COMPUTATIONAL PHYSICS

The ROOT library

Introduction

Calculator

Functions

Histograms

Embedding ROOT in user programs

ROOT – Introduction

- ROOT is an object-oriented program and library developed by CERN.
- ROOT is supported by CERN and is continuously evolving.
- Originally designed for particle physics data analysis and contains several features specific to this field.
- However, as a data handling tool or graphics user interface (GUI) it can be used almost everywhere...
- A tremendous power (and fast !) → it is written in ***C++*** and can be easily embedded/linked in ***ANY*** C++ source code/project !
  - *Embedded C++ interpreter CLING (ROOT6)....a command line to play !*
- Fully cross platform: UNIX/Linux, MacOS, Windows

ROOT – Resources

- **ROOT Resources Main ROOT page**

– http://root.cern.ch

- **Class Reference Guide**

– https://root.cern.ch/guides/reference-guide

- **C++ tutorial** *(useful if you already know C++…)*

– http://www-root.fnal.gov/root/CPlusPlus/index.html

- **Hands-on tutorials:**

– https://root.cern.ch/courses
– https://www.youtube.com/watch?v=s9PTrWOnDy8

ROOT – Typical functionalities

- 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 (*e.g. minimizing chi2 in curve fitting…*)
- tree and ntuple: information storage
- 2D graphics: lines, shapes (rectangles, circles,...), pads, canvases
- 3D graphics: 3D-polylines, 3D shapes (box, cone,...)
- Monte Carlo: monte-carlo simulation and particle tracing
- graphics user interface (GUI): develop standalone applications…

ROOT – How-to…

- Help on the way on the linux shell… `>$root --help`

```
MyMacBook-267:my_ROOT_scripts rcoelho$ root --help

Usage: root [-l] [-b] [-n] [-q] [dir] [[file:]data.root] [file1.C ... fileN.C]
Options:
  -b : run in batch mode without graphics
  -x : exit on exception
  -e expression: request execution of the given C++ expression
  -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
  -t : enable thread-safety and implicit multi-threading (IMT)
 --web: display graphics in a web browser
 --notebook : execute ROOT notebook
 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
```

- To run the ROOT CLING command line interface : `>$root`
- To quit the command line : `root [0] .q`

ROOT – CLING interpreter

- Inside ROOT command line interface (CLI), some useful commands exist

```
MyMacBook-267:my_ROOT_scripts rcoelho$ root -l

root> .?                       List of cling executive commands available e.g. root>.q
root> .q                       Quit ROOT
root> .x ROOT_macro.C          Execute a ROOT macro file....(later um macro files...)
root> .x ROOT_macro.C(3)       Execute a ROOT macro file with parameter
root> .L ROOT_macro.C          Load a ROOT macro file...
root> .! <shell_command>       Execute a UNIX shell command e.g. root>.! Ls
root> <arrow keys up and down to scroll previous commands, even from previous session !>
```

ROOT global pointers

- gROOT - instance of the *TROOT class* works as an entry point to the ROOT system, providing access to the stored ROOT objects (e.g. *Reset()*)
- 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 – calculator

- The CLI of ROOT can easily be used as a (non-basic) calculator

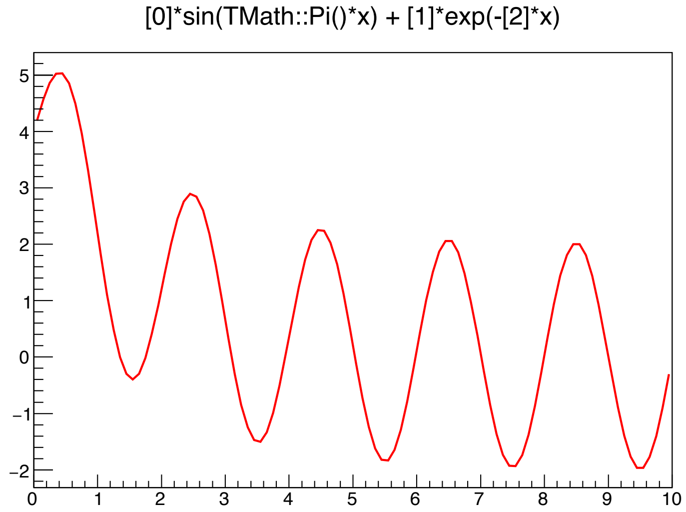
```
$ root -l

root [0] 1+3  //we don't need the ; as legal C++...
(int) 4
root [1] 5+7/4 //only using integers so...
(int) 6
root [2] 5+7/4. //using a double the ratio becomes a double...
(double) 6.7500000
root [3] 7 != 5 //of course 7 is not 5 so.....
(bool) true
root [4] TMath::Pi()  //Tmath class has some handy members....
(double) 3.1415927
root [5] TMath::Exp(1)  //The usual Neper number...
(double) 2.7182818
root [6] TMath::Sin(TMath::Pi()/2.)  //sin of 90degrees
(double) 1.0000000
root [7] double result=0.
(double) 0.0000000
root [8] for (int i=0;i<15;i++) {result+=1./TMath::Factorial(i);}  //Taylor
series for exp(1) !
root [9] printf("%.14e",result)  //print the result with 14 decimal places
scientific format
2.71828182845823e+00(int) 20  -----> 6 extra to layout the number e.g. e+00
```

ROOT – Function Plotter

- The CLI of ROOT can easily be used as a (non-basic) calculator

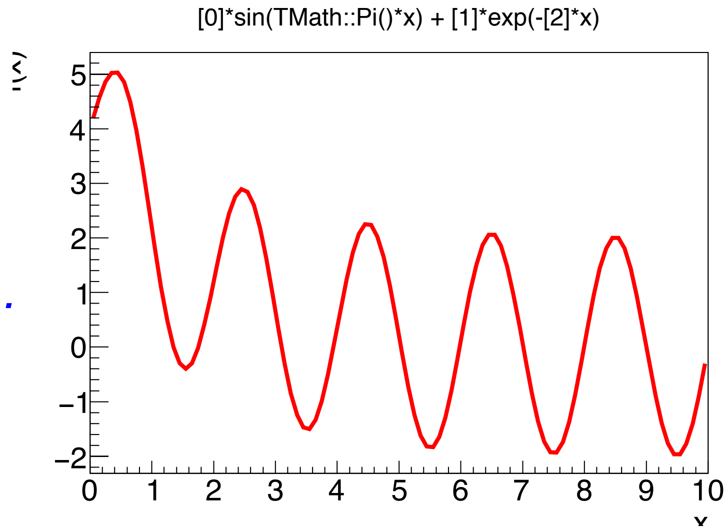
```
TF1 *f1 = new TF1("f1","sin(x)/x",0.,10.)
f1->Draw()
f2 = new TF1("f2","[0]*sin(TMath::Pi()*x) +
[1]*exp(-[2]*x)",0.,10.)
f2->SetParameters(2.,4.,0.6) //Set Parameters
[0], [1] and [2]
//f2->SetParameter(0,2.);f2-
>SetParameter(1,4.);f2->SetParameter(2,0.6);
f2->Draw()
```



- A bit ugly...let’s improve...

```
TF1 *f3=f2
f3->SetLineWidth(4)
f3->GetXaxis()->SetTitle("x”)
f3->GetXaxis()->SetTitleSize(0.06)
f3->GetXaxis()->SetLabelSize(0.06);
f3->GetYaxis()->SetTitle("f(x)”)
f3->GetYaxis()->SetTitleSize(0.06)
f3->GetYaxis()->SetLabelSize(0.06)
f3->Draw()
```

…iaics…



ROOT – Graphics window

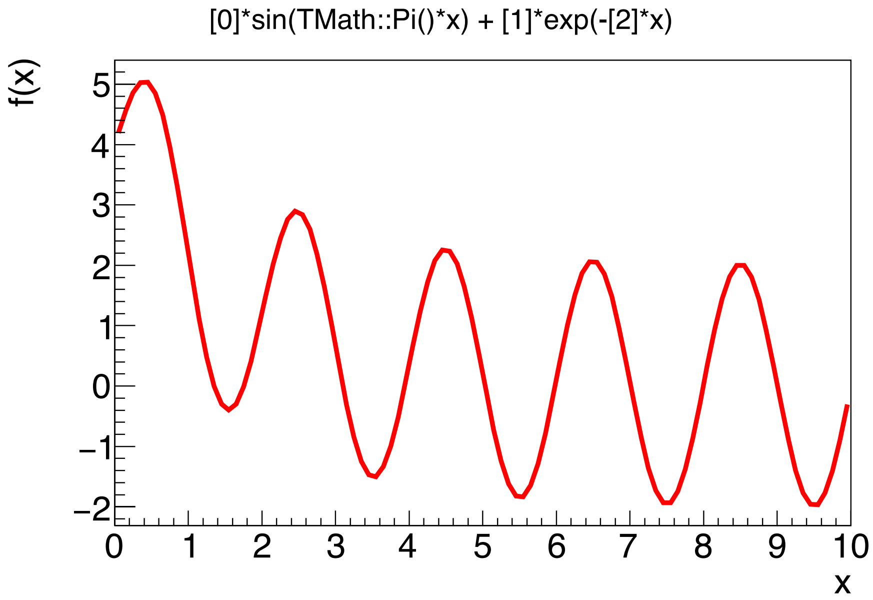
- We can define the graphics window properties beforehand...

```
TCanvas *c1 = new TCanvas("c1", "c1",
10,45,700,500);
TPad *pad1 = new TPad("pad1","1st pad",
0.00,0.00,1.00,1.00,0);
TF1 * f3 = new
TF1("f3","[0]*sin(TMath::Pi()*x) + [1]*exp(-
[2]*x)",0.,10.)
f3->SetParameters(2.,4.,0.6)
f3->SetLineWidth(4);
f3->GetXaxis()->SetTitle("x");
f3->GetXaxis()->SetTitleSize(0.06)
f3->GetXaxis()->SetLabelSize(0.06);
f3->GetYaxis()->SetTitle("f(x)");
f3->GetYaxis()->SetTitleSize(0.06)
f3->GetYaxis()->SetLabelSize(0.06);

pad1->SetLeftMargin(0.133);
pad1->SetBottomMargin(0.1326316);
pad1->SetRightMargin(0.04);
pad1->Draw();pad1->cd();
f3->Draw()

c1->Modified();
c1->cd();c1->SetSelected(c1);
```

background color



→ Tpad is handy since we can easily Plot more than one plot per "*page*" !

→ Each Pad can have different dimensions and/or margins or even background color !

→ Tcanvas derived from Tpad.

ROOT – 2D Plots

- Also 2D Plots can be drawn

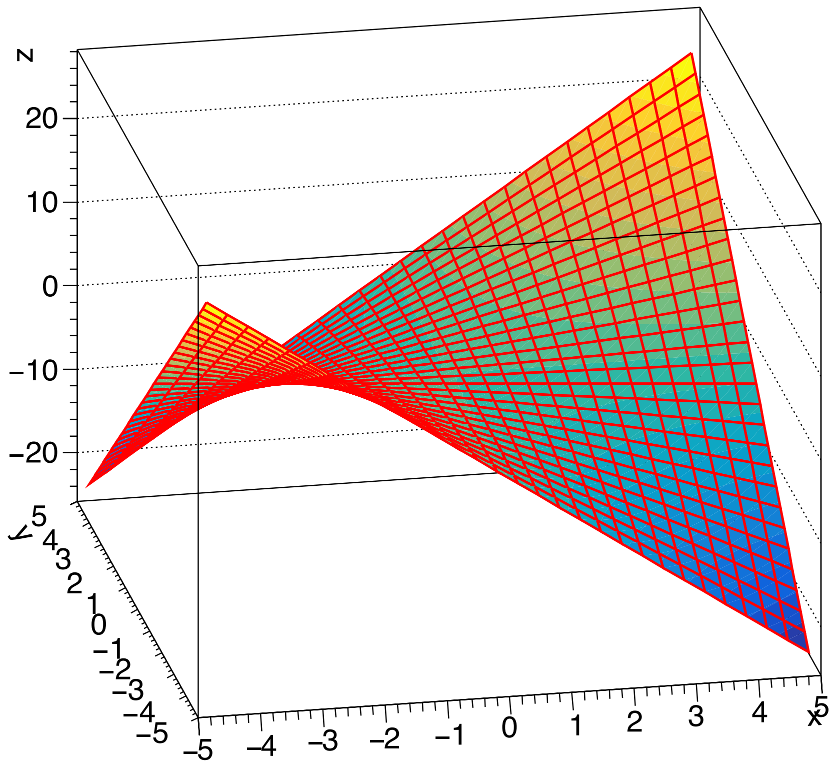
```
gStyle->SetOptTitle(0);
TCanvas *c = new TCanvas("c", "Phys Comput
canvas", 0, 0, 500, 500);
TF1 *f2 = new TF2("f2","x*y",-5,5,-5,5);

f2->GetXaxis()->SetTitle("x");
f2->GetYaxis()->SetTitle("y");
f2->GetZaxis()->SetTitle("z");

c->SetTheta(26); //angle with hor.plane
c->SetPhi(11); //rotation around z-axis
f2->Draw("surf1");

c->Modified();
c->SaveAs("2Dfunction_surfplot.eps");
```

$$f(x,y) = x.y$$



→ SetOptTitle(0) removes the title from plot canvas

→ SaveAs(*filename*) saves the Canvas on typical graphics filename

ROOT – Histograms *(class sneak peek)*

- Another fundamental ROOT class...with many different constructors !

#bins, first value, last value

```
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);
     TH1F& operator=(const TH1F &h1);
     virtual ~TH1F();
```

- And methods inherited from base TH1...namely to fill the Histogram !

```
//////////////////////////////////////////
/// Increment bin with abscissa X by 1.
…
/// The function returns the corresponding bin number which has its content incremented by 1
Int_t TH1::Fill(Double_t x)
```

ROOT – Histograms *(class sneak peek)*

- Another fundamental ROOT class...with many different constructors !

#bins, irregularly spaced bins (floats)

```
class TH1F : public TH1, public TArrayF {
public:
     TH1F();
     TH1F(const char *name,const char *title,Int_t nbins,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);
     TH1F& operator=(const TH1F &h1);
     virtual ~TH1F();
```

- And methods inherited from base TH1...namely to fill the Histogram !

```
//////////////////////////////////////////
/// Increment bin with abscissa X by 1.
…
/// The function returns the corresponding bin number which has its content incremented by 1
Int_t TH1::Fill(Double_t x)
```

ROOT – Histograms *(class sneak peek)*

- Another fundamental ROOT class...with many different constructors !

#bins, irregularly spaced bins (double)

```
class TH1F : public TH1, public TArrayF {
public:
     TH1F();
     TH1F(const char *name,const char *title,Int_t nbi    uble_t xlow,Double_t xup);
     TH1F(const char *name,const char *title,Int_t nbins   onst 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);
     TH1F& operator=(const TH1F &h1);
     virtual ~TH1F();
```

- And methods inherited from base TH1...namely to fill the Histogram !

```
//////////////////////////////////////////
/// Increment bin with abscissa X by 1.
…
/// The function returns the corresponding bin number which has its content incremented by 1
Int_t TH1::Fill(Double_t x)
```

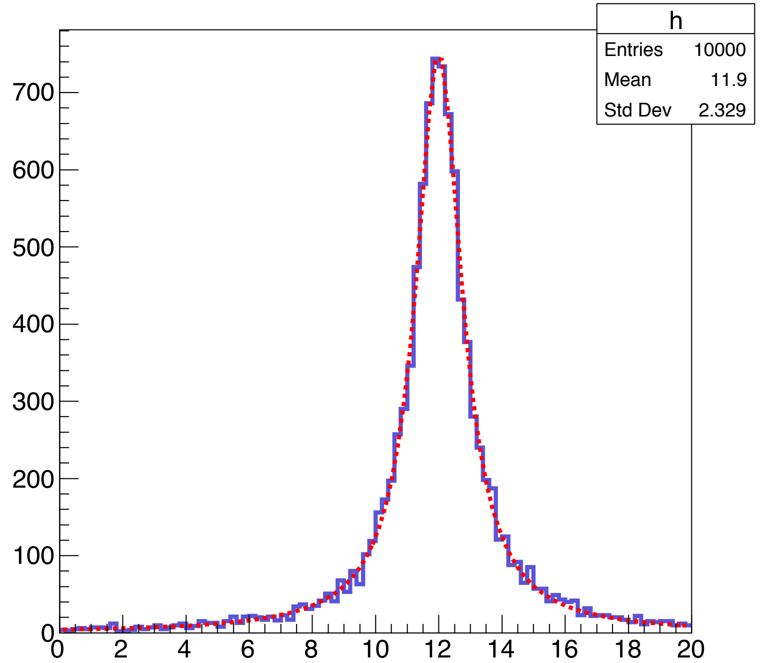
ROOT – Histograms

- Set the range of the histogram (#bins, first/last bin value) and fill it with some random numbers following some statistics (*function given !*)

```
gStyle->SetOptTitle(0); //no title
TCanvas *c = new TCanvas("c", ”my canvas", 0, 0, 500, 500);
TF1 *f1 = new TF1("f1","[0]/( 1.+ [1]*(x-[2])*(x-[2]) )",0.,20.); //f1 defined in [0,20]
f1->SetParameters(1.,1.2,12.); //set function params
TH1F *h = new TH1F("h", "histogram", 100, 0., 20.); //I want 100 bins between 0 and 20…
for (int i=0; i<10000; i++) {
          h->Fill(f1->GetRandom()); //draw a random in [0,20] using f1 function statistics
}
// cosmetics
h->GetXaxis()->SetRangeUser(0.,20.);
h->SetLineWidth(4); h->SetLineColor(9);
h->Draw();

f1->SetParameters(700.,1.2,12.);//fit guess params
h->Fit("f1"); //Double_t * f1->GetParameters()
h->SetStats(true); //false to remove the Stats Box…

// retrieve function used on fit and plot
TF1 *fg = h->GetFunction("f1");
fg->SetLineWidth(4);
fg->SetLineStyle(2);
c->Modified();
c->SaveAs("General_histogram_fit.eps");
```



ROOT – Macros in ROOT

- A macro is a lightweight program to execute a sequence of valid ROOT commands.

```
void test(int np){
gStyle->SetOptTitle(0); //no title
TCanvas *c = new TCanvas("c", ”my canvas", 0, 0, 500, 500);
TF1 *f1 = new TF1("f1",”gaus(0)”,-2.,2.); //gaus(0) – gaussian (3 param, 1st index=0)
f1->SetParameters(1.,0.5,0.75); //Ampl=1, center=0.5, width=0.75
TH1F *h = new TH1F("h", "histogram", 1000, -2., 2.); //1000 even bins between -2 and 2
for (int i=0; i<np; i++) {
          h->Fill(f1->GetRandom());
}
cout << h->GetMean() << endl; //do you expect to get 0.5 ?
h->Draw();
c->Modified();
c->SaveAs(”Uniform_histogram_user_input.eps");
}
//ROOT has built-in shortcut functions e.g. gaus(0), expo(0) for exp([0]+[1]*x) besides
the usual mathematical ones: sqrt(x), pow(x,N), pi, cosh(x), log(x),...
```

- To run simply type on the CLI: `$ root –l test.C(100000)`

ROOT – Embedding in user programs

- ROOT has dedicated compilation flags and libraries to be included/ linked !

```
#ifndef Function_H
#define Function_H
#include "TF1.h"
class Function {
  TF1 * f1;
public:
  Function(double a,double b);
  ~Function();
  void SetParam(double p1,double
p2,double p3);
  double GetRand();
  void plot();
  double MyFunc (double *x, double *p );
};
#endif
```

```
#include "TF1.h"
#include "TCanvas.h"
#include "Function.h"

double MyFuncOut (double *x, double *p ) {
 return p[0]*TMath::Gaus(x[0],p[1],p[2] );
}

Function::Function(double a,double b) {
f1 = new TF1("f1",MyFuncOut,a,b,3); //3 params}
Function::~Function() {delete f1;}
void Function::SetParam(double p1,double
p2,double p3) {
   f1->SetParameters(p1,p2,p3); //set params
}
void Function::plot() {
  TCanvas *c = new TCanvas("c", "my canvas", 0,
0, 500, 500);
  f1->Draw();
  c->Modified();
  c->SaveAs("My_function.eps");
}
```

ROOT – Embedding in user programs

```
#include <iostream>
#include "Function.h"
using namespace std;
int main() {
   Function * A=new Function(-5.,5.);
   A->SetParam(3.,2.,1.5);
   cout << "From the statistics -> " << A->GetRand() << endl;
   A->plot();
}
```

```
CC := g++
CCFLAGS := -std=c++14
ROOT_FLAGS := $(shell echo $$(root-config --cflags --libs) )
INCLUDE := ./
LD := $(CC)
################################################
SOURCES := $(shell ls . | grep cpp)
OBJECTS := $(SOURCES:.cpp=.o)

all: Main.exe
Main.exe: $(OBJECTS)
          $(LD) -o $@ $(OBJECTS) $(ROOT_FLAGS) -I $(INCLUDE)
%.o: %.cpp
          $(CC) $(CCFLAGS) -c $^ $(ROOT_FLAGS) -I $(INCLUDE)
clean:
          rm *.o *.exe
```