What is ROOT?
Why do we use it?

Answer:

ROOT does what physicists do:

It makes plots.
Another function to be fit

hlst2
Nint = 10000
Mean = 6.714
RMS = 4.012
Can you spot the pun in this plot?
The typical analysis task that you will be asked to do:

Take variables in an \textit{n-tuple}, perform some computations, and make \textit{histograms}.

So what is a \textit{histogram}, what is an \textit{n-tuple}, and how do we perform the computations?
Anatomy of a histogram

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 create this histogram:

```
TH1F hist("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...
Example:
Nent = 1000
Mean = 0.0015
RMS = 0.9675

Too few bins
### Anatomy of an n-tuple (a simple form of a ROOT Tree)

<table>
<thead>
<tr>
<th>Row</th>
<th>event</th>
<th>ebeam</th>
<th>px</th>
<th>py</th>
<th>pz</th>
<th>zv</th>
<th>chi2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>150.14</td>
<td>14.33</td>
<td>-4.02</td>
<td>143.54</td>
<td>22.26</td>
<td>0.94</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>149.79</td>
<td>0.05</td>
<td>-1.37</td>
<td>148.60</td>
<td>0.61</td>
<td>1.02</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>150.16</td>
<td>4.01</td>
<td>3.89</td>
<td>145.69</td>
<td>16.57</td>
<td>0.89</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>150.14</td>
<td>1.46</td>
<td>4.66</td>
<td>146.71</td>
<td>11.47</td>
<td>1.02</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>149.94</td>
<td>-10.34</td>
<td>11.07</td>
<td>148.33</td>
<td>0.37</td>
<td>0.85</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>150.18</td>
<td>17.08</td>
<td>-12.14</td>
<td>143.10</td>
<td>22.09</td>
<td>0.90</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>150.02</td>
<td>5.19</td>
<td>7.79</td>
<td>148.59</td>
<td>2.28</td>
<td>1.06</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>150.05</td>
<td>7.55</td>
<td>-7.43</td>
<td>144.45</td>
<td>21.40</td>
<td>0.97</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>150.07</td>
<td>0.23</td>
<td>-0.02</td>
<td>147.78</td>
<td>6.96</td>
<td>0.93</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>149.96</td>
<td>1.21</td>
<td>7.27</td>
<td>146.99</td>
<td>7.17</td>
<td>1.02</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>149.92</td>
<td>5.35</td>
<td>3.98</td>
<td>140.70</td>
<td>38.81</td>
<td>1.08</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>149.88</td>
<td>-4.63</td>
<td>-0.08</td>
<td>147.91</td>
<td>4.01</td>
<td>0.86</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>150.11</td>
<td>-1.96</td>
<td>11.46</td>
<td>147.41</td>
<td>6.76</td>
<td>1.08</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>150.02</td>
<td>-4.97</td>
<td>4.29</td>
<td>145.06</td>
<td>17.79</td>
<td>0.92</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>149.86</td>
<td>0.26</td>
<td>0.10</td>
<td>144.69</td>
<td>22.26</td>
<td>0.93</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>150.28</td>
<td>-7.50</td>
<td>1.49</td>
<td>147.64</td>
<td>7.07</td>
<td>1.02</td>
</tr>
</tbody>
</table>

An n-tuple 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 n-tuples this summer.
Why ROOT?

- It knows about n-tuples and histograms.
- It can handle large volumes of data (millions of physics events; files of gigabytes->terabytes in size).
- Multi-platform (Windows, Mac, many UNIX flavors)
- It's free.

But...

- You have to know some C++ in order to use ROOT effectively, in order to perform computations.
- What does C++ look like? Well...
#define Analyze_cxx
#include "Analyze.h"
#include "TH2.h"
#include "TStyle.h"
#include "TCanvas.h"

void Analyze::Loop()
{
    // In a Root session, you can do:
    // Root > .L Analyze.C
    // Root > Analyze t
    // Root > t.GetEntry(12); // Fill t data members with entry number 12
    // Root > t.Show(); // Show values of entry 12
    // Root > t.Show(16); // Read and show values of entry 16
    // Root > t.Loop(); // Loop on all entries

    // This is the loop skeleton
    // To read only selected branches, Insert statements like:
    // METHOD1:
    // fChain->SetBranchStatus("*",0); // disable all branches
    // fChain->SetBranchStatus("branchname",1); // activate branchname
    // METHOD2: replace line
    // fChain->GetEntry(i); // read all branches
    // by b_branchname->GetEntry(i); //read only this branch
    if (fChain == 0) return;

    Long64_t nentries = fChain->GetEntries();

    Long64_t nbytes = 0, nb = 0;
    for (Long64_t jentry=0; jentry<nentries; jentry++) {
        Long64_t ientry = LoadTree(jentry);
        nb = fChain->GetEntry(jentry); nbytes += nb;
        // if (Cut(ientry) < 0) continue;
    }
}
All the documents you've seen (and will see) during the class today can be found at:

http://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

You will learn how to:
• look up ROOT command references
• plot a function
• histogram a variable
• fit a histogram
• create C++ code for an n-tuple
• get a variable from an n-tuple
• apply cuts

-- but not necessarily in this order!

There's lots of optional material to help turn you into a ROOT expert. Try to go over as much of it as you can.
A Brief ROOT Demonstration