What is ROOT?
Why do we use it?

Answer:

ROOT does what physicists do:

It makes plots.
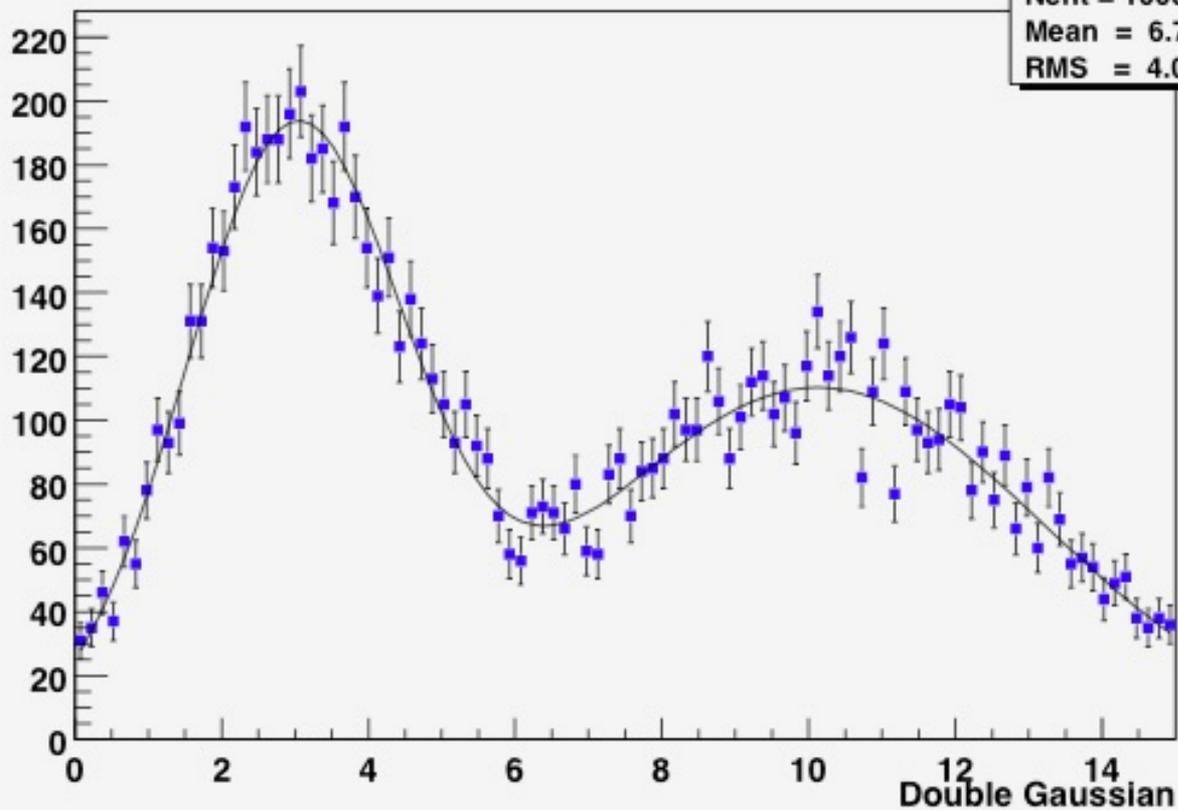
sin(x)*sin(y)/(x*y)
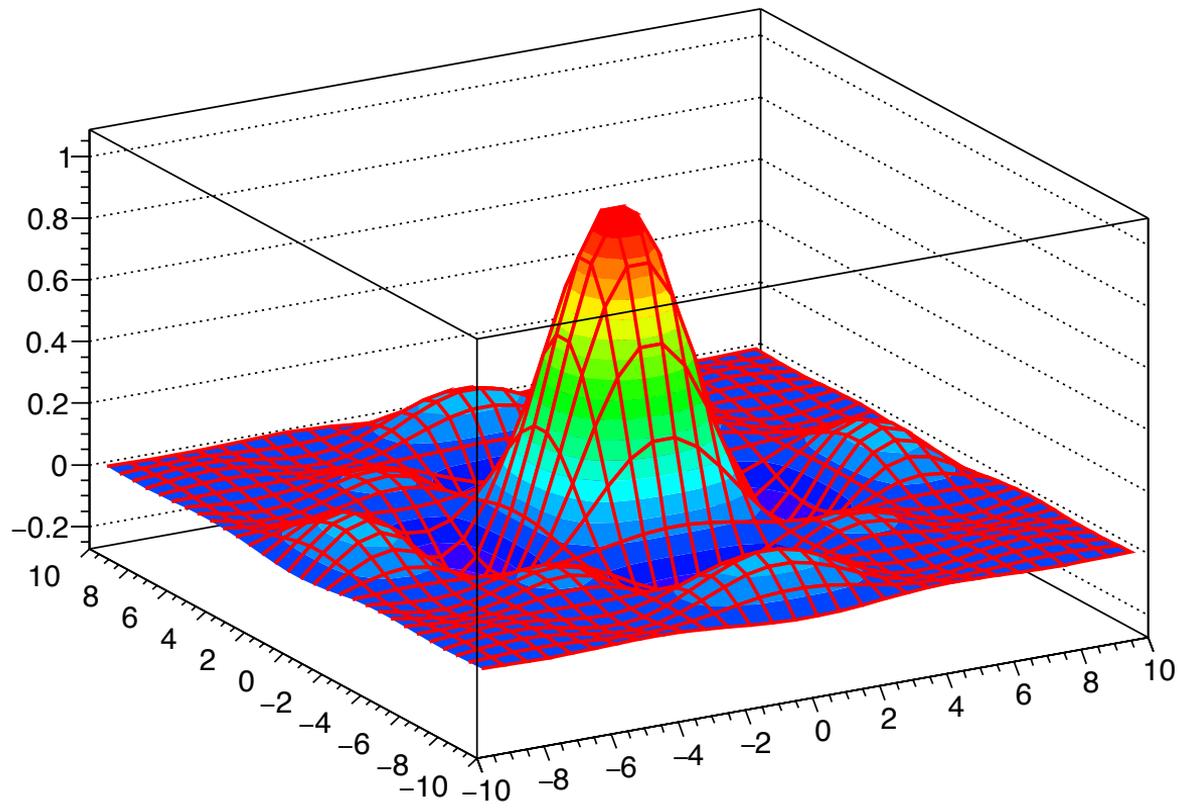
Number of charged atoms in 'Nights in the Gardens of Spain'
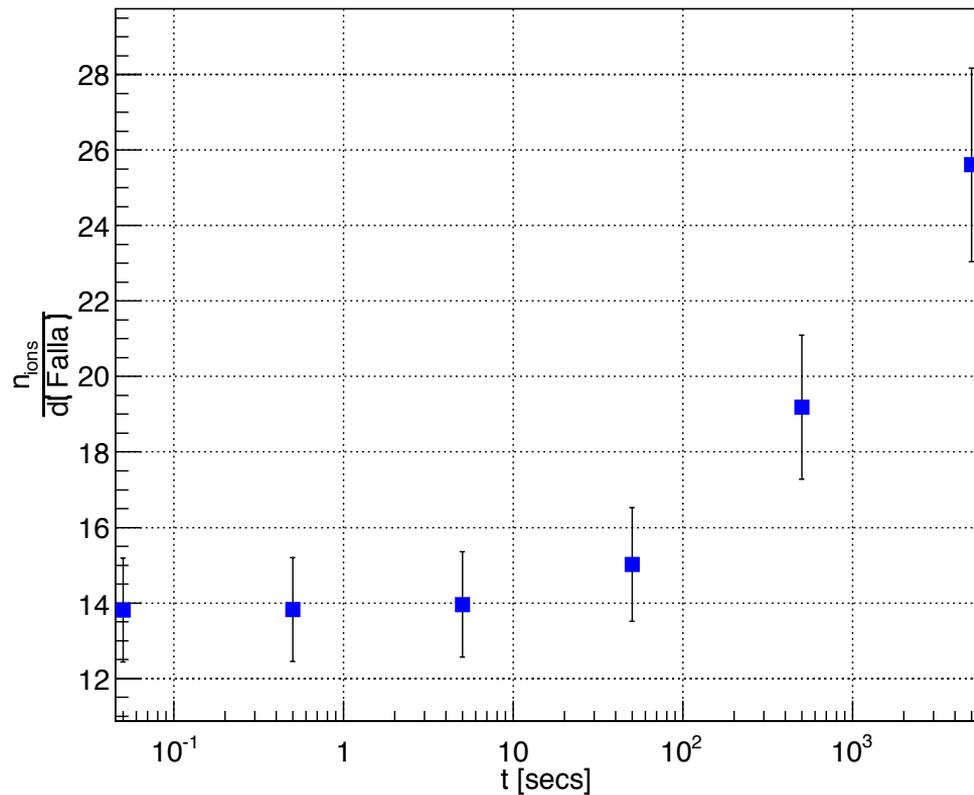
Can you spot the pun in this plot?

The typical analysis task that you will be asked to do:

Take variables in an ntuple, perform some computations, and make histograms.

So what is a histogram, what is an ntuple, and how do we perform the computations?

Anatomy of a histogram

Sample histogram

Example
Nent = 10000
Mean = -0.005034
RMS = 0.974

Number of events

Measurement
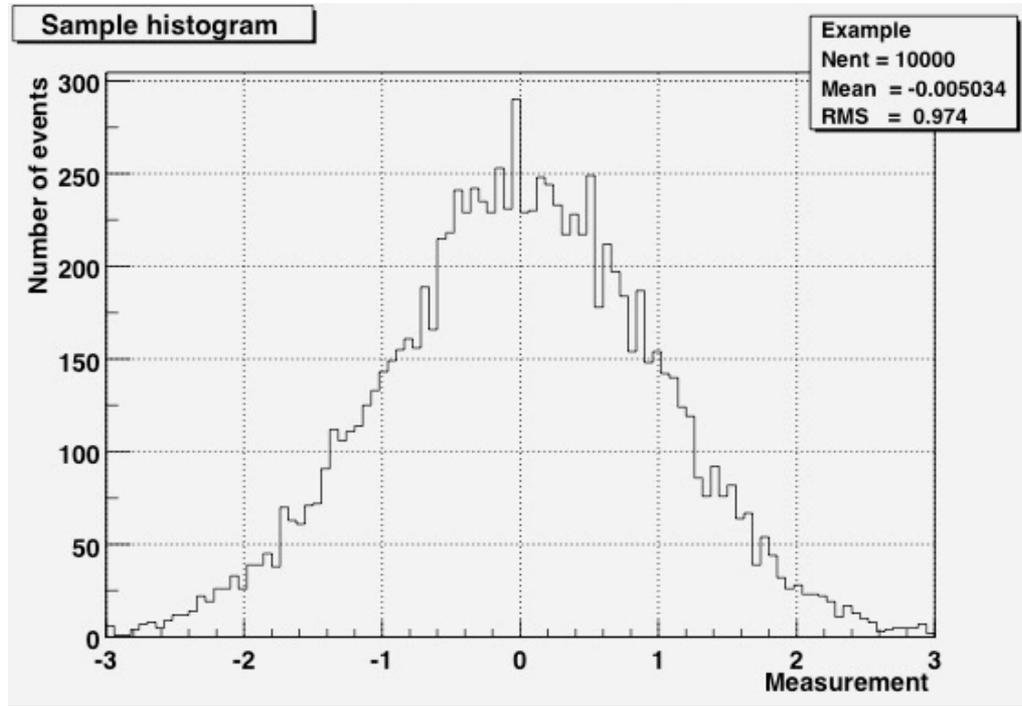
Properties of a histogram

- Name or Identifier
- Title (to be displayed on plot)
- Number of bins
- Lower bin limit
- Upper bin limit

A ROOT command that might be used to define this histogram:

```
TH1F myPlot("Example","Sample histogram",100,-3,3)
```
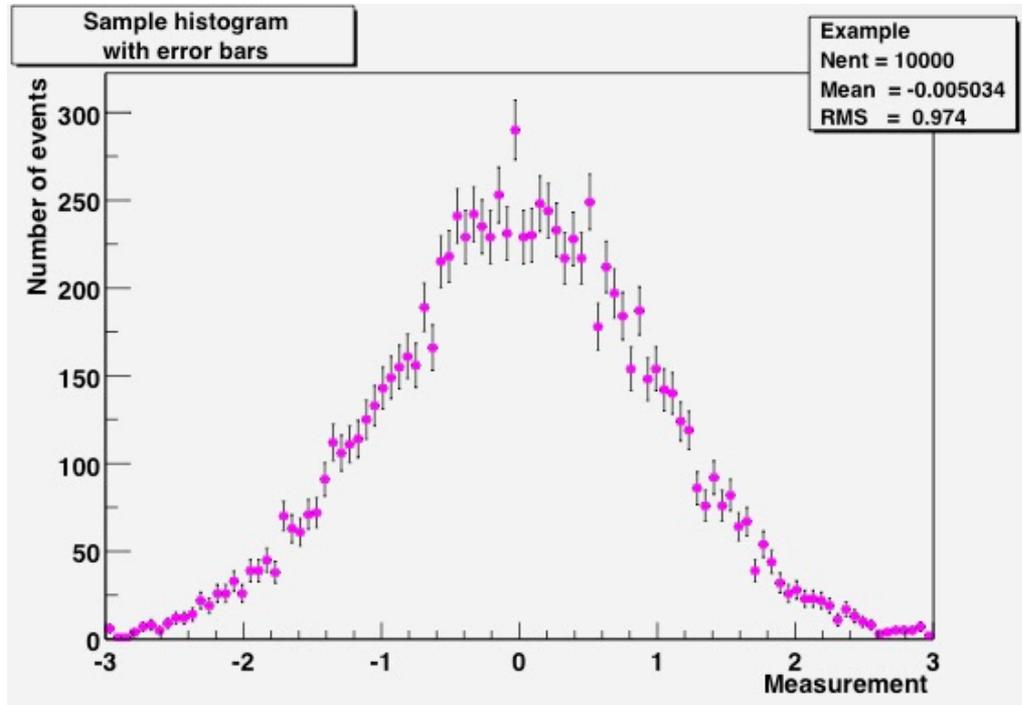
Don't forget the errors!

For simple histograms, the error in one bin is the square root of the number of events in that bin.

There's an art to histogram design...
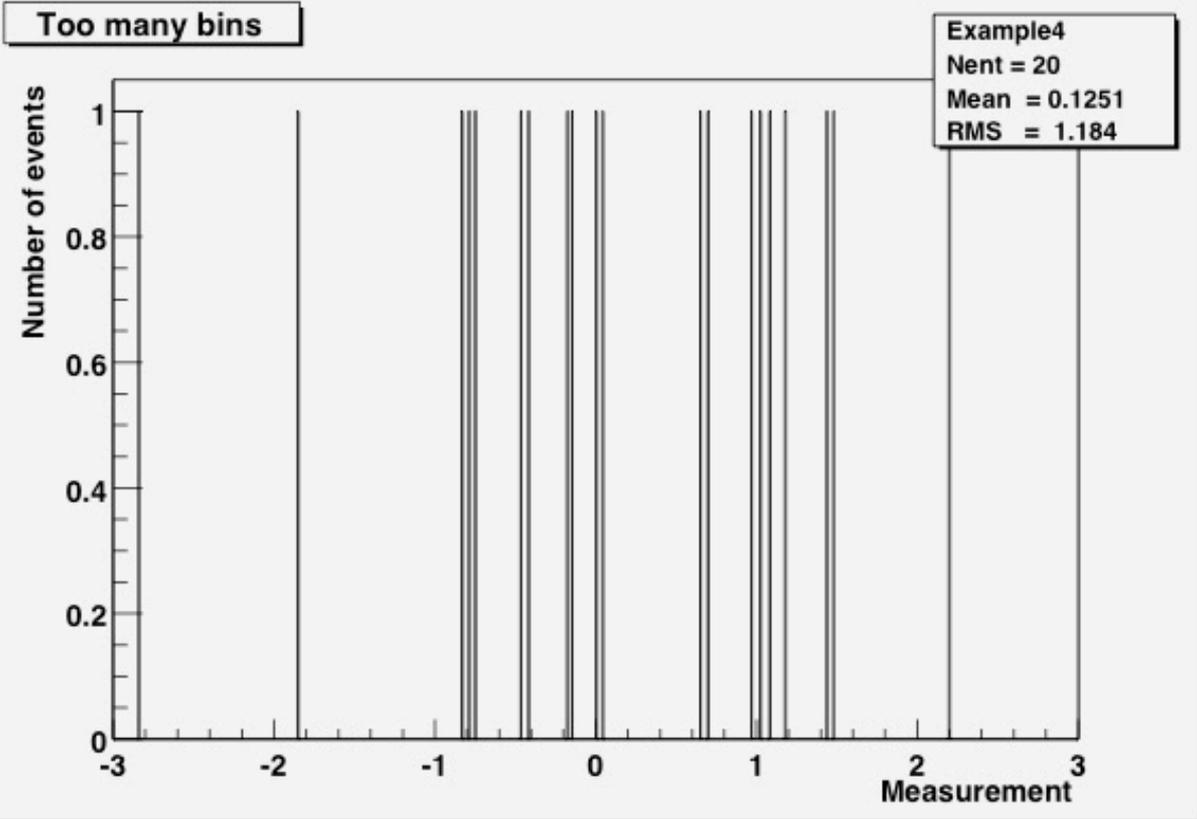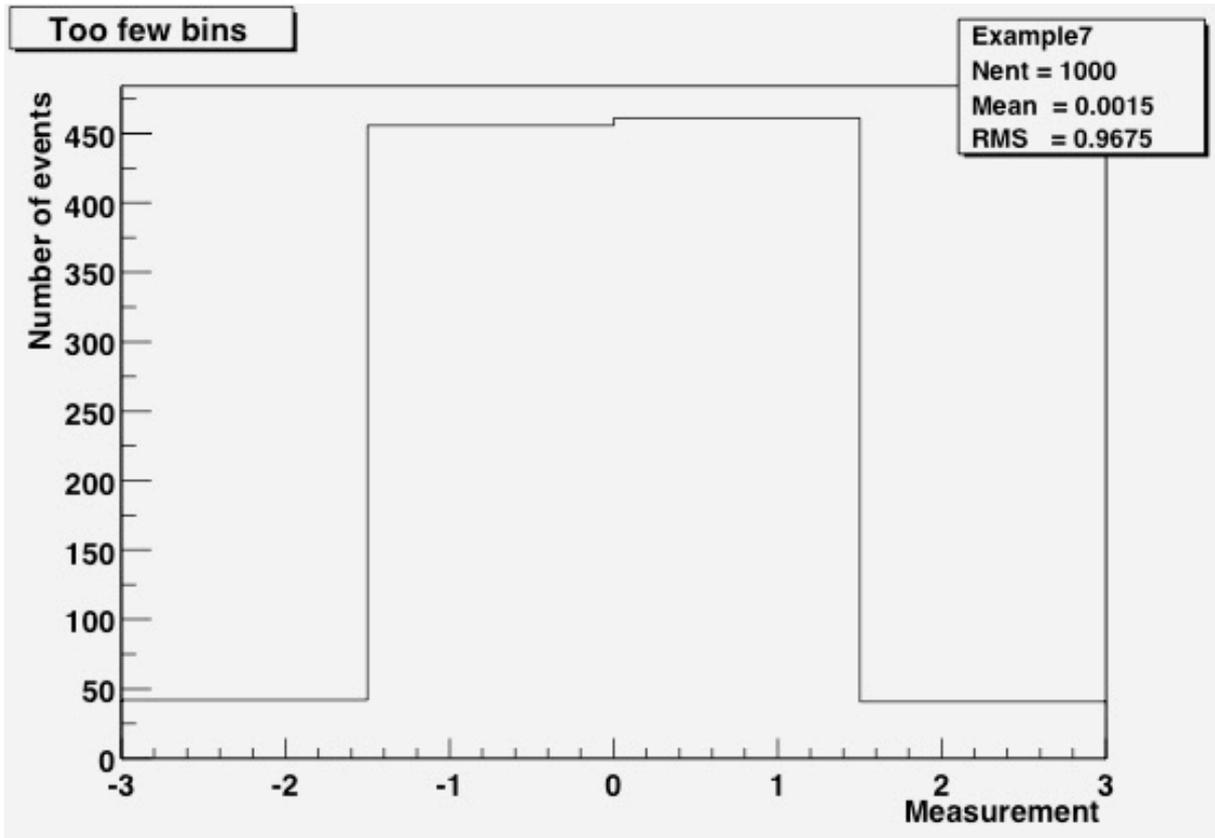
I found a claim that an optimal number of bins is $\sim \sqrt[3]{N}$, where N is the number of entries in the histogram. I have not substantiated this on my own.

# Anatomy of an ntuple (a simple form of a ROOT Tree)

<-- Entries

Row	event	ebeam	px	py	pz
0	0	150.14	14.33	-4.02	143.54
1	1	149.79	0.05	-1.37	148.60
2	2	150.16	4.01	3.89	145.69
3	3	150.14	1.46	4.66	146.71
4	4	149.94	-10.34	11.07	148.33
5	5	150.18	17.08	-12.14	143.10
6	6	150.02	5.19	7.79	148.59
7	7	150.05	7.55	-7.43	144.45
8	8	150.07	0.23	-0.02	147.78
9	9	149.96	1.21	7.27	146.99
10	10	149.92	5.35	3.98	140.70
11	11	149.88	-4.63	-0.08	147.91

An ntuple is an ordered list of numbers.

A ROOT Tree can be an ordered list of any collections of C++ objects.

Probably you'll only be asked to work with ntuples this summer, but in the Expert section of the tutorial you can see what it's like to work with a ROOT Tree.

Why ROOT?

- ## It knows about ntuples and histograms

  and 4-vectors and object persistency and schema evolution
  and detector geometry and Feynmann diagrams
  and linear algebra and function-fitting and multi-variable analysis and…

- ## It can handle large volumes of data

  millions of physics events; files of gigabytes->terabytes in size; complex
  file structures; multi-threaded and batch processing

- ## Multi-platform (Windows, Mac, many UNIX flavors)

- ## It's free.

- Some python-based alternatives to ROOT are beginning to show up (e.g.,
  uproot, coffea), but most such alternatives are wrappers around ROOT.

- HDF5 can also handle large amounts of heterogenous structured data, but
  (I think) it's organized as "datasets" as opposed to complex C++ object
  hierarchies.

But...

- ROOT is open-source, with a complicated design history.

- User-interface issues and documentation are often neglected.

- It's not a pre-packaged "app." ROOT is not easy to install.

- "ROOT is not your friend."

- ROOT is pretty much only used in high-energy physics.

- You have to know some C++ in order to use ROOT effectively, in order to perform computations.

- What does C++ look like? Well...

```cpp
#define AnalyzeHistogram_cxx

#include "AnalyzeHistogram.h"
#include <TH2.h>
#include <TStyle.h>

//******** Definition section *********
TH1* chi2Hist = 0;

void AnalyzeHistogram::Begin(TTree * /*tree*/)
{
  TString option = GetOption();

  //******** Initialization section *********
  chi2Hist = new TH1F("chi2","Histogram of Chi2",100,0,20);
  chi2Hist->GetXaxis()->SetTitle("chi2");
  chi2Hist->GetYaxis()->SetTitle("number of events");
}

void AnalyzeHistogram::SlaveBegin(TTree * /*tree*/)
{
  TString option = GetOption();
}

Bool_t AnalyzeHistogram::Process(Long64_t entry)
{
  //******** Loop section *********
  tree1->GetEntry(entry);
  chi2Hist->Fill(chi2);

  return kTRUE;
}

void AnalyzeHistogram::SlaveTerminate()
{}

void AnalyzeHistogram::Terminate()
{
  //******** Wrap-up section *********
  chi2Hist->Draw();
}
```

If you prefer Python, there's pyroot

```python
import ROOT

# Open the file.
myfile = ROOT.TFile( 'experiment.root' )

# Retrieve the n-tuple of interest.
mychain = ROOT.gDirectory.Get( 'tree1' )
entries = mychain.GetEntriesFast()

# Create a 2D histogram
myHist = ROOT.TH2D("hist2D","chi2 vs ebeam",100,0,20,100,149,151)
myHist.GetXaxis().SetTitle("chi2")
myHist.GetYaxis().SetTitle("ebeam [GeV]")

for jentry in xrange( entries ):

    # Copy next entry into memory and verify.
    nb = mychain.GetEntry( jentry )
    if nb <= 0:
        continue

    # Fetch the variables from the entry and fill the histogram.
    chi2 = mychain.chi2
    ebeam = mychain.ebeam
    myHist.Fill(chi2,ebeam)

# Display the scatterplot.
myHist.Draw()
```

Web Links

All the documents you've seen (and will see) during these tutorial sessions can be found here (this is the *only* link you need to write down from this lecture):

https://www.nevis.columbia.edu/~seligman/root-class/

ROOT and C++ links, including links to reference books on C++ and statistics, can be found at:

http://www.nevis.columbia.edu/~seligman/root-class/links.html

The Hands-on Course

Basic Data Analysis using ROOT

ROOT basics

Over the next 2-3 days, you will learn how to:
- look up ROOT command references
- plot a function
- histogram a variable
- fit a histogram
- get a variable from an n-tuple
- apply cuts
- do a quick study using TreeViewer (optional)
- create C++ or python code for an n-tuple
- use the Jupyter notebook server for quick coding

-- but not necessarily in this order!

Wait… there's more!

The written tutorial includes intermediate topics, advanced topics, and an appendix. They're there if you have the time during the tutorial sessions, or for reference later on as you work with ROOT.

Intermediate topics include:
- Advanced histogram techniques
- JupyterLab
- Dataframes

Advanced ROOT, and becoming a ROOT expert:
- Creating an x-y plot
- Working with large numbers of histograms
- Extracting your own ntuples

The appendix includes:
- Statistics jargon that physicists use
- Programming tips
- Batch systems

A Brief ROOT Demonstration

- Using the command line
- Using the notebook server