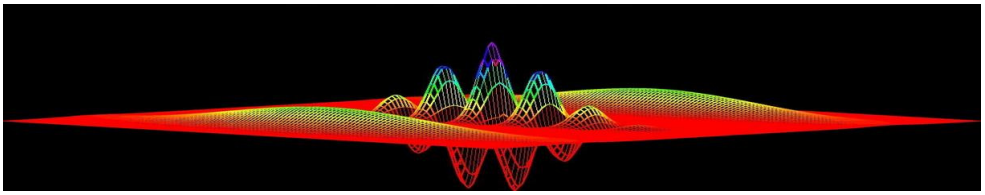
Computational Physics

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 - outline

- ✔ ROOT installation
- ✔ general concepts
- ✔ interactive use and macros
- ✔ canvas and graphics style
- ✔ histograms and other objects
- ✔ fitting
- ✔ input/ouput
- ✔ using ROOT from user programs
- ✔ DUBNA

site : `http://root.cern.ch`

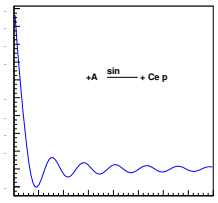
Users Guide : `http://root.cern.ch/drupal/content/root-users-guide-600`

ROOT - function plotter

☐ **ROOT can be used to plot functions : classes TF1, TF2**

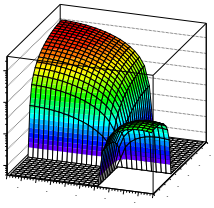
☐ **Plot function :** $f_1(x) = A\frac{sin(Bx)}{x} + Ce^x$

```
[0] gROOT->Reset();
[1] gStyle->SetOptTitle(0);
[2] TCanvas *c = new TCanvas("c", "Phys Comput canvas", 0, 0, 500, 500);
[3] TF1 *f1 = new TF1("f1","1.+ [0]*sin([1]*x)/x + [2]*exp(-x)",0.1, 40.);
[4] f1->SetParameters(1.,1.,1.);
[5] f1->SetLineColor(2);
[6] f1->GetHistogram()->GetXaxis()->SetTitle("x");
[7] f1->Draw();
[8] TLatex l(10.,2.0,''f(x) = 1 +A#frac{sin(Bx)}{x} + Cexp(-x)'');
[9] l.SetTextSize(0.04);
[10] l.Draw();
[11] c->Modified();
[12] c->SaveAs("Sfunctionplotter.eps");
```



☐ **Plot function :** $f_2(x,y) = \frac{sin(x)\cdot sin(y)}{x\cdot y}$

```
[13] TF2 *f2 = new TF2("f2","sin(x)*sin(y)/(x*y)",0,5,0,5);
[14] gPad->SetTheta(25);
[15] gPad->SetPhi(-110);
[16] gPad->SetLogz();
[17] f2->Draw("surf1"); //"", plot contours
```



ROOT - the histogram class

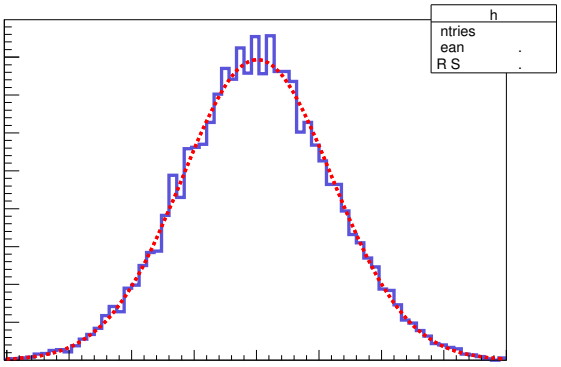
TH1.h

```
class TH1F : public TH1, public TArrayF {

public:
   TH1F();
   TH1F(const char *name,const char *title,Int_t nbinsx,Double_t xlow,Double_t xup);
   TH1F(const char *name,const char *title,Int_t nbinsx,const Float_t  *xbins);
   TH1F(const char *name,const char *title,Int_t nbinsx,const Double_t *xbins);
   TH1F(const TVectorF &v);
   TH1F(const TH1F &h1f);
   virtual ~TH1F();

   ...
};
```


ROOT - histograms

Let's make an histogram from random numbers generated from a gaussian of mean 5. **and standard deviation** 1.2

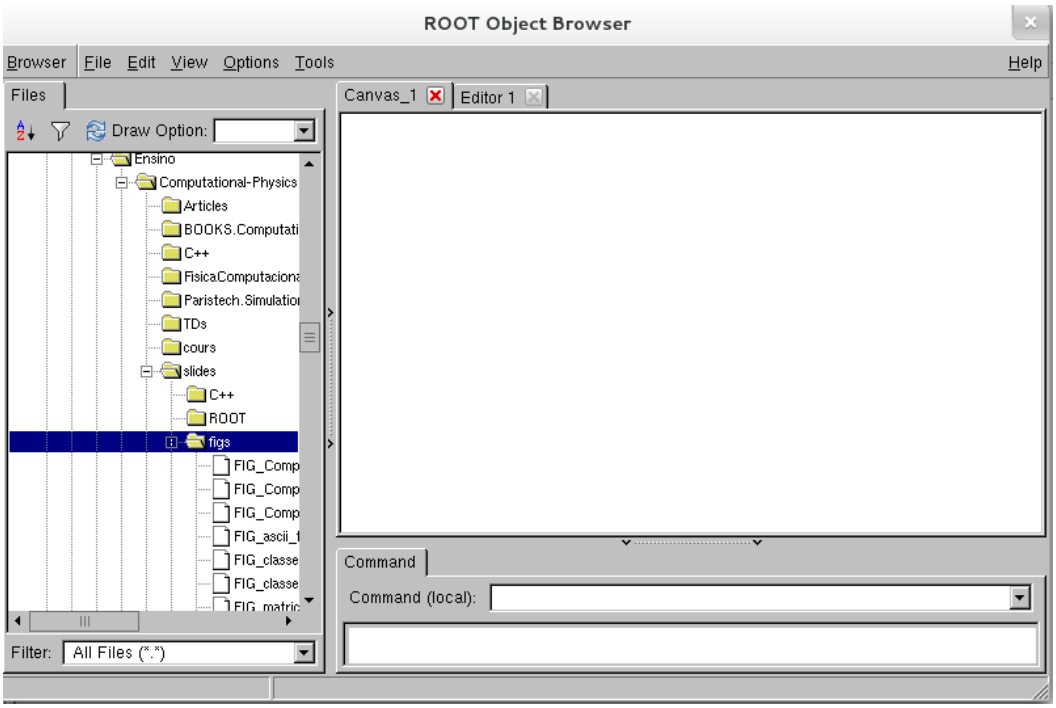
```
[0] gStyle->SetOptTitle(0); //no title
// define gaussian function
[1] TF1 *f1 = new TF1("f1","gaus()",0.,12.);
[2] f1->SetParameters(1.,5.,1.2); //set gaussian params

// histogram to store randoms

[3] TH1F *h = new TH1F("h", "histogram", 100, 0., 12.);
[4] for (int i=0; i<10000; i++) {h->Fill(f1->GetRandom());}

// cosmetics

[5] h->GetXaxis()->SetRangeUser(1.,9.);
[6] h->SetLineWidth(4);
[7] h->SetLineColor(9);
[8] h->Draw();
[9] h->Fit("f1");

// retrieve function used on fit and plot

[10] TF1 *fg = h->GetFunction("f1");
[11] fg->SetLineWidth(4);
[12] fg->SetLineStyle(2);
[13] fg->DrawCopy("same"); //superimpose plots
```



ROOT - browser

Your directories and ROOT files (root objects) can be browsed by instantiating the *TBrowser* class

```
[0] TBrowser *b = new TBrowser()
```



ROOT - init

Reset all ROOT parameters before running any C++ macro and define the graphics options

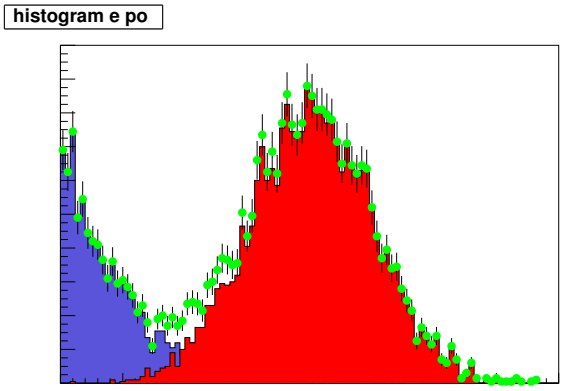
```
[0] gROOT->Reset();
[1] gROOT->SetStyle("Plain");
[2] gStyle->SetOptStat(1111); // =0 to reset
[3] gStyle->SetOptTitle(0); // supress title box
[4] gStyle->SetOptFit(1111); // print fit results
[5] gStyle->SetPalette1); // better than default
```

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 loadded 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
```