A brief introduction to GEANT4 and the Monte Carlo Method

Alexandre Lindote
Office E16, alexandre.lindote@uc.pt

Logistics and grading

- 7 classes in total (tentative dates: 19-26/09, 3-17-31/10, 14-28/11)
- Slides and class material will be available here, slides shared on UCStudent: https://www.lip.pt/~alex/G4Classes/
- Total of 5 values for the GEANT4 mini-course
  - 2 (very easy) home-works: **0.5 each**
  - 2 simple simulations (probably done in class with my help): **0.75 each**
  - Deadline for submission is **2 weeks**
  - Final project: **2.5**
    - deadline to be defined, but before the final exam
- For the homeworks I expect a simple report with relevant plots (1-2 pages) and the code
- For the project you'll have to write a detailed report (more details later)
- You can do this on your own, but **preferably** in groups of 2

More on logistics…

- Computers in this room have the required software (GEANT4 and ROOT)
  - We will use Linux, not Windows
  - You must select **Ubuntu** upon restart
  - By default, we will use ROOT for analysis of the results, but feel free to use a different software if you're familiar with it (GNUPlot, Python, MatLab, etc.)
- Later on you will need to use GEANT4 on your own computer
  - You will need it for the final project
  - I'll send instructions for the installation
  - It will (hopefully!) work on Windows, macOS and Linux
- If you don't have a computer, or have problems installing GEANT4, you can use the computers in this room

Plan for the classes

- **Lesson 1:**
  - Concept of Monte Carlo simulation
  - Random numbers
  - Distribution generators
  - Some examples of Monte Carlo sampling
- **Lesson 2:**
  - What is Object Oriented Programming?
  - Introduction to GEANT4
  - Basic simulation structure — Mandatory classes in GEANT4
  - Concept of Run, Event and Track
  - Basic geometry concepts in GEANT4 (materials and volumes)
  - Visualisation tools
  - Particle generators and particle tracking

Plan for the classes

- **Lesson 3:**
  - List of available particles and physics processes
  - Following the simulation in real time (step-by-step)
  - Optional (but very useful) classes
  - Storing simulation results
  - Running the simulation in batch mode
  - A simple example: simulate the Bragg peak for alpha particles
- **Lessons 4 — … :**
  - More examples: gamma shielding, radioactive decay, range of electrons, neutron interactions, etc.
  - Distribution of the final projects before the last lesson

Lesson 1 - Monte Carlo Simulation

Monte Carlo Simulation

- Useful when the problem is too complex for an analytical solution
- The goal is to predict the evolution of complex systems using the known probability (and final state) of each individual process
- Each probability is described by a function (or known distribution), which is randomly sampled
- It is possible to obtain an approximation of the mean response of the full system by running the simulation many times
- In physics, distributions are usually in time (*e.g.* radioactive decay) or in space (*e.g.* Compton scatter), but more complex quantities are also used (*e.g.* final energy and angular distribution after nuclear decays)

Monte Carlo Simulation

- This method was developed in Los Alamos during World War II, by people working in the Manhattan Project
- It was a secret project, so it (obviously) needed a catchy code name: "Monte Carlo"
  - from the similarity with games of chance in the Monte Carlo casino

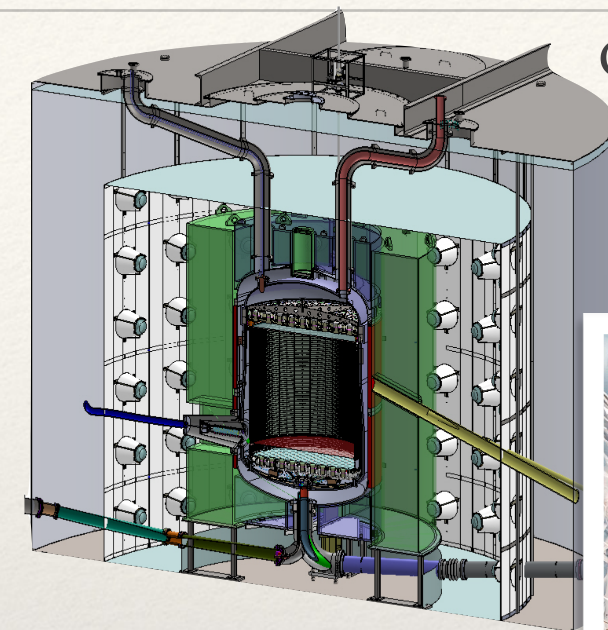
- First used to estimate shielding requirements for gamma radiation and neutron scattering (nuclear bombs)
(we will do both, yay!)
- Used in many other scientific areas (meteorology, economy, social sciences, etc.)

Monte Carlo Simulation

- Some frequent uses in physics:
  - designing and optimising experiments
  - developing data analysis ahead of running an experiment
  - help interpreting experimental results
- It is nowadays a crucial component in every large physics experiment!
- Widely used in medical physics too, in the development and optimisation of imaging techniques, but also in treatment planning (*e.g.* proton therapy, brachytherapy)

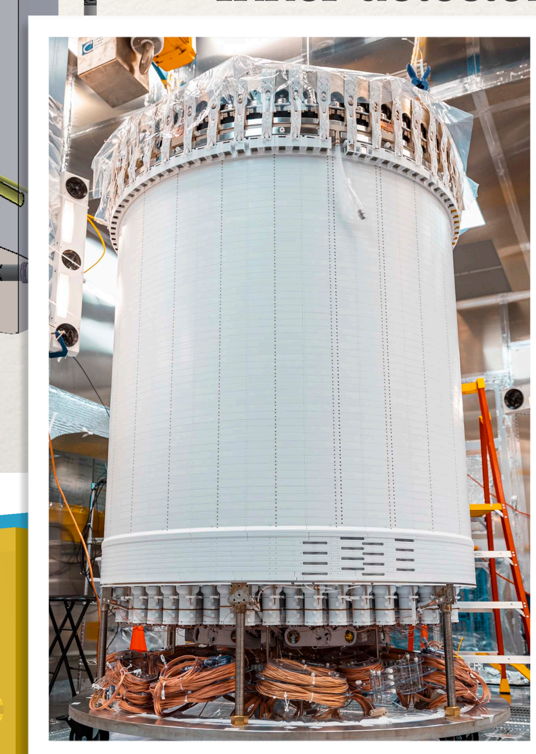
An example from my own experience

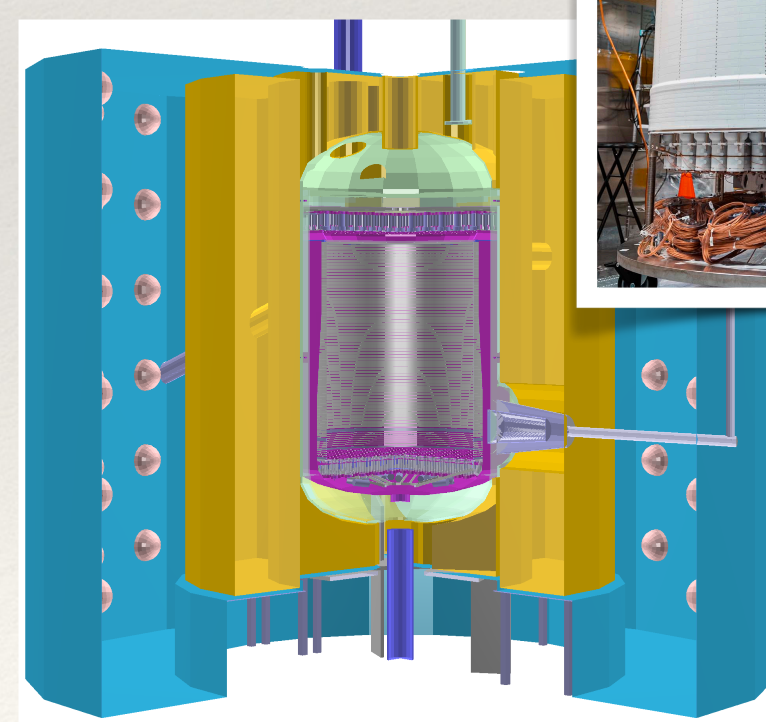
- I work in the LUX-ZEPLIN (LZ) experiment, a detector to search for interactions of dark matter with (normal) baryonic matter
- This is a detector with 10 tons of liquified xenon, working 1.5 km underground in an old gold mine
- From the design and optimisation to construction and installation, several years are needed (the concept started in 2013, the detector only started operating in 2021!)



CAD design

Inner detector





GEANT4 geometry rendering

An example from my own experience

- In the meantime, the Monte Carlo simulation of the experiment is used to:
  - Optimise the geometry of the various subsystems
  - Select building materials based on their radioactive content
  - Generate fake data to develop the data processing and analysis tools
  - Estimate how sensitive the experiment will be (basically its physics reach)

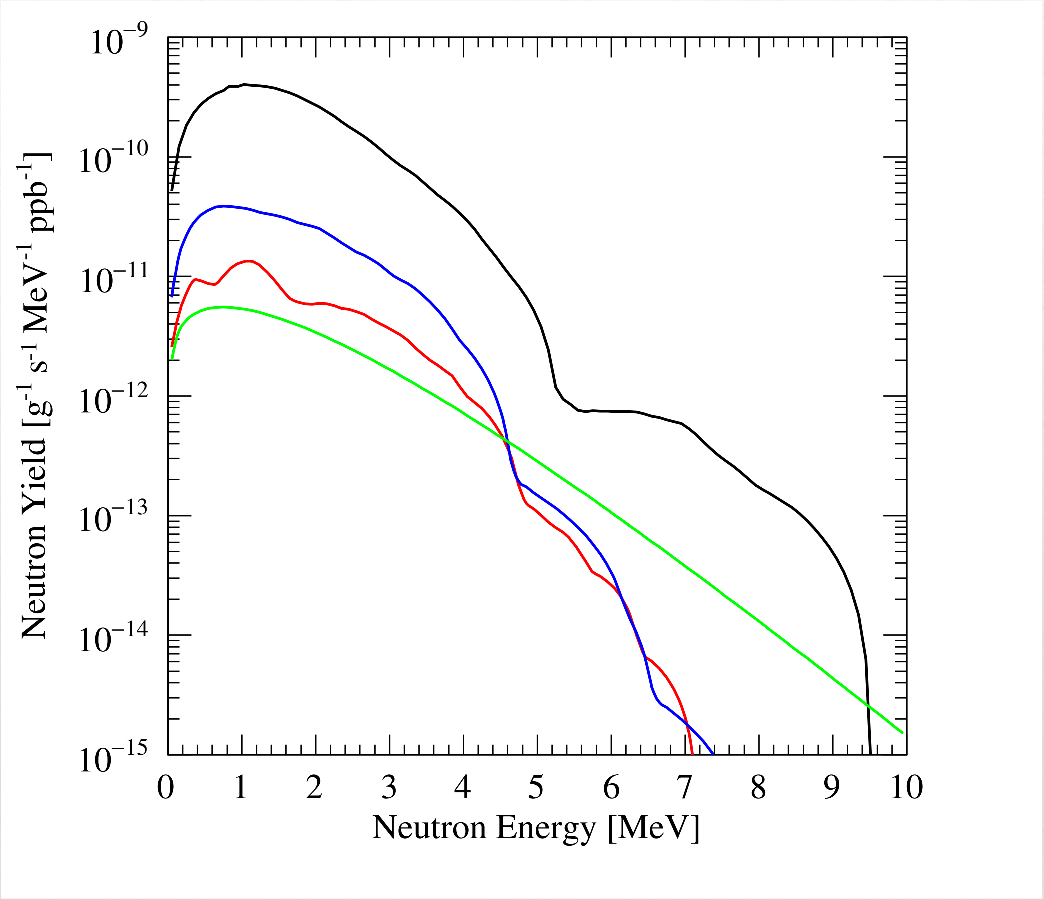
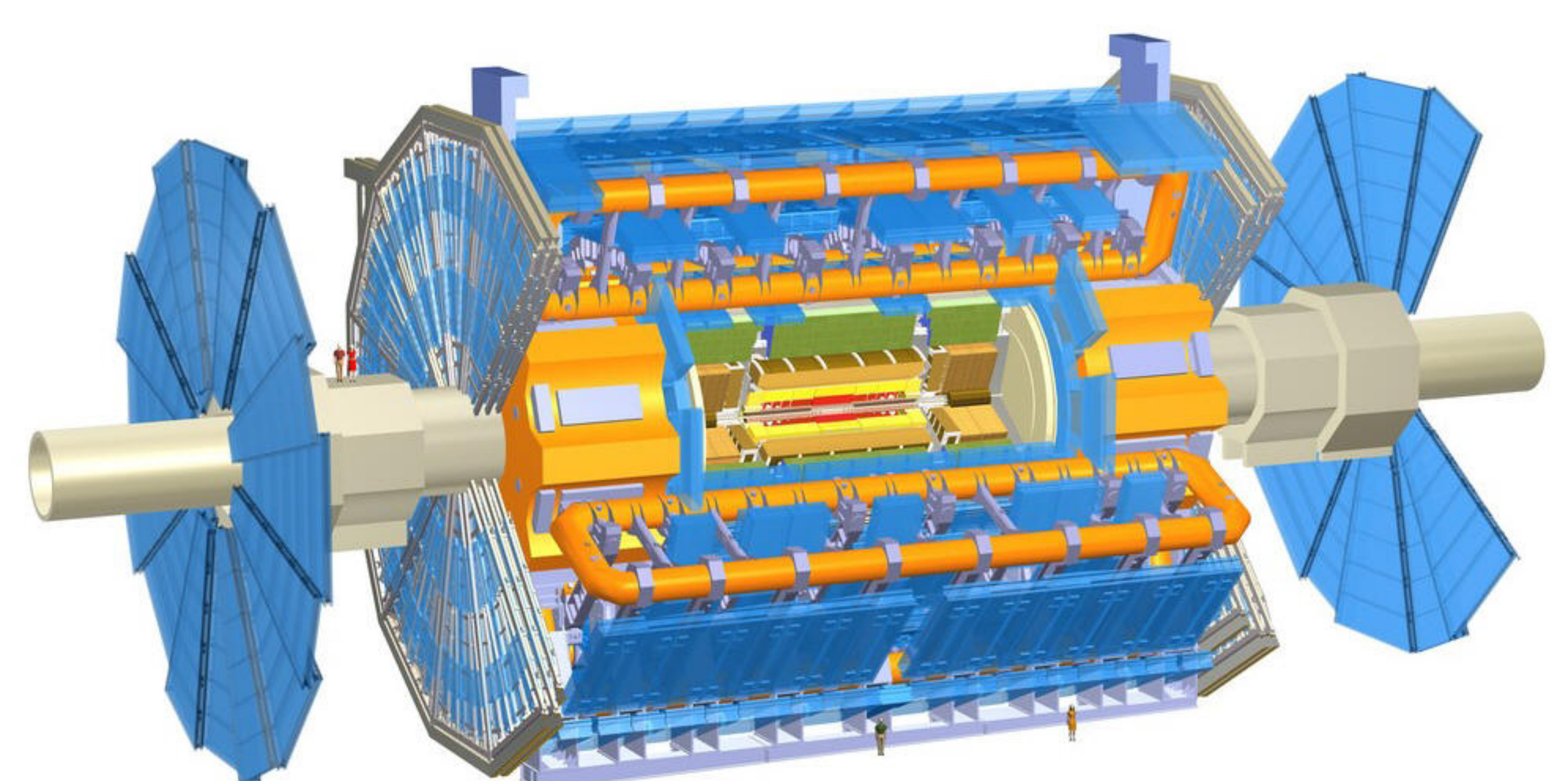


FIG. 10. Neutron spectra from (α, n) reaction from uranium decay chains in equilibrium (^{238}U and ^{235}U are combined together) in 3 materials: black - PTFE (C_2F_4), blue - ceramics (Al_2O_3), red - titanium. The green curve shows the spectrum from spontaneous fission (same for all materials).

Monte Carlo Simulation

- Maybe the best example of MC use in physics are the LHC experiments



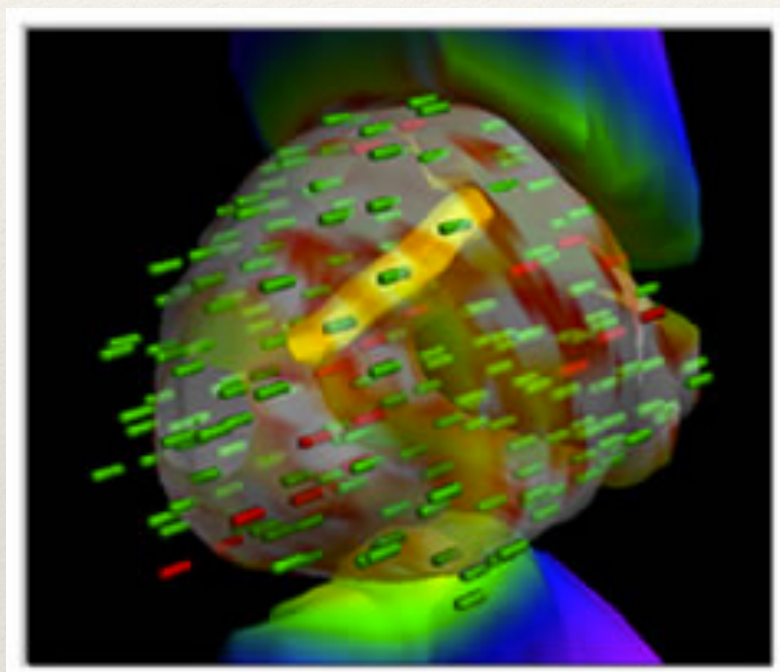
GEANT4 rendering of the ATLAS detector

In fact, GEANT4 was developed for the LHC experiments!

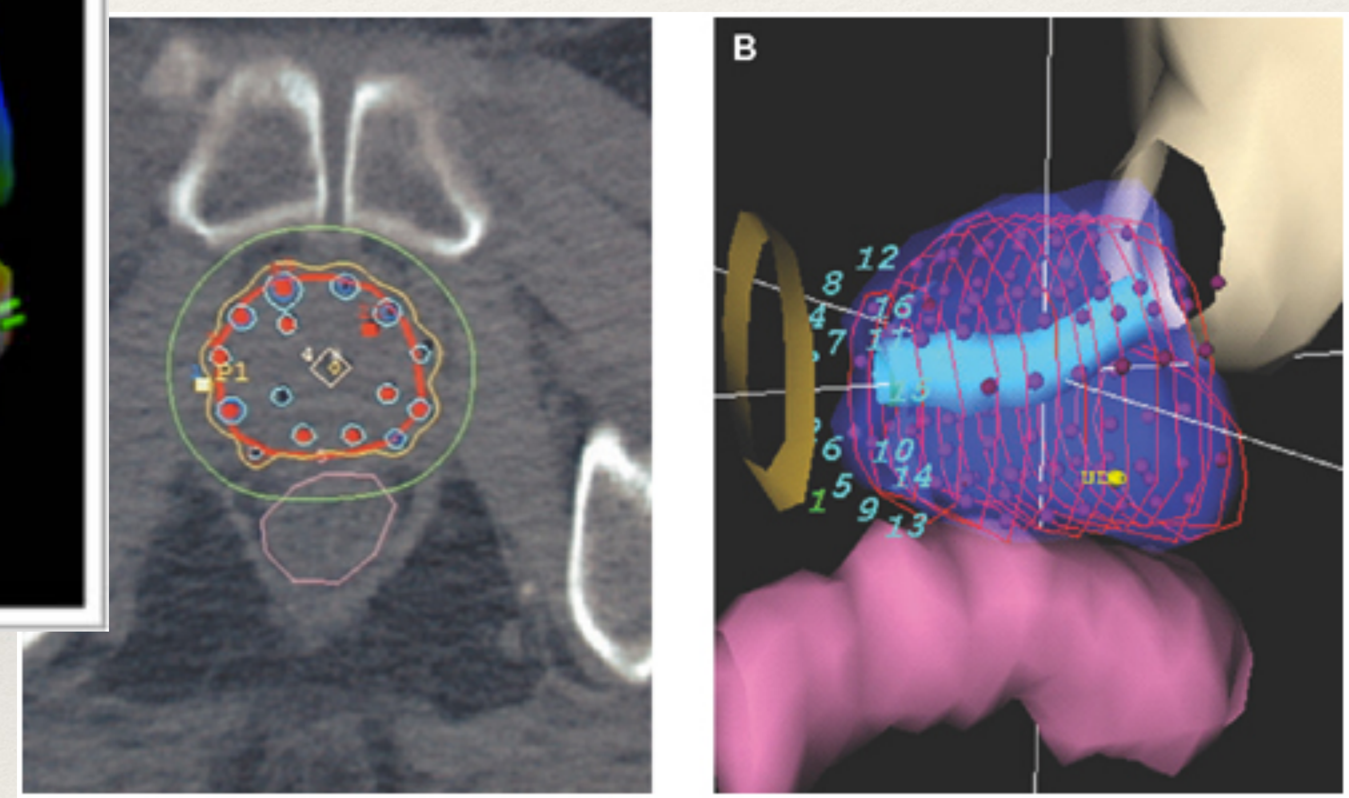
Monte Carlo Simulation

- In medical physics: brachytherapy for prostate cancer treatment

^{125}I seeds



Radioactive seeds implanted in a prostate



Iso-dose contours

Monte Carlo Simulation

- Some frequent uses in physics:
  - designing and optimising experiments
  - developing data analysis ahead of runnin…
  - interpreting experimental resul…
- It is nowadays … in every large experim…
- … medical physics too, in the development … ptimisation of imaging techniques, but also in treatment planning (*e.g.* proton therapy, brachytherapy)

Critical in every simulation is the ability to generate "high quality" random numbers efficiently

Random Numbers

Random Numbers

- How to generate random numbers:
  - Get a random number table
  - Using random processes:
    - throw a dice
    - draw numbers from a hat
    - use a number pool from electronic noise (*e.g.* */dev/random* in Unix/Linux)
      Try this command in a terminal:
      ➡ *$ od -An -N2 -i /dev/random*

- Actually, this is not what we want
  - we need a sequence that can be reproduced
    ➡ allows us to repeat the simulated "experiment" if necessary (*e.g.* study particular events, solve problems with the code, share results)
  - must be fast and easy to use

Pseudo-random numbers

- Not actual random numbers, but rather a sequence of seemingly uncorrelated numbers that can be easily reproduced
- Generated using an iterative algorithm
  - each new “random” number is generated using one (or more) of the previous ones
  - using the same initial value (called ***seed***) it is always possible to reproduce the sequence

Pseudo-random numbers

- What do we want from a pseudo-random generator?
  - a distribution between 0 and 1
    - easy to transform into whatever interval we want
    - can be used to generate non-uniform distributions (more on this later)
  - speed
  - reproducibility
  - a very long period
    - number of generated numbers until the sequence starts repeating itself
  - must be statistically consistent with a random sequence:
    - uniform distribution, non-sequential numbers, etc.
    - there are several tests (which we will not cover in detail, see, *e.g.* *http://www.maths.uq.edu.au/~kroese/mccourse.pdf*)

Example of a random number generator

- LCGs (*Linear Congruential Generators*)

$$X_t = (aX_{t-1} + c) \bmod m\ , \quad t = 1, 2, \ldots\ ,$$

mod is the remainder of the integer division

  - a, c and m are integers (usually $c=0$)
  - period is, at most, m (depends on the remaining parameters)
  - $R_t = X_t / m$ — used to normalize to the interval [0, 1]
  - The first value used (X_0) is called the ***seed***

- Simple example: $a=5$, $m=32$, $c=0$. Initial seed, $X_0=3$

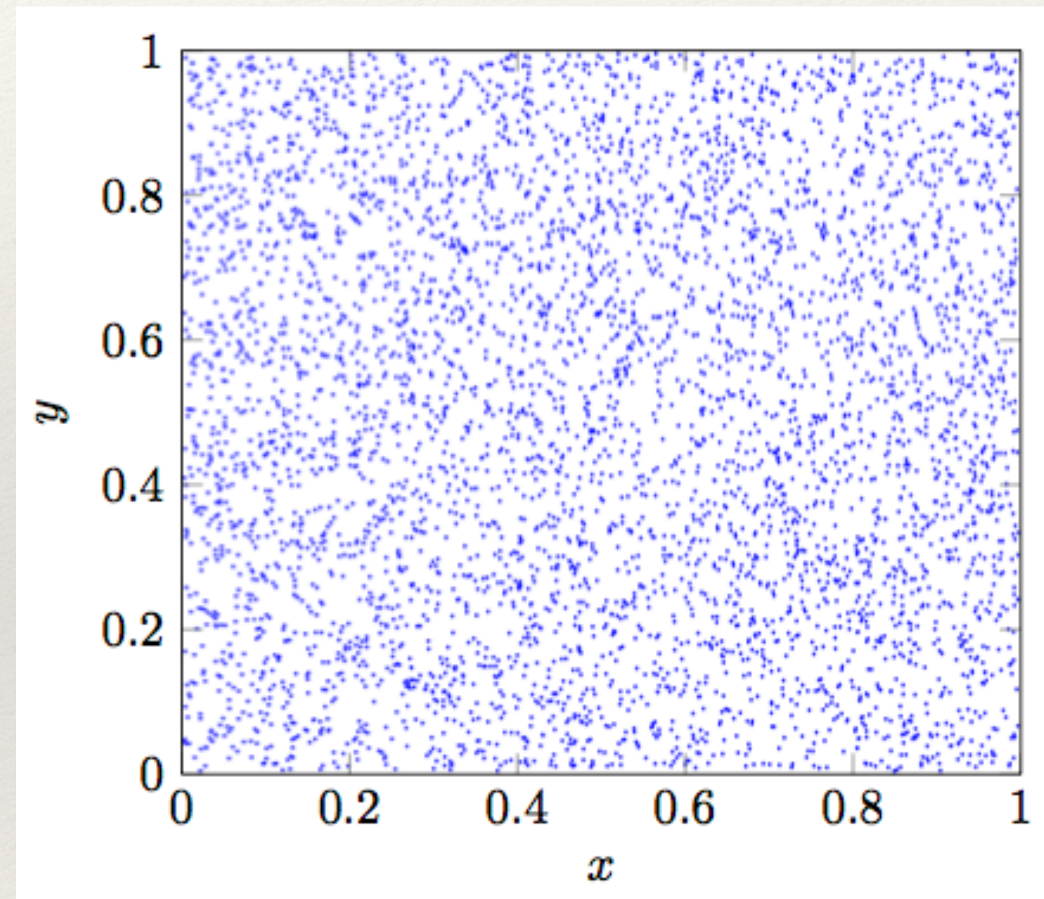
  i) $X_1 = 5*3 \bmod 32 = 15$ ($R_1 = 0.46875$)

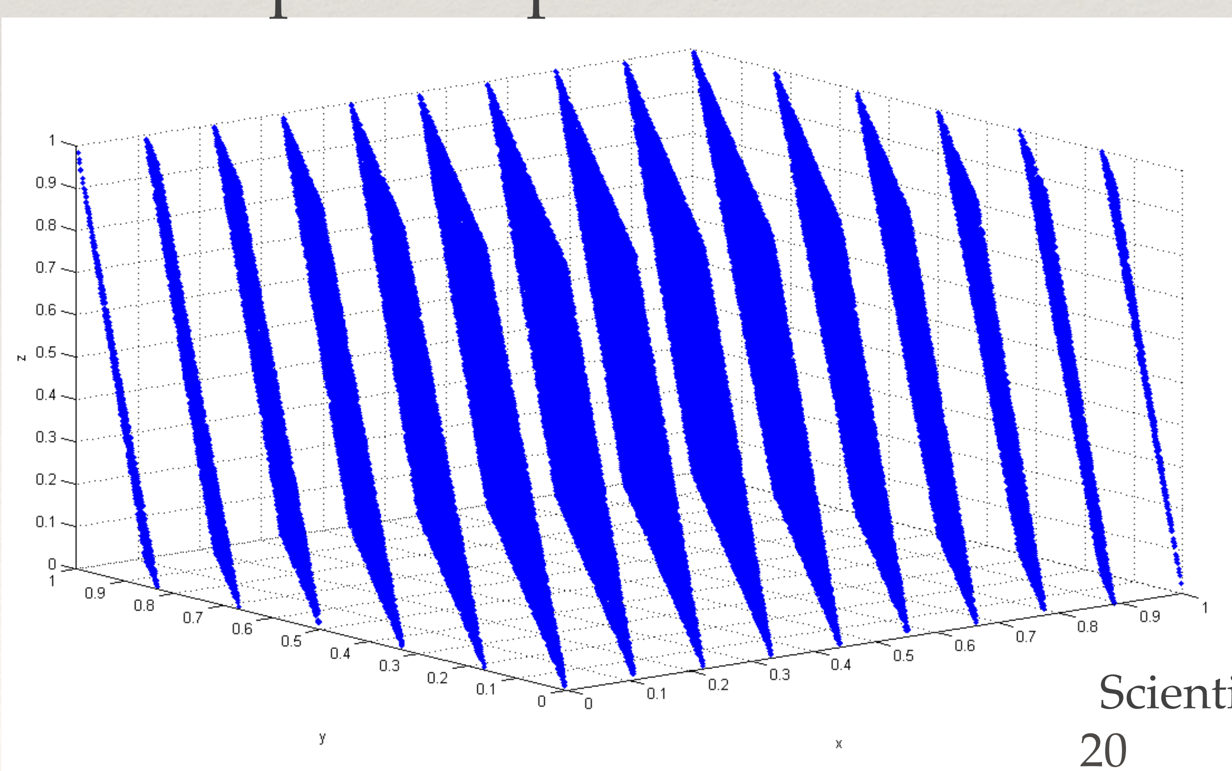
  ii) $X_2 = 5*15 \bmod 32 = 11$ ($R_2 = 0.34375$)

  iii) $X3 = 5*11 \bmod 32 = 23$ ($R_3 = 0.71875$)

LCG Generators

- Quality depends heavily on the choice of parameters
  - *Minimal standard* (available in C++11): $a = 7^5 = 16807$, $c = 0$, $m = 2^{31} - 1$
  - RANDU (IBM, 1960s-70s) $a = 2^{16}+3 = 65539$, $c = 0$, $m = 2^{31}$ Sequences of 3 consecutive numbers fall in parallel planes!





This was the default in every IBM computer, widely used at the time.
Scientific results of this period were affected by this problem

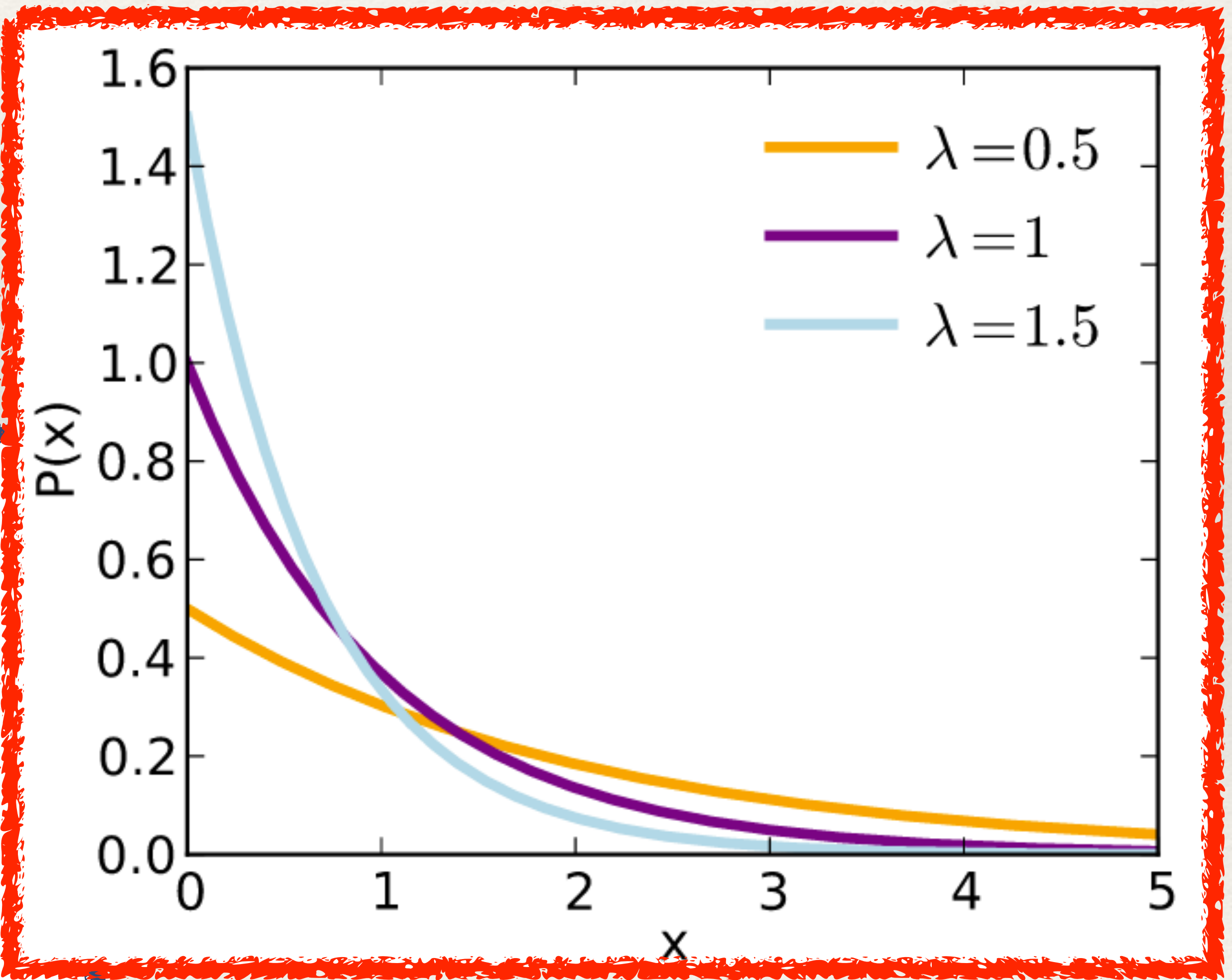
Exercise 1

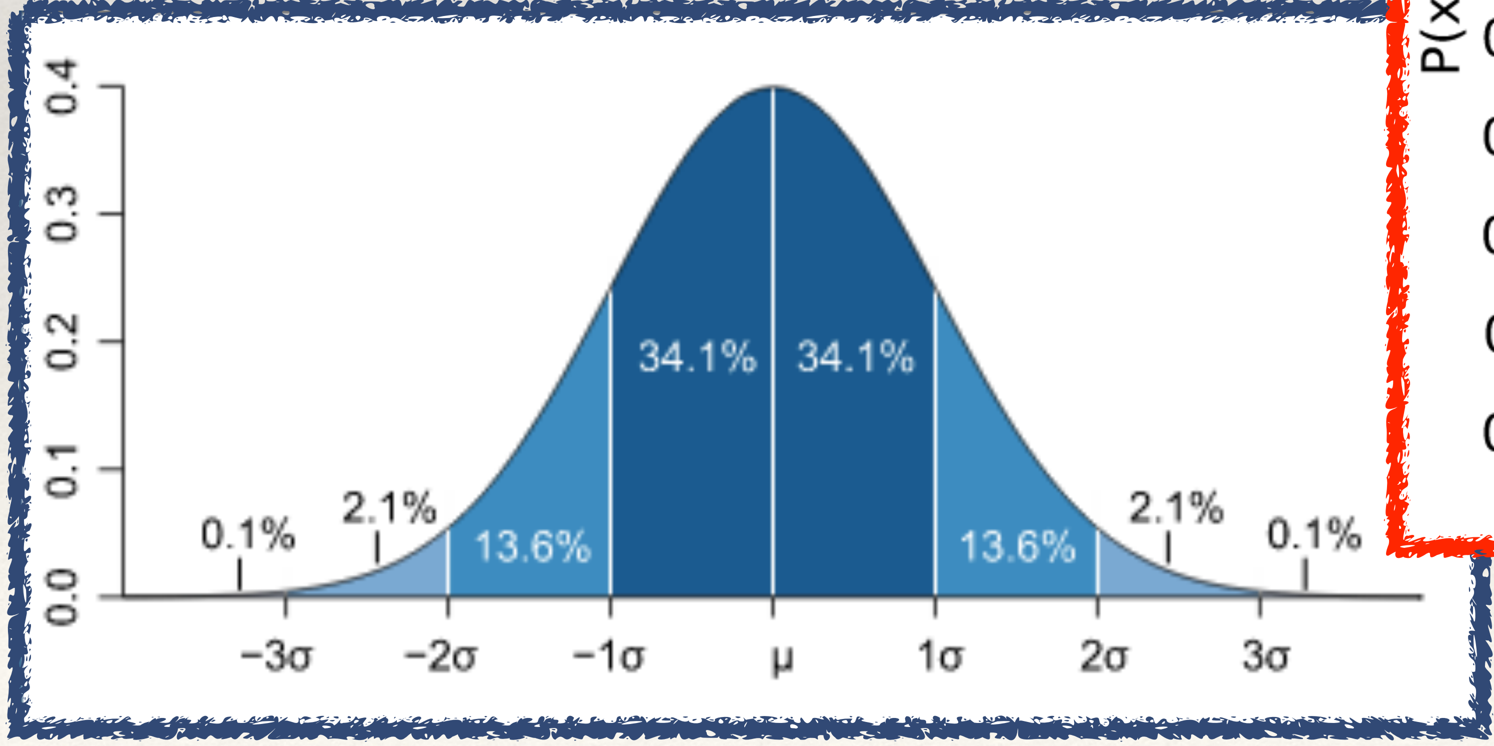
- Test the uniformity of the *stdlib* (C, C++) random generator (*exercise01.cc*)
  - This is an LCG generator
  - m depends on the specific system (stored in variable *RAND_MAX*)
  - get a sequence of N random numbers between 0 and 1
  - make the histogram using ROOT (use the plot_histogram.C script)
    - *root -l*
    - *.x plot_histogram.C*
    - *.q* (to quit root)
  - Vary N and check evolution of the average and the standard deviation (*e.g.* N = 100, 1k, 10k, 100k, 1M, 10M)

Download the prototype code from www.lip.pt/~alex

Non-uniform distributions

- In general, the distributions we need are <u>not uniform</u>:
  - ➡ Exponentials (*e.g.* radioactive decay)
  - ➡ Gaussians (*e.g.* energy resolution)
  - ➡ Other analytic functions
  - ➡ Experimental data points



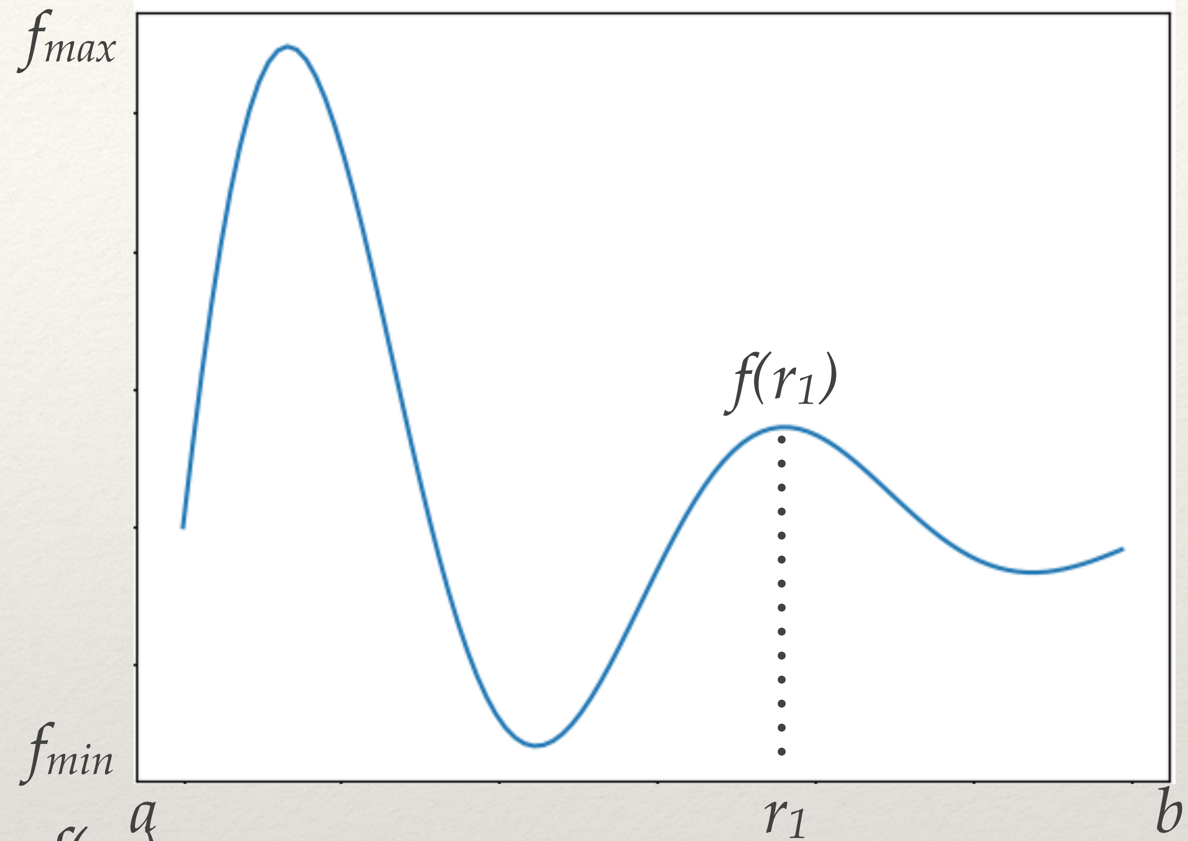


Methods to generate non-uniform distributions

- We'll only talk about the two more usual ones:
    - Rejection method
    - Inverse transform method

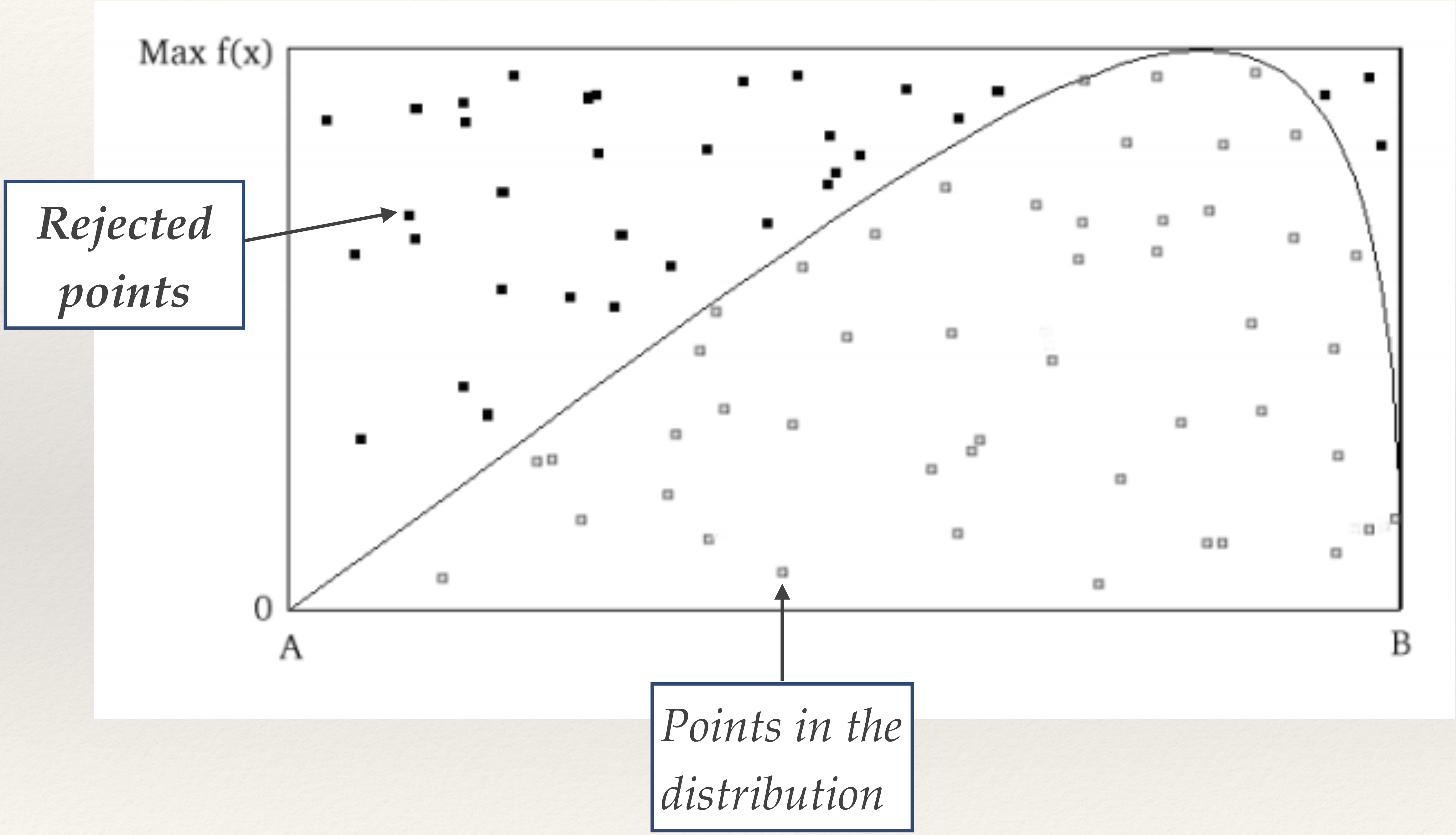
Rejection Method

1. Enclose the function to sample (f) using a rectangle:
   - $a < x < b$; $f_{min} < y < f_{max}$
   - it may be necessary to ignore long tails…
2. Using a uniform generator, sample in x between the limits of the rectangle: $r_1 \in [a, b]$



3. Estimate the value of the function at point r_1: $f(r_1)$
4. Sample a second random number within the vertical limits of the rectangle: $r_2 \in [f_{min}, f_{max}]$
5. If $r_2 \leq f(r_1)$ we can take r_1 as a sample from distribution f
6. If $r_2 > f(r_1)$, discard r_1

Rejection Method



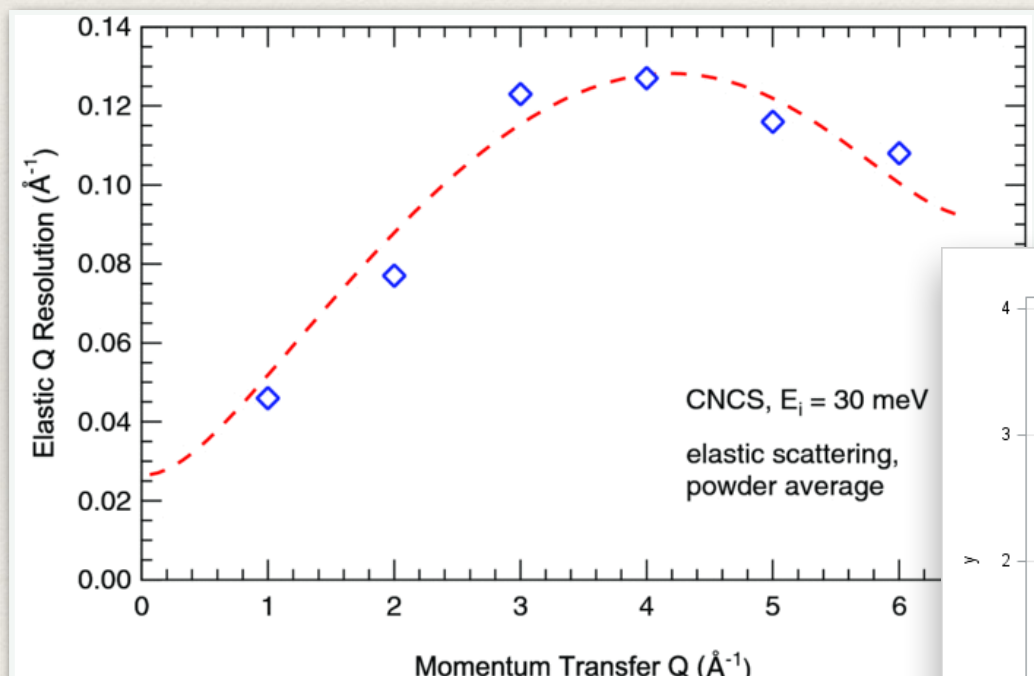
Using a sufficiently large number of samples, we can reproduce the function *f(x)*

Rejection Method

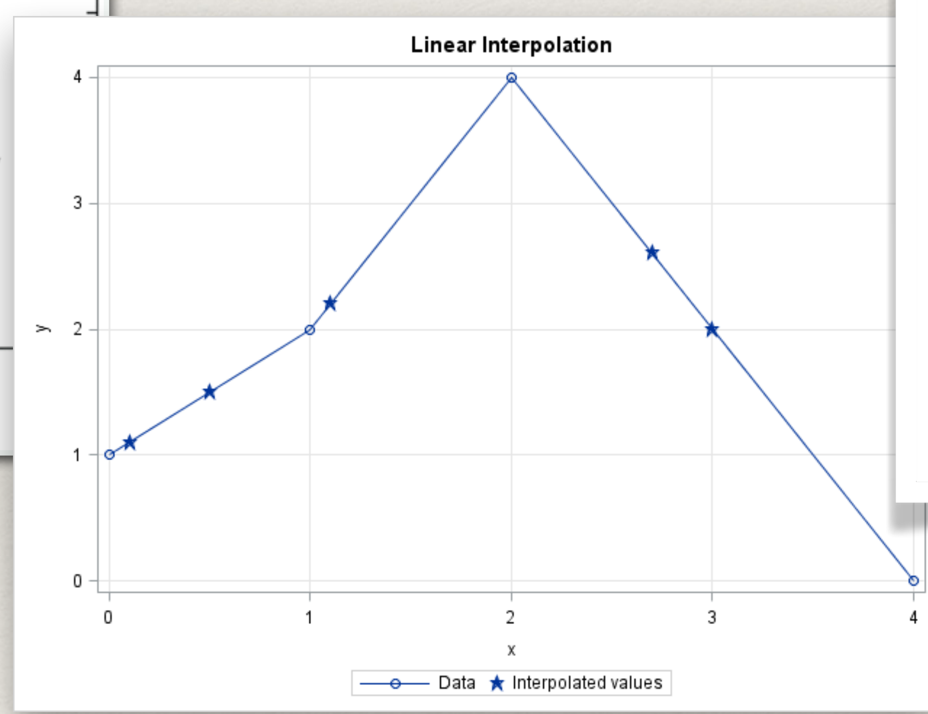
- Based on the ratio between the total area of the rectangle and the area under the function to sample
- This means it can also be used to estimate areas!
  - Using the ratio of accepted / total samples
- Pro: can be used with any function
- Con: computationally slow
  - needs 2 uniform randoms for each trial
  - the function must be calculated (at least once) in each iteration
  - a (possibly significant) fraction of the trials is rejected

Rejection method

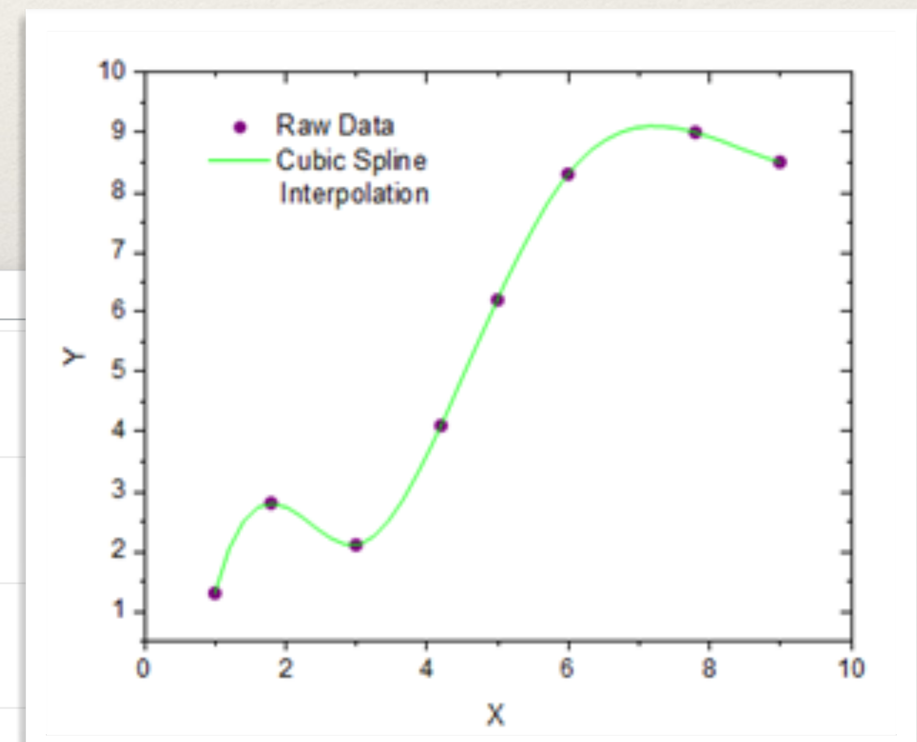
- For a data based distribution we can use a fit to the data, or interpolate between consecutive data points



Function fit



Linear interpolation



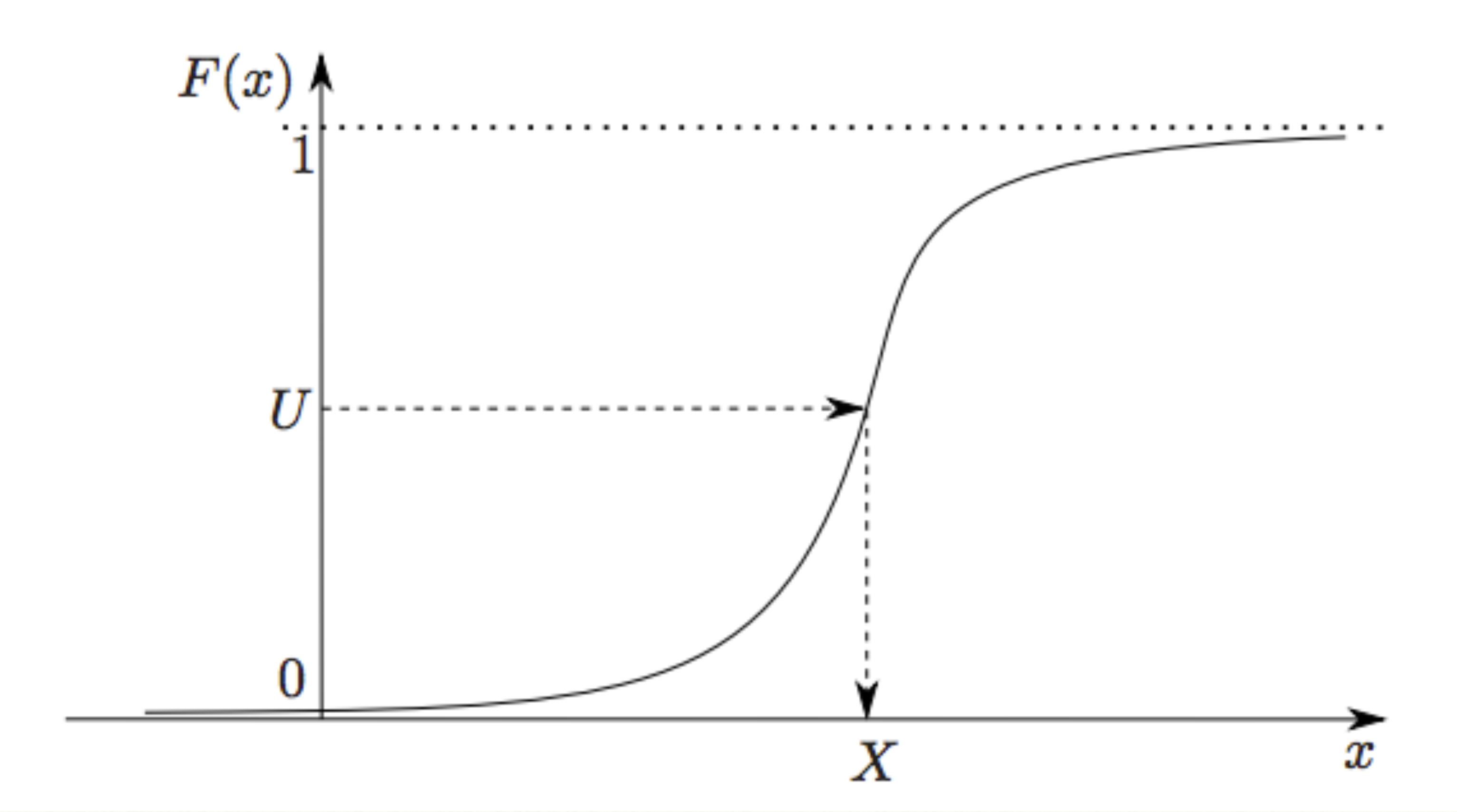
Cubic spline interpolation

Inverse Transform Method

1. Get the **cumulative distribution function** (*cdf*) of the (probability) function to sample:

$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t$$

2. The *cdf* is the fraction of the integral of the function up to the value *x*. It can be interpreted as the probability of obtaining a value smaller than *x* when sampling the function
   - ➡ <u>grows continuously</u>
   - ➡ <u>has a maximum of 1</u>



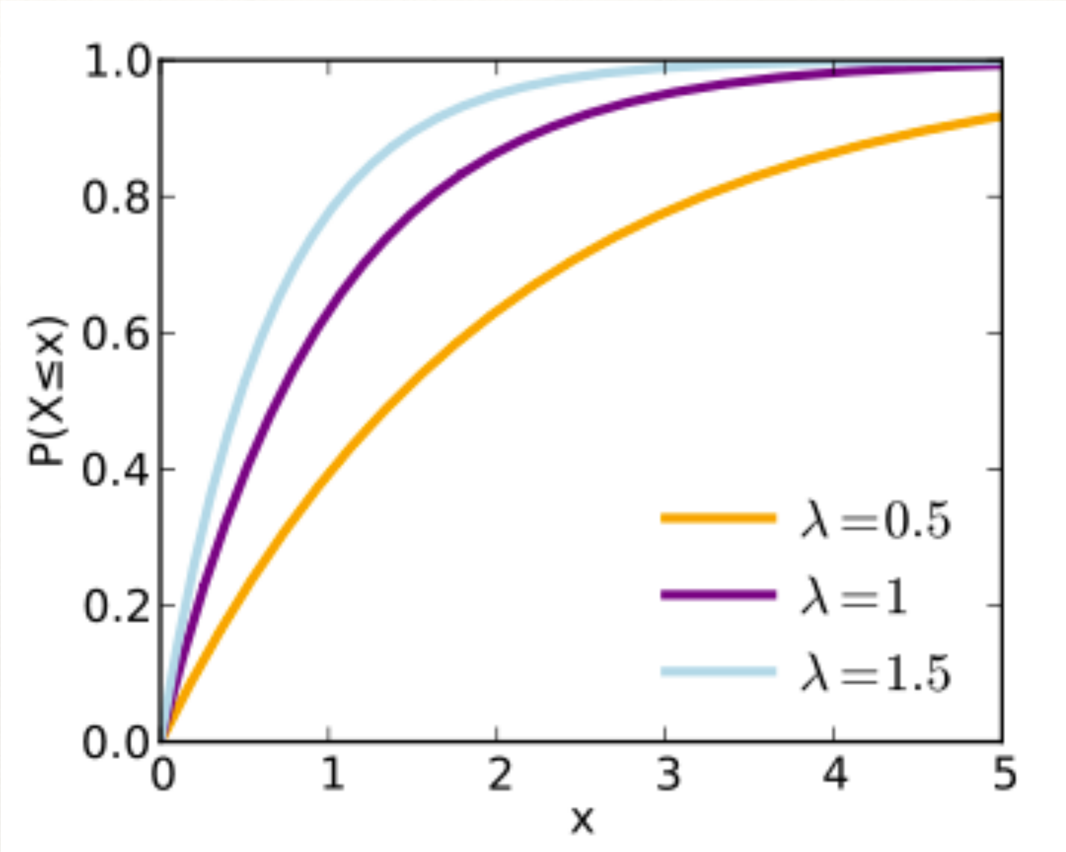
3. Get a random from a uniform distribution (*U*), which is a sample in *F(x)*

4. We may now get a sample of *f(t)* by inverting *F(x)*

$$X = F^{-1}(U)$$

Inverse Transform Method

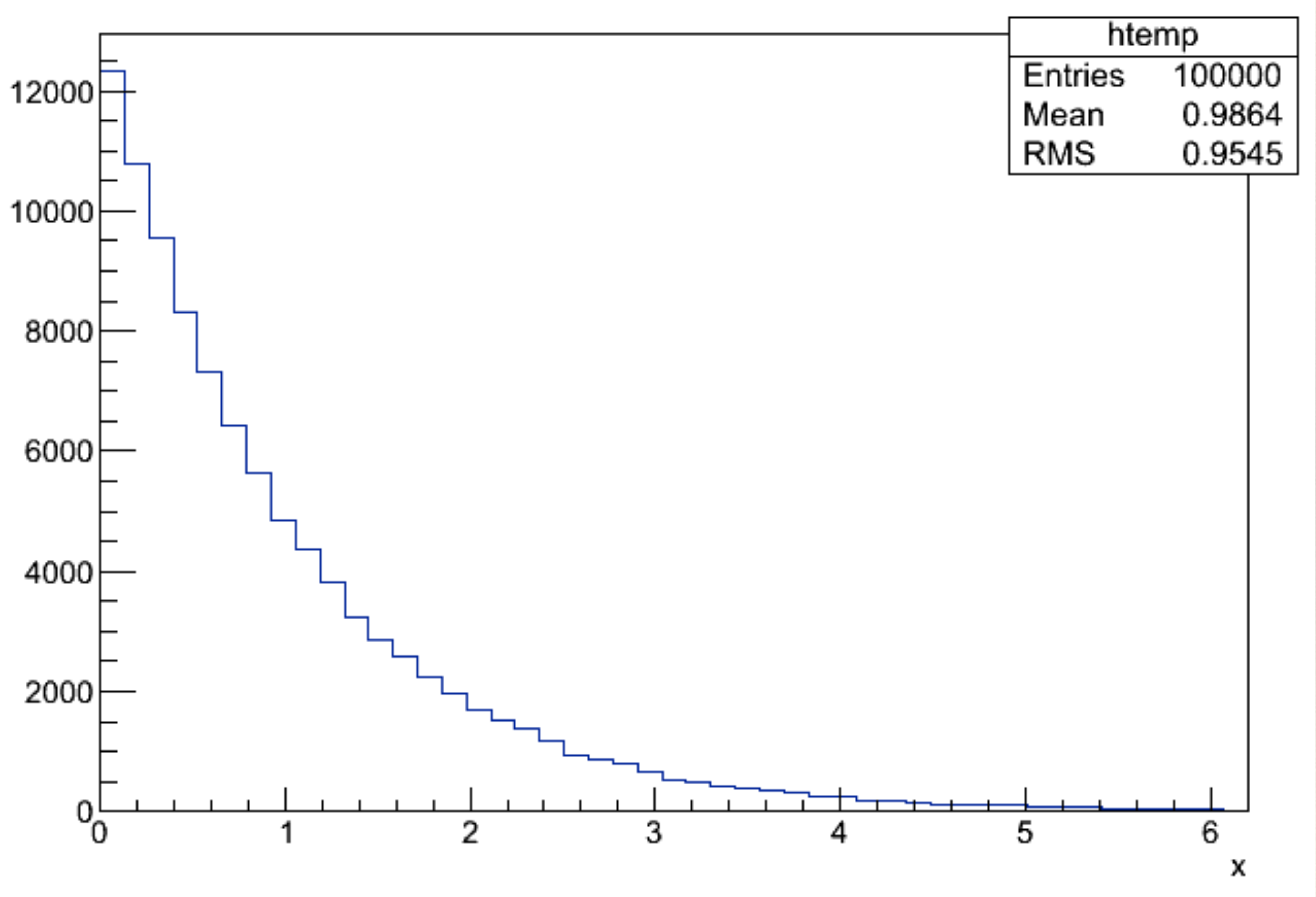
- A simple (but very useful) example: exponential distribution
  - ➡ $f(t) = \lambda e^{-\lambda t}$
- The cumulative function is
  - ➡ $F(x) = 1 - e^{-\lambda x}$



- Using a random number (U) from a uniform generator, we get a new random number (x) which follows the exponential distribution:
  - ➡ $x = F^{-1}(U) = -ln(1 - U)/\lambda$

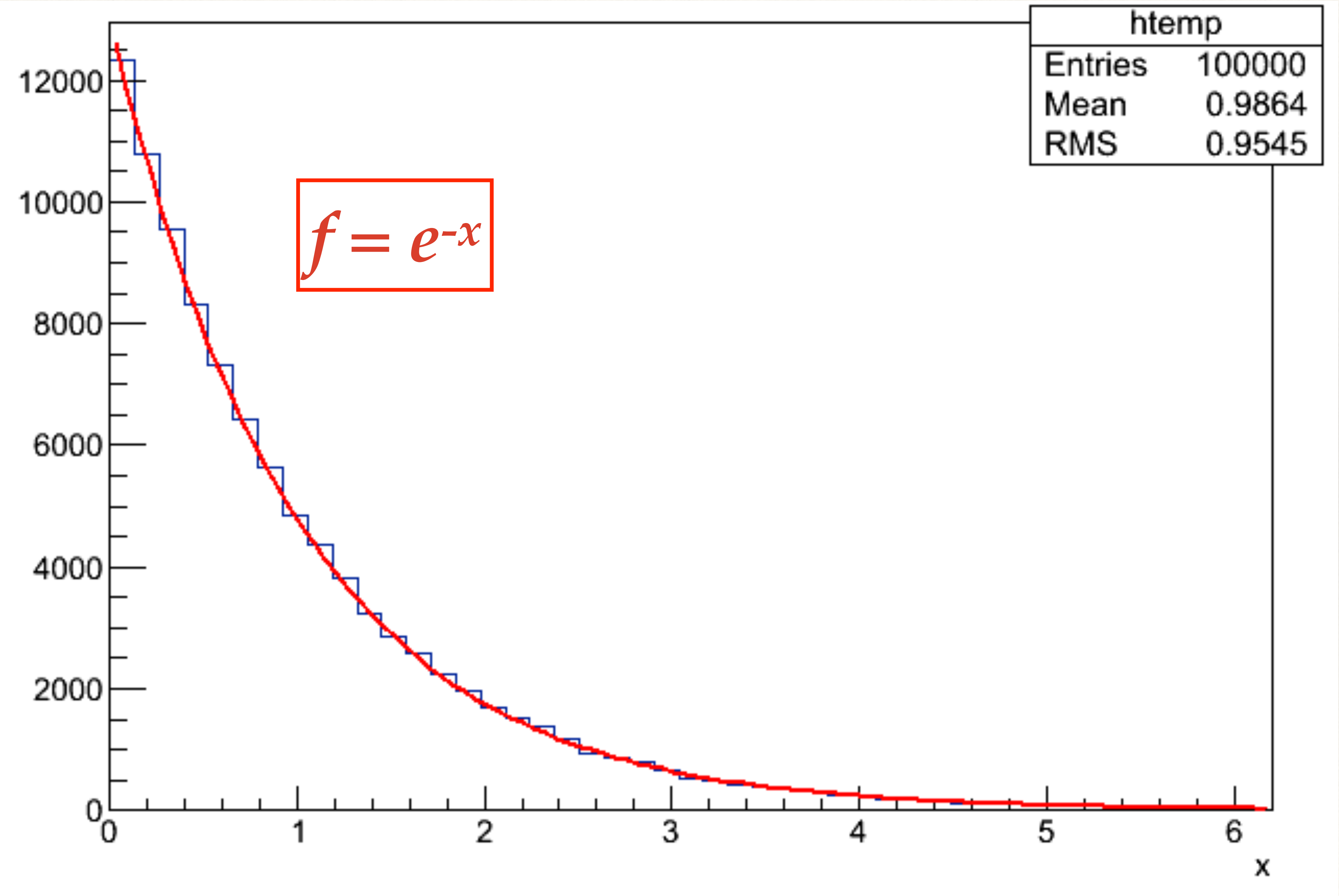
Inverse Transform Method

- Distribution obtained with 100k samples



Inverse Transform Method

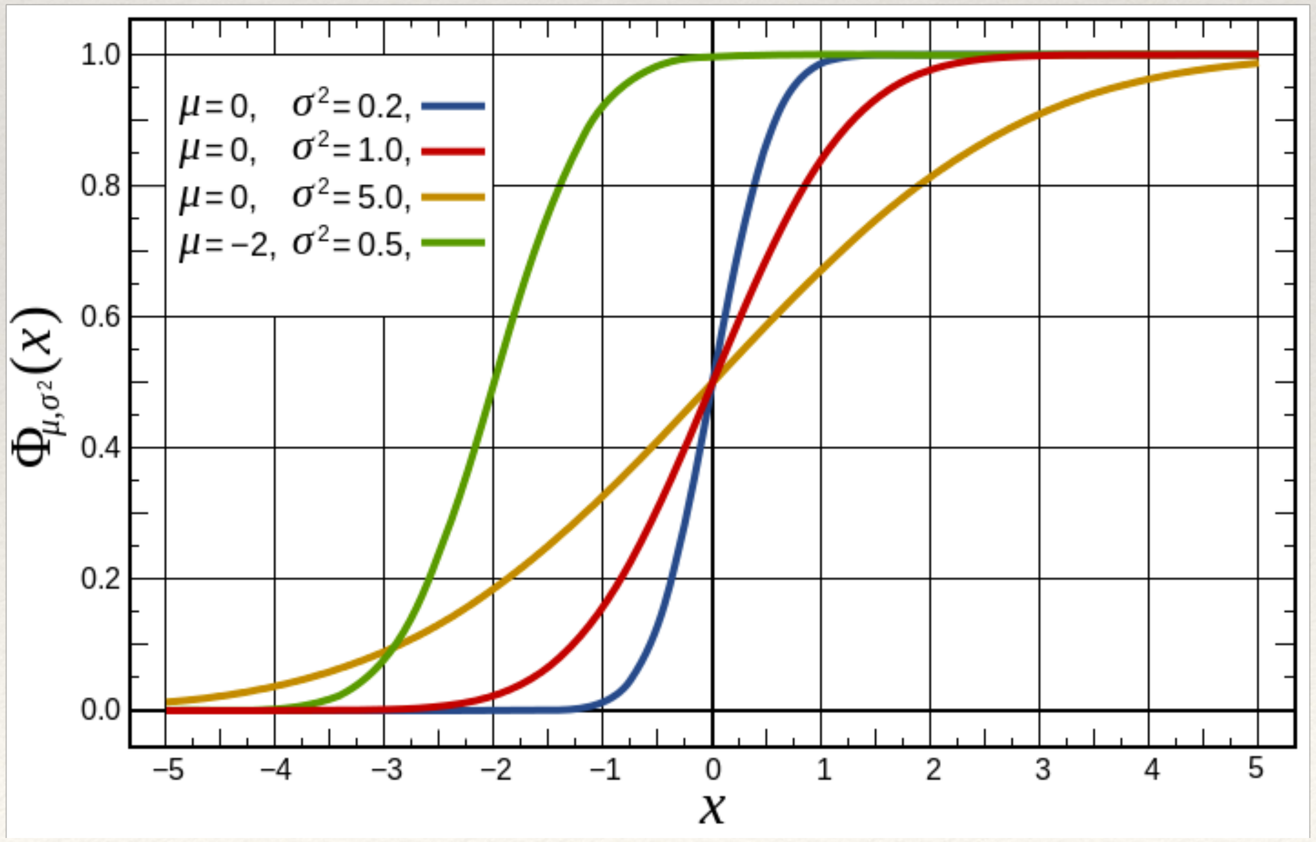
- Distribution obtained with 100k samples



Gaussian distribution

- Also called *normal* distribution
- Probably the most useful distribution in physics
- No exact integral, but there are several numerical approximations to the *cdf*

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Gaussian distribution

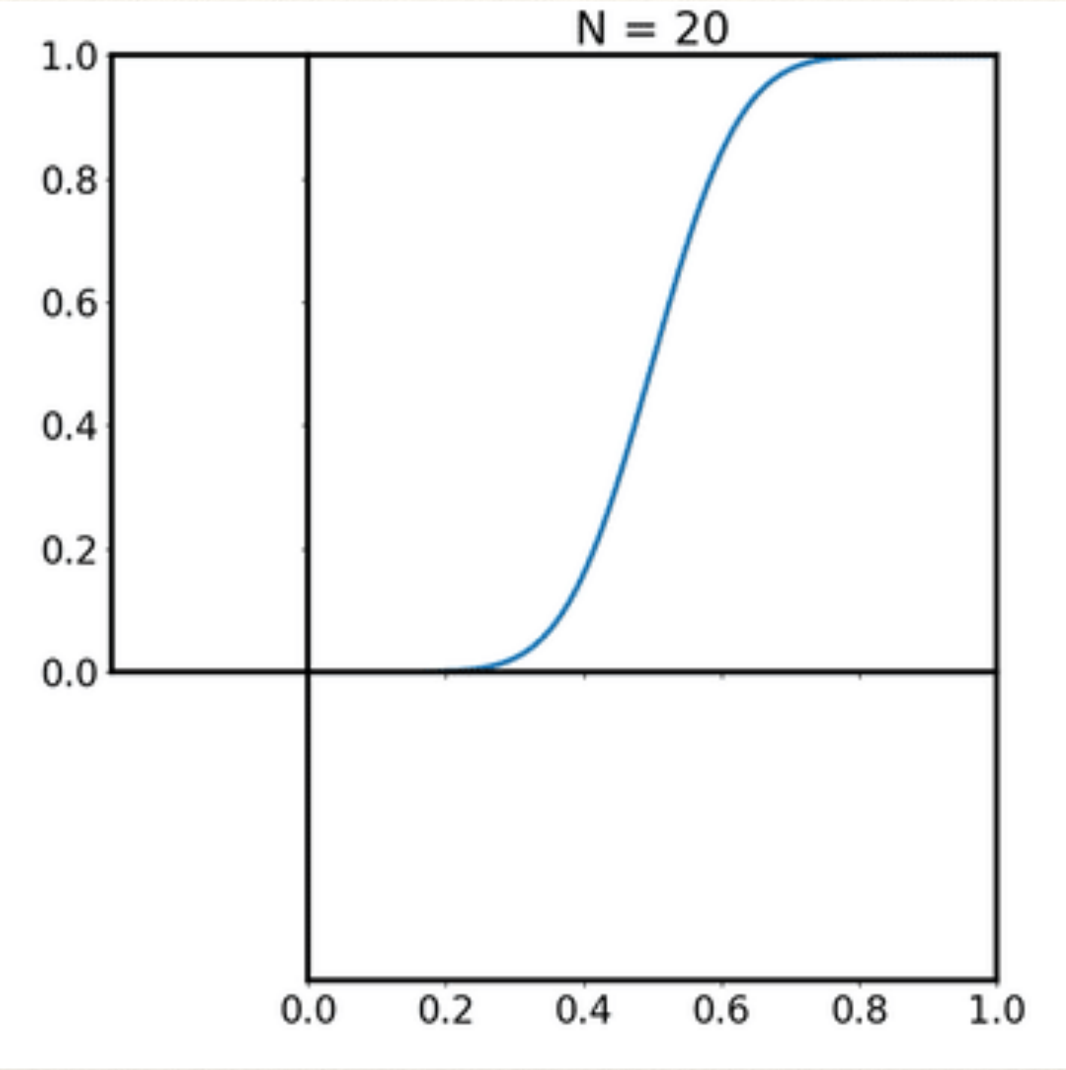
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- The Box-Muller method is a good, simple and often used approximation:

$$Z = (-2\ln(r_1))^{1/2}\cos(2\,\pi\, r_2)$$

  - Use 2 random numbers from a uniform distribution (r_1, r_2)
  - Z will follow a gaussian distribution with $\mu = 0$ and $\sigma = 1$
  - Use $x = \mu + Z\sigma$ to get a distribution with the required mean (μ) and width (σ)
- The rejection method also works!

Gaussian distribution



An animation of how inverse transform sampling generates random values

Exercise 2

- Estimate the value of **π** using the Rejection Method

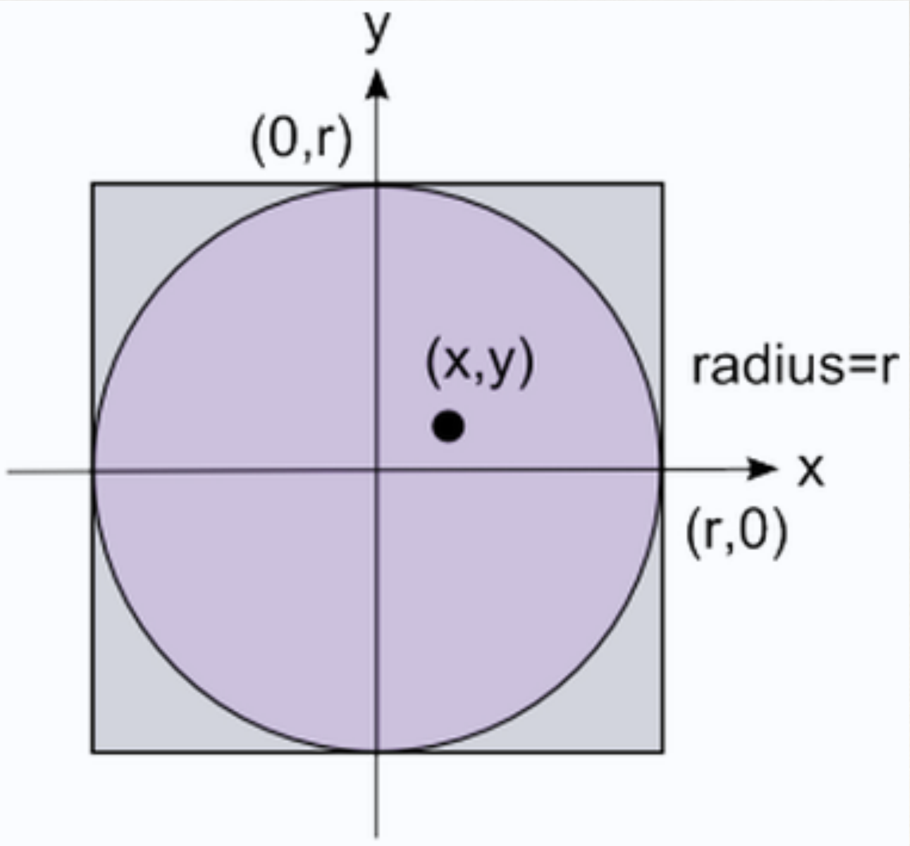
Exercise 2

- Estimate the value of **π** using the Rejection Method

- Tip: Consider a circle inside a square and use the ratio between the areas

$$\frac{A_{circle}}{A_{square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

$$\pi = 4 * \frac{A_{circle}}{A_{square}}$$

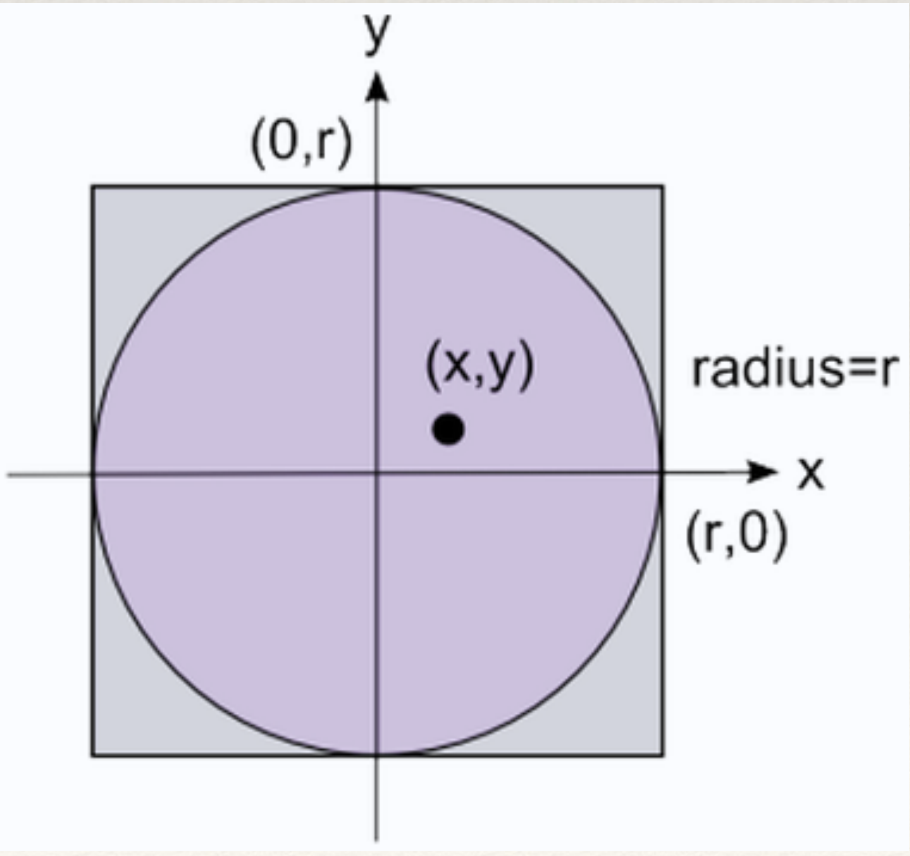


Exercise 2

- Estimate the value of **π** using the Rejection Method
  - Vary the number of samples (100, 1000, 10k)
  - Check what happens to the relative error to the "real" value of **π** as you increase the number of samples
- <u>Tip:</u> Consider a circle inside a square and use the ratio between the areas

$$\frac{A_{circle}}{A_{square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

$$\pi = 4 * \frac{A_{circle}}{A_{square}}$$



Make a copy of exercise 1 and start from there
If you get stuck, have a look at <u>exercise02.cc</u>

Exercise 3 - Homework

- Write a generator for the exponential distribution using the inverse transform method. Create plots of the distribution for at least two different values of λ

OR

- Write a generator for the Gaussian distribution using the Box-Muller method. Create plots with the default gaussian parameters for the method ($\mu = 0$ and $\boldsymbol{\sigma} = 1$) and at least a different set of (μ, $\boldsymbol{\sigma}$)

Preferably in C++ (it's easy to adapt example 1 for this), but you may use any language (or even just write the algorithm)