Introduction to ROOT

Ian Lam

ROOT

- Mainly used in particle physics – developed by/for CERN-LHC
- Capable of handling (storage, querying) large amounts of data
- Excellent for statistics and fitting – robust
- I'll be demonstrating it in C++ since that is what I am familiar with. Can also be used in Python (ref. Mark Anderson).

ROOT - Recall

- Open root using “root –l”
- See a setup reminiscent of something like MatLAB.
- Can type stuff line by line
- Obviously, when you quit ROOT, you would have to re-type everything.
- Write scripts/macros and execute them.

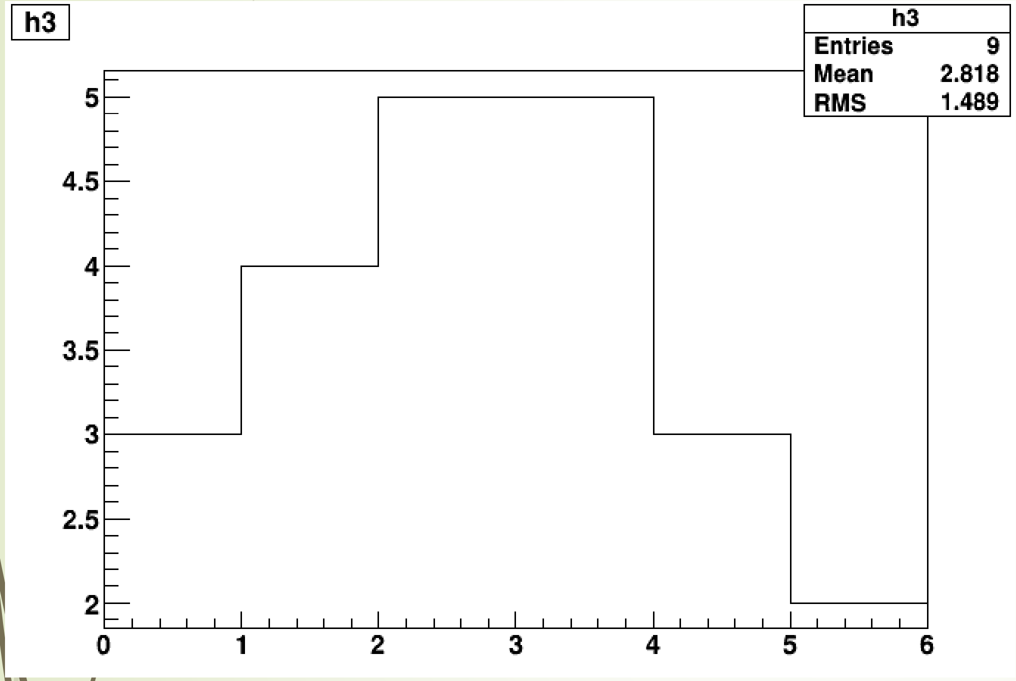
ROOT

- Few modes:
  - .x script.cc (execute)
  - .L script.cc (load)
  - .L script.cc+ (compile)
  - .L script.cc++ (force compile)
- execute and load are for short and simple tests. But as your code grows in complexity, with many functions and moving parts, best to compile.
- Compiling can help in debugging before running – catch silly things like passing wrong variable types or typos.

ROOT

- Usually, the first task that is given to new students: use ROOT to plot a histogram of the distribution of electrons at a particular energy in the center of the detector.
- Histograms and trees - main objects.
- Let’s start with histograms!

ROOT-Histograms



- Can think of it as baskets containing items.
- Eg: 1.35 will fall into basket **(bin)** 2
- Area of bin is the amount of stuff.

ROOT - Histograms

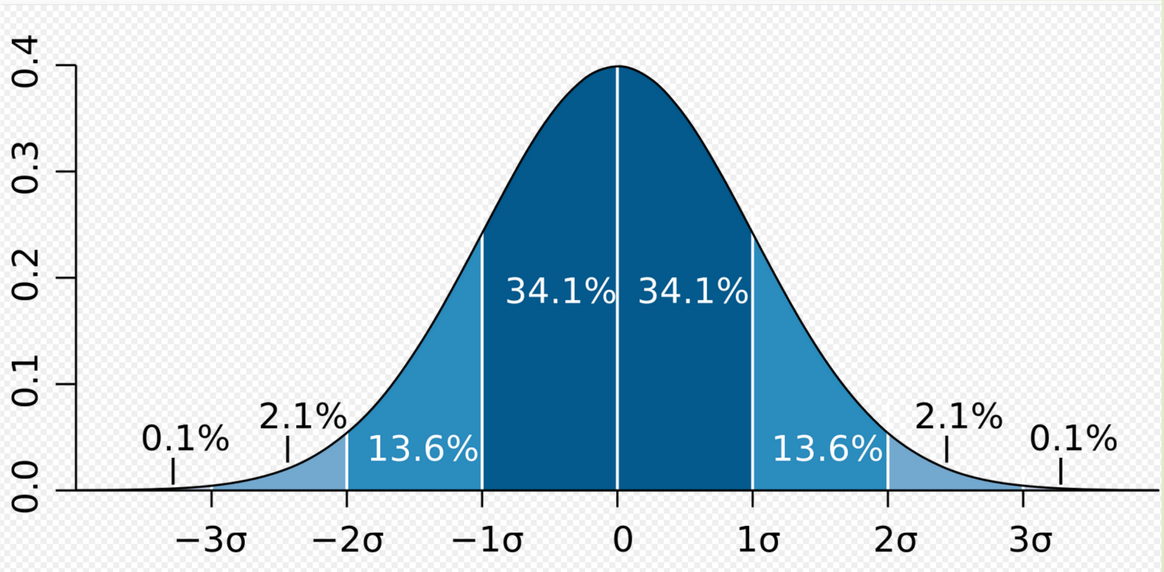
- TH1D h1 = TH1D ("h1", “The Title", 6, 0, 6)
  - 1st: histogram name – usually the same as the variable
  - 2nd: histogram title – appears when you draw it
  - 3rd: number of bins
  - 4th: starting edge
  - 5th ending edge
  - underflow and overflow bins (0 and nbins+1)
- Each bin has an index/number. Starts from 1 going left to right.
- h1.SetBinContent(binnum, content)
  - binnum: bin number
  - content: how much stuff in the bin

ROOT - Histograms

- Tip: .root_hist (root history, not root histogram)
  - notice how you can press “up” to see previous inputs? Wonder where it is stored?
  - can access it if you want to make a copy of what you typed.
- Obviously, it is going to be painful setting bin contents by hand for histograms. Write a macro/script to do it!

Histogram Tutorial

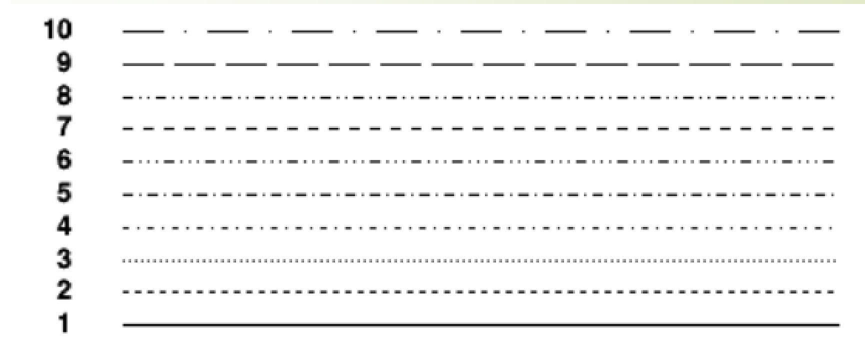
- Draw random numbers from Gaussian distribution.
- Fill.
- Plot!

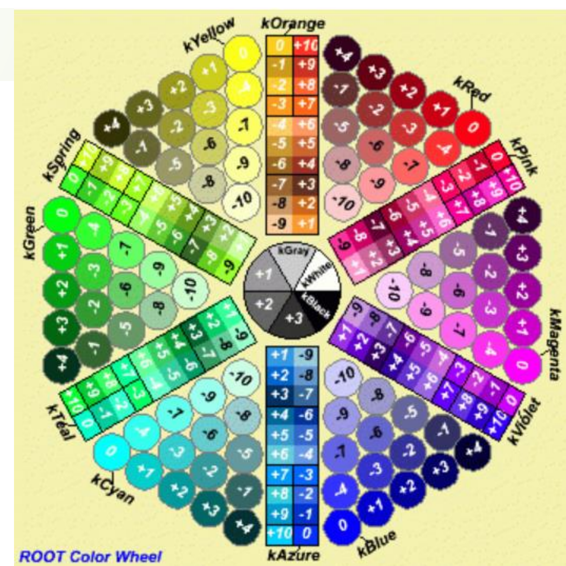


Histogram - Beautify

- h1.SetLineWidth(2)
- h1.SetLineColor(kRed) {kRed+3, kGreen-2, etc}
- h1.SetLineColor(2)
- h1.SetLineStyle(3)
- h1.GetXaxis().SetTitle(“energy”)
- h1.GetYaxis().SetTitle(“counts”)
- h1.GetXaxis().SetTitle("#gamma_{5}^{true}")

Google: root cern color wheel, line style, line width – can’t miss it





Histogram - Beautify

- In guided ROOT exercises:
- TLegend leg = TLegend (0.58,0.73,0.90,0.90);
- leg.AddEntry(h1, “True energy","lep");
- TLatex *tex1 = new TLatex(x-point,y-point, “this plot”);
- TLine *l1 = new TLine(x1, y1, x2, y2);
- Make a second histogram and overlay them.

Histogram - Fit

- Fit the histogram to a pre-defined Gaussian
- $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5\left\{\frac{x-\mu}{\sigma}\right\}^2}$
- Fit options:
  - default: chi-squared
  - likelihood
  - elaborate more on Monday / Phillipe's stat's lecture (Tue)

Histogram Tutorial

- Goal:
  - histogram creation and manipulation
  - drawing random numbers from a distribution
  - fitting histograms
  - beautify

ROOT – Data storage

- Usually in lab courses, when you use instrumentation to take measurements, the outputs are stored in .csv, .txt, your lab notebook, etc.
- Becomes very unwieldy when dealing with millions of measurements, and each measurement has many attributes.

ROOT – Data storage

- Roughly, in particle physics, an 'event' is a single measurement that triggers the detector.
  - Now, SNO+ has a trigger rate of ~2000 Hz. So, one hour of data recording would give about ~7.2 million recorded events. 24 hours running would give….a lot.
- Each event can have various parameters associated with it.
  - For SNO+, the raw output file records the number of PMTs hit, timing information etc.
- These are then used to 'reconstruct' details about an event, like 'energy', 'position in detector', etc.
- A .csv file with ~10^6 rows and ~50 columns isn't going to work to well.

ROOT – Trees

- ROOT files are structured based on trees, branches, leaves.
- Trees : the main 'directory' *(class: TTree)*
- Branch: sub-directories
- Leaves: attributes of events and where the data is held.
- Deal with a one-dimensional tree, sometimes called an 'Ntuple'. *(class: TNtuple)*
- Trees with the same structure (has to be exactly same) can be linked together into a *TChain* object.

Trees

Can think of a one layer tree in table form.

ROOT reads column-wise i.e. only need to load the attributes of interest.

Regular database storage reads row-wise i.e. need to load whole event into memory.

Event	energy	posx	…
1	5.3	100	…
2	4.6	150	…
3	6.5	189	…
4	2.4	105	…
5	8.3	107	…
…	…	…	…

Might not seem like a big deal but we usually deal on the order of millions of events, with many of attributes.

Source: https://indico.desy.de/indico/event/8607/session/4/material/0/1

Trees

- Useful commands to probe quickly (in terminal) the structure of the tree.
- Say we have a TTree object named "output"
  - **output->Print()** : Shows structure of the tree i.e. what branches are there and how many entries there are.
  - **output->Scan():** Shows the branches/variables with their values in tabular form.
    - When you have many variables, can choose to show some of them by doing **output->Scan("Var1:Var6:Var10")**
  - **output->Show(index):** shows all attributes related to the event stored at index
- Or can use TBrowser: in ROOT, do "TBrowser b;"
  - Since it is a GUI, there might be a lag if you are ssh-ing into a server.

Source: https://root.cern.ch/root/htmldoc/guides/users-guide/Trees.html

Trees

- Quick demonstration of previous slide with a prepared tree.

Tree Tutorial

- What this exercise will highlight:
  - Generate random numbers
  - Create a non-default function.
  - Passing variables to functions.
  - Combining trees with same structure in memory.
  - Create and save an ntuple

ROOT

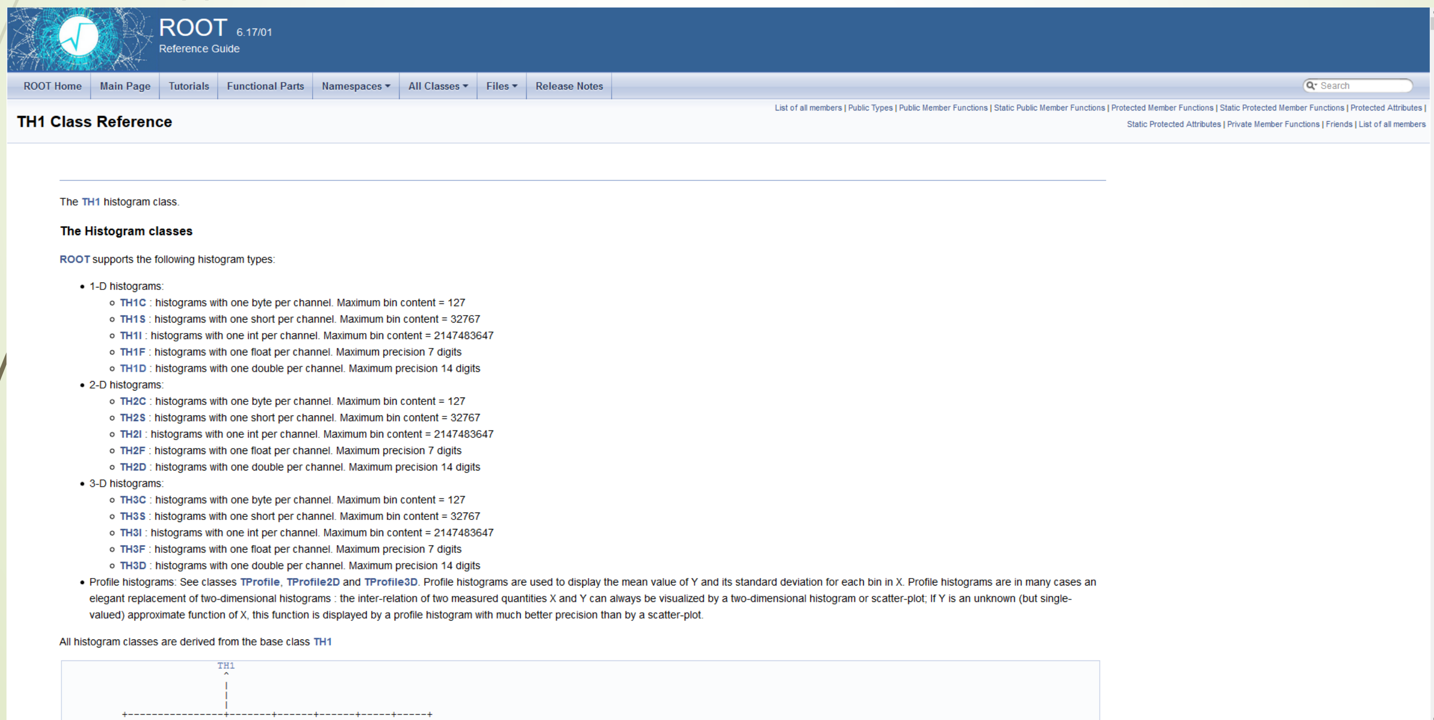
- Congratulations! You now know how to use ROOT to perform a simple data analysis in ROOT:
  - Get data files from your detector, probably in TTree format.
  - Open the data files, and look at distribution of events.
  - Select specific events (i.e. events greater than 5MeV)
  - Make pretty histograms, and fit them.

ROOT – Searching for info

- Google "root cern <what you want to search>"
  - Example, "root cern fit":
    - https://root.cern.ch/root/htmldoc/guides/users-guide/FittingHistograms.html

ROOT – How to search for class information

- Example: type "root cern th1" in google.
- TH1 is the 1D histogram class for ROOT.
- TH1D is for 1D histograms that use the data type 'double', TH1F is for data type 'float' etc.



ROOT 6.17/01
Reference Guide

ROOT Home | Main Page | Tutorials | Functional Parts | Namespaces | All Classes | Files | Release Notes

Search

List of all members | Public Types | Public Member Functions | Static Public Member Functions | Protected Member Functions | Static Protected Member Functions | Protected Attributes | Static Protected Attributes | Private Member Functions | Friends | List of all members

TH1 Class Reference

The TH1 histogram class.

The Histogram classes

ROOT supports the following histogram types:

- 1-D histograms:
  - TH1C : histograms with one byte per channel. Maximum bin content = 127
  - TH1S : histograms with one short per channel. Maximum bin content = 32767
  - TH1I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH1F : histograms with one float per channel. Maximum precision 7 digits
  - TH1D : histograms with one double per channel. Maximum precision 14 digits
- 2-D histograms:
  - TH2C : histograms with one byte per channel. Maximum bin content = 127
  - TH2S : histograms with one short per channel. Maximum bin content = 32767
  - TH2I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH2F : histograms with one float per channel. Maximum precision 7 digits
  - TH2D : histograms with one double per channel. Maximum precision 14 digits
- 3-D histograms:
  - TH3C : histograms with one byte per channel. Maximum bin content = 127
  - TH3S : histograms with one short per channel. Maximum bin content = 32767
  - TH3I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH3F : histograms with one float per channel. Maximum precision 7 digits
  - TH3D : histograms with one double per channel. Maximum precision 14 digits
- Profile histograms: See classes TProfile, TProfile2D and TProfile3D. Profile histograms are used to display the mean value of Y and its standard deviation for each bin in X. Profile histograms are in many cases an elegant replacement of two-dimensional histograms : the inter-relation of two measured quantities X and Y can always be visualized by a two-dimensional histogram or scatter-plot; If Y is an unknown (but single-valued) approximate function of X, this function is displayed by a profile histogram with much better precision than by a scatter-plot.

All histogram classes are derived from the base class TH1

```
TH1
 ^
 |
+---------------+-------+------+------+-----+-----+
```

Public Member Functions

virtual	**~TH1** () Histogram default destructor. More...
virtual Bool_t	**Add** (TF1 *h1, Double_t c1=1, Option_t *option="") Performs the operation: `this = this + c1*f1` if errors are defined (see TH1::Sumw2), errors are also recalculated. More...
virtual Bool_t	**Add** (const TH1 *h1, Double_t c1=1) Performs the operation: `this = this + c1*h1` If errors are defined (see TH1::Sumw2), errors are also recalculated. More...
virtual Bool_t	**Add** (const TH1 *h, const TH1 *h2, Double_t c1=1, Double_t c2=1) Replace contents of this histogram by the addition of h1 and h2. More...
virtual void	**AddBinContent** (Int_t bin) Increment bin content by 1. More...
virtual void	**AddBinContent** (Int_t bin, Double_t w) Increment bin content by a weight w. More...
virtual Double_t	**AndersonDarlingTest** (const TH1 *h2, Option_t *option="") const Statistical test of compatibility in shape between this histogram and h2, using the Anderson-Darling 2 sample test. More...
virtual Double_t	**AndersonDarlingTest** (const TH1 *h2, Double_t &advalue) const Same function as above but returning also the test statistic value. More...
virtual void	**Browse** (TBrowser *b) Browse the Histogram object. More...
virtual Int_t	**BufferEmpty** (Int_t action=0) Fill histogram with all entries in the buffer. More...
virtual Bool_t	**CanExtendAllAxes** () const Returns true if all axes are extendable. More...
virtual Double_t	**Chi2Test** (const TH1 *h2, Option_t *option="UU", Double_t *res=0) const χ^2 test for comparing weighted and unweighted histograms More...
virtual Double_t	**Chi2TestX** (const TH1 *h2, Double_t &chi2, Int_t &ndf, Int_t &igood, Option_t *option="UU", Double_t *res=0) const The computation routine of the Chisquare test. More...
virtual Double_t	**Chisquare** (TF1 *f1, Option_t *option="") const Compute and return the chisquare of this histogram with respect to a function The chisquare is computed by weighting each histogram point by the bin error By default the full range of the histogram is used. More...
virtual void	**ClearUnderflowAndOverflow** () Remove all the content from the underflow and overflow bins, without changing the number of entries After calling this method, every undeflow and overflow bins will have content 0.0 The Sumw2 is also cleared, since there is no more content in the bins. More...
TObject *	**Clone** (const char *newname=0) const Make a complete copy of the underlying object. More...
virtual Double_t	**ComputeIntegral** (Bool_t onlyPositive=false)

- Public Member Functions are the functions available for use for the particular class. If you scroll further, there are also Private and Protected Functions. These are used internally in the background of a class and you can’t access them, I think.

ROOT - Forum

- If you run into an issue or can't figure out how to do something, likely someone else also has a similar question.
- Have to sign up for a free account to post questions/replies.
- https://root-forum.cern.ch/
- FYI: The name "Rene Brun" will often show up. He is the main person behind current ROOT (ported ROOT from FORTRAN to C++).