ROOT Tips & Tricks for Beginners

Ole Hansen

Jefferson Lab

Software Carpentry Workshop
Jefferson Lab
May 21–23, 2018

Download this talk:
https://redmine.jlab.org/documents/31

Brief Introduction

ROOT in a Nutshell

- ROOT = Framework for large scale data handling
- Features
  - efficient data storage, access and query (petabytes!)
  - advanced statistical analysis
  - scientific visualization
  - simulation support: detector geometry, event display
  - parallel query engine (PROOF)
  - interactive C++11 interpreter
  - . . . and many more
- Open Source. GNU LGPL. ∼2M lines of code.
- Supported by ∼10 staff, primarily at CERN and Fermilab
- Tens of thousands of users in high-energy and nuclear physics, and elsewhere (*e.g.* finance)

This talk covers the current version, ROOT 6

Large Collection of Libraries

Graphics & Visualization

- 1D, 2D and 3D histograms
- Large variety of 2D/3D plot types
- Can save graphics in many formats (*e.g.* PDF, JPEG, PNG)

Minimization

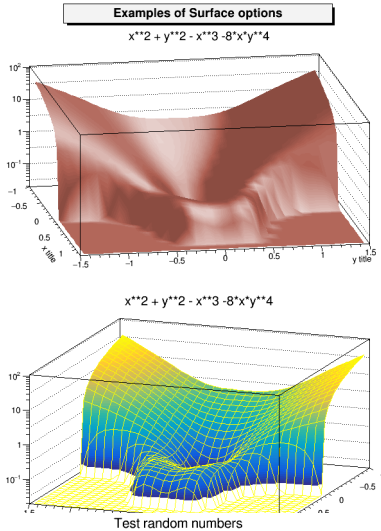
- Extensive library of fitting and minimization algorithms
- Widely used in data analysis and significance testing

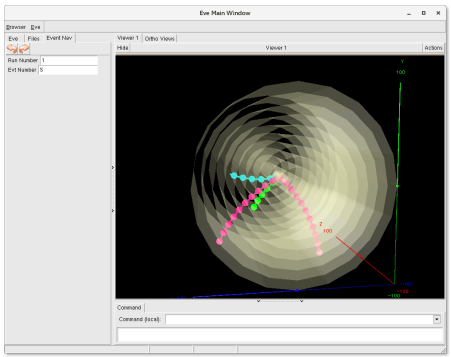
Mathematical Functions

- TMath: commonly used math functions, constants, statistics
- ROOT::Math (MathMore): very large collection of special functions, interface to GNU Scientific Library (GSL)

Geometry Toolkit and Event Display

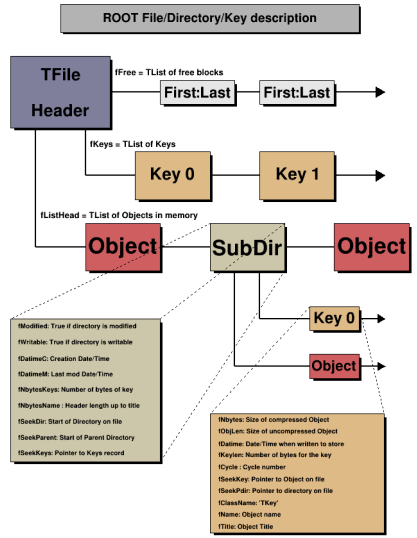
- Detailed description of detector geometry and materials
- Used extensively in simulations and event reconstruction
- 3D visualization of geometry, detector hits and tracks

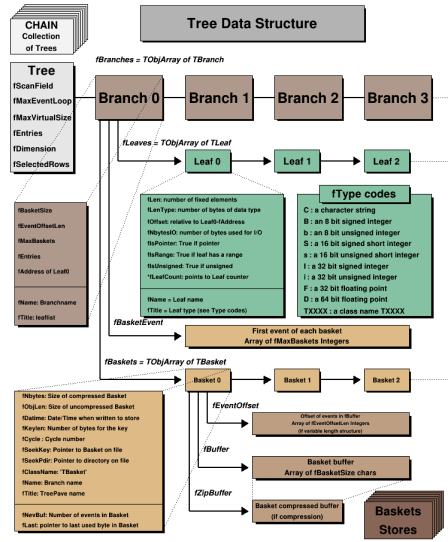




The Big Thing: C++ Object I/O and Persistency

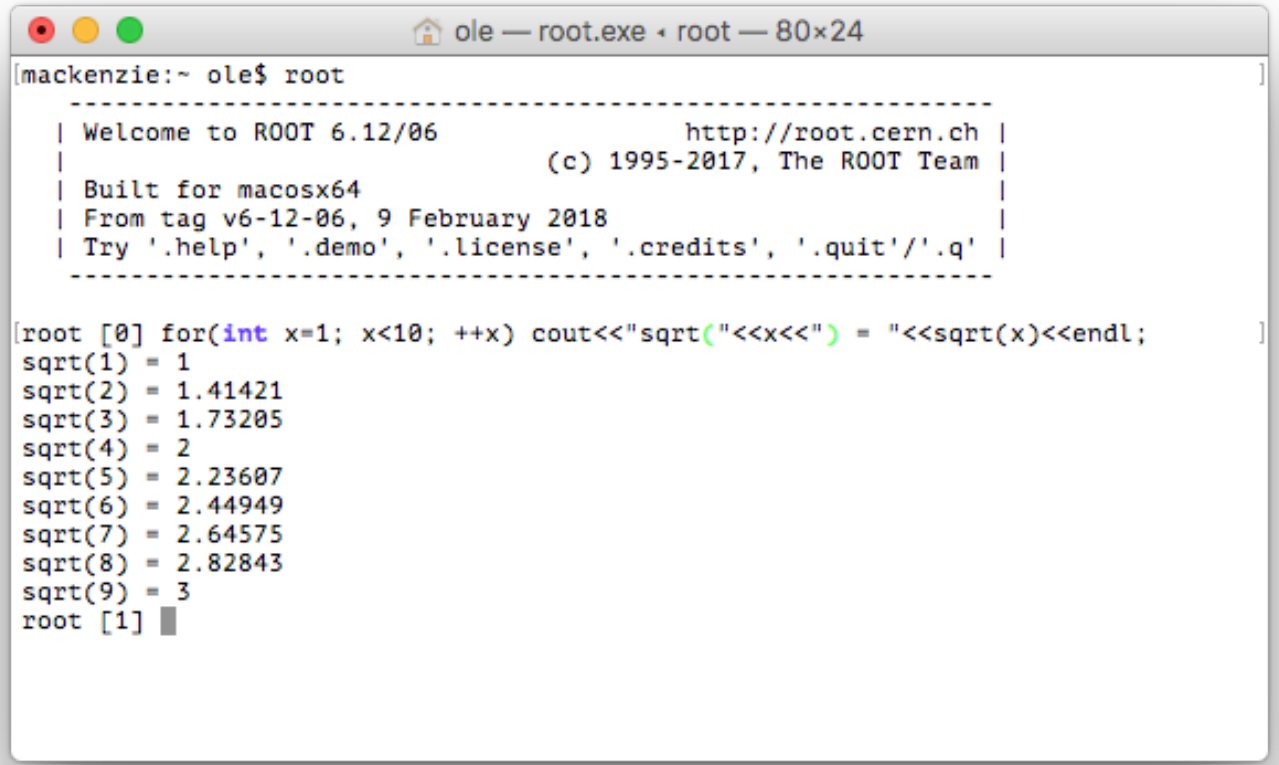
- ROOT lets you write C++ objects to file
  - Extraordinary: impossible in native C++!
  - Achieved with serialization of objects based on C++ reflection, made possible by class dictionaries generated with Cling
- Can write single objects, collections (containers), entire object trees
  - Simple interface for all ROOT objects: `obj->Write()`
- Evolution from flat n-tuples used in low and medium-energy physics
- Cornerstone for the storage of experimental data in HEP. Used for storage of hundreds of petabytes at LHC





The Other Big Thing: The C++ Interpreter

- C++11 interpreter: `Cling` (based on LLVM/Clang)
- Interactive interface to all of ROOT
- Just-in-time compilation
- Allows rapid prototyping and testing
- Same language for scripting and compiled code



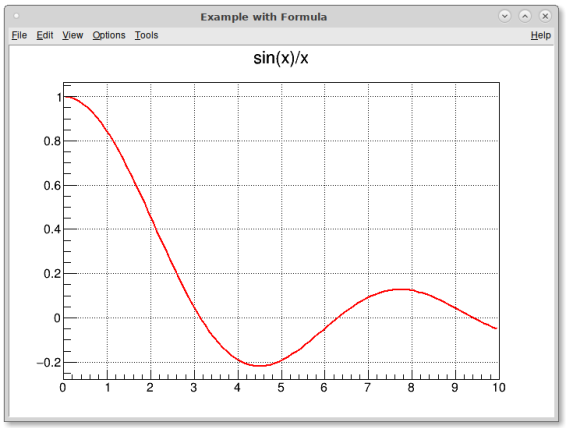
Python Bindings

- Access to all ROOT classes from Python (although in a rather C++-ish way)
- May also call Python subshell from ROOT prompt
- More information: `https://root.cern.ch/pyroot`

ole@archie:~

File Edit View Search Terminal Help

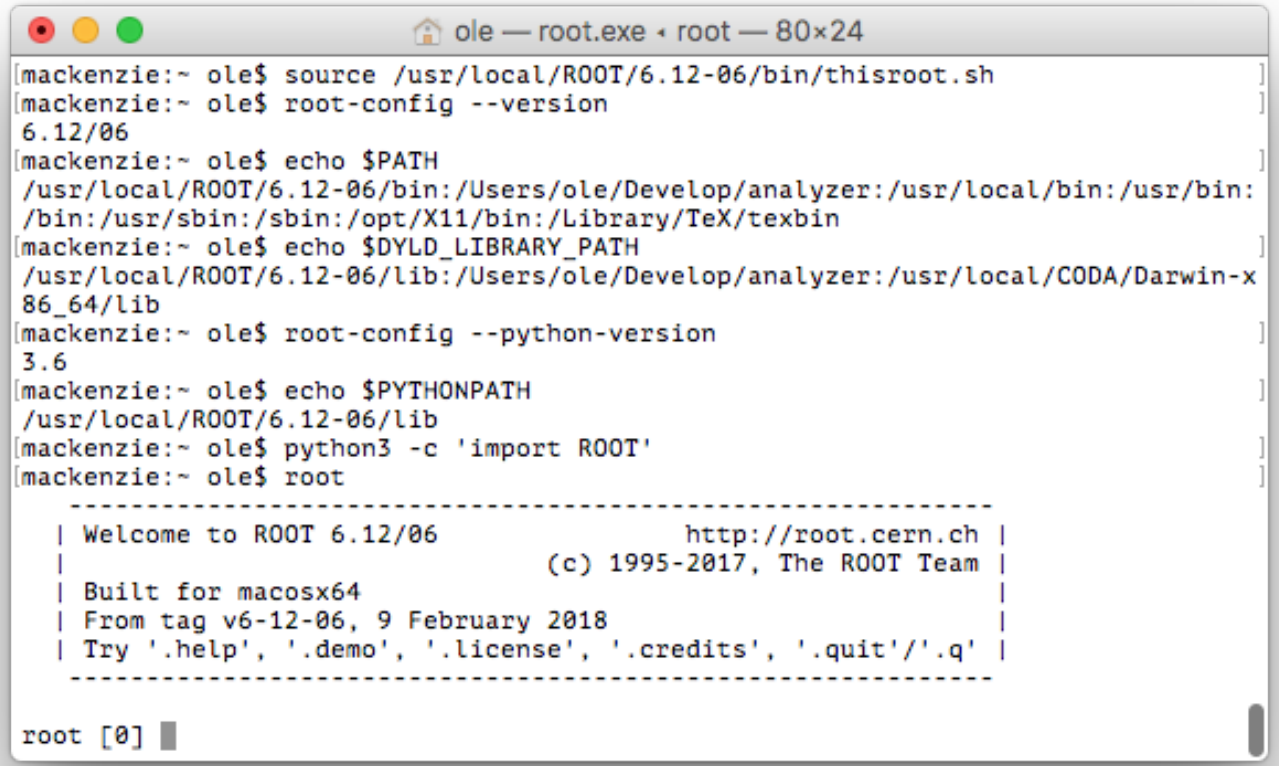
```
[ole@archie ~]$ python
Python 3.6.5 (default, May 11 2018, 04:00:52)
[GCC 8.1.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import ROOT
>>> c1 = ROOT.TCanvas( 'c1', 'Example with Formula', 200, 10, 700, 500 )
>>> type(c1)
<class 'ROOT.TCanvas'>
>>> c1.SetGridx()
>>> c1.SetGridy()
>>> f1 = ROOT.TF1( 'f1', 'sin(x)/x', 0, 10 )
>>> f1.Draw()
>>> quit()
[ole@archie ~]$ root -l
root [0] TPython::Exec( "import math" );
root [1] TPython::Exec( "x = math.sqrt(2)" );
root [2] TPython::Exec( "print(x)" );
1.4142135623730951
root [3] TPython::Prompt();
>>> print(x)
1.4142135623730951
>>>
root [4]
```



Using ROOT

ROOT Setup

- Installation guide
  `https://redmine.jlab.org/projects/podd/wiki/ROOT_Installation_Guide`
- If necessary, source the setup script `thisroot.{sh,csh}`
- Alert: the binary releases from CERN only work with Python 2.7



Getting the support files

Please clone the Git repository for this tutorial

```
$ git clone https://github.com/hansenjo/ROOTtutorial.git
$ cd ROOTtutorial
$ ls
Makefile            line.dat            make_data_numpy.py*
README.md           make_2D.C           make_histos.C
add42.C             make_data.py*       make_prof.C
read_fit_data.py*
```

Exercise 1: ROOT Command Line Basics

- May issue any C++11 statement, plus a few special ones (see later)
- May omit trailing ";" → return value printed
- May omit type specification → treated like "auto"
- Typing just the name of a defined variable prints it

Defining variables

```
$ root -l   # start up without logo (banner/splash screen)
root [0] int i = 10
(int) 10
root [1] float e = 2.718;
root [2] e
(float) 2.71800f
root [3] pi = 3.141592653
(double) 3.1415927
```

Exercise 2: ROOT Command Line Arithmetic

ROOT as a (hyper-charged) pocket calculator

```
root [4] 5+7*(3-2)
(int) 12
root [5] sqrt(2)
(double) 1.4142136
root [6] double angle = 45.;
root [7] sin(angle * TMath::DegToRad())
(double) 0.70710678
root [8] TMath::Erf(1)
(Double_t) 0.84270079
root [9] cout << setprecision(16) << TMath::Pi() << endl;
3.141592653589793
```

Exercise 3: Functions in ROOT

Working with functions (aka ROOT as a graphing calculator)

```
// Define a function with two parameters over range [0,10]
root [0] f1 = new TF1("f1","[0]*sin([1]*x)/x",0,10);

// Need to set the parameters (they default to zero)
root [1] f1->SetParameter(0,1.5);
root [2] f1->SetParameter(1,-1.);

// Draw the function
root [3] f1->Draw()

// Change plotting range
root [4] f1->SetMaximum(1.);

// Get function value, derivative, integral
root [5] f1->Eval(4.45)
(double) 0.32554148
root [6] f1->Derivative(4.45)
(double) 0.014278938
root [7] f1->Integral(0,5)   # requires root-mathmore
(double) -2.3248969

// What's the function's value at 0? Hmm ...
```

Exercise 4: C++11 in ROOT

C++11 in ROOT: a few simple examples

```
// Initialize a std::vector: initializer list
root [1] vector<double> dvars {3.45, 1.5, 9.91, 6.28, -2.718}
(std::vector<double> &) { 3.45000, 1.50000, 9.91000, 6.28000, -2.71800 }

// Much simpler looping over containers (vectors etc.)
root [2] for( auto x : dvars ) cout << x << ", "; cout << endl;
3.45, 1.5, 9.91, 6.28, -2.718,

// STL algorithms. Not C++11, but they now actually work in ROOT:
root [3] std::sort(dvars.begin(), dvars.end());
root [4] dvars
(std::vector<double> &) { -2.71800, 1.50000, 3.45000, 6.28000, 9.91000 }

// Define functions on the fly: lambda expressions
root [5] std::sort(dvars.begin(), dvars.end(),
  [](double a, double b) {return b<a;} );

root [6] for( auto x : dvars ) cout << x << ", "; cout << endl;
9.91, 6.28, 3.45, 1.5, -2.718,
```

Exercise 5: Macros/Scripts

add42.C

```
int add42(int value) {
  return value+42;
}
int add53(int value) {
  return value+53;
}
```

Running and Loading a Macro/Script

```
// Run plain script. Runs function with same name as the file
root [1] .x add42.C(11)
(int) 53

// Load, then execute
// In this way, you can use more than one function defined in the script
root [2] .L add42.C
root [3] add42(11)
(int) 53
root [4] add53(12)
(int) 65
```

Exercise 6: Compiling Macros/Scripts

Compiling a macro/script

```
// Compile on the fly -- much faster for bigger scripts
root [0] .L add42.C+
root [1] add42(11)
(int) 53

// Reuse compiled script -- rebuilt only if changed
// Execute directly
root [0] .x add42.C+(11)
(int) 53

// Load, then execute
root [0] .L add42.C+
root [1] add42(11)
(int) 53

// Force recompilation
root [2] .L add42.C++
```

Exercise 7: Other Useful Commands

Control commands

```
// Quit ROOT ;)
root [0] .q

// Get help on commands
root [0] .?
root [1] .help

// Run shell command
root [2] .!pwd
root [3] .!cat add42.C

// Redirect output to a file (behaves slightly different w/older ROOTs)
root [4] .> output.txt   // overwrites
cout << TMath::C() << endl;
.>                       // stops redirection
root [7] .!cat output.txt
root [5] cout << TMath::C() << endl;
2.99792e+08
root [6] .>
root [8] .>> output.txt  // appends if file exists
(your pick)
```

Graphs, Fitting

Exercise 8: Plotting and Fitting Data (Python Example)

Create data sets

```
ROOTtutorial $ ./make_data.py mydata.dat 25
====== Wrote 25 points to mydata.dat
ROOTtutorial $ ./make_data_numpy.py numpy.dat 25
====== Wrote 25 points to numpy.dat
```

Read data, plot and fit using **TGraph**

```
ROOTtutorial $ ./read_fit_data.py mydata.dat
====== Reading mydata.dat
====== 25 data points read into arrays x and y
====== Plotting data
====== Fitting data

****************************************
Minimizer is Minuit / Migrad
Chi2      =      31.7084
NDf       =           23
Edm       =  6.68367e-22
NCalls    =           32
p0        =    -0.398806   +/-   0.484114
p1        =      1.18761   +/-   0.032565
====== Press Enter to exit...
```

Exercise 9: TGraph C++ Example

Create data sets

```
root [0] gr = new TGraph("line.dat");
root [1] gr->SetMarkerStyle(20);
root [2] gr->Draw("AP")
root [3] line = new TF1("line","[0]+[1]*x",-15,15);
root [4] gr->Fit(line);

****************************************
Minimizer is Minuit / Migrad
Chi2      =      46.5141
NDf       =           23
Edm       =  1.94426e-22
NCalls    =           25
p0        =      3.77461   +/-   0.284349
p1        =     -2.61215   +/-   0.0394038
```

Histograms

Exercise 10: Creating One-Dimensional Histograms

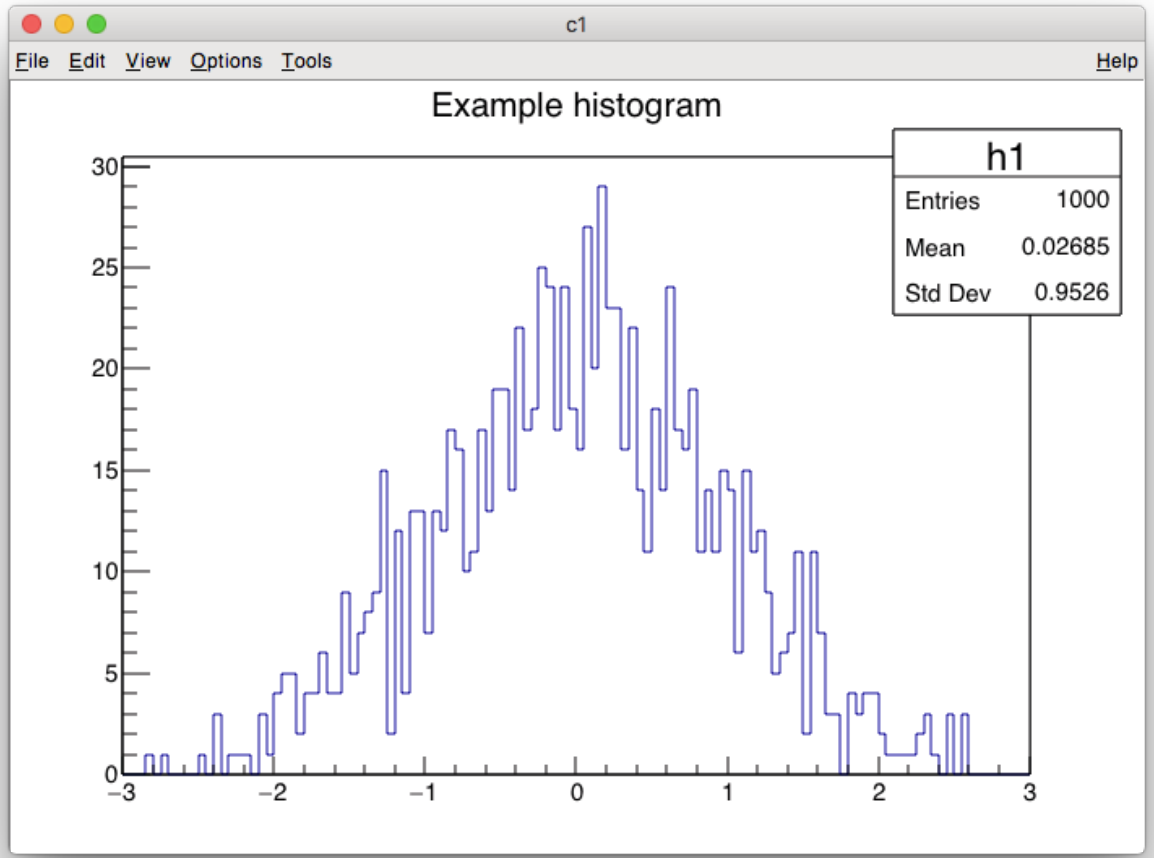
Creating & Filling Histograms

```
// Creating
//                              name  description       #bins min,max
root [0] TH1* h1 = new TH1I("h1","Example histogram",120,-3.,3.);

// Filling
   h1->Fill(x);

// Filling with 1000 random values
root [2] for ( int i=0; i<1000; ++i ) {
root (cont'ed, cancel with .@) [3]double x = gRandom->Gaus(0,1);
root (cont'ed, cancel with .@) [4]h1->Fill(x);
root (cont'ed, cancel with .@) [5]}

root [6] h1->GetEntries()
(Double_t) 1000.0000
```

Exercise 11: Plotting Histograms

Drawing any histograms

```
root [7] h1->Draw();
```



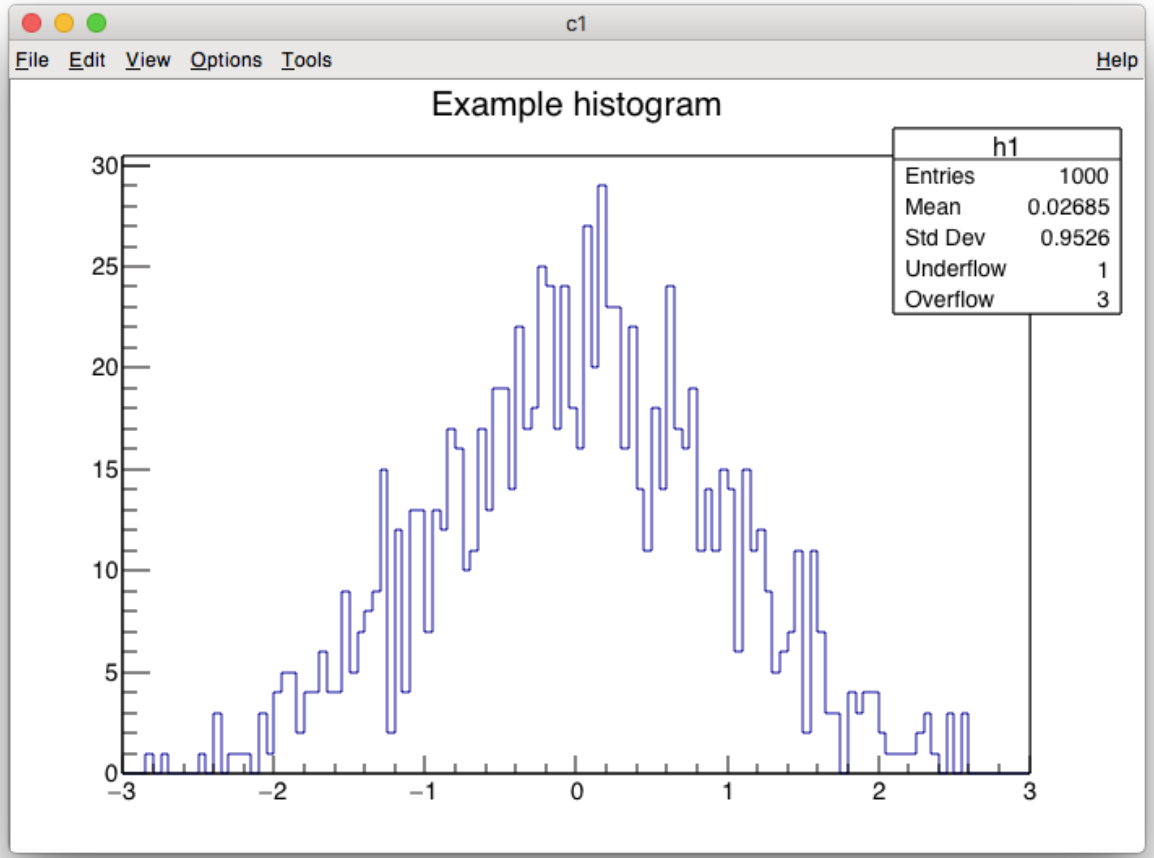
The box shows

- Name
- Number of entries
- Mean value
- Std. deviation (RMS)

Exercise 12: Plotting Histograms (cont.)

Always show over/underflow!

```
root [8] gStyle->SetOptStat(111111);
```



Note how the statistics box automatically changes

Exercise 13: Histogram Drawing Options

- Draw error bars on every bin

```
root [9] h1->Draw("E");
```

- Draw another histogram without replacing current plot. Can be used to plot one histogram on top of another

```
// Restart ROOT, then
root [0] .x make_histos.C
root [1] .ls
 OBJ: TH1I    h1    Gaussian distribution
 OBJ: TH1I    h2    Shifted narrower Gaussian
root [2] h1->Draw();
root [3] h2->SetLineColor(kRed);
root [4] h2->Draw("SAME");
```

- Display one axis with logarithmic scale

```
root [5] gPad->SetLogy();       // y-axis log scale on
root [6] gPad->SetLogy(false);  // off again
```

Exercise 14: ROOT Histograms from Python

- Python version of Exercises 11 and 12
- Tab-completion works after using a ROOT class for the first time

Creating & Filling Histograms in Python

```
$ python3
Python 3.6.5 (default, Mar 30 2018, 09:38:52)
>>> import ROOT
>>> h1 = ROOT.TH1I("h1","Python Example",120,-3.,3.)
>>> for i in range(10000) :
...         x = ROOT.gRandom.Gaus(0,1)
...         bin = h1.Fill(x)
... (RET)
>>> h1.GetEntries()
10000.0
>>> h1.Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
>>> ROOT.gStyle.SetOptStat(111111)
>>> h1.Draw("E")
```

More About Histograms

- `TH1` is the base class for *all* histogram classes (even 2D and 3D . . .)
- `TH1`, `TH2` and `TH3` are generic. Must use specialized ones (e.g. `TH1D`) for constructing.
  - TH1C: 1 byte (char) per channel, maximum bin content = 127
  - TH1S: 2 bytes (short), max. content = 32767
  - TH1I: 4 bytes (int), max. content = 2147483647
  - TH1F: 4 bytes (float), max. precision 7 digits
  - TH1D: 8 bytes (double), max. precision 14 digits
- `TH1(2,3)F` and `TH1(2,3)D` are most common, unless you either need integer precision (`TH1(2,3)I`) or need to save memory (`TH1(2,3)(C,S)`).
- There's a lot more
  - Axis properties, labels
  - Bin boundaries, content, error
  - Operations: Add, Divide
  - Weighted histograms
  - etc. See the ROOT User's Guide

Exercise 15: 2D Histograms

Creating and Filling a 2D histogram

```
// Create and fill a 2D histogram

root [0] .!cat make_2D.C
#include "TH2F.h"
#include "TRandom.h"

TH2* h2D;
void make_2D()
{
  h2D = new TH2F("h2D","2D Gaussian peak",160,-4.,4.,160,-4.,4.);
  for( int i=0; i<100000; ++i ) {
    double x = gRandom->Gaus(0,1);
    double y = gRandom->Gaus(1,2);
    h2D->Fill(x,y);
  }
}
root [1] .x make_2D.C
root [2] .ls
 OBJ: TH2F     h2D     2D Gaussian peak
```

Exercise 16: Drawing 2D Histograms

Drawing 2D histograms

```
// Scatter plot
root [3] h2D->Draw();

// Color Z
root [4] h2D->Draw("COLZ");

// Contour plot
root [5] h2D->Draw("CONTZ");

// "Lego" plot (3D)
root [6] h2D->Draw("LEGO");
```

Exercise 17: Projection Histograms

Projection and Profile Histograms

```
//  Projection of 2D histogram onto one of its axes
// This sums all y-bins with the same x value
root [7] TH1* hX = h2D->ProjectionX();
root [8] hX->Draw();

// Likewise, this sums all x-bins with the same y value
root [9] TH1* hY = h2D->ProjectionY();
root [10] hY->Draw();
```

Exercise 18: Profile Histograms

Projection and Profile Histograms

```
//  Profile histogram: Plot the mean and standard deviation
// of all y-bins with the same x value, and vice versa
root [11] .!less make_prof.C
#include "TProfile.h"
#include "TRandom.h"
TProfile *hp, *hpy;
void make_prof() {
  hp  = new TProfile("hp", "x-profile of shifted Gaussian",160,-4.,4.);
  hpy = new TProfile("hpy","y-profile of shifted Gaussian",160,-4.,4.);
  for( int i=0; i<100000; ++i ) {
    double x = gRandom->Gaus(0,1);
    double y = gRandom->Gaus(1,2);
    hp->Fill(x,y); hpy->Fill(y,x);
  }
}
root [12] .x make_prof.C
root [13] hp->Draw();
root [14] hpy->Draw();
```

Exercise 19: Fitting Histograms

Fitting Histograms

```
// Basic
root [0] .x make_histos.C
root [1] h1->Fit("gaus");
root [2] gStyle->SetOptFit(1111);

// More complex (identical to above, but flexible now)
root [3] f1 = new TF1("f1","[0]*TMath::Gaus(x,[1],[2])");
root [4] f1->SetParameters(1,0,1);
root [5] h1->Fit(f1);

// Retrieving fit results
root [6] ff = h1->GetFunction("f1");
root [7] ff->GetParameter(0)
root [8] ff->GetParError(0)

// More detailed information about the fit
root [9] r = h1->Fit(f1,"S");
(TFitResultPtr &) ...

// Can now retrieve full fit information from r
root [10] C = r->GetCorrelationMatrix();  // note operator "->" !
root [11] C.Print();
```

Exercise 20: More Fitting Options

More Fitting Options

```
// Restrict fit range
root [12] h1->Fit("gaus","","",-1.5,1.5);

// Fit a second function to a different part of the histogram
// with the "+" option
root [21] h1->Fit("expo","+","",2,4);

// Verbose - more detail than you ever want to know
root [22] h1->Fit("gaus","V");

// Likelihood fit
root [23] h1->Fit("gaus","L");

// Pass histogram plotting options as 3rd parameter
root [24] c1->Clear();
root [25] h1->Fit("gaus","L","E");
```

Files and Trees

Exercise 21: Opening and browsing ROOT files

ROOT files and the browser

```
// Start ROOT
$ root

// Open a remote ROOT file via http (could also download first)
root [0]
 f = TFile::Open("http://hallaweb.jlab.org/podd/download/g2p_3132_example.root")
// (Ignore warnings about "no dictionary for class THa...")

// List the file contents. "KEY": on disk, "OBJ": in memory
root [1] .ls
TWebFile**      http://.../g2p_3132_example.root
 TWebFile*      http://.../g2p_3132_example.root
  KEY: THaRun   Run_Data;2     g2p run 3132 optics data
  KEY: TTree    T;1     Hall A Analyzer Output DST
  KEY: TH2F     Llsl;1  L u1 slope vs. local slope
  KEY: TH2F     Llsz;1  L u1 slope vs. cluster size

// Draw saved histogram:
root [2] Llsl->Draw("COLZ")

// Start up the user-friendly ROOT browser
root [3] new TBrowser;
```

Exercise 22: Plotting Tree Variables

A very powerful method to analyze flat trees is the `TTree::Draw` method

Working with ROOT Trees (flat ones, at least)

```
// Basic histogram
root [4] T->Draw("L.vdc.u1.wire");

// Applying a selection
root [5] T->Draw("L.vdc.u1.wire","L.vdc.u1.wire>100");

// Avoidng automatic binning: defining histograms on the fly
root [6] T->Draw("L.vdc.u1.wire>>hw(368,0,368)","L.vdc.u1.wire>100");
```

Exercise 23: Plotting correlations

Plotting 2D correlations

```
// 2D histograms: Inspecting correlations
root [7] T->Draw("L.vdc.u1.time:L.vdc.u1.rawtime");

// Using histogram drawing options (see earlier)
root [8] T->Draw("L.vdc.u1.time:L.vdc.u1.rawtime","","CONTZ");

// This one looks interesting, but doesn't show quite what you think it does
root [9] T->Draw("L.vdc.u1.wire:L.vdc.u1.slope","abs(L.vdc.u1.slope)<1");

// CAUTION: u1.wire and u1.slope are NOT parallel arrays!
root [10] T->Draw("Ndata.L.vdc.u1.wire:Ndata.L.vdc.u1.slope","","LEGO");
root [11] gPad->SetLogz()

// The 2D plot shows only elements with indices i=j, which may or
// may not be meaningful

// Feel free to play around more with these data
```

Thanks!