Root Guided Tutorials – Day 1

Brian Krar

EIEIOO

May 8th 2020

Introduction

Goal: Getting comfortable with ROOT

ROOT is an object-oriented C++ analysis package

- You should be using it this summer if you are doing any data analysis

Introductory look at some of the features of ROOT

- With hands-on coding exercises

Will start with a few “mini-exercises” (interactive sessions) just to get warmed up

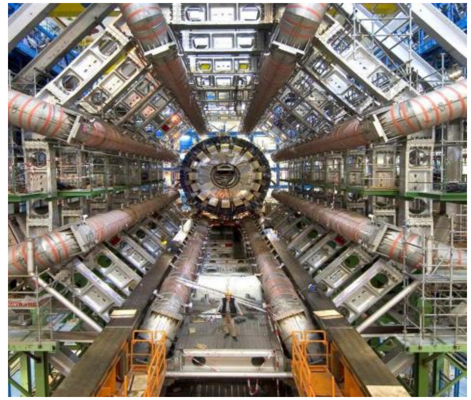
- Then some more involved tutorials -- using macros!
- Feel free to jump around as you see fit – slides should be posted already
- Encourage students do some searching yourselves online for other code/problems that might be more relevant to your summer project

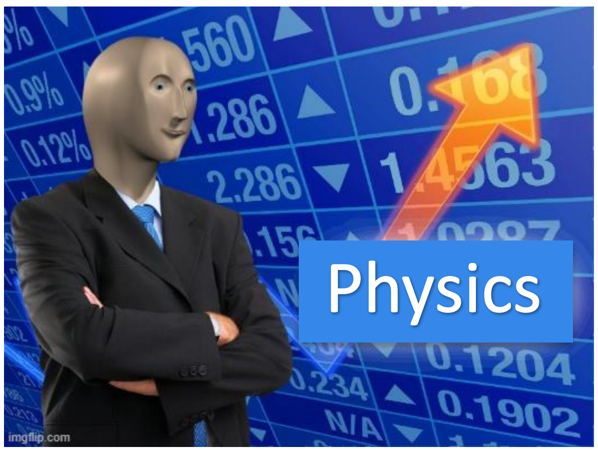
The best way to learn ROOT is by example!

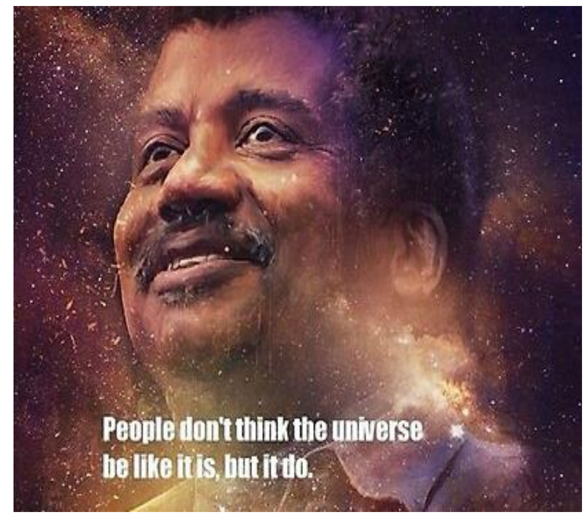
- And there are excellent tutorials on how to do specific tasks made by the developers:

https://root.cern.ch/doc/master/group__Tutorials.html

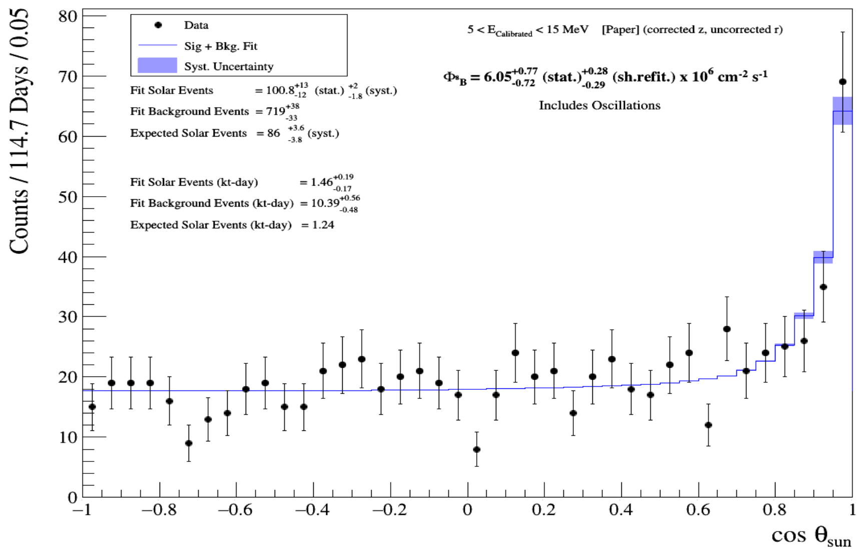
What I Expected:







What I Got:



Welcome to the ROOT forum!

News

system · 3 · Jun '16

ROOT 149 is the tool of choice for analyzing, storing and graphing High Energy Physics data.

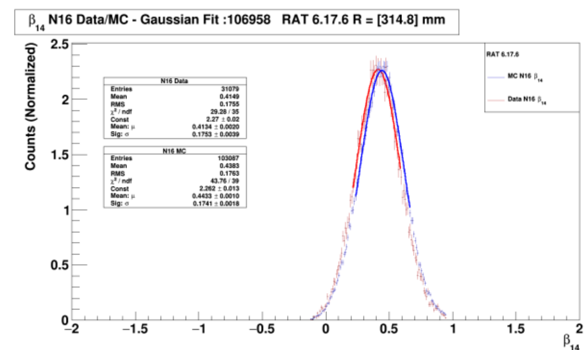
Use this forum for getting help on ROOT or discussing its features, including math, cling, graphics, or the I/O.

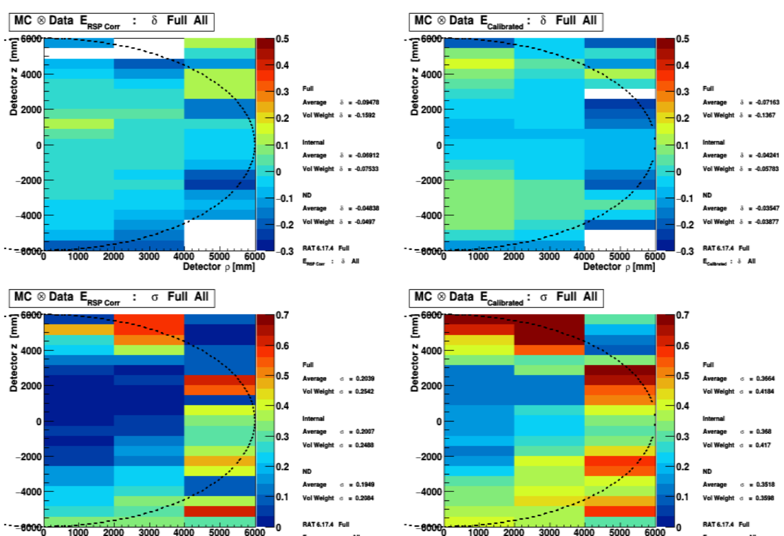
For documentation and download check the ROOT web site 149.
Please report bugs in our bug tracker 67, not here.

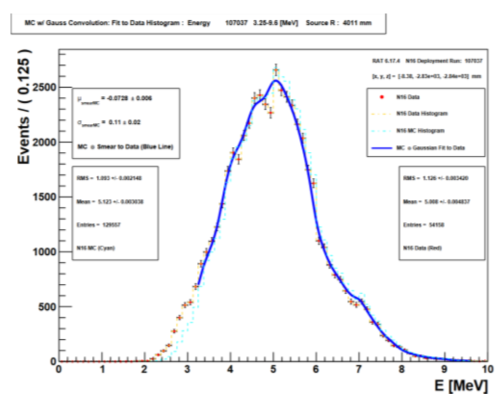
We definitely appreciate if you can help answering questions!

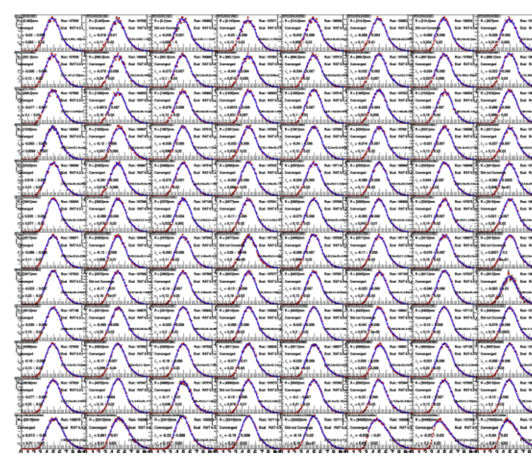
Have fun!

Your ROOT team.









https://www.youtube.com/watch?v=HluANRwPyNo

ROOT Interactive Session – Simple Plot

ROOT is based on CINT, a powerful C/C++ interpreter.

Blocks of lines can be entered within {...}.

Previous typed lines can be recalled.

```
Root > float x=5; float y=7;
Root > x*sqrt(y)
   (double)1.322875655532e+01
Root > for (int i=2;i<7;i++) printf("sqrt(%d) = %f",i,sqrt(i));
   sqrt(2) = 1.414214
   sqrt(3) = 1.732051
   sqrt(4) = 2.000000
   sqrt(5) = 2.236068
   sqrt(6) = 2.449490
Root > TF1 f1("f1","sin(x)/x",0,10)
Root > f1.Draw()
```

ROOT Data Types:

- Similar to C++:
  - Basic types: first letter is capitalised and have suffix "_t":
    int → Int_t   float → Float_t   double → Double_t
  - Names of root classes start with "T" e.g.
    TDirectory, TFile, TTree, TH1F, TGraph, …
- Some ROOT types (classes):
  - TH1F - Histogram filled using floating precision data
  - TH1D - Histogram filled using double precision data
  - TFile – a file containing persistent data
  - TDirectory – a directory (useful to keep a TFile tidy/organised)
  - TTree – can store per-event info in branches and leaves
  - TF1 – 1-dimensional function, TF2, …
  - TString – a ROOT string object (better than a C/C++ string)
  - TObjString – a persistable root string

Interactive ROOT - Histograms

1D Histograms:

Declare with:

```
TH1F h1(arguments …)
```

Make your first 1D histogram:

n_{bins} x_{min} x_{max}

```
TH1F  h_name("h_name", "h_title", 10, 0.0, 10.0);
```

`h_name` = key name of histo

`h_title` = name which appears on plotted histogram

Now draw the (currently empty) histo:

```
h_name.Draw();
```

Fill with a few entries:

```
h_name.Fill(1.);
h_name.Fill(3,10.7);
h_name.SetLineColor(8);
```

the number to fill the histogram with *(default value is 1.0)*

x value to fill histogram at

Try drawing the histogram when you have a few entries

```
h_name.Draw();        //do this occasionally to update the histogram
```

2D Histograms:

```
TH2F  h_name("h_name", "h_title", 10, 0.0, 10.0, 20, -10.0, 20.0);
```

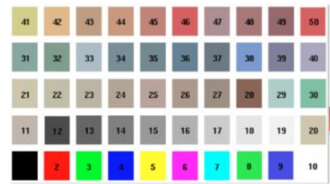
x axis co-ordinates   y axis co-ordinates

- 2D histograms behave the same as 1D histograms
- have some interesting Draw() options
  - surf - draw a surface
  - surf1 - draw a surface with colour contors
  - cont - draw a contour plot
  - contz0 - draw a contour plot with the y axis scale shown
  - lego - draw a 2D histogram
  - box - draw boxes (default is to spread points out according to the defined bins)
  - text - draw 2D grid of number of entries per bin.

Some useful commands to play with now that you've got a histogram

`h_name.SetFillColor(Color_t color = 1)`	Change the fill colour.
`h_name.SetFillStype(Style_t styl = 0)`	Change the fill style.
`h_name.SetLineColor(Color_t color = 1)`	Change the line colour.
`h_name.SetLineStyle(Style_t styl = 0)`	Change the line style.
`h_name.SetLineWidth(Width_t width = 1)`	Change the line width.

Line colours and styles are described in the 'Graphical Objects Attributes' section of the ROOT user guide.



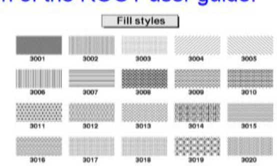
kRed
kOrange
kYellow
kSpring
kGreen
kTeal
kCyan
kAzure
kBlue
kViolet
kMagenta
kBlack
kPink

Make sure you use colours wisely! There is nothing more annoying than seeing a talk projected onto a screen with half a dozen invisible lines!

Try and stick to 'safe' colors like blue, red and black.

Can define new colours using the TColor class.

Line colours and styles are described in the 'Graphical Objects Attributes' section of the ROOT user guide.



Available fill styles shown left

Remember to give axis labels a sensible title:

```
h_name.SetXTitle("This is the x-axis")
h_name.SetYTitle("This is this y-axis")
```

(you made the histogram so you know what is in it! It is good form to pay some courtesy to people you show the plot to by adding titles to the axes abd means you'll have one less question to answer!)

Aside: Adding a Legend to a canvas:

```
leg_hist = new TLegend(0.5,0.6,0.79,0.79);
leg_hist->SetHeader("Some histograms");
leg_hist->AddEntry(h_name,"First histo","l");
leg_hist->AddEntry(hist_2,"Second histo","l");
leg_hist->Draw();
```

Interactive ROOT – Histograms (Cont.)

Add a Second Histogram:

```
TH1F *hist_2 = new TH1F("hist_2", "Another histo", 100, 0, 20);   //Note that we've used pointers here
hist_2->Fill(3,10);
hist_2->Fill(5,4);
hist_2->Fill(3);
hist_2->SetLineColor(4) //blue
hist_2->Draw("same");
```

Change Plot Labels:

```
h_name.GetXaxis().SetTitle("Label of x axis");
h_name.GetYaxis().SetTitle("Label of y axis");
```

Copying Histograms:

You can make an identical copy of a histogram by cloning, with the command:

```
TH1F *hist_new=(TH1F*)hist_2->Clone();
hist_new->SetName("hist_new");
hist_new->SetLineColor(kYellow);
hist_new->Fill(1,4);
hist_new->Draw("same");
```

Values vs. Pointers

- **A value type contains an instance of an object**
- **A pointer *points* to the instance of an object**
- **Create a pointer**

  `root [ ] TF1* f1 = new TF1("func", "sin(x)", 0, 10)`
- **Create a value type**

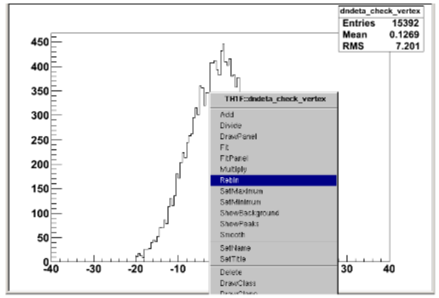
  `root [ ] TF1 f2("func", "cos(x)", 0, 10)`
- **One can point to the other**

  ```
  TF1 f1b(*f1)        // dereference and create a copy
  TF1* f2b = &f2      // point to the same object
  ```

Interactive ROOT – Histogram - Fit (Cont.)

Fill Histogram With Gaussian Info:

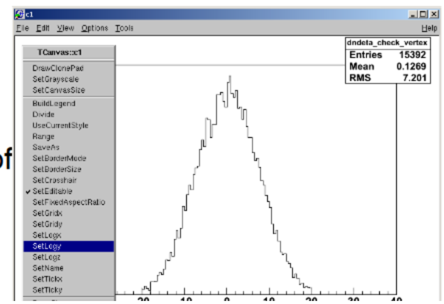
```
root [ ] TH1F h("h","h",80,-40,40)
root [ ]  TRandom r;
root [ ] for (i=0;i<15000;i++) { h.Fill(r.Gaus(0,7));}
root [ ] h.Draw()
```



- **Rebinning**
  root [ ] h.Rebin(2)
- **Change ranges/canvas**
  - with the mouse, very easy!
  - with the context menu
  - command line

  root [ ] h.GetXaxis()->
  SetRangeUser(2, 5)
- **Log-view**
  - right-click in the white area at the side of the canvas and select SetLogx (SetLogy)
  - command line

  root [ ] gPad->SetLogy()



Fit Histogram With Gaussian (Either Interactively or w/ Command Line):

- **Interactive**
  - **Right click on the histogram and choose "fit panel"**
  - **Select function and click fit**
  - **Fit parameters**
    - are printed in command line
    - in the canvas: options - fit parameters
- **Command line**
  root [ ] h->Fit("gaus")
  - Other predefined functions polN (N = 0..9), expo, landau

Interactive ROOT – Files

Save Histogram to File:

- Open a file for writing
  root [ ] file = TFile::Open("file.root", "RECREATE")
- Write an object into the file
  root [ ] h->Write()
  root [ ] hist->Write()
- Close the file
  root [ ] file->Close()

NEW
READ
RECREATE
UPDATE
....

Open Histogram, and Plot:

- **Open the file for reading**
  root [ ] file = TFile::Open("file.root")
- **Read the object from the file**
  root [ ] hist->Draw()
  (only works on the command line!)

Can Also View using TBrowser

You've already met `TFiles` – this part should help you understand a bit more how to use them

- Files can contain directories, histograms and trees (ntuples) etc.
- These are 'persistent' objects
- In root you make an object persistent by inheriting from `TObject`

A few file commands/constructors that you've already met:

- Open an existing file (read only)
  `TFile myfile("myfile.root");`

- Open a file to replace it
  `TFile myfile("myfile.root", "RECREATE");`

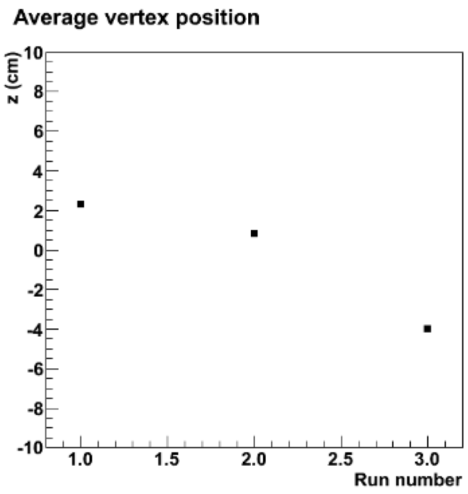
  or append to it:
  `TFile myfile("myfile.root", "UPDATE");`

Interactive ROOT – Graphs

- A graph is a data container filled with distinct points
- TGraph: x/y graph without error bars
- TGraphErrors: x/y graph with error bars
- TGraphAsymmErrors: x/y graph with asymmetric error bars

Graph Example

```
graph = new TGraph;
graph->SetPoint(graph->GetN(), 1, 2.3);
graph->SetPoint(graph->GetN(), 2, 0.8);
graph->SetPoint(graph->GetN(), 3, -4);
graph->Draw("AP");
graph->SetMarkerStyle(21);
graph->GetYaxis()->SetRangeUser(-10, 10);
graph->GetXaxis()->SetTitle("Run number");
graph->GetYaxis()->SetTitle("z (cm)");
graph->SetTitle("Average vertex position");
```



Interactive ROOT – Trees and nTuples

A tree is a data type that is convenient for HEP analysis

Creating and Filling a Tree:

- You want to store objects in a tree which is written into a file
- Initialization

```
root [ ] TFile* f = TFile::Open("events.root",
"RECREATE");
root [ ] TTree* t = new TTree("Events","Event Tree");
root [ ] Int_t      var1;
root [ ] Float_t    var2;
root [ ] Float_t    var3;
root [ ] t->Branch("var1", &var1, "var1/I");
root [ ] t->Branch("var2", &var2, "var2/F");
root [ ] t->Branch("var3",  &var3, "var3/F");
```

Fill the TTree

TTree::Fill copies content of member as new entry into the tree

```
root [ ] var1=5; var2=3.1; var3=10.;
root [ ] t->Fill();
root [ ] var1=1; var2=7; var3=4.5;
root [ ] t->Fill();
```

Inspect the tree

Flush the tree to the file close the file

```
root [ ] t->Print();
root [ ] t->Show(1);

root [ ] t->Write();
root [ ] f->Close();
```

Reading Entries from a Tree:

- Open the file, retrieve the tree and connect the branch with a pointer to TMyEvent

```
TFile *f = TFile::Open("events.root");
TTree *tree = (TTree*)f->Get("Events");
Float_t var2;
tree->SetBranchAddress("var2", &var2);
```

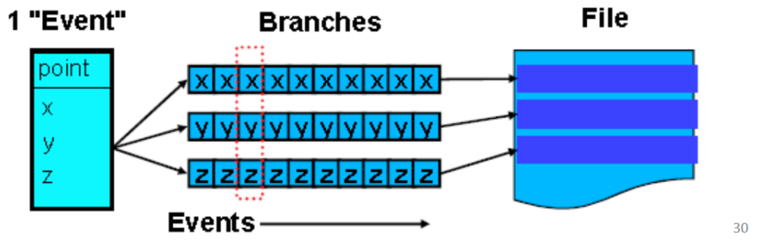
- Read entries from the tree and use the content of the class

```
Int_t nentries = tree->GetEntries();
for (Int_t i=0;i<nentries;i++) {
  tree->GetEntry(i);
  cout << var2 << endl;
}
```

Interactive ROOT – Trees and nTuples

A tree is a data type that is convenient for HEP analysis

- The class TTree is the main container for data storage
  - It can store any class and basic types (e.g. Float_t)
  - When reading a tree, it is designed to access only a subset of the object attributes (e.g. only particle's energy) so that certain branches can be switched off → speed up of analysis when not all data is needed



A TNtuple is a subset of a tree, restricted to only float type variables

- Create a TNtuple

  root [ ] ntuple = new TNtuple("ntuple", "title", "x:y:z");

  – "ntuple" and "title" are the name and the title of the object

  – "x:y:z" reserves three variables named x, y, and z
- Fill it

  root [ ] ntuple->Fill(1, 1, 1);
- Get the contents

  root [ ] ntuple->GetEntries(); number of entries

  root [ ] ntuple->GetEntry(0); for the first entry

  root [ ] ntuple->Args()[1]; for y (0 for x, 2 for z)

  – These could be used in a loop to process all entries
- List the content

  root [ ] ntuple->Scan();

Draw a histogram of the content

– to draw only x

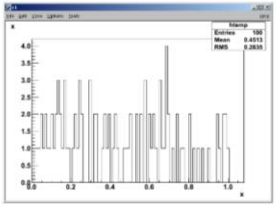
root [ ] ntuple->Draw("x");

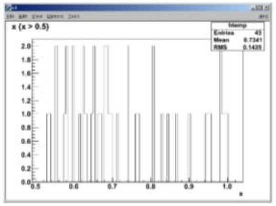
– draw all x that fulfill x > 0.5

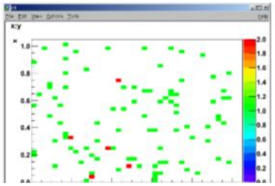
root [ ] ntuple->Draw("x", "x > 0.5");

– to draw x vs. y in a 2d histogram

root [ ] ntuple->Draw("x:y ", "", "COLZ");







ROOT Macros

In reality, basically all of what you do in ROOT will be in macros (not using the command line)

- You will create these macros with your text editor of choice (Emacs, Vim etc.)

You will be coding your macros in C++ (see yesterday's tutorial)

- PyRoot is available for students who prefer python programming (but this is outside the scope of this course)
- PyRoot can make sense to use for "quick--and--dirty" analysis efforts, if that's the kind of work you'll be doing this summer
- But C++ code, when compiled, is faster than python – so can be better for more complex analyses

Recall the two ways of running scripts:

- Un-named scripts:

```
{
  ~~#include <iostream.h>~~
  cout << "Hello, World!\n";
}
```

includes not needed

  - Code must be enclosed in curly braces!
  - Execute with
    ```
    root[] .x script.C
    ```

- Named scripts:

```
#include <iostream.h>
int main() {
  cout << "Hello, World!\n";
}
```

  - More like normal C++ programs, recommended form!
  - Execute with:
    ```
    root[] .L script.C
    root[] main()
    ```

ROOT Macro – Function Plotting – Exercise 1

Let's combine a few of the concepts we've been looking at in a macro:

Go back to the very first thing we did on the command line
Plot the Function sin(x)/x between [0,10]:

```
#include "TF1.h"
using namespace std;

void plotFunction() {

   TF1 * f1 = new TF1("f1","sin(x)/x",0,10);
   f1->Draw();
}
```

Now I want you to extend this:

a) Add a second function f2 = a*sin(b*x)/x,   Set a = 1, b = 2
b) Plot this function as a blue line
c) Find the value and derivative of f2 at x =1
d) Find the integral of f2 from [3,6]
e) Change the value of [a,b] to the following:
   - [a,b] = [5,5] , and Plot on same graph in Green
   - [a,b] = [5,1] , and Plot on same graph in Cyan
   - [a,b] = [0,0.5] , and Plot on same graph in Orange
   - **Hint – Will need to use DrawClone**
f) Change the range of the x axis to [3,6]
   - **Hint – Will need to include "Taxis.h"**

`h_name.SetFillColor(Color_t color = 1)`	Change the fill colour.
`h_name.SetFillStype(Style_t styl = 0)`	Change the fill style.
`h_name.SetLineColor(Color_t color = 1)`	Change the line colour.
`h_name.SetLineStyle(Style_t styl = 0)`	Change the line style.
`h_name.SetLineWidth(Width_t width = 1)`	Change the line width.

Line colours and styles are described in the 'Graphical Objects Attributes' section of the ROOT user guide.

Can define new colours using the TColor class.

kRed
kOrange
kYellow
kSpring
kGreen
kTeal
kCyan
kAzure
kBlue
kViolet
kMagenta
kBlack
kPink

Make sure you use colours wisely! There is nothing more annoying than seeing a talk projected onto a screen with half a dozen invisible lines!

Try and stick to 'safe' colors like blue, red and black.

ROOT Macro – Exercise 2 - Histogram Plotting and Operations

Create a 1D histogram with 100 bins between [-5,5] and fill it with 10000 gaussian distributed random numbers with mean 2 and sigma 1

- Print the Mean and RMS of the histogram
- Plot the histogram
- Save the histogram as “gaus.root” file

Hint: For generating gaussian random numbers use gRandom->Gaus(mean, sigma) – Need “TRandom3.h”

ROOT Macro – Exercise 2 - Histogram Plotting and Operations

Create a 1D histogram with 100 bins between [-5,5] and fill it with 10000 gaussian distributed random numbers with mean 2 and sigma 1

- Print the Mean and RMS of the histogram
- Plot the histogram
- Save the histogram as “gaus.root” file

Hint: For generating gaussian random numbers use gRandom->Gaus(mean, sigma) – Need “TRandom3.h”

Code should look something like this:

```
#include "TH1D.h"
#include "TCanvas.h"
#include "TRandom3.h"
#include "TFile.h"
using namespace std;

void PlotGaus(){

TH1D * h1 = new TH1D("h1", "h1", 100, -5, 5);

for (int i = 0; i < 10000; i++) {

    double mean1 = 0.; double width1 = 1.;
    double gaus_val = gRandom->Gaus(mean1, width1);
    h1->Fill(gaus_val);
}

    h1->SetLineColor(2);
    string ctitle1 = "Random Gaus Hist";
    h1->SetTitle(ctitle1.c_str());
    h1->SetName(ctitle1.c_str());

    TFile f("gaus_hist.root","RECREATE");
    f.Write("h1");
    f.Close();

    Double_t Mean_h1 = h1->GetMean(); //define mean of histogram
    Double_t Sigma_h1 = h1->GetRMS(); //define sigma of histogram

    cout << "Mean h1   : "<< Mean_h1 << "    Mean h1   : "<< Sigma_h1 << endl;

    h1->GetXaxis()->SetTitle("E (Gaus) [MeV]");
    h1->GetYaxis()->SetTitle("# Events (Gaus)");

    TCanvas *c1 = new TCanvas("c1","c1",1800,800);
    c1->cd(1);
    h1->Draw();
    c1->Print("random.pdf");
}
```

ROOT Macro – Exercise 2 - Histogram Plotting and Operations

Now I want you to extend that code:

- Fit a Gaussian to the peak of your gaussian sampled Histogram
- Plot Both the Fit and the Histogram together

- Create a second histogram - this time with Landau Function sampled with a mean of 0, sigma of 1 from [-5,5]
- Plot both Histograms on the same canvas, but different Pads
- Save entries from each of these histograms as an [x,y] pair in a space separated text file (called “hist_output.txt”)
- Add more features to the canvas– add Legend, stats box, Set axis titles, histogram titles

ROOT Macro – Graph Plotting – Exercise 3

Create Macro that makes a TGraph Plot from this hist_output.txt file

- Label graph axes (call x-axis “Gaus” and y-axis “Landau”)

Now It's Your Turn!

Read through some of the official ROOT "primer"
https://d35c7d8c.web.cern.ch/sites/d35c7d8c.web.cern.ch/files/ROOT5Primer.pdf

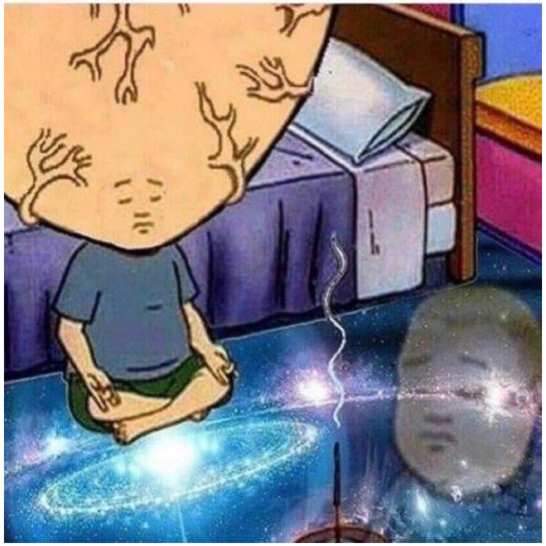
Try some of the following exercises yourself
- **Solutions can/will be posted as needed**

Look through some of the additional ROOT resources I've listed
- **Can also search for some yourself online. There is a wealth of resources (and completed tutorials/code) for ROOT online**

Practice, Practice, Practice!

Again, highly encourage you to go through at least some of the official ROOT tutorials:
https://root.cern.ch/doc/master/group__Tutorials.html

You, after reading the ROOT tutorials and references



Working With Actual Physics (Simulation)

Independent Exercise

I've simulated ~30,000 electrons at 5MeV at the centre of the SNO+ detector

- This should be located in your neutrino guest directory – Sim1.root, Sim2.root

Also included is ex_make_cuts.cc macro

- Code that loads .root files and applies cuts
- Puts results into several different 1D, 2D histograms

Working With Trees:

- We want to load up these events, and perform some cuts
- Try performing Cuts in energy and position
- Count the number of events that pass/fail these cuts, and efficiency

Working With Histograms, Fits:

You are now equipped to apply your own cuts and plot the histograms, and make the plots look nice

- Include axis labels, Stats. Box, Legends etc.
- Can Fit a Gaussian to the energy (over what range is best? – consider the chi2/dof of the fit?)

Some Other Suggested Problems to Try

Run and familiarize yourself with the create_tree.cc and EventData.h Files that I'll attach with these slides

- The EventData.h file creates a Particle class (similar to what we did yesterday) and the create_tree.cc fills a tree with particle event information
- **Exercise: Create a macro that reads the eventdata.root file that is created by it**
- **This macro should be able to cut**

Create your own Custom Gaussian Function (rather than using the build-in one) and fit the gaus.root file you generated with it.

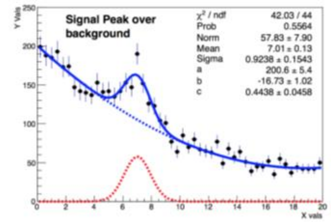
- Compare the output with a fit from the build-in Gaussian. Plot both on the same histogram

Fill a 2D histogram with Gaussian and Lorentzian data. Plot the profiles/projections Compare the different 2D plotting options (see page 6)

- For help with this problem, see 36 of the ROOT primer I've linked

Create a fake Gaussian + Exponential (sig+BG) Histogram dataset, and perform a fit for the Gaussian signal

- Refer to the ROOT primer for help with this exercise



Some Selected ROOT Resources

https://www.youtube.com/watch?v=s9PTrWOnDy8
Very good video series that goes through ROOT. Highly recommend.

https://www.nevis.columbia.edu/~seligman/root-class/files/RootClass.pdf
Very good intro tutorial to follow along with.
Adapted some of the exercises, but this is more comprehensive

https://www.slac.stanford.edu/BFROOT/www/doc/workbook/root1/root1.html
Some intro Resources from the BarBar collaboration (some of which was done here)

https://indico.cern.ch/event/607726/contributions/2475150/attachments/1490075/2315743/ROOT_lecture.pdf
Good Intro resource, especially for plotting and such. Also where I sourced the HEP example from

http://pprc.qmul.ac.uk/~bevan/GCL/ROOT.pdf
Very readable intro slides

http://ific.uv.es/~fiorini/ROOTTutorial/root_tutorial.pdf
Good introductory slides, used some of them in this presentation

https://github.com/root-project/training
Github repository of root sample problems. Nice resource, and good to test your new github skills

Again, this is not an exhaustive list! Don't be afraid to do a little digging yourself!