Computational methods for Medical Physics

Monte Carlo (a bit more than) Basics

Dr. George DEDES

WS 2016-2017

Past and next lectures' content

- Previous lecture:
  - Started discussing the notion of the Monte Carlo technique
  - Presented MC basic idea, which is a statistical/stochastic method to solve integrals (Sampling and Hit-or-miss methods)
  - Briefly mentioned a few simplistic methods for numerical calculation of integrals
  - Studied one of the slightly more advanced MC methods for integral estimation (Importance Sampling)
- This lecture:
  - Continue on advanced MC methods for integral estimation (Control Variates)
  - Random numbers and generators
  - Sampling from distributions

External material

- Material form this lecture was taken from the following lectures:

  - Aalto University School of Science
    Department of Biomedical Engineering and Computational Science
    Lecture on Computational Science
    Dr. Ricu Linna
    Dr. Laura Juvonen

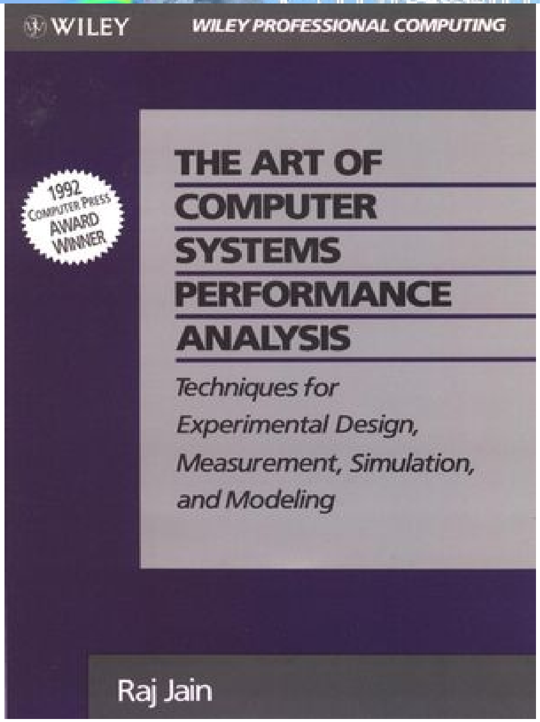
  - University of Helsinki
    Department of Physics
    Basics of Monte Carlo Simulations Lecture
    Priv. Doz. Dr. Flyura Djurabekova

  - University of Helsinki
    Department of Physics
    Scientific Computing III
    Dr. Antti Kuronen

- Also from:
  - A note on random number generators
    Christophe Dutang
    Diethelm Wuertz
  - Rice University
    Department of Computer Science
    Computer Systems Performance Analysis
    Dr. John Mellor-Crummey
  - Hochschule Ravensburg-Weingarten
    http://www.youtube.com/watch?v=aM_LXwQZy1E
    Advanced Mathematics for Engineers 2
    Dr. Wolfgang Ertel

External material

- Books:
  - The Art of Computer Systems Performance Analysis
    Raj Jain



- Assume again that we have a known function f(x), that varies very fast in the region [a,b] where we want to estimate its integral

- We can always use the simple MC Sampling or Hit-or-miss methods, but towards the end of the last lecture we started becoming a bit more skilled (and we want to keep this trend!)

- Back to probability theory:

$$Var(f-g) = Var(f) + Var(g) - 2Cov(f,g)$$

- This tells us that if we find such a known function g(x) for which:

$$Var(f-g) < Var(f)$$

- Then we could estimate our f integral more efficiently (accurately - fast) http://beam.acclab.helsinki.fi/~avchacho/mc/lec/mc_5-2x2.pdf

- Assume again that we have a known function f(x), that varies very fast in the region [a,b] where we want to estimate its integral

- We can always use the simple MC Sampling or Hit-or-miss methods, but towards the end of the last lecture we started becoming a bit more skilled (and we want to keep this trend!)

- Back to probability theory:

$$Var(f-g) = Var(f) + Var(g) - 2Cov(f,g)$$

- This tells us that if we find such a known function g(x) for which:

$$Var(f-g) < Var(f)$$

- For that we need $2Cov(f,g) > Var(g)$

http://beam.acclab.helsinki.fi/~avchacho/mc/lec/mc_5-2x2.pdf

- Assume again that we have a known function f(x), that varies very fast in the region [a,b] where we want to estimate its integral

- We can always use the simple MC Sampling or Hit-or-miss methods, but towards the end of the last lecture we started becoming a bit more skilled (and we want to keep this trend!)

- Back to probability theory:

$$Var(f-g) = Var(f) + Var(g) - 2Cov(f,g)$$

- This tells us that if we find such a known function g(x) for which:

$$Var(f-g) < Var(f)$$

- Which means that f and g are positively correlated => have similar shapes http://www.lce.hut.fi/teaching/S-114.1100/lect_9.pdf

- Stating already a drawback of the method: $I_g = \int_a^b g(x)\,dx$ has to be known

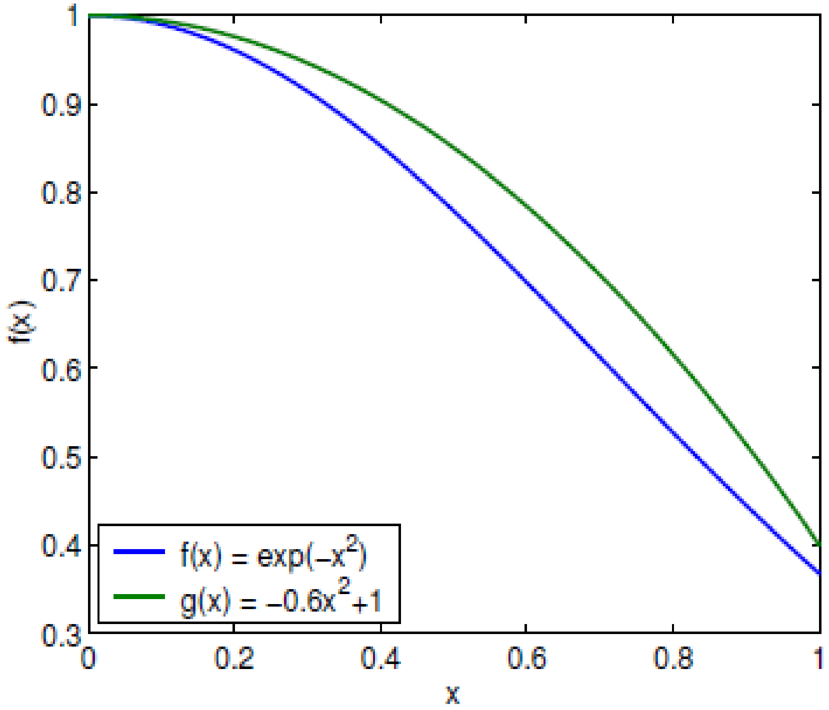
- When the requirements mentioned so far are fulfilled, we can write:

$$I = \int_a^b f(x)\,dx = \int_a^b [f(x) - g(x)]\,dx + \int_a^b g(x)\,dx = \int_a^b [f(x) - g(x)]\,dx + I_g$$

- Estimate I in the usual sampling way:

$$I = \frac{b-a}{N}\sum_{i=0}^{N}[f(x_i) - g(x_i)] + I_g$$

- Example: Estimation of the integral of $f(x)=e^{-x^2}$ in [0,1]



- We can get similar shape in [0,1] from $g(x)=-0.6x^2+1$

- g(x) is easy to integrate:

$$I_g = \int_0^1 g(x)\,dx = \int_0^1 (-0.6x^2+1)\,dx$$

$$I_g = -0.6\frac{x^3}{3} + x\Big|_0^1 = 0.8$$

- Finally we just have to estimate the integral of a slowly varying function

$$I = \frac{1}{N}\sum_{i=0}^{N}[f(x_i) - g(x_i)] + I_g$$

which is more efficient since all the random points will have a similar impact on the <f-g> estimation

- Last lecture - Programming exercise 2:
  - Write a program (in any programming language, preferable in C++, C, FORTRAN) that integrates a Gaussian function

    $$f(x) = e^{-x^2}$$

    in [0,1] using two MC integration techniques:
    - MC Sampling
    - Importance sampling (hint use e^{-x} as your g(x))
  - Compare the two methods

- This lecture - Programming exercise 3:
  - Write a program (in any programming language, preferable in C++, C, FORTRAN) that integrates a Gaussian function

$$f(x) = e^{-x^2}$$

    in [0,1] using control variates

    - Hint: use $g(x)=-0.6x^2+1$

  - Compare the results with that from exercise 2

- The simple and straightforward uniform random sampling technique is inefficient when used on fast varying functions
  - Not all subregions of [a,b] are of the same importance
  - The effective result is like "wasting" random numbers and the related calculations
- We need to "flatten" the function f so as to get rid of less contributing subregions
  - Either divide with a similarly behaving weighting function g (equivalent to non-uniform sampling)
  - Subtract g from f. When I_g known then I is dependent on a more stable function and uniform random sampling becomes efficient again

Summary of MC technique

- The most important message of this slide is that the Monte Carlo technique is NOT necessarily a simulation technique
- It is a method to solve multidimensional integrals by using a stochastic process
- The integrals might not even represent physical processes (they might of course do)
    - We have been simply estimating areas between a function and the x-axis
    - All those integrals had no randomness or stochastic nature
    - They had well defined value, we just used a stochastic method to estimate them
- Everything we did so far was no simulation of a stochastic/random process
- To reach our final goal then, MC simulations, let's discuss about randomness

- All this time we have been using random numbers like drawing floating point numbers from a “magic black box”

- Time to say a few things about how we get those numbers
  => Discuss about Random Number Generators (RNG)

- Disclaimer: Random number generation is a mathematics research field by itself
  => we are not going to get even close to the very fundamentals of it
  => we are going to barely scratch the surface of the topic

- Nevertheless, for the “hardcore” enthusiasts:

  - *The Art of Computer Programming*
    D. E. Knuth
  - *Random Number Generators: Good Ones are Hard to Find*
    S. Park, K. Miller
  - *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*
    M. Matsumoto, T. Nishimura

- Random numbers are used in a wide variety of applications, such as Natural Sciences, Economics, Insurance, Cryptography, Lotteries, Art …

- There are different kinds of random numbers

  - Real random numbers (RRN)
    Obtained by some physical processes, mechanical devices and also humans

  - Pseudo random numbers (PRN)
    Produced by computer deterministic algorithms

  - Quasi random numbers (QRN)
    Correlated, non-independent number sequences uniformly distributed

- We will focus on pseudo random numbers (PRN) and generators

Monte Carlo
Randomness by Dilbert

- A short educative story by DILBERT about randomness:

- We use a computer deterministic algorithm to create (almost) random numbers
- Deterministic means that for exactly the same input parameters/conditions, different invocations of the algorithm will yield the same result
- Some terminology:
  - Seed: Initial state(s) used by the algorithm in order to start generation
  - Pseudo-Random sequence: A set of PRNs initiated by a seed
  - Period: Repeated pattern in a PRN sequence
  - Cycle length, Tail: Structures related to the period

- Assume the following PRN sequence:

2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8, 2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8,

- Assume the following PRN sequence:

2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8, 2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8,

A repeated number doesn't necessarily signify a period

- Assume the following PRN sequence:

2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8, 2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8,

A number repeated twice also doesn't necessarily signify a period

- Assume the following PRN sequence:

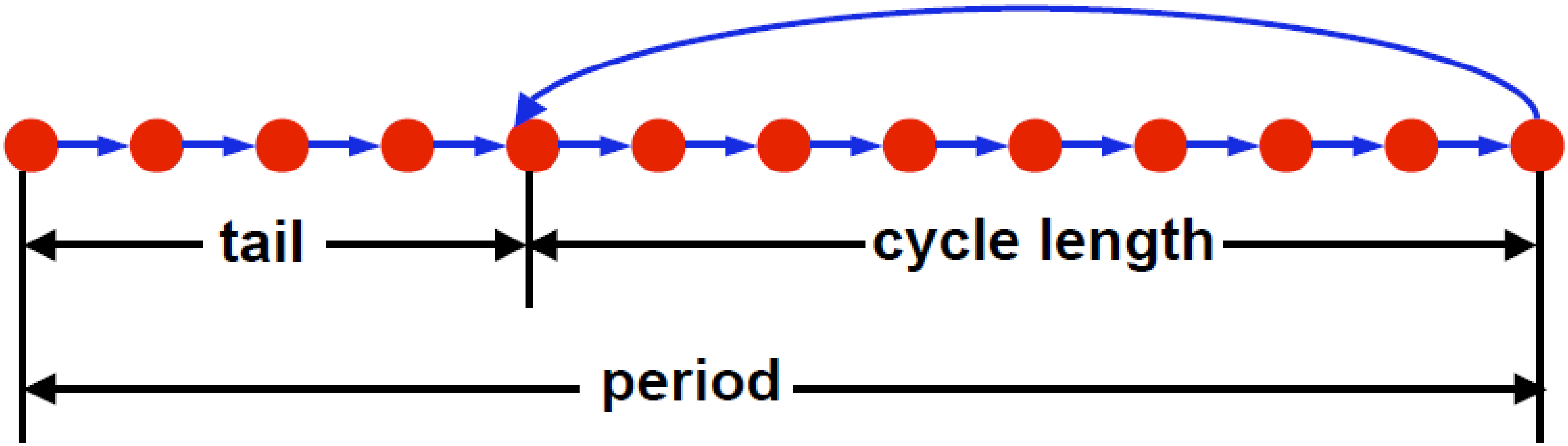
$$\underbrace{2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8,}\ \underbrace{2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8,}$$

An exactly repeated pattern
though signifies a period

- Assume the following PRN sequence:

$$\underbrace{2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8,}_{\text{Period = 12}}\ \underbrace{2, 5, 9, 1, 6, 3, 7, 9, 2, 2, 3, 8,}_{\text{Period = 12}}$$

- We use a computer deterministic algorithm to create (almost) random numbers
- Deterministic means that for exactly the same input parameters/conditions, different invocations of the algorithm will yield the same result
- Some terminology:
  - Seed: Initial state(s) used by the algorithm in order to start generation
  - Pseudo-Random sequence: A set of PRN initiated by a seed
  - Period: Repeated pattern in a PRN sequence
  - Cycle length, Tail: Structures related to the period


tail
cycle length
period

- Properties of a PRN generator
  - Statistical properties: Uniform distribution, no correlations
  - Long period: Finite, but the longer the better (low repetitiveness)
  - Reproducibility: For various (testing) reasons,
    results of applications based on RN should be reproducible
    => RN should be reproducible
  - Speed: We don’t want PRNG to be the main limiting factor in our application
  - Parallelizable: For cluster/parallel computing, need uncorrelated, non-overlapping sequences
- Usually PRN are integers that are converted via normalization to floating point numbers in the interval [0,1]

Monte Carlo
Middle Square method

- First described in the 11th century, but put into practice by von Neumann in 1949
  - The seed is a 4-digit number
  - Square the seed
  - Get the 4 middle numbers of the result as a PRN and also use it as input for the next iteration

Iteration	Input	Square	PRN
0	7662	58706244	0.7062
1	7062	49871844	0.8718
2	8718	76003524	0.0035
3	0035	00001225	0.0012
4	0012	00000144	0.0001
5	0001	00000001	0.0000

Monte Carlo
Middle Square method

- First described in the 11th century, but put into practice by von Neumann in 1949
  - The seed is a 4-digit number
  - Square the seed
  - Get the 4 middle numbers of the result as a PRN and also use it as input for the next iteration
  - Fast convergence to 0!
  - An example of a bad PRN generator

- A better algorithm is the Linear Congruential PRN generator
- Based on a very simple formula:

$$\boxed{x_n = (a x_{n-1} + \mathrm{b}) mod\ m}$$

- Starting from a seed integer number, we apply recursively the formula and get a sequence of PRN
- a, b, m parameters are fixed

- Let's do a simple example:
  - a=4, b=3, m=5

$$\boxed{x_n = (4x_{n-1} + 3) mod\ 5}$$

  - And let's take x_0=2 as seed

n	PRN
1	2
2	1
3	2

Period = 2

- Let's retry a simple example in order to get a longer period:
  - a=3, b=4, m=5

$$\boxed{x_n = (3x_{n-1} + 4) mod\ 5}$$

  - And let's take x_0=2 as seed

n	PRN
1	2
2	0
3	4
4	1
5	2

Period = 4

- Now let’s do a really nasty trick:
  - a=3, b=4, m=5

$$\boxed{x_n = (3x_{n-1} + 4) mod\ 5}$$

  - And let’s take x_0=3 as seed
  - Why does this happen?

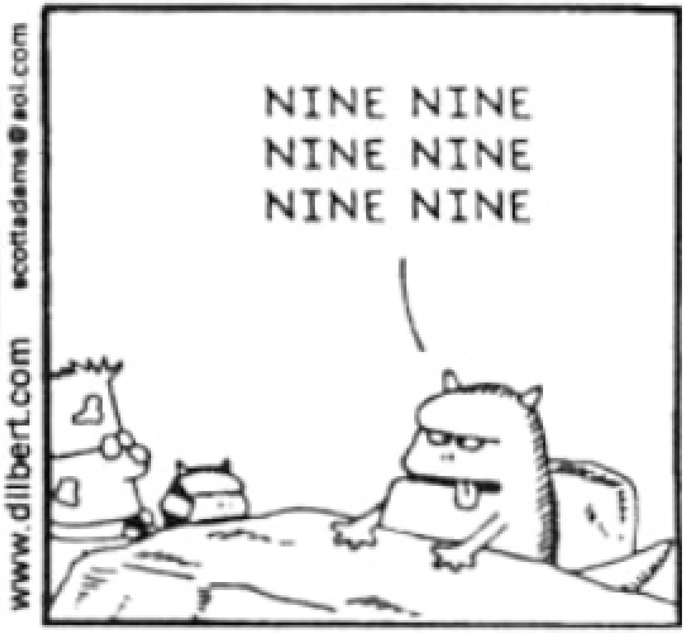
n	PRN
1	3
2	3
3	3
4	3
5	3

- Now let's do a really nasty trick:
  - a=3, b=4, m=5

$$\boxed{x_n = (3x_{n-1} + 4) mod\ 5}$$

  - And let's take x_0=3 as seed
  - Why does this happen?

n	PRN
1	3
2	3
3	3
4	3
5	3



- Yet another example:
  - a=1, b=4, m=5

$$\boxed{x_n = (x_{n-1} + 4) mod\ 5}$$

  - And let's take x_0=2 as seed

n	PRN
1	2
2	1
3	0
4	4
5	3
6	2

Period = 5

- Yet another example:
  - a=1, b=4, m=5

$$\boxed{x_n = (x_{n-1} + 4) mod\ 5}$$

  - How can we change a, b, x_0 to get a longer period for this specific generator?
  - We basically can't
  - The period is defined by the m parameter. Why?

- An easy trick to allow us for a maximum period longer than m is:

$$x_n = f(x_{n-1}) mod\ m \qquad period \leq m$$

$$x_n = f(x_{n-1}, x_{n-2}) mod\ m \qquad period \leq m^2$$

$$x_n = f(x_{n-1}, x_{n-2}, x_{n-3}) mod\ m \qquad period \leq m^3$$

$$\ldots\ldots\ldots \qquad \ldots\ldots\ldots$$

Monte Carlo
Linear Congruential method

- Another example:

$$x_n = (ax_{n-1} + bx_{n-2} + c) mod\ m$$

- With a=1, b=1, c=2, m=3
  and seeds x_{n-1}=1 , x_{n-2}=2

$$x_n = (x_{n-1} + x_{n-2} + 2) mod\ 3$$

n	PRN
1	1
2	2
3	2
4	0
5	1
6	0
7	0
8	2
9	1
10	2
11	2
12	0
13	1
14	0
15	0
16	2

- In general the PRN generators of the type:

$$\boxed{x_n = (a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_k x_{n-k} + c) mod\ m}$$

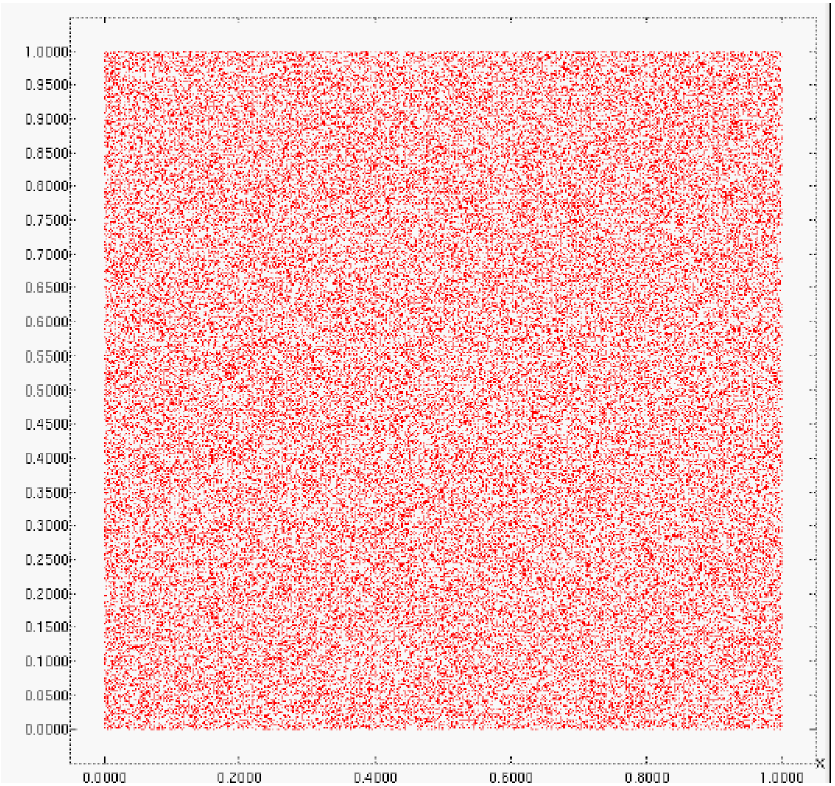
are called Multiple Recursive generators

- Periods vary from 10^9, which is considered rather short for nowadays MC applications, to -for example- 10^{6000}

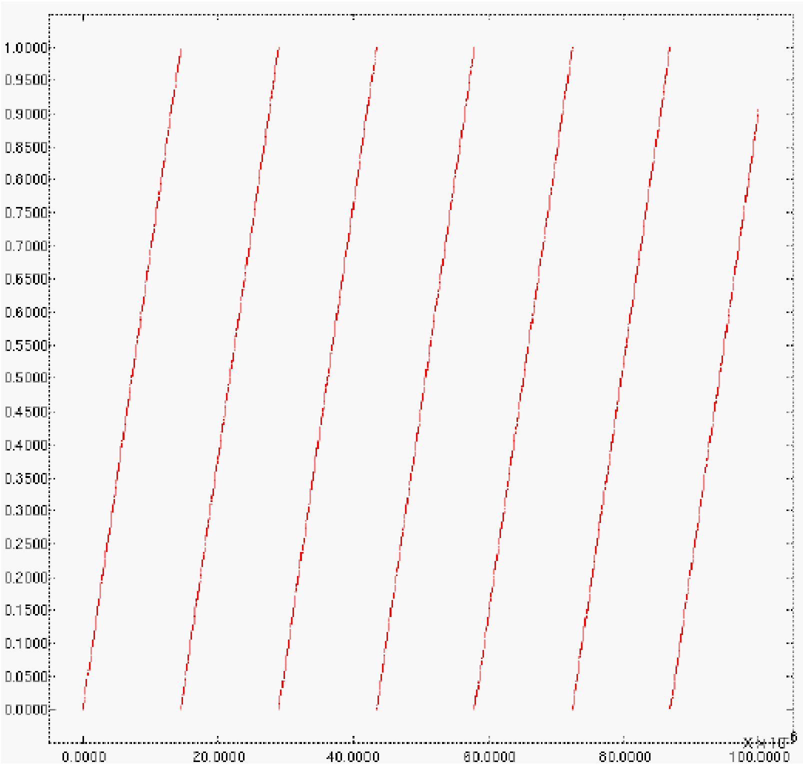
- A list of other than Linear Congruential type of PRN generators can be found in:

http://en.wikipedia.org/wiki/List_of_pseudorandom_number_generators

- A last comment following the discussion about long periods:
  - Recall that a few slides ago we saw that a long periodicity is not the only requirement from a good PRN generator
  - For example, uniformity is another one:



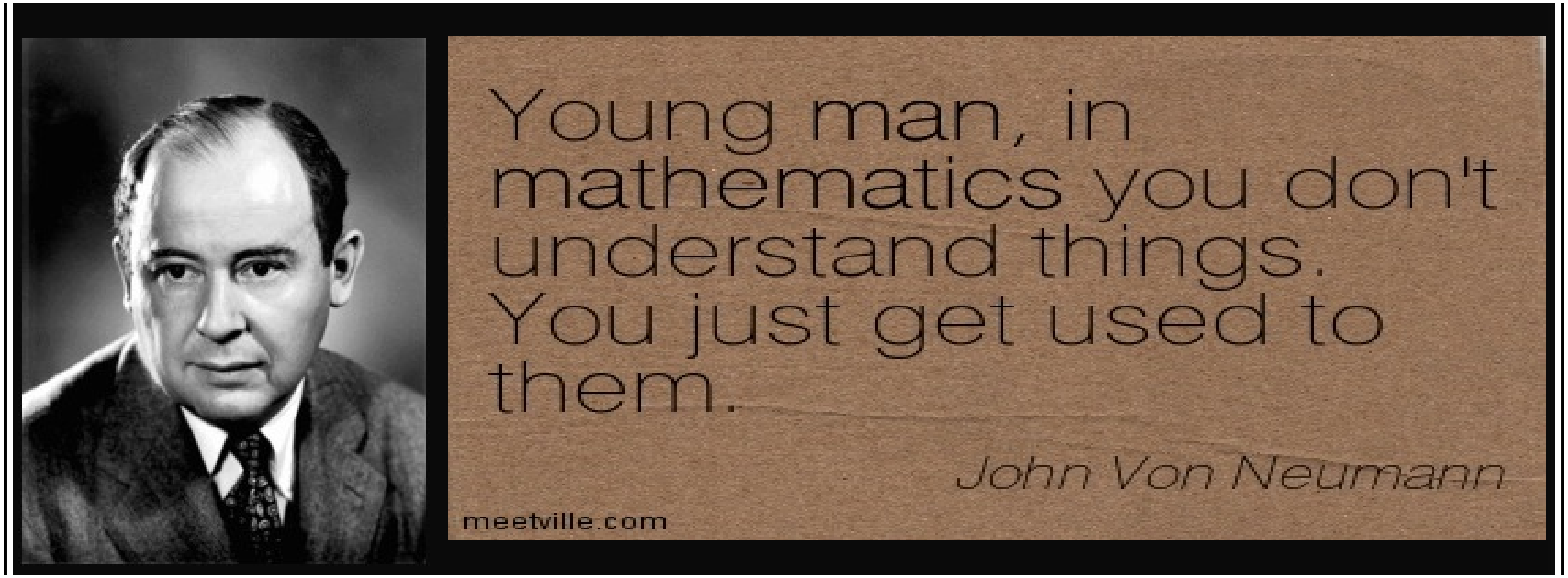
$x, y \in [0, 1]$



$x \in [0, 10^{-4}]$, $y \in [0, 1]$

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

(John von Neumann)


Young man, in mathematics you don't understand things. You just get used to them.
John Von Neumann
meetville.com

- Everything discussed so far had to do with uniformly distributed "probability" of events

- We (almost*) always used uniform random sampling, uniform random number generators etc …

- In addition, we have not explained how to do a MC simulation

- When MC technique is used to solve a problem of statistical/probabilistic nature (for example radioactive nuclei decays), then the process is called MC Simulation

* With the exception of Importance Sampling that we'll meet again in the next slides

- In MC Simulation, the function that we have to treat with the MC technique, represents the pdf of the possible outcomes of the process under study

- In some cases, this pdf is constant (uniform pdf) and a direct usage of a random number generator provides the solution to the simulation problem
(Everyday examples: Coin toss, Dice roll, Lottery, …)

- In a great number of other interesting cases, the pdf of the process is not uniform.
Then we have to cope with non-uniform sampling
(Examples: Radioactive nuclei decay, Neutron elastic scattering, Stock market)

- We will study next methods for non-uniform sampling, else called
"Sampling from (specific) distributions"

* With the exception of Importance Sampling that we'll meet again in the next slides

Monte Carlo
Inverse Transform method

- We have seen this method already in the Importance Sampling

- Quite simple and elegant, sometimes called the “Golden Rule for Sampling”

- First described in 1947 by von Neumann in a letter to Ulam:

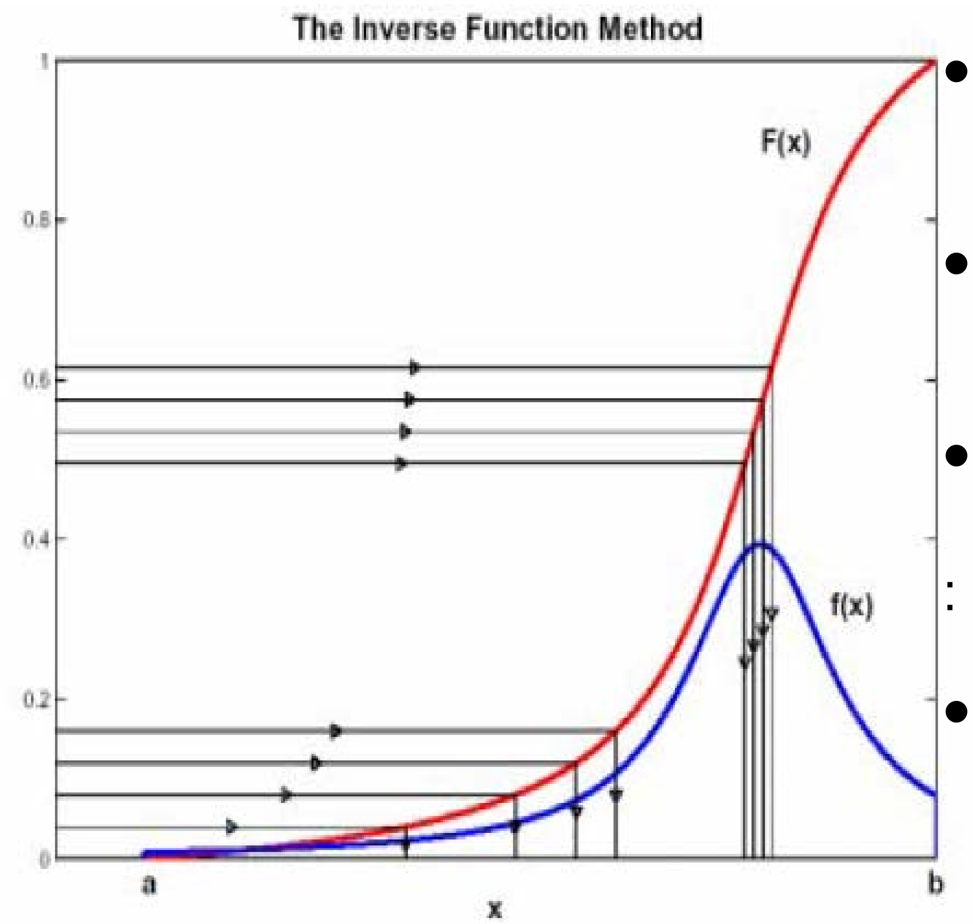
  - Sample y from a uniform distribution U[0,1]

  - Normalize pdf f(x) and calculate cdf F(x)=y
    (if f(x) is a pure pdf, then it is already normalized)

  - Invert cdf F(x) so that $x=F^{-1}(y)$

  - x is then distributed according to f(x)

- With F(x) being the cumulative of f(x), the slope of F(x) is related to the shape of f(x)



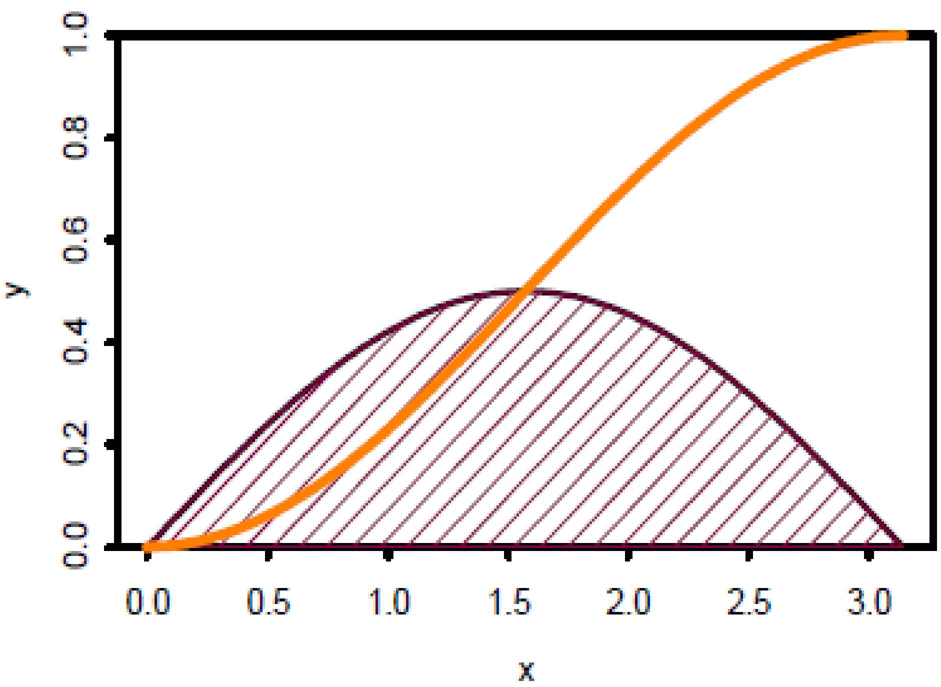
- By transforming a uniform y via $x=F^{-1}(y)$
- Achieve sampling in a non-uniform x
- Denser (most probable sampling) where F(x) has a steeper slope

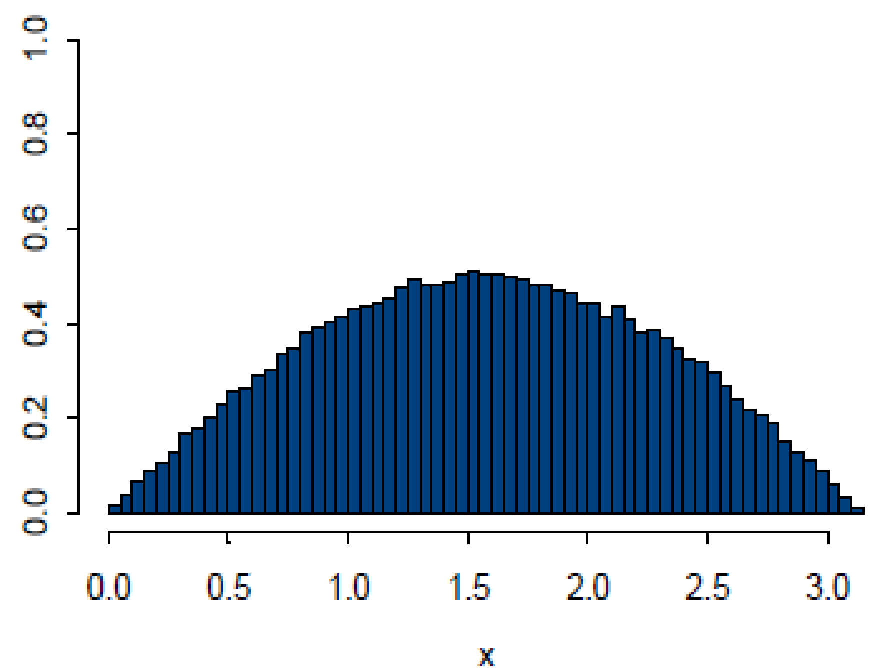
:

- Higher probability to get a random x in the region where the pdf f(x) is higher

Monte Carlo
Inverse Transform method

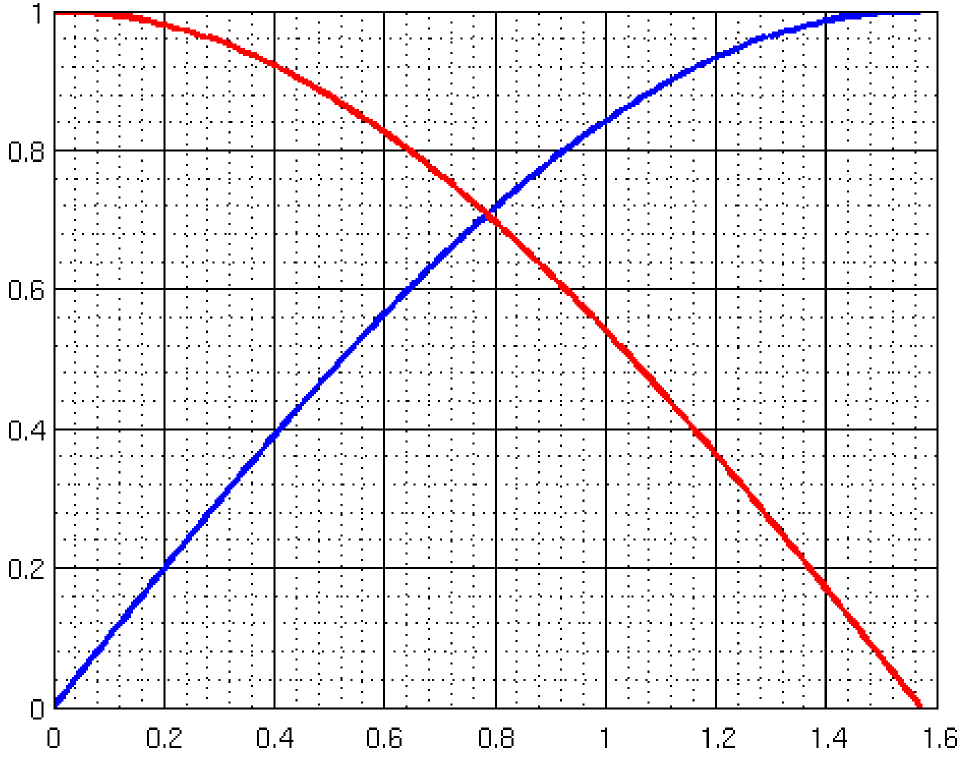
- Plotting the frequency of x will strongly resemble the pdf f(x)
  (with some statistical fluctuation depending on the total number of trials)





Monte Carlo
Inverse Transform method

- Example of f(x) = sin(x) in (0,π/2)
  - normalization: $$\int_0^{\pi/2} \sin(x)\, dx = 1$$
  - cdf calculation: $$\int_0^{x} \sin(t)\, dt = 1 - \cos(x) = y$$
  - invert cdf: $$x = \ cos^{-1}(1 - y)$$
  - sample for uniform y (0, π/2)



- There is always a “but”:

- In order to use the Inverse Transform method, cdf F(x) has to be known and simple enough to be inverted

- There is a good number of cases that the application of the Inverse Transform is impossible or too complicated

- In the next lecture we will present a flexible alternative called the Acceptance – Rejection method or simply Rejection method

- Website of the lecture material (slides):

http://www.physik.uni-muenchen.de/lehre/vorlesungen/wise_16_17/Vorlesung_-Computational-methods-in-medical-physics/vorlesung/index.html

- Exercises to be sent via email (G.Dedes@physik.uni-muenchen.de) as a zip file:
    - Containing all source code and make files
    - Short report on the findings
- The name of the zip file should be:
    - Exercise3_NameLastname.zip