19th April, 2016, EHEP DST-SERC School, Delhi University
Bipul Bhuyan, IIT Guwahati

Exercise 0: Introduction to root

This tutorial should provide you with a very basic introduction to the root data analysis framework. If you have never tried root before, read the root documentation in https://root.cern.ch. To succeed as a HEP experimentalist, you must also learn RooFit, which is a toolkit for data modeling that allows for modeling probability distributions in a compact and abstract way. It is distributed with ROOT nowadays. Install RooFit along with ROOT. Instructions can be found in: http://roofit.sourceforge.net/docs/index.html.

1. Native root (C++)

Open a root session in a shell by typing:

> root

(where the > symbol stands for your prompt. This should start root. If not, check your environment. You now have the root C++ interpreter running, so you can enter almost any valid C++ code. Or just use root as a calculator

root[0] 1 + log(2)

The object in root we will use most is the histogram, so lets create one:

root[1] TH1F myhisto("myhisto","My most fabulous histogram",10,0,100)

This creates a histogram (of type TH1F), to which root will refer by its name ("myhisto") and which will be displayed with the title given. It has ten bins, the lowest value is 0 and the highest 100. Now we can fill some values into the histogram (root will return the corresponding bin number, if you do not like this, terminate the statements with a semicolon):

root[2] myhisto.Fill(42)
root[3] myhisto.Fill(3.141592)
root[4] myhisto.Fill(66)
root[5] myhisto.Fill(99)
root[6] myhisto.Fill(69)
root[7] myhisto.Fill(17.7)

Now let us draw the histogram:

root[8] myhisto.Draw()

This opens a canvas, the surface root draws on, per default, and the canvas will be named c1. In the canvas window, you can open the Editor from the view menu. It allows you to change the style of various objects on the canvas (which you can also move around with the mouse); be aware though that there is no undo function. All the

things that can be set from Editor, can also be set from the command line, e.g.

root[9] myhisto.SetLineColor(2)

for the change to become visible, you have to draw the histogram again

root[9] myhisto.Draw()

if you want the histogram to be drawn with error bars, try

root[9] myhisto.Draw("E1")

If you do not want to rewrite all the code every time you start root, you can use macro files. They end in .C and can be executed directly from root. Make it a habit to start them with a comment stating the objective of the macro, your name and email. Create a file **exercise0.C** in your exercise directory and fill it similar to the following:

```
 /* exercise0.C:
Example macro file for the tutorial
Bipul Bhuyan, bhuyan@iitg.ernet.in
*/

void fillHistogram(unsigned int nentries)
{
TH1F * histo = new TH1F("histo","Another histogram",10,0,100);
for(unsigned int i=0; i < nentries; i++)
      {
        histo->Fill(fmod(i*777,100));
       }
}
```

Note that we create the histogram using new, to make sure it does not go out of scope when the function returns. In root, we can now load the macro file using

root[10] .L exercise0.C
(all root comments start with the .).

Now we can call the function like any other:
root[11] fillHistogram(100)

let's now draw the new histogram

root[12] histo->Draw()

and we could also draw the old histogram on the same canvas

root[13] myhisto.Draw("same")

be aware that the root interpreter does not really differentiate between the . and ->

member access. Now let us save the histograms to a file:

root[14] TFile* file = new TFile("exercise0.root","RECREATE")
and now we can write the histograms and close the file

root[15] myhisto.Write()
root[16] histo->Write()
root[17] file->Close()
root[18] delete file

Now try to read the histograms back in:

root[19] TFile* fileagain = new TFile("exercise0.root","READ")
root[20] TH1F * histoagain = (TH1F*)fileagain->Get("histo");

You retrieve objects from root files using their name (usually the first parameter in the constructor) with the Get() function, which will always return a pointer to a TObject , the base class of everything in root. You have to manually cast to the type you are expecting (here a TH1F pointer). Draw your re-loaded histogram

root[21] histoagain->Draw()

you can quit root with

root[22] .q

2. Random Numbers

There are several random number generators in root, with the best being TRandom3. When using random numbers, you also need to include a "seed" value. The default is to re-seed each time the program compiles. The argument of 0 tells the random number generator to use a seed based on the machine clock. To make a random variable, use the syntax:

TRandom3 r(0); // make a random variable r, with a seed generated by the machine

It is often useful to get random numbers in a particular distribution. For instance, this will generate a Gaussian distribution with a mean of 0 and a sigma of 1.

Double_t rand;

rand=r.Gaus(0,1); // Mean 0, sigma 1

You can also make uniform distributions, and a number of other useful distributions.

Double_trand;

rand=r.Uniform(−1,1); // Uniform random number distribution between −1 and 1.

Exercise 1: Generate 10,000 random numbers for a Gaussian distribution with mean 0 and sigma 1. Save the generated events in a root file. Write a root macro to read back the root file and plot the histogram.

Exercise 2: Write a root macro with the following command lines. Execute the macro and explain the objective of this code.

```
void fit()
{
	// style setting
	gROOT->SetStyle("Plain");
	gStyle->SetOptFit(111111);
	gStyle->SetFrameBorderMode(0);
	gStyle->SetFillColor(0);
	gStyle->SetCanvasColor(0);
	gStyle->SetFrameFillColor(0);
	gStyle->SetCanvasBorderMode(0);


	// create a random number generator
	gRandom = new TRandom3();

	// create a histogram
	TH1D * hist = new TH1D("data", ";x;N", 20, 0.0, 100.0);

	// fill in the histogram
	for (int i = 0; i < 100; ++i)
		hist->Fill(gRandom->Gaus(65.0, 5.0));


	// define a fit function = gauss
	TF1 * f1 = new TF1("gauss", "[0] / sqrt(2.0 * TMath::Pi()) / [2] * exp(-(x-
[1])*(x-[1])/2./[2]/[2])", 0, 100);

	//set parameter start values (mandatory).
	f1->SetParNames("Constant","Mean","Sigma");
	f1->SetParameters(700.,hist->GetMean(),hist->GetRMS());
	f1->SetParLimits(0,  100.0, 700.0);
	f1->SetParLimits(1, 50.0,  90.0);
	f1->SetParLimits(2,  0.1,  10.0);
	f1->SetLineWidth(2);
	f1->SetLineColor(2);

	// create a canvas to draw the histogram
	TCanvas * c1= new TCanvas("c1", "fitted data",5,5,800,600);
	// perform fit
	hist->Fit("gauss");
```

```
    hist->Draw();
    hist->SaveAs("fit.eps");
}
```

20th April, 2016

Exercise 3: Discussed in the class.

Exercise 4: Discussed in the class.

Exercise 5: A simple rooFit code for fitting, plotting, toy data generation on one-dimensional PDF.

```
#ifndef __CINT__
#include "RooGlobalFunc.h"
#endif
#include "RooRealVar.h"
#include "RooDataSet.h"
#include "RooGaussian.h"
#include "TCanvas.h"
#include "RooPlot.h"
#include "TAxis.h"
using namespace RooFit ;

void rf101_basics()
{
  // S e t u p   m o d e l
  // ----------------------

  // Declare variables x,mean,sigma with associated name,
title, initial value and allowed range
  RooRealVar x("x","x",-10,10) ;
  RooRealVar mean("mean","mean of gaussian",1,-10,10) ;
  RooRealVar sigma("sigma","width of gaussian",1,0.1,10) ;

  // Build gaussian p.d.f in terms of x,mean and sigma
  RooGaussian gauss("gauss","gaussian PDF",x,mean,sigma) ;

  // Construct plot frame in 'x'
  RooPlot* xframe = x.frame(Title("Gaussian p.d.f.")) ;


  // P l o t   m o d e l   a n d   c h a n g e   p a r a m e t
e r   v a l u e s
  // ---------------------------------------------------------
------------------

  // Plot gauss in frame (i.e. in x)
  gauss.plotOn(xframe) ;

  // Change the value of sigma to 3
  sigma.setVal(3) ;

  // Plot gauss in frame (i.e. in x) and draw frame on canvas
  gauss.plotOn(xframe,LineColor(kRed)) ;
```

```
  // G e n e r a t e   e v e n t s
  // ----------------------------

  // Generate a dataset of 1000 events in x from gauss
  RooDataSet* data = gauss.generate(x,10000) ;

  // Make a second plot frame in x and draw both the
  // data and the p.d.f in the frame
  RooPlot* xframe2 = x.frame(Title("Gaussian p.d.f. with
data")) ;
  data->plotOn(xframe2) ;
  gauss.plotOn(xframe2) ;


  // F i t   m o d e l   t o   d a t a
  // ----------------------------

  // Fit pdf to data
  gauss.fitTo(*data) ;

  // Print values of mean and sigma (that now reflect fitted
values and errors)
  mean.Print() ;
  sigma.Print() ;

  // Draw all frames on a canvas
  TCanvas* c = new
TCanvas("rf101_basics","rf101_basics",800,400) ;
  c->Divide(2) ;
  c->cd(1) ; gPad->SetLeftMargin(0.15) ; xframe->GetYaxis()-
>SetTitleOffset(1.6) ; xframe->Draw() ;
  c->cd(2) ; gPad->SetLeftMargin(0.15) ; xframe2->GetYaxis()-
>SetTitleOffset(1.6) ; xframe2->Draw() ;


}
```

Exercise 5: Exercise in RooFit:

RooRealVar x("x","x",0.0,0, 10.0);
RooRealVar mean ("mean","mean", 6.5,0,10);
RooRealVar sigma("sigma","sigma",0.4,0,10);
RooRealVar nsig("nsig","nsig",90,-10,1000);
RooGaussian g("g","Generation Pdf",x,mean,sigma);
RooRealVar argpar ("argpar", “Argus shape paramerter",-1.0,-10,10);
RooRealVar cutoff ("cutoff", “Argus cutoff",10.0,0,20);
RooArgusBG a ("a", "Argus PDF", x, cutoff, argpar);
RooRealVar nbkg("nbkg","nbkg",210,-10,1000);
RooAddPdf fit ("fit", "g+a", RooArgList(g, a), RooArgList(nsig, nbkg));
RooDataSet *data = fit.generate(x,300);

```
RooPlot* xframe = x.frame(); data->plotOn(xframe);
fit->plotOn(xframe);
fit->plotOn(xframe,Components(RooArgSet(ge,a))); xframe->Draw();
```

Exercise 6: ToyMC

```
RooMCStudy toymc ("fit", fit, "x", "evh" );
toymc.generateAndFit(1000,300);
RooPlot* xfr = nsig.frame(20,160,25);

// 4 - plots
toymc.plotParamOn(xfr);
RooPlot* exfr = toymc.plotError(nsig,0,30,25);
RooPlot* pxfr = toymc.plotPull(nsig,-5,5,25);
RooPlot* xll    = toymc.plotNLL(-1200,-500,50);
```