BES3常用软件工具

柯百謙
山西師範大學

ROOT Download & Installation

- Download from ROOT Web site http://root.cern.ch
- Binaries for Linux, MacOS and Windows
- Source files can be built with:

```
./configure
make
make install (as root)
```

- see the instructions on the Web site for building from sources

源程序编译安装：
 https://root.cern.ch/building-root

源程序下载：
root_v6.08.06.source.tar.gz 149M

- *For the training ROOT should have been already installed !*

Download   **Documentation**   **News**   **Support**

Home » Download

Downloading ROOT

Latest ROOT Releases

Pro Release 6.08/06 – 2017-03-02 **Version 5**

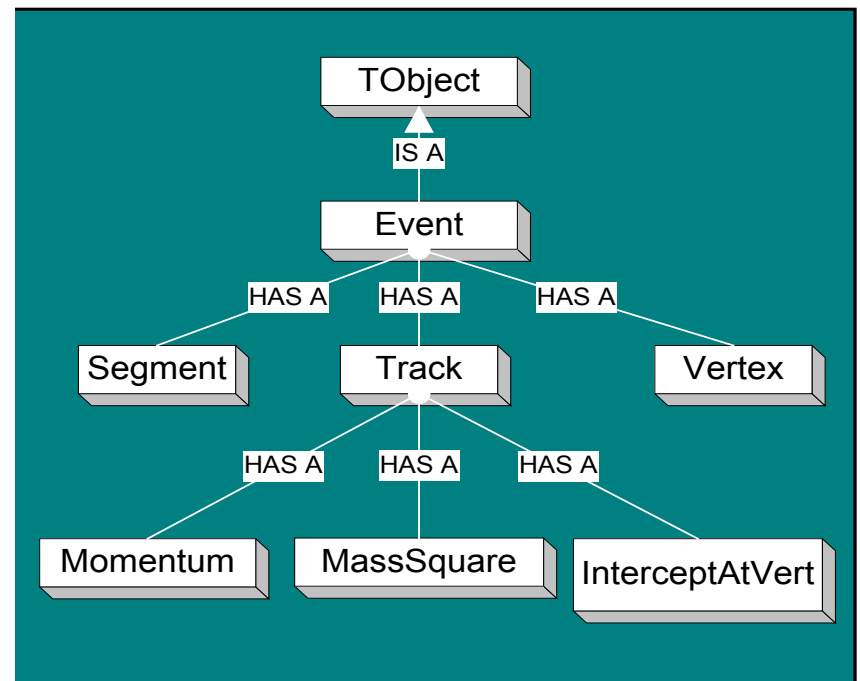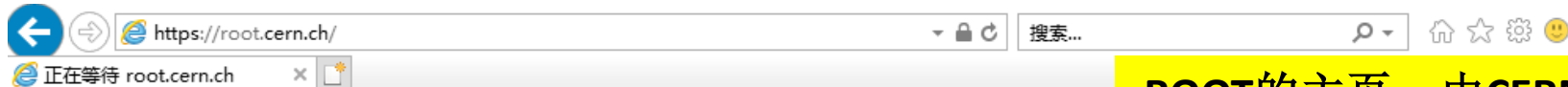**Old** Release 6.06/08 – 2016-09-01 Release 5.34/36 – 2016-04-05

Why ROOT ?

- The analysis of data coming from LHC experiments and not only requires a powerful and general toolkit
  - Visualisation
  - Statistical studies
  - Data reduction
  - Multivariate techniques

- A scalable and reliable persistency method is needed to write the data on disks and tapes.

Object Oriented Concepts

- Class: the description of a "thing" in the system
- Object: instance of a class
- Methods: functions for a class

- Members: a "has a" relationship to the class.

- Inheritance: an "is a" relationship to the class.

Introduction to ROOT

ROOT的主页，由CERN维护。

基于面向对象设计的数据分析软件框架！

Nature 522 (2015) 68-72

ROOT tutorials and test

- $ROOTSYS/tutorials: contains many example scripts. They assume some basic knowledge of C++ and ROOT

  copy them to your home directory and try!

- $ROOTSYS/test: a set of examples that represent all areas of the framework and a gold mine

  to learn some advanced examples!

Start and Quit ROOT

- To start ROOT, you can type root at the system prompt. This starts up CINT, the ROOT command line C/C++ interpreter, and it gives you the ROOT prompt (root[0]).

- To quit ROOT, type .q at the ROOT prompt

- It is possible to launch ROOT with some command line options, as shown below:

```
> root -h
Usage: root [-l] [-b] [-n] [-q] [file1.C ... fileN.C]
 Options:
-b : run in batch mode without graphics
        以批处理形式运行不显示图，远程操作需要，减少等待时间和流量。
-n : do not execute logon and logoff macros as
specified in .rootrc
-q : exit after processing command line script files
        运行完脚本文件后自动退出,非常有用，在脚本中运行root需要。
-l : do not show the image logo (splash screen)
```

Command-Line Interface

- a powerful C/C++ interpreter giving you access to all available ROOT classes, global variables, and functions via the command line.

- By typing C++ statements at the prompt, you can create objects, call functions, execute scripts, etc.

- Use up and down arrows to recall commands:
    $HOME/.root_hist

- Use emacs commands to navigate

```
root[] 1+sqrt(9)
(const double)4.00000000000000000e+00
root[] for (int i = 0; i<2; i++) cout << "Hello" << i << endl
Hello 0
Hello 1
```

Start ROOT from $ROOTSYS/Tutorials

- Execute root under $ROOTSYS/tutorials, you can start by executing the standard ROOT demos with a session like
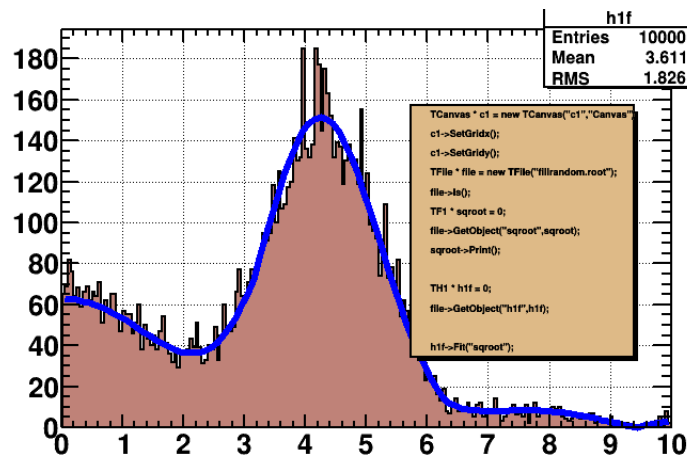
```
Root> .x demos.C
```

出一个菜单，选择运行示例！

- You can execute the standard ROOT graphics benchmark with

```
Root> .x benchmark.C
```

运行后以此运行示例，显示画出的图，并给出测试结果：

- You can try: *root <filename.C>* to check the separate macro outputs.

- More tutorials at *https://root.cern.ch/code-examples*

The ROOT Script Processor

- Un-named Scripts: script1.C

```
{
#include <iostream.h>
cout << " Hello" << endl;
float x = 3.; float y = 5.; int i = 101;
cout <<" x = "<<x<<" y = "<<y<<" i = "<<i<< endl;
}
```

```
root[] .x script1.C <enter>
```

- Named Scripts: script2.C

```
#include <iostream.h>
int run (int j=10)
{
cout << " Hello" << endl;
float x = 3.; float y = 5.; int i= j;
cout <<" x = "<< x <<" y = "<< y <<" i = "<< i << endl;
return 0;
}
```

The ROOT Script Processor

```
root [] .L script2.C
root [] .func
...
script2.C2:7 0 public: int run(int j=10);
root [] run(<tab>
int run(int j = 10)
root [] run()
 Hello
 x = 3 y = 5 i = 10
(int)0
root [] run(1)
 Hello
 x = 3 y = 5 i = 1
(int)0
```

- Change the function name to the script prefix name:

  int run(int j=10) ➔ int script2(int j=10)

  then you can execute the macro via:

```
root[] .x script2.C(8)
```

Running Code

Macro: a file that is interpreted by CINT

```
int mymacro(int value)
{
    int ret = 42;
    ret += value;
    return ret;
}
```

- Create a new file, `mymacro.C`
- Edit the file and include these above lines.
- Execute from the root prompt:

```
root [0] .x mymacro.C(42)
```

Compiling Code: ACLic

Automatic Compiler of Libraries for CINT

- Load code as shared lib, much faster:

```
root [0] .x mymacro.C+(42)
```

Use "+" instead of writing a Makefile…

- Uses the system's compiler, takes seconds
- Subsequent **.x mymacro.C+(42)** check for changes, only rebuild if needed
- Exactly as fast as *Makefile* based stand-alone binary!
- CINT knows types and functions in the file
  - e.g. call

```
root [1] mymacro(43)
```

Controlling ROOT

- Useful CINT commands from the ROOT prompt:
  - quit ROOT

    ```
    root [1] .q
    ```

  - to get the list of available commands

    ```
    root [1] .?
    ```

  - to access the shell of the OS (e.g UNIX or MS/DOS)

    ```
    root [1] .!   <OS_command>
    ```
    e.g.: .! pwd

  - to execute a macro (add a + at the end for compiling with ACLIC)

    ```
    root [1] .x <file_name>
    ```
    e.g.: .x mymacro.C
    .x mymacro.C+

  - to load a macro

    ```
    root [1] .L <file_name>
    ```
    e.g.: .L mymacro.C
    .L mymacro.C+

Plotting a function

- Start using one of basics ROOT classes - the function class TF1:

```
root [0] TF1 * f1 = new TF1("f1","sin(x)/x",0,10);
```

pointer   object name   function type (formula)   function range

- Draw the function:

```
root [1] f1->Draw();
```

The Function will be drawn in a ROOT Canvas
    The Canvas has a GUI Menu
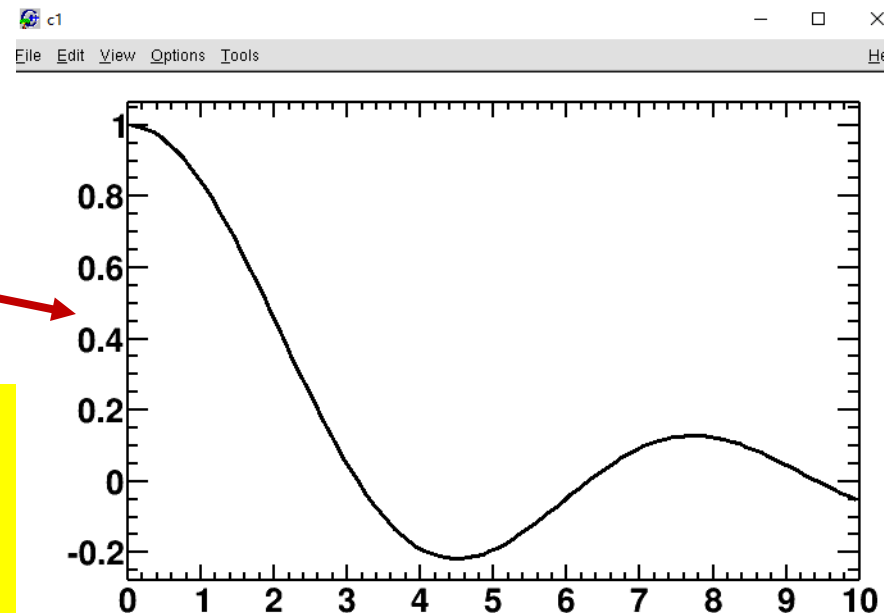
Function with Parameters

- Use the ROOT formula syntax to create a function with parameters, e.g. 2 parameters called [0] and [1]:

```
root [2] TF1 * f2 = new TF1("f2","[0]*sin([1]*x)/x",0,10);
```

- Need to set the parameter values (by defaults the parameters have zero initial values):

```
root [3] f2->SetParameter(0,1);
root [4] f2->SetParameter(1,1);
root [5] f2->Draw();
```

- Can also use the GUI (function editor) to change the function parameters.

Fast Generate Your Macro Code

Save the plot to c1.C file:

$$\#\mathrm{frac}\{\sin(x)\}\ \{x\}$$


```
{
//=========Macro generated from canvas: c1/c1
//========= (Fri Mar 21 08:10:57 2008) by ROOT version5.18/00
TCanvas *c1 = new TCanvas("c1", "c1",423,36,699,534);
c1->Range(-1.119108,-0.4477822,11.24013,1.22563);
c1->SetBorderSize(2);
c1 >SetFrameFillColor(0);
c1->TF1 *func1 = new TF1("func1","sin(x)/x",0.1,10);
func1->SetFillColor(19);
func1->SetFillStyle(0);
func1 >SetLineWidth(3);
func1->func1->Draw("");
TPaveText *pt = new TPaveText(0.01,0.9401777,0.1371429,0.995,"blNDC");
pt->SetName("title");
pt->SetBorderSize(2);
pt->SetFillColor(19);
TText *text = pt->AddText("sin(x)/x");
pt->Draw();
tex = new
TLatex(0.261097,0.768812,"#frac{sin(x)}{x}");
tex->SetLineWidth(2);
tex->Draw();
c1->Modified();
c1->cd();
c1->SetSelected(c1);
c1->ToggleToolBar();
}
```
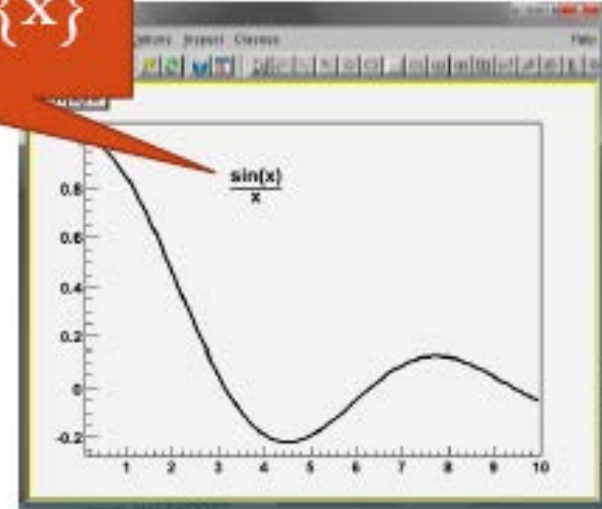
Add these three lines to put a math formula on the plot and re-run the macro c1.C

Macros Applications and Libraries

- write a C++ code file, where ROOT class can be directly used

- execute the C++ code file without compilation (take it as Macros) -- convenient!!!

```
> root myMacro.C
root[] .x myMacro.C
> root -b -q 'myMacro.C(3)' > myMacro.log
> root -b -q 'myMacro.C("text")' > myMacro.log
> root -b -q "myMacro.C(\"text\")" > myMacro.log
```

- Use ACLiC to build a shared library

- Based on ROOT libraries to produce your own libraries or executables

Mathematical Functions in ROOT

- **TMath**: a namespace providing the following functionality:
  - Numerical constants.
  - Trigonometric and elementary mathematical functions.
  - Functions to work with arrays and collections (e.g sort, min max of arrays,...)
  - Statistic Functions (e.g. Gauss)
  - Special Mathematical Functions (e.g. Bessel functions)
  - For more details, see the reference documentation of TMath.

```
TMath::Gaus( x, mean, sigma);
```

- Functions provided in **ROOT::Math** namespace
  - special functions (many implemented using the Gnu Scientific Library)
  - statistical functions
  - For more details, see the reference documentation of ROOT::Math functions

```
ROOT::Math::cyl_bessel_i(nu,x);
```

Plotting Measurements

- The Graph class (**TGraph**):
  - for plotting and analyzing 2-dimensional data (X,Y),
  - contains a set of N distinct points (X_i, Y_i) $i = 1,..N$.
  - can be constructed from a set of x,y arrays:

```
root [1] double x[] = { 1,2,3,4,5};
root [2] double y[] = { 0.5,2.,3.,3.2,4.7};
root [3] TGraph * g = new TGraph(5,x,y);
```

  - or directly from a text file containing rows of (X,Y) data

```
root [1] TGraph * g = new TGraph("XYData.txt");
```
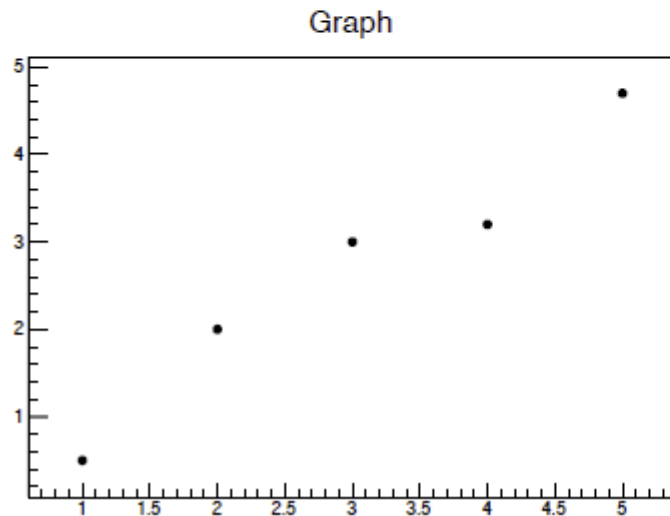
Displaying a Graph

- To display the graph:

```
root [4] g->Draw("AP");
```

- – option "A" means displaying the axis,
- – option "P" means displaying the points.
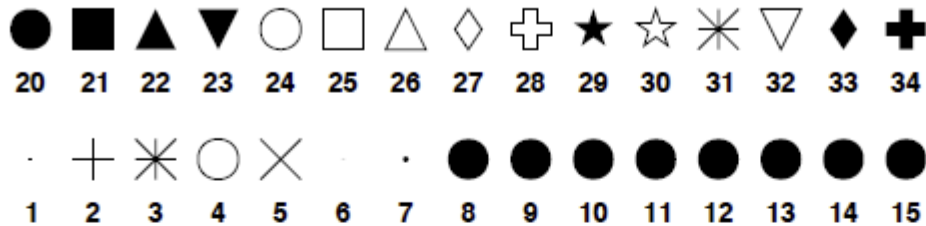

Graph

To change the point markers do:

```
root [4] g->SetMarkerStyle(20);
```

Markers and Colors

- Available markers in ROOT

- Available Colors
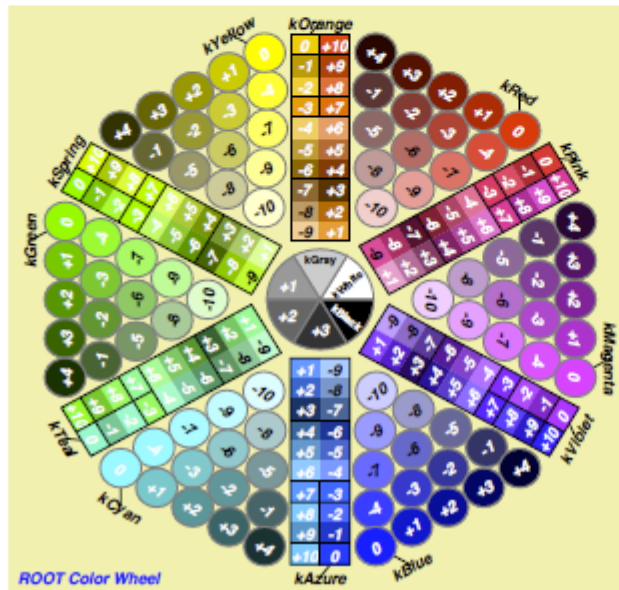
you can access them from the Canvas View Menu

Types of Graphs

- ROOT provides various types of Graphs:
  - **TGraph** : (x,y) data points.
  - **TGraphErrors**:
    - (x,y) data points with error bars (σ_x, σ_y).
  - **TGraphAsymmErrors**:
    - (x,y) data points with asymmetric error bars [(σ^-_x, σ^+_x),(σ^-_y, σ^+_y)].

```
{
    TCanvas *c4 = new TCanvas("c4","c4",200,10,600,400);
    double x[] = {0, 1, 2, 3, 4};
    double y[] = {0, 2, 4, 1, 3};
    double ex[] = {0.1, 0.2, 0.3, 0.4, 0.5};
    double ey[] = {1, 0.5, 1, 0.5, 1};
    TGraphErrors* ge = new TGraphErrors(5, x, y, ex, ey);
    ge->Draw("ap");
    return c4;
}
```
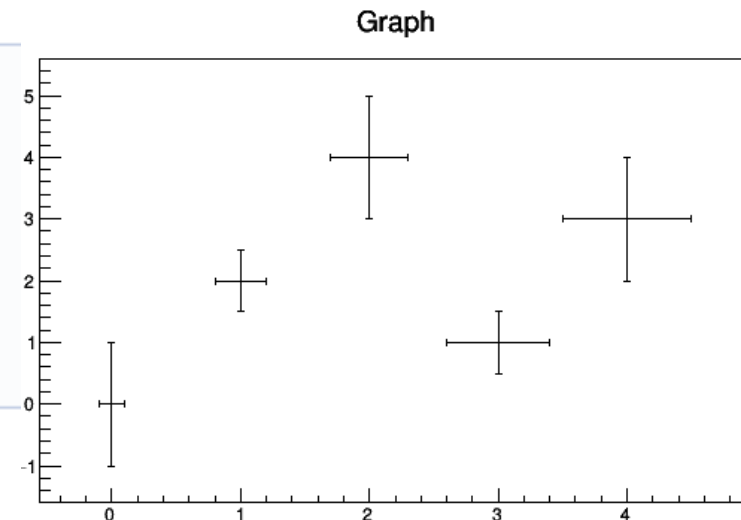

Graph

Graphs Drawing Options

- The drawing of Graphs is done via the `TGraphPainter`
  - see https://root.cern.ch/doc/master/classTGraphPainter.html
  - the documentation lists all drawing options for the different types of graphs available in ROOT

Graphs can be drawn with the following options:

Option	Description
"A"	Axis are drawn around the graph
"L"	A simple polyline is drawn
"F"	A fill area is drawn ('CF' draw a smoothed fill area)
"C"	A smooth Curve is drawn
"*"	A Star is plotted at each point
"P"	The current marker is plotted at each point
"B"	A Bar chart is drawn
"1"	When a graph is drawn as a bar chart, this option makes the bars start from the bottom of the pad. By default they start at 0.
"X+"	The X-axis is drawn on the top side of the plot.
"Y+"	The Y-axis is drawn on the right side of the plot.

Histograms

- ## What is a histogram?

  - *from Wikipedia:*

  - a **histogram** is a graphical representation showing a visual impression of the distribution of data. It is an estimate of the probability distribution of a continuous variable and was first introduced by Karl Pearson.[1]

  - A histogram consists of tabular frequencies, shown as adjacent rectangles, erected over discrete intervals (bins), with an area equal to the frequency of the observations in the interval.

  - The height of a rectangle is also equal to the frequency density of the interval, i.e., the frequency divided by the width of the interval. The total area of the histogram is equal to the number of data entries.

Example histogram

h1	
Entries	1000
Mean	0.01102
RMS	1.007
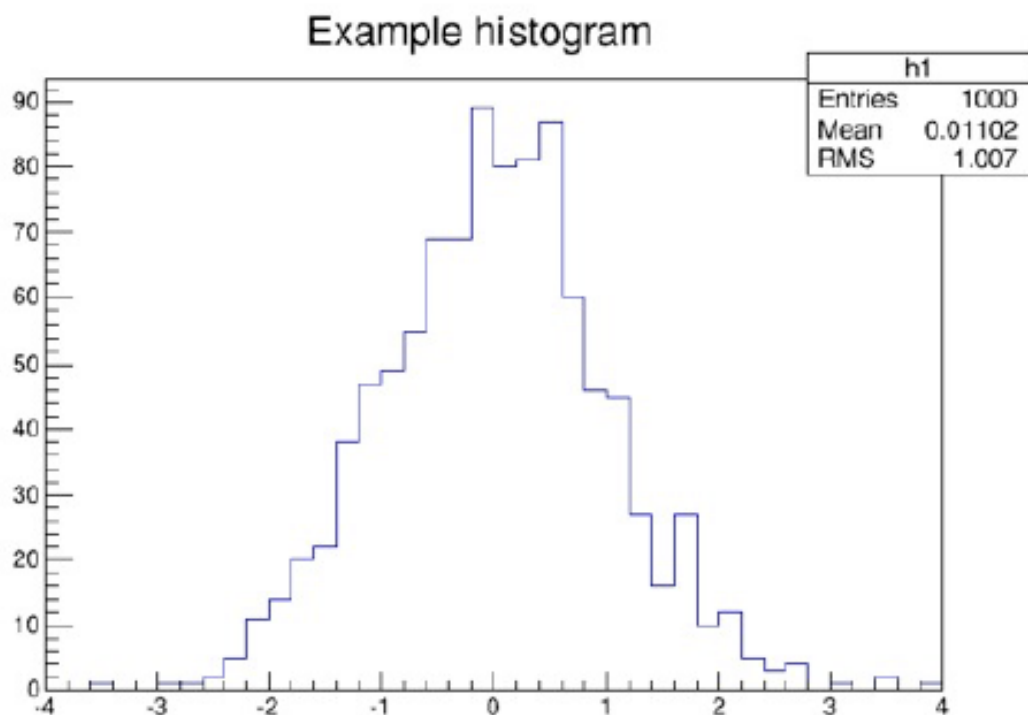
Histograms in ROOT

- Used to display and estimate the distribution of a variable (e.g. observed energy spectrum)
  - visualize number of events in a certain range, called *bin*
  - *bins* typically have equal widths, but not always
    - ROOT supports histograms with equal and variable bins
- Histograms can be used for further analysis
  - e.g to understand the underlying parent distribution
- ROOT provides various types of histograms depending on:
  - contained data type (double, float, integer, char)
  - choice of uniform or variable bins
  - dimension (1,2 or 3)

How to use ROOT Histograms

- Example of creating a one-dim. histogram:

```
TH1D * h1 = new TH1D("h1","Example histogram",40,-4.,4.);
```

histogram type   name   title   axis settings
TH1D: one-dimension   number   min,max
using double types   of bins   values

- Filling histogram:

```
h1->Fill(x);
```

Fill the histogram with one observation "

```
for (int i = 0; i<1000; ++i) {
    double x = gRandom->Gaus(0,1);
    h1->Fill(x);
}
```

Fill the histogram with 1000 gaussian distributed random numbers

Displaying ROOT histograms

- Drawing histograms in a ROOT canvas:

```
h1->Draw();
```

Example histogram

histogram name

	h1
Entries	1000
Mean	0.01102
RMS	1.007

The histogram statistics:
- Total number of entries
- Sample Mean
- Sample Standard Deviation (RMS)

number of observed events/bin

variable x
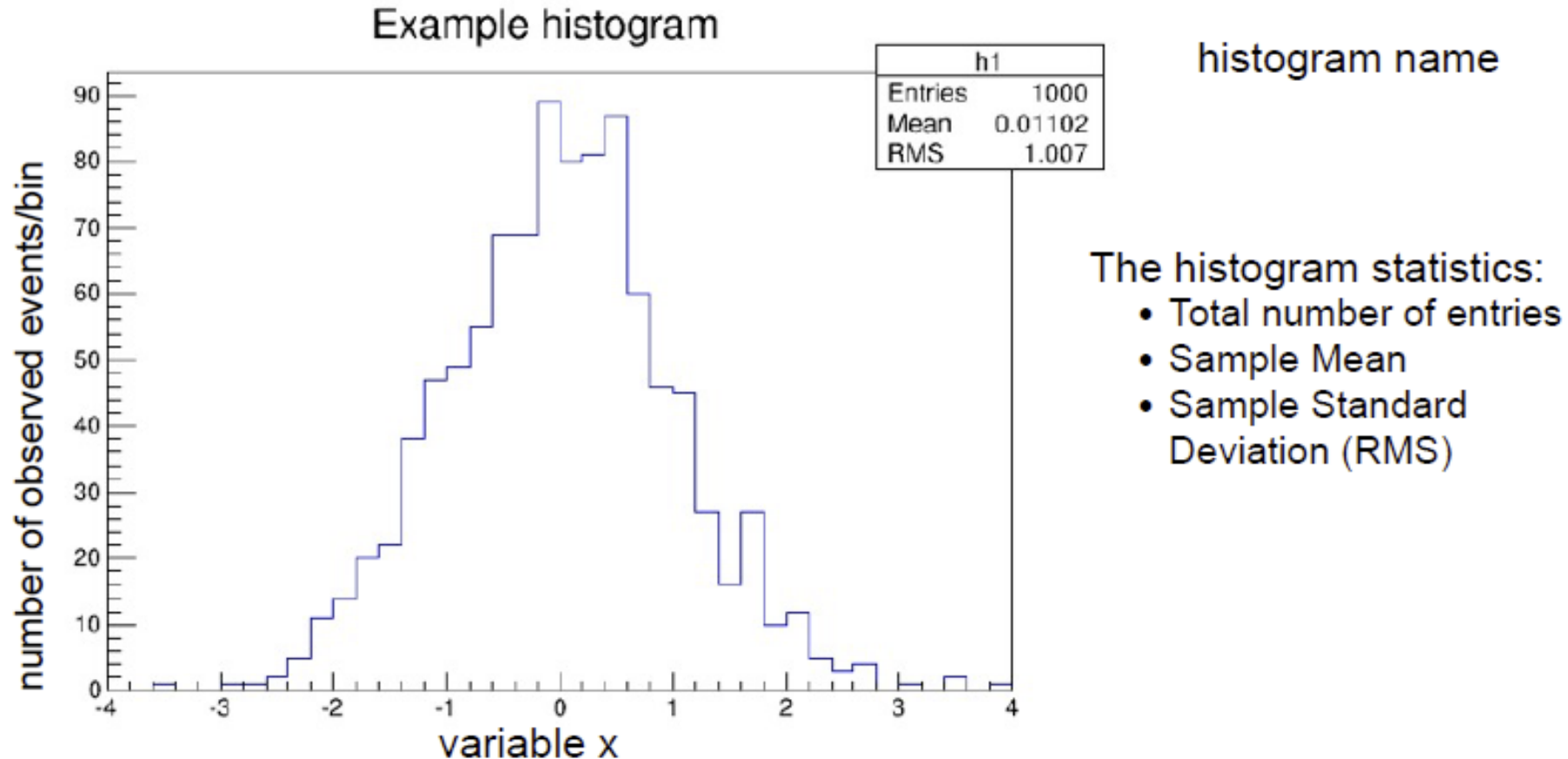
Histogram Statistics

- To extract statistics information from an histogram:

```
root [] h1->GetEntries()
(const Double_t)1.000000000000000000e+03
root [] h1->Integral()
(const Double_t)1.000000000000000000e+03
root [] h1->GetMean()
(const Double_t)1.10172792035927100e-02
root [] h1->GetMeanError()
(const Double_t)3.18311744869313878e-02
root [] h1->GetRMS()
(const Double_t)1.00659011976944801e+00
root [] h1->GetRMSError()
(const Double_t)2.25080393328414077e-02
root [] h1->GetSkewness()
(const Double_t)1.17820738464490191e-01
root [] h1->GetKurtosis()
(const Double_t)2.58961968358840000e-01
```

Histogram Drawing Options

- Various drawing options are available:
 - draw error bars on every bin

    ```
    h1->Draw("E");
    ```

  - "SAME"

    ```
    h1->Draw("SAME");
    ```

    - draw the histogram on the canvas without replacing what is already there

    - use to plot one histogram on top of another

- The default drawing option is "HIST" for histograms without errors (unweighted histograms) and "E" for weighted histograms

- For displaying the histogram in log scale in one axis, e.g. the y axis:

    ```
    gPad->SetLogy();
    ```


Example histogram
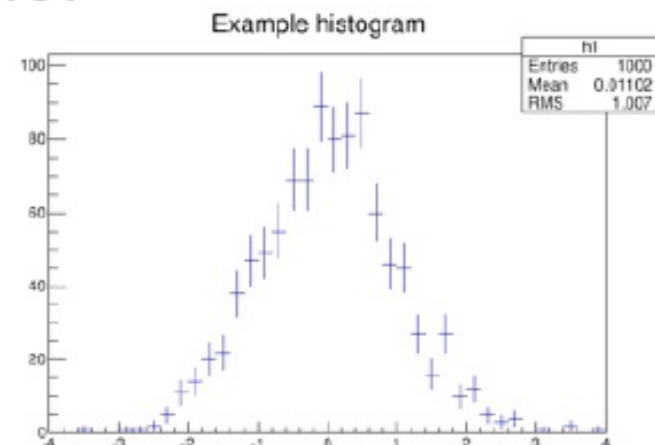
The error bars options

Option	Description
"E"	Default. Shows only the error bars, not a marker.
"E1"	Small lines are drawn at the end of the error bars.
"E2"	Error rectangles are drawn.
"E3"	A filled area is drawn through the end points of the vertical error bars.
"E4"	A smoothed filled area is drawn through the end points of the vertical error bars.
"E0"	Draw also bins with null contents.

```
{
    TCanvas *c1 = new TCanvas("c1","c1",600,400);
    TH1F *he = new TH1F("he","Distribution drawn with error bars (option E1)  ",100,-3,3);
    Int_t i;
    for (i=0;i<10000;i++) he->Fill(gRandom->Gaus(0,1));
    gStyle->SetEndErrorSize(3);
    gStyle->SetErrorX(1.);
    he->SetMarkerStyle(20);
    he->Draw("E1");
    return c1;
}
```


Distribution drawn with error bars (option E1)

he	
Entries	10000
Mean	0.005243
Std Dev	0.9812
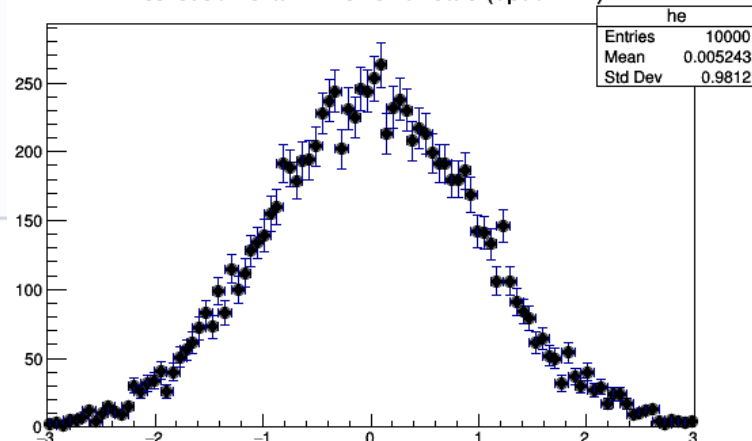
Histogram Drawing Options

- Histogram drawing is handled internally by the **THistPainter** class.
- The documentation for all the drawing options can be found in the class reference page

https://root.cern.ch/doc/master/classTHistPainter.html

THistPainter Class Reference

Histogram Library » Histograms and graphs painting classes.

The histogram painter class.

Implements all histograms' drawing's options.

- Introduction
- Histograms' plotting options
  - Options supported for 1D and 2D histograms
  - Options supported for 1D histograms
  - Options supported for 2D histograms
  - Options supported for 3D histograms

Global Pointers

- `gSystem` : Interface to the operating system.

- `gStyle` : Interface to the current graphics style.

- `gPad` : Interface to the current graphics Pad.

- `gROOT` : Entry point to the ROOT system.

- `gRandom`: Interface to the current random number generator.

Compiling C++ code using ROOT

- Command "`root-config`" tells you necessary compiler flags:

```
root-config --incdir
/Users/moneta/root/5.34.04/include

root-config --libs
-L/Users/moneta/root/5.34.04/lib -lCore -lCint -lRIO -lNet -lHist
-lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -
lPhysics -lMathCore -lThread -lpthread -Wl,-rpath,/Users/moneta/
root/5.34.04/lib -lm -ldl
```

- To compile a file `example.cxx` that uses root, use:

```
g++ -c -I `root-config --incdir`example.cxx
```

- To compile and link a file `example.cxx` that uses root, use:

```
g++ -I `root-config --incdir` -o example
example.cxx `root-config --libs`
```

非常有用

The inverted quotes tell the shell to run a command and paste the output into the corresponding place.

Summary

- How to work from the ROOT prompt
  - how to run interpreted C/C++ code
  - what is a ROOT macro
  - how to run compiled macro with ACLic
- Started looking at some basics ROOT objects
  - functions
  - graphs for plotting measurements
  - histograms
- How to plot these objects