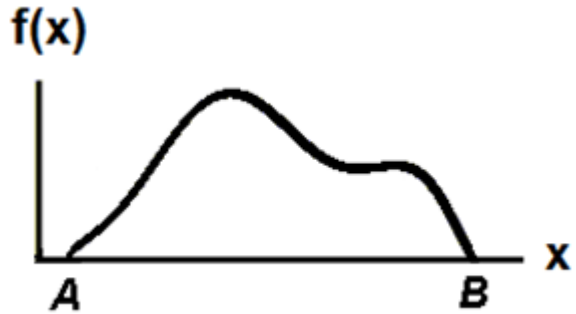
MIDDLE EAST TECHNICAL UNIVERSITY

Particle Data Analysis in High Energy Physics

Lecture 12

Monte Carlo Methods



Ahmet Bingül

METU, Physics

Apr 2026

Introduction

The deterministic systems are described by some mathematical rules.

But, some systems are not deterministic known as random or stochastic.

The Monte Carlo Method is a numerical technique for calculating probabilities and related quantities by using sequences of random numbers.

Pioneers

https://en.wikipedia.org

Enrico Fermi

Stanislaw Ulam

J. von Neumann

N. Metropolis

Some Applications

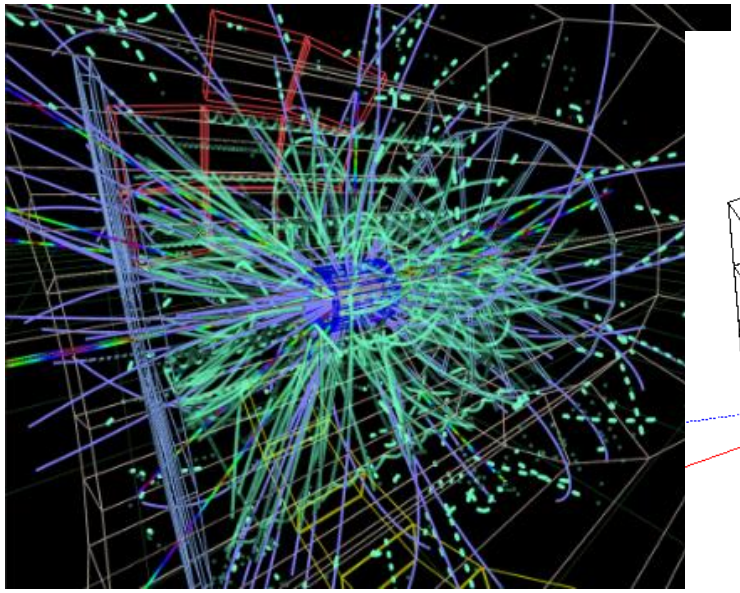
MC methods are often used to simulate experimental data.

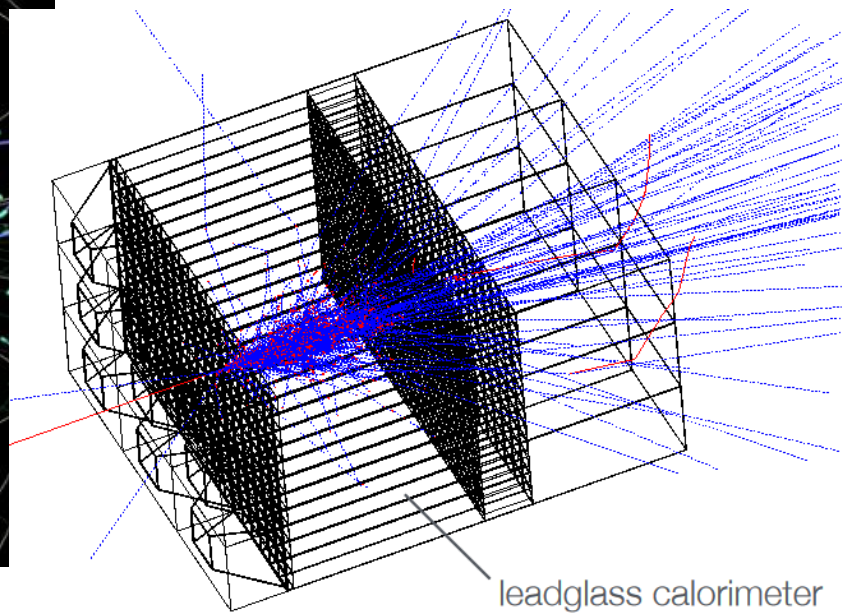
Applications Field	Example
Nuclear Physics	Simulation of Radioactive decay
Detector Physics	Simulation of Particle Interaction with Matter
Particle Physics	Event Generator
Engineering	Tolerance Analysis
Bioinformatics	Protein Folding
Statistics	Distribution Functions
Economy	Modelling Stock Exchange
Medicine	Treatment Planning in Radiation Therapy

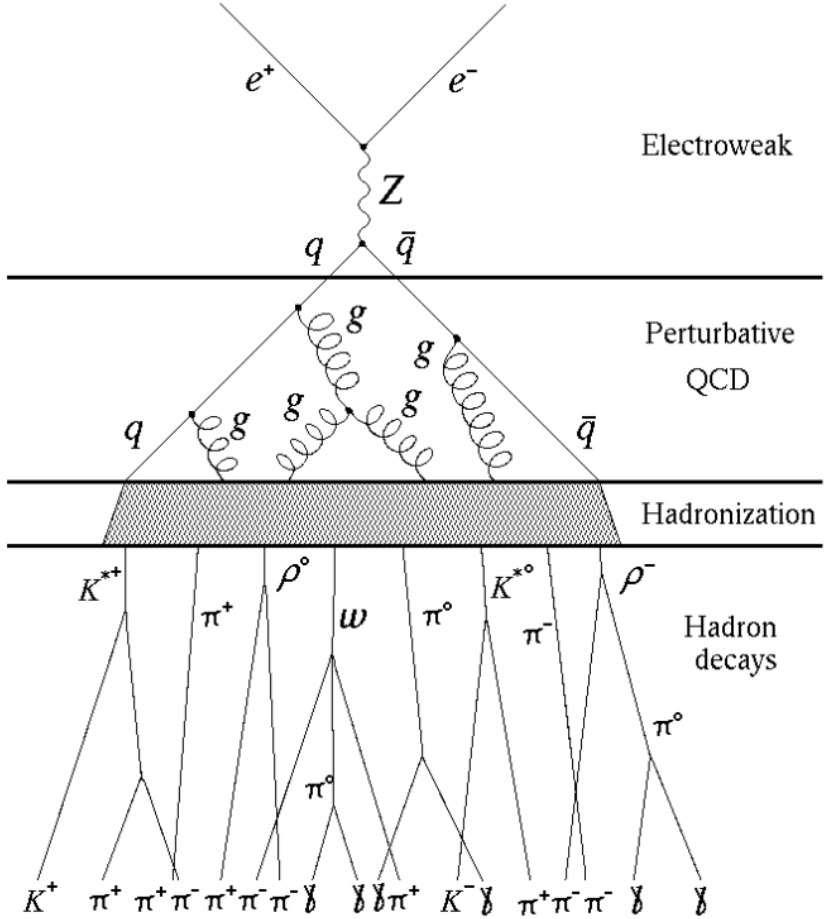
In Particle Physics

Simulation of experimental data is typically done in two stages:

1. Event generation (Pythia, Sherpa, Herwig, …)
2. Detector simulation (Geant4, Fluka, Phits, …)







Random Numbers

In principle, the best way to obtain a series of random numbers:

$$\{x_1, x_2, x_3, \ldots\ x_n\}$$

is to use some process in nature:

- Throw a coin or dice
- Lotto results of last Sunday.
- Number of particles from radioactive decay in every 1 min.
- Number of cosmic muons falling on 10cm x 20 cm area in 10 s.
- Brightness level of a star observed in atmoshpere in every 1 s.

These are not very efficient.

Therefore, **random number generator** algorithms have been developed to be used in computers.

Uniformly Distributed Random Numbers

Linear Conguential Method (1948)

- uses 32-bit integers
- have a period of at most $2^{31} \sim 10^9$.

The method is employed by an equation of the form:

$$x_{i+1} = (a\, x_i + b\,) \bmod m$$

where mod means modulo. Constants a, b and m are chosen carefully such that the sequence of numbers becomes chaotic and evenly distributed. The resulting values are therefore more correctly called **pseudorandom**.

RULES:

- First initial number (called seed), x_0, is selected.
- $m > x_0, a, b \geq 0$
- The range of values is 0 to m. The period generator is $m - 1$.
- Divide by m to convert to range 0.0 to 1.0

Example 1

if we select $a = b = x_0 = 7$ and $m = 10$ results in the squence:

$$x = \{7, 6, 9, 0, \quad 7, 6, 9, 0, \ ...\}$$

and dividing each number by 10 gives:

$$r = \{0.7, 0.6, 0.9, 0.0, \quad 0.7, 0.6, 0.9, 0.0, \quad ...\}$$

very poor!

```
# poor pseudo-random numbers
a, b, x0 = 7, 7, 7
m = 10
x = x0

for i in range(10):
   x = (a*x + b) % m
   r = x/m
   print(r)
```

However, some good generators have been proposed:

- IBM in 1960: $x_{i+1} = (\ 69069\ x_i\ ) \bmod 2^{31} - 1$
- Park and Miller: $x_{i+1} = (\ 16807\ x_i\ ) \bmod 2^{31} - 1$
- P.L. L'Ecuyer: $x_{i+1} = (\ 40692\ x_i\ ) \bmod 2147483399$

Some advanced algorithms:

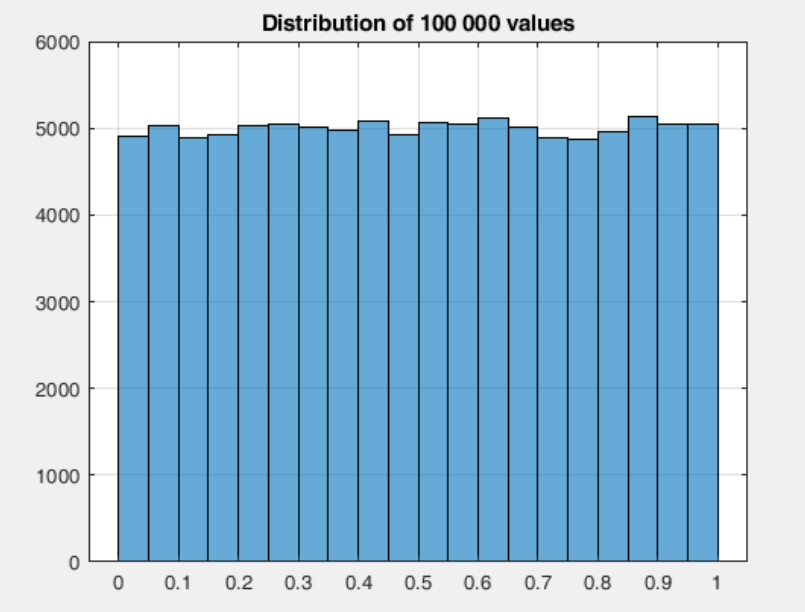
- RANLUX has period of 10^{143} (Root TRandom class uses this algorithm)
- Mersenne twister has period of 10^{6000}

Example 2: Implementation of RANDU

```
m = 2**31-1; # maximum value
x = 314;     # the seed

for i in range (10):
    x = 69069*x % m    # integer
    r = x / m;         # real
    print(r)
```

OUTPUT

```
0.01009910647296305
0.5351849810849806
0.69145855852005
0.3511784213367749
0.5423833097062927
0.8728181039322251
0.6736204948619103
0.2939596172859704
0.49680632469095587
0.9160400796290674
```

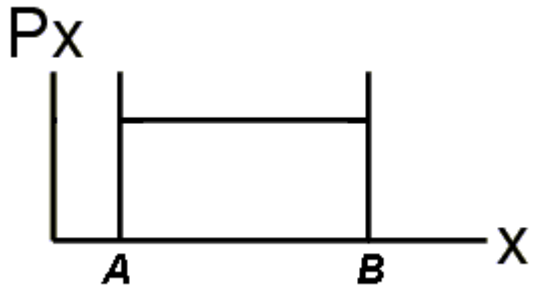


Uniformly Distributed Random Numbers

Assume that ***r*** is a uniform random real number in the range (0,1) and *A* and *B* are real numbers, *M* and *N* are integer numbers.
Then the value

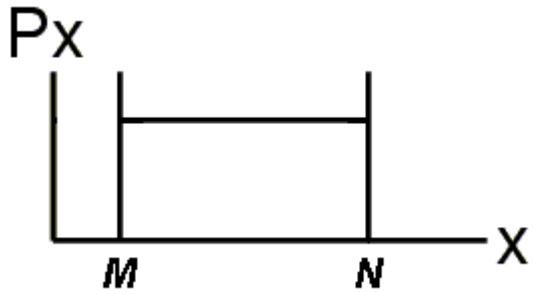
$$x = A + (B - A)r$$

will be *<u>random real number</u>* in the range $[A, B]$.



$$x = M + \text{int}(Nr)$$

will be *<u>random integer number</u>* in the range $[M, N]$.



Two examples:

Random decay angle in the range [0, 2π]

```
phi = 2*pi*r;
```

Random integer in the range [1, 6]

```
D = 1 + int(6*r);
```

Built-in Random Number Generators

C++

https://cplusplus.com/reference/random/

Obsolete C++

`int k = rand();` returns random integer in the range [0, RAND_MAX]

```
double r = double(k)/RAND_MAX;
```

Root

`TRandom rnd;` uses RANLUX algorithm

`double r = rnd.Uniform();` returns random real in the range (0,1).

Python

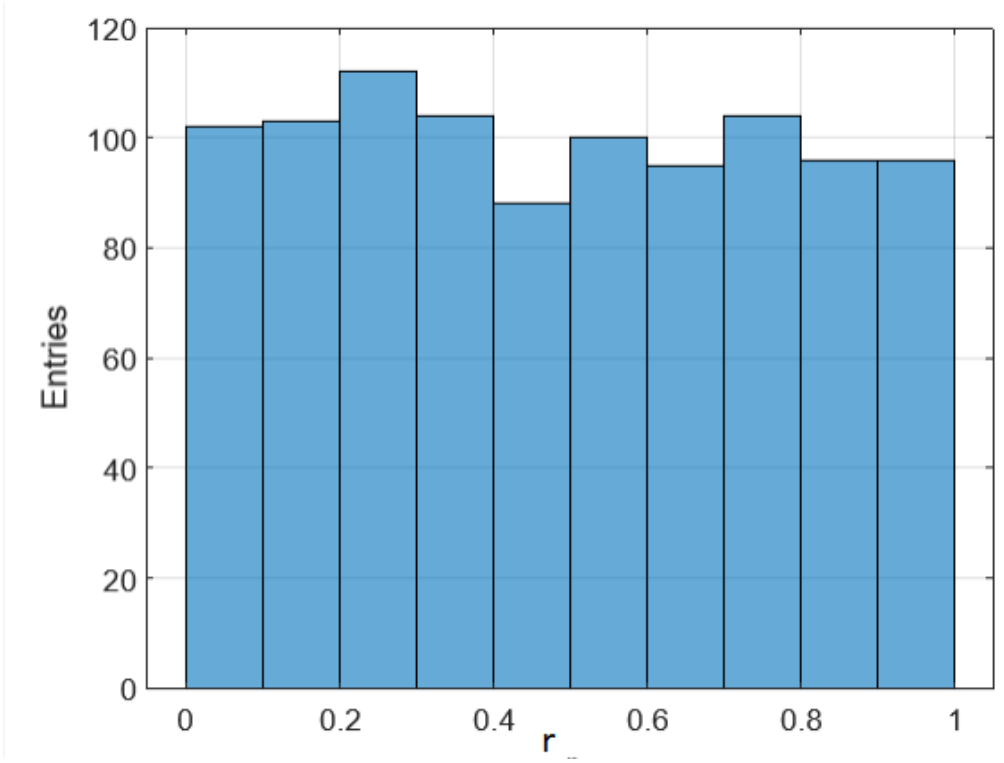
```
import random
```

`r = random.random()` returns random real in the range (0,1).

`r = random.randrange(6)` generates random integer from 1 to 6.

Example 3: Using Trandom (again)

```
import ROOT
seed = 1234
rnd = ROOT.TRandom(seed)

h = ROOT.TH1F("h", ";r;Entries",10,0,1)
for i in range(1000):
  r = rnd.Uniform()
  h.Fill(r)
h.Draw()
```



Box-Muller Algorithm for Standard Normal Distribution

1. Generate two uniformly distributed random numbers u_1 and u_2 in the range [0,1]

2. Set

$$\phi = 2\pi u_1, \qquad r = \sqrt{-2 \ln u_2}$$

3. Then

$$z_1 = r \cos\phi \quad \text{and} \quad z_2 = r \sin\phi$$

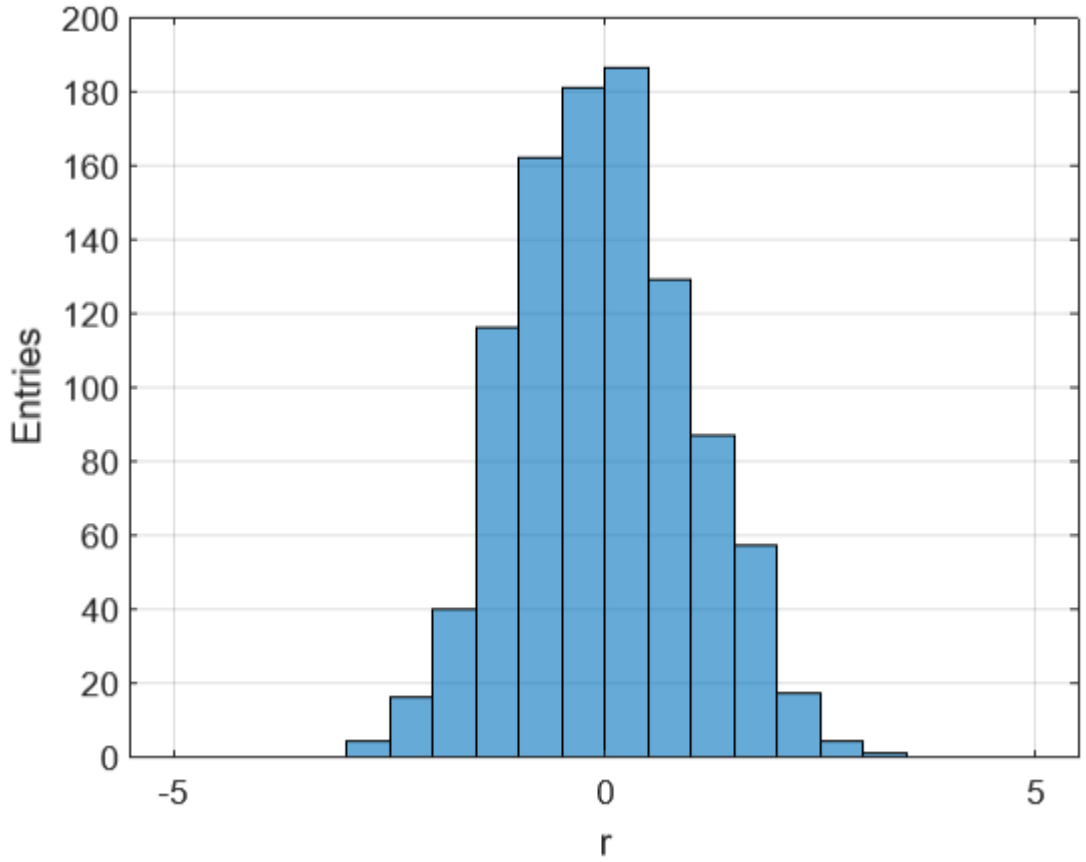
are two independent rv's following a standard normal distribution

In Root, to get a random number taken from standard normal distribution we use:

```
rnd = ROOT.TRandom()
rnd.Gaus(0,1)
```

Example 4: using Gaus() function

```
import ROOT
rnd = ROOT.TRandom(314)

h = ROOT.TH1F("h", ";r;Entries",20,-5,5)
for i in range(1000):
  r = rnd.Gaus(0,1)
  h.Fill(r)
h.Draw()
```



Example 5: ECAL Detector Resolution

In electromagnetic calorimeter, measured photon energy, E, is proportional to number of electrons, *n*, which are counted in the ECAL shower, because the number of charges is related with energy deposition in any space. Thus: $E \propto n$.
But, this counting is a statistical process carrying a statistical uncertainty $\sigma_n = \sqrt{n}$. Then,

$$\frac{\sigma_E}{E} \propto \frac{\sigma_n}{n} = \frac{\sqrt{n}}{n} \rightarrow \frac{\sigma_E}{E} = \frac{R}{\sqrt{E}}$$

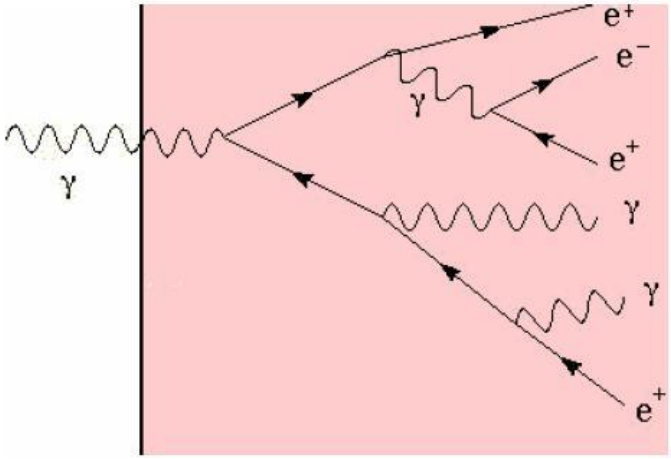
Or in general

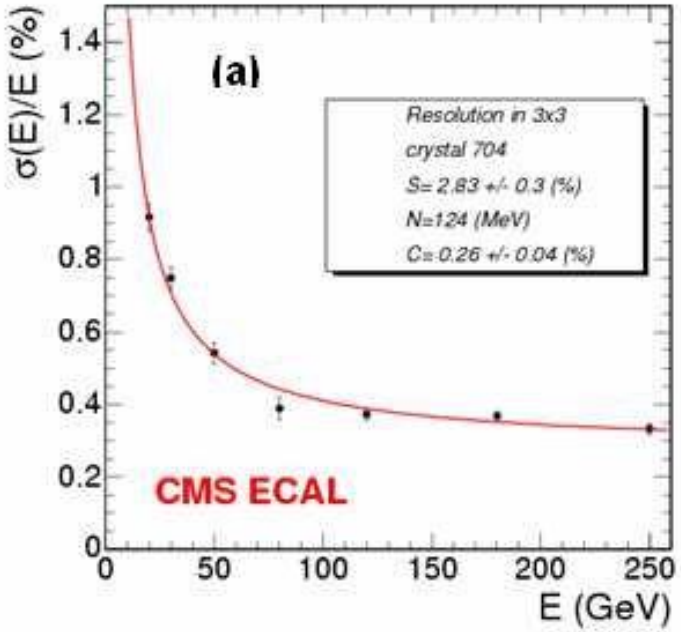
$$\frac{\sigma_E}{E} = \frac{R}{\sqrt{E}} + k$$

R = stocastic term

k = constant term

for ATLAS R = 0.10 GeV$^{1/2}$ and for CMS R = 0.05 GeV$^{1/2}$



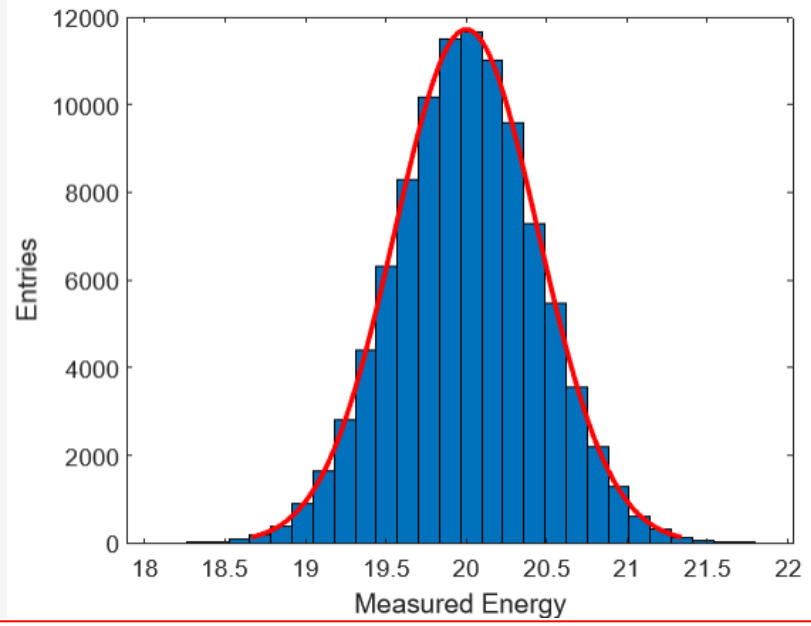


We can model ECAL by smearing the photon energies (E is in GeV).

$$\sigma_E = 0.1\sqrt{E}$$

For example, assume that E = 20 GeV. Then, the following code block can roughly simulate the response of ATLAS ECAL for n = 100 000 photons.

```
import ROOT
rnd = ROOT.TRandom()
h = ROOT.TH1F("h", ";Measured Energy;Entries",10,18,22)
E = 20
sigmaE = 0.1*sqrt(E)
for i in range(100000):
  r = rnd.Gaus(E,sigmaE)
  h.Fill(r)
h.Fit("gaus")
h.Draw()
```

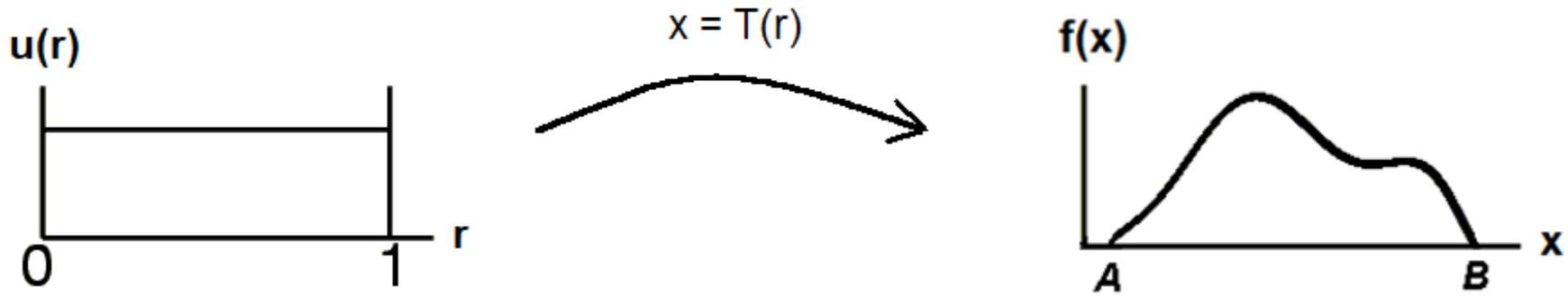


Random Distributions

In simulations of random processes, we often require a non-uniform distribution of random numbers. Two standard methods are:

- The Transformation Method
- The Rejection Method

The aim of both these methods is to convert a uniform distribution of random numbers of the form $u(r)$ into a non-uniform distribution $f(x)$. In particle physics the values of x can then be treated as simulated measurements.



Transformation Method (Continues RV)

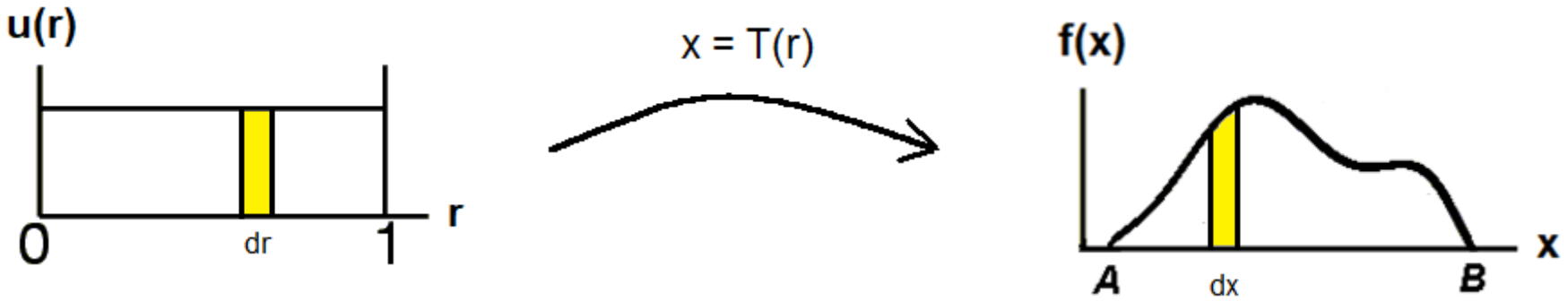
Let $u(r)$ be the uniform distribution function in the range (0,1).

Consider a distribution $f(x)$ from which we want to draw random numbers, x.

Conservation of random numbers!

$$u(r)dr = f(x)dx$$

The aim is to find a transformation function $x = T(r)$ such that the distribution of random variable x is $f(x)$.



Since $u(r) = 1$, we need to solve D.E.

$$u(r)dr = f(x)dx$$
$$dr = f(x)dx$$
$$r = \int f(x)dx$$

Right hand side is CDF:

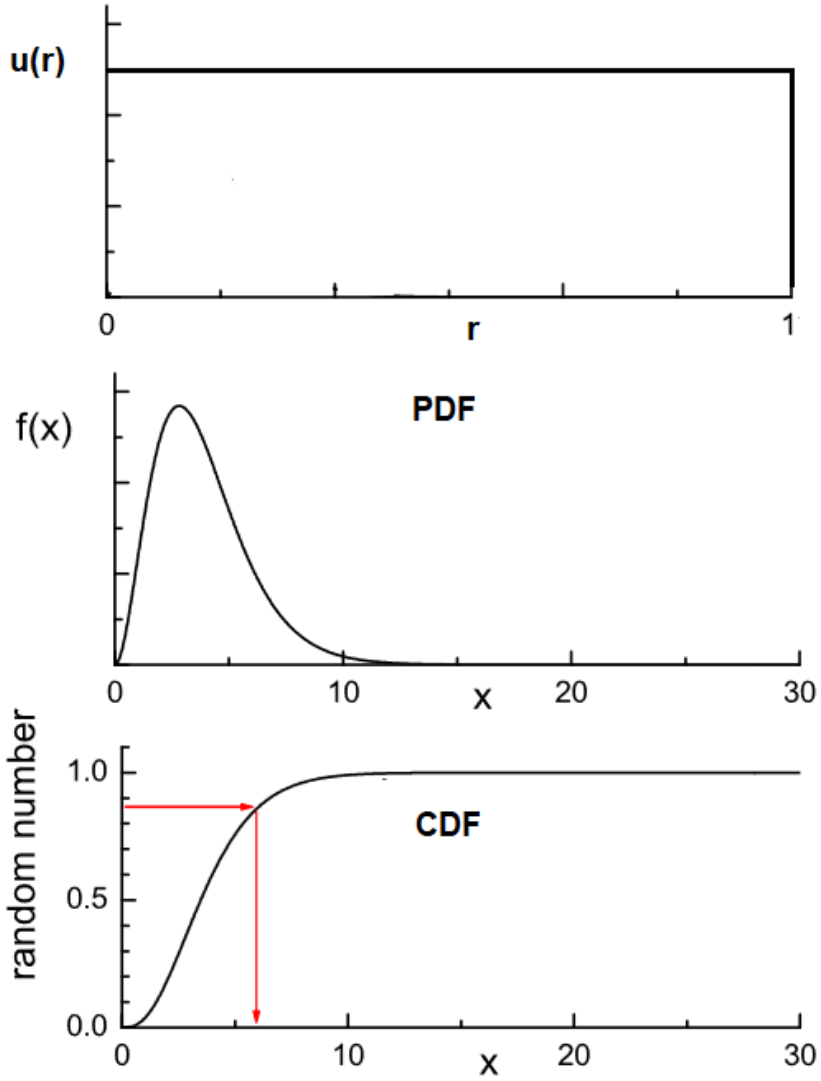
$$r = \int_0^r u(r)dr = \int_{-\infty}^{x} f(x)dx = F(x)$$

$$F(x) = r$$

We want to get x from inverse CDF.

$$x = F^{-1}(r) = T(r)$$

$T(r)$ is known as the **Transformation Function.**



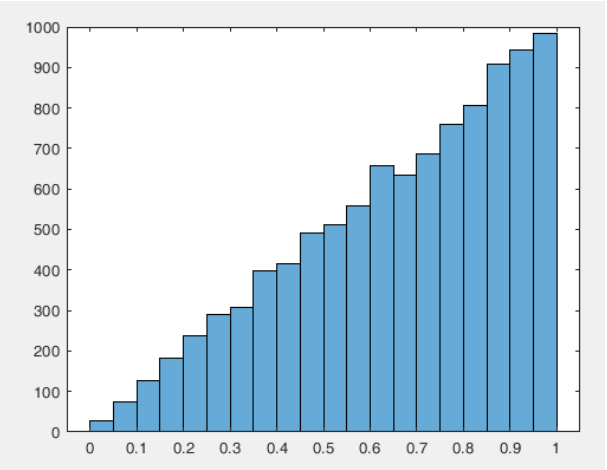
Example 6

We want a random distribution function

$$f(x) = 2x \quad [0 < x < 1]$$

Then we can find the transformation function $T(r)$ as follows:

$$r = \int 2x dx = x^2 \quad => \quad x = \sqrt{r} = T(r)$$



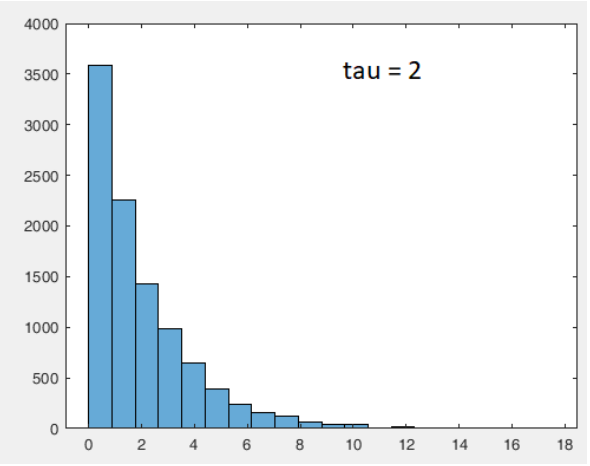
We want a random distibution function

$$f(t) = \frac{1}{\tau} e^{-t/\tau} \quad [0 < t < \infty]$$

Transformation function:

$$r = \int f(t) dt = 1 - e^{-t/\tau} \quad =>$$

$$t = -\tau \ln(1 - r) = -\tau \ln(r) = T(r)$$



- In Root, we use function **`double Exp(double tau)`**

```
import ROOT
rnd = new TRandom()
rnd.Exp(2)        # returns single value from this distribution whose mean is 2
```

Example 7: Uniform point on a sphere

$$\frac{\mathrm{d}p}{\mathrm{d}\Omega} = \frac{\mathrm{d}p}{\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi} = \text{const} \equiv k \qquad \frac{\mathrm{d}p}{\mathrm{d}\theta\,\mathrm{d}\phi} = k\sin\theta \equiv f(\phi)g(\theta)$$

Distributions for θ and ϕ:

$$f(\phi) \equiv \frac{\mathrm{d}p}{\mathrm{d}\phi} = \text{const} = \frac{1}{2\pi}, \qquad 0 \le \phi \le 2\pi$$

$$g(\theta) \equiv \frac{\mathrm{d}p}{\mathrm{d}\theta} = \frac{1}{2}\sin\theta, \qquad 0 \le \theta \le \pi$$

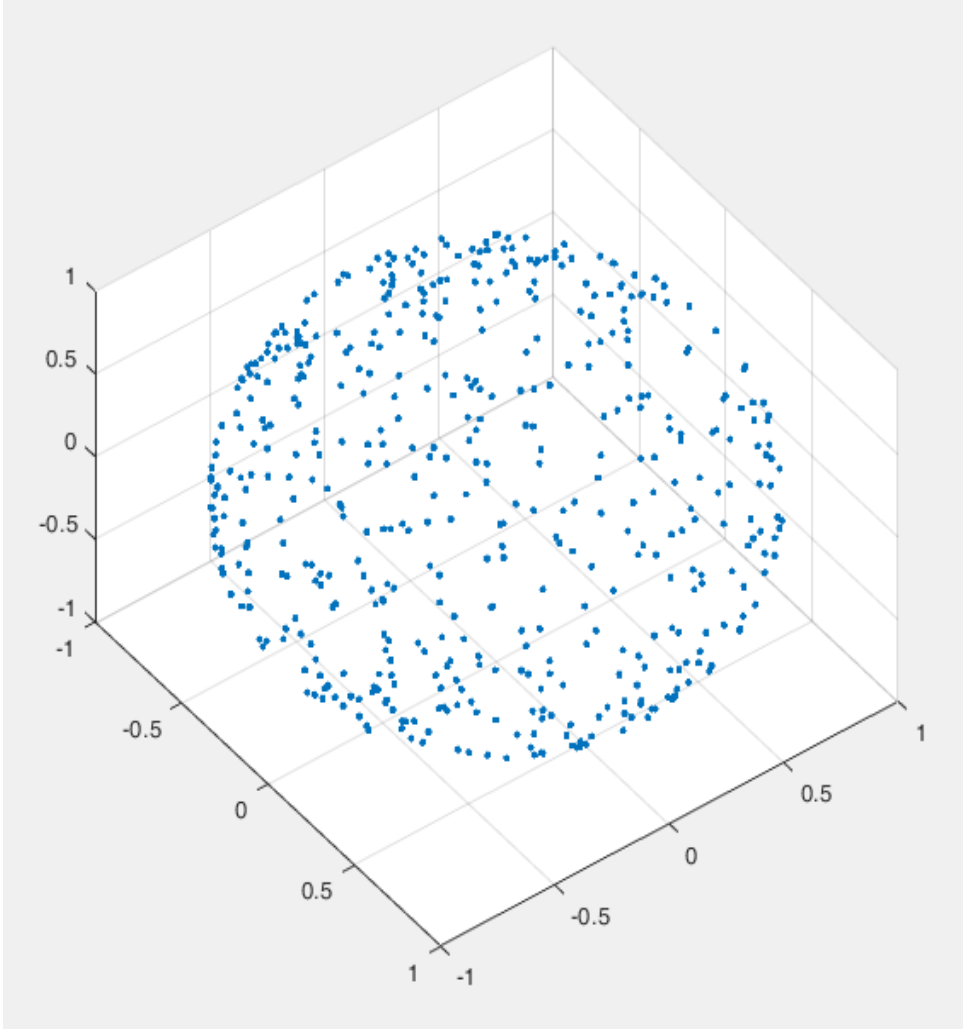
Calculating the inverse of the cumulative distribution we obtain:

$$\phi = 2\pi r_1$$

$$\theta = \arccos(1 - 2r_2) \qquad [\text{as } G(\theta) = \frac{1}{2}(1 - \cos\theta)]$$

Upshot: ϕ and cos θ need to be distributed uniformly

```python
import ROOT
import numpy as np

n = 500
R = 1.0

# Generate random spherical coordinates
phi   = 2 * np.pi * np.random.rand(n)
theta = np.arccos(1 - 2 * np.random.rand(n))

# Convert to Cartesian
x = R * np.sin(theta) * np.cos(phi)
y = R * np.sin(theta) * np.sin(phi)
z = R * np.cos(theta)

# Create a TGraph2D (PyROOT's equivalent of plot3)
graph = ROOT.TGraph2D(n)
for i in range(n):
    graph.SetPoint(i, x[i], y[i], z[i])

# Draw
c = ROOT.TCanvas("c", "Sphere", 800, 800)
graph.SetMarkerStyle(6)   # small dot, like '.'
graph.SetMarkerColor(ROOT.kBlue)
graph.Draw("P0")          # P = markers only, 0 = no error bars
```



Transformation Method (Discrete RV)

Suppose **X** can take on *n* distinct values **X** = $\{x_1 , x_2 , x_3 , \ldots x_n\}$ with

PDF: f(x) = $\{f_1 , f_2 , f_3 , \ldots f_n\}$ and

CDF: F(x) = $\{f_1 , f_1 + f_2 , \ldots , f_1 + f_2 + f_3 + \ldots + f_n\}$

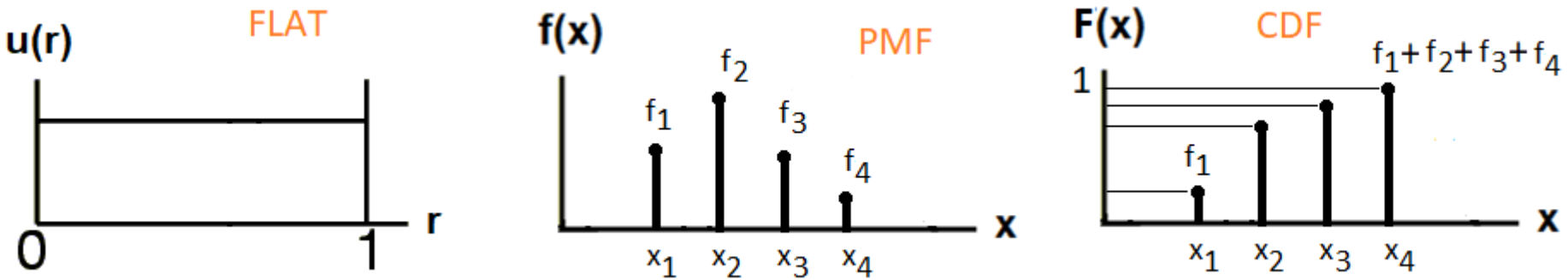
Then to generate a sample value of **X**

1. Generate uniform random number r in the range [0, 1].

2. for j = 1 to n

Set **X** = x_j if $F_{j-1} < r \leq F_j$

end

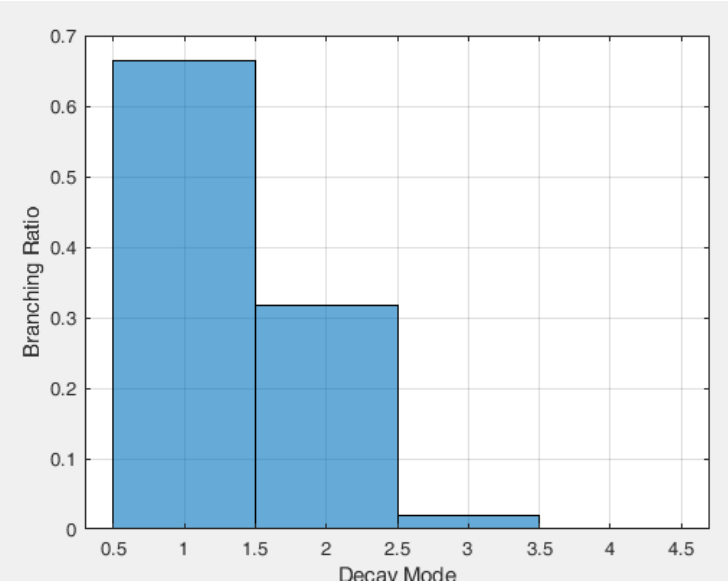


Example 8: Branching Ratio

D*± decays into 3 different channels. Select a decay mode randomly.

Mode		Fraction (Γ_i / Γ)
Γ_1	$D^0\pi^+$	$(67.7 \pm 0.5)\%$
Γ_2	$D^+\pi^0$	$(30.7 \pm 0.5)\%$
Γ_3	$D^+\gamma$	$(1.6 \pm 0.4)\%$

```
import ROOT
import numpy as np
x = [1, 2, 3]                    # Decay modes
f = [0.677, 0.307, 0.016]        # Branching Ratios (BR)
F = [0.677, 0.984, 1.000]        # Cumulative Distribution Function (CDF)
N = 1000                         # Total number of simulated decays

h1 = ROOT.TH1D("h1", "Decay Simulation;Channel;Branching Ratio", 3, 0.5, 3.5)
rng = ROOT.TRandom3(0)  # ROOT's random number generator (0 uses a seed based on time)

for i in range(N):
    r = rng.Uniform()    # Generate a uniform random number in [0, 1]
    for j in range(len(F)):
        if j == 0:
            if r <= F[j]:
                h1.Fill(x[j])
                break
        else:
            if r > F[j-1] and r <= F[j]:
                h1.Fill(x[j])
                break
# Normalize the histogram so the sum of bin contents = 1 (PDF style)
scale = 1.0 / h1.Integral()
h1.Scale(scale)
h1.Draw("HIST")  # Draw as a histogram (bar chart style)
```



Rejection Method

The Transformation Method is useful when the function $T(r)$ can easly be evaluated. However, there are cases when desired distribution may not be known in analytic form.

Two examples are as follows:

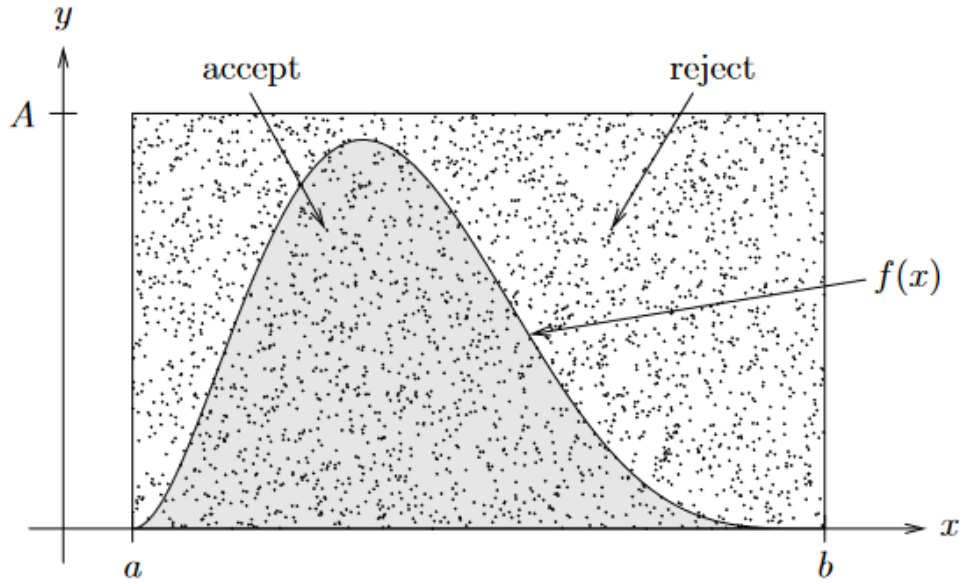
Gaussian fuction : $f(x) = e^{-x^2}$

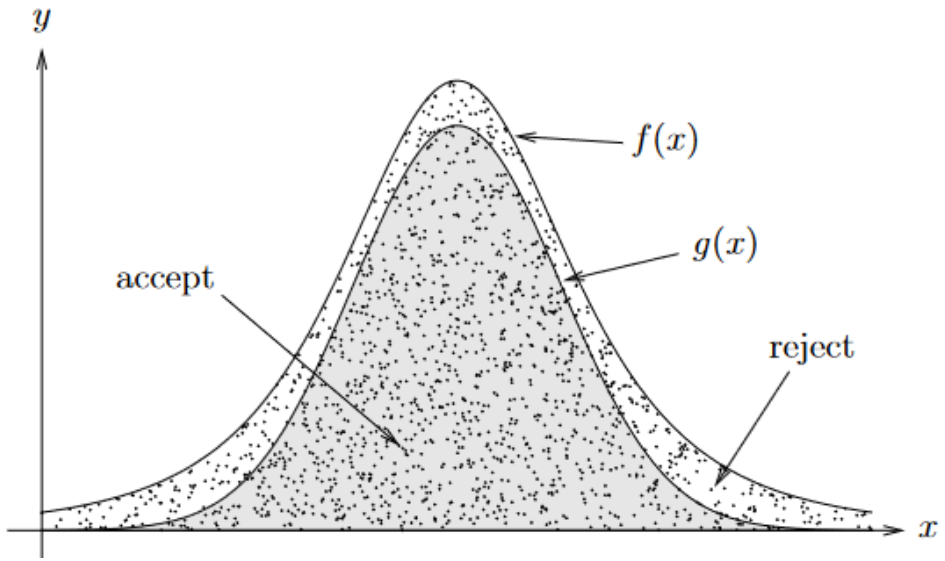
Quadratic function : $f(x) = 1 + x^2$

Such problems can be handled with algorithm known as **Rejection Method.**

It has the advantage of being able to create a distribution for *any function*.

- Algorithm
  - Generate random number *x* uniformly between *a* and *b*
  - Generate second random *y* number uniformly between 0 and *A*
  - Accept *x* if *y* < f(*x*)
  - Repeat many times
- The efficiency of this algorithm can be quite small
- Improvement possible by choosing a majorant, i.e., a function which encloses *g*(x) and whose integral is known ("importance sampling")





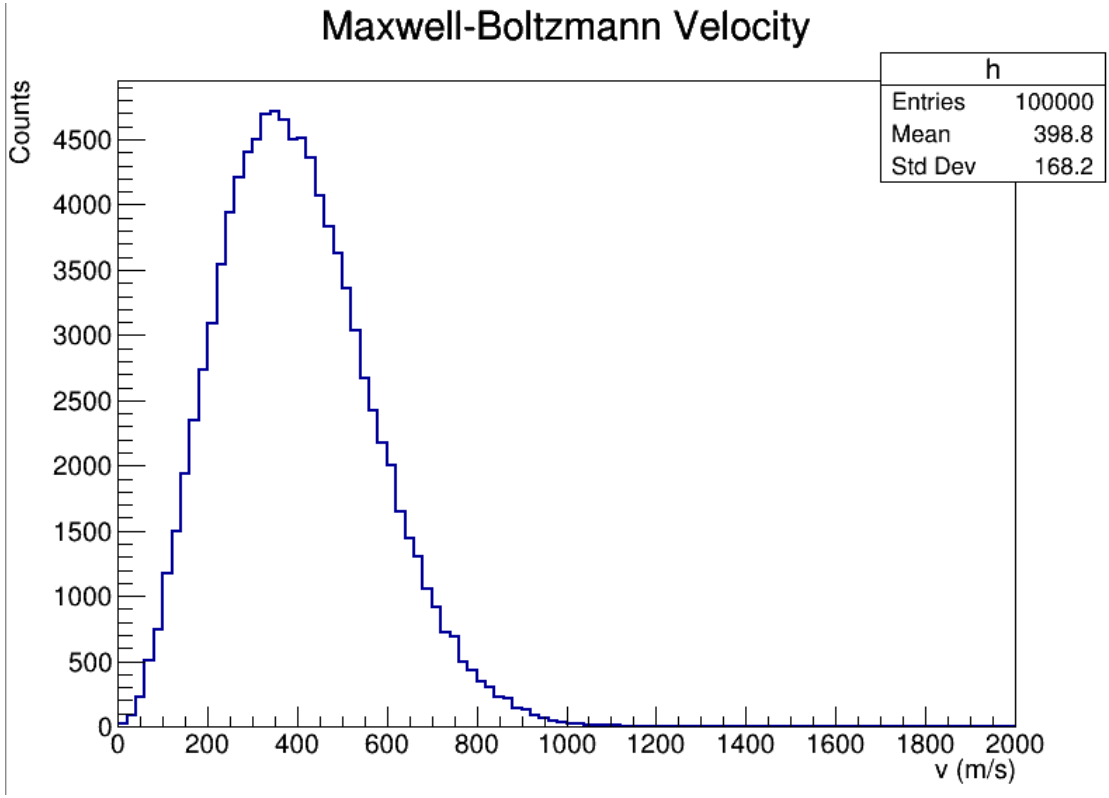
Example 9a: Maxwell Distribution in C++

```
double Maxwell_Boltzman(double m, double T){
  //-------------------------------------------------------------
  // Returns, for an atom of mass m (kg) and temperature T (K),
  // a velocity in m/s which is randomly selected from
  // a Maxwell-Boltzman Distribution function using Rejection Method.
  //-------------------------------------------------------------
  double const kB = 1.38e-23;
  double kT   = kB*T;
  double C    = M_PI*sqrt(2.0) * pow(m/(M_PI*kT),1.5);
  double vp   = sqrt(2.0*kT/m);
  double vmin = 0.0;
  double vmax = 10*vp;
  double fmax = C * vp*vp * exp(-1.0), Ptest, fmb;

   // Rejection algorithm
   while(1)
   {
     double r = rnd.Uniform();
     double v = rnd.Uniform();

     Ptest = fmax*r;
     v     = vmin + (vmax-vmin)*v;
     fmb   = C * v*v * exp(-0.5*m*v*v/kT);

     if (fmb > Ptest) return v;
   }
 }
```

$$f(v) = \left[\frac{m}{2\pi kT}\right]^{\frac{3}{2}} 4\pi v^2 \exp\left(-\frac{mv^2}{2kT}\right)$$

Example 9b: Maxwell Distribution in Python

```
import ROOT
import math
rnd = ROOT.TRandom3(0)

# Function
def Maxwell_Boltzman(m, T):
     kB = 1.38e-23
     kT = kB * T
     C = math.pi * math.sqrt(2.0) * (m / (math.pi * kT))**1.5
     vp = math.sqrt(2.0 * kT / m)
     vmin = 0.0
     vmax = 10.0 * vp
     fmax = C * vp * vp * math.exp(-1.0)

     while True:
             r = rnd.Uniform()
             v = rnd.Uniform()
             Ptest = fmax * r
             v = vmin + (vmax - vmin) * v
             fmb = C * v * v * math.exp(-0.5 * m * v * v / kT)
             if fmb > Ptest:
                 return v
#------------------------------------------------------------
# Main program
if __name__ == "__main__":

 # Physical parameters
 m = 6.63e-26 # mass (kg) ~ Argon atom
 T = 300.0 # room temperature (K)
 h = ROOT.TH1F("h", "Maxwell-Boltzmann Velocity;v (m/s);Counts", 100, 0, 2000)

 for i in range(100000):
     v = Maxwell_Boltzman(m, T)
     h.Fill(v)

 h.Draw()
 input("Press Enter to exit...")
```



Monte Carlo Integration

Naïve Monte Carlo integration:

$$\underbrace{\int_a^b f(x)\,\mathrm{d}x}_{=:\,I} = (b-a)\int_a^b f(x)\,u(x)\,\mathrm{d}x = (b-a)\langle f(x)\rangle \approx \underbrace{(b-a)\cdot\frac{1}{n}\sum_{i=1}^{n} f(x_i)}_{=:\,\hat{I}}$$

$u(x)$: uniform distribution in $[a, b]$

x_i: uniformly distributed random numbers

Typical Error (Standard deviation) $\sigma[I] = \dfrac{b-a}{\sqrt{n}}\sigma[f]$

This approach is not efficient in 1D integrals.

But, very good at higher dimensions.

Example 10:

Evaluate the following integrals via MC integration method.

(a) $\int_0^{\pi} \sin(x)\, dx = 2.0$

(b) $\int_0^{2\pi} \int_0^{\pi} \sin(\theta) d\theta\, d\phi = 4\pi \approx 12.5664$

```
# solution of part (a)
n = 10000
s = 0
pi = 3.1415926
for i in range(n):
    x = rnd.Uniform(0,pi)
    f = sin(x)
    s = s + f

integ = (pi-0) * s/n
print(integ)
```

```
# solution of part (b)
n = 10000
s = 0
pi = 3.1415926
for i in range(n):
    theta = rnd.Uniform(0,pi)
    phi   = rnd.Uniform(0,2*pi)
    f = sin(theta)
    s = s + f

integ = (2*pi-0)*(pi-0) * s/n
print(integ)
```

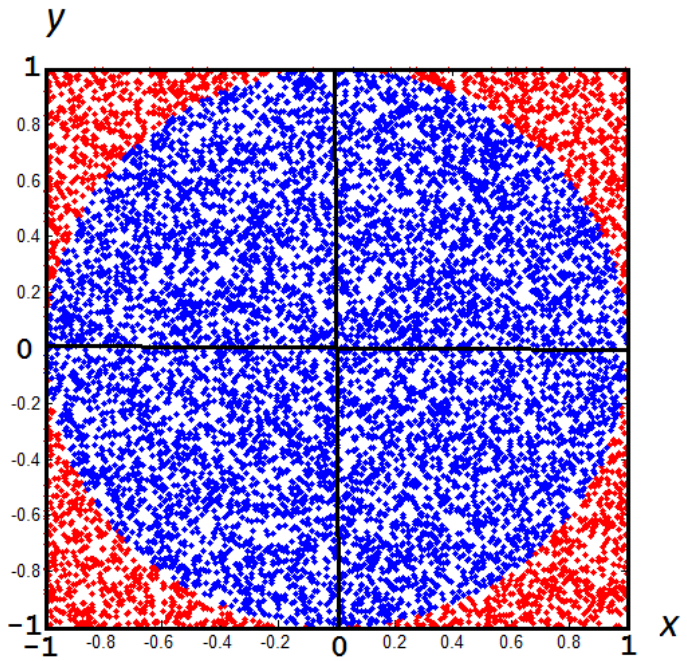
Example 11: Estimating π using MC

Populate a square with *n* randomly-placed points [the blue and red points]

$-1.0 \le x < +1.0$
$-1.0 \le y < +1.0$

Count the number of points *m* that lie inside a circle of unit Radius [the blue points] then $m / n \approx \pi / 4 \Rightarrow \pi \approx 4\, m / n$

blue area = $\pi\, r^2 = \pi\, 1^2 = \pi$ units
red area = $2 \times 2 = 4$ units



Convergence:

n	
10^3	3.168
10^4	3.1692
10^5	3.14836
10^6	3.140310
10^7	3.1419192
π	3.1415927

Case Study: Two Body Decay Simulation

This is a classic and very instructive Monte Carlo application.

Consider the decay of neutral pions to photons: $\pi^0 \to \gamma\gamma$.
Assume that

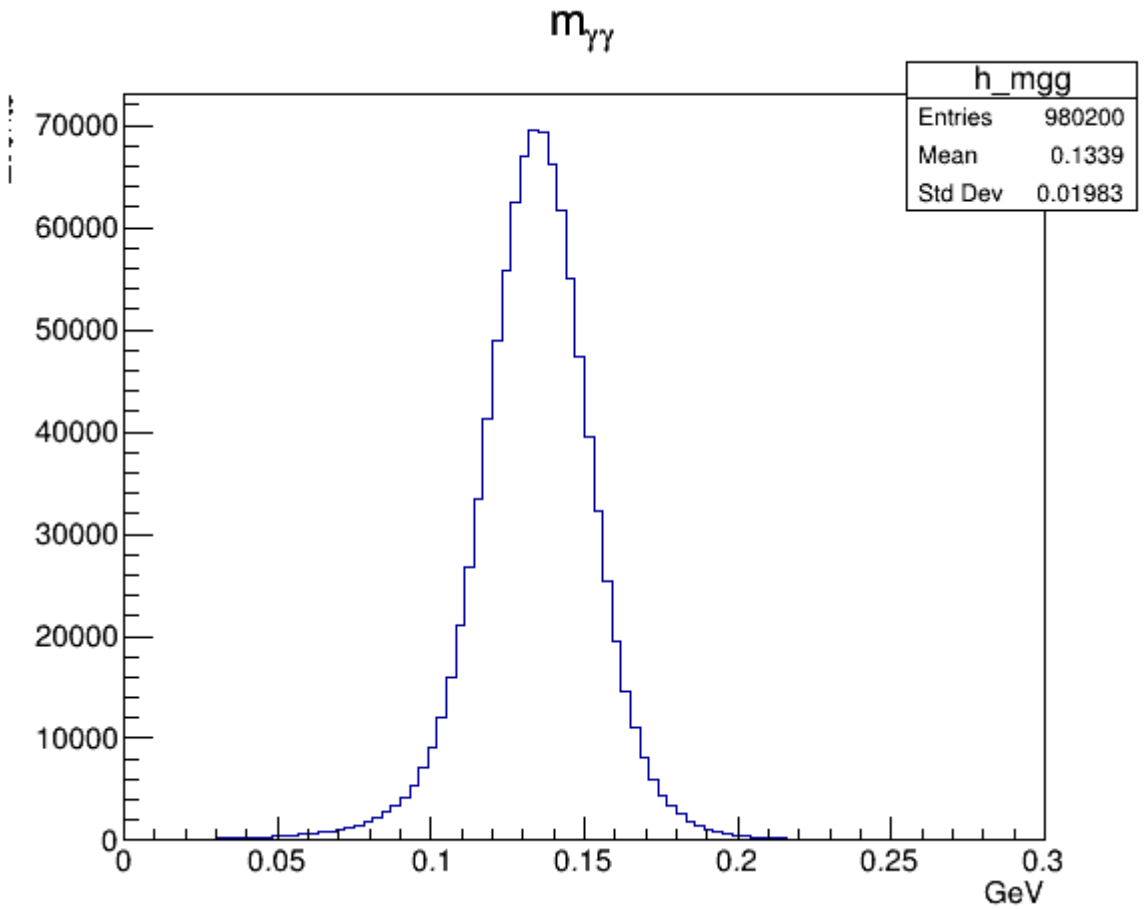
- Pions are moving along z-axis and have the momenta of $\mathbf{p}_{\pi^0} = (0, 0, 1.0\ \mathrm{GeV})$.
- Photons are measured in an ECAL whose enegy resolution is modelled by the formula $\sigma_E/E = 0.1/\sqrt{E}$ where E is the enery of the photon in GeV.
- ECAL acceptance region is $|\eta| \leq 5$ (pseudorapidity $\eta = -\ln[\tan(\theta/2)]$).

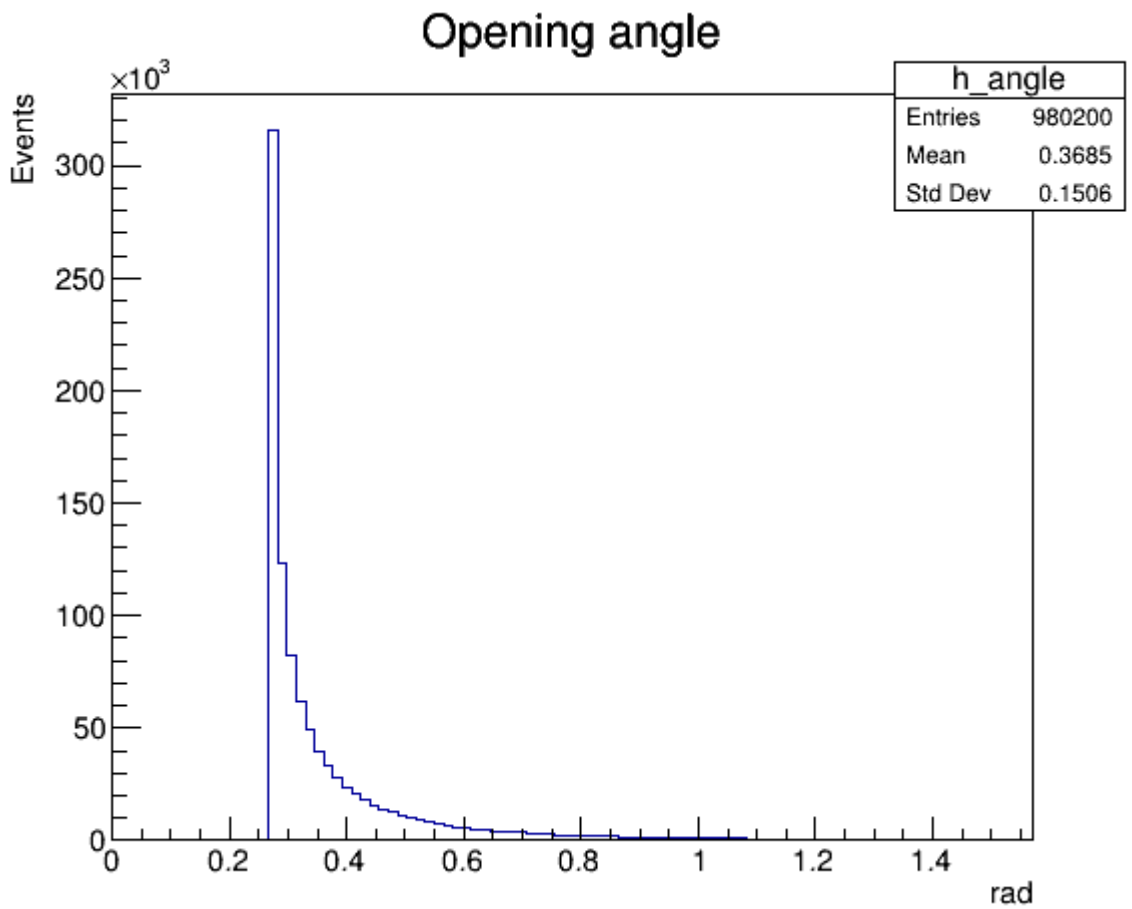
Write a program to do simulation of $n = 10^6$ pion decays and obtain the distributions of the two-photon invariant mass and opening angle between photons.

Key Steps:

1. Generate π^0 with fixed lab momentum.
2. Decay isotropically in the rest frame $\pi^0 \to \gamma\gamma$ back-to-back.
3. Boost photons to lab frame.
4. Apply ECAL smearing and acceptance ($|\eta| \leq 5$).
5. Fill invariant mass and opening angle histograms.

AI Result:


$m_{\gamma\gamma}$
h_mgg
Entries 980200
Mean 0.1339
Std Dev 0.01983
70000
60000
50000
40000
30000
20000
10000
0
0
0.05
0.1
0.15
0.2
0.25
0.3
GeV


Opening angle
h_angle
Entries 980200
Mean 0.3685
Std Dev 0.1506
Events
×10³
300
250
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
rad

Some Applications

1. Modelling Cherenkov Radiation Photons
   https://github.com/abingul/ceren

2. Simulation of Two-Body Decay
   http://www1.gantep.edu.tr/~bingul/simulation/twoBody

3. Simulation of Stern-Gerlach Experiment
   http://www1.gantep.edu.tr/~bingul/seminar/spin

4. Simulation of Fussion Chain Reaction
   http://www1.gantep.edu.tr/~bingul/simulation/fission/

5. Tolerance Analysis of a Lens
   See Section 9.3 at:
   http://www1.gantep.edu.tr/~bingul/ep208/latex/ep208_LectureNotes.pdf

6. Predicting Potential Energy Function in Schrödinger Equation via MC *coming soon* …

References

http://www1.gantep.edu.tr/~bingul/seminar/monte-carlo/page1.html

http://www.columbia.edu/~mh2078/MonteCarlo/Generating_RVars_MasterSlides.pdf

https://cas.web.cern.ch/sites/default/files/lectures/thessaloniki-2018/cas-montecarlov6.pdf

https://www.physi.uni-heidelberg.de/~reygers/lectures/2020/smipp/

Exercises

1. Write a computer program to simulate tossing a coin and output the probability of getting heads.
2. Write a computer program to simulate tossing four coins at a time and output the probability of getting at least two heads.
3. Write a computer program to simulate tossing a die and output the probability of getting four.

4. Consider you have a square of side 1 unit. Three points are selected randomly inside the square to form a triangle. Write a program to obtain the probability of getting a wide-angle triangle.

Use 30 million random numbers totally.

5. From a uniform distribution, u(r), we want to obtain a non-uniform distribution of random numbers according to the distribution function f(x) = 2x.

a) Determine the transformation function x = T(r).

b) Plot the distribution u(r) and f(x)

6. For each the following target distribution functions, determine the transformation functions $x = T(r)$. Plot the distributions.

(a) $f(x) = 1 - x$

(b) $f(x) = 2x^2$

(c) $f(x) = \exp(-x)$

7. Use Rejection Method to obtain distribution of $n = 10^6$ random variables taken from the following functions:

(a) $f(x) = 1 + x^2$ $(-1 < x < 1)$

(b) $f(x) = \sin(x)/x$ $(-10 < x < 10)$

(c) $f(x) = x*\exp(-x)$ $(0 < x < 10)$

8. (a) Write a program to evaluate the following integral using MC integration method.
Generate *n* random numbers in the solution where *n* is input

$$\int_0^{\pi}\int_0^{2\pi} 9\sin(\theta)d\phi d\theta$$

(b) Evaluate the true value of the integral and plot n vs MC integration results to compare outcomes, for $n = 10^1, 10^2, 10^3, \ldots, 10^8$.

9. Find the volume of the intersection of a cone and a torus

- *Hard to solve analytically*
- *Easy to solve by scattering points homogeneously inside a cuboid containing the intersect.*

10. Consider the reaction: $\nu_e + p \rightarrow n + e^+$ where proton is at rest.

a) Find the treshold energy of the reaction.

b) Write a program to do the simulation of the reaction.

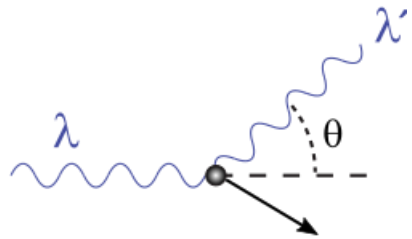
Input: neutrino energy

Output: Histograms of energies of products.

11. Perform the simulation of Compton Scattering process by smearing the energy of scattered photon by using the formula: $\sigma_E = R\sqrt{E}$ where E is a positve parameter and E is the energy of photon in keV. Investigate the effect of parameter R on the distribution of scattering angle θ.

See also:

http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/compdat.html



12. Develop a Monte Carlo simulation of multiple Coulomb scattering for a charged particle beam traversing a material slab and produce a histogram of the scattered particle angular distribution.