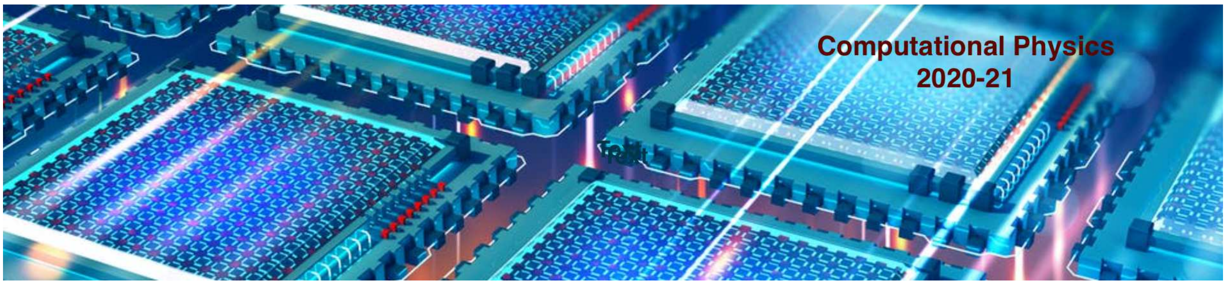
Computational Physics

numerical methods with C++ (and UNIX)

2020-21



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: fernando.barao@tecnico.ulisboa.pt

Computational Physics

ROOT

A data analysis graphics tool with a C++ interpreter

Fernando Barao, Phys Department IST (Lisbon)

ROOT - outline

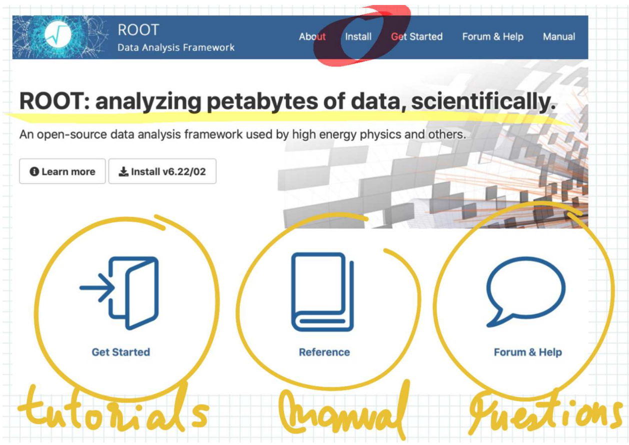
- ✔ ROOT installation
- ✔ general concepts
- ✔ interactive use and macros
- ✔ canvas and graphics style
- ✔ histograms and other objects
- ✔ fitting
- ✔ input/ouput
- ✔ using ROOT from user programs

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

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

ROOT - introduction

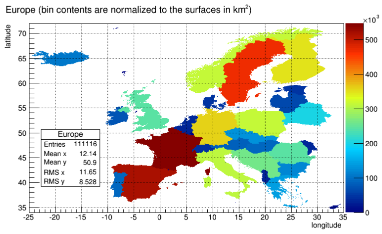
- ✔ ROOT is and object oriented framework designed for solving data handling issues in High Energy Physics such as data storage and data analysis (display, statistics, ...)
- ✔ ROOT was the next step after the PAW data analysis tool developped in Fortran on $90's$ and widely used by physicists
- ✔ ROOT is supported by the CERN organization and it is continuously evolving
- ✔ ROOT is nowadays used in other fields like medicine, finance, astrophysics, ... as a data handling tool

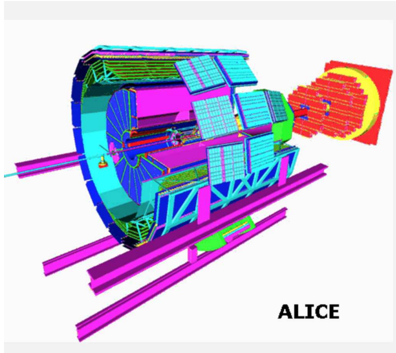


ROOT - categories

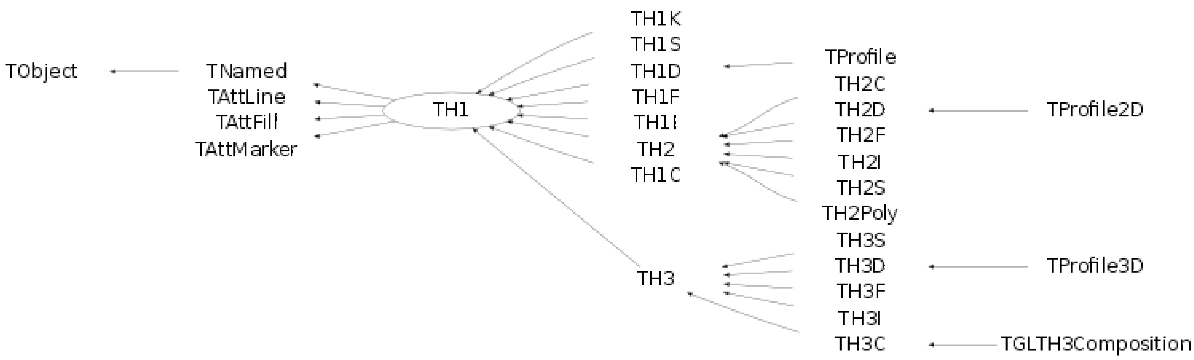
many fields/categories covered:

- ✔ base: low level building blocks (TObject,...)
- ✔ container: arrays, lists, trees, maps, ...
- ✔ physics: 2D-vectors, 3D-vectors. Lorentz vector, Lorentz Rotation, N-body phase space generator
- ✔ matrix: general matrices and vectors
- ✔ histograms: 1D,2D and 3D histograms
- ✔ minimization: MINUIT interface,...
- ✔ tree and ntuple: information storage
- ✔ 2D graphics: lines, shapes (rectangles, circles,...), pads, canvases
- ✔ 3D graphics: 3D-polylines, 3D shapes (box, cone,...)
- ✔ detector geometry: monte-carlo simulation and particle tracing
- ✔ graphics user interface (GUI):
- ✔ networking: buttons, menus,...
- ✔ database: MySQL,...
- ✔ documentation





ROOT: TH1 class inheritance



TObject: Mother of all ROOT objects.

The TObject class provides default behaviour and protocol for all objects in the ROOT system. It provides protocol for object I/O, error handling, sorting, inspection, printing, drawing, etc. Every object which inherits from TObject can be stored in the ROOT collection classes.

```
> root -l
 // allocate an array of pointers to TObject
 TObject **OBJ = new TObject*[100]
 // allocate objects
 OBJ[0] = new TH1D()
 OBJ[1] = new TF1()
 OBJ[2] = new TList()
 OBJ[3] = new TMatrixD()
 OBJ[4] = new TCanvas()
 // list objects in memory
 .ls
 // use TBrowser instead of a listing
 new TBrowser()
 // quit
 .q
```

ROOT: start

✔ **root command help**

```
> root --help # get help

Usage: root [-l] [-b] [-n] [-q] [dir] [[file:]data.root] [file1.C ... fileN.C]
Options:
  -b : run in batch mode without graphics
  -n : do not execute logon and logoff macros as specified in .rootrc
  -q : exit after processing command line macro files
  -l : do not show splash screen
  -x : exit on exception
 dir : if dir is a valid directory cd to it before executing

  -?       : print usage
  -h       : print usage
  --help   : print usage
  -config  : print ./configure options
  -memstat : run with memory usage monitoring (example: root -l -q -memstat macro.C)
```

✔ **start root**

```
> root -l
```

✔ **quit root**

```
> .q
```


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->SetPalette(1); // better than default
```

ROOT: CINT/CLINT interpreter

✔ CINT commands

```
root> .<command>
                .q : quit
                .? : list of commands
      .x <macro.C> : execute C++ macro
      .L <macro.C> : load macro (interpreted), .L <macro.C>++ (compiled)
     .!<shell cmd> : run shell cmd
                     .!ls = list files on current directory
                     .!pwd = print current directory name
.class <classname> : quick help and reference (for instance: .class TH1F)
```

✔ ROOT global pointers

https://root.cern.ch/root/html534/guides/users-guide/CINT.html

- **gROOT** instance of the *TROOT class* works as an entry point to the ROOT system, providing access to the stored ROOT objects
- **gSystem** defines an interface to the underlying operating system (*TUnixSystem*)
- **gStyle** defines attributes of objects: lines, canvas, pad, histograms,...
- **gRandom** instance of *TRandom3 class* providing a quick access to random number generator

ROOT: CINT/CLINT interpreter

my first macro with ROOT: *mvec.C*

```cpp
void mvec() { // same name as filename!!!
  std::vector<int> v(100,0); // fill 0's
  // make vector from 0,...,99
  std::iota(v.begin(), v.end(),0);
  // print
  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(cout, " "));
  cout << '\n';
}
```

running ROOT

```
 ------------------------------------------------------------------
| Welcome to ROOT 6.22/01                        https://root.cern |
| (c) 1995-2020, The ROOT Team; conception: R. Brun, F. Rademakers |
| Built for macosx64 on Jul 27 2020, 20:20:00                      |
| From heads/v6-22-00-patches@v6-22-00-61-g393f8a213a              |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
 ------------------------------------------------------------------

root [0] .L mvec.C
root [1] mvec()
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 9
root [2]
```

ROOT: automatic compiler (ACLIC)

compiling *mvec.C* using automatic compiler of ROOT (ACLIC)

```
root -l mvec.C++
root [0]
Processing mvec.C++...
Info in <TMacOSXSystem::ACLiC>: creating shared library
                                /usr/local/cern/FC/MY.FB/aulas.C++_examples/ROOT/./mvec_
In file included from input_line_12:6:
././mvec.C:5:8: error: no member named 'iota' in namespace 'std'
  std::iota(v.begin(), v.end(),0);
  ~~~~~^
././mvec.C:7:60: error: use of undeclared identifier 'cout'
  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(cout, " "));
                                                           ^
././mvec.C:8:3: error: use of undeclared identifier 'cout'
  cout << '\n';
  ^
  Error in <ACLiC>: Executing '/usr/local/cern/root_6.22.00/bin/rootcling -v0
  "--lib-list-prefix=/usr/local/cern/FC/MY.FB/aulas.C++_examples/ROOT/mvec_C_ACLiC_map"
  -I"/usr/local/cern/root_6.22.00/include"
  (...)
```

Notice that includes have to be provided (like in 'normal' compiled code!!!)

ROOT: automatic compiler (ACLIC)

compiling *mvec2.C* using automatic compiler of ROOT (ACLIC)

```
/ to be compiled
// root -l mvec2.C++ (or, .L mvec2.C++)
#include <vector>
#include <iostream>
#include <algorithm> // copy
#include <numeric> // iota

void mvec2() { // same name as filename!!!
  std::vector<int> v(100,0); // fill 0's
  // make vector from 0,...,99
  std::iota(v.begin(), v.end(),0);
  // print
  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(cout, " "));
  cout << '\n';
}
```

running ROOT

```
root [0] .L mvec2.C++
Info in <TMacOSXSystem::ACLiC>: creating shared library /usr/local/cern/FC/MY.FB/aulas.C++_examples/ROOT/./mvec2_C.so
root [1] mvec2()
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 4
5 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 8
7 88 89 90 91 92 93 94 95 96 97 98 99
```

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 on heap
root [2] gDirectory->ls("-m") // list objects
root [3] gROOT->Reset() // removes stack objects
root [4] delete h; // deletes object from heap
```

Track memory leak within CINT

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

Note: check your in your *.rootrc* file if MemStat is 1!

```
root [0] gObjectTable->Print() //Display list of objects known in current RO
root [1] .x hadd.C
root [2] gObjectTable->Print()
root [3] .g // Displays all objects defined within root session
root [4] gEnv->Print() //Display the settings of the current ROOT environmen
```


ROOT - calculator

□ **ROOT used as a calculator**

```
> root -l
root [0] 7+2/6 //do not put ";" at the end to get answer
(const int)7
root [1] 7+2/6.
(const double)7.33333333333333304e+00
root [2] 1>2 //evaluate expression
(const int)0
root [3] TMath::Pi()
(Double_t)3.14159265358979312e+00
root [4] TMath::Sin(10.*TMath::Pi()/180.) //compute sin(10 degrees)
(Double_t)1.73648177666930331e-01

root [18] double result = 0.
(const double)0.00000000000000000e+00
root [19] for (int i=0;i<10;i++) {result += TMath::Power(0.5,i);}
root [20] result
(double)1.99804687500000000e+00
```

ROOT: conventions and Types

Conventions

- ✔ class names begin with **T**
  TH1F, TF1, TTree, ...
- ✔ non-class types end with _t
  Int_t, ...
- ✔ data members begin with **f**
  fEntries, ...
- ✔ constants begin with **k**
  kRed, kTrue, ...
- ✔ global variables begin with **g**
  gRandom, gSystem, ...

data types

To prevent problems with different variables sizes on different hardwares, ROOT defines machine indenependent types

- ✔ **Char_t**: signed character (1 byte)
  UChar_t: signed character (1 byte)
- ✔ **Short_t**: signed short integer (2 bytes)
- ✔ **Int_t**: signed integer (4 bytes)
  UInt_t: unsigned integer (4 bytes)
- ✔ **Long64_t**: signed long integer (8 bytes)
  ULong64_t: unsigned long integer (8 bytes)
- ✔ **Float_t**: float (4 bytes)
  Double_t: double (8 bytes)
- ✔ **Bool_t**: boolean

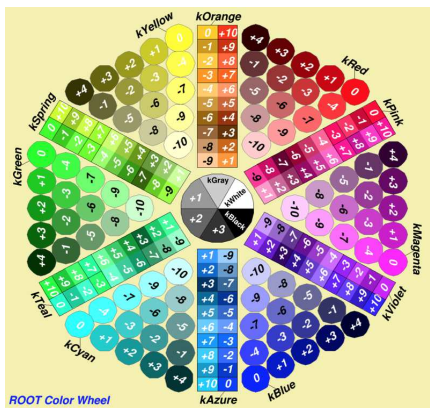
ROOT: colors

Basic colors

The following image displays the 50 basic colors.

```
{
   TCanvas *c = new TCanvas("c","Fill Area colors",0,0,500,200);
   c->DrawColorTable();
   return c;
}
```


```
myObject.SetFillColor(kRed);
myObject.SetFillColor(kYellow-10);
myLine.SetLineColor(kMagenta+2);
```

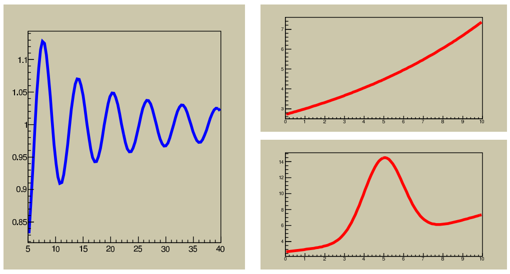


ROOT: canvas window

- The graphics window in ROOT is made using the *TCanvas class*
- Let's open a canvas and divide it in three *pads* where the graphics objects will be drawn

```
[0] gROOT->Reset();
[1] gStyle->SetOptTitle(0);
[2] TF1 *f1 = new TF1("f1","1.+ [0]*sin([1]*x)/x + [2]*exp(-x)",0.1, 40.);
[3] f1->SetParameters(1.,1.,1.);
[4] f1->SetLineColor(kBlue);
[5] f1->SetRange(5.,40.);
[6] TCanvas *c = new TCanvas("c", "Phys Comput canvas", 0, 0, 900, 500);
[7] TPad *pad1 = new TPad("pad1","The 2nd pad",0.02,0.02,0.48,0.98,21);
[8] TPad *pad2 = new TPad("pad2","The 2nd pad",0.51,0.52,0.98,0.98,21);
[9] TPad *pad3 = new TPad("pad3","The 3rd pad",0.51,0.02,0.98,0.49,21);
[10] pad1->Draw(); pad2->Draw(); pad3->Draw();
[11] pad1->cd(); f1->SetLineWidth(4); f1->DrawCopy();
//expo=exp(A+Bx)
[12] TF1 *f2 = new TF1("f2","expo(0)",0.,10.);
[13] f2->SetParameters(1., 0.1);
[14] pad2->cd(); f2->SetLineWidth(4); f2->Draw();
//exp+gaus
[15] TF1 *f3 = new TF1("f3","expo(0)+gaus(2)",0.,10.);
[16] f3->SetParameters(1., 0.1, 10., 5., 1.);
[17] pad3->cd(); f3->SetLineWidth(4); f3->Draw();
[18] c->Modified();
```

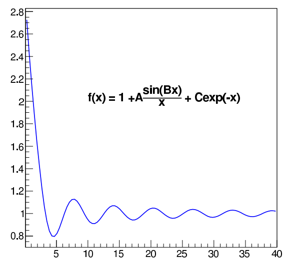
root.cern.ch/root/html/TFormula.html



ROOT: function plotter

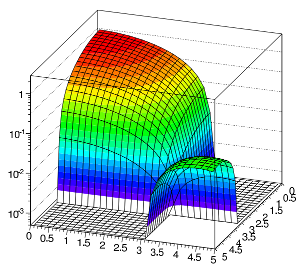
- ROOT can be used to plot functions: classes TF1, TF2
- Plot function: $f_1(x) = A\frac{sin(Bx)}{x} + Ce^x$

```
[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");
```



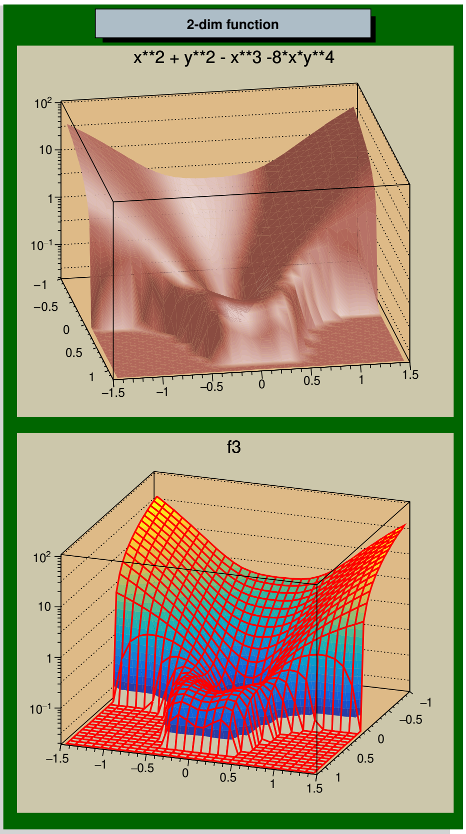
- Plot function: $f_2(x,y) = \frac{sin(x)\cdot sin(y)}{x\cdot y}$

```
[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: 2-dim function

```
TCanvas *c1 = new TCanvas("c1","2-dim function",10,10,500,900);
c1->SetFillColor(kGreen+3);
gStyle->SetFrameFillColor(42);
title = new TPaveText(.2,0.96,.8,.995);
title->SetFillColor(33); title->AddText("2-dim function");
title->Draw();

TPad *pad1 = new TPad("pad1","",0.03,0.50,0.98,0.95,21);
TPad *pad2 = new TPad("pad2","",0.03,0.02,0.98,0.48,21);
pad1->Draw(); pad2->Draw();

TF2 *f2 = new TF2("f2", "x**2 + y**2 - x**3 -8*x*y**4",
                  -1,1.2,-1.5,1.5);
f2->SetContour(48); f2->SetFillColor(45);

pad1->cd(); pad1->SetPhi(-80);
pad1->SetLogz(); f2->Draw("surf4");

TF2 *f3 = new TF2("f3", []( Double_t *x, Double_t *par) {
  return pow(x[0], 2.) + pow(x[1], 2.) - pow(x[0], 3.) -
  8*x[0]*pow(x[1],4.); }, -1,1.2,-1.5,1.5);

pad2->cd(); pad2->SetTheta(25); pad2->SetPhi(-110);
pad2->SetLogz(); f3->Draw("surf1"); c1->SaveAs("fig.pdf");
```



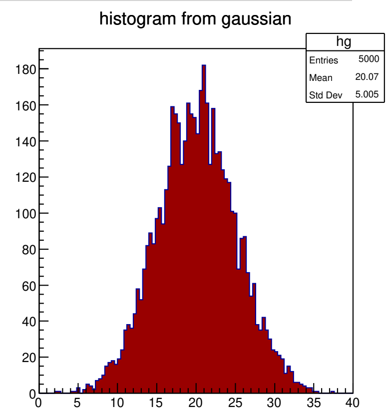
ROOT: random numbers

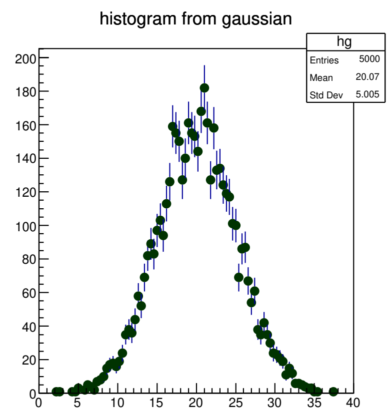
✔ random numbers can be generated with ROOT classes: TRandom3, ...

distributions easy to generate

```
- Exp(tau)
- Gaus(mean, sigma)
- Rndm()
- Uniform(x1,x2)
- Landau(mpv, sigma)
- Poisson(mean)
- Binomial(ntot, prob)
```

```
TRandom3 *random = new TRandom3(time());
TH1F *hg = new TH1F("hg", "title", 50, 0., 50.);
for (int i=0; i<1000; ++i) {
  Double_t x = random->Gaus(20.,5);
  hg->Fill(x);
}
hg->SetFillColor(kRed+2);
TCanvas *c = new TCanvas("c", "c", 0, 0, 400, 700);
c->Divide(1,2);
c->cd(1);
hg->Draw();
c->cd(2);
hg->SetMarkerStyle(20);
hg->SetMarkerColor(kGreen+4);
hg->Draw("EP");
// clone histogram
TH1F *hnew = (TH1F*)hg->Clone();
hnew->SetName("hnew");
```

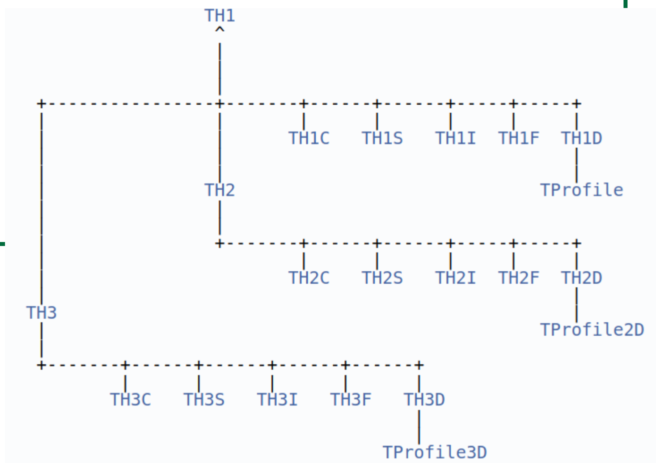




ROOT - the histogram class

TH1.h

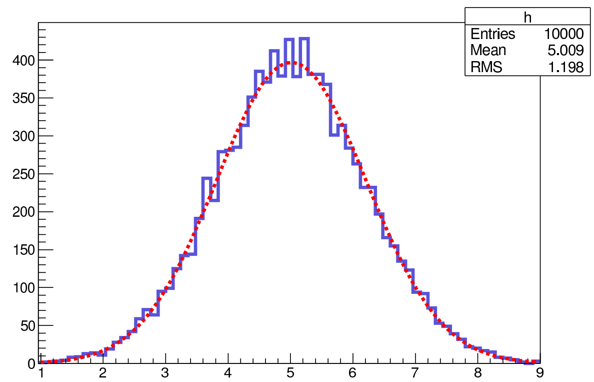
```
class TH1F : public TH1, public TArrayF {

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

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

// histogram to store randoms

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

// cosmetics

 h->GetXaxis()->SetRangeUser(1.,9.);
 h->SetLineWidth(4);
 h->SetLineColor(9);
 h->Draw();
 h->Fit("f1");

// retrieve function used on fit and plot

 TF1 *fg = h->GetFunction("f1");
 fg->SetLineWidth(4);
 fg->SetLineStyle(2);
 fg->DrawCopy("same"); //superimpose plots
```



ROOT: graphs

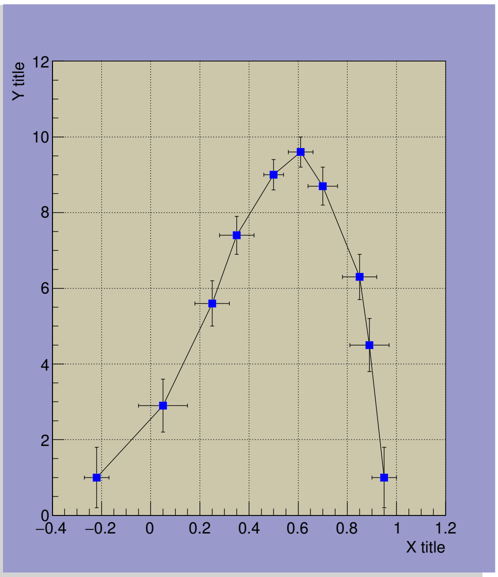
- ✔ ROOT provides several classes to display data (unbinned data) $y(x)$
- ✔ Data fitting and interpolation is also available
- ✔ **TGraph**: data points with no errors
- ✔ **TGraphErrors**: data points with symmetric errors
- ✔ **TGraphAsymmErrors**: data points with assymmetric errors
- ✔ **TGraphBentErrors**: data points with assymmetric errors
- ✔ **TMultiGraph**: several graphs added to plot

useful methods

```
- TGraph::Print()
- TGraph::Eval(Double_t)
```


ROOT: graphs

```
void graphs2() {
  TCanvas *c1 = new TCanvas("c1","gerrors2",
  20,10,400,600);
  c1->SetFillColor(kBlue-8);
  c1->SetGrid();

  // draw a frame to define the range
  TH1F *hr = c1->DrawFrame(-0.4,0,1.2,12);
  hr->SetXTitle("X title");
  hr->SetYTitle("Y title");
  c1->GetFrame()->SetFillColor(21);
  c1->GetFrame()->SetBorderSize(12);

  // create first graph
  vector<Double_t> x1{-0.22, 0.05, 0.25, 0.35,
                          0.5, 0.61,0.7,0.85,0.89,0.95};
  vector<Double_t> y1{1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};
  vector<Double_t> ex1{.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
  vector<Double_t> ey1{.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};
  TGraphErrors *gr1 = new TGraphErrors(x1.size(),x1.data()
                      ,y1.data(),ex1.data(),ey1.data());
  gr1->SetMarkerColor(kBlue);
  gr1->SetMarkerStyle(21);
  gr1->Draw("LP");
}
```



ROOT: Input/Output

- ROOT objects can be saved and retrieved from memory using TFile object
- saving ROOT objects

```
// open file
TFile *fout = new TFile("output.root", "recreate");
// list contents of the file
fout->ls();
// write histogram to file
histogram->Write();
// close file
fout->Close();
```

- reading ROOT objects

```
// open file
TFile *fin = new TFile("input.root");
// list contents of the file
fout->ls();
// read histogram from file
TObject* obj = fin->Get("object_name");
// we need to cast it to correct object
TH1F *h = (TH1F*)obj;
// close file
fin->Close();
```


ROOT classes 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 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".

Computational Physics

ROOT

A data analysis graphics tool with a C++ interpreter

Fernando Barao, Phys Department IST (Lisbon)

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 on heap
root [2] gDirectory->ls("-m") // list objects
root [3] gROOT->Reset() // removes stack objects
root [4] delete h; // deletes object from heap
```

Track memory leak within CINT

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

Note: check your in your *.rootrc* file if MemStat is 1!

```
root [0] gObjectTable->Print() //Display list of objects known in current RO
root [1] .x hadd.C
root [2] gObjectTable->Print()
root [3] .g // Displays all objects defined within root session
root [4] gEnv->Print() //Display the settings of the current ROOT environmen
```

77-2

Computational Physics

ROOT

A data analysis graphics tool with a C++ interpreter

Fernando Barao, Phys Department IST (Lisbon)

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->SetPalette(1); // better than default
```


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 on heap
root [2] gDirectory->ls("-m") // list objects
root [3] gROOT->Reset() // removes stack objects
root [4] delete h; // deletes object from heap
```

Track memory leak within CINT

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

Note: check your in your *.rootrc* file if MemStat is 1!

```
root [0] gObjectTable->Print() //Display list of objects known in current RO
root [1] .x hadd.C
root [2] gObjectTable->Print()
root [3] .g // Displays all objects defined within root session
root [4] gEnv->Print() //Display the settings of the current ROOT environmen
```