

# 第三章：实验统计分析工具ROOT

授课人：董燎原

中国科学院高能物理研究所

# Introduction

## Start Using ROOT

# Outline

- Starting to work from the ROOT prompt
  - ROOT as a calculator
- Creating and plotting functions
- Plotting data measurements
- Introduction to histograms
- Plotting one-dimensional histograms

# ROOT Download & Installation

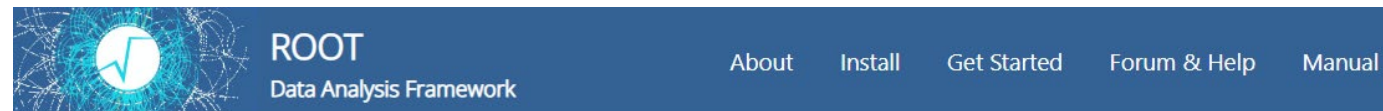
- Download from ROOT Web site <http://root.cern.ch>
- Binaries for Linux, MacOS and Windows
- Source files can be built with:  
    `./configure`  
    `make`  
    `make install (as root)`
- see the instructions on the Web site for building from sources

源程序编译安装:

<https://root.cern.ch/building-root>

源程序下载:

[root\\_v6.30.04.source.tar.gz](#) 176M

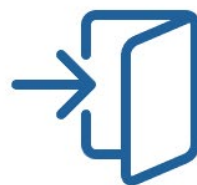


## ROOT: analyzing petabytes of data, scientifically.

An open-source data analysis framework used by high energy physics and others.

[Learn more](#)

[Install v6.30.04](#)



Start

Version 6 [🔗](#)

Release 6.30.04 - 31 Jan 2024



Reference

Version 5

Release 5.34/38 - 12 Mar 2018



Forum

# Starting Up ROOT

- Set environment variables:

```
export ROOTSYS=/usr/local/root (一般默认安装在此目录)  
export PATH=$PATH:$ROOTSYS/bin  
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib
```

or

```
Source /usr/local/root/bin/thisroot.sh
```

- Put the above commands in the login file (e.g., *.bash\_profile*)
- ROOT is prompt-based Launch ROOT:

```
$ root
```

登录ROOT时不需要显示logo:  
alias rt='root -l'

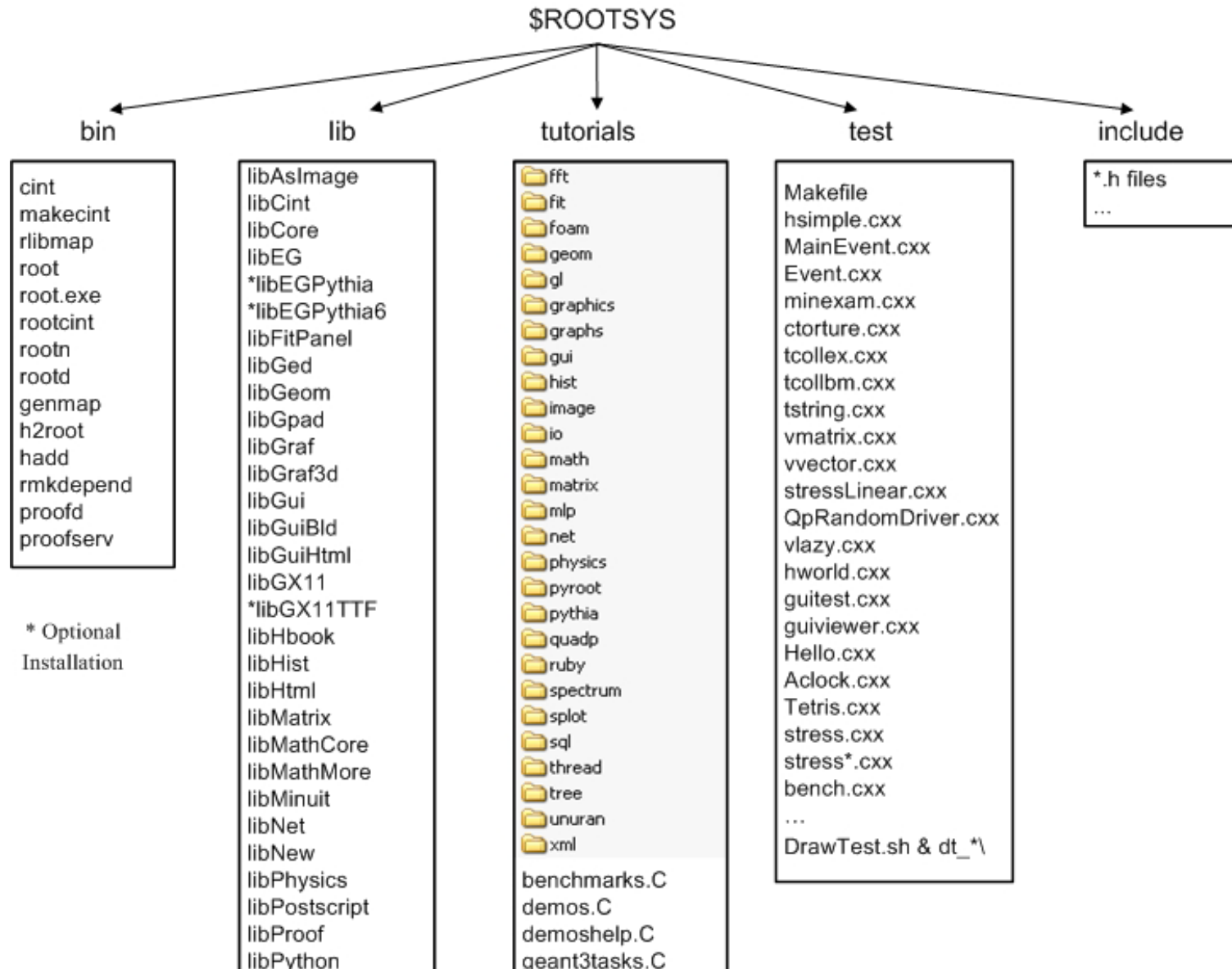
The ROOT prompt will appear:

```
root [0] _
```

- It “speaks” C++:

```
[dongly@lxslc712 ~]$ root  
-----  
| Welcome to ROOT 6.24/08                                     https://root.cern |  
| (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers |  
| Built for linuxx8664gcc on Sep 29 2022, 13:04:57             |  
| From tags/v6-24-08@v6-24-08                                 |  
| With c++ (GCC) 4.8.5 20150623 (Red Hat 4.8.5-44)            |  
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.'q' |  
|-----  
  
root [0] █
```

# The Framework Organization



# \$ROOTSYS/bin

<code>root</code>	shows the ROOT splash screen and calls <code>root.exe</code>
<code>root.exe</code>	the executable that <code>root</code> calls, if you use a debugger such as <code>gdb</code> , you will need to run <code>root.exe</code> directly
<code>rootcint</code>	is the utility ROOT uses to create a class dictionary for CINT
<code>rmkdepend</code>	a modified version of <code>makedepend</code> that is used by the ROOT build system
<code>root-config</code>	a script returning the needed compile flags and libraries for projects that compile and link with ROOT

others: `proofd`, `proofserv`, `rootd`...

# ROOT tutorials and test

- \$ROOTSYS/tutorials: contains **many example scripts**.

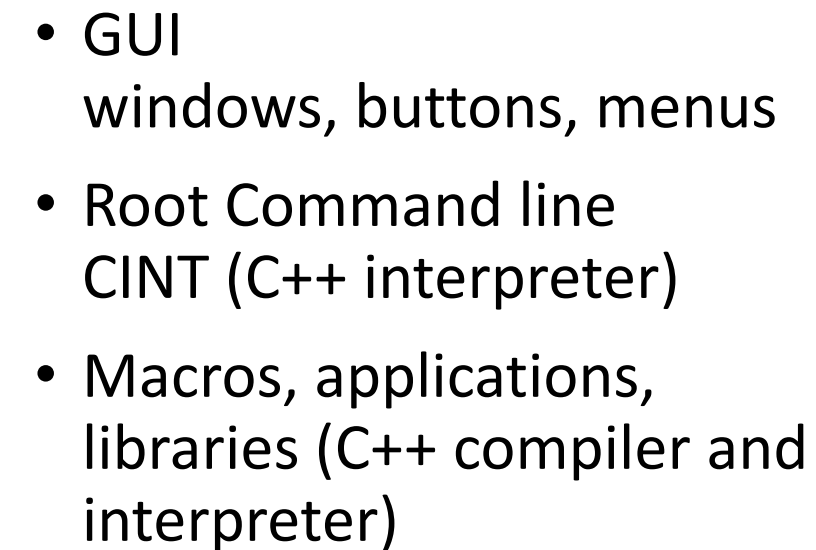
They assume some basic knowledge of C++ and ROOT

*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!*



[illegible]

# Environment Setup

- The behavior of a ROOT session can be configured with the options in the *rootrc* file. At start-up, ROOT looks for a .rootrc file in the following order:

<code>./rootrc</code>	<code>//local directory</code>
<code>\$HOME/.rootrc</code>	<code>//user directory</code>
<code>\$ROOTSYS/etc/system.rootrc</code>	<code>//global ROOT directory</code>

- to see current settings, you can do:

```
root[] gEnv->Print()
```

- Default Logon and Logoff Scripts:

in local directory:

rootalias.C (only loaded when root starts up)

rootlogon.C (executed when root starts up)

rootlogoff.C (executed when root ends)

in \$HOME directory: .rootlogon.C

examples: check the relevant files

`$ROOTSYS/tutorials`

一个简单的rootlogoff.C:

```
I RootLogoff.C
{
std::cout<<"Have a Nice day, 886 !"<<std::endl<<std::endl;
}
```

每次退出root都说: Have a Nice day !

# Example: rootlogon.C

```
{
//my own style
gStyle->SetOptTitle(1);
gStyle->SetOptStat(1111111);
gStyle->SetOptFit(1111);
gStyle->SetHistLineWidth(1);

gSystem->Load("libRooFit");
gSystem->Load("libPhysics");

// BES style changed from BABAR style
TStyle *besStyle= new TStyle("BES","BABAR approved plots style");

// use plain black on white colors
besStyle->SetCanvasBorderMode(0);
besStyle->SetCanvasColor(0);
besStyle->SetFrameBorderMode(0);
besStyle->SetFrameBorderSize(3);
besStyle->SetFrameLineStyle(1);
besStyle->SetFrameLineWidth(2);
besStyle->SetFrameLineColor(0);
besStyle->SetPadBorderMode(0);
besStyle->SetPadColor(0);

besStyle->SetStatColor(0);
//besStyle->SetFillColor(0);

// set the paper & margin sizes
besStyle->SetPaperSize(20,26);
besStyle->SetPadTopMargin(0.05);
besStyle->SetPadRightMargin(0.05);
besStyle->SetPadBottomMargin(0.16);
besStyle->SetPadLeftMargin(0.13);

// use large Times-Roman fonts
//besStyle->SetTextFont(132);
besStyle->SetTextFont(62);
besStyle->SetTextSize(0.08);
```

```
besStyle->SetLabelFont(62,"x");
besStyle->SetLabelFont(62,"y");
besStyle->SetLabelFont(62,"z");
besStyle->SetLabelSize(0.06,"x");
besStyle->SetLabelSize(0.06,"y");
besStyle->SetLabelSize(0.06,"z");
besStyle->SetTitleSize(0.06,"x");
besStyle->SetTitleSize(0.06,"y");
besStyle->SetTitleSize(0.06,"z");

// use bold lines and markers
besStyle->SetMarkerStyle(20);
besStyle->SetMarkerSize(0.5);
besStyle->SetMarkerColor(2);
besStyle->SetLineWidth(2);
//besStyle->SetLineStyleString(2,"[12 12]"); // postscript dashes

// get rid of X error bars and y error bar caps
besStyle->SetErrorX(0.001);

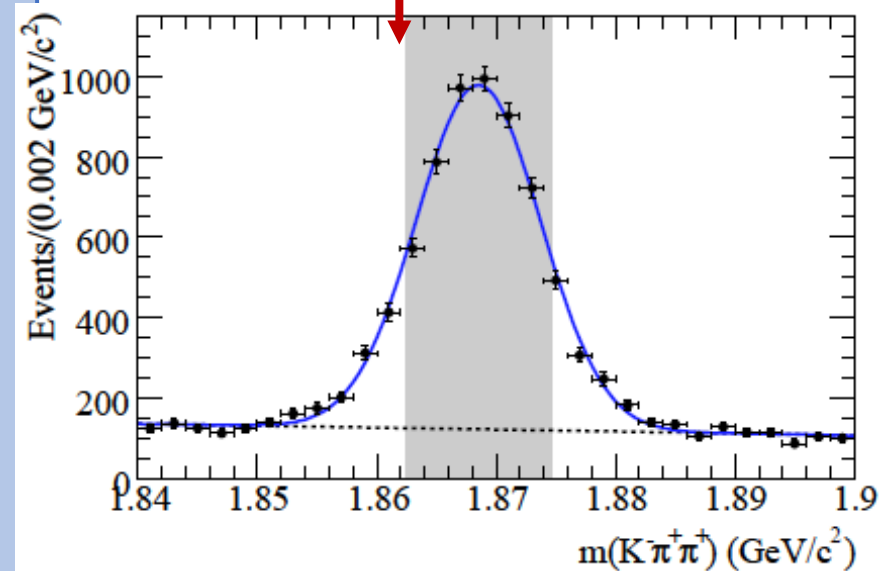
// do not display any of the standard histogram decorations
besStyle->SetOptTitle(0);
besStyle->SetOptStat(0);
besStyle->SetOptFit(0);

// put tick marks on top and RHS of plots
besStyle->SetPadTickX(1);
besStyle->SetPadTickY(1);

besStyle->SetOptStat("nemr");
besStyle->SetOptTitle(0);
besStyle->SetHistLineWidth(2);

printf("Welcome to ROOT!\n");
gROOT->SetStyle("BES");
gStyle->SetLineWidth(2);
}
```

进入root前，自动  
运行rootlogon.C，  
设置做图的默认参  
数，以便画出达到  
发表水准的图。



# 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)4.000000000000000000e+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

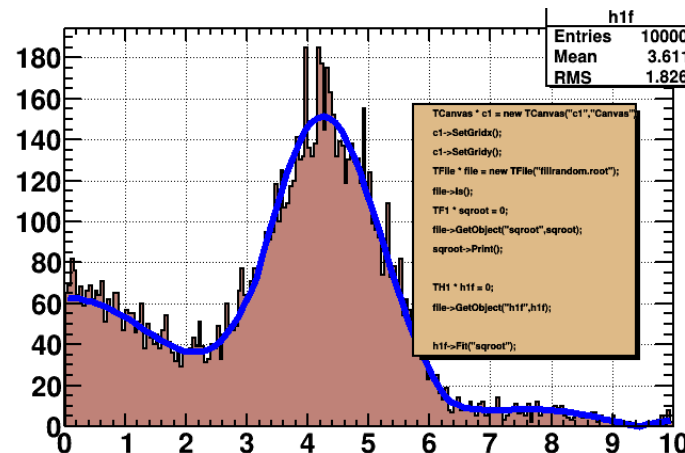
```
Root> .x demos.C
```

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：



```
-----ROOT 5.34/14 benchmarks summary (in ROOTMARKS)-----
For comparison, a Pentium IV 2.4Ghz is benchmarked at 600 ROOTMARKS
hsimple   = 185.73 RealMARKS,   = 358.33 CpuMARKS
hsum      = 332.26 RealMARKS,   = 293.88 CpuMARKS
fillrandom = 772.56 RealMARKS,   = 300.00 CpuMARKS
fit1      = 329.89 RealMARKS,   = 300.00 CpuMARKS
tornado   = 369.50 RealMARKS,   = 900.00 CpuMARKS
na49      = 185829.23 RealMARKS, = inf CpuMARKS
geometry  = 416.66 RealMARKS,   = 3600.00 CpuMARKS
na49view  = 2015.25 RealMARKS,   = 3000.00 CpuMARKS
ntuple1   = 1101.56 RealMARKS,   = 884.21 CpuMARKS

*****
* Your machine is estimated at 678.73 ROOTMARKS *
*****
```

- 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)
```



# Object definition in CINT Script

```
// file name: draw1.C
void draw1(){
    TF1 f("f","sin(x)",0,20);
    f.Draw();
}
```

**local variable/object** is destroyed when the function exits.

**modified ones:**

```
// file name: draw2.C
void draw2(){
    TF1 *f=new TF1("f","sin(x)",0,20);
    f->Draw();
}
```

```
// file name: draw3.C
{
    TF1 f("f","sin(x)",0,20);
    f.Draw();
}
```

*what the difference?*

# ROOT As Pocket Calculator

Calculations:

```
root [0] sqrt(42)
(const double)6.48074069840786038e+00
root [1] double val = 0.17;
root [2] sin(val)
(const double)1.69182349066996029e-01
root [2] TMath::Erf(1.)
(Double_t)8.42700792949714783e-01
```

Uses C++ Interpreter CINT

# ROOT Prompt

- ? Why C++ and not a scripting language?!
- ! You'll write your code in C++, too. Support for python, ruby,... exists.
- ? Why a prompt instead of a GUI?
- ! ROOT is a programming framework, not an office suite. Use GUIs where needed.

# 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)
```

# Compiled versus Interpreted

? Why compile?

! Faster execution, CINT has limitations, validate code.

? Why interpret?

! Faster Edit → Run → Check result → Edit cycles ("rapid prototyping").  
Scripting is sometimes just easier.

? Are Makefiles dead?

! Yes! ACLiC is even platform independent!

# 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

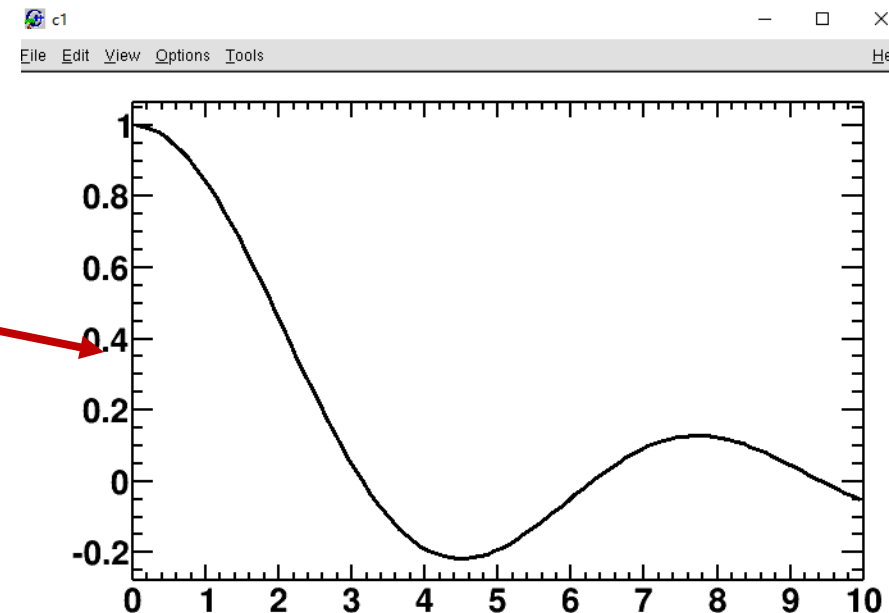
function type  
(formula)

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]:

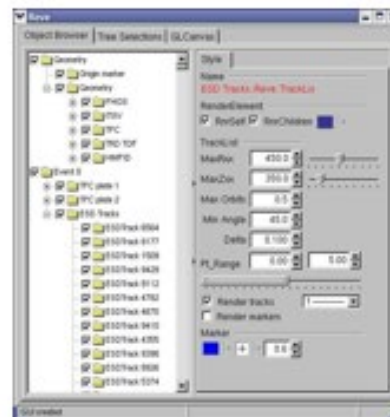
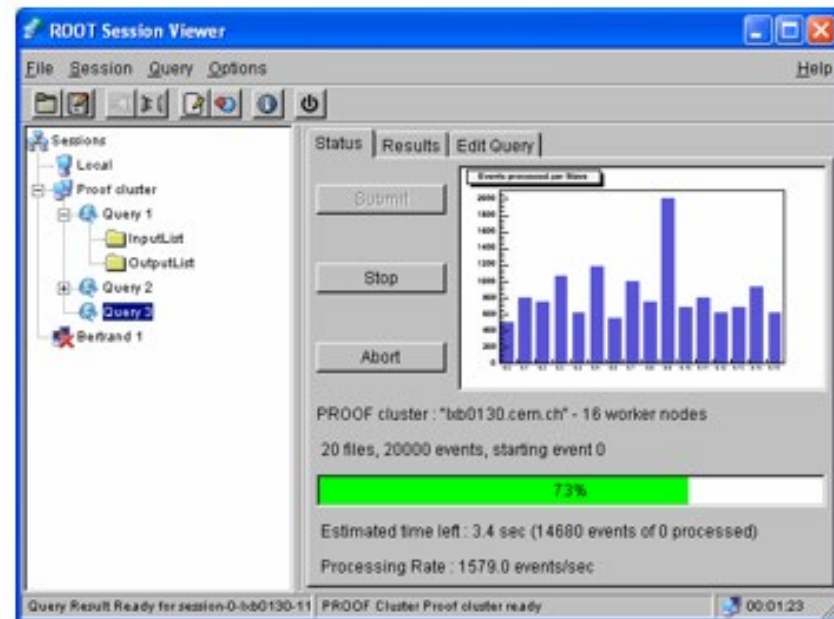
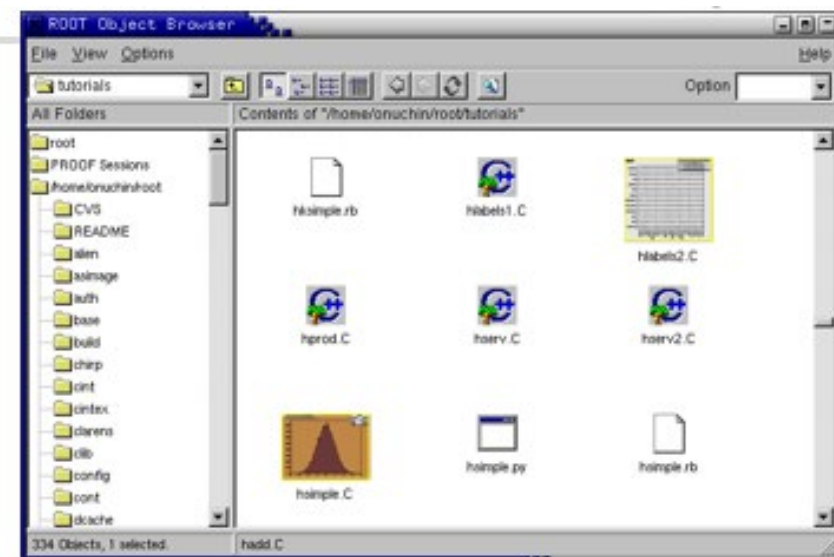
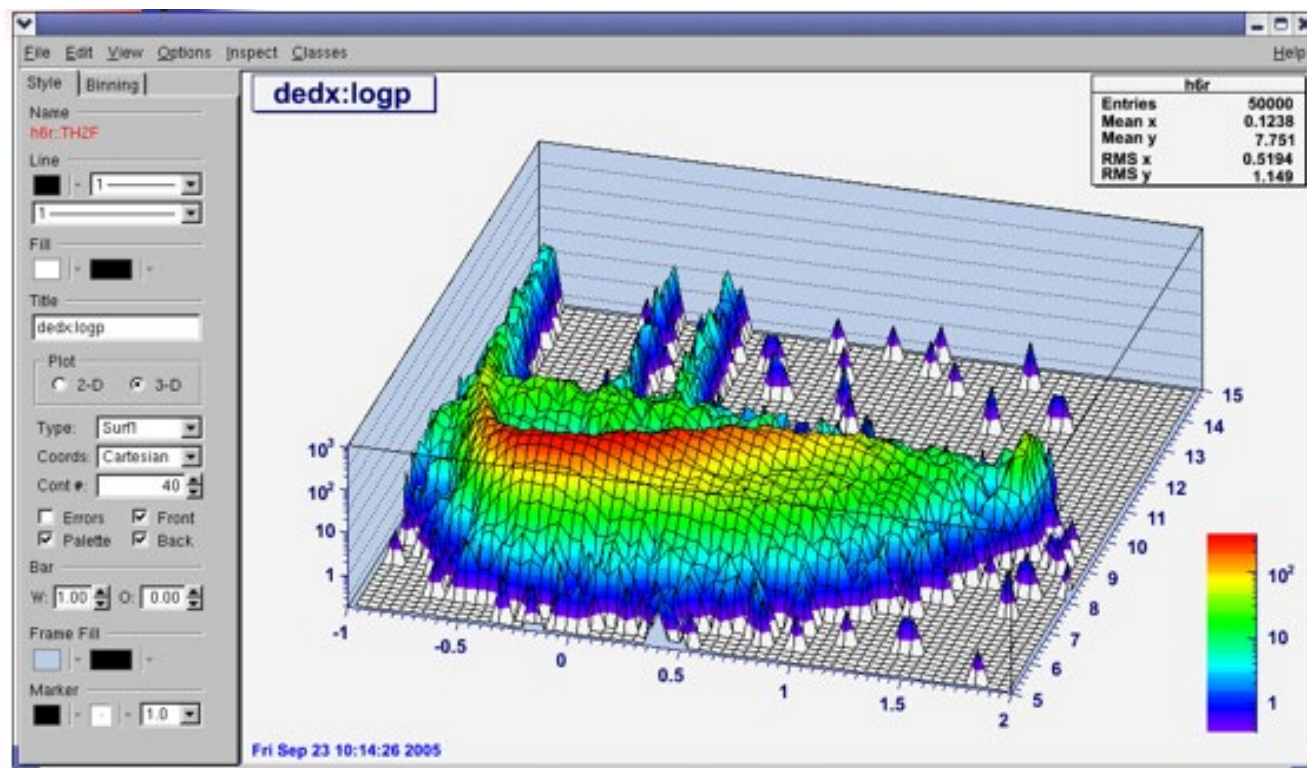
```
root [2] TF1 * f2 = new TF1("f2","[0]*sin([1]*x)/x",0,10);
```

- 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();
```

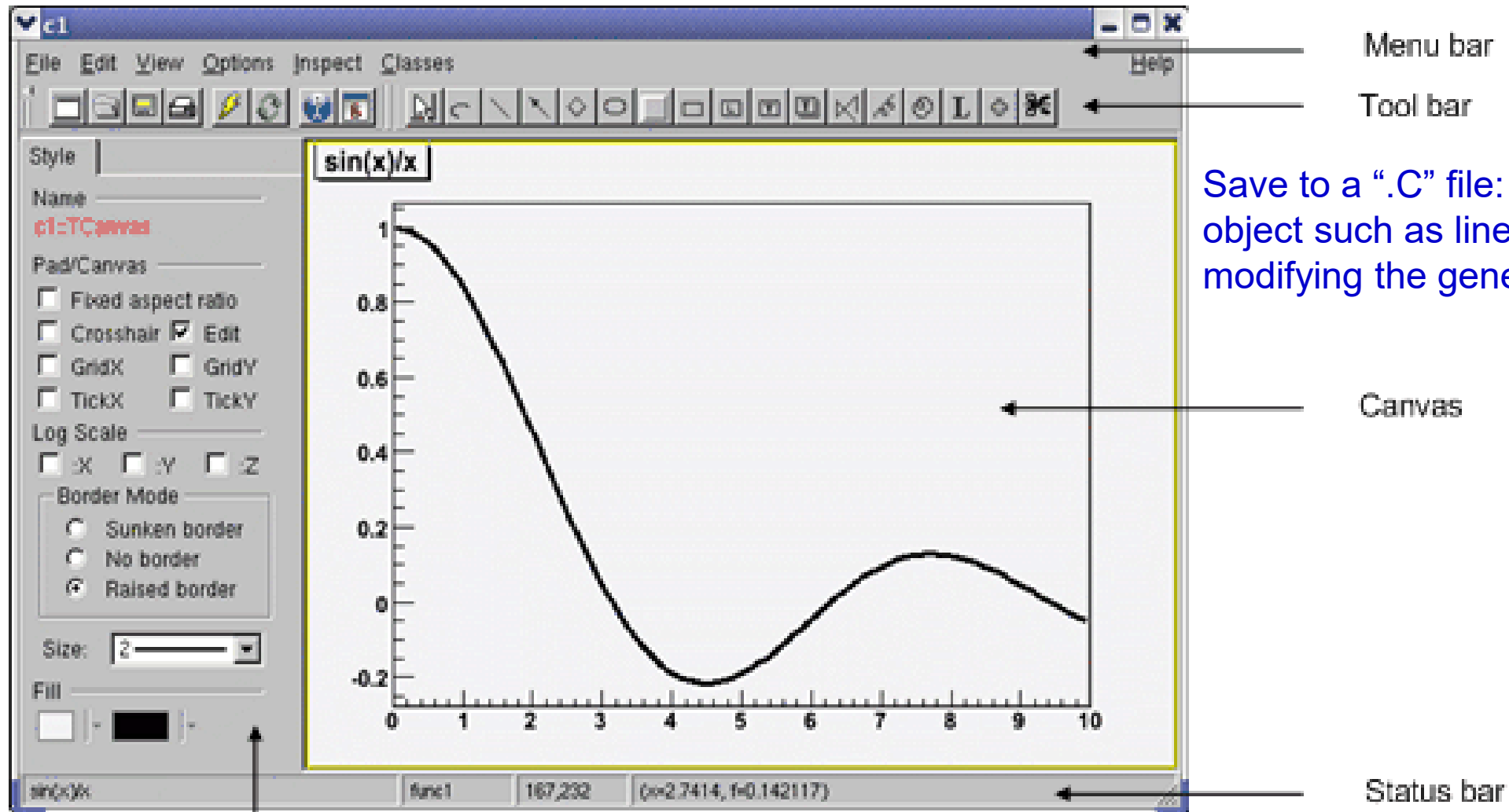
- Can also use the GUI (function editor) to change the function parameters.

# GUI (Graphical User Interface)



# GUI Editor for ROOT Objects

- View the GUI Editor from the Canvas View Menu



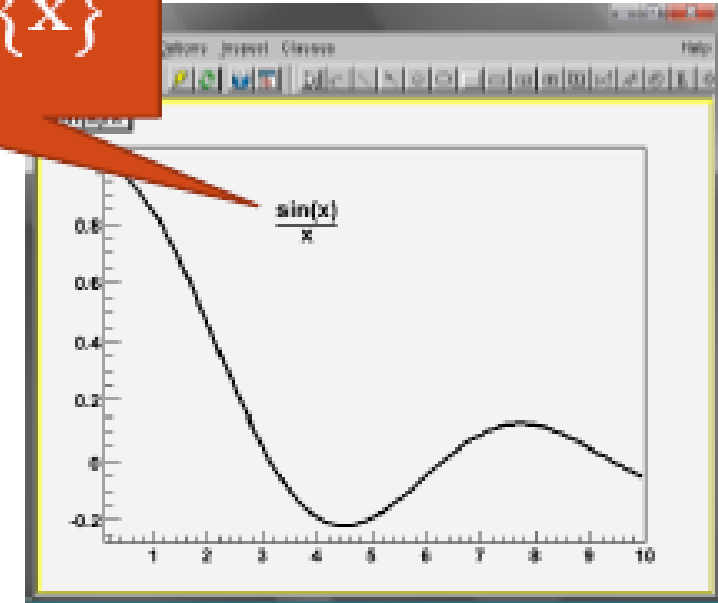
Save to a ".C" file: you can add new object such as line, arrow, on the plot by modifying the generated ".C" file.

# Fast Generate Your Macro Code

Save the plot to c1.C file:

```
{
//=====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,"b1NDC");
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();
}
```

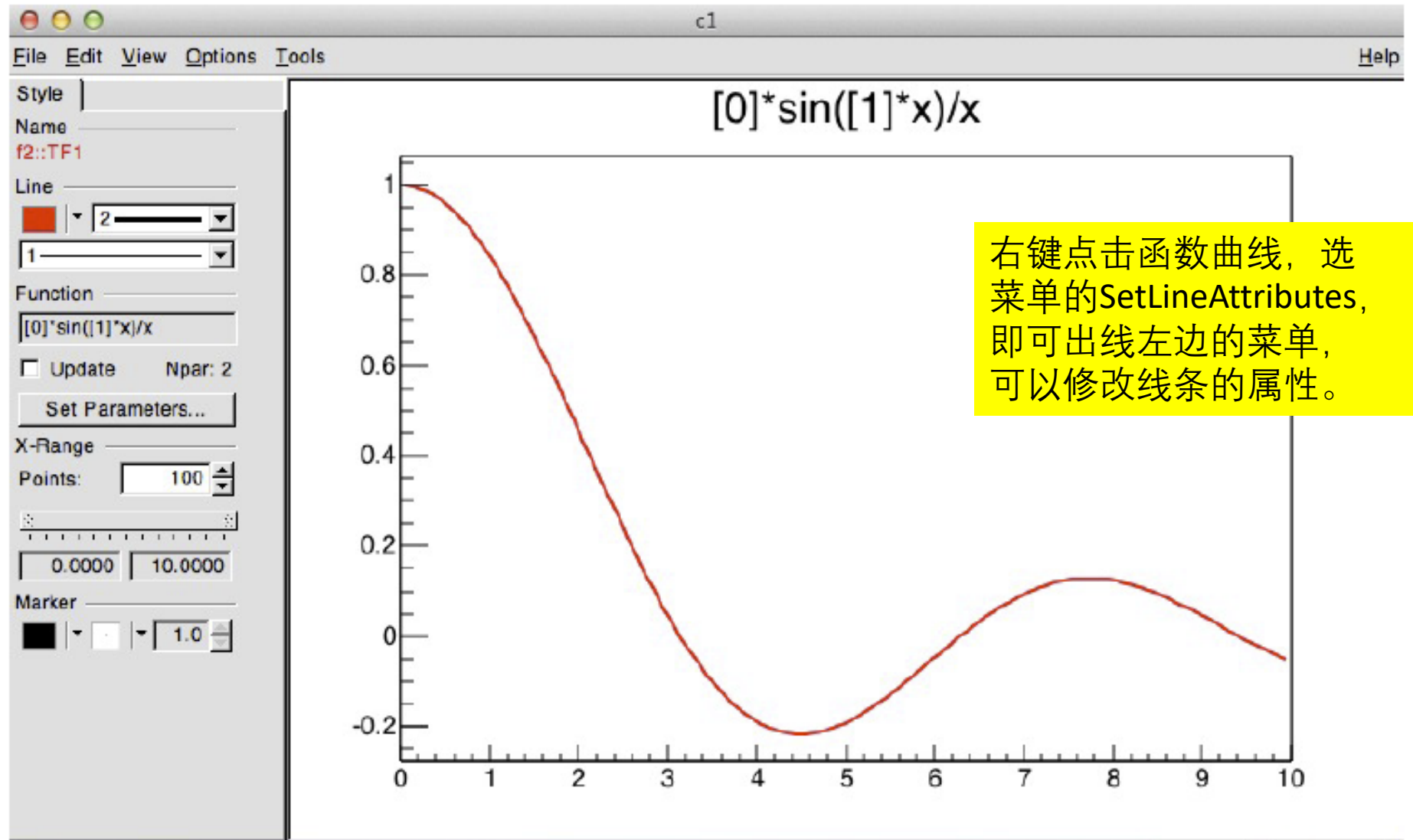
$\#frac{\sin(x)}{x}$



Add these three lines to put a math formula on the plot and re-run the macro c1.C

# Function GUI Editor

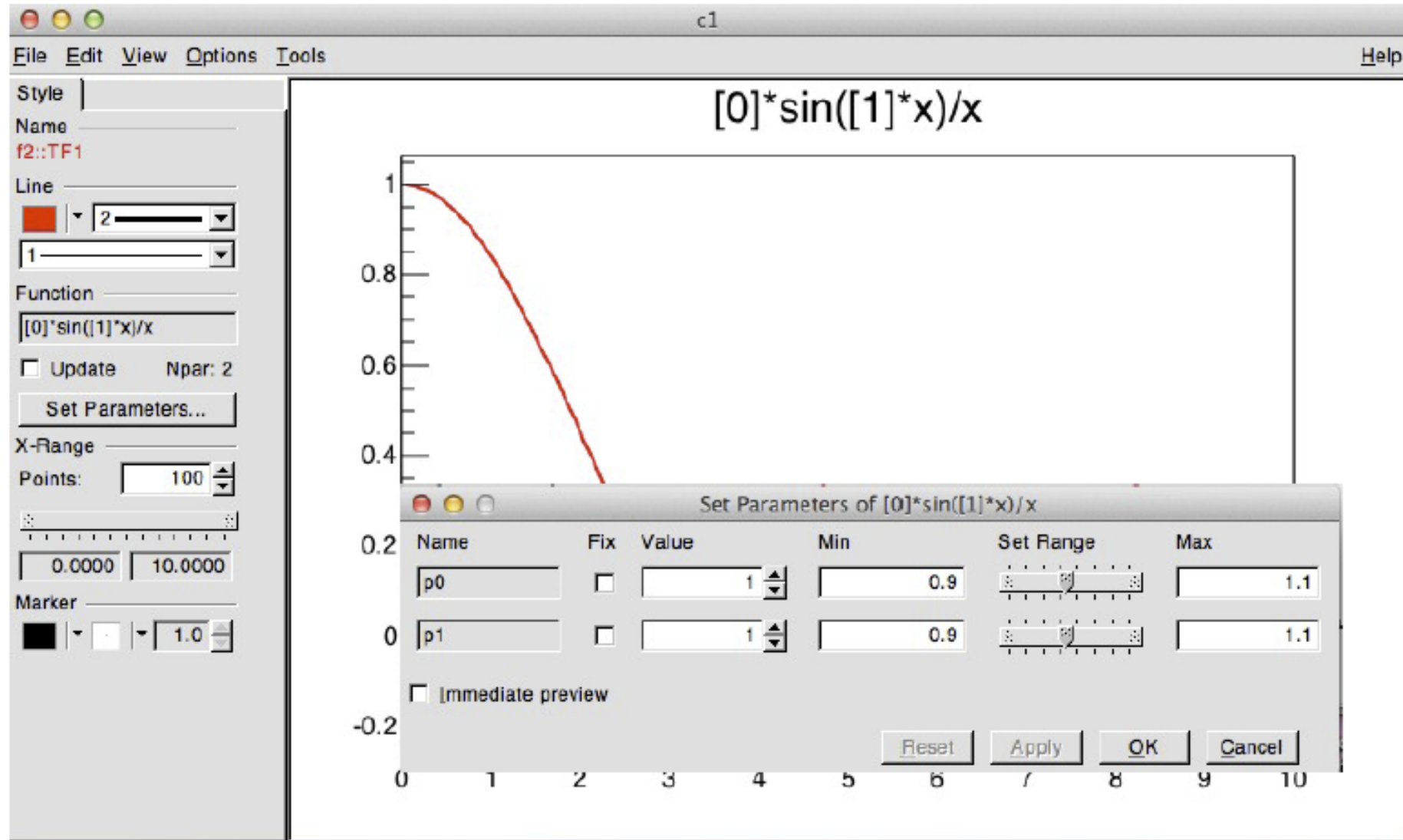
- Click on the function with the mouse





# Function GUI Editor: Set Parameters

- Click on “Set Parameters”



# 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);
```

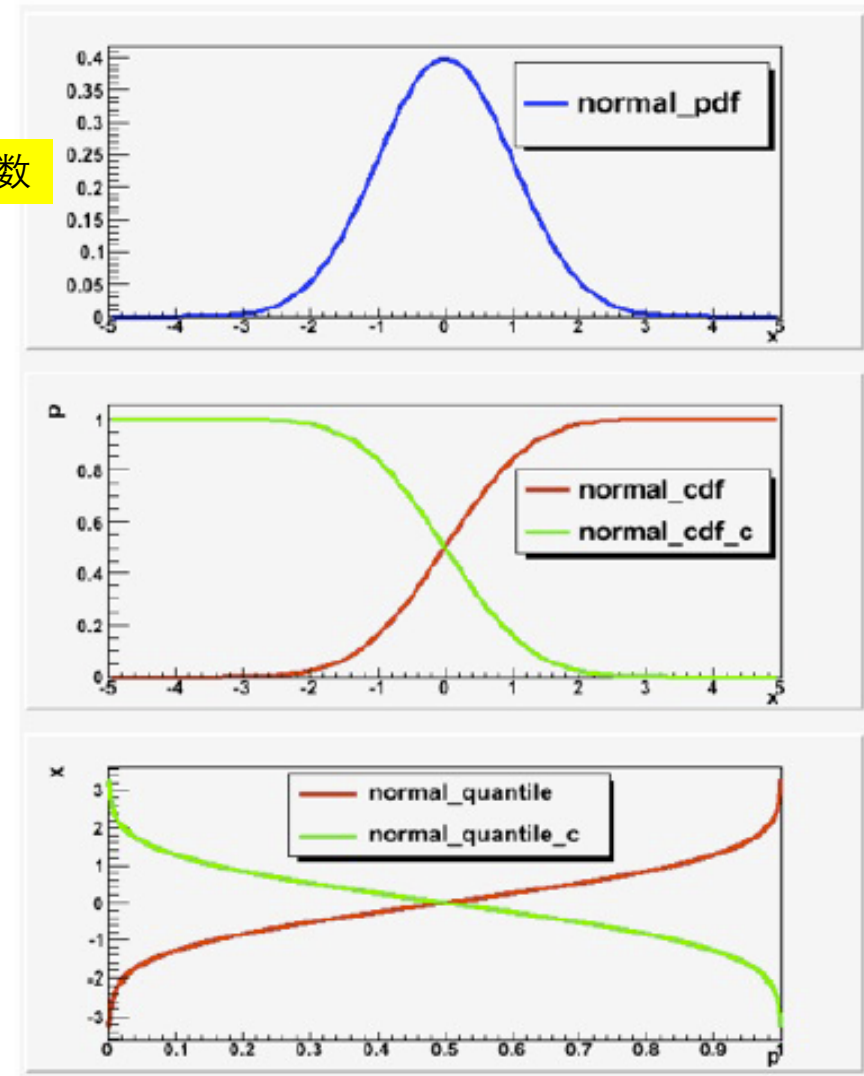


# ROOT::Math Statistical Functions

- Statistical Functions are provided in a coherent naming scheme
  - probability density functions (pdf)
    - i.e. normal distribution:
      - `normal_pdf( x, sigma, mu)`
  - cumulative distributions (cdf)
    - lower tail: `normal_cdf( x, sigma, mu)`
    - upper tail: `normal_cdf_c( x, sigma, mu)`
  - inverse of cumulative distributions (quantiles)
    - inverse of lower cumulative
      - `normal_quantile( z, sigma)`
    - inverse of lower cumulative
      - `normal_quantile_c( z, sigma)`
- All major statistical distributions available
  - *normal, lognormal, Landau, Cauchy,  $\chi^2$ , gamma, beta, F, t, poisson, binomial, etc..*
- Defined as free functions in ROOT::Math namespace

概率分布函数

累积分布函数



# Examples for using Distributions

- p values

- probability after a fit

```
double prop = ROOT::Math::chisquared_cdf_c(val, ndf);
```

- significance : (number of sigma's)

- probability  $\alpha$  to observe  $s$  or larger signal in the case of pure background fluctuation

$$\alpha = \int_{n\sigma}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx = 1 - \frac{1}{2} \operatorname{erf} \left( \frac{n}{\sqrt{2}} \right)$$

```
double sig = ROOT::Math::normal_quantile_c(alpha, 1.);
```

- implementation is careful to avoid numerical error

- uses inverse error function or its complement depending on whether  $\alpha$  is close to 0 or 1.

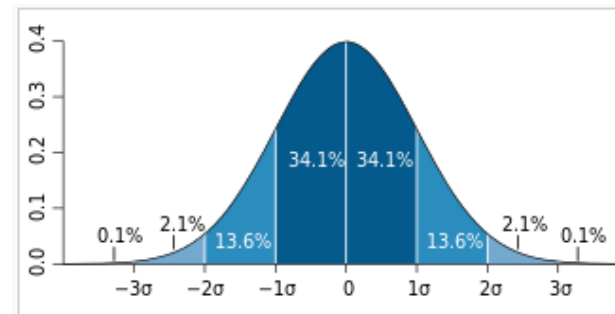
正态变量落在  $\mu \pm \sigma$  的概率:

$$p(\mu - \sigma < X \leq \mu + \sigma) \approx 68.3\%$$

$$p(\mu - 3\sigma < X \leq \mu + 3\sigma) \approx 99.73\%$$

$$p(\mu - 5\sigma < X \leq \mu + 5\sigma) \approx 99.9999\%$$

粒子物理实验  
一般把 $5\sigma$ 当做发现标准



# Plotting Measurements

- The Graph class (**TGraph**):
  - for plotting and analyzing 2-dimensional data (X,Y),
  - contains a set of N distinct points  $(X_i, Y_i)$   $i = 1, \dots, N$ .
  - 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

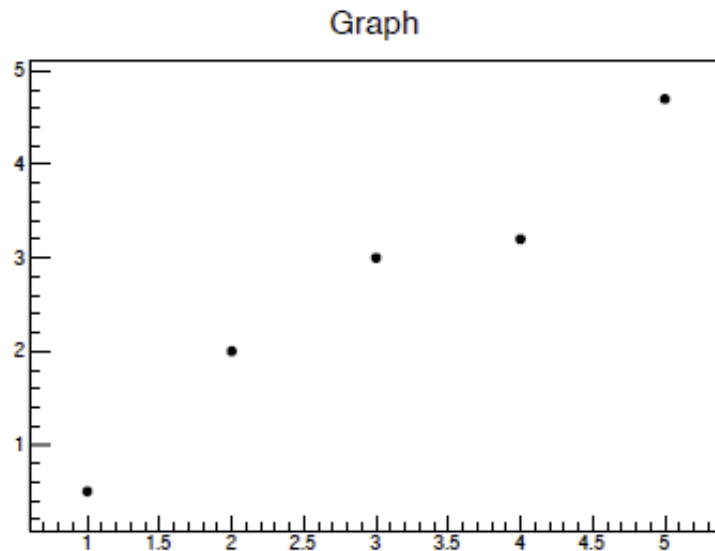
```
root [1] TGraph * g = new TGraph("XYData.txt");
```

# Displaying a Graph

- To display the graph:

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

- option “A” means displaying the axis,
- option “P” means displaying the points.

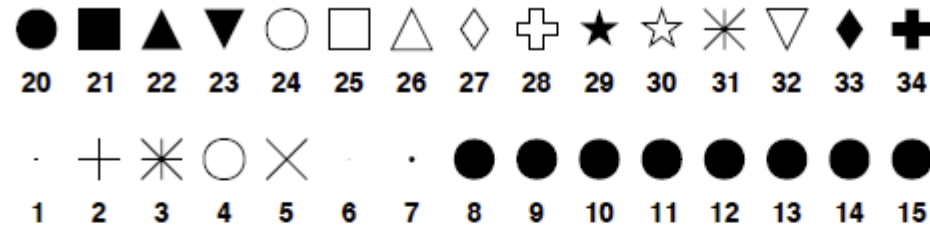


To change the point markers do:

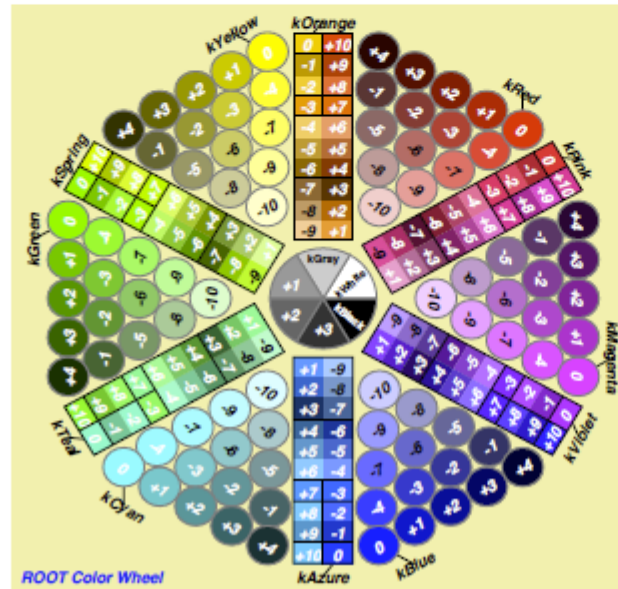
```
root [4] g->SetMarkerStyle(20);
```

# Markers and Colors

- Available markers in ROOT



- Available Colors

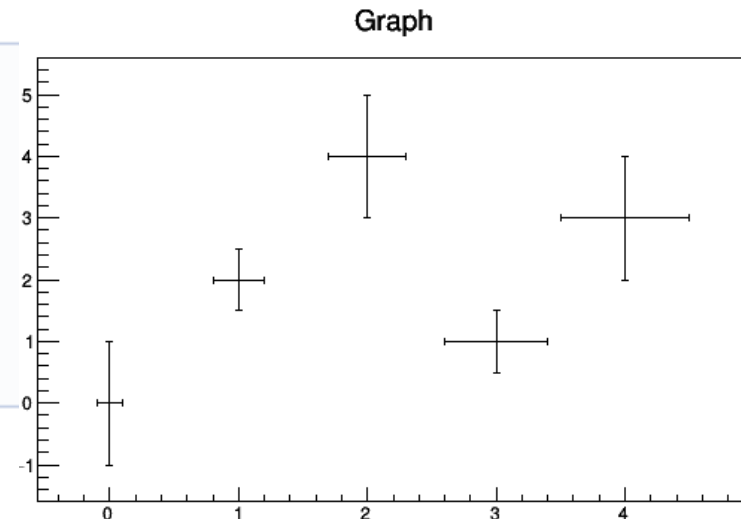


you can access  
them from the  
Canvas View Menu

# Types of Graphs

- ROOT provides various types of Graphs:
  - **TGraph** : (x,y) data points.
  - **TGraphErrors**:
    - (x,y) data points with error bars ( $\sigma_x, \sigma_y$ ).
  - **TGraphAsymmErrors**:
    - (x,y) data points with asymmetric error bars [ $(\sigma^-_x, \sigma^+_x), (\sigma^-_y, \sigma^+_y)$ ].

```
{  
  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;  
}
```



# 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.

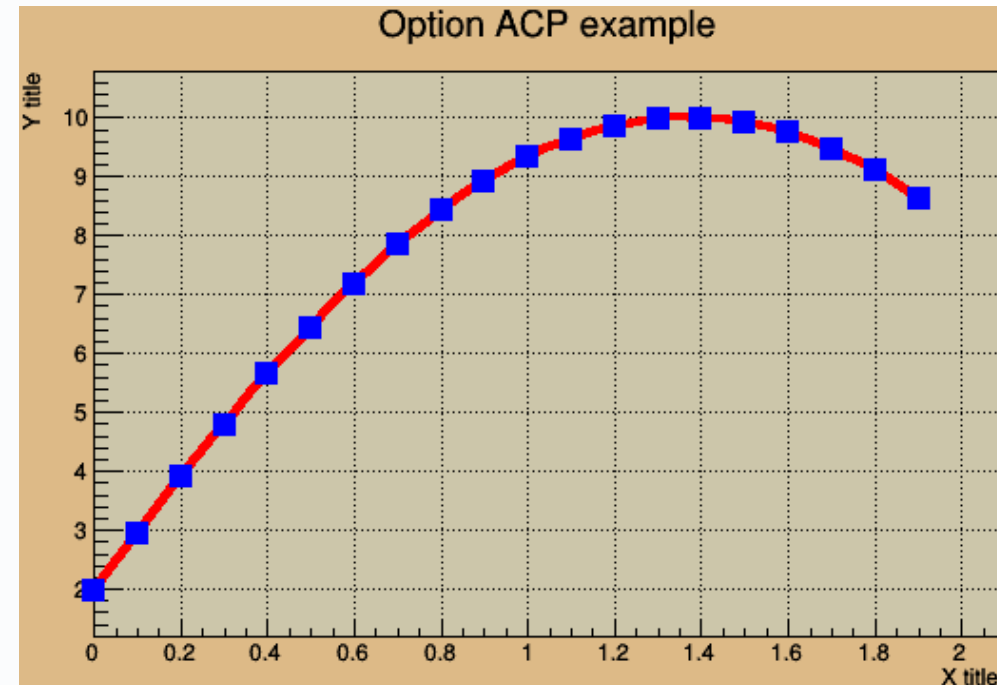
# Drawing options can be combined

```
{
    TCanvas *c1 = new TCanvas("c1","c1",200,10,600,400);

    c1->SetFillColor(42);
    c1->SetGrid();

    const Int_t n = 20;
    Double_t x[n], y[n];
    for (Int_t i=0;i<n;i++) {
        x[i] = i*0.1;
        y[i] = 10*sin(x[i]+0.2);
    }
    gr = new TGraph(n,x,y);
    gr->SetLineColor(2);
    gr->SetLineWidth(4);
    gr->SetMarkerColor(4);
    gr->SetMarkerSize(1.5);
    gr->SetMarkerStyle(21);
    gr->SetTitle("Option ACP example");
    gr->GetXaxis()->SetTitle("X title");
    gr->GetYaxis()->SetTitle("Y title");
    gr->Draw("ACP");

    // TCanvas::Update() draws the frame, after which one can change it
    c1->Update();
    c1->GetFrame()->SetFillColor(21);
    c1->GetFrame()->SetBorderSize(12);
    c1->Modified();
    return c1;
}
```

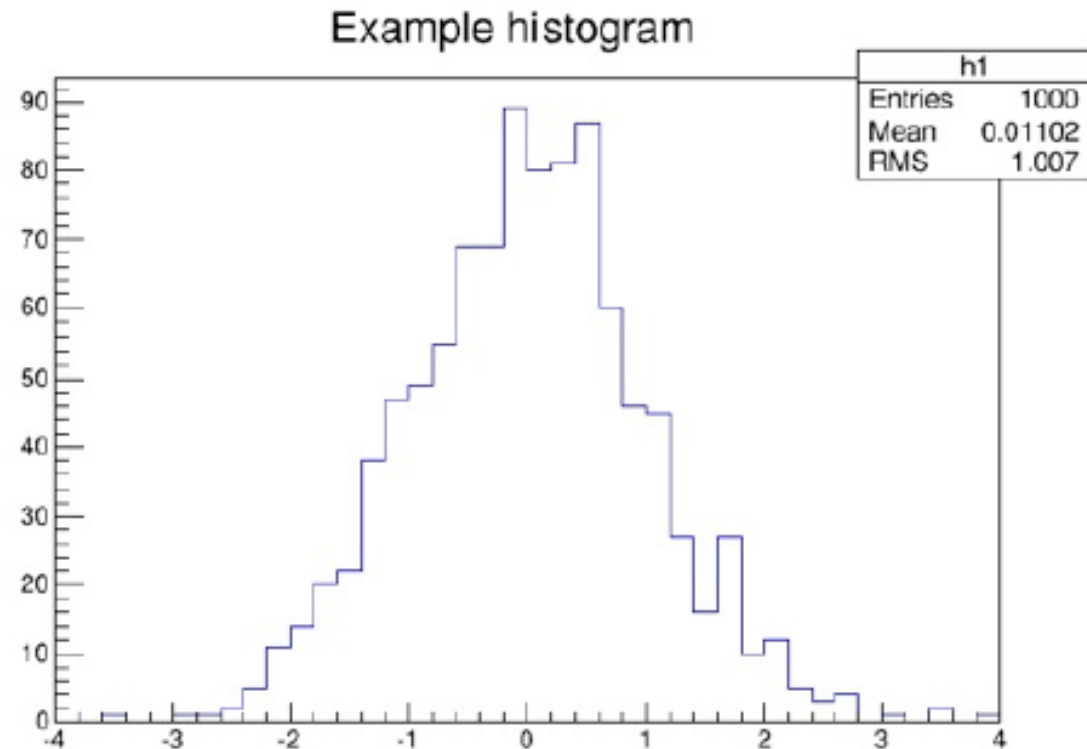




# Histograms

- 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](#).<sup>[1]</sup>
- 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.



# 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  
TH1D: one-dimension  
using double types

↑  
name

↑  
title

↑  
axis settings  
number  
of bins

↑  
min,max  
values

- Filling histogram:

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

Fill the histogram with one observation “

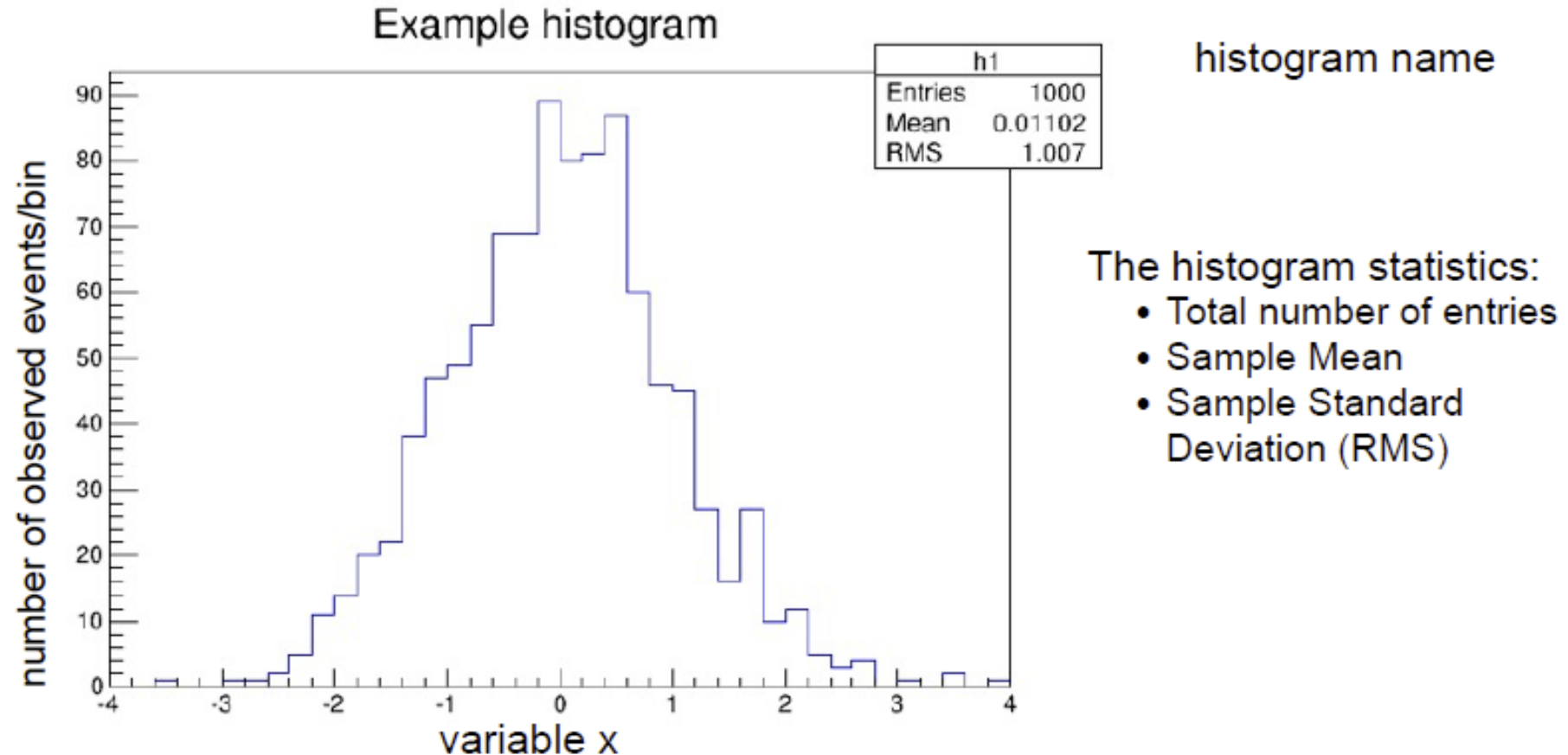
```
for (int i = 0; i<1000; ++i) {  
    double x = gRandom->Gaus(0,1);  
    h1->Fill(x);  
}
```

Fill the histogram with  
1000 gaussian distributed  
random numbers

# Displaying ROOT histograms

- Drawing histograms in a ROOT canvas:

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



# Histogram Statistics

- To extract statistics information from an histogram:

```
root [] h1->GetEntries()  
(const Double_t)1.0000000000000000e+03  
root [] h1->Integral()  
(const Double_t)1.0000000000000000e+03  
root [] h1->GetMean()  
(const Double_t)1.1017279203592710e-02  
root [] h1->GetMeanError()  
(const Double_t)3.1831174486931387e-02  
root [] h1->GetRMS()  
(const Double_t)1.0065901197694480e+00  
root [] h1->GetRMSError()  
(const Double_t)2.2508039332841407e-02  
root [] h1->GetSkewness()  
(const Double_t)1.1782073846449019e-01  
root [] h1->GetKurtosis()  
(const Double_t)2.5896196835884000e-01
```

# Histogram Drawing Options

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

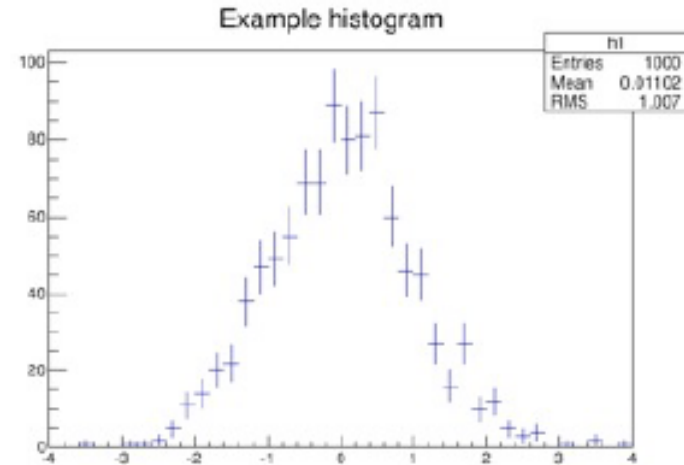
```
h1->Draw( "E" );
```

- “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:

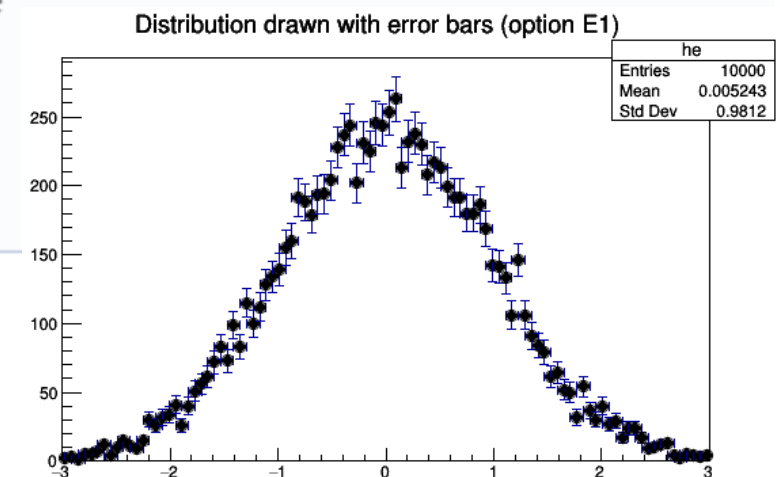
```
gPad->SetLogy( );
```



# 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;  
}
```





# 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