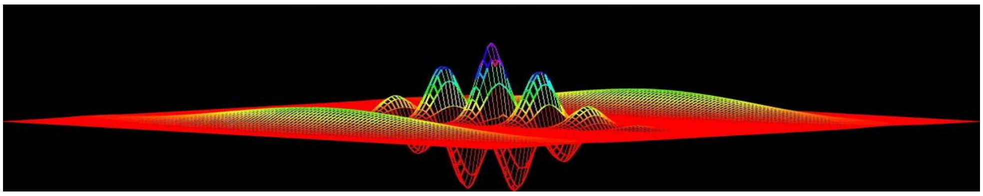
Computational Physics

numerical methods with C++ (and UNIX)



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: barao@lip.pt

Computational Physics

ROOT

A data analysis graphics tool with a C++ interpreter

Fernando Barao, Phys Department IST (Lisbon)

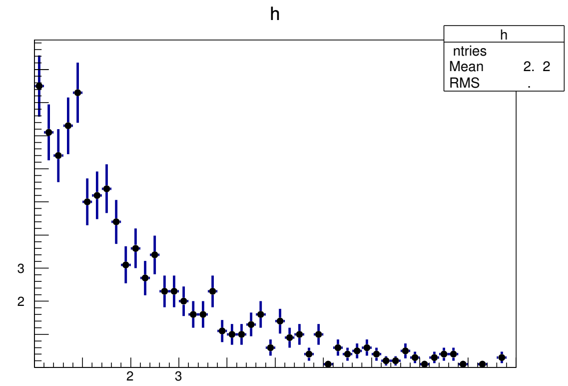
ROOT - graphics and muons

muons particles arrive all time to earth from cosmic rays showers induced by primary protons coming from the galaxy

their lifetime is short ($2.2\ \mu$sec) but Lorentz boost dilates their time and we can "see" them at earth surface ! ! !

let’s simulate decay time that follows and exponential law with a time constant of $2.2\ \mu$sec

```
TCanvas *c = new TCanvas();

//create histogram object
//0 to 10 microseconds (object name = "h")
TH1F h("h","h",50, 0.,10.);

//get exponential random
for (int i=0; i<1000; i++) {
  h.Fill(gRandom->Exp(2.2));
}
h.SetLineWidth(3);
h.SetMarkerStyle(20);
//DrawCopy provide us with a copy of the histogram
//here is mandatory because h goes out of scope
h.DrawCopy("E");

c->Update();
```



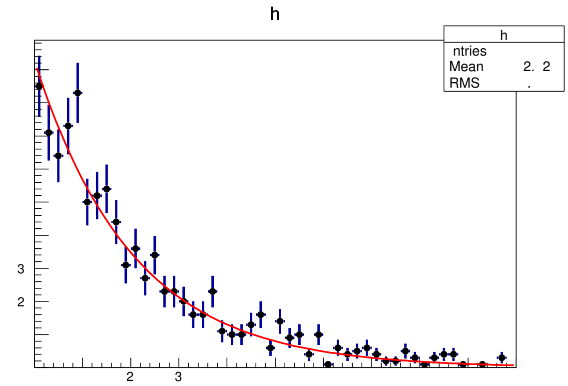
ROOT - muons (cont.)

let’s fit now the exponential law to the data
$N(t) = N_0 e^{-t/\tau}$ (2-parameters law)

```
TCanvas *c = new TCanvas();
//fit distribution
h.Fit("expo");
TF1 *f = h.GetFunction(‘‘expo’’);

h.DrawCopy("E");
f->DrawCopy("same");

//create histogram object
//0 to 10 microseconds (object name = "h")
TH1F h("h","h",50, 0.,10.);

//get exponential random
for (int i=0; i<1000; i++) {
  h.Fill(gRandom->Exp(2.2));
}
h.SetLineWidth(3);
h.SetMarkerStyle(20);
//DrawCopy provide us with a copy of the histo
//here is mandatory because h goes out of scop
h.DrawCopy("E");

c->Update();
```

TFormula

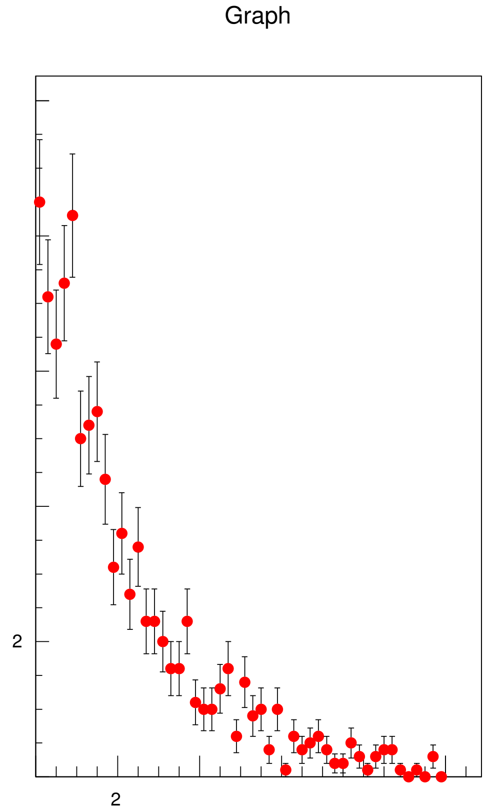
```
Example of valid expressions:
 sin(x)/x
 [0]*sin(x) + [1]*exp(-[2]*x)
 x + y**2
 x^2 + y^2
 [0]*pow([1],4)
 2*pi*sqrt(x/y)
 gaus(0)*expo(3)  + ypol3(5)*x
 gausn(0)*expo(3) + ypol3(5)*x
```



ROOT - retrieving histogram data

let's retrieve the histogram data and store them in a data file

```
///////////////////////// output measurements
ofstream F("muon-data.txt");
float *xbin = new float[h.GetNbinsX()];
float *vbin = new float[h.GetNbinsX()];
float *ey = new float[h.GetNbinsX()];
cout << "histo: nb bins = " << h.GetNbinsX() << endl
for (Int_t i=0;i<h.GetNbinsX();i++) {
  xbin[i] = h.GetBinCenter(i+1);
  vbin[i] = h.GetBinContent(i+1);
  ey[i] = h.GetBinError(i+1);
  F << xbin[i] << " " << vbin[i] << " "
    << ey[i] << endl;
}
F.close();
///////////////////////// make graph
TGraphErrors g("muon-data.txt",''%lg %lg %lg'');
g.SetMarkerColor(2);
g.SetMarkerStyle(20);
g.SetMarkerSize(1);
g.DrawClone(''AP'');
c1->Update();
```

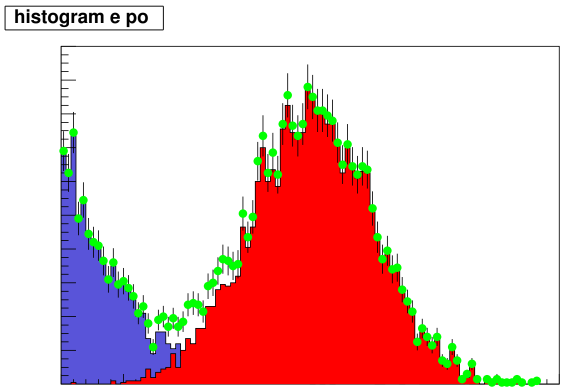


ROOT - running macros

A C++ function is known as a macro in ROOT. Let's make a macro that we name *hadd* for adding two histograms.

hadd.C

```
void hadd() {
  gROOT->SetStyle("Plain");
  gStyle->SetOptStat(0);
  TCanvas *c = new TCanvas();

  TH1F *hg = new TH1F("hg","histogram gauss", 100,1.,9.);
  for (int i=0; i<5000; i++) {hg->Fill(gRandom->Gaus(5,1.));}
  TH1F *he = new TH1F("he","histogram expo", 100,1.,9.);
  for (int i=0; i<5000; i++) {he->Fill(gRandom->Exp(1.));}

  TH1F *hsum = new TH1F(*hg); //dereference hg pointer
  hsum->Add(he,1.);

  he->GetYaxis()->SetRangeUser(0.,200.);
  he->SetFillColor(9);
  he->DrawCopy();
  hg->SetFillColor(kRed);
  hg->DrawCopy("same");
  hsum->SetMarkerStyle(20);
  hsum->SetMarkerColor(3);
  hsum->SetMarkerSize(1.2);
  hsum->Draw("Esame"); //draw with errors
}
```



ROOT - running macros (cont.)

The macro can be run at the unix prompt :

```
> root -l hadd.C
```

It can also be run with the CINT interpreter :

```
> root -l
root [0] .x hadd.C
```

It can also be loaded in CINT interpreter :

```
> root -l
root [0] .L hadd.C
root [1] hadd()
```

Running macro *hadd.C* within a macro

```
{
 gROOT->LoadMacro("hadd.C");
 hadd(); // calling function
}
```


ROOT - running macros (cont.)

Macros can be compiled with ACLIC (automatic compiler of libraries for CINT)
The compiled code runs much faster and language error checks easier !
The compilation process produces a shared library (.so) that can be used in ROOT
The shared library must be loaded before using user functions or classes

```
> root -l
root [0] .L hadd.C+
root [1] gSystem->Load("hadd_C.so")
root [2] hadd()
```

Notice that the include files of all functions being used in the code have to be added to the macro

```
#include "TROOT.h"
#include "TCanvas.h"
#include "TH1F.h"
#include "TRandom.h" //gRandom
#include "TStyle.h" //gStyle
void hadd() {
 ...
}
```

ROOT - running macros (cont.)

Remarks :

☐ the directory path name cannot be used with *.L*. In case of need replace it by :

```
root [0] gSystem->cd("directory_path")
root [0] gSystem->CompileMacro("hadd.C")
```

☐ If include files belonging to specific directories need to be include add their dir path through :

```
root [0] .include "-I$HOME/dir_path"

or within a macro:

gROOT->ProcessLine(".include my/include/dir");

or still:

gSystem->AddIncludePath(" -I/my/include/dir");
```


ROOT - compiling macros with g++

Compile a macro with g++

☐ The macro C++ code can be compatible with both running within ROOT and being compiled with **g++** by including at the end of the user code a conditional C++ code using the *__CINT__* pre-processor ROOT variable

```
void macro(int argc, char** argv) {
...
}
#ifndef __CINT__
#include "TApplication.h"
int main(int argc, char** argv) {
  gROOT->Reset();
  TApplication app("Comput Phys IST application", &argc, argv);
  macro(app.Argc(), app.Argv());
  app.Run();
}
#endif
```

ROOT - compiling macros with g++ (cont.)

Compile a macro with g++

- ☐ In this case, apart the inclusion of the header files *(.h)* as we did before when using ACLIC compiler, we need to link our C++ code with the *ROOT libraries*.
- ☐ ROOT libraries can be found using the *root-config tool*

```
> g++ -o macro.exe macro.C \      # compile
  -I `root-config --incdir` \   # ROOT include dir
  `root-config --cflags --libs`   # ROOT C++ flags and libs
```


ROOT - storing ROOT objects

ROOT allows to store objects (instances of ROOT classes) in a ROOT file - a file that is created using the *class TFile* of ROOT
Generically, all objects derived from the ROOT *class TObject* can be stored

✔ **macro that opens ROOT file and store objects (histogram and function)**

```
#include "TFile.h"
#include "TROOT.h"
#include "TH1F.h"
#include "TF1.h"
void ROOTio() {
  gROOT->Reset();
  TF1 *f1 = new TF1("f1","gaus()",0.,12.);
  TH1F *h = new TH1F("h", "histogram", 100, 0., 12.);
  // store objects
  TFile F("MyRootFile.root","RECREATE");
  f1->Write();
  h->Write();
  F.Close();
}
```

ROOT - storing ROOT objects (cont.)

✔ **retrieve stored objects using a macro**

```
#include "TFile.h"
#include "TROOT.h"
#include "TH1F.h"
#include "TF1.h"
#include "TIter.h"
#include "TKey.h"
#include "TString.h"
void ROOTio() {
  gROOT->Reset();
  TFile F("MyRootFile.root"); // open file
  TIter nextkey(F.GetListOfKeys());
  while ( TKey *key = (TKey*)nextkey() ) {
    TObject *obj = key->ReadObj(); // pointer to base class
    char* obj_name= obj->GetName();
    if ( obj->IsA()->InheritsFrom( "TH1" ) ) {
      TH1 *h = (TH1*)obj;
      h->DrawClone();
    }
    {
  F.Close();
}
```


ROOT - storing ROOT objects (cont.)

✔ **retrieve stored objects using CINT interpreter**

```
 > root -l MyRootFile.root
   Attaching file MyRootFile.root as _file0 ...
 root [0] _file0->ls(); // list objects
 root [1] h->Draw(); // draw histo
}
```

ROOT - additional tools

reset ROOT

The *gROOT->Reset()* command calls the destructors of all objects created on *stack memory* (created before runtime)

Warning ! Memory leakage comes from the fact that objects created by the user using the *new operator* are stored in the *heap memory* and are not deleted !

```
root [0] TH1F h("h","title", 100, 0.,10.); //object on stack
root [1] TH1F *h = new TH1F("h","title", 100, 0.,10.); //object or
root [2] gROOT->Reset() // removes stack objects
root [2] delete h; // deletes object from heap
```

Track memory leak within CINT

check the memory occupation within ROOT with the command *gObjectTable->Print()*

```
root [0] gObjectTable->Print()
root [1] .x hadd.C
root [2] gObjectTable->Print()
```


ROOT - classes interesting for Physics

There are many classes in ROOT useful to be used in Physics

- ✔ **TVector3**
  Suppose you are following a particle for studying its interactions in materials.
  In fact, a *particle track* can be defined with three-dimensional points
- ✔ **TLorentzVector** a four-dimensional vector used for relativistic and kinematics computations, both in spacetime (x, y, z, t) and in momentum space (px, py, pz, E). It is implemented as a TVector3 and a *Double_t* variable.

```
// using constructors:
TLorentzVector A(1.,2.,3.,4.);
TLorentzVector A(TVector3(5.,6.,7.),8.);

// setting elements
A.SetVect(TVector3(1,2,3));
A.SetXYZT(x,y,z,t);

//accessing elements
A.X() //
A.T() //
A.Px() //
A.E() //
```

ROOT - classes interesting for Physics (cont.)

There are many classes in ROOT useful to be used in Physics

- ✔ **TPhaseSpace**
  It generates kinematics events for decays of a particle into **N** particles with provided masses.
- ✔ **TParticlePDG**
  Interesting class to access the PDG database for retrieving information on particles properties, quantum numbers, decay channels and branching ratios. The all Particle Data Group booklet is provided in a ASCII file called "pdg_table.txt" .