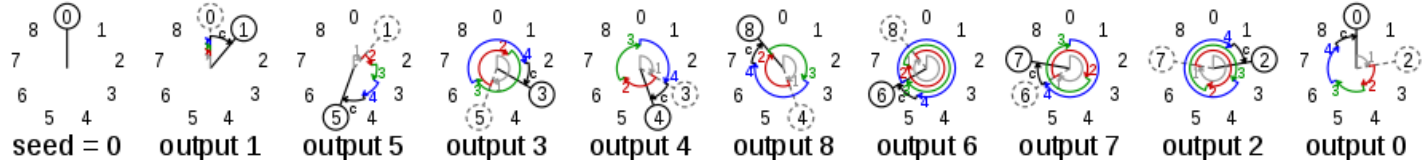


Image Credit

Generating Arbitrary Distributions

- Most frequently, you want a random variable x distributed according to some (predefined) distribution function $f(x)$
- Since most computers have a uniform random number generator, derive x from a uniformly distr. random number u:
  - $0 < u < 1$: $f(u) = x$
- Two basic techniques
  - Inverse transform method
  - von Neumann's accept-reject method

Inverse Transform Method

- Theorem:
  - If a is chosen with PDF $f(a)$, and F is a cumulative distribution function of f, then $F(a)$ is a random variable uniformly distributed between $[0, 1]$
  - $F(a) = \int_{-\infty}^{a} f(x)\,dx$
- Therefore can implement x with PDF $f(x)$:
  - Generate u:
    - $u \in [0, 1]$, $f(u) = 1$
  - Invert CDF:
    - $x = F^{-1}(u)$

Example

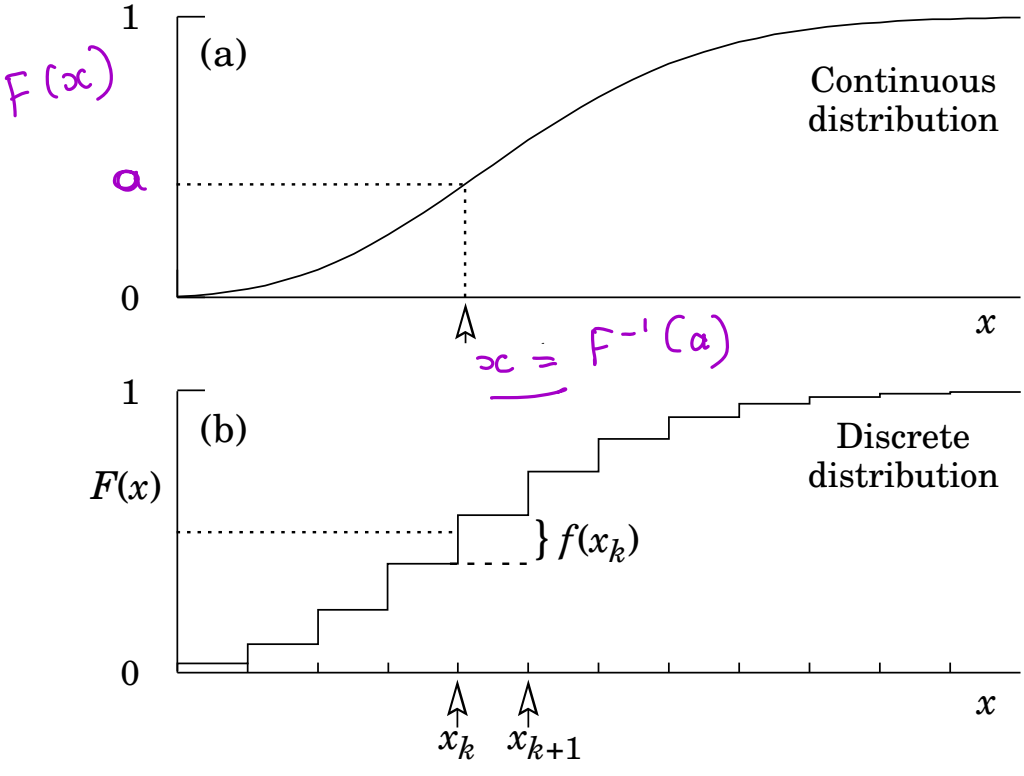
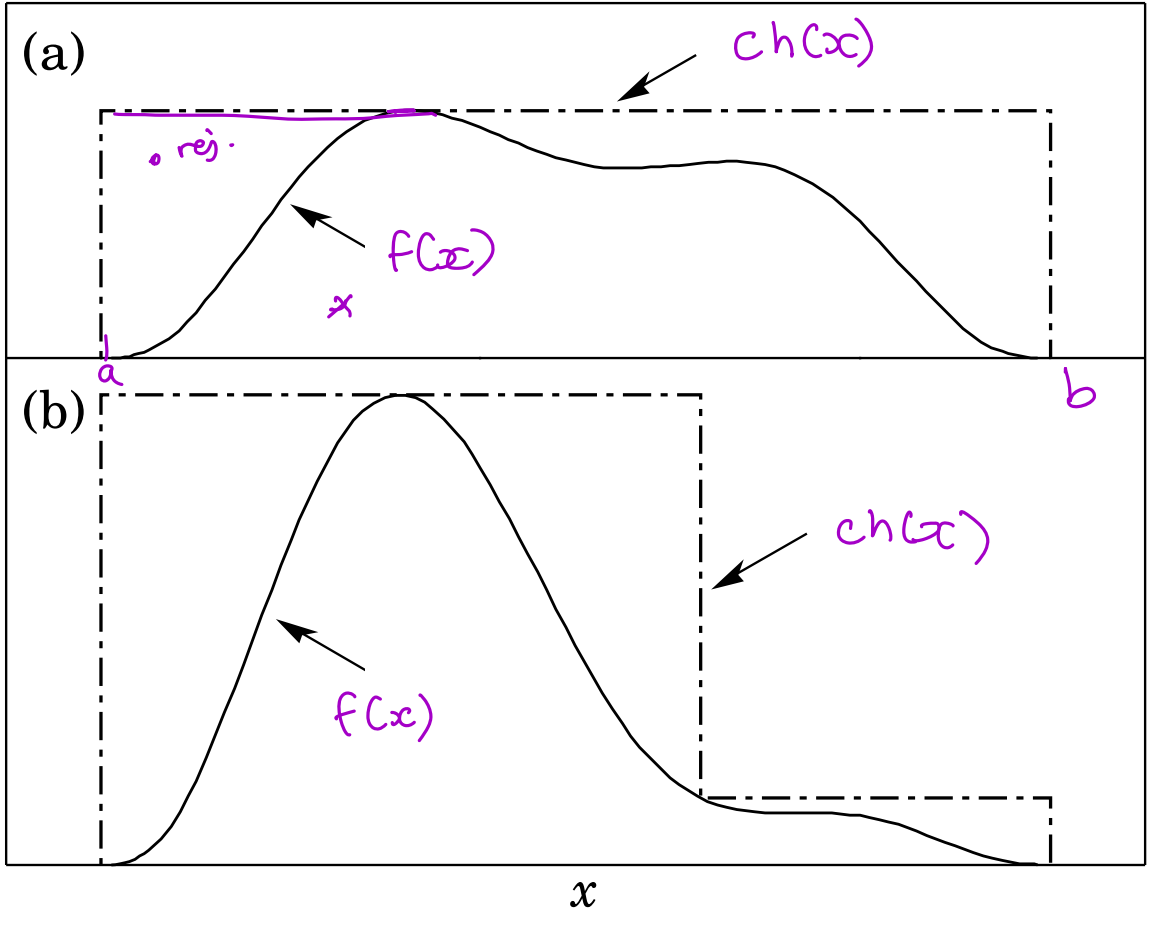


Figure 39.1: Use of a random number chosen from a uniform distribution (0,1) to find a random number x from a distribution with cumulative distribution function $F(x)$.

http://pdg.lbl.gov

Accept-Reject Method

- If $F(x)$ is an unknown, or difficult to compute analytical function, then finding F^{-1} may be impractical
- Let's suppose however that $f(x)$ is computable and finite, such that for any x we can find an easily generated PDF $h(x)$ and a constant c
  - $ch(x) > f(x)$ for all x
- Accept-reject method:
  - Generate x according to $h(x)$, and uniformly distributed u
  - If $u < f(x)/ch(x)$, accept x;
  - Otherwise regenerate x and u


(a)
ch(x)
rej.
f(x)
x
a
b
(b)
ch(x)
f(x)
x

Algorithms

- Exponential decay
  - $f(t) =$ $e^{-t/\tau}$ $\quad a < t < b$
  - Generate: $u \in$ $[0,1]$, $f(u) =$ 1
  - Define
    - $\alpha =$ $e^{-a/\tau}$
    - $\beta =$ $e^{-b/\tau}$
  - Then $t =$ $-\tau \ln(\beta + u(\alpha - \beta))$
- Gaussian distribution
  - If u_1 and u_2 are uniform on $[0,1]$, then
    - $z_1 =$ $\sin(2\pi u_1)\sqrt{-2\ln u_1}$
    - $z_2 =$ $\cos(2\pi u_2)\sqrt{-2\ln u_2}$
- are independent and Gaussian dist. with mean 0 and $\sigma =$ 1

Algorithms

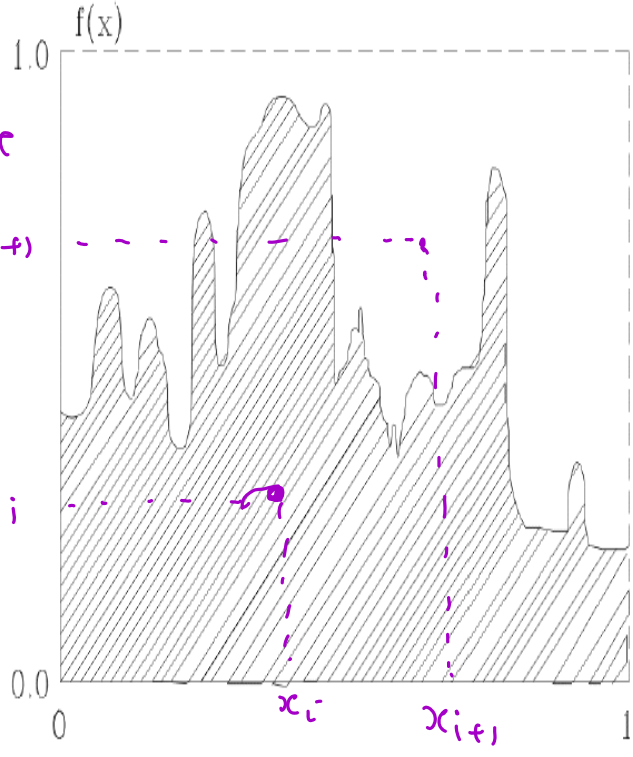
- Poisson Distribution
  - Iterate until a successful choice
    - Begin with $k =$ 1 and set $A =$ 1
    - Generate u
    - Replace A with uA
    - If A < e^{-u} (where u is the Poisson parameter)
      - accept $n_k =$ k-1 and stop
    - Else increment k by 1 and generate a new u and repeat
      - Always start with the value of A left from the previous try

Applications: MC Integration

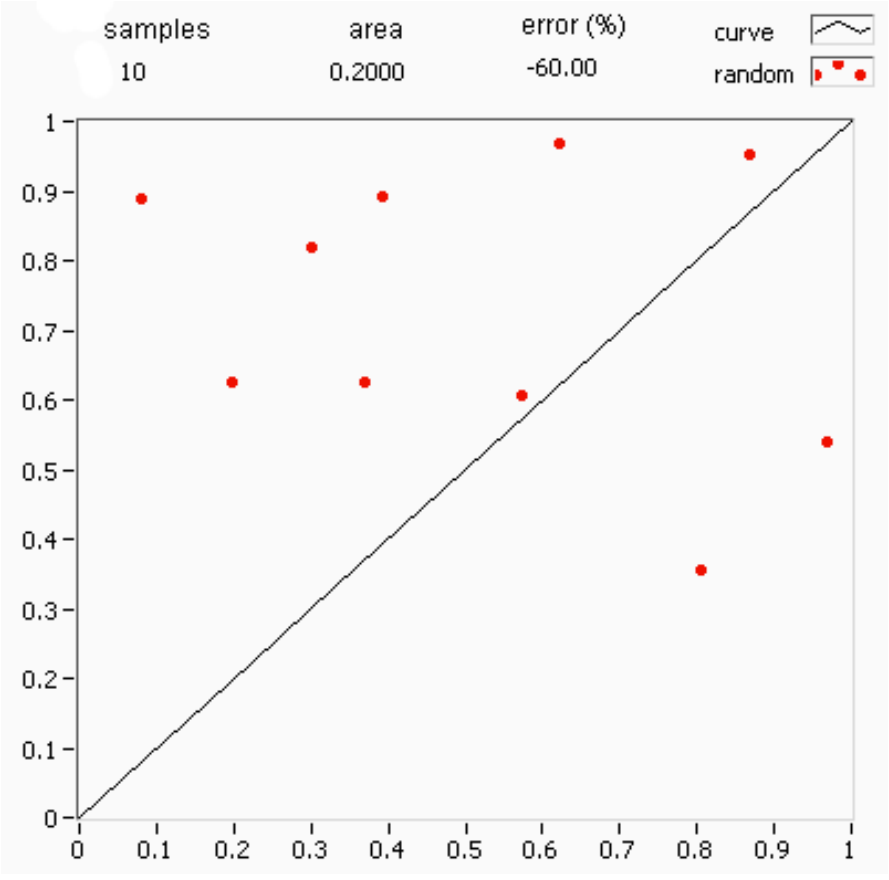
- Suppose we want to compute a multi-dimensional hypervolume
  - $Z = \int_{\Omega} f(x)\, dx$
- Generate a random space point x (e.g. uniformly distributed in space)
- If x is within Ω, increment sum $\sum_P = f(x)$
- Repeat N times
- Uncertainty on Z typically scales as $\frac{1}{\sqrt{N}}$

1-dimensional MC

- Suppose we want to find the area under the curve for $y = f(x)$ for a given interval
- Now let's assume we are throwing darts at the picture of $f(x)$
- Each dart will land on a point (x_i, y_i), where $0 < x_i < 1$ and $0 < y_i < 1$
- If the value of $y_i < f(x_i)$, then we count it as a hit
- Repeat the process for a large number of i
- The area under the curve is the number of hits / total number of throws



Example



https://en.wikipedia.org/wiki/Monte_Carlo_method

Link to gif

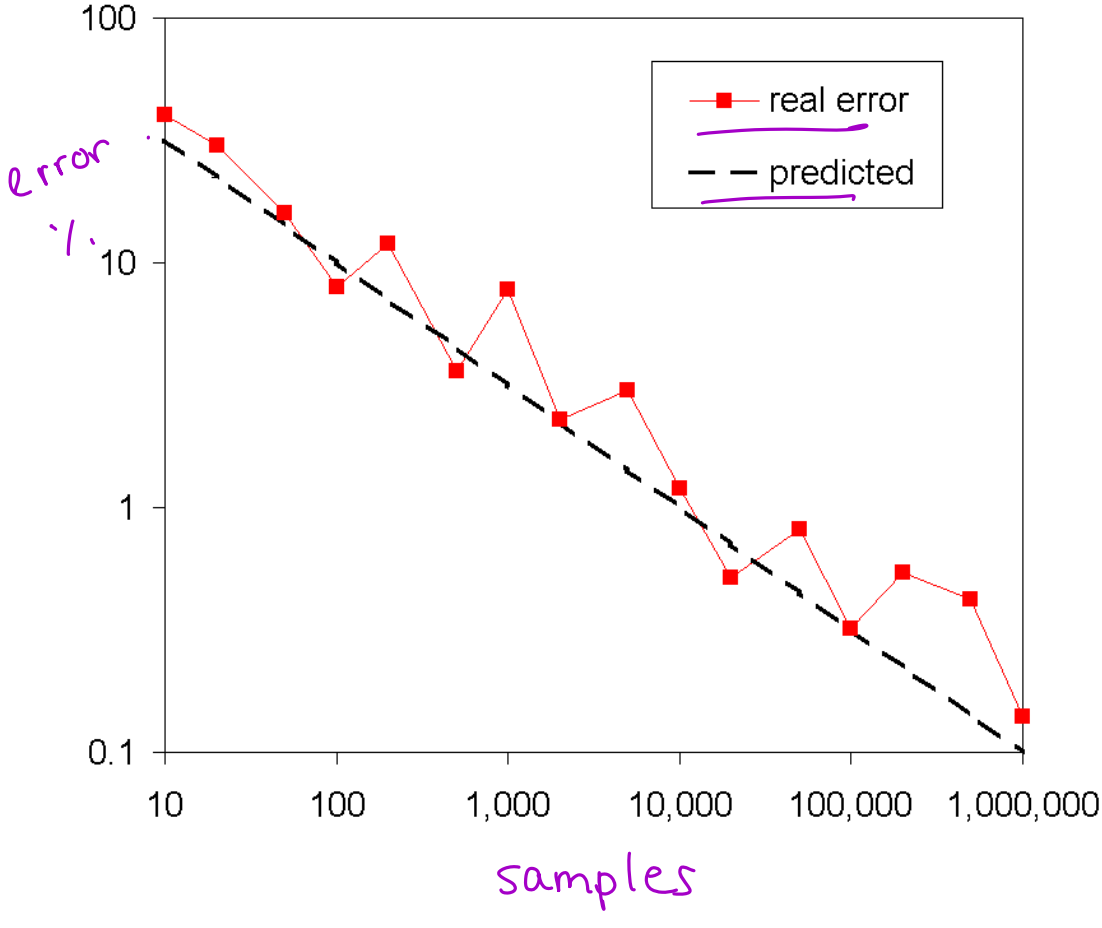
Error Estimate for MC Integrals

- Uncertainties for MC integrals follow the binomial distribution
  - Let the area under the curve be S, and the area populated by random points (darts) is S_0
  - Let N_0 be the number of points thrown, and N the number of points accepted
  - Then the estimator of S is:
    - $\hat{S} = S_0 \frac{N}{N_0}$
  - Define the efficiency of accepting points as
    - $\epsilon = \frac{S}{S_0}$
    - $\Rightarrow \hat{\epsilon} \approx \frac{N}{N_0}$

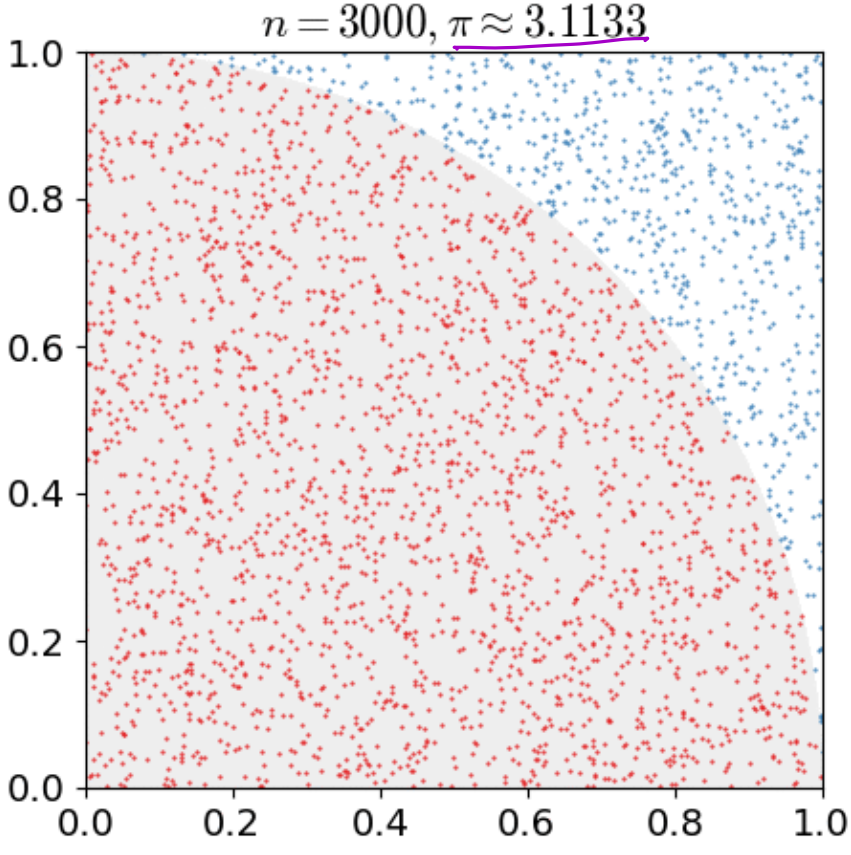
Error Estimate for MC Integrals

- N is a random variable, which follows a binomial distribution ("choose N out of N_0").
- Estimator of $\varepsilon = \frac{N}{N_0}$ is therefore also a random variable
- Its uncertainty is given by
  - $\sigma(\hat{\epsilon}) = \sqrt{\frac{\varepsilon(1-\varepsilon)}{N_0}}$
- You can show that in the limit $N \ll N_0$ $(\varepsilon \to 0)$
  - $\sigma(\hat{\epsilon}) \approx \sqrt{\frac{\varepsilon}{N_0}} \approx \frac{\varepsilon}{\sqrt{N}}$
  - $\sigma(\hat{S}) = S_0\,\sigma(\hat{\varepsilon}) = \frac{S}{\sqrt{N}}$

Error Estimate



Example: Compute π by MC



Link to gif

https://en.wikipedia.org/wiki/Monte_Carlo_method