ROOT

Computer Day Tutorial

Morgan Askins

11 May 2015

ROOT Data Analysis Framework

ROOT is a set of data analysis packages, written in C++ at CERN, used mostly in particle physics. While the base code itself using some high-level C++ implementation, ROOT itself is meant to be useable by novices.

Faces of ROOT

Our use of ROOT can be split into 3 different areas

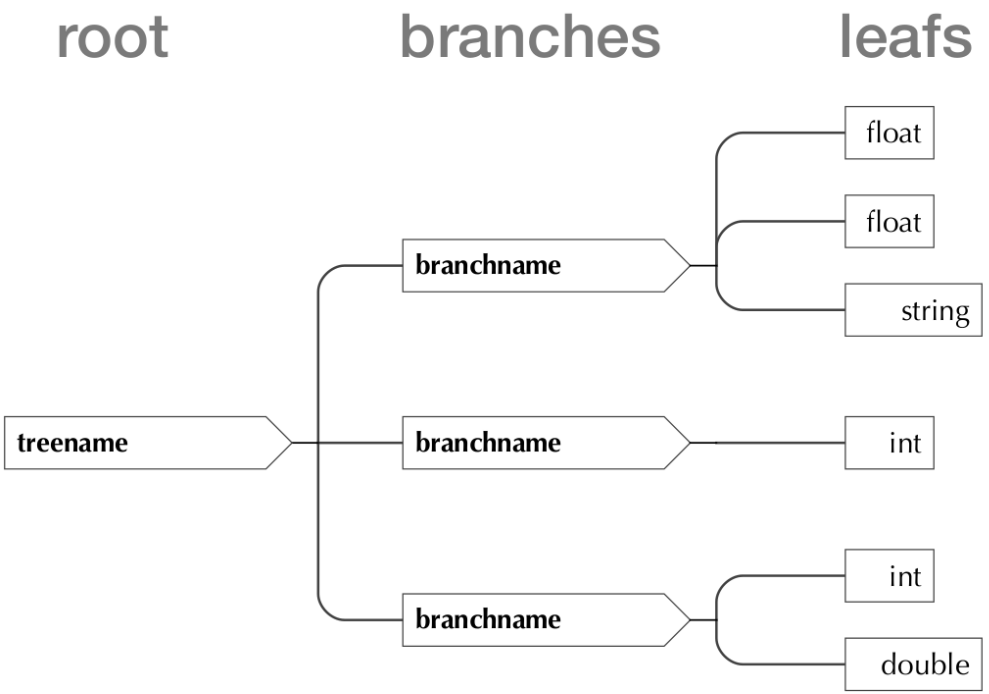
1. ROOT files structures (.root files)
2. C++ interpreter (CINT and CLING)
3. Data analysis packages and interaction with .root files

ROOT Files

When taking or simulating data you have freedom to choose how you store that data, whether it be proprietary binaries, matlab files, text files, ROOT files, etc.

- ROOT files benefit from being able to directly interact with ROOT analysis packages
- They contain a structure that is meaningful in particle physics
- They hit a good balance between speed and size.

ROOT File Structure



	Event 0	1	...	n
afloat	0.5	0.43	...	0.42
bfloat	0.2	-2.2	...	-1.2
astring	“a”	“b”	...	“c”
aint	0	1	...	900
bint	3	4	...	-8
adouble	1e-5	3.2e-6	...	1.3e3

Slide Complements Tina Pollmann

Example: Storing data

ROOT data is stored in a TFile in a directory style format.
TFile/TTree/TBranch/TLeaf

```
#include <TFile.h>

TFile fname("filename.root", "recreate");
/* Make some Trees, Branches, Histograms etc */
TTree tree("treename", "tree title");
fname.Write();
fname.Close();
// And if you prefer writing to heap over stack
TFile* fname = new TFIle("filename.root", "recreate");
fname->Write();
fname->Close();
delete fname;
```

Example: Accessing data

ROOT data can be accessed in the same way.

```
#include <TFile.h>

TFile fname("filename.root", "read");
/* Now you can access TObjects directly */
TTree* tree = (TTree*)fname.Get("treename");
// Make some plots, do some analysis
fname.Close();
```

C++ Interpreter

Advantages of an interpreter

- Prototype code
- Explore data structure
- Instant analysis
- Can load production macros

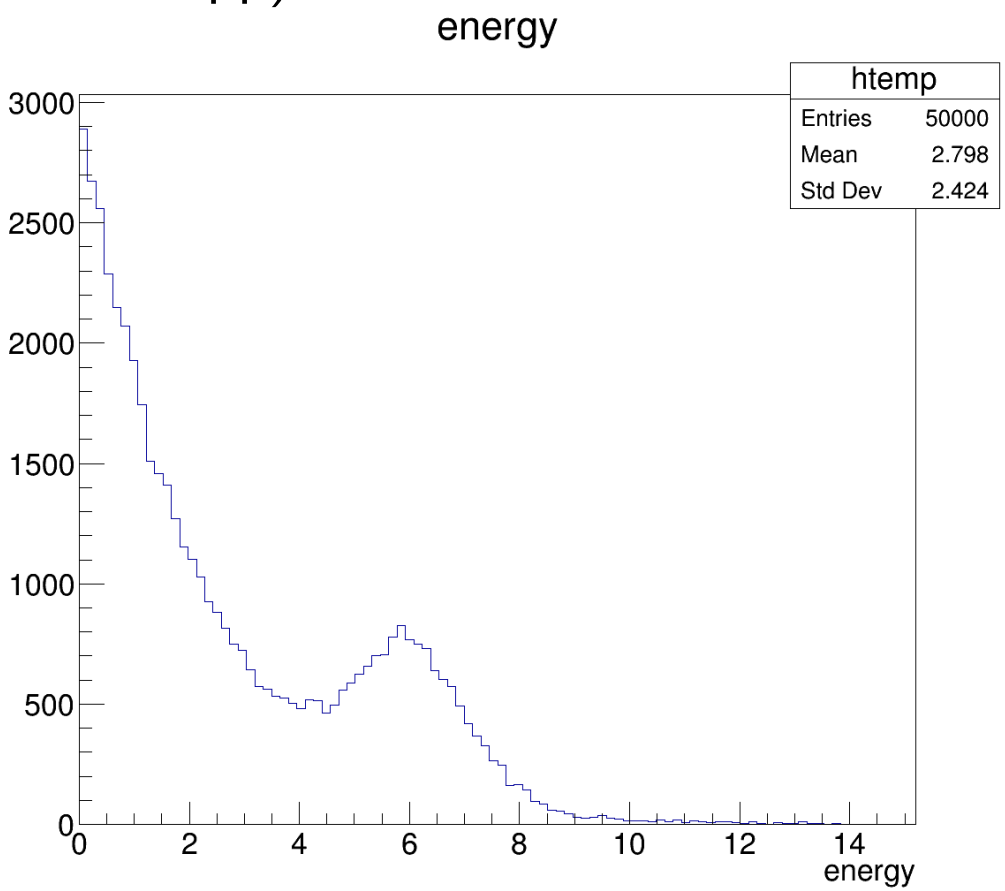
```
root [0] TFile fname("data.root", "read");
root [1] fname.ls()
TFile**         data.root
 TFile*         data.root
  KEY: TTree    dtree;1 data tree
root [2] TTree* tr = (TTree*)fname.Get("dtree");
root [3] tr->Print();
******************************************************************************
*Tree    :dtree     : data tree                                              *
*Entries :    10000 : Total =          161745 bytes  File  Size =     153020 *
*        :          : Tree compression factor =   1.05                       *
******************************************************************************
*Br    0 :energy    : energy/D                                               *
*Entries :    10000 : Total  Size=      80708 bytes  File Size  =      75402 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.06     *
*.............................................................................*
*Br    1 :time      : time/D                                                 *
*Entries :    10000 : Total  Size=      80694 bytes  File Size  =      77107 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.04     *
*.............................................................................*
```

Analysis Tools

Most importantly ROOT is used for data analysis. Here we create some sample data (see sample_data.cpp)

```
tree->Draw("energy");
```

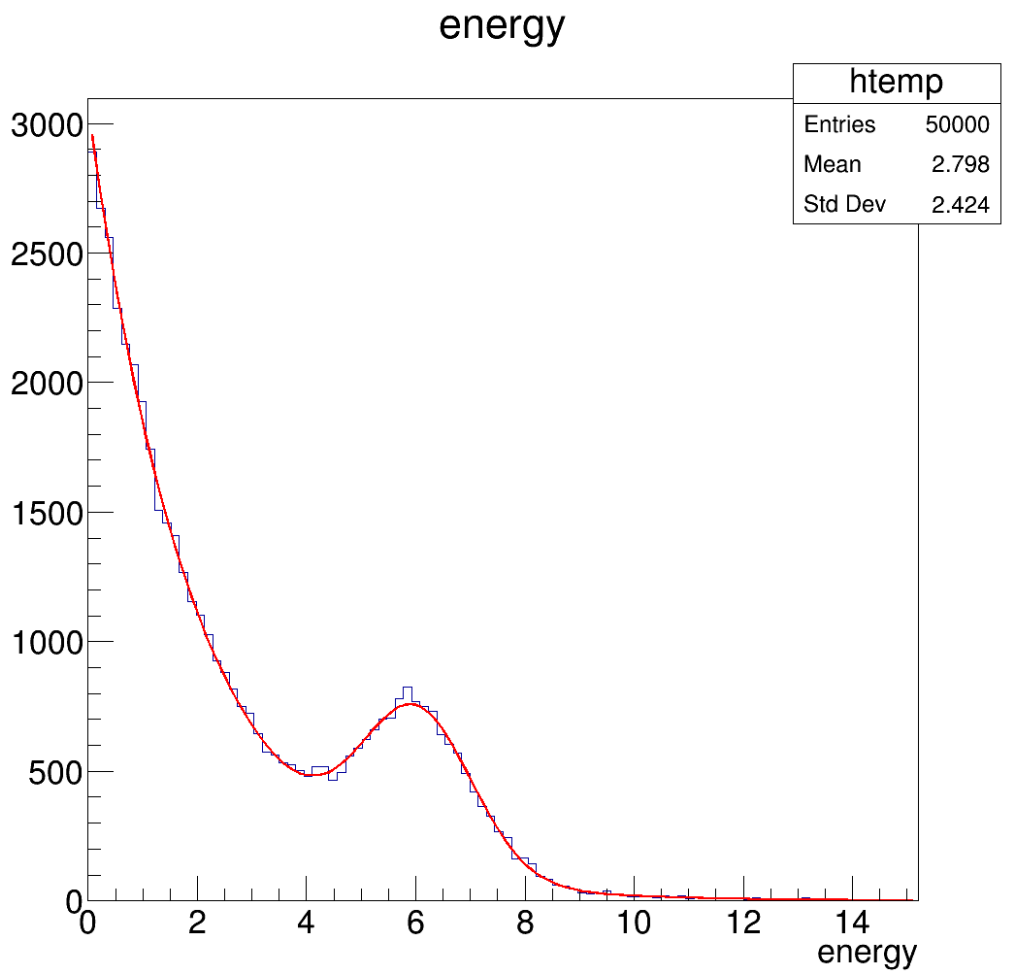
Drawn with TH1F (one dimensional floating point Histogram)



Fitting with TF1

We can use a TF1 to fit our data: `TF1* ff = new TF1("name", "[0]*TMath::Exp(-x/[1])+[2]*TMath::Gaus(x,[3],[4])", 0, 14)`

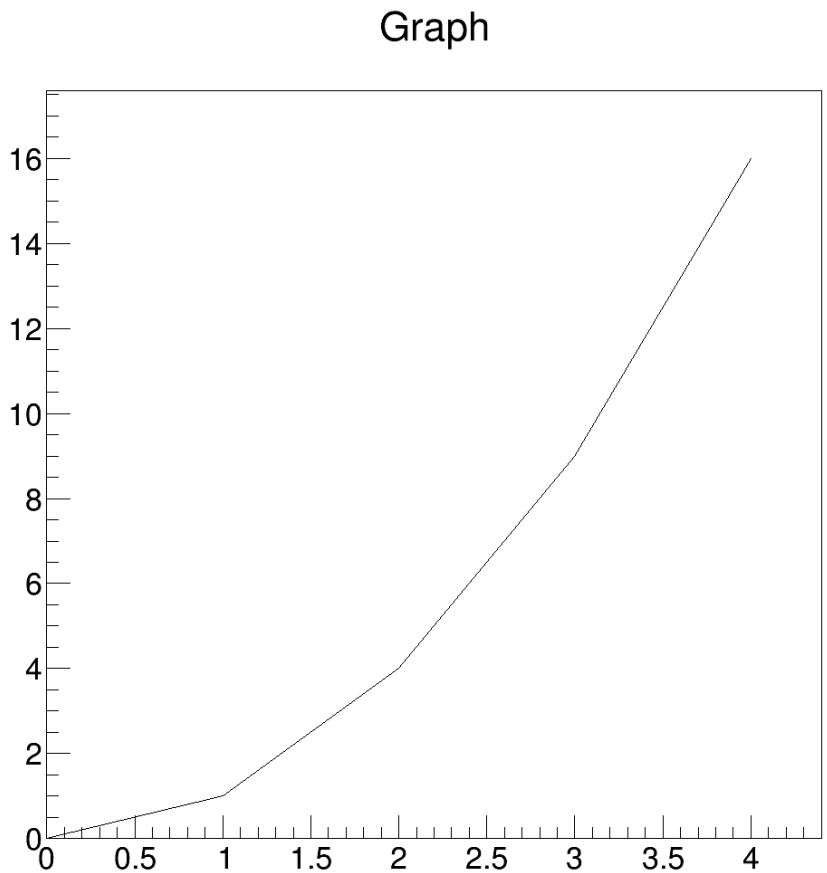
```
ff->SetParameters(1,1,1,1,1);
htemp->Fit(ff);
ff->Draw("same");
```



Graphs

ROOT can also generate graphs and fit them in the same manner

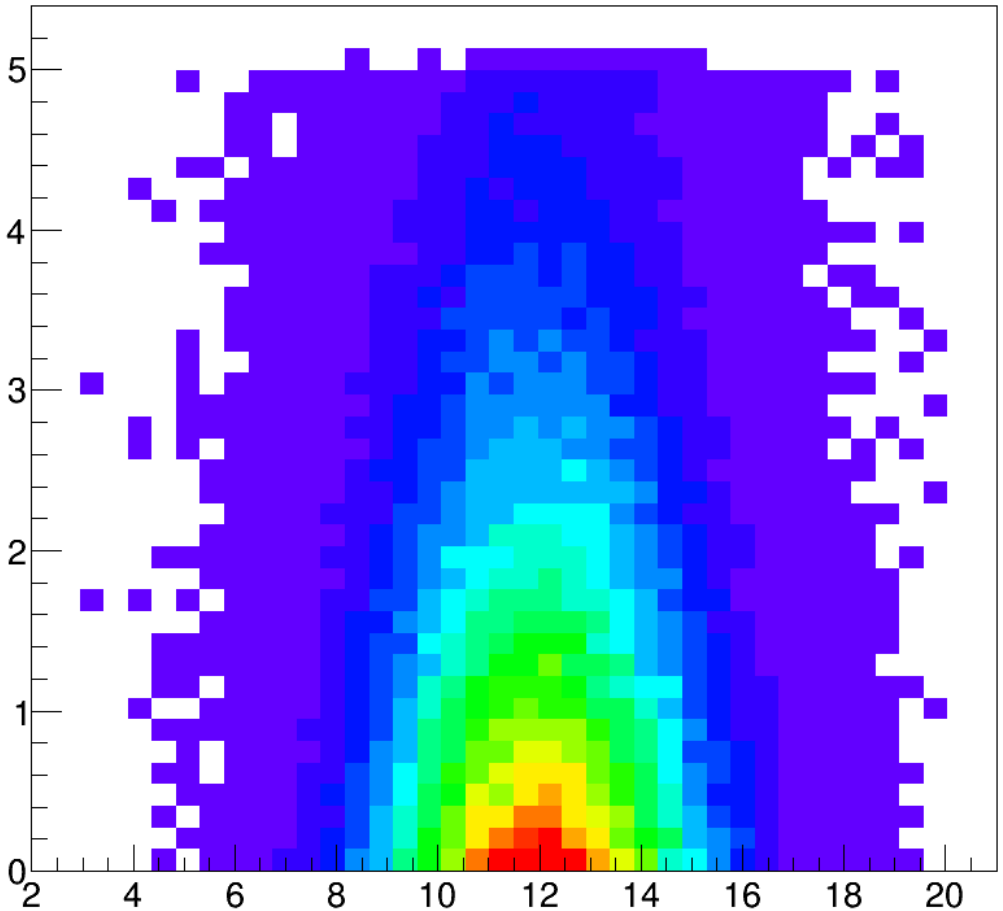
```
double x[5] = {0,1,2,3,4};
double y[5] = {0,1,4,9,16};
TGraph gr(5, x, y);
gr.Draw();
```



2D Histograms

2D Histograms are also possible (and a lot of fun). See 2d_data.cpp.

```
cdata->Draw("two:one", "", "COL");
```



TChain

Most of the time data will be separated into multiple files. In real data this is split into various "runs" depending on configuration, time, and other reasons. In simulation one can separate different backgrounds and signals into different files and scale them. See chainable.cpp.

```
TChain chain("atree");
chain.Add("tone.root");
chain.Add("ttwo.root");
// Could also do chain.Add("t*.root");
chain.Draw("energy");
```

Running your code

- Type what you want straight into the interpreter
- Write a root macro file and load it into the interpreter (Many ways to do this)
  `root myscript.c` or `root [0] .x myscript.c`
- Compile with cint (root -q -b myscript.c)
- Compile the code with gcc and run it (See TRint for graphics)
  `g++ ‘root-config --cflags --glibs‘ myscript.c`

Excercise 1: Installation

The level of difficulty in this task varies greatly depending on operating systems. Here is a general guide for installing ROOT.

1. Open a web browser and navigate to google.ca
2. Type "Install root cern (os name and version)"
3. Pray your guide simply says

Note: The version you want is probably some form of 5.34 (ask your collaboration) most collaborations have yet to migrate to 6.xx

Excercise 2: Pretty Plots

Given the data from sample_data.cpp:

1. Fit the data to an exponential + gaussian
2. Draw both fits separately over data
3. Plot data with statistical errors
4. Add a TLegend with data, fit of exp, and fit of gaus
5. Make a TCanvas with 2 pads and add the same plot on a logscale

https://root.cern.ch/root/html534/tutorials/