Basic ROOT Tutorial

Roger Wendell
raw22@phy.duke.edu
2010.07.01

Topics

What is ROOT?
Interactive ROOT session
  - command line vs. macros vs. user-compiled code
Opening files/ accessing information
Trees and histograms
Fitting
Other useful things...

Exercises
What is ROOT?

- ROOT
- C++ based **Object Oriented data analysis** framework
- Developed by physicists at CERN
- Uses some known and trusted HEP libraries (with extensions) as well as several home grown tools
- A development project for more than a decade, official at CERN since 2002
- Used in a variety of HEP experiments
What is **Object Oriented** programming?

- Object oriented programming
  - “Higher” Level type of programming
  - Somewhat arranged in a way that is human understandable
  - Main pieces of program (Objects) are inherently modular
  - Objects are collections of properties and functions that do work, *self responsible*

**Bank Account Object**

**Properties:**
- Account Number
- Account Holder's Name
- Current Balance....

**Functions:**
- Compute Balance
- Make a Withdrawal
- Make a Deposit....
What is data analysis?

Data Analysis

The process of turning your data into something that is human understandable (and therefore meaningful)

Your Data:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>150.142</td>
<td>1.45592</td>
<td>4.66344</td>
<td>146.706</td>
</tr>
<tr>
<td>4</td>
<td>149.942</td>
<td>-10.342</td>
<td>11.0689</td>
<td>148.325</td>
</tr>
<tr>
<td>5</td>
<td>150.185</td>
<td>17.0842</td>
<td>-12.1425</td>
<td>143.101</td>
</tr>
<tr>
<td>6</td>
<td>150.018</td>
<td>5.19219</td>
<td>7.78532</td>
<td>148.594</td>
</tr>
<tr>
<td>7</td>
<td>150.052</td>
<td>7.54787</td>
<td>-7.43332</td>
<td>144.446</td>
</tr>
<tr>
<td>8</td>
<td>150.071</td>
<td>0.231547</td>
<td>-0.021123</td>
<td>147.784</td>
</tr>
</tbody>
</table>

Your Result:

A CDF top quark result
What is ROOT, really?

- **ROOT**
- C++ based **Object Oriented data analysis** framework
  - Collection of programming structures (objects) that you can use to convert your data into a meaningful result

(some of ) What ROOT can do:

<table>
<thead>
<tr>
<th>Plotting</th>
<th>Fitting / Computation</th>
<th>Detector Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Plotting" /></td>
<td><img src="image2.png" alt="Fitting / Computation" /></td>
<td><img src="image3.png" alt="Detector Visualization" /></td>
</tr>
</tbody>
</table>
ROOT can be used in many ways

- **Complied Code**
  - ROOT objects come in a series of libraries that can be linked into a standard C++ program
    ```c++
    # include <iostream>
    # include "TH1F.h"
    ...
    int main( int argc , char * argv[] )
    {
      // code
      TH1F * myHistogram = new TH1F("histogram","this is my title", 100, 0.0, 1.0);
      ...
    }
    (today)
    ```

- **Command line**
  - *pseudo* C++ code written within the ROOT interpreter (CINT)
  - Good for making quick plots or computations, checking files etc.

- **Unnamed macros**
  - Processed within CINT, all commands within a single “function”
    ```c++
    {
      // code
      TH1F * myHistogram = new TH1F("histogram","this is my title", 100, 0.0, 1.0);
      ...
    }
    ```

- **Named macros**
  - CINT interpreted scripts allowing multiple function declaration
ROOT **Command Line** : Interactive ROOT

- Commands are in C++ style, and interpreted with CINT
- Standard C++ syntax is *allowed* but *not required*
- CINT Syntax is quite sloppy
  - “->” and “.” are interchangeable, “;” is not required at the end of commands etc., no type checking, etc..
- Best to use standard C++ syntax to prevent poor programming practices from leaking into your other projects

OK, Let’s get started with ROOT

- Documentation
  - [http://root.cern.ch/root/html526/ClassIndex.html](http://root.cern.ch/root/html526/ClassIndex.html)
- Directory containing files used in this lecture

Start ROOT from your terminal:
- > root
ROOT **Command Line**: Hello World! And Calculator

```bash
raw@sukap007[ /home/raw 22 ] root -b
**************************************************************************
* WELCOME to ROOT *
* Version 5.26/00b  9 February 2010 *
* You are welcome to visit our Web site *
  http://root.cern.ch *
**************************************************************************

ROOT 5.26/00b (tags/v5-26-00b#32327, Feb 11 2010, 14:21:13 on linuxx8664gcc)

CINT/ROOT C/C++ Interpreter version 5.17.00, Dec 21, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.

```c
root [0] const char * OutputString = "Hello World";
root [1] cout << OutputString << endl;
Hello World
```

- Declare a “C” style string
- Print the string to the terminal

```c
root [2]
root [3]
root [4]
root [5]
root [6]
root [7]
root [8]
```

- Declare some variables
- Do some math with them...

**Other Math:**

- `cos(...)` , `sqrt(...)`, `log(...)`, `log10(...)`, `atan(...)`, `exp(...)`, several others

```c
root [8] .q
```

Use “.q” to quit root
**ROOT Functions**

- Functions are one of the fundamental objects used in ROOT
- An integral part of fitting distributions
- Neutrino oscillation probability:
  \[ P(\nu_{\alpha} \rightarrow \nu_{\beta}) = \sin^2 2\theta \sin^2 \left( \frac{1.27 \Delta m^2 L}{E} \right) \]
  \[ \left[ \frac{eV^2 km}{GeV} \right] \]

```c
root [0] TF1 * nuOscProb = new TF1("prob", "sin(1.27*2.1e-3*x / [0])**2", 0.0, 1.0e4);
```

- Declare a pointer to an object of type TF1
  - TF1 means one-dimensional function
  - The pointer's name is "nuOscProb"

- Define the function using C-style math
  - [0] – numbers in "[..]" are parameters,
    - They can be set externally
    - Here we use it as the neutrino energy
  - x - is the evaluation variable, here it stands for \( L \) the neutrino path length

```c
root [1] nuOscProb->SetParameter(0, 3.0);
```

// Set the energy to 3.0 GeV and draw
ROOT Functions : An Exercise

- Create and draw a Gaussian whose mean is 50 and standard deviation is 10 on the range [0, 100]

- Here is the formula: Note there are 3 parameters

\[
f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},
\]
ROOT Command Line: Some Objects

Let's open a file and see what's inside

Declare a pointer to a TFile object

Create the object it will point to (ie open the file histograms.root)

Use the pointer “input” to list its file’s contents with the member operator “->” and function “ls()”

There are two histograms in this file
Both are objects with properties and functions we can use to display our data

Declare a pointer to a TH1F object

Tell it to point to the object “GausHist1d” stored in our file
**ROOT Canvases and Drawing options**

- Objects in ROOT are drawn on canvases known as TCanvas.
- If you don't specify your TCanvas, ROOT will provide one for you.
- This isn't always ideal: you may want to draw things of a certain size, or change the color, title, or other attributes of the drawing space.

```plaintext
root[3] TCanvas * myCanvas = new TCanvas("myCanvas", "Example Canvas", 300, 600);
r
root[4] myCanvas->SetFillColor( kWhite );
r
root[5] myCanvas->Divide(1,2);
```

Split the Canvas into two (1 col, 2 rows)

- Change some of the attributes of the histogram:
  ```plaintext
  root[6] my1Dhist->SetTitle("I made this title");
r
  root[7] my1Dhist->SetMarkerColor( kRed );
r
  root[8] my1Dhist->SetMarkerStyle(21);
r
  root[9] my1Dhist->SetMarkerSize(0.3);
  
  Change to the first drawing pad:
  ```plaintext
  root[10] myCanvas->cd(1);
r
  ```

```
<e : Error Bars
P : Using Markers (points)
```

Save what's drawn on the canvas to file (specified by suffix):
```plaintext
root[12] myCanvas->SaveAs("a_plot.gif");
Info in <TCanvas::Print>: GIF file a_plot.gif has been created
```
ROOT Drawing Options and Macros

Histograms are drawn via the THistPainter class in ROOT. You can find all drawing options by looking at the web documentation for THistPainter:


- Histogram axis properties are controlled by TAxis objects which are members of histograms

  ```
  root [1] my1Dhist->GetXaxis()->SetTitle("X axis");
  root [2] my1Dhist->GetYaxis()->SetTitle("Y axis");
  root [3] my1Dhist->GetXaxis()->CenterTitle();
  root [4] my1Dhist->GetYaxis()->CenterTitle();
  ```

- Unnamed macros are just a collection of ";" separated commands placed within a single file and between a global set of braces

  - They can be executed like:
    ```
    root [1] .x MacroName.C
    ```
Write a short Macro to open and Draw the two-dimensional histogram contained in the file histograms.root. Label the X and Y axes something interesting. Experiment with two drawing styles for this type of histogram and plot each one in a different panel of your canvas.

These two websites will be helpful:


The history of commands you have entered into CINT is at:

$HOME/.root_hist
ROOT **TTree**

- **Trees in ROOT**
- A TTree is a data structure for organizing and manipulating several data variables at once.
- Capable of drawing histograms on the fly including making selection cuts on the data.
- Uses ROOT's internal compression algorithms to reduce the data size.
- Very useful for data storage (Several experiments now use TTrees in this way.)

Variables are stored in **TBranches**.

```
Tree

Event #  Time  Angle  Energy

37^{th}  106.5ns  0.308  1.7 GeV
n
```
Create pointer to “tree1” that exists in file “f1”

Print structure of tree to screen
This tree contains 7 variables:
  - event, ebeam, px, py, pz, zv, chi2

Turn on statistics box

Draw scatter plot (py vs. px) for events with ebeam>150
ROOT **TTree**: More about Arguments

- Arguments to many functions in ROOT objects are passed by character strings.
- Strings are parsed for both logic and mathematics.
- For trees:
  - Any variable in the tree can be manipulated as part of an argument.

```cpp
```

**What to draw for each event**

- Semicolon `:` indicates adding a new dimension.
- Can be functions of variables: `sqrt( py )`.
- Can be combinations of variables: `ebeam/px :py**2`.

**Selection Cuts**: I.e. which events or entries to draw

- Multiple cuts are allowed, combined with C-style logic operators.
- Can be functions of variables.
- Can be combinations of variables.

**Drawing Options**

Options for n-dimensional histograms go here as in previous examples.
ROOT TTree: What if I want to use a TTree to fill a histogram?

- Step 1: Define a histogram with a suitable range
  ```
  root [2] TH1F * h = new TH1F("hBeamEnergy", "Beam Energy", 200, 148.0, 152.0);
  ```

- Step 2: Project the TTree contents into the histogram
  ```
  ```

  **Optional Cuts**
  Project into the **NAME** of a histogram, **not** its pointer!
  Variable used to fill the projected histogram. Make sure the dimensions of your histogram and your projection are the same!

  **A Caution**
  If you Project into an undeclared histogram name, ROOT will automatically create one for you **AND** a pointer to it of the same name

  ```
  root [3] mytree->Project("undeclared", "px", "ebeam < 149.0");
  ```
  Now Exists!!!
ROOT TTree: When using complicated cuts multiple times...

- Consider Encapsulating your cuts as **TCut** objects
  
  ```cpp
  root[14] TCut * px_plane = new TCut("px/log(px**2 + py**2) > 0.10" );
  root[15] TCut * py_plane = new TCut("py/log(px**2 + py**2) > 0.10" );
  ```

  Can be as complicated as any other string

  ```cpp
  root[16] mytree->Draw("ebeam", *px_plane );
  ```

- **TCut** objects can be combined using C-style operators as usual
  
  - They can be combined with other string cuts
    
    ```cpp
    root[17] mytree->Draw("ebeam", *px_plane && *py_plane );
    root[18] mytree->Draw("ebeam", *px_plane && "py > 5" );
    ```

  ROOT will expand the TCuts as if they were a normal string
ROOT TTree : An Exercise

- Using the Tree contained in tree.root make a distribution of the **total momentum** of each whose beam energy was **outside** of the mean by more than 0.2
- Use TCut objects to make your event selections
- Project this distribution into a histogram, draw it and save it to a file
ROOT **Simple Fitting**

- Often you will have to perform fits to your data to extract particle masses, lifetimes, etc.
- There are several ways to do simple fits within root
  - (You will learn much more about fitting in the next lecture)

**Use ROOT's built in functions:**

```c
root [0] TF1 * func1 = new TF1("func1", "gaus" );
root [1] TF1 * func2 = new TF1("func2", "expo" );
```

**Define your own functions** – much like we did earlier in the lecture

```c
root [2] TF1 * f = new TF1("gs", "[0]*exp( -0.5 * ((x-[1])/([2]))**2) + [3]*sin([4]*x )" );
```

- You must give root some place to start for each of the parameters:
  ```c
  root [4] f->SetParameter( 3, 1.0 );
  ```
- Now simply tell your Histogram or Tree to “Fit” using this function!
  - The best values of the parameters will be stored in the TF1
    ```c
    root [5] myHistogram->Fit("gs");
    ```
Available ROOT fit functions may be found in TFormula class.

A few examples:

- gaus
- expo
- polN \ (N=0,1,2,...)
ROOT : Full Exercise

- Write a Macro to examine the contents of the file `double.root`
- Locate the TTree within and plot the “Mass” variable
- Plot the “Mass” variable on a separate canvas after selecting only positively charged events and those events that are within 30 degrees of the beam direction
- Project this distribution into an appropriate one-dimensional histogram and fit it using the sum of two gaussians
- What are the mean and standard deviation of each peak?
- What is the Chi Squared value of the fit?
- How many degrees of freedom?
ROOT : Random Numbers

Random numbers can be produced and used in a few ways
- Generated from a user defined function

```
root [0] TF1 * nuOscProb = new TF1("prob", "sin( 1.27*2.1e-3*x / [0] )**2 ", 0.0, 1.0e4);
root [1] nuOscProb->SetParameter(0, 1.0 );
root [2] nuOscProb->SetParameter(1, 3.0 );
root [4] nuOscProb->GetRandom( );
(Double_t)9.99481458133042725e+03
```

- Useful for filling histograms

```
root [5] TH1F * h = new TH1F("h", "random h", 1000, 0.0, 1e4 );
root [6] h->FillRandom( f , 10000 );
```
ROOT: TRandom3 Numbers

ROOT also provides several pure random number generators
- See the TRandom documentation for more details
- TRandom3 has the best compromise of speed and performance

TRandom3 * rand = new Trandom3();
Double_t  myNum = rand->Gaus(30.0, 10.0);

Can generate numbers from these distributions
- Exp(tau)
- Integer(imax)
- Gaus(mean,sigma)
- Rndm()
- Uniform(x1)
- Landau(mpv,sigma)
- Poisson(mean)
- Binomial(ntot,prob)

Can be used to fill histograms:
TH1F * hlandau = new TH1F("l", ",", 100, 0.0, 100.);
for( int i = 0 ; i < 10000 ; i++ )
  hlandau->Fill( rand->Landau(30.0, 15 ) );
ROOT: Other interesting tools

TLorentzVector
A general 4-vector class with implemented functionality to do almost everything you typically need to do with 4-vectors:
  dot products
  rotations
  boosting
  angle between vectors
  magnitude...

TFractionFitter
Fits data histogram using multiple MC histograms (instead of a defined function)

TFitter, TMinuit
Classes for fitting

THStack
Takes a collection of histograms and draws them “stacked” on each other.

(next lecture)
Summary

- ROOT is a versatile piece of software that has many tools to help make your data into meaningful and understandable results.
- The ROOT documentation is your friend, use it!
- There will be more tutorials this summer, particularly on advanced fitting techniques so please stay tuned.

Some links for more information

ROOT Classes  http://root.cern.ch/root/Categories.html
ROOT Discussion Forum  http://root.cern.ch/phpBB2/

BaBar ROOT tutorials

Nevis ROOT tutorial
  http://www.nevis.columbia.edu/~seligman/root-class/