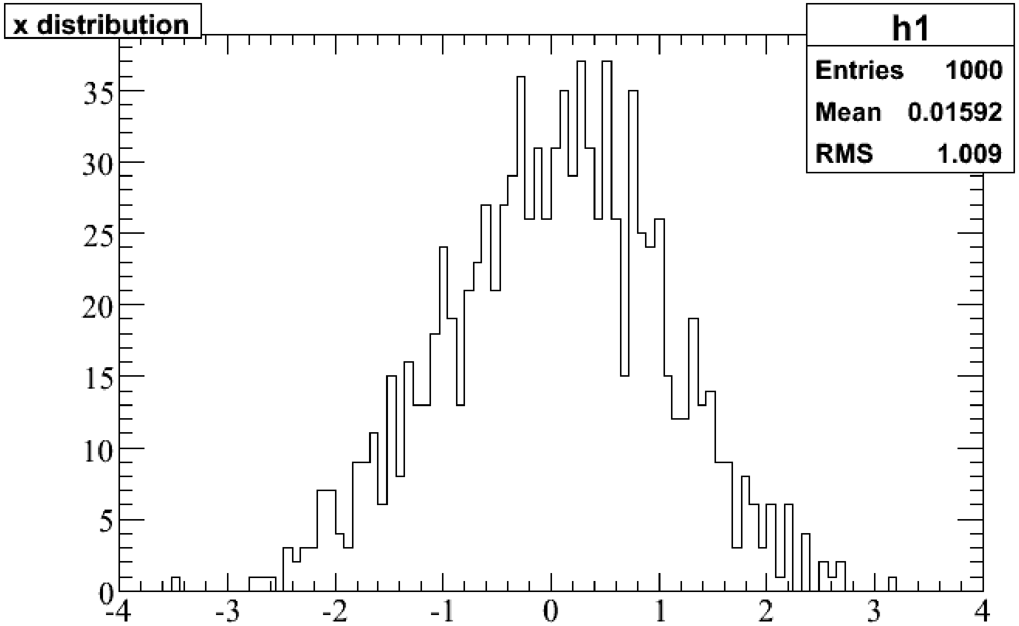
ROOT : A super quick recap

1-Dimensional Histograms : TH1I,TH1F,TH1D

- `TH1F* name = new TH1F("name","Title", Bins, lowest bin, highest bin);`

- **Example**:

```
TH1F* h1 = new TH1F("h1","x distribution",100,-4,4);
Float_t x = 0;
for(Int_t i=0; i<1000; i++){
    x=gRandom->Gaus(0,1);
    h1->Fill(x);
}
h1->Draw();
```



The complete list of Methods can be found in:
http://root.cern.ch/root/html/TH1F.html

Histograms

```
// using different constructors

TH1I* h1 = new TH1I("h1", "h1 title", 100, 0.0, 4.0);
TH2F* h2 = new TH2F("h2", "h2 title", 40, 0.0, 2.0,30, -1.5, 3.5);
TH3D* h3 = new TH3D("h3", "h3 title", 80, 0.0, 1.0,100, -2.0, 2.0,50, 0.0, 3.0);

// cloning a histogram
TH1F* hc = (TH1F*)h1->Clone("title of the cloned histogram");

// projecting histograms
// the projections always contain double values !

TH1D* hx = h2->ProjectionX(); // ! TH1D, not TH1F
TH1D* hy = h2->ProjectionY(); // ! TH1D, not TH1F
```

Histograms : A complex example

```
// histograms filled and drawn in a loop
void hsum() {

TCanvas *c1 = new TCanvas("c1","The HSUM example",200,10,600,400);
c1->SetGrid();


// Create some histograms.
TH1F *total  = new TH1F("total","This is the total distribution",100,4,4);
TH1F *main   = new TH1F("main","Main contributor",100,-4,4);
TH1F *s1     = new TH1F("s1","This is the first signal",100,-4,4);
TH1F *s2     = new TH1F("s2","This is the second signal",100,-4,4);

total->Sumw2();  // store the sum of squares of weights
total->SetMarkerStyle(21);
total->SetMarkerSize(0.7);
main->SetFillColor(16);
s1->SetFillColor(42);
s2->SetFillColor(46);
```

Macro: hSum.C

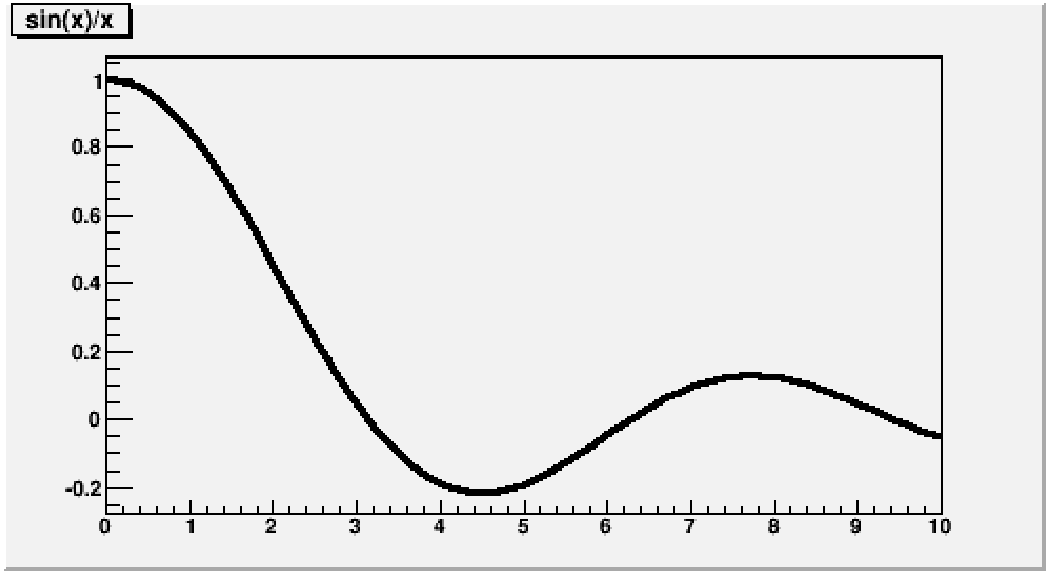
Histograms : A complex example

```
// Fill histograms randomly

    gRandom->SetSeed();

    Float_t xs1, xs2, xmain;
    for (Int_t i=0; i<10000; i++) {
        xmain = gRandom->Gaus(-1,1.5);
        xs1   = gRandom->Gaus(-0.5,0.5);
        xs2   = gRandom->Landau(1,0.15);
        main->Fill(xmain);
        s1->Fill(xs1,0.3);
        s2->Fill(xs2,0.2);
        total->Fill(xmain);
        total->Fill(xs1,0.3);
        total->Fill(xs2,0.2);
     }
  }
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
```

Macro: hSum.C

Formulas and Functions

```
void formula2(){
    TCanvas *c1 = new TCanvas("c1","Example with Formula",500,500);
    // Create a one dimensional function and draw it //
    TF1 * fun1 = new TF1("fun1","sin(x)/x",0,10);
    c1->cd(1);
    fun1->Draw();
    cout << " Deriv " << fun1->Derivative(2.) << endl;
    cout << " Integral " << fun1->Integral(0.,3.) << endl;
    cout << " Func val " << fun1->Eval(1.2456789) << endl;
  }
```



Macro: formula2.C

Creating a TF1 with Parameters

The second way to construct a **TF1** is to add parameters to the expression. Here we use two parameters:

```
root[] TF1 *f1 = new TF1("f1","[0]*x*sin([1]*x)",-3,3);
```

The parameter index is enclosed in square brackets. To set the initial parameters explicitly you can use:

```
root[] f1->SetParameter(0,10);
```

This sets parameter 0 to 10.
You can also use SetParameters to set multiple parameters at once.

```
root[] f1->SetParameters(10,5);
```

This sets parameter 0 to 10 and parameter 1 to 5.
We can now draw the **TF1**:

```
root[] f1->Draw();
```

Creating a TF1 with a User Function

```
// define a function with 3 parameters
Double_t fitf(Double_t *x,Double_t *par){
   Double_t arg = 0;
   if (par[2] != 0) arg = (x[0] - par[1])/par[2];
   Double_t fitval = par[0]*TMath::Exp(-0.5*arg*arg);
   return fitval;
}


void userfunctexample() {
   // Create a TF1 object using the function defined above. The last
   //  parameter specify the number of parameters for the function.
   TF1 * func = new TF1("fit",fitf,-3,3,3);
   // set the parameters to the mean and RMS of the histogram
   func->SetParameters(500,0.,0.5);
   // give the parameters meaningful names
   func->SetParNames ("Constant","Mean_value","Sigma");
   // call TH1::Fit with the name of the TF1 object
   func->Draw();
}
```


Macro: userfunctexample.C

Write and Read a ROOT file

- **Write (in a macro):**

```
TFile *myfile = new TFile("fillrandom.root","RECREATE");
myfile->cd();
form1->Write();
sqroot->Write();
h1f->Write();
myfile->Close();
```

- **Read (in a macro):**

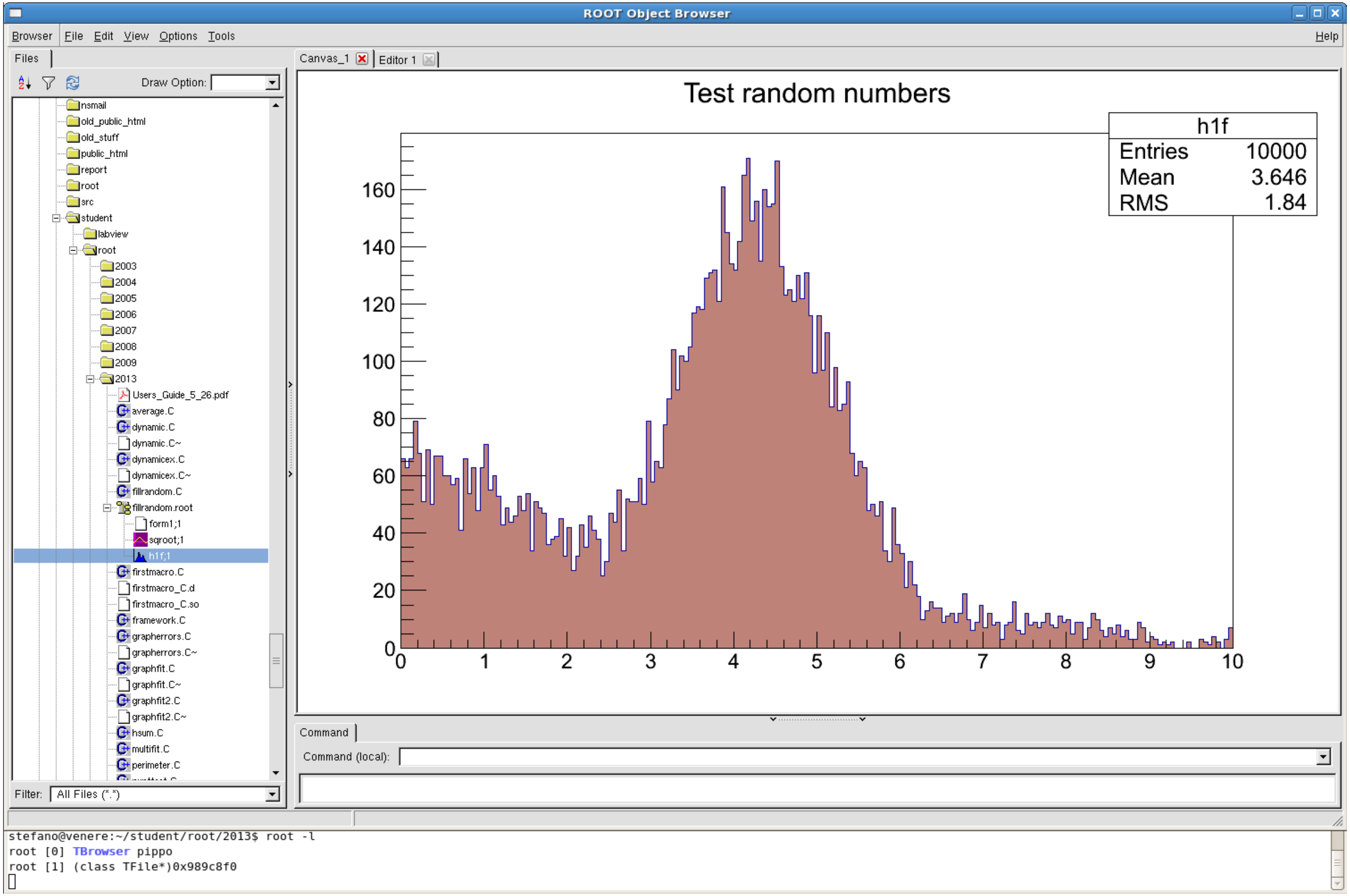
```
TFile *pfile = new TFile("fillrandom.root");
TH1F * ph1 = (TH1F *) pfile->Get("h1f"); // take the object from the TFile
```

- **Read (in the terminal):**

```
root [0] TFile *pfile = new TFile("fillrandom.root");
root [1] pfile->ls(); // it is useful only to know what is stored inside your
TFile if you have no idea of how it has been created.
root [2] TH1F * ph1 = (TH1F *) pfile->Get("h1f"); // take the object from the
TFile
```

TBrowser : A GUI to analyze the TFile

```
root [0] new TBrowser()
```



Passing arguments to a macro

```
// define a function with 3 parameters
Double_t fitf(Double_t *x,Double_t *par){
   Double_t arg = 0;
   if (par[2] != 0) arg = (x[0] - par[1])/par[2];
   Double_t fitval = par[0]*TMath::Exp(-0.5*arg*arg);
   return fitval;
}

void userfunctexample(Int_t min = -3, Int_t max = 3) {
   // Create a TF1 object using the function defined above. The last
   //  parameter specify the number of parameters for the function.
   TF1 * func = new TF1("fit",fitf,min,max,3);
   // set the parameters to the mean and RMS of the histogram
   func->SetParameters(500,0.,0.5);
   // give the parameters meaningful names
   func->SetParNames ("Constant","Mean_value","Sigma");
   // call TH1::Fit with the name of the TF1 object
   func->Draw();
   TString nameout = “Funz.”;
   nameout+=min;
   nameout+=”.”;
   nameout+=”.root”;
   TFile *fo = new Tfile(nameout,”RECREATE”);
   fo->cd();
   func->Write();
   fo->Close();
}
```

Documentation

- https://wwwusers.ts.infn.it/~lea/lacd2016.html
- root.cern.ch
- In any case

Documentation

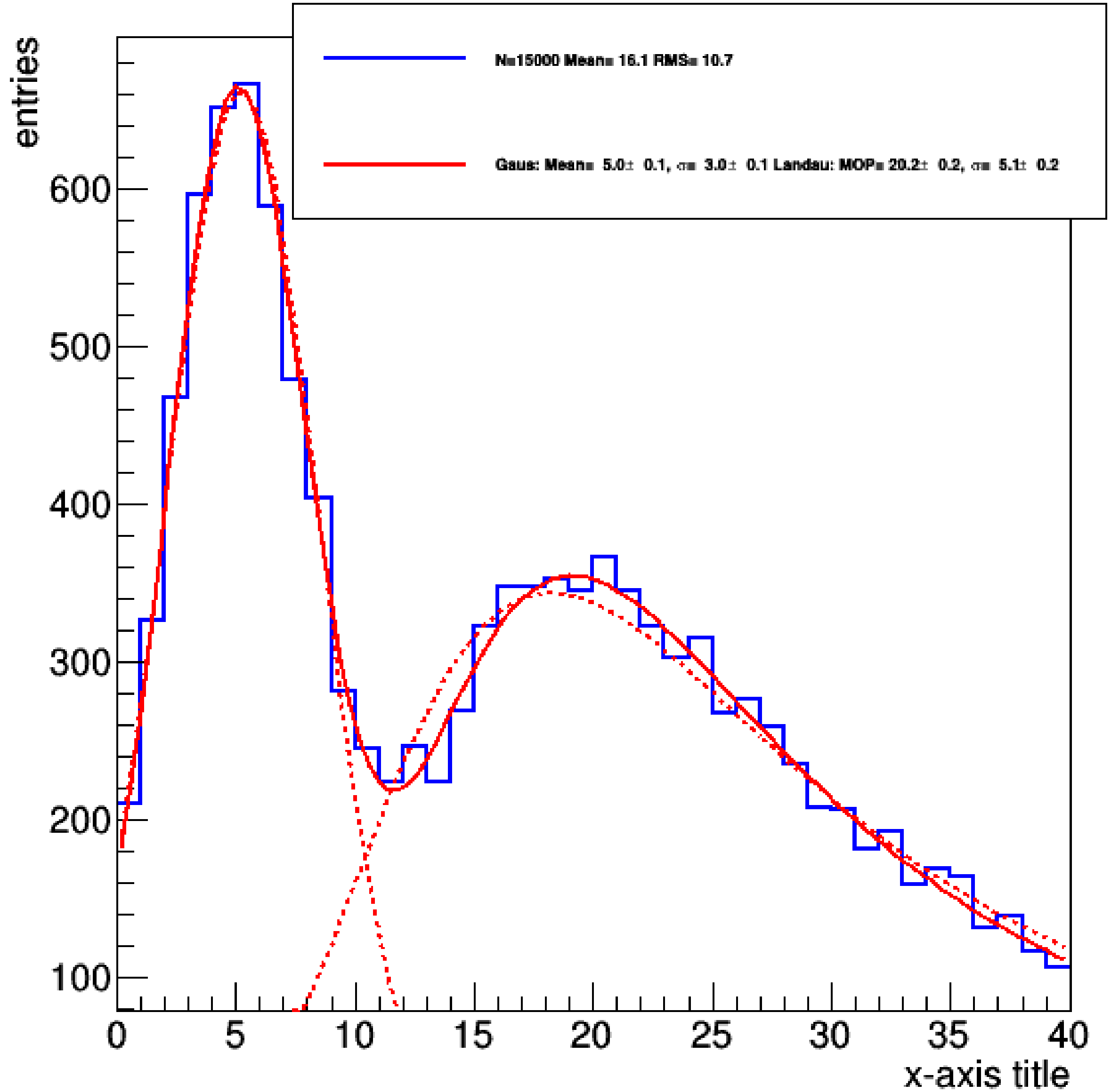
- https://wwwusers.ts.in
- root.cern.ch
- In any case

Esercitazione 10

Esercizio 1

- Fill a histogram randomly (n=10,000) with a Landau distribution with a most probable value at 20 and a “width” of 5 (use the ROOT website to find out about the Landau function)
- Fill the same histogram randomly (n=5,000) with a Gaussian distribution centered at 5 with a “width” of 3
- Write a compiled script with a fit function that describes the total histogram nicely (it might be a good idea to fit both peaks individually first and use the fit parameters for a combined fit)
- Add titles to x- and y-axis
- Include a legend of the histogram with number of entries, mean, and RMS values
- Add text to the canvas with the fitted function parameters
- Draw everything on a square-size canvas (histogram in blue, fit in red)
- Save as png, eps, and root file

Esercitazione 10



Simulations with ROOT

Thanks to Prof. Massimo Masera for the next slides

Monte Carlo technique

- Monte Carlo refers to any procedure that makes use of random numbers.
- Monte Carlo methods are used in:
  - Simulation of natural phenomena
  - Simulation of experimental apparatus
  - Numerical analysis (e.g. Integral evaluation)
- What is a Random Number?
  - Is e.g. 3 a random number?
  - A sequence of random numbers is a set of numbers that have nothing to do with the other numbers in the sequence

Monte Carlo technique

- General procedure:
  - A sequence of m random numbers with uniform distribution in the [0,1] interval is extracted.
  - This sequence is used to produce a second sequence distributed according to a generic f(x), which is the pool of simulated data

Pseudo-Random numbers

- The sequence is reproducible, because the algorithm for the generation is deterministic.
- General characteristic of a random generator:
  - Statistical independence
  - “Long” repetition period
  - The sequence looks random, when indeed it is not

Random generators in ROOT

Are implemented in the **TRandom** class: fast generator with a short (10^9) period, based of the linear congruential method.

- TRandom1: inherits from TRandom and implements RANLUX
- TRandom2: inherits from TRandom has a period of 1026, but only 3 32 bits word
- TRandom3:inherits from TRandom and implements the Mersenne-Twister generator which has a period of $2^{19937}-1$ ($\approx 10^{6002}$).

Here are the CPU times obtained using the four random classes on an lxplus machine with an Intel 64 bit architecture and compiled using gcc 3.4:

	TRandom (ns/call)	**TRandom1** (ns/call)	**TRandom2** (ns/call)	**TRandom3** (ns/call)
`Rndm()`	-	-	6	9
`Gaus()`	31	161	35	42
`Rannor()`	116	216	126	130
`Poisson(m=10)`	147	1161	162	239
`Poisson(m=10)UNURAN`	80	294	89	99

BAD

SLOW

FAST

GOOD (default)

Simulation of a radioactive decay of a single nucleus

- Radioactive decay is an intrinsic random process: the probability of decay is constant (independent of the age of nuclei)
- The probability that a nucleus decays in the time Δt is p :

$$p = \alpha \Delta t \ \ (\mathrm{per}\, \alpha \Delta t \ll 1)$$

- Let's consider a system initially having N_0 unstable nuclei: how does the number of nuclei vary with time?

Simulation of a radioactive decay of a single nucleus

- The algorithm which has to be implemented is the following

```
LOOP from t=0 to T, step Δt
        Reset the number of decayed nuclei (N_dec) in the current time bin

        LOOP over each remaining parent nucleus (from 0 to N_tot)
                Decide if the nucleus decays:
                if(gRandom->Rndm()<αΔt)
                    increment the number of decayed nuclei in this time bin
                endif
        END loop over nuclei

        N_tot=N_tot-N_dec
        Plot N_tot vs t
END loop over time
END
```

Esercitazione 10 – Esercizio 2 (Decay.C)

- Write a macro to implement the algorithm shown in the previous slide. Show the number of remaining nuclei as a function of time for the two following cases:
  - N_0=1000, α=0.01 s^{-1}, Δt=1 s
  - N_0=50000, α=0.03 s^{-1}, Δt=1 s
- Show the results in linear and logarithmic scale, for t between 0 and 300 seconds
- Compare the results with the expected result

$$dN = -N\alpha dt$$

$$\Downarrow$$

$$N = N_0 e^{-\alpha t}$$

Possible Solution

- In the next slide a possible solution implemented using a macro in ROOT is shown
- The ROOT classes are used to generate random numbers, to store informations in the histograms and for input/output operations
- The macro (decay.C) is composed by two funtions
- It can be interpret or exectuded by CLANG (CINT)

```
#if !defined(__CINT__) || defined(__MAKECINT__)
#include <TF1.h>
#include <TFile.h>
#include <TH1D.h>
#include <TMath.h>
#include <TRandom3.h>
#include <Riostream.h>
#endif

// Declare function
Double_t exponential(Double_t *x, Double_t *par);
//_______________________________________________________________
void Decay(Int_t n0 = 50000, Double_t alpha = 0.03, Double_t Delt = 1.0, Double_t ttot = 300, Int_t seed = 95689){

  gRandom->SetSeed(seed);
  Int_t Nbins = static_cast<Int_t>(ttot/Delt); // number of time intervals
  cout << "Numersof bins: "<<Nbins<<" di ampiezza "<<Delt<<" s";
  Double_t timetot = Delt*Nbins;  // totale time = ttot
  cout<<" Tempo totale "<<timetot<<endl;
  // histogram booking
  TH1D *h1 = new TH1D("h1","Remaining nuclei",Nbins+1,-Delt/2.,timetot+Delt/2.);
  h1->SetFillColor(kOrange-4);
  //Theoretical function
  TF1 *fteo = new TF1("fteo",exponential,0.,timetot,2);
  fteo->SetLineColor(kRed);
  Double_t N0 = n0;
  Double_t ALPHA = alpha;
  fteo->SetParameters(N0,ALPHA);
  fteo->SetParNames("normalizzazione","coefficiente");

  Double_t prob = alpha*Delt; //probability
  h1->Fill(0.,static_cast<double>(n0));
  for(Double_t time=Delt; time<timetot+Delt/2.; time+=Delt){
    Int_t ndec = 0;
    for(Int_t nuclei=0; nuclei<n0;nuclei++)if(gRandom->Rndm()<prob)ndec++;
    n0-=ndec;
    h1->Fill(time,static_cast<double>(n0));
  }

  TFile *file = new TFile("decay.root","recreate");
  h1->Write();
  fteo->Write();
  h1->Draw("histo");
  fteo->Draw("same");
  file->Close();
}

//_______________________________________________________________
Double_t exponential(Double_t *x, Double_t *par){
  Double_t xx = x[0];
  return par[0]*exp(-par[1]*xx);
}
```

Header files, need to compile the macro

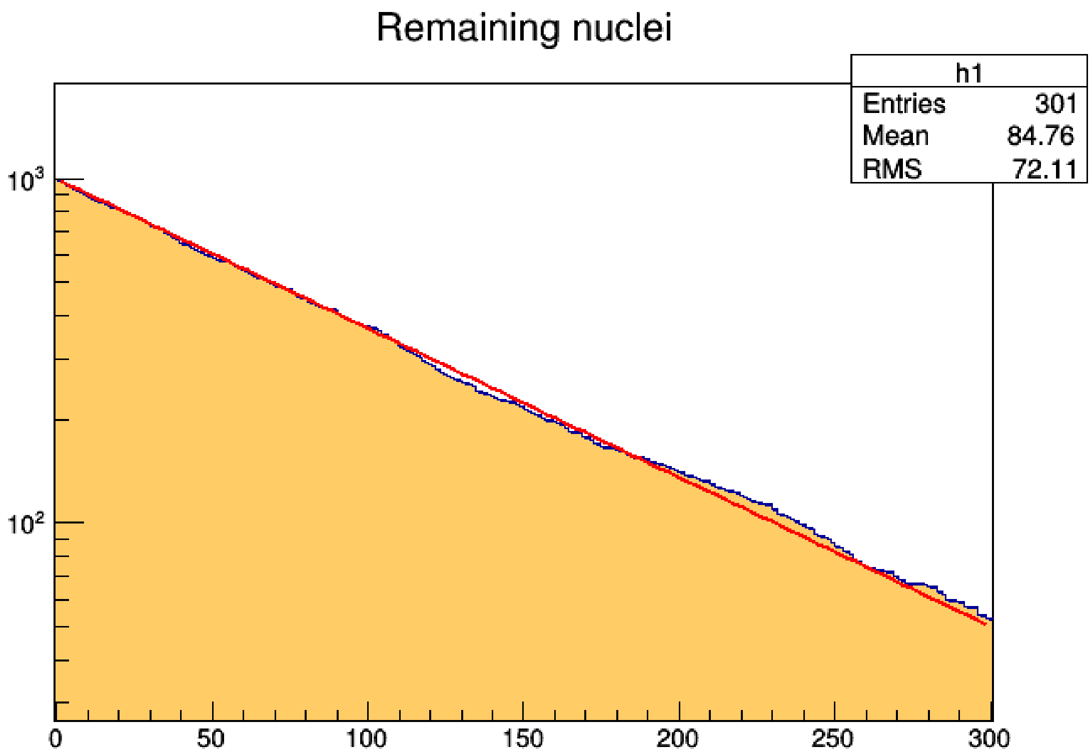
```
#if !defined(__CINT__) || defined(__MAKECINT__)
#include <TF1.h>
#include <TFile.h>
#include <TH1D.h>
#include <TMath.h>
#include <TRandom3.h>
#include <Riostream.h>
#endif

// Declare function
Double_t exponential(Double_t *x, Double_t *par);
//
void Decay(Int_t n0 = 50000, Double_t alpha = 0.03, Double_t Delt = 1.0, Double_t ttot = 300, Int_t seed = 95689){

  gRandom->SetSeed(seed);
  Int_t Nbins = static_cast<Int_t>(ttot/Delt); // number of time intervals
  cout << "Numersof bins: "<<Nbins<<" di ampiezza "<<Delt<<" s";
  Double_t timetot = Delt*Nbins;  // totale time = ttot
  cout<<" Tempo totale "<<timetot<<endl;
  // histogram booking
  TH1D *h1 = new TH1D("h1","Remaining nuclei",Nbins+1,-Delt/2.,timetot+Delt/2.);
  h1->SetFillColor(kOrange-4);
  //Theoretical function
  TF1 *fteo = new TF1("fteo",exponential,0.,timetot,2);
  fteo->SetLineColor(kRed);
  Double_t N0 = n0;
  Double_t ALPHA = alpha;
  fteo->SetParameters(N0,ALPHA);
  fteo->SetParNames("normalizzazione","coefficiente");

  Double_t prob = alpha*Delt; //probability
  h1->Fill(0.,static_cast<double>(n0));
  for(Double_t time=Delt; time<timetot+Delt/2.; time+=Delt){
    Int_t ndec = 0;
    for(Int_t nuclei=0; nuclei<n0;nuclei++)if(gRandom->Rndm()<prob)ndec++;
    n0-=ndec;
    h1->Fill(time,static_cast<double>(n0));
  }

  TFile *file = new TFile("decay.root","recreate");
  h1->Write();
  fteo->Write();
  h1->Draw("histo");
  fteo->Draw("same");
  file->Close();
}

//_______________________________________________________________
Double_t exponential(Double_t *x, Double_t *par){
  Double_t xx = x[0];
  return par[0]*exp(-par[1]*xx);
}
```

Functions are declared before their implementation. Default values can be Passed as “default” argument of the function

```
#if !defined(__CINT__) || defined(__MAKECINT__)
#include <TF1.h>
#include <TFile.h>
#include <TH1D.h>
#include <TMath.h>
#include <TRandom3.h>
#include <Riostream.h>
#endif

// Declare function
Double_t exponential(Double_t *x, Double_t *par);
//_____________________________________________________________
void Decay(Int_t n0 = 50000, Double_t alpha = 0.03, Double_t Delt = 1.0, Double_t ttot = 300, Int_t seed = 95689){

  gRandom->SetSeed(seed);
  Int_t Nbins = static_cast<Int_t>(ttot/Delt); // number of time intervals
  cout << "Numersof bins: "<<Nbins<<" di ampiezza "<<Delt<<" s";
  Double_t timetot = Delt*Nbins;  // totale time = ttot
  cout<<" Tempo totale "<<timetot<<endl;
  // histogram booking
  TH1D *h1 = new TH1D("h1","Remaining nuclei",Nbins+1,-Delt/2.,timetot+Delt/2.);
  h1->SetFillColor(kOrange-4);
  //Theoretical function
  TF1 *fteo = new TF1("fteo",exponential,0.,timetot,2);
  fteo->SetLineColor(kRed);
  Double_t N0 = n0;
  Double_t ALPHA = alpha;
  fteo->SetParameters(N0,ALPHA);
  fteo->SetParNames("normalizzazione","coefficiente");

  Double_t prob = alpha*Delt; //probability
  h1->Fill(0.,static_cast<double>(n0));
  for(Double_t time=Delt; time<timetot+Delt/2.; time+=Delt){
    Int_t ndec = 0;
    for(Int_t nuclei=0; nuclei<n0;nuclei++)if(gRandom->Rndm()<prob)ndec++;
    n0-=ndec;
    h1->Fill(time,static_cast<double>(n0));
  }

  TFile *file = new TFile("decay.root","recreate");
  h1->Write();
  fteo->Write();
  h1->Draw("histo");
  fteo->Draw("same");
  file->Close();
}
```

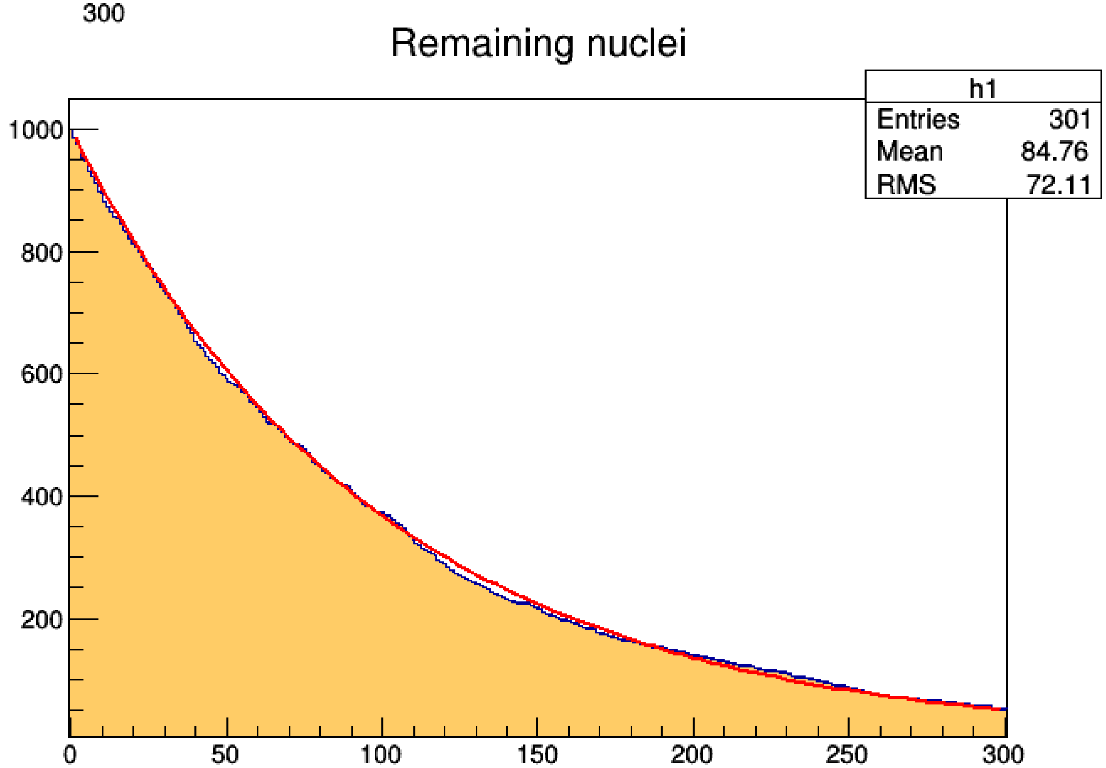
Definition of the “exponential function”
That can be used in the TF1 definition

```
//_____________________________________________________________
Double_t exponential(Double_t *x, Double_t *par){
  Double_t xx = x[0];
  return par[0]*exp(-par[1]*xx);
}
```



Continuous line: expected exponential.

Histogram: simulation result.



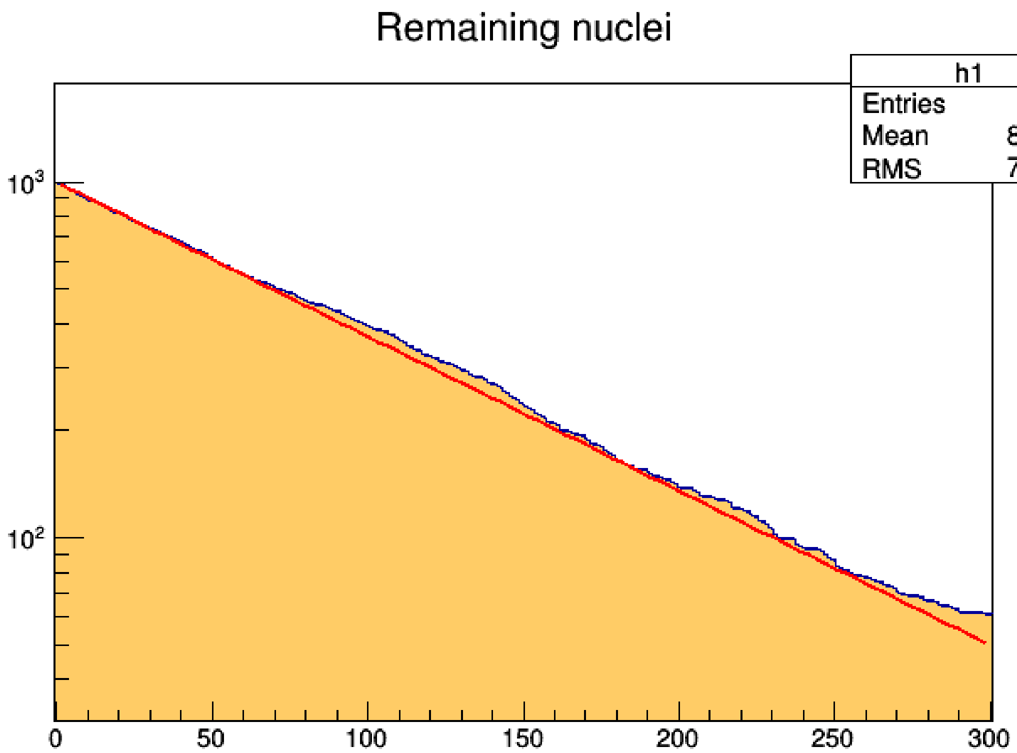
Result for:

N_0=100,

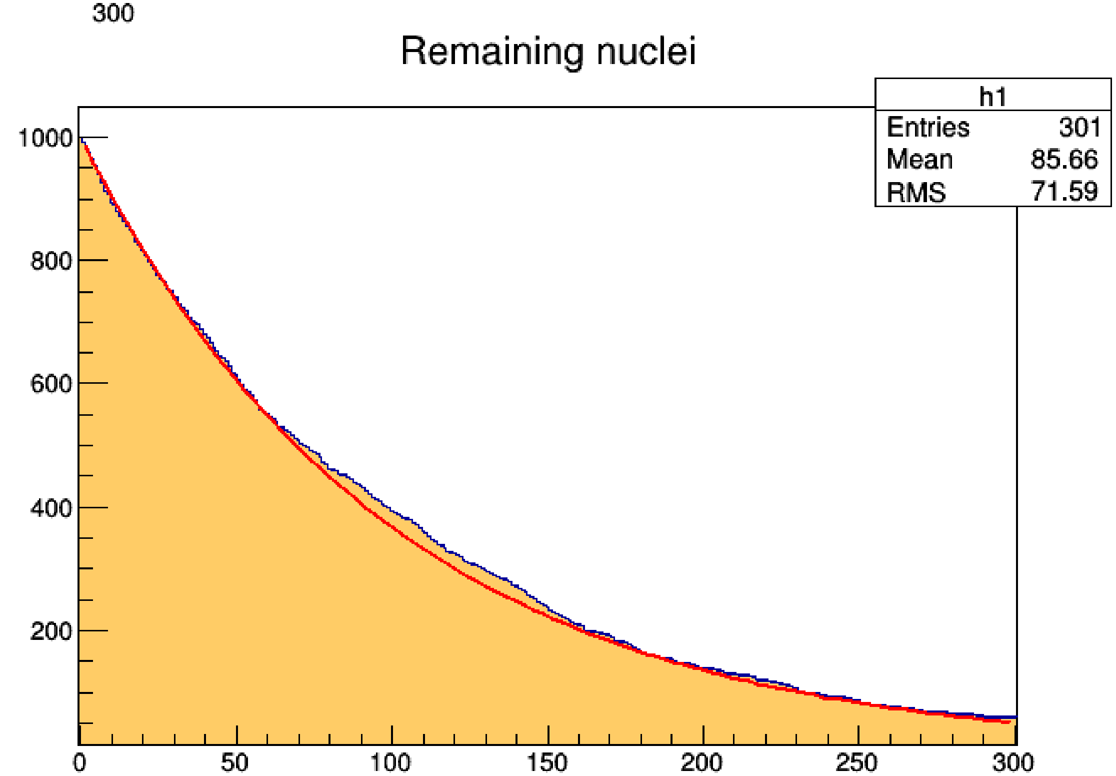
$\alpha=1\times10^{-2}\ s^{-1}$

Δt=1 s

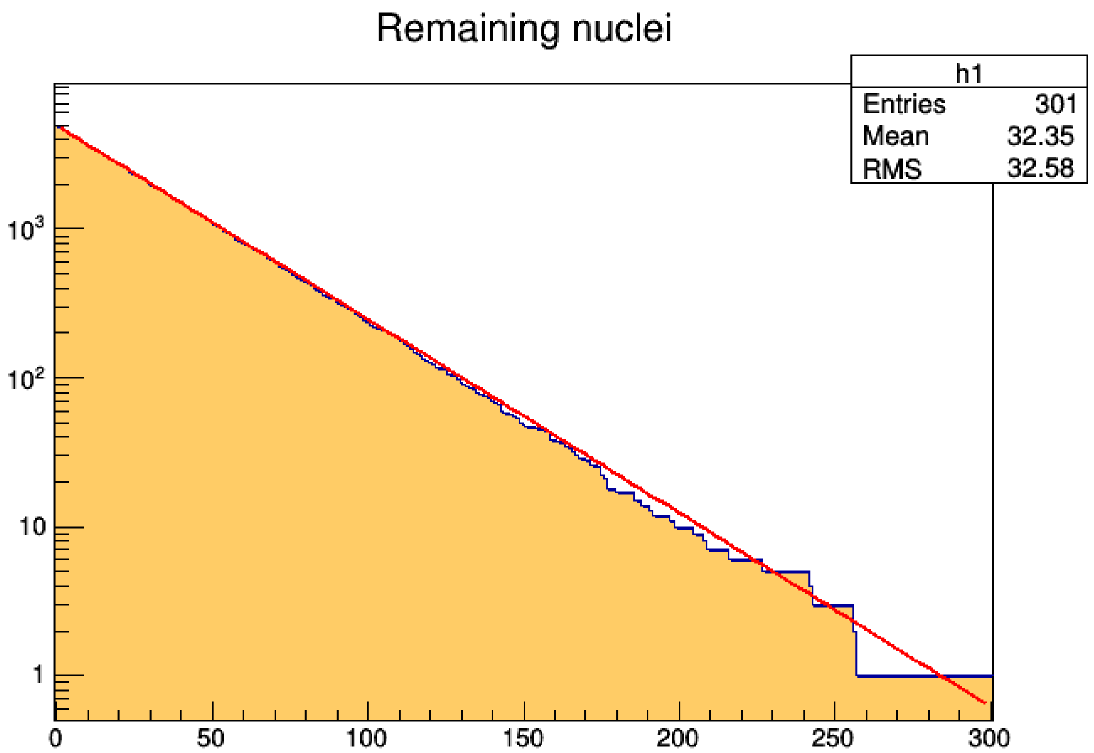
Statistical fluctuation are very important



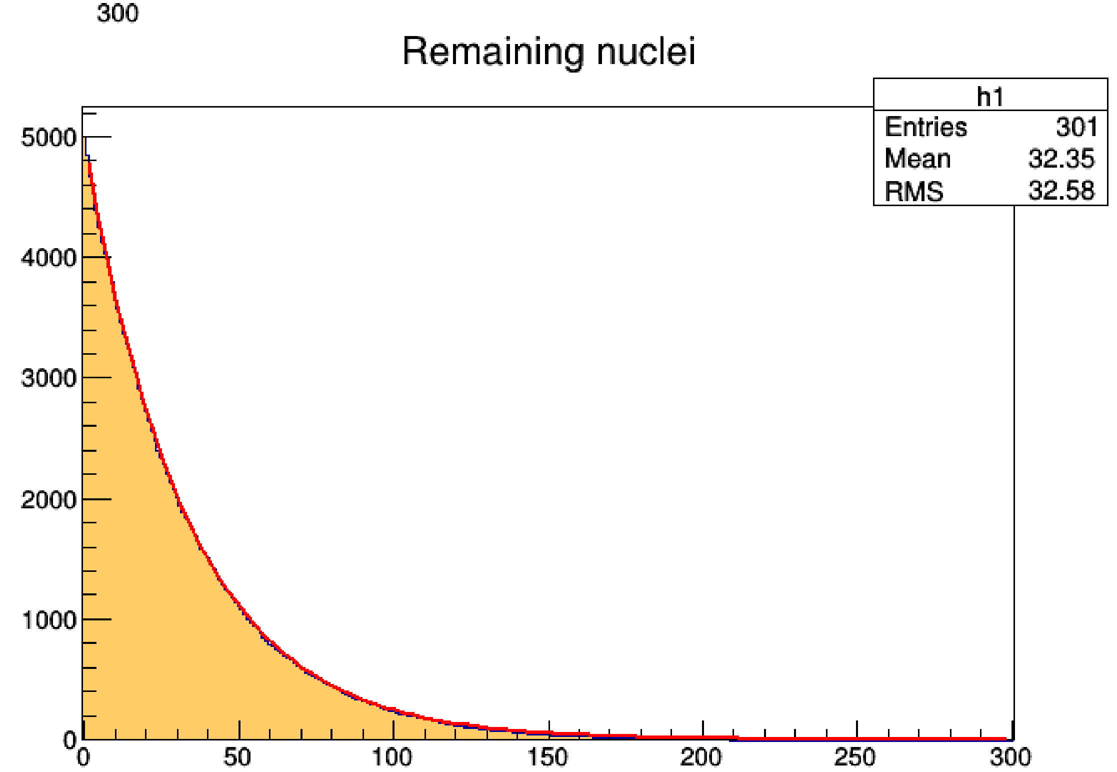
Same parameters as before, but different seed: the results are different!



Risultato per:
N_0=100,
$\alpha = 1\times10^{-2}\ s^{-1}$
Δt=1 s



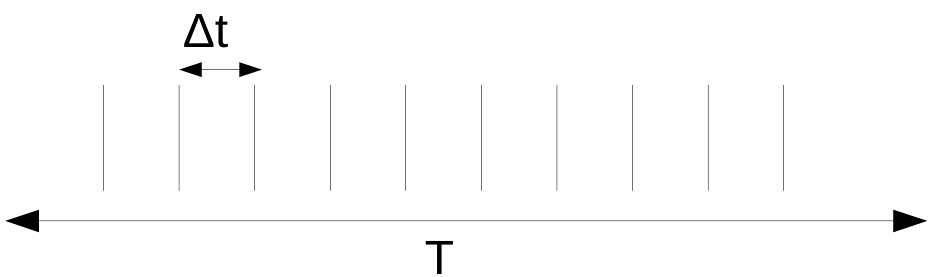
The importance of fluctuations depends on the number N of residual nuclei



Result for:
N_0=5000,
$\alpha=3\times10^{-2}\ s^{-1}$
Δt=1 s

How many decays in a fixed time interval T?

- Let's assume the number of decays in time T is much less than the number of parent nuclei. (i.e. let's assume a constant probability to observe a decay)
- The time T can be break into *m* shorter intervals of duration Δt (Δt/m):



- The probability to observe 1 decay in time Δt is :

$$p=\beta\,\Delta t$$

How many decays in a fixed time interval T?

$$p = \beta \Delta t$$

- β = αN (Note that α is the decay probability)
- Δt must be small enough so that βΔt << 1
- The probability of observing n decays in time T is given by the binomial distribution:

$$P(\text{n decays in m intervals}) = \binom{m}{n} p^n (1-p)^{(m-n)}$$

$$P = \frac{m!}{n!(m-n)!} \left(\frac{\beta T}{m}\right)^n \left(1 - \frac{\beta T}{m}\right)^{(m-n)}$$

How many decays in a fixed time interval T?

- In the limit of large m (m→ ∞ and Δt → 0)

$$\left(1-\frac{\beta T}{m}\right)^{m} \rightarrow e^{-\beta T}$$

$$\left(1-\frac{\beta T}{m}\right)^{-n} \rightarrow 1$$

$$\frac{m!}{(m-n)!} \rightarrow m^{n}$$

$$P(n;\beta T)=\frac{(\beta T)^{n}}{n!}e^{-\beta T}=\frac{\mu^{n}e^{-\mu}}{n!}$$

Known as Poisson distribution

Esercitazione 10 – Esercizio 3 (poisson.C)

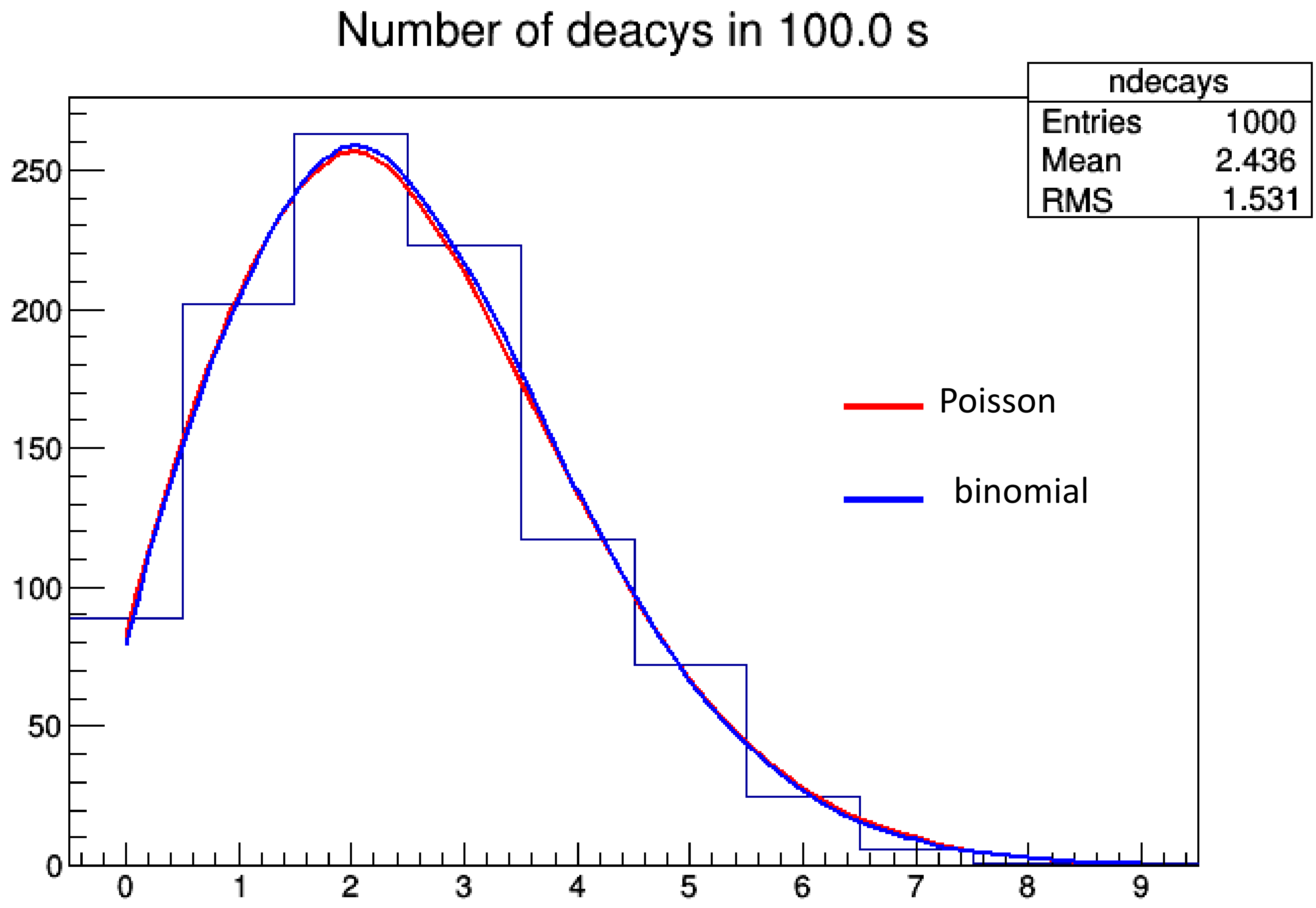
- Modify the program written for exercise 2 to simulate an experiment that counts the number of decays observed in a time interval, T .
- Allow the experiment to be repeated and histogram the distribution of the number of decays for the following two cases:

  N_0=1000, $\alpha=2.5\times10^{-5}\ s^{-1}$, Δt=1 s, T = 100 s

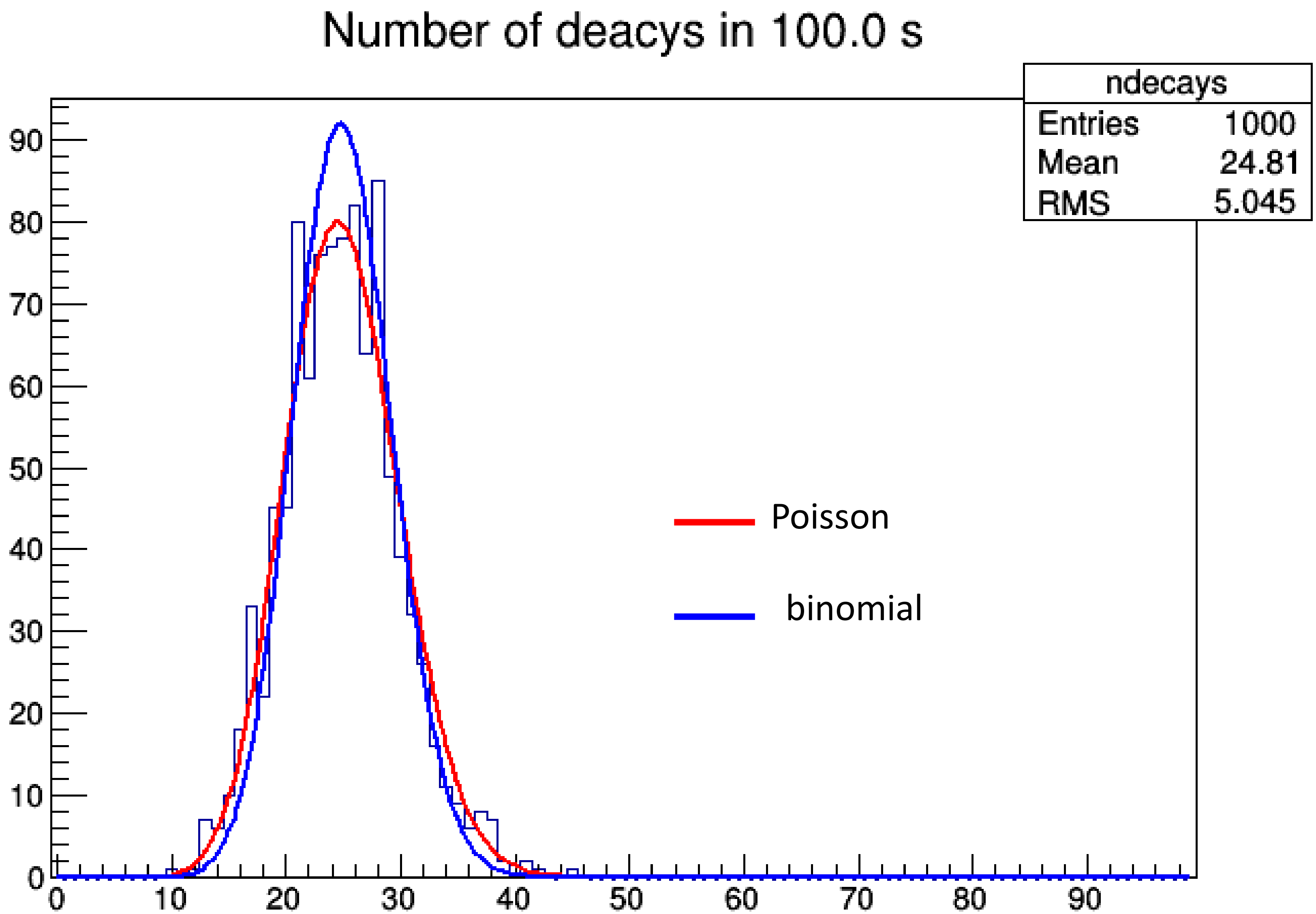
  N_0=1000, $\alpha=2.0\times 10^{-4}\ s^{-1}$, Δt=1 s, T = 100s

- Compare the distribution of n for 1000 experiments with Binomial and Poisson distributions.

Result for: N_0=1000, α=2.5X10^{-5}, Δt=1 s, T=100 s



Result for: N_0=1000, α=2.5X10^{-4}, Δt=1 s, T=100 s



```
// Function declaration
double Decay(int n0,double alpha, double Delt,double timetot);  // Decay simulation

double poissonian(double xx, double norm, double p); // POISSONIAN density
double binomial(double xx, double norm, double ntrials, double prob);   // BINOMIAL density
void poisson(int n0 = 1000,double alpha = 2.5e-5, double Delt = 1.,double time=100.,unsigned int seed = 1234); //main function

/////////////////////////////////////////////////////////////////////////////
void poisson(int n0,double alpha, double Delt,double time,unsigned int seed){

  gRandom->SetSeed(seed);

  TString title = Form("Number of deacys in %4.1f s",time);
  int nbins = static_cast<int>(n0*alpha*time*4);
  cout<<"numer of bins"<<nbins<<endl;
  double high = nbins-0.5;
  TH1F *ndecays = new TH1F("ndecays",title,nbins,-0.5,high);

  //theoretical function (poissonian)
  double mu = alpha*n0*time;
  title =Form("Poissonian distribution with parameter %10.4g ",mu);

  TH1F *fteo = new TH1F("fteo",title,nbins,-0.5,high);

  //theoretical function 2(binomial)
  double ntrials = time/Delt;
  double prob = alpha*n0*Delt;
  title = Form("Binomial with parametris p= %10.4g e m = %8.2g",prob,ntrials);

  TH1F *fteo2 = new TH1F("fteo2",title,nbins,-0.5,high);

  cout<<"Numer of nuclei: "<<n0<<endl;
  cout<<"Parameter of poissonian distribution "<<mu<<endl;
  cout<<"Parameters of binomial distribution p= "<<prob<<" m ="<< ntrials<<endl;


  // Fill histo with theoretical functions
  for (double x=0.;x<high;x+=1.)
    fteo->Fill(x,poissonian(x,knorm,mu));
  for (double x=0.;x<high;x+=1.)
    fteo2->Fill(x,binomial(x,knorm,ntrials,prob));

  // Simulation of the decay
  for(int exper = 0; exper<knorm; exper++){
    if((exper%100)==0)cout<<"Processinf event n. "<<exper<<endl;
    double nodecay=Decay(n0,alpha,Delt,time);
    ndecays->Fill(nodecay);
  }
```

```
///////////////////////////////////////////////////////////////////////////
double Decay(int n0,double alpha, double Delt,double timetot){
  double prob = alpha*Delt; //probabilita
  int n0init = n0;
  for(double time=0.; time<timetot; time+=Delt){
    int ndec = 0;
    for(int nuclei=0; nuclei<n0;nuclei++)if(gRandom->Rndm()<prob)ndec++;
    n0-=ndec;
  }
  return static_cast<double>(n0init-n0);
}
///////////////////////////////////////////////////////////////////////////
double poissonian(double xx, double norm, double p){
  return norm*(TMath::PoissonI(xx,p));
  //  return norm*(TMath::Power(param,xx)*TMath::Exp(-param-Gamma(xx+1)));
}
///////////////////////////////////////////////////////////////////////////
double binomial(double xx, double norm, double ntrials, double prob)  {

  Double_t n = TMath::Binomial(ntrials,xx);
  n *= TMath::Power(prob,xx)*TMath::Power((1-prob),(ntrials-xx));
  n *= norm;
  return n;
```