Introduction to ROOT

Asif Saddique

National Centre for Physics (NCP), Islamabad, Pakistan

ROOT Intro, Basics and Functions
Tutorial # 1

November 26, 2014

ICTP-NCP School on LHC Physics

About these Tutorials

- Aim
  - To give you a brief intro of ROOT as a user (not as an expert)
  - To provide you a base to build your ROOT knowledge on top of it.

- Acknowlegement
  - A major portion of these lectures is being extracted from the tutorials given at CERN Summer Schools (CSC).

What is ROOT ?

ROOT is an object-oriented farmework especially designed for the high-energy physics data analysis.

Framework ?

Programming inside a framework is like a living in a city, where plumbing, electricity, telephone, and transportation are services provided by the city. In your house, you have interfaces to the services such as light switches, electrical outlets, and telephones.

In a framework

- all you have to learn is the interface, which in this analogy is the same as learning how to use a telephone.
- you have less and problem focussed code to write. For example, fitting and histogramming are implemented and ready to use and customize.

What is ROOT ?

ROOT is an object-oriented farmework especially designed for the high-energy physics data analysis.

Object-oriented ?

- Encapsulation enforces data abstraction and increases opportunity for reuse.
- Sub classing and inheritance make it possible to extend and modify objects.
- Complexity is reduced because there is little growth of the global state, the state is contained within each object, rather than scattered through the program in the form of global variables
- Objects may come and go, but the basic structure of the program remains relatively static, increases opportunity for reuse of design.

What is ROOT ?

ROOT[1] is an object-oriented farmework especially designed for the high-energy physics data analysis.

It provides

- an efficient data storage, access and query system (PetaBytes)
- advanced statistical analysis algorithms (multi dimensional histogramming, fitting, minimization and cluster finding)
- scientific visualization: 2D and 3D graphics, Postscript, PDF, LateX
- geometrical modeller
- and many other things....

The data recorded by a detector is stored in form of events, and analyses are performed on event-by-event basis.

[1] ROOT is an open source project, started in 1995.

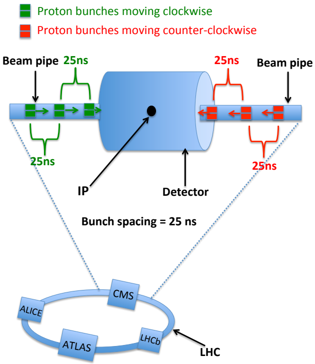
Even-by-event Analysis

The general structure of a data analysis code is following:

```
MyHeaders.h
void MyCode()
{
global variables
for(Int_t evt=0; evt<nentries; evt++){ //Event Loop starts
Main analysis goes here where you fill the data into histograms
}//Event Loop ends
Drawing and saving plots is done here
}
```

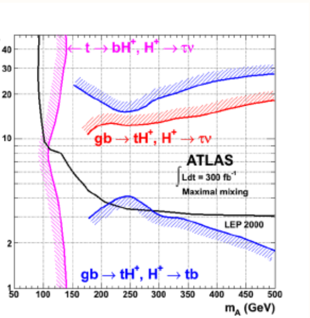
What is an event ?

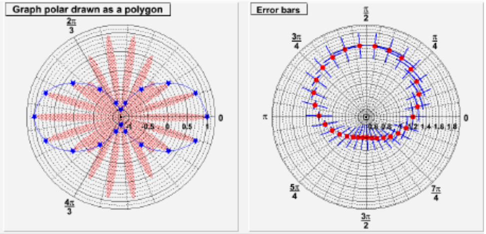
- The data recorded, which is the outcome of a bunch crossing is an event.

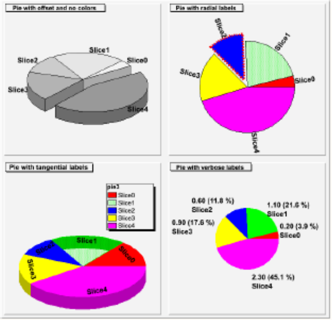


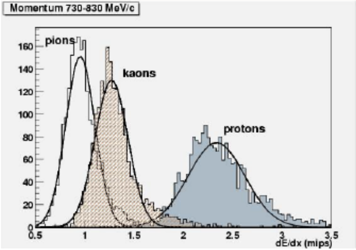
- In the example given, the detector has 25 ns to record an event, and to prepare for the next event (bunch crossing).
- After many steps, different variable are stored event-wise in the data files (*.root format).

Graphics

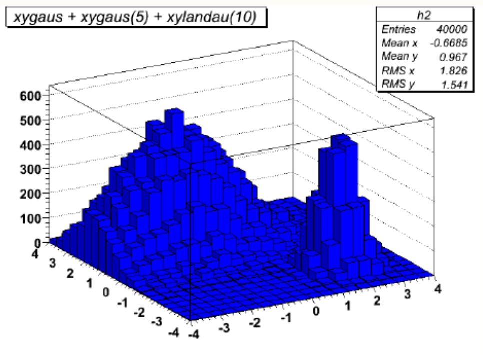

$t \to bH^+, H^+ \to \tau\nu$
$gb \to tH^+, H^+ \to \tau\nu$
ATLAS
$\int Ldt = 300\ fb^{-1}$
Maximal mixing
LEP 2000
$gb \to tH^+, H^+ \to tb$
m_A (GeV)


Graph polar drawn as a polygon
Error bars

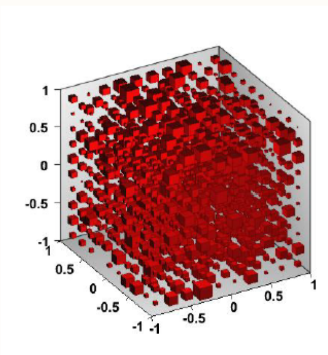

Pie with offset and no colors
Pie with radial labels
Pie with tangential labels
Pie with verbose labels
0.60 (11.8 %)
Slice2
1.10 (21.6 %)
Slice1
0.90 (17.6 %)
Slice3
0.20 (3.9 %)
Slice0
2.30 (45.1 %)
Slice4


Momentum 730-830 MeV/c
pions
kaons
protons
dE/dx (mips)

2D and 3D

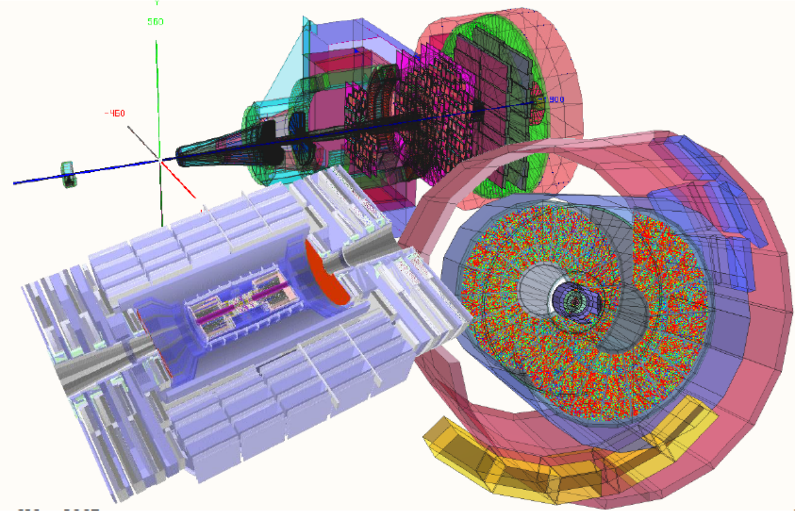


2D Histogram



3D Histogram

Geomatory



Global Variables in ROOT Framework

There are global ROOT variables that apply to the running session.

- gROOT:
  Through this pointer, you get the access to every object created in a ROOT program.
- gFile:
  It points to the current file opened in ROOT session.
- gDirectory:
  It points to the current directory.
- gPad:
  It points to the currently active pad (canvas) in the root session.
- gRandom:
  It is is a pointer to the current random number generator, which by default is based on the "Mersenne-Twister" generator.

We will learn about these variables at different places during these tutorials.

Installing ROOT

- Before installing ROOT, you need to build several libraries:
  http://root.cern.ch/drupal/content/build-prerequisites
- Download the source file for the latest ROOT version:
  http://root.cern.ch/drupal/content/downloading-root
- Unzip the source file:

```
$ gzip -dc root_ <version>.source.tar.gz | tar -xf -
```

- Build:

```
$ cd root
$ ./configure –help
$ ./configure [<arch>] // set arch appropriately if no proper default
$ (g)make // or, make -j n for n core machines
```

- Source (you are ready to run ROOT):

```
$ source PATH_TO_ROOT/bin/thisroot.sh
```

LET'S FIRE UP ROOT !

Assuming ROOT is properly installed in your system.

Running ROOT

- ROOT is prompt-based.

```
$ root
root[0] _
```

- ROOT prompt has a powerful C++ interpreter, called CINT.

```
root[0] const char* Rver = gROOT→GetVersion();
root[1] cout<<Rver<<endl;
5.34/18
root[2] for (int i = 0; i<3; i++) cout << "Hello" << i << endl
Hello 0
Hello 1
Hello 2
```

By typing C++ statements at the prompt, you can access all the available ROOT classes, create objects, global variables, functions and run your scripts.

Running a Code

- To run a script "mycode.C" in ROOT prompt[2]:

```
root[0] .x mycode.C
```

Equivalently : Load and Run

```
root[0] .L mycode.C
root[1] mycode()
```

- To compile faster:

```
root[0] .x mycode.C+
```

It checks for changes, only rebuilds if needed.

- For any help regarding CINT's commands:

```
root[2] .h
```

- Quit:

```
root[3] .q
```

[2] From terminal "$ root mycode.C"

ROOT as a calculator

ROOT can be used as a pocket calculator.

```
root[0] 1+sqrt(9)
(const double)4.00000000000000000e+00
root [1] double val=0.17;
root [2] sin(val)
(const double)1.69182349066996029e-01
```

Now let's try to build and draw a simple function by ROOT.

Plotting a function

The function class TF1 is the one of the basics ROOT classes.

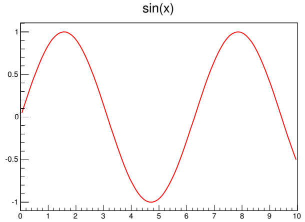
- Let's draw a function "*sin*(*x*)" in the range [0-10]:

```
root[0] TF1 * f1 = new TF1("f1","sin(x)",0,10);
```

  Do we need to learn how this complicated TF1 was build....No!!!

- Now draw the function:

```
root[1] f1→ Draw();
```



In prompt, also try root [ ] gPad→SetFillColor(34); f1→Draw()

Plotting a function

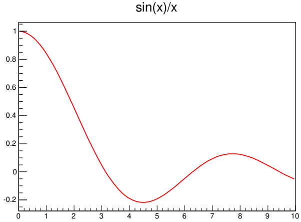
The function class TF1 is the one of the basics ROOT classes.

- Now try to change the function by "$sin(x)/x$" in the range [0-10]:

```
root[0] TF1 * f1 = new TF1("f1","sin(x)/x",0,10);
```

- Now draw the function:

```
root[1] f1→ Draw();
```

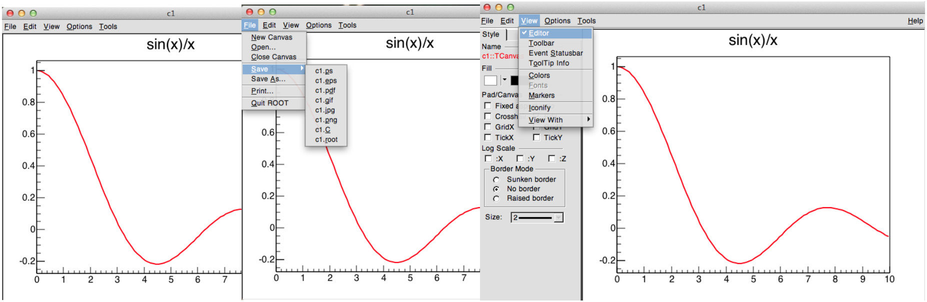


- All the plots appear in ROOT canvas.

ROOT Canvas

ROOT canvas has a GUI menu to

- save in different formats
- to edit the style of the plot.
- modify function, axis, bins.
- many other options...



Whatever you can do by GUI, can also be done by code.

Function With Parameters

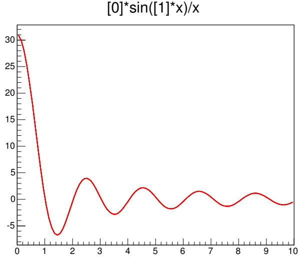
The function class TF1 can be used to create with any number of parameters.

- Let's create a function "$p_0 sin(p_1 x)/x$" with two parameters.

```
root[0] TF1 * f2 = new TF1("f2","[0]*sin([1]*x)/x",0,10);
```

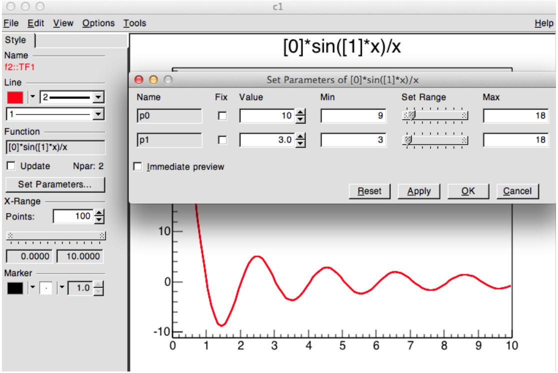
- We need to set values of parameters:

```
root[2] f2→SetParameter(0,10);
root[3] f2→SetParameter(1,3);
root[4] f2→ Draw();
```



Function Parameters in GUI

Function parameters can also be set by GUI menu in the ROOT canvas.



Extracting Information from TF1

One can extract several informations from a function. For example:

- Extract Parameter values:

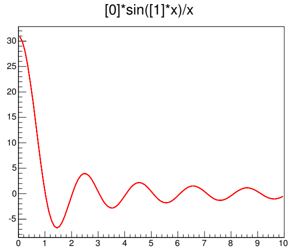
```
root[2] f2→GetParameter(0);
```

- Extract value of a function at any point:

```
root[3] f2→Eval(2.5);
```

- Integrate in given range:

```
root[4] f2→Integral(0,4);
```



For other options: `http://root.cern.ch/root/html/TF1.html`

Since function is defined as:

TF1 * pointer = new TF1("name","formula",min,max);

Therefore, any function available in ROOT can be typed in formula.

ROOT has almost all the major functions in **ROOT::TMath Namespace**.

Mathematical Functions in ROOT

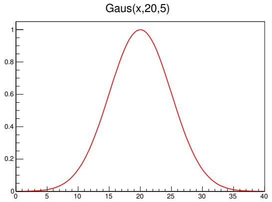
TMath namespace provides:

- Numerical constants.
- Functions to work with arrays.
- Statistical functions (e.g. Gauss) and other special functions (e.g. Bessel functions). For example, a Gauss function can be defined as:

```
root[ ] TMath::Gaus( x, mean, sigma);
```

so a Gauss TF1 can be defined as

```
root[ ] fb1 = new TF1("m1","Gaus(x,20,5)",0,40); fb1→Draw( )
```

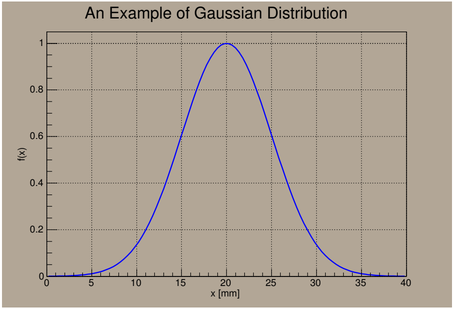


For other options, please see http://root.cern.ch/root/htmldoc/TMath.html

Cosmetics

You can avail many options to make your plot prettier.

```
void myfunc()
{
TF1 *fb1 = new TF1("m1","Gaus(x,20,5)",0,40);
fb1→SetTitle("An Example of Gaussian Distribution");
fb1→GetXaxis()→SetTitle("x [mm]");
fb1→GetYaxis()→SetTitle("f(x)");
fb1→GetXaxis()→CenterTitle();
fb1→GetYaxis()→CenterTitle();
fb1→SetLineColor(kBlue);
fb1 →Draw( );
gPad→SetGridx();
gPad→SetGridy();
gPad→SetFillColor(24);
fb1 →Draw( );
}
```



Notice Please: This time we have packed all the prompt-based commands in a script "myfunc.C" and can execute the code in terminal window by:

```
$ root myfunc.C
```

Exercises

- Drawing a function and Extracting information out of it

Exercise#1 Draw a function $f(x) = p_0 \frac{\ln(p_1 x)}{x}$ for parameter values $p_0 = 5$ and $p_1 = 3$ in the range 0 to 8. Set the blue color to function and extract following information from the function. Value of function at $x = 3$, integral of function from 2 to 8 and function derivative at $x = 2$.
Hint: Type function "[0]*ln([1]*x)/x" and use "myfunc.C" from RootTut/function/myfunc.C and try to modify it.

Thanks