程序设计

- **程序由两个主要方面组成**
  - 算法的集合：将指令组织成程序来解决某个特定的问题
  - 数据的集合：算法在数据上操作以提供问题的解决方案
- 它们之间的关系就是所谓的程序设计方法(programming paradigm)
- **程序设计方法：**

  面向过程的编程（procedural programming）：
  - "what to do" in each step (Fortran, Pascal, C)

  面向对象的编程（Object-oriented programming）：
  - "what is" (object, their properties, etc)

面向对象编程

- **面向过程的程序**（如C语言、PASCAL、FORTRAN ）**设计：**

程序 =（算法(过程或函数)）+（数据结构）

缺点：不利于软件的重用，过程与数据相互依赖，无法分开维护，开发过程复杂，效率低。

- **面向对象的程序**（如C++）**设计：**

数据结构及其相关算法被封装成类，对象是类的一个实例。

对象=（算法+数据结构）；程序=（对象+对象+…）

优点：提高了程序的重用，简化程序设计，提高开发效率。

对象成为程序的基本构件，编程类似在搭积木。

面向对象编程

- **通过继承机制和动态绑定机制扩展了抽象数据类型**
  - 继承机制是对现有代码的重用
  - 动态绑定是指对公共接口的重用。
- **面向对象程序设计的基本特征**
  - **封装：** 实现了数据隐藏，保护对象的数据不被外界随意改变，使对象成了相对独立的功能模块。
  - **抽象：** 是特定属性的事物的一个概括，以隐藏事物固有的复杂性。
  - **继承：** 允许一个类继承其它类(称为基类)的属性和方法，该类称为派生类；是类的层次结构之间共享数据和方法的机制，允许和鼓励类的重用。
  - **多态：** 不同类的对象对同一消息作出不同的响应。

类与对象的概念

- 客观世界中任何一个事物都可以看成一个对象，对象之间通过一定的渠道相互联系。
- 从计算机角度来看，一个对象应包括两个要素：

  一是数据(即活动主体) ，二是需要进行的操作。

  对象就是一个包含数据以及与这些数据有关的操作的集合。
- 每一个实体都是对象，有一些对象具有相同的结构和特性。在C++中，对象的类型就称为“类”(c l a s s) , 即类代表了某一批对象的共性和特征。
- 类是对某一类对象的抽象，对象是某一种类的实例。C++中，可先声明类类型，然后去定义若干不同类型的对象，即对象就是一个类类型的变量，而类类型相当于产生对象的模板

类与对象的概念

- C++语言通过类将数据结构和与之相关的操作封装起来，形成一个整体，具有良好的外部接口，防止对数据结构内部未经授权的访问，提高了程序模块之间的独立性。
- C中结构struct 可把相关联的数据元素组成一个单独的统一体：只有数据，没有操作；C++的类既包含数据成员，又含有操作(称为成员函数) 。

```
struct  Student
{
   int num;     //学号
   float sum;   //总分
};
```

```
class Student
{
public:
  void   set_num(int); //成员函数
  void   set_sum(int); //成员函数
private:
  int num;             //数据成员
  float sum;
};
```

限制成员函数与数据

- 限制类与外部访问的接口：公共的public和受限制的（private和protected）
- 只有类本身才能访问该类受限制的数据，类本身负责在内部维护
- 用户通过授权访问和修改数据成员的成员函数访问/ 修改数据
- 减少类与其它代码的关联程度，做到功能独立
- 类的保护机制使得程序更加可靠和易于维护
- 类定义中，不写访问控制说明符时，默认为private
- protected和private的区别，体现在类的继承中

类class

- 成员函数对数据成员的引用可在其声明之前：C++编译器先扫描类的数据成员的说明后，再处理成员函数代码体。
- 关键字public 、private和protected表示存储控制类型。

private声明的数据成员和成员函数是“私有的”，外部不能调用
public方式声明的成员是“公有的”，外界可以调用；
protected成员同私有成员相似，不能被外部调用，但可被派生类的成员函数调用。

- 存储控制类型的声明出现的次数及顺序是任意的。

```
void  abc(){
    Student a;     //定义类对象
    a.sum=181.5;   //错误，不能访问sum ，因为它是私有成员
    a.set_sum(100); //正确，使sum赋值为100
}
```

Introduction to ROOT

https://root.cern.ch/

搜索...

正在等待 root.cern.ch

ROOT的主页，由CERN维护。

ROOT
Data Analysis Framework

Download Documentation News Support About Development Contribute

Getting Started | Reference Guide | Forum | Gallery

ROOT is ...

基于面向对象设计的数据分析软件框架！

A modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

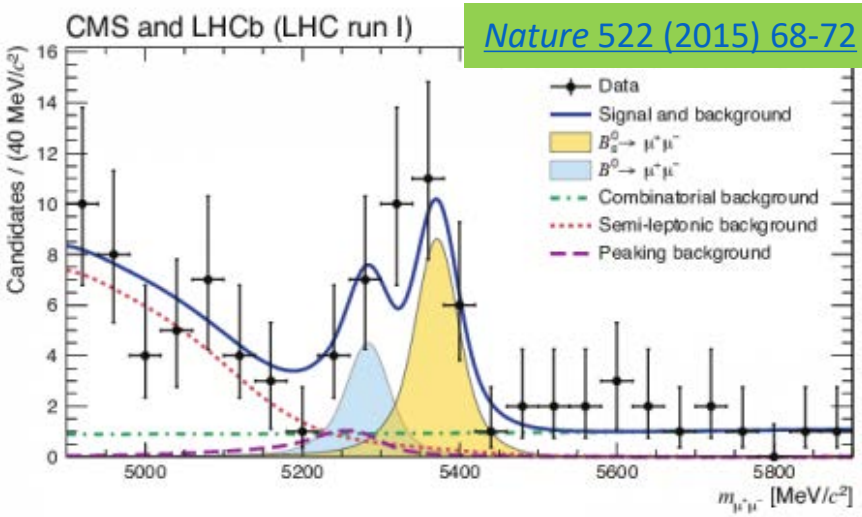
Start from examples or **try it in your browser!**

Download or Read More ...

Under the Spotlight

2018-01-17 **ROOT Users' Workshop 2018**

The ROOT team would like to invite you to the 11th ROOT Users' Workshop.



Nature 522 (2015) 68-72

Other News

2016-04-16 The status of reflection in C++

2016-01-05 Wanted: A tool to 'warn' user of inefficient (for I/O) construct in data model

What’s ROOT ?

Brief overview of ROOT and its functionality before starting to use it

ROOT in a nutshell

- Framework for large scale data handling
- Provides, among others,
  - an efficient data storage, access and query system (PetaBytes)
  - advanced statistical analysis: histogramming, fitting, minimization and multi-variate analysis algorithms
  - scientific visualization: 2D and 3D graphics, Postscript, PDF, LateX
  - geometrical modeler
  - PROOF parallel query engine

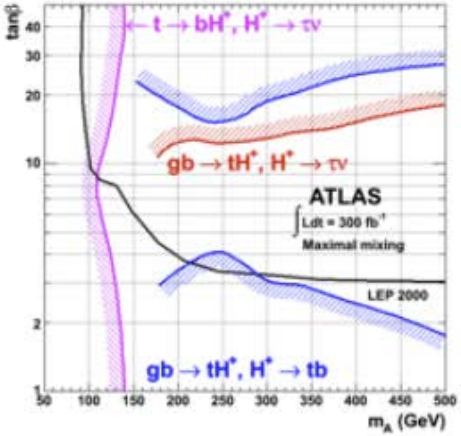
- An Open Source Project

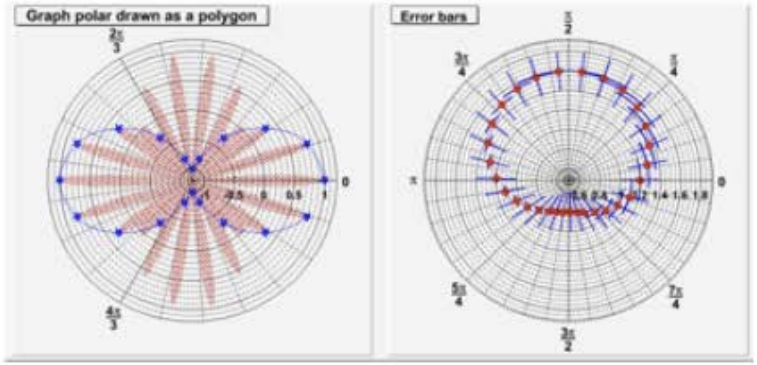
Why ROOT ?

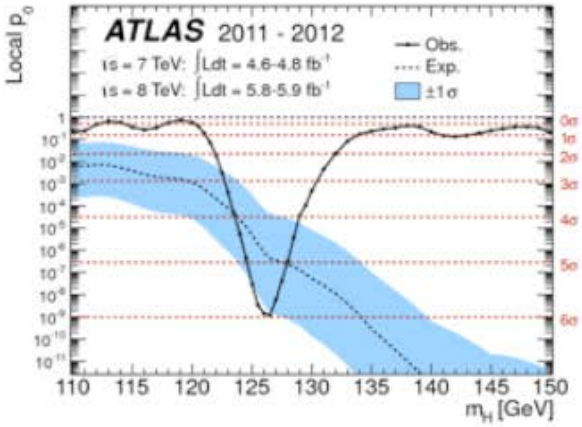
- The analysis of data coming from LHC experiments and not only requires a powerful and general toolkit
  - Visualisation
  - Statistical studies
  - Data reduction
  - Multivariate techniques
- A scalable and reliable persistency method is needed to write the data on disks and tapes.

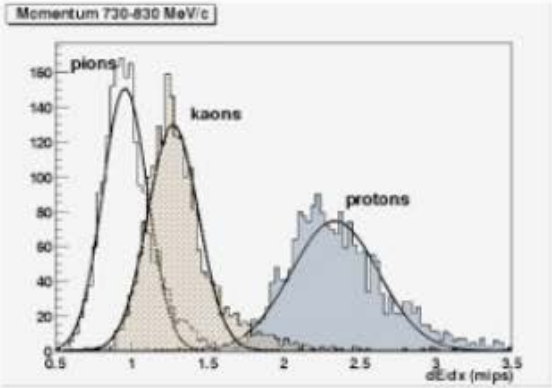
Graphics in ROOT

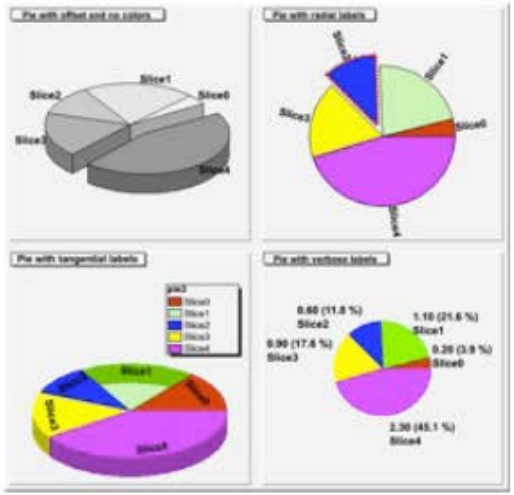
- Many plots available for data analysis:

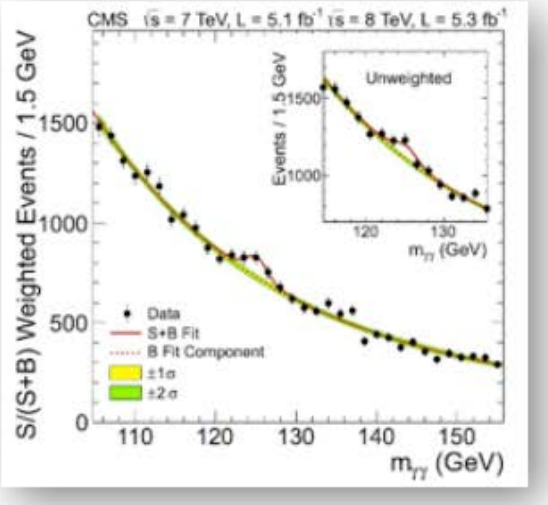






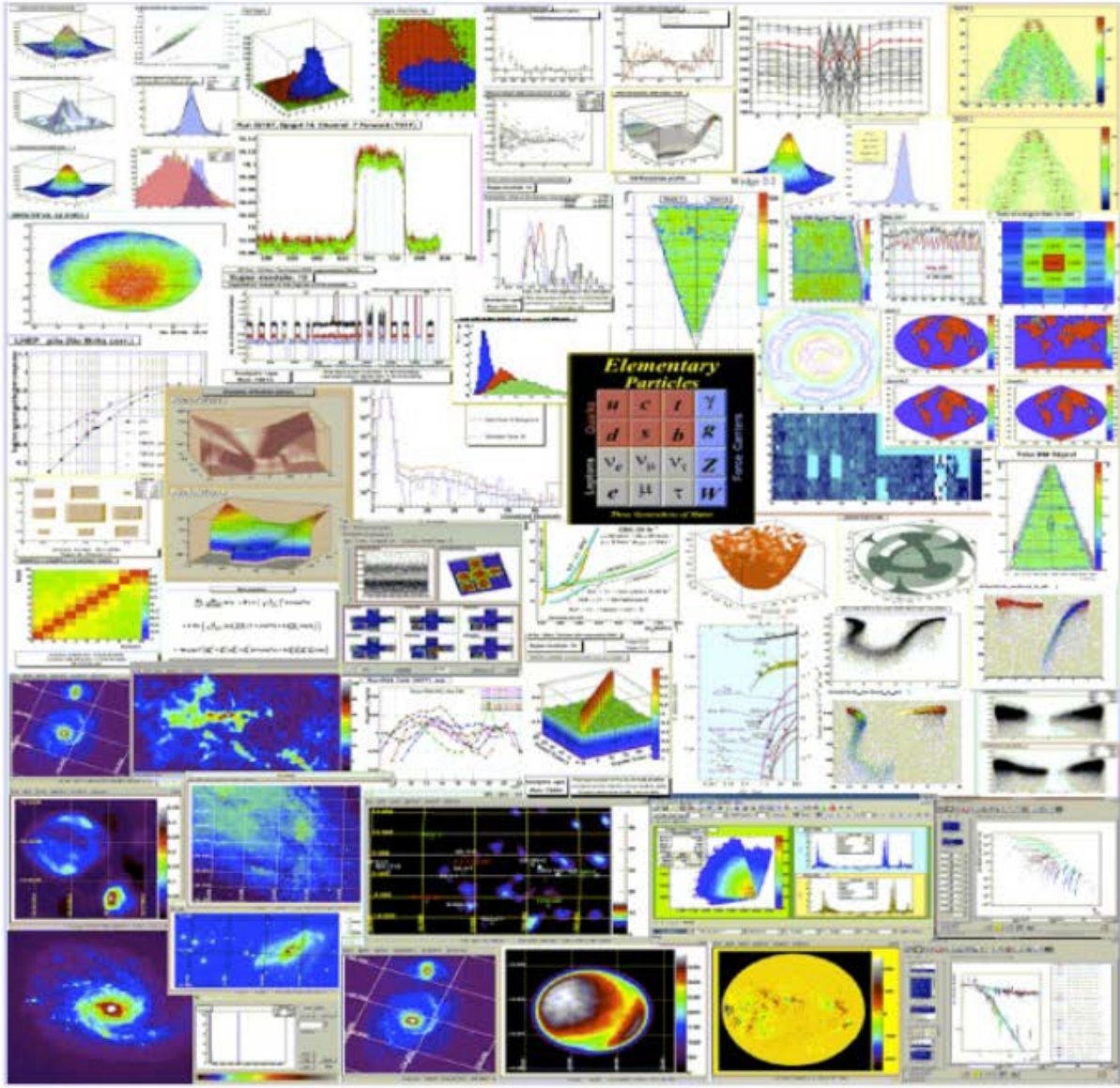




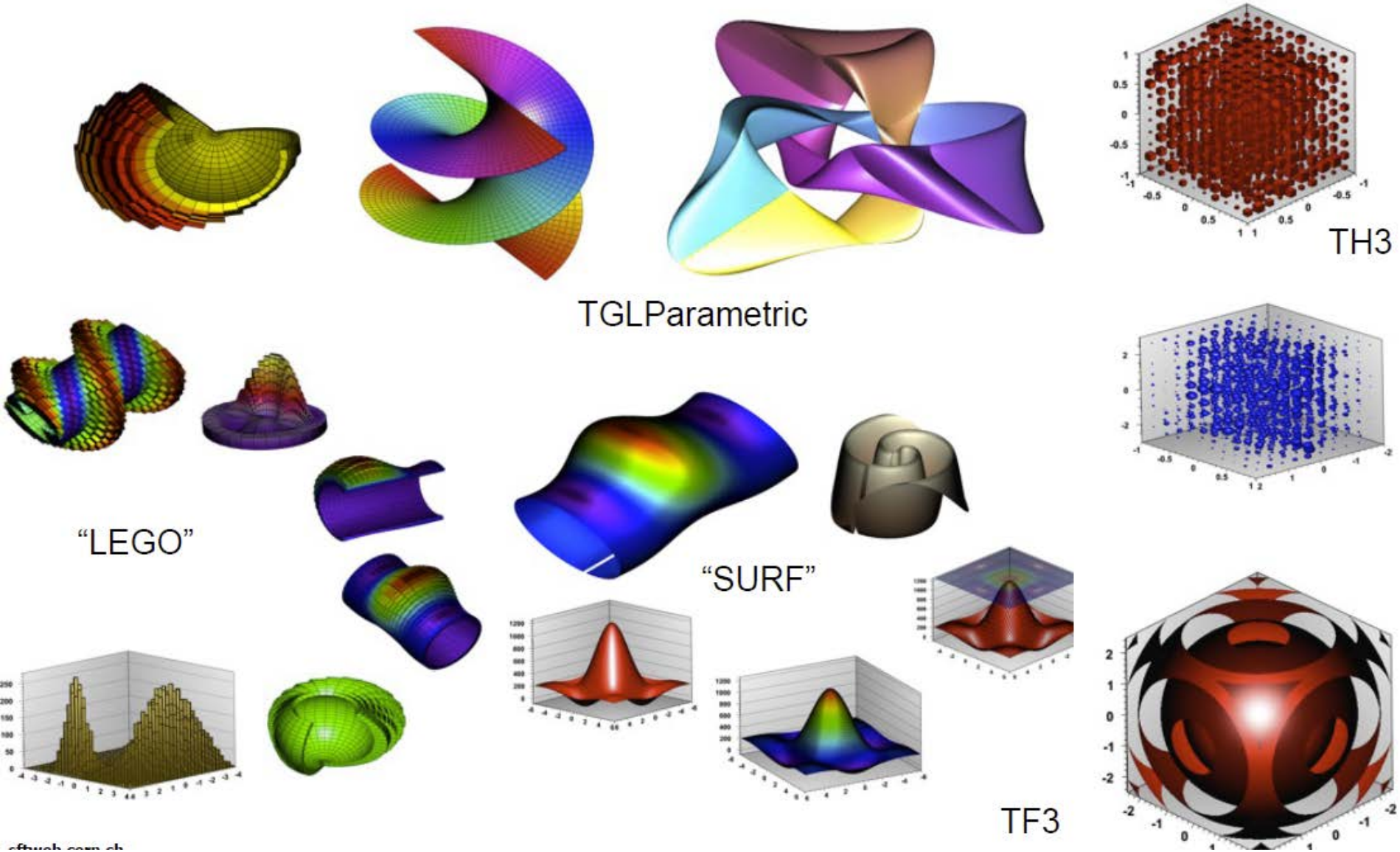


2D Graphics

- New functionality added at every new release
- Always new requests for new style of plots
- Can save graphics in many formats: `ps,pdf,svg,jpeg, png,c,root`

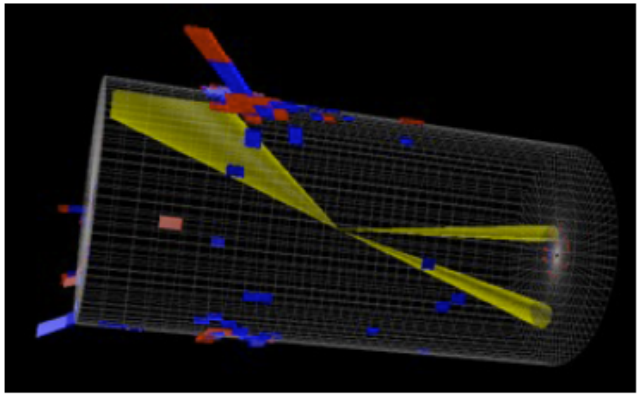


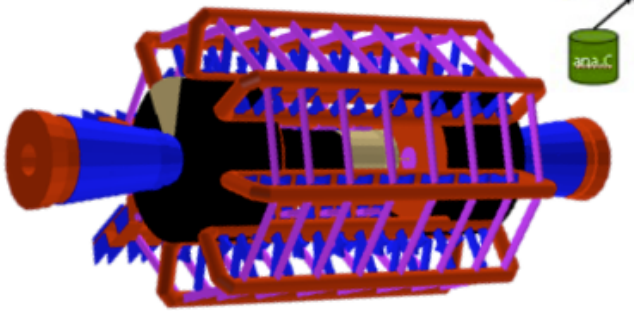
3D Graphics

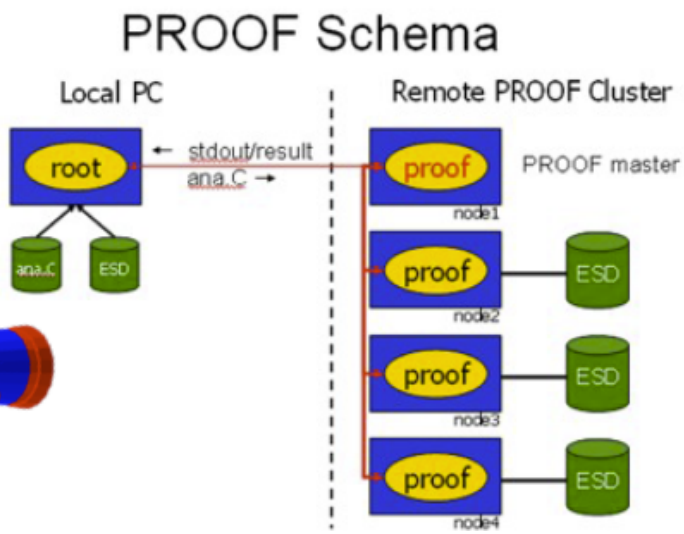


Other ROOT Features

- Geometry Toolkit
  - Represent geometries as complex as LHC detectors.
- Event Display (EVE)
  - Visualise particles collisions within detectors.
- PROOF: Parallel ROOT Facility
  - Multi-process approach to parallelism
  - A system to run ROOT queries in parallel on a large number of distributed computers
  - Proof-lite: does not need a farm,
    uses all the cores on a desktop machine







Introduction

Start Using ROOT

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 !*

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@lxslc601 29]$ root
  ******************************************
  *                                        *
  *       W E L C O M E  to  R O O T       *
  *                                        *
  *   Version   5.34/09      26 June 2013  *
  *                                        *
  *  You are welcome to visit our Web site *
  *          http://root.cern.ch           *
  *                                        *
  ******************************************

ROOT 5.34/09 (v5-34-09@v5-34-09, Jun 26 2013, 17:10:36 on linuxx8664gcc)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] gROOT->GetVersion();
root [1] gROOT->GetVersion()
(const char* 0x7fbbe761cb51)"5.34/09"
root [2]
```

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

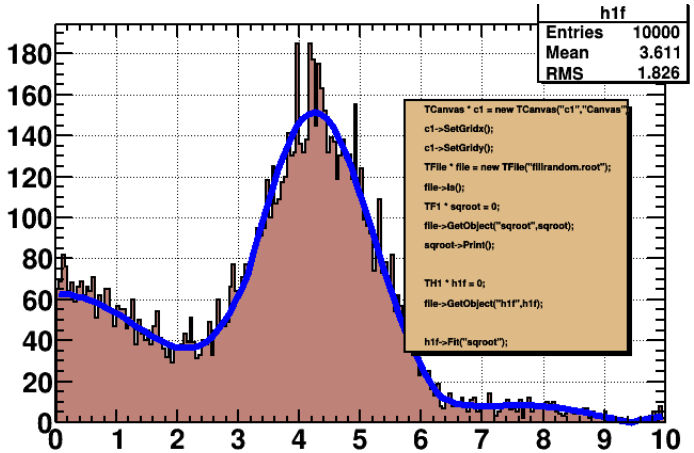
```
Root> .x demos.C
```

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：




```
---------------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)
```

ROOT As Pocket Calculator

Calculations:

```
root [0] sqrt(42)
(const double)6.4807406984078603 8e+00
root [1] double val = 0.17;
root [2] sin(val)
(const double)1.6918234906699602 9e-01
root [2] TMath::Erf(1.)
(Double_t)8.4270079294971478 3e-01
```

Uses C++ Interpreter CINT

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

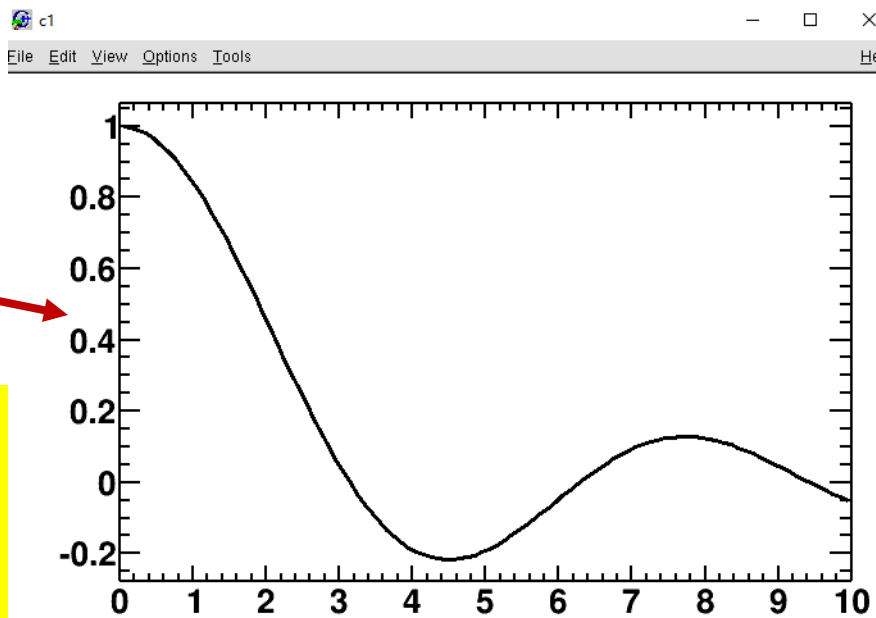
- Start using one of basics ROOT classes - the function class TF1:

```
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]:

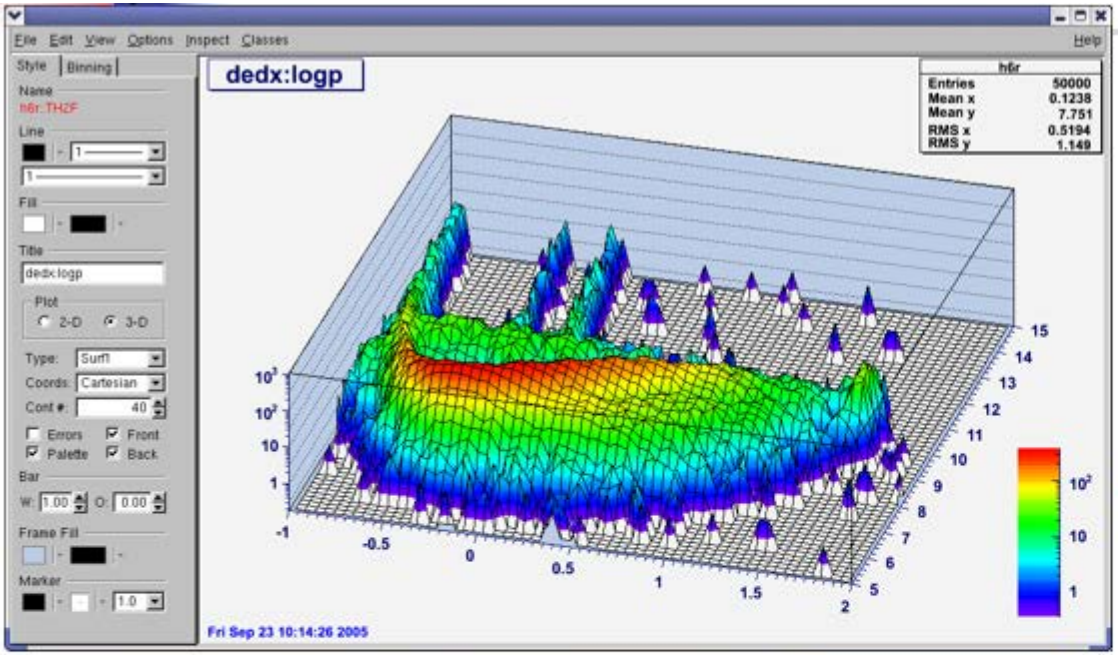
```
root [2] TF1 * f2 = new TF1("f2","[0]*sin([1]*x)/x",0,10);
```

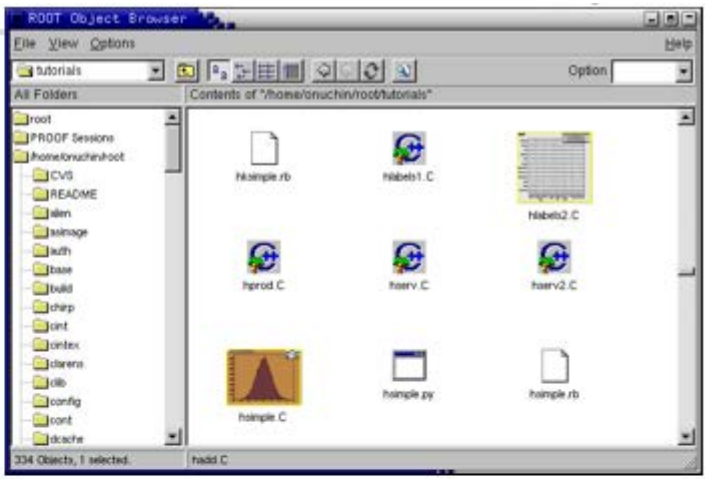
- Need to set the parameter values (by defaults the parameters have zero initial values):

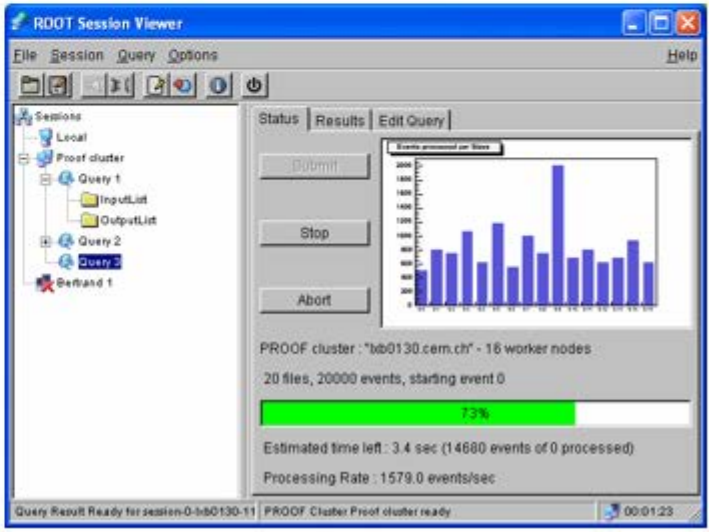
```
root [3] f2->SetParameter(0,1);
root [4] f2->SetParameter(1,1);
root [5] f2->Draw();
```

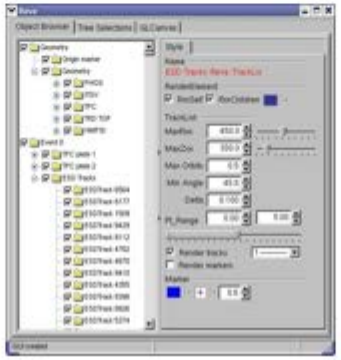
- Can also use the GUI (function editor) to change the function parameters.

GUI (Graphical User Interface)



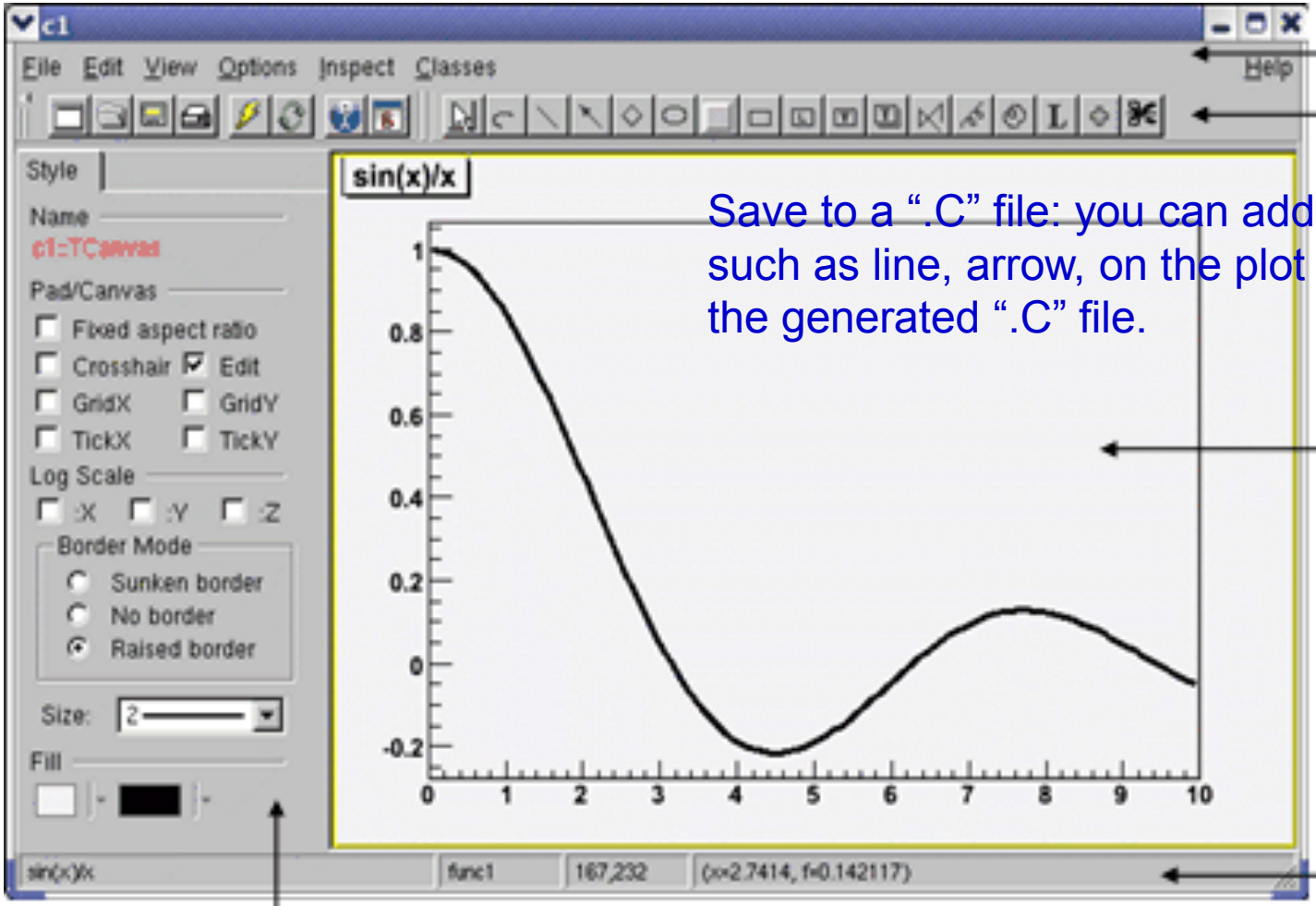






GUI Editor for ROOT Objects

- 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.

Menu bar

Tool bar

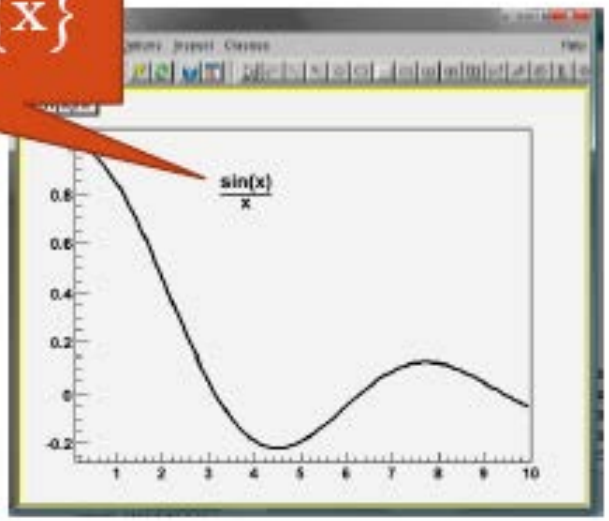
Canvas

Status bar

Editor frame

Fast Generate Your Macro Code

#frac{sin(x)}{x}



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,"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

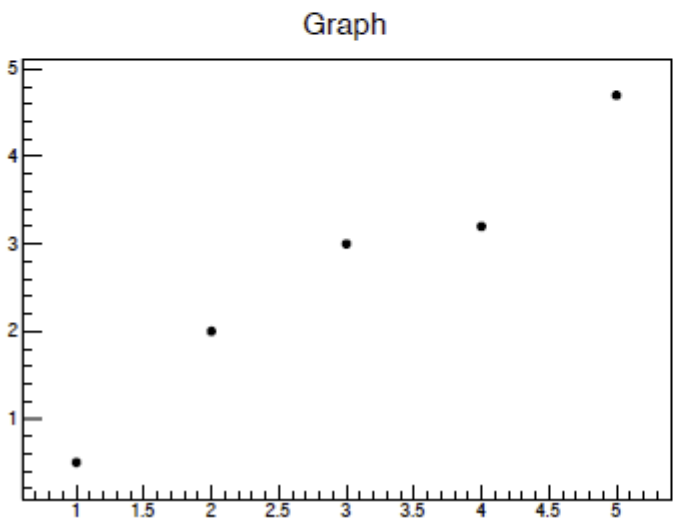
```
root [1] TGraph * g = new TGraph("XYData.txt");
```

Displaying a Graph

- To display the graph:

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

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

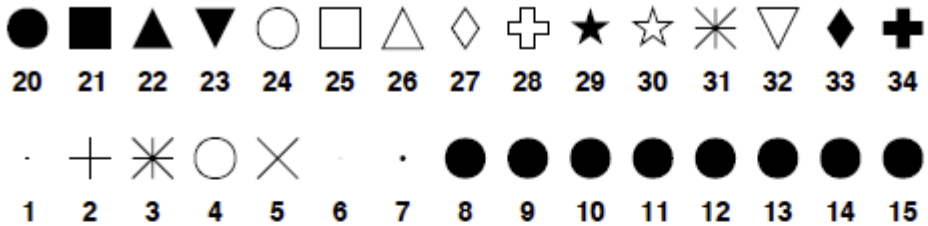


To change the point markers do:

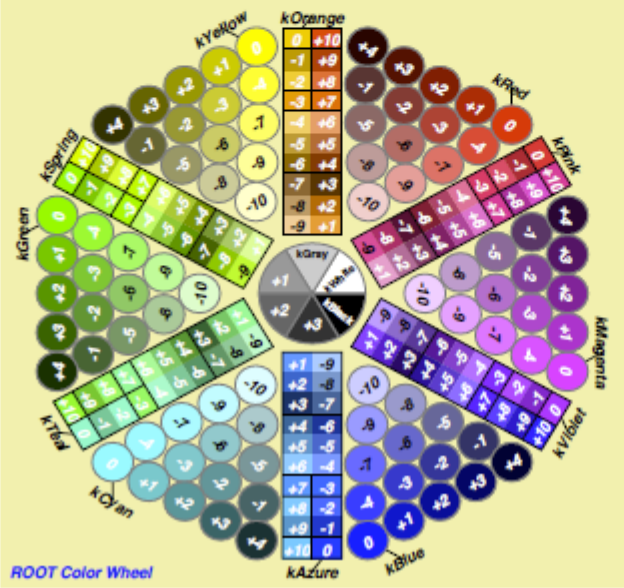
```
root [4] g->SetMarkerStyle(20);
```

Markers and Colors

- Available markers in ROOT



- Available Colors

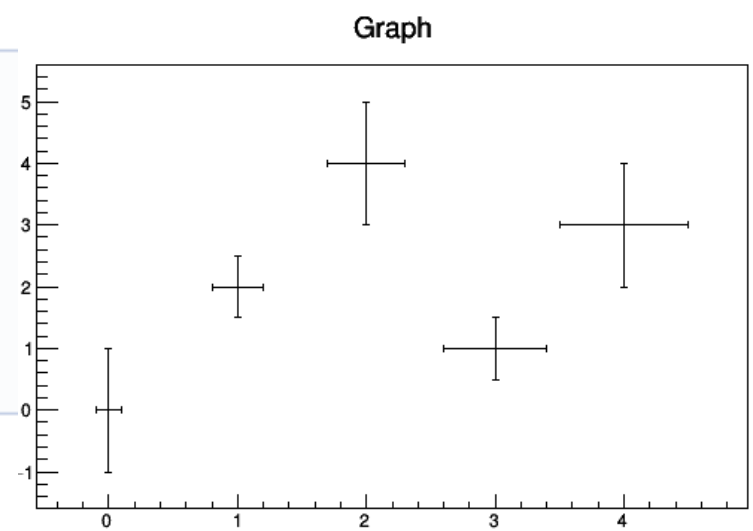


you can access them from the Canvas View Menu

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;
}
```



Graphs Drawing Options

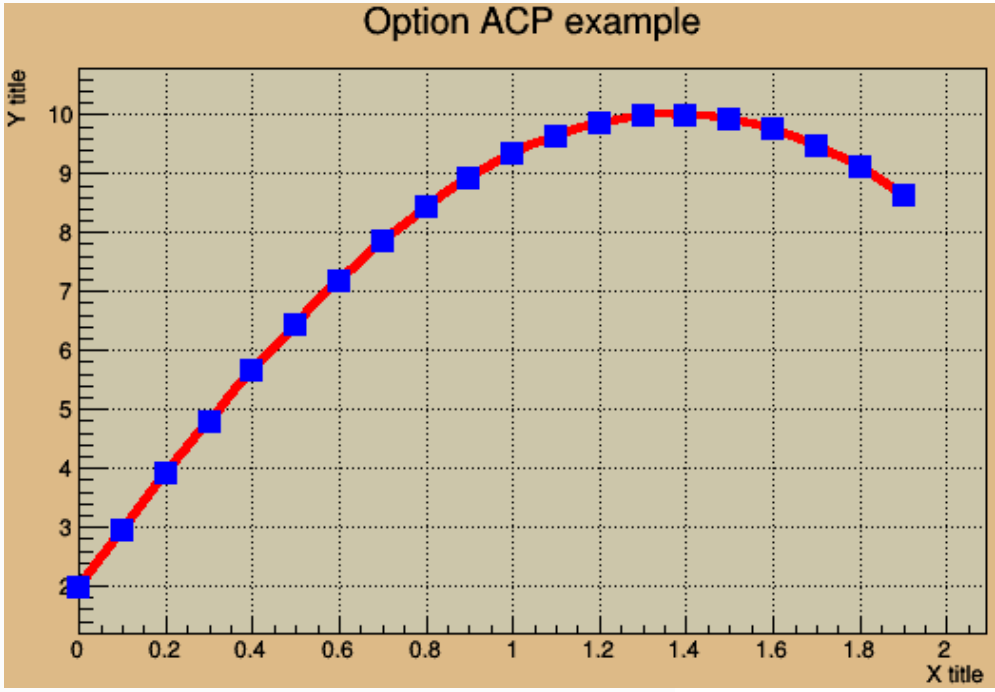
- The drawing of Graphs is done via the **`TGraphPainter`**
  - see https://root.cern.ch/doc/master/classTGraphPainter.html
  - the documentation lists all drawing options for the different types of graphs available in ROOT

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;
}
```



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.

Histograms 直方图

Outline

- We have seen:
  - what is a histogram;
  - how to create a one-dimensional histogram;
  - how to draw the histogram
- We will see and work on with the exercises
  - how to extract information from an histogram
  - how to manipulate histograms
  - histograms in multi-dimension
  - what is a profile histogram
  - weighted histograms
  - sparse histograms
- More detailed on the ROOT Graphics Pad and the Canvas
- Visualization techniques in ROOT

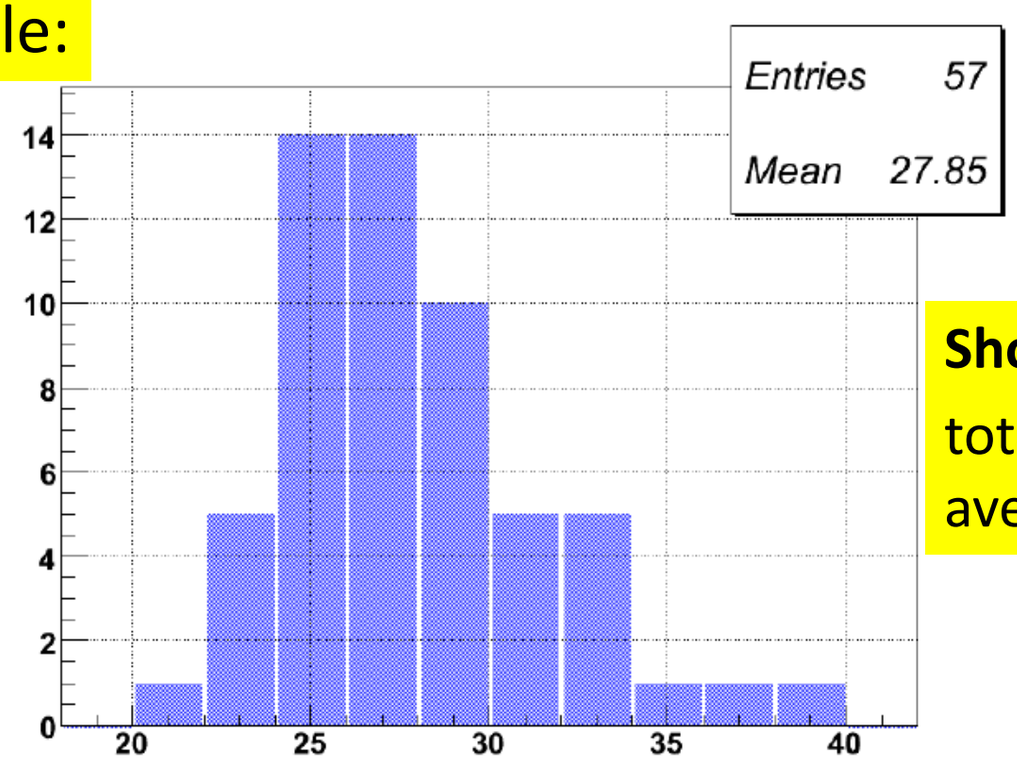
What is Histogram

- In statistics, a graphical representation, showing a visual impression of the distribution of data.
- An estimate of the probability distribution of a variable
- Each rectangle erected over discrete intervals (bins)
- Area of a rectangle equals to the frequency of the observations in the interval (bin).
- Height of a rectangle also equals 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.
- A histogram may also be normalized displaying relative frequencies, with the total area equaling 1.

Example:

Table of Ages (binned)

Age	Number
20-22	1
22-24	5
24-26	14
26-28	14
28-30	10
30-32	5
32-34	5
34-36	1
36-38	1
38-40	1



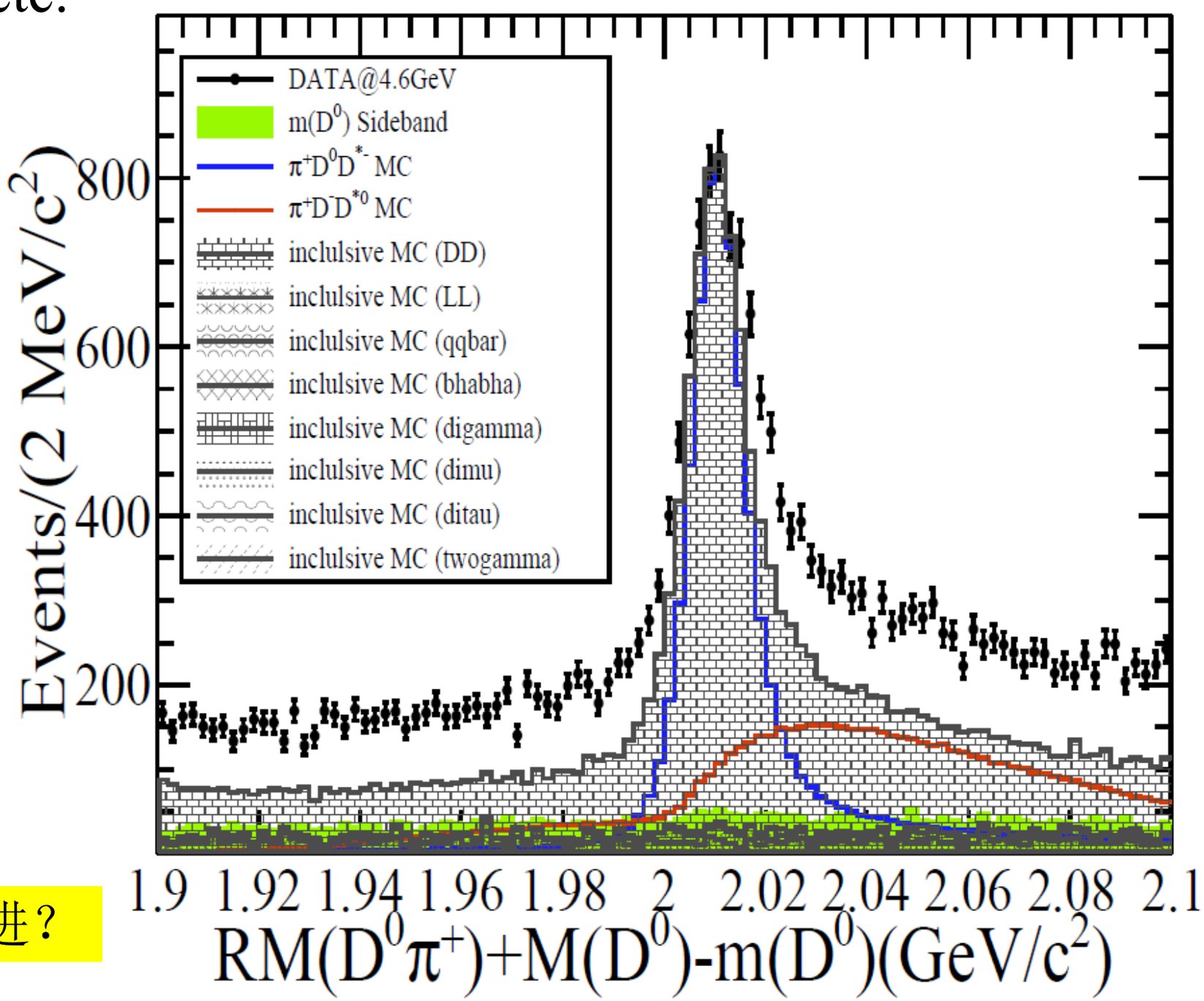
Shows distribution of ages:

total number: 57 participants;

average: 27 years 10 months 6 days

What Histogram Tells

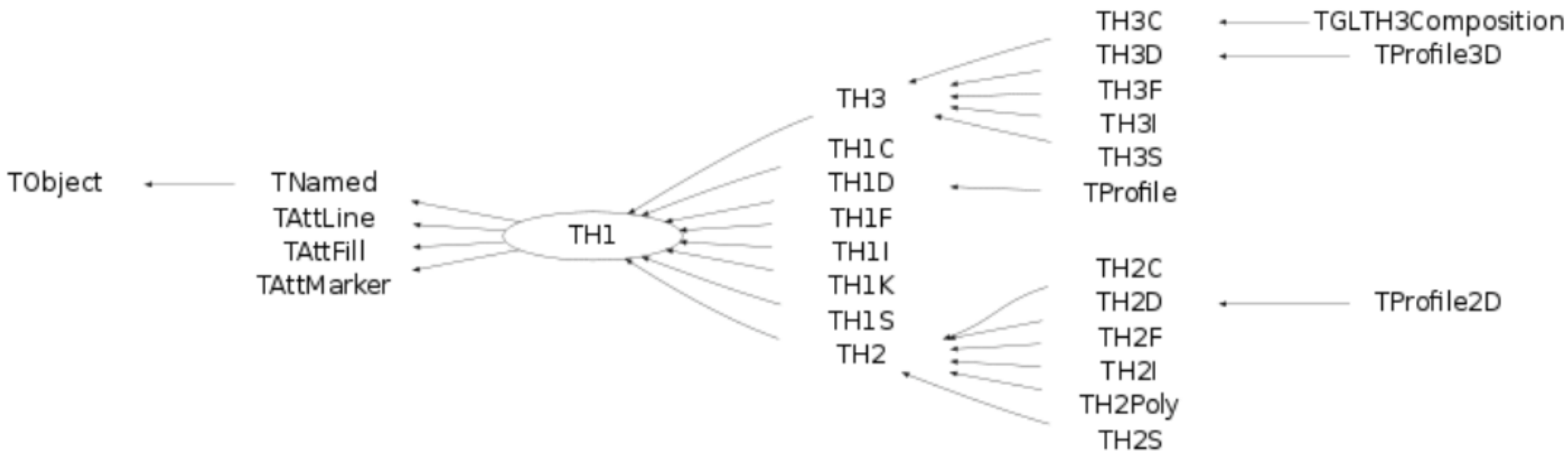
- Presenting data to determine which causes dominate
- Understanding the distribution of occurrences of different problems, causes, consequences, etc.
- What is the most common system response?
- What distribution (center, variation and shape) does the data have?
- Does the data look symmetric or is it skewed to the left or right?



右图有什么地方需要改进？

Histogram Classes

ROOT supports the following histogram types:



1-D histograms:
- **TH1C**: one byte (255) per channel.
- **TH1S**: one short (65535) per channel.
- **TH1I**: one integer (2147483647) per channel.
- **TH1F**: one float (7 digits) per channel.
- **TH1D**: one double (14 digits) per channel.

2-D histograms: …
3-D histograms: …
Profile histograms
- **TProfile**: one dimensional profiles
- **TProfile2D**: two dimensional profiles
- …

- TH1 is the base class for all Histogram classes.
- TH1,TH2,TH3 are generic. Specialized should be used for constructing the objects.
- Most used classes are TH1(2,3)F and TH1(2,3)D.

Creating Histograms

TH1 constructor: the name of the histogram (name), the title (title), the number of bins (nbinsx), the x minimum (xlow), x maximum (xup) and array of low-edges for each bin (xbins).

```
 TH1F();
 TH1F(const TVectorF& v);
 TH1F(const TH1F& h1f);
 TH1F(const char* name, const char* title, Int_t nbinsx,
const Float_t* xbins);
 TH1F(const char* name, const char* title, Int_t nbinsx,
const Double_t* xbins);
 TH1F(const char* name, const char* title, Int_t nbinsx,
Double_t xlow, Double_t xup);
 virtual ~TH1F();
```

For equidistant bins:

```
 TH1F * h1 = new TH1F("h1","h1 title",100,0,4.4);
 TH2F * h2 = new TH2F("h2","h2 title",40,0,4,30,-3,3);
```

For non-equidistant bins:

```
TH1D * h1 = new TH1D("h1","Example histogram",nbins,xbins);
```

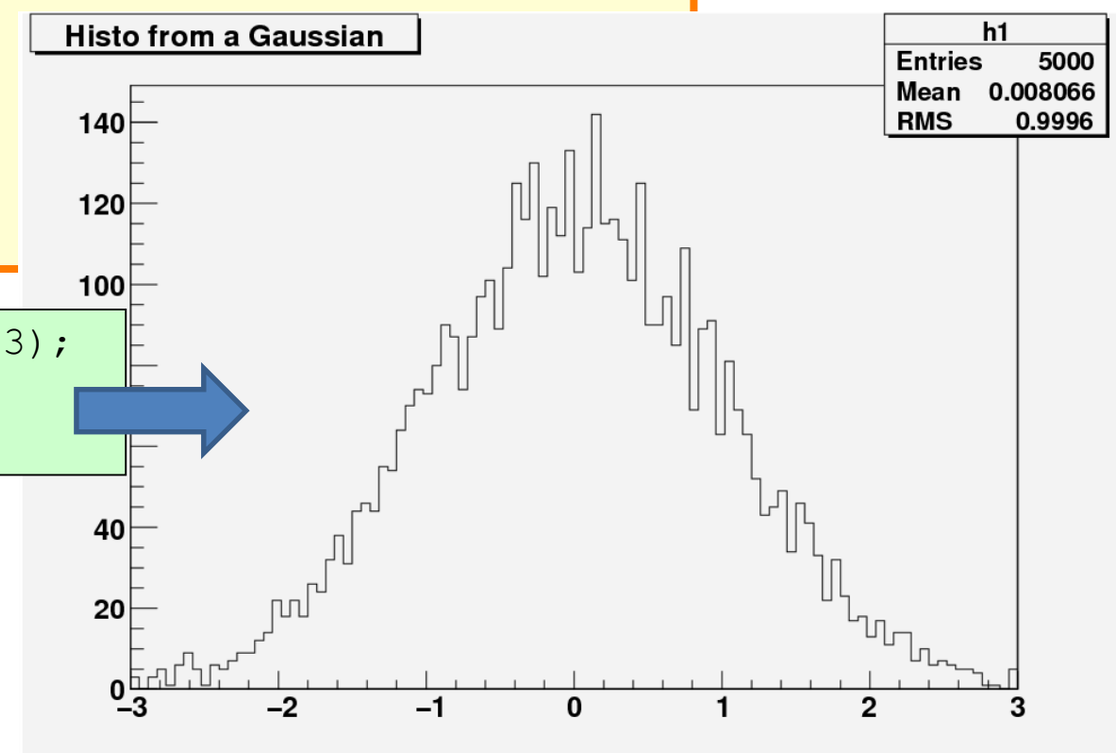
- `xbins:` array of (`nbins+1`) values with the bin-edges

Filling Histograms

- A histogram is typically filled with statements like:

```
h1->Fill(x);
h1->Fill(x,w); //with weight
h2->Fill(x,y);
h2->Fill(x,y,w);
h3->Fill(x,y,z);
h3->Fill(x,y,z,w);
```

```
root[] TH1F h1("h1","Histo from a Gaussian",100,-3,3);
root[] h1.FillRandom("gaus",5000);
root[] h1.Draw();
```



- Increment a bin number by calling
  TH1::AddBinContent()
  replace the existing content via
  TH1::SetBinContent()
  access the bin content of a given bin:
  TH1::GetBinContent()

```
Double_t binContent = h->GetBinContent(bin);
```

Drawing Histograms

- call the **`Draw`** method of a histogram (**TH1::Draw**) to draw the histogram
- it creates a THistPainter object and saves a pointer to painter as a data member of the histogram
- By default, the **TH1::Draw** clears the pad before drawing the new image of the histogram. You can use the "SAME" option to leave the previous display in tact and superimpose the new histogram.
- The same histogram can be drawn with different graphics options
- Most options can be concatenated without spaces or commas
- The options are not case sensitive
- You can also set the default drawing option with **TH1::SetOption**. To see the current option use **TH1**::GetOption.

Try examples in tutorials: *hist/draw2dopt.C*

Axis and Bins

- The histogram class has an axis class which contains the bins

```
TAxis * axis = h1->GetXaxis();
```

  – one can query the number of bins, lower/upper bin edge from the axis :

```
axis->GetNbins();
axis->GetBinLowEdge(bin_number);
axis->GetBinCenter(bin_number);
axis->GetBinUpEdge(bin_number);
```

  – for the corresponding bin number given x value use

```
bin_number = axis->FindBin(x_value);
```

  – one can set the axis range (e.g. for zooming the histogram)

```
axis->SetRange(firstbin,lastbin);
```

  – N.B. : axis bin number starts from 1
    - bin number 0 is the **Underflowbin**
    - bin number `nbin+1` is the **Overflowbin**

Giving Titles to the X, Y and Z Axis

Because the axis title is an attribute of the axis,
you have to get the axis first and then call **TAxis**::SetTitle.

```
h->GetXaxis()->SetTitle("X axis title");
h->GetYaxis()->SetTitle("Y axis title");
```

The histogram title and the axis titles can be any **TLatex** string.

```
h->GetXaxis()->SetTitle("E_{T}");
```

specify the histogram title and the axis titles at creation time given in the "title" parameter. They must be separated by "**;**":

```
TH1F* h=new TH1F("h","Histogram title;X Axis;Y Axis;Z Axis",100,0,1);
TH1F* h=new TH1F("h","Histogram title;;Y Axis",100,0,1);
TH1F* h=new TH1F("h",";;Y Axis",100,0,1);
h->SetTitle("Histogram title;Another X title Axis");
```

Setting Axis Attributes

Use TH1::GetXaxis() to get the pointer to the TAxis object to set the Axis attributes

See Methods in
Taxis class

```
TAxis*     TH1::GetXaxis() const
TAxis*     TH1::GetYaxis() const
TAxis*     TH1::GetZaxis() const
```

```
h->GetXaxis()->SetLabelColor(color)
               SetLabelFont(font)
               SetLabelOffset(offset)
               SetLabelSize(size)
               SetTickLength(length)
               SetTitleOffset(offset)
               SetTitleSize(size)
               SetTitleColor(color)
               SetTitleFont(font)
```

Getting Data From an Histogram

- To extract content and error from bin number `ibin` of the histogram `h1`:

```
root [1] h1->GetBinContent(ibin)
(const Double_t)1.10000000000000000e+01
root [2] h1->GetBinError(ibin)
(const Double_t)3.31662479035539981e+00
```

- By default histograms have a bin error equal to $\sqrt{N}$, where N is the bin content.
  - It is assumed that the observed bin content follows a Poisson distribution.
- One can set a different error for each bin:

```
root [1] h1->SetBinError(ibin,error)
```

- N.B: when setting the error in each bins, the histogram will store them in an internal array and will change its default style when drawn

Histogram Operations (1)

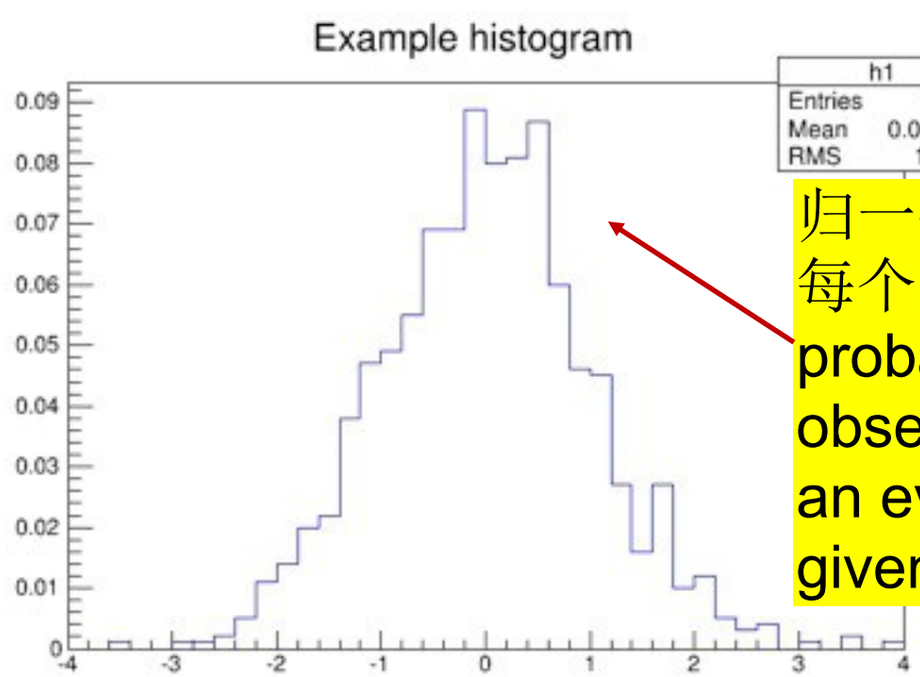
- Histogram scaling (normalization) 直方图的归一化
  - useful for plotting histograms in the same pad
    （归一化到同一积分可直观比较两种分布之间的差别）
  - useful for seeing histogram as an estimate of a probability density function (PDF) （可直观估计几率密度函数）

Method 1:

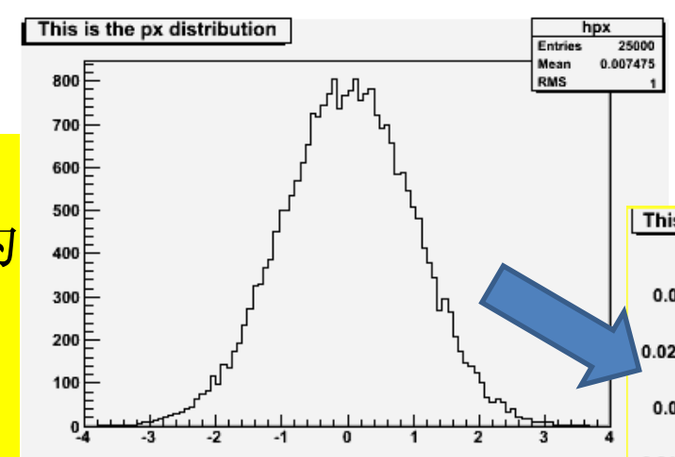
```
Double_t scale = norm/h->Integral();
h->Scale(scale); //归一化到norm（=1.0）
```

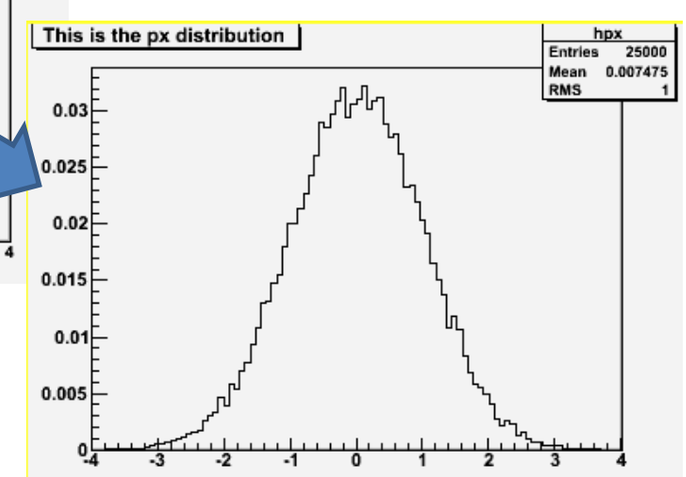
Method 2:

```
h->DrawNormalized("option_t", norm);
```



归一化到1后，每个bin的值为probability of observing an event in a given bin





N.B. : After scaling the error will not be correct. We will see later how to have correct errors
归一化后，BinContent变了，BinError也变化了，需要确保误差运算正确无误。

Histogram Operations (2)

- Add Histograms: 直方图的相加和相减
  - merge two histograms which have same axis:

```
TH1::SetDefaultSumw2();
TH1D *h3 = new TH1D("h3","h1+h2",nbin,xmin,xmax);
h3->Add(h1,h2,a,b); //h3 = a*h1+b*h2 (一般a=b=1)
```

相加：常用于实验数据的叠加。

  - can also be used to subtract histograms

```
h3->Add(h1,h2,a,-b); //h3 = a*h1-b*h2 (一般a=b=1)
```

相减：常用于实验中本底的扣除。

注意：

1. 两个直方图的区间大小和区间数必须相同，才能操作。
2. 在归一化和加减乘除中，要正确得到统计误差，需要调用： TH1::SetDefaultSumw2(); 或对每个直方图（如his）调用 his->Sumw2();

静态函数SetDefaultSumw2中的注释：

// static function. When this static function is called with sumw2=kTRUE,
// all new histograms will automatically activate the storage of the sum of squares of errors,
// ie TH1::Sumw2 is automatically called.

Histogram Operations (3)

- Divide Histograms: 直方图相除和相乘

```
TH1D *h3 = new TH1D("h3","h1/h2",nbin,xmin,xmax);
h3->Sumw2();
h3->Divide(h1,h2,a,b);
H3->Multiply(h1,h2,a,b);
```

相除：常用于实验中的效率的估计。
相乘：常用于实验中的对分布的修正（如效率修正等）。

– if h1 is a subset of h2, the bin content of h3 is binomially distributed ➠use option "B" to get the correct bin errors (h1和h2不独立：如h1包含于h2)

```
h3->Divide(h1,h2,a,b,"B");
```

二项分布误差：

$$\sigma = \sqrt{\frac{\frac{n_1}{n_2}(1-\frac{n_1}{n_2})}{n_2}}$$

Miscellaneous Operations

TH1::`Smooth()` - smoothes the bin contents of a 1D histogram.

TH1::`Integral(Option_t *opt)` - returns the integral of bin contents in a given bin range. If the option "width" is specified, the integral is the sum of the bin contents multiplied by the bin width in x.

TH1::`GetMean(int axis)` - returns the mean value along axis.

TH1::`GetRMS(int axis)` - returns the Root Mean Square along axis.

TH1::`GetEntries()` - returns the number of entries.

TH1::`Add(const TH1* h1, Double_t c1 = 1)`

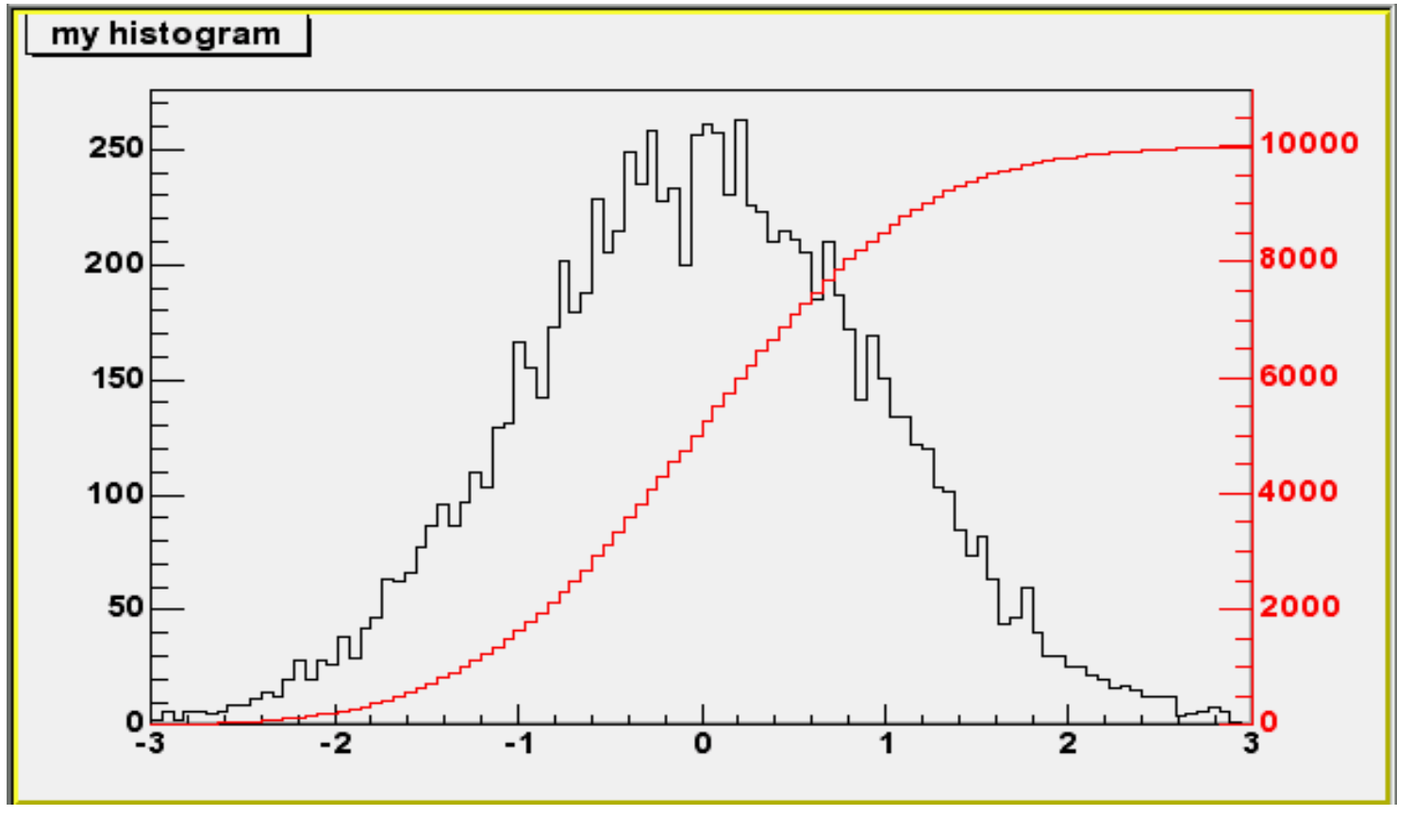
TH1::`Add(TF1* h1, Double_t c1 = 1, Option_t* option = "")`

TH1::`Add(const TH1* h1, const TH1* h2, Double_t c1 = 1, Double_t c2 = 1)` - add histograms weighted by `c1 and c2`.

Superimposing Histograms with Different Scales

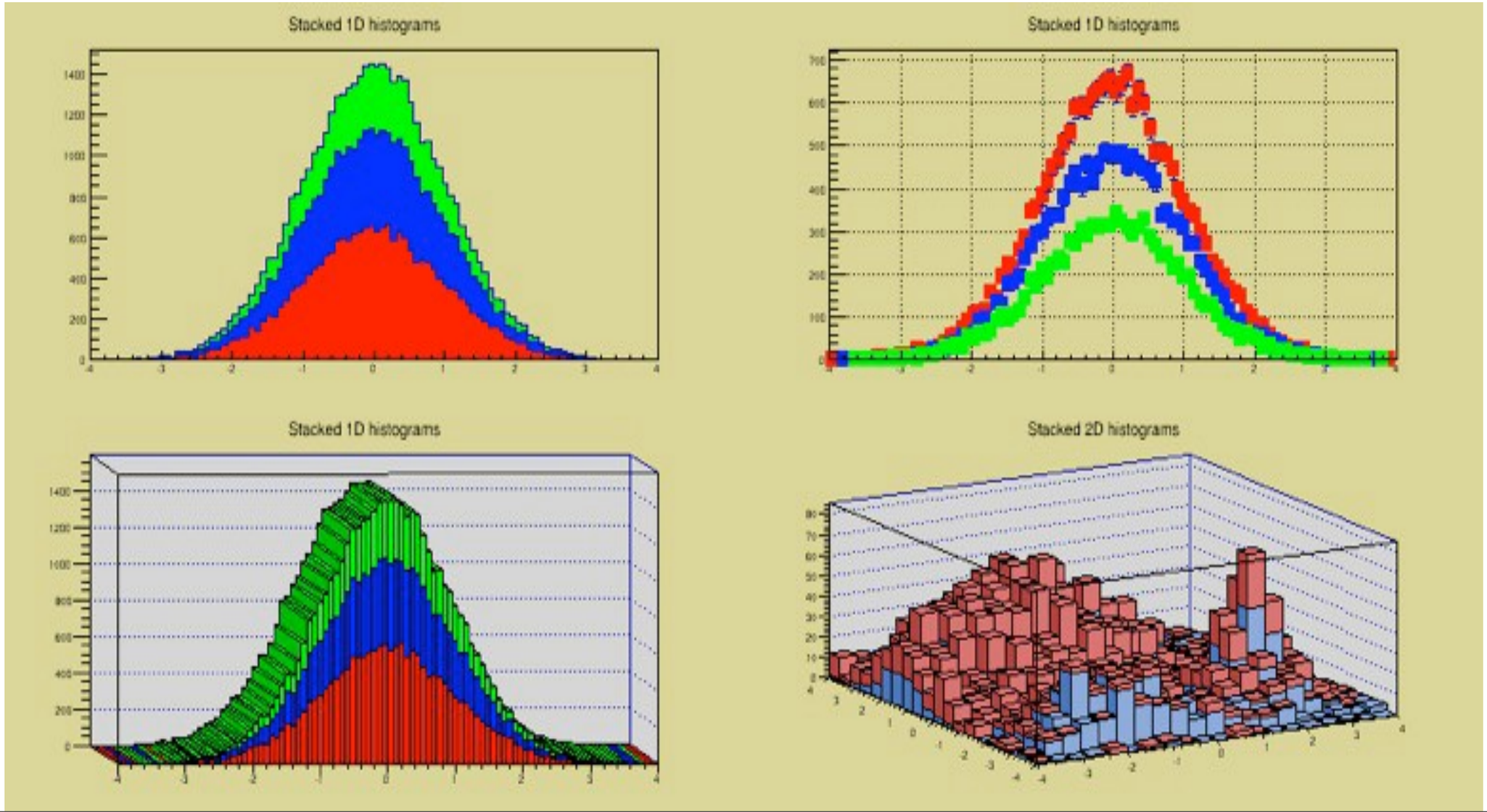
```
TH1F *h1 = new TH1F("h1","my histogram",100,-3,3);
... //（填高斯分布）
h1->Draw();
TH1F *hint1 = new TH1F("hint1",“my histogram",100,-3,3);
... //（填积分值）
//scale hint1 to the pad coordinates
Float_t rightmax = 1.1*hint1->GetMaximum();
Float_t scale = gPad->GetUymax()/rightmax;
hint1->SetLineColor(kRed);
hint1->Scale(scale);
hint1->Draw("same");
... //（draw an axis on the right side 看下面例子的代码在右边画一个轴）
```

example:
$ROOTSYS/tutorials/hist/twoscales.C



Histogram Stacks

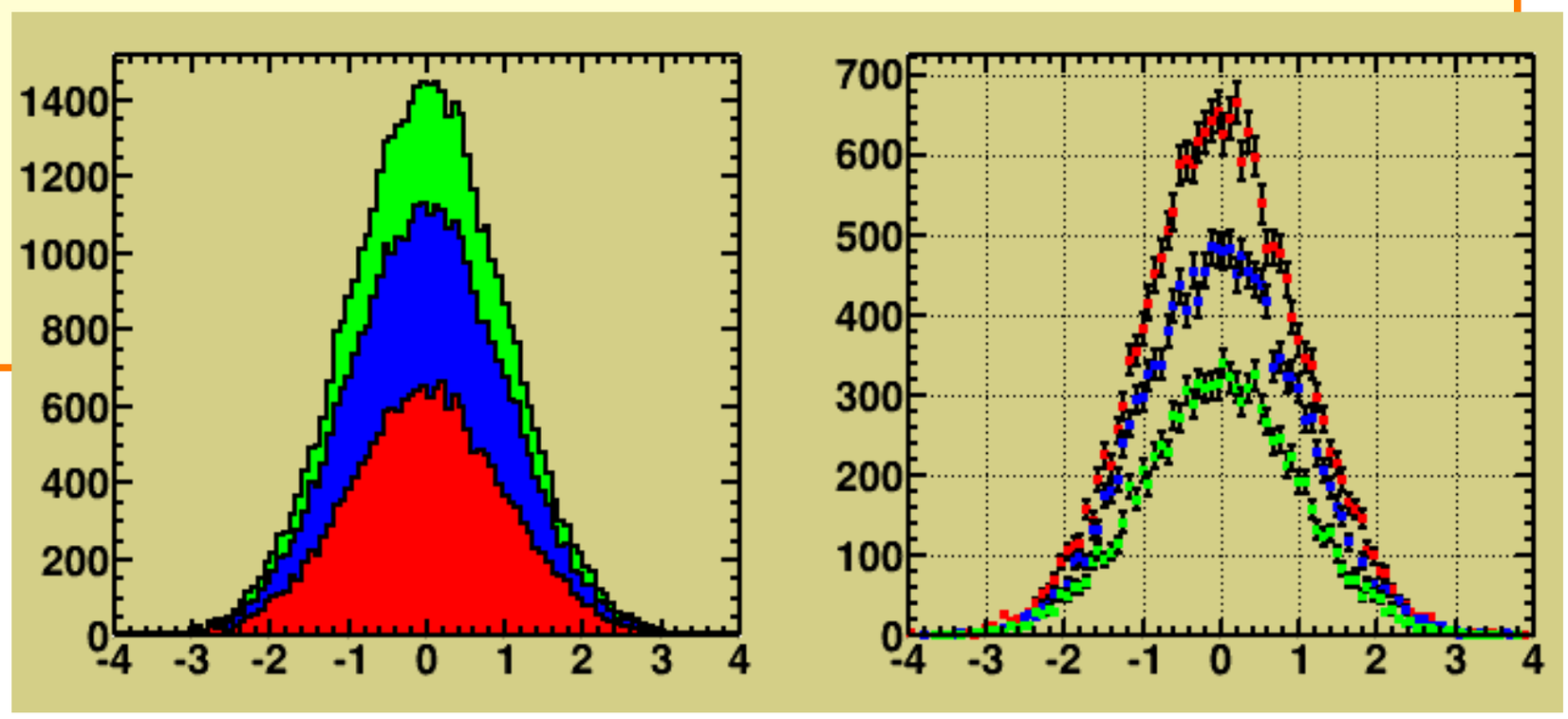
- `THStack`
  - Collection of 1D or 2D histograms.
  - Allow to draw several histograms in one go, on top of each other.
  - The frame is computed automatically.
  - Often better than doing several plots with option "SAME"



Histogram Stacks

A **THStack** is a collection of **TH1** (or derived) objects. Use **THStack**::Add(**TH1** *h) to add a histogram to the stack.

```
TCanvas c1("c1","stacked hists",10,10,700,900);
THStack hs("hs","test stacked histograms");
hs.Add(h1); \\red
hs.Add(h2); \\blue
hs.Add(h3); \\green
c1.Divide (1,2);
c1.cd(1);
hs.Draw();
c1.cd(2);
hs->Draw("nostack");
```



常用于实验数据中不同类型本底的叠加。

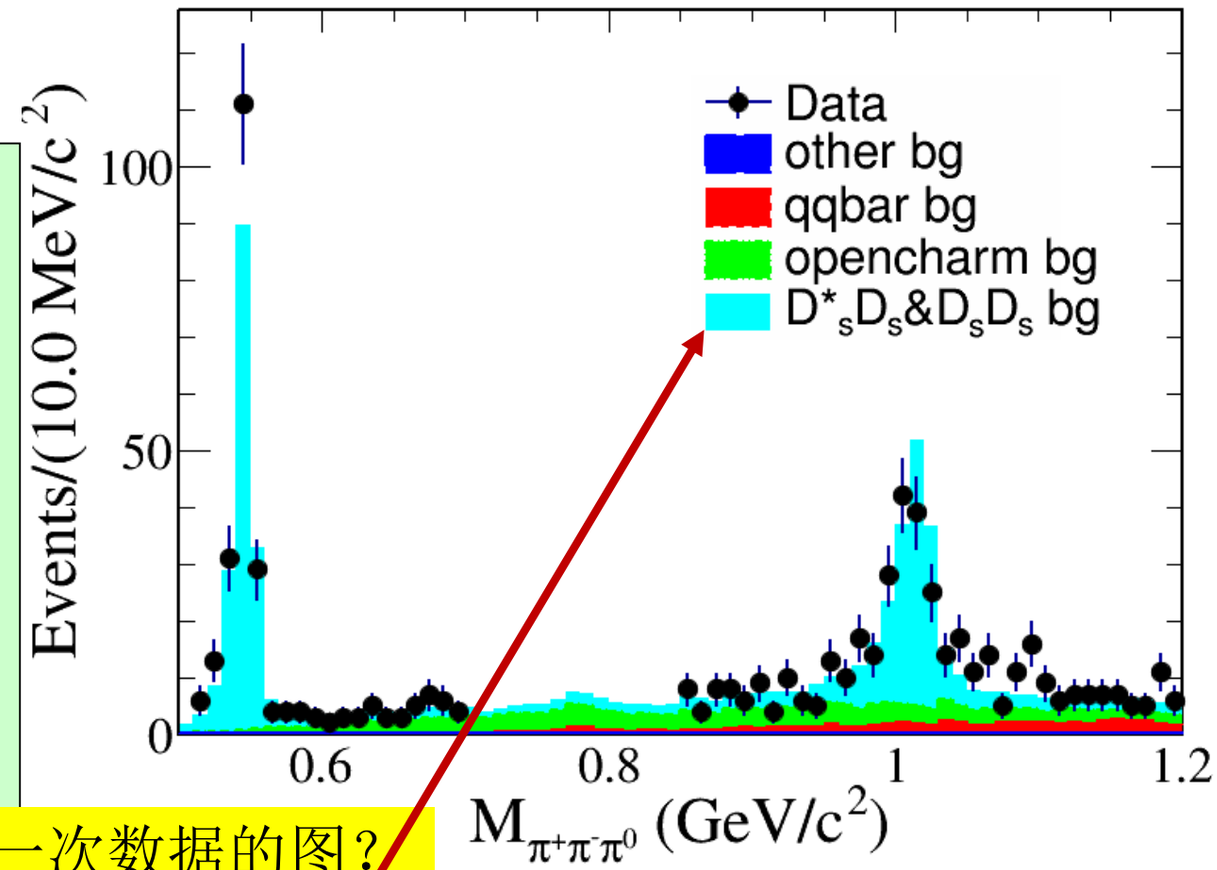
example: $ROOTSYS/tutorials/hist/hstack.C

Histogram Stacks：example

一个显示本底成分的例子：

```
Hist_data->Draw("E1");
Hist_others->SetFillColor(kBlue);
Hist_opencharm->SetFillColor(kGreen);
Hist_qqbar->SetFillColor(kRed);
Hist_DsSTDs_DsDs->SetFillColor(7);
THStack *hs = new THStack("hs","Stacked");
hs->Add(Hist_others);
hs->Add(Hist_qqbar);
hs->Add(Hist_opencharm);
hs->Add(Hist_DsSTDs_DsDs);
hs->Draw("same");
Hist_data->Draw("sameE1");
```

为何还要再画一次数据的图？




```
leg = new TLegend(0.55,0.60,0.85,0.9);
   leg->AddEntry(Hist_data,"Data","lep");
   leg->AddEntry(Hist_others,"other bg","f");
   leg->AddEntry(Hist_qqbar,"qqbar bg","f");
   leg->AddEntry(Hist_opencharm,"opencharm bg","f");
   leg->AddEntry(Hist_DsSTDs_DsDs,"D*_{s}D_{s}&D_{s}D_{s} bg","f");
   leg->SetTextSize(0.06);
   leg->SetLineColor(10);
   leg->SetFillColor(10);
   leg->Draw();
```

2D Histograms

- Frequency distribution of (X,Y) observations.
  - Construct 2D histogram specifying the number of bins in X axis, the minimum and maximum of axis range and the same for the Y axis

```
TH2D * h2 = new TH2D("h2","Example 2D Hist",40,-4.,4.,40,-4.,4);
```

  - fill 2D histogram with 10000 x,y normal data

```
for (int i = 0; i<10000; ++i) {
  double x = gRandom->Gaus(0,1);
  double y = gRandom->Gaus(1,2);
  h2->Fill(x,y);
}
```

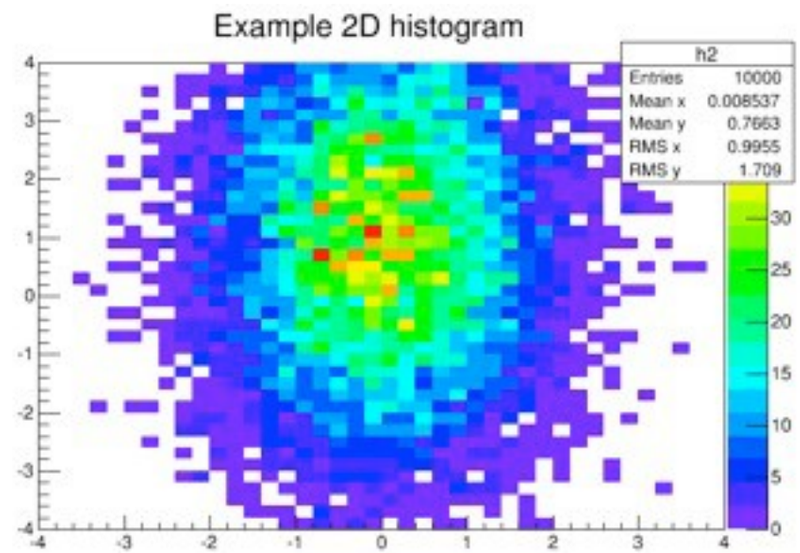
  - use `h2->Draw()` for drawing, several options are available
    - Color, Contour, lego, surface and box plots

Drawing 2D histograms

- Color plots:

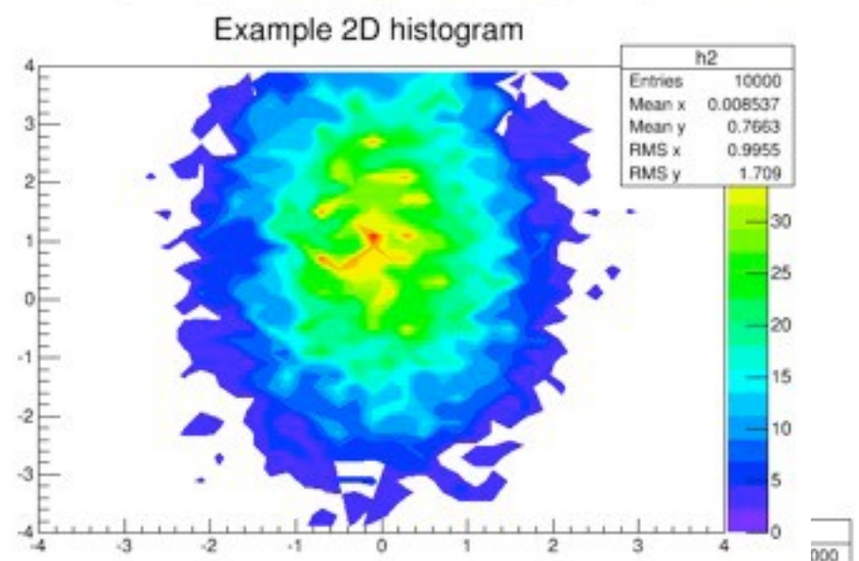
```
h2->Draw("COLZ");
```

"Z" means drawing the color palette for the bin content (i.e. the Z axis)



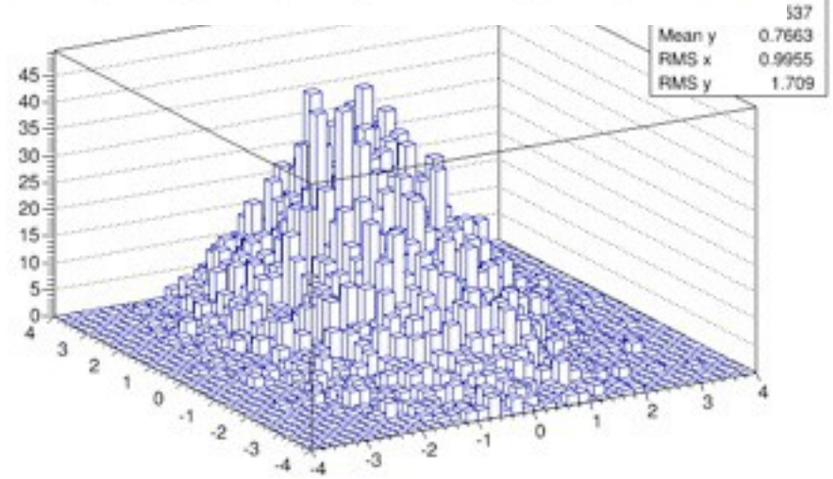
- Contour plots:

```
h2->Draw("CONTZ");
```



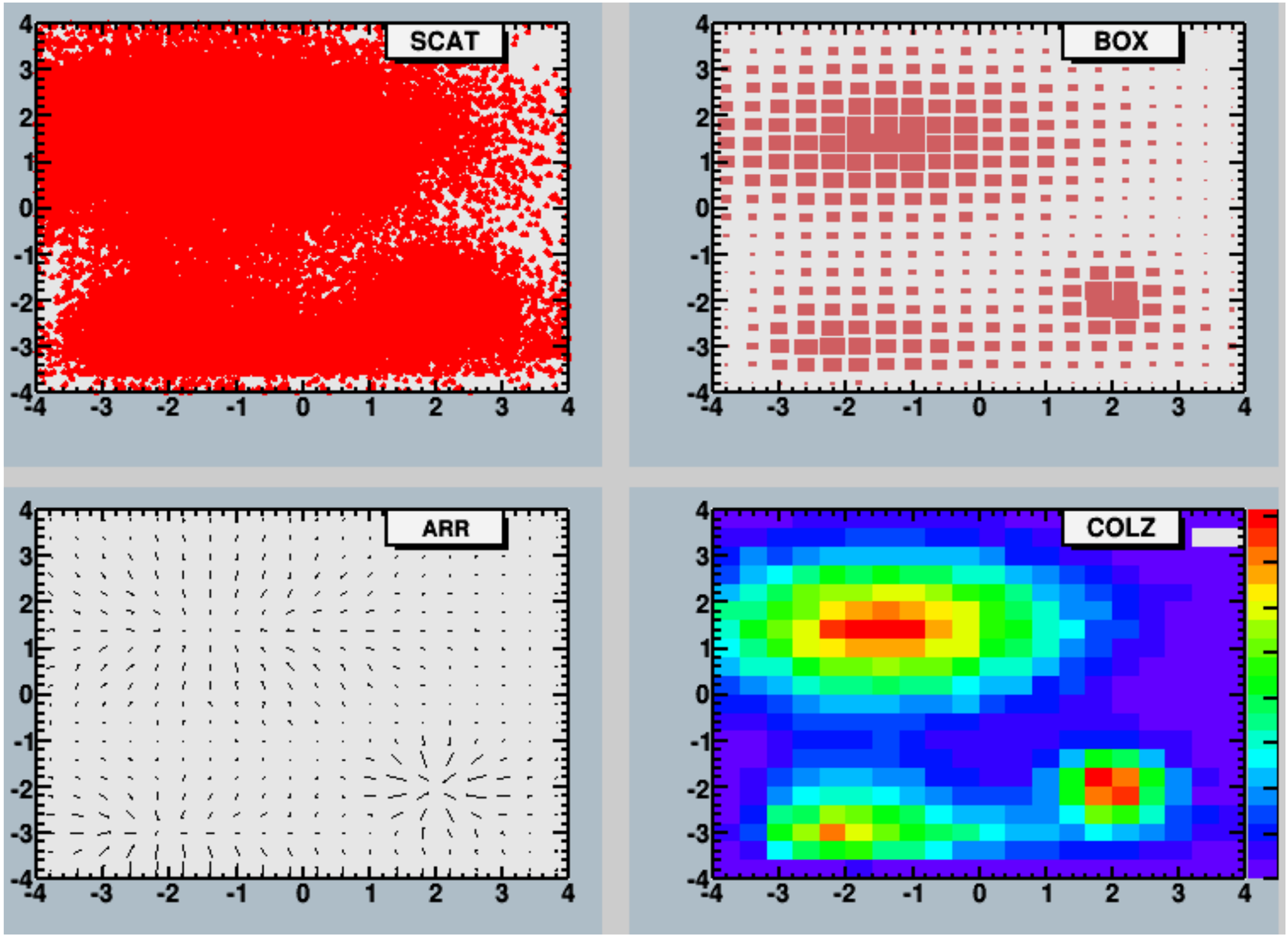
- Lego plots:

```
h2->Draw("LEGO");
```



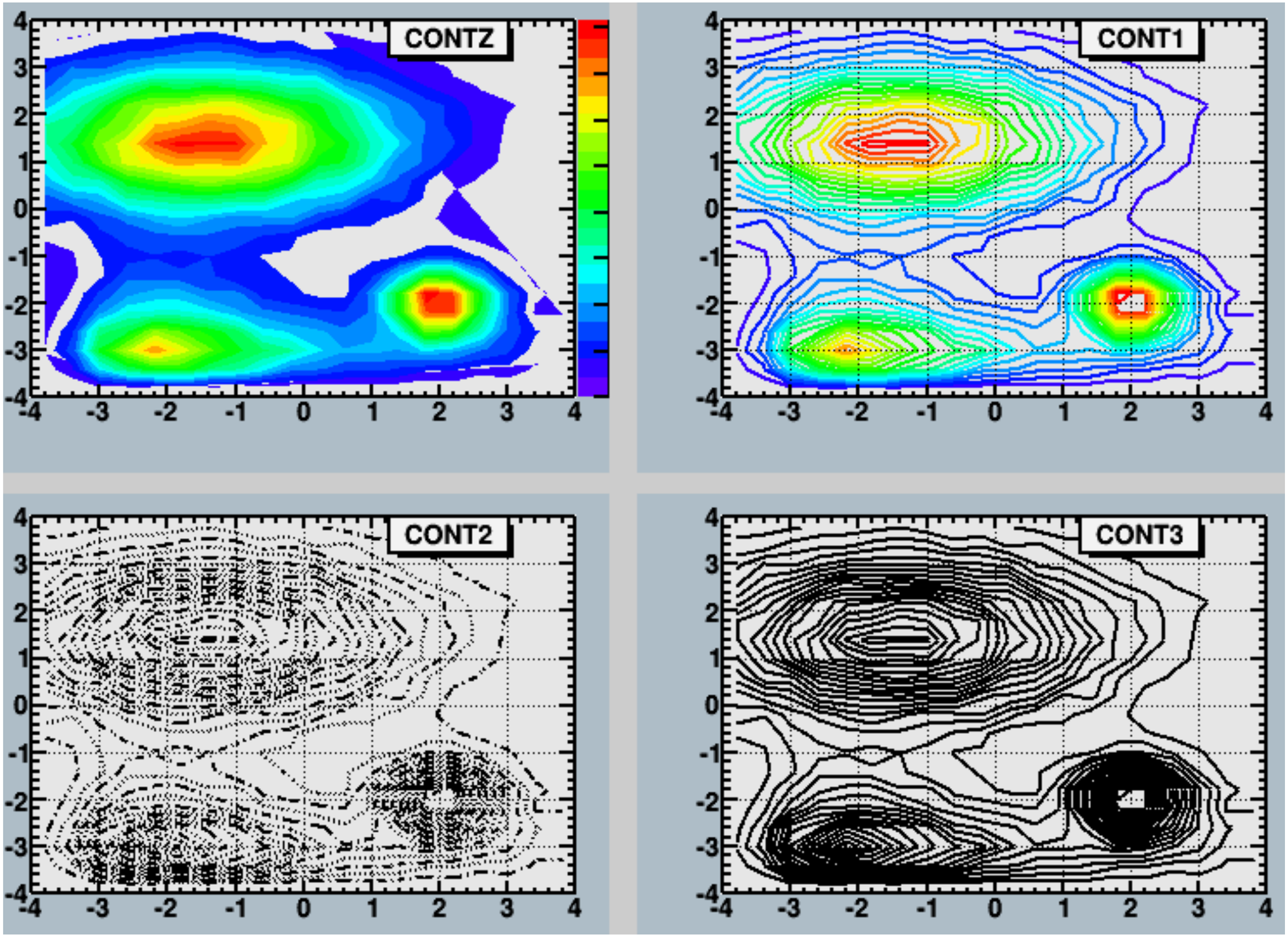
2-D drawing options (*hist/draw2dopt.C*)

root *hist/draw2dopt.C*



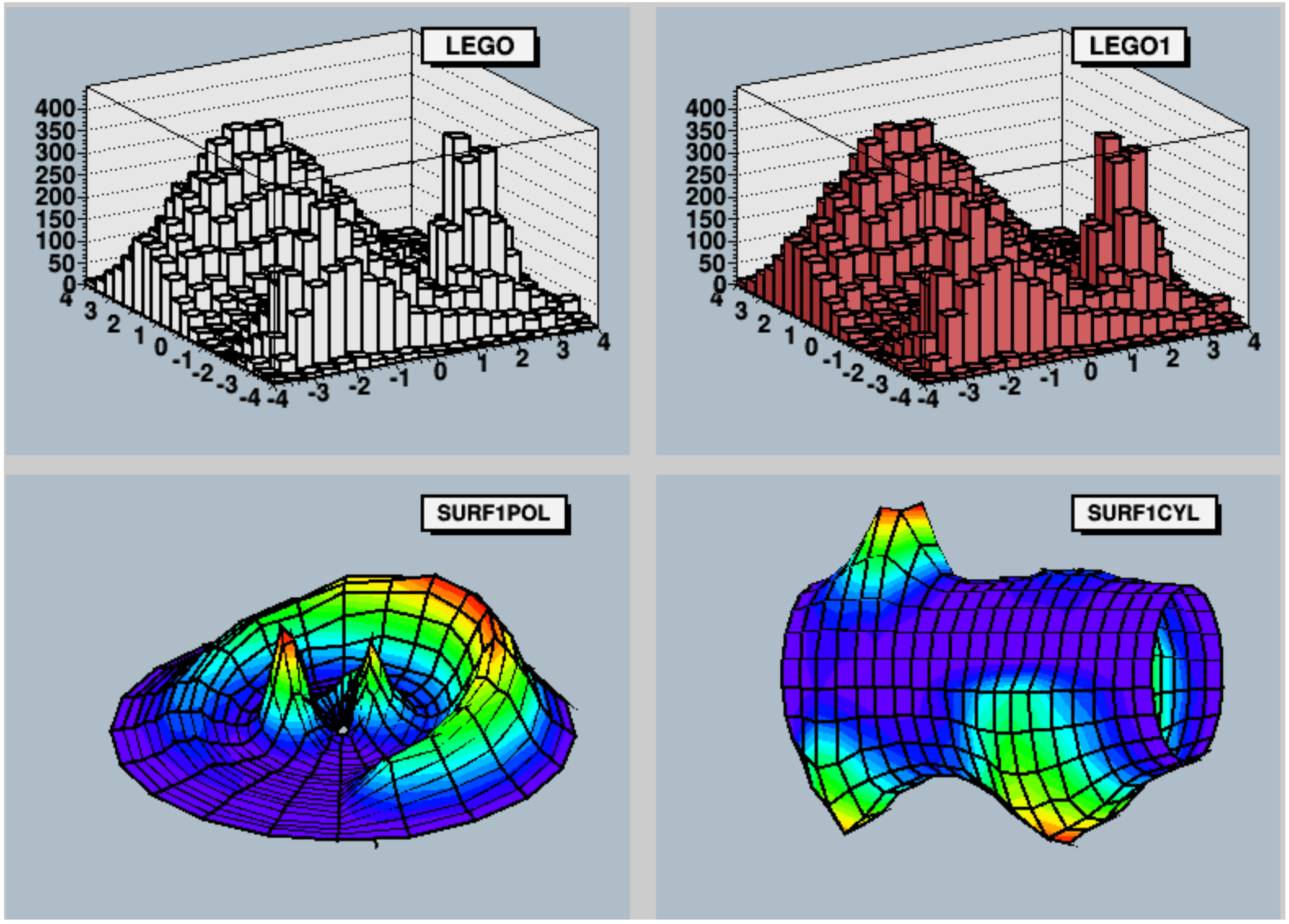
2-D drawing options (*hist/draw2dopt.C*) (2)

root *hist/draw2dopt.C*



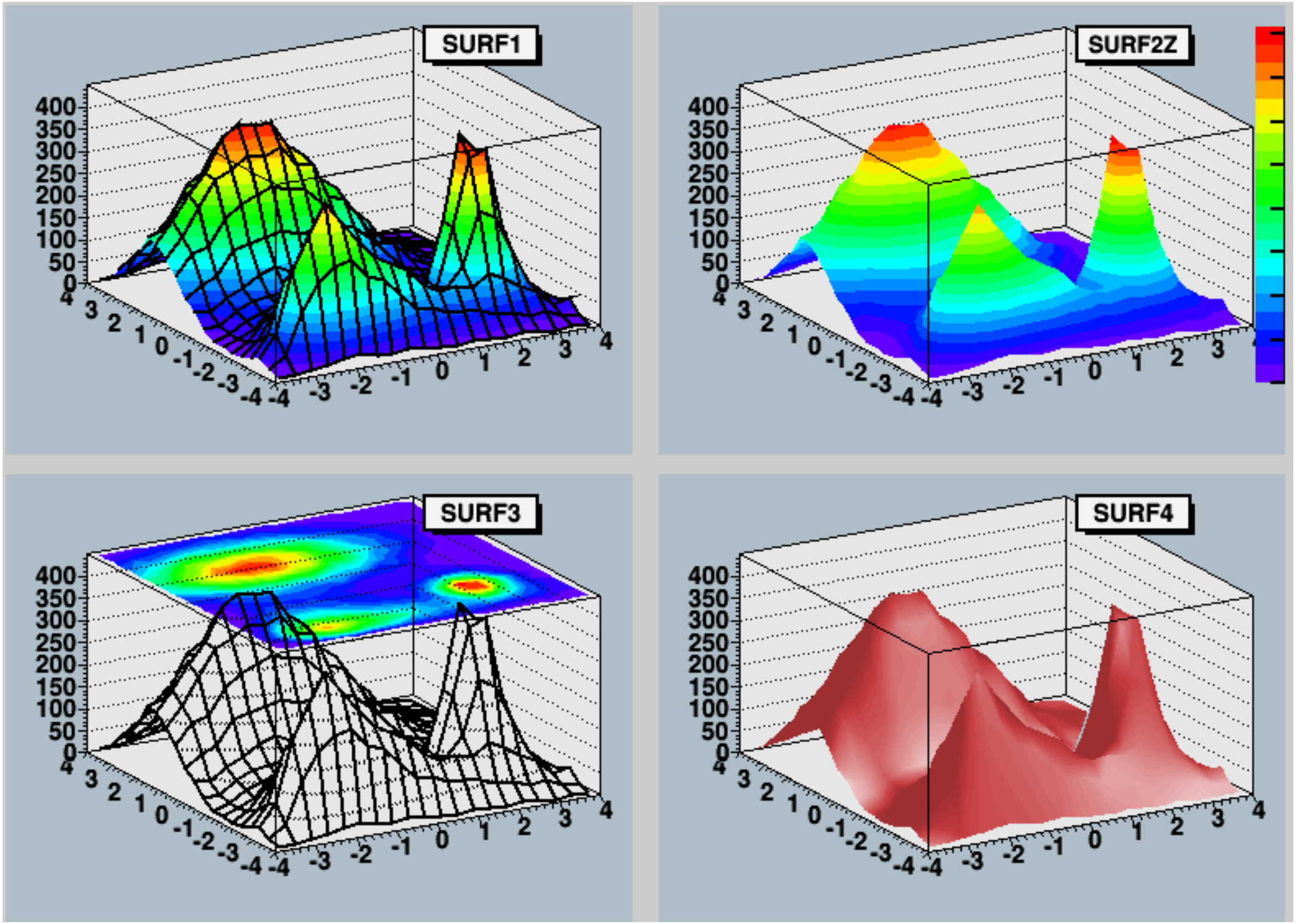
2-D drawing options (*hist/draw2dopt.C*) (3)

root *hist/draw2dopt.C*



2-D drawing options (*hist/draw2dopt.C*) (4)

root *hist/draw2dopt.C*

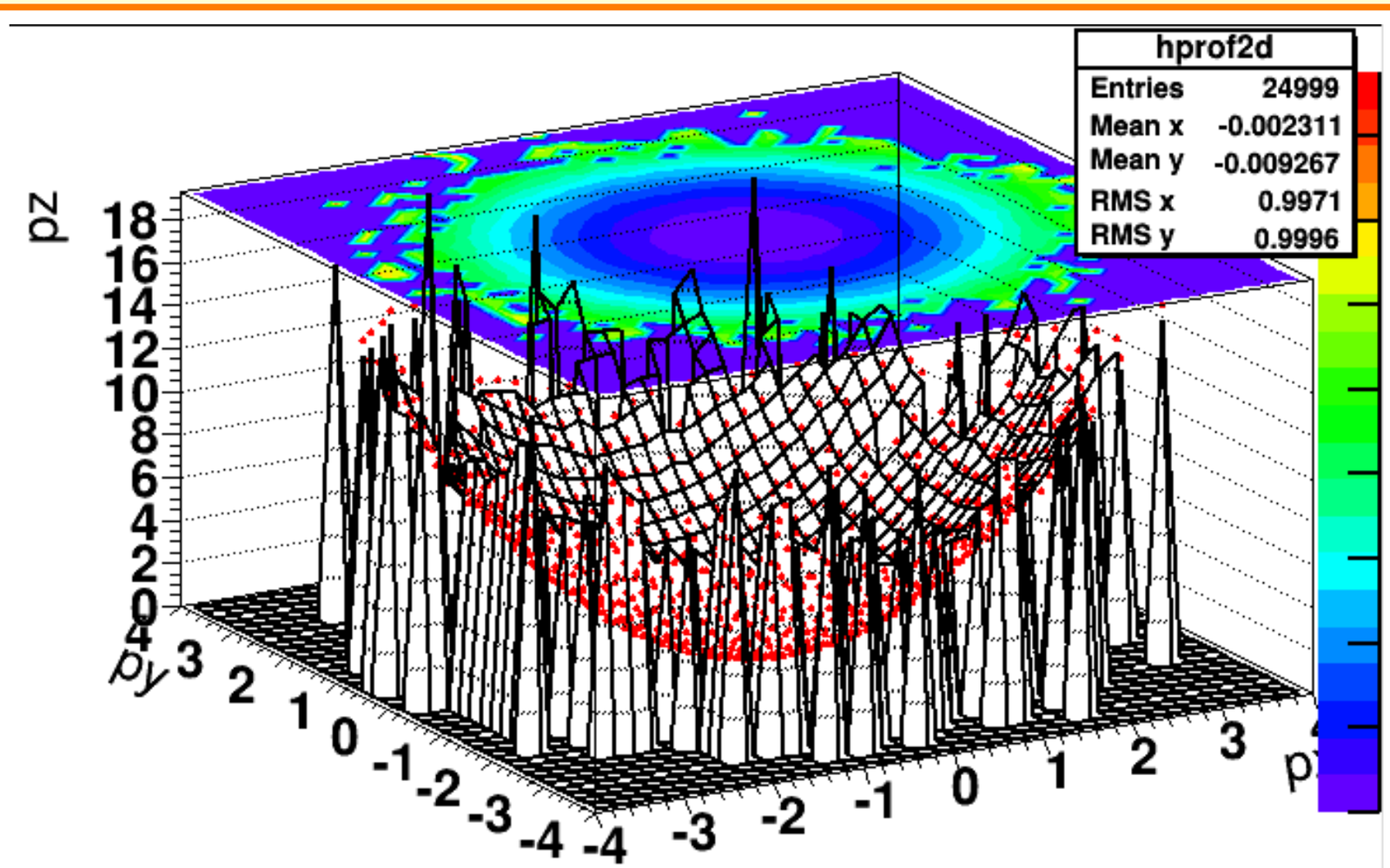


TProfile2D

root Profile2D.C

```
Profile2D(){
 gStyle->SetPalette(1);
 TCanvas *c1 = new TCanvas("c1","Profile histogram example",200,10,700,500);
 hprof2d = new TProfile2D("hprof2d","TProfile2D;px;py;pz",40,-4,4,40,-4,4,0,20);
 Float_t px, py, pz;
 for ( Int_t i=0; i<25000; i++) {
   gRandom->Rannor(px,py);
   pz = px*px + py*py;
   hprof2d->Fill(px,py,pz,1);
 }
 hprof2d->Draw("SURF3ZE");
}
```

Return 2 numbers distributed following a gaussian with mean=0 and sigma=1



Weighted Histograms

- Histogram can be filled with a "weight"
  - observations do not contribute equally, but some of them contribute more or less than others;
  - the weight expresses how much an observation contributes.
    - Filling a 1D histogram with observation x and weight w:

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

    - Filling a 2D histogram with observation x,y and weight w:

```
h1->Fill(x,y,w);
```

  - A weighted histogram will have and display as:
    - bin content = sum of all the weights accumulated in the bin;
    - bin error = $\sqrt{W2}$: W2 = sum of the weight square in the bin.

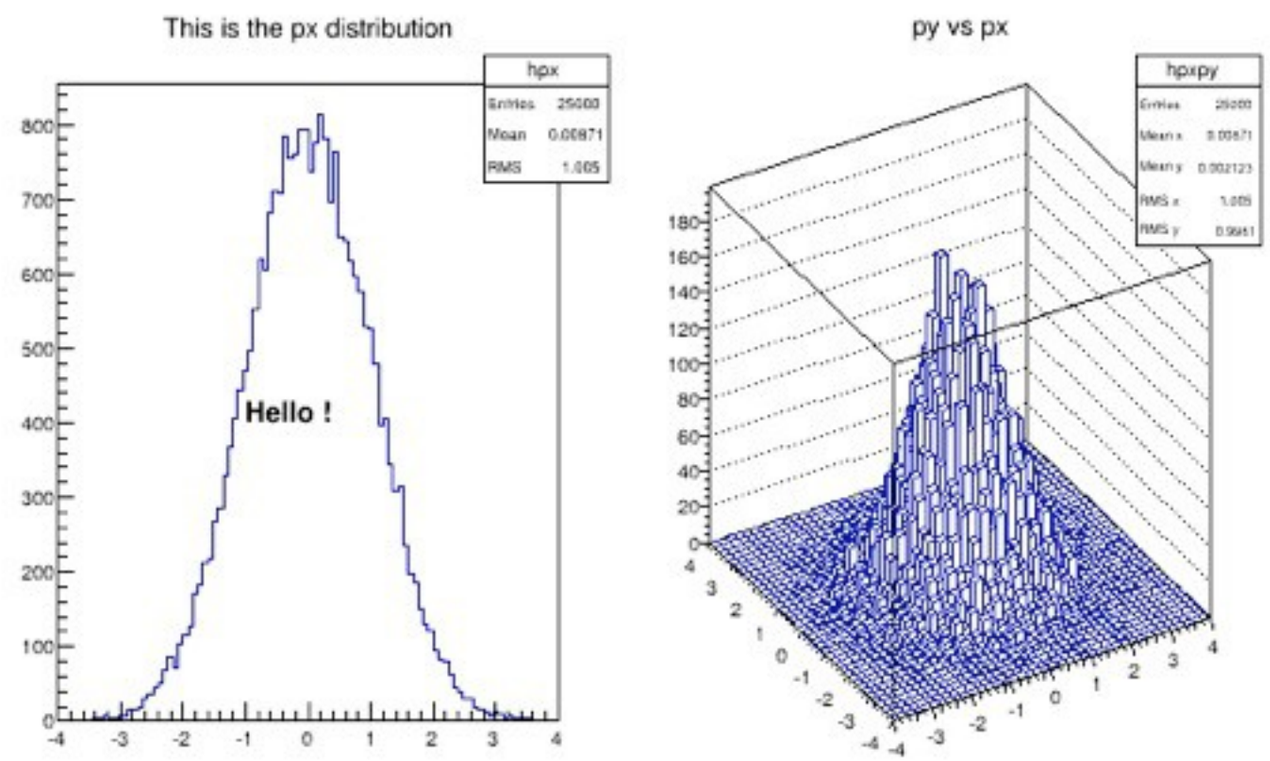
(in ROOT versions <= 5.34, if one has called TH1::Sumw2() before filling the histogram)

The Graphics Canvas

A canvas is the graphics window in which the ROOT graphics will be displayed. It can be created using the class `TCanvas`. One can create as many canvases as needed during a ROOT session.

A `TCanvas` usually contains at least one `TPad`. Most of the time it contains several, each of them having its own coordinate system. A simple way to quickly make several pads in a canvas is to use the method `Divide()` like in the following example:

```
TCanvas *c = new TCanvas("c","my canvas",
600,400);
c->Divide(2,1);
c->cd(1);
hpx->Draw();
c->cd(2);
hpxpy->Draw("lego");
c->cd(1);
TText *T = new TText(-1.,400.,"Hello !");
T->Draw();
```



Visualization Techniques in ROOT

How ROOT visualize
2, 3, 4 and N data variables

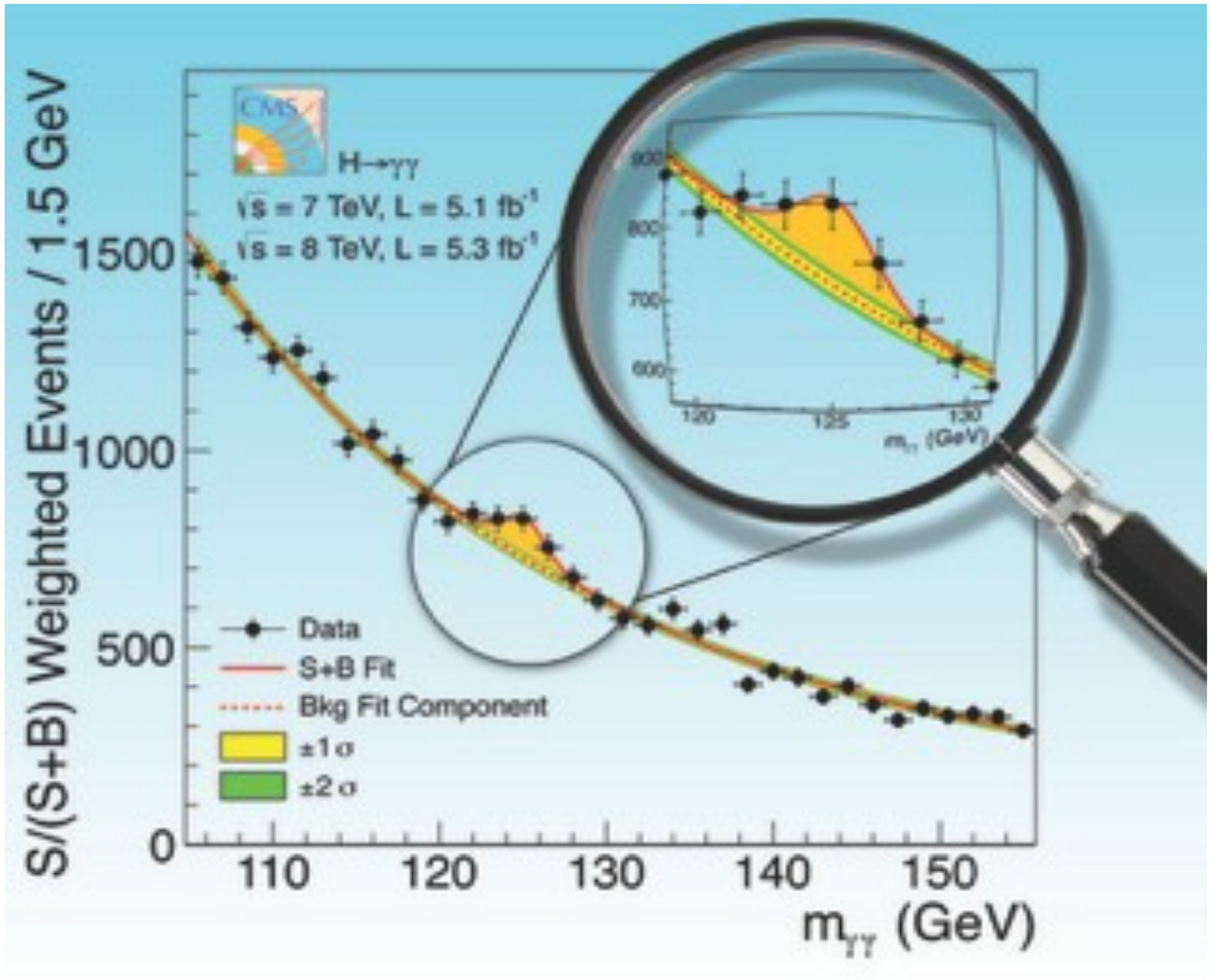
Dig $ROOTSYS/tutorials/hist

- Run the example macros
- Try to understand the codes of the example macros:

  references: *http://root.cern.ch/drupal/content/reference-guide*

  class header descriptions
- "*grep*" those macros to find what you need

Fitting

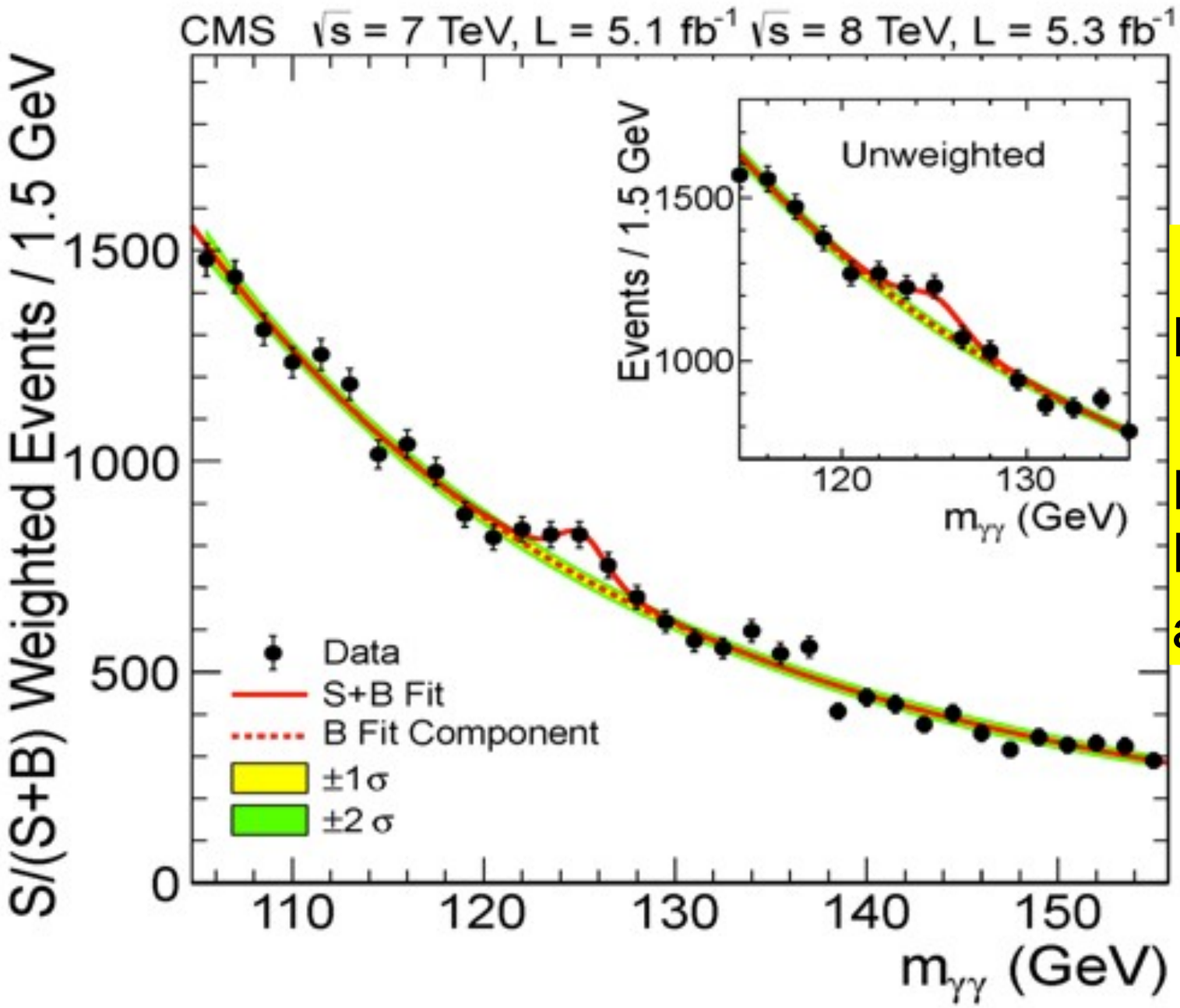

CMS
H→γγ
√s = 7 TeV, L = 5.1 fb⁻¹
√s = 8 TeV, L = 5.3 fb⁻¹
S/(S+B) Weighted Events / 1.5 GeV
1500
1000
500
0
110
120
130
140
150
m_γγ (GeV)
Data
S+B Fit
Bkg Fit Component
±1σ
±2σ

Outline

- Introduction to Fitting:
  - what is fitting,
  - how to fit a histogram in ROOT,
  - how to retrieve the fit result.
- Building complex fit functions in ROOT.
- Interface to Minimization.
- Common Fitting problems.
- Using the ROOT Fit GUI (Fit Panel).
- Random number generations in ROOT.
- How to generate random numbers from distributions.

What is Fitting ?

- **What is Fitting ?**
  - It is the process used to estimate parameters of an hypothetical distribution from the observed data distribution



Example

Higgs search in CMS

(H ➡ γγ)

Fit for the expected number of Higgs events

and for the Higgs mass

What is Fitting (2)

- A histogram (or a graph) represents an estimate of an underlying distribution (or a function).
- The histogram or the graph can be used to infer the parameters describing the underlying distribution.
- Assume a relation between the observed variables y and x:

  $y = f(x, \theta)$

  - $f(x, \theta)$ is the fit (model) function, θ 为待拟合的参数
  - for an histogram y is the bin content

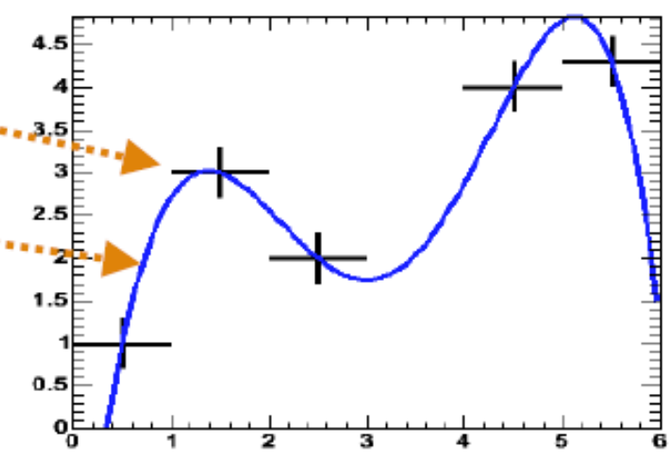
- One typically minimizes the deviations between the observed y and the predicted function values:
  - Least square fit (χ^2) :
    - minimize square deviation
    - weighted by the observed errors

      $\sigma = \sqrt{N}$ for the histograms

$$\chi^2 = \sum_i \frac{(Y_i - f(X_i, \theta))^2}{\sigma_i^2}$$

A well known estimator – the χ^2 fit

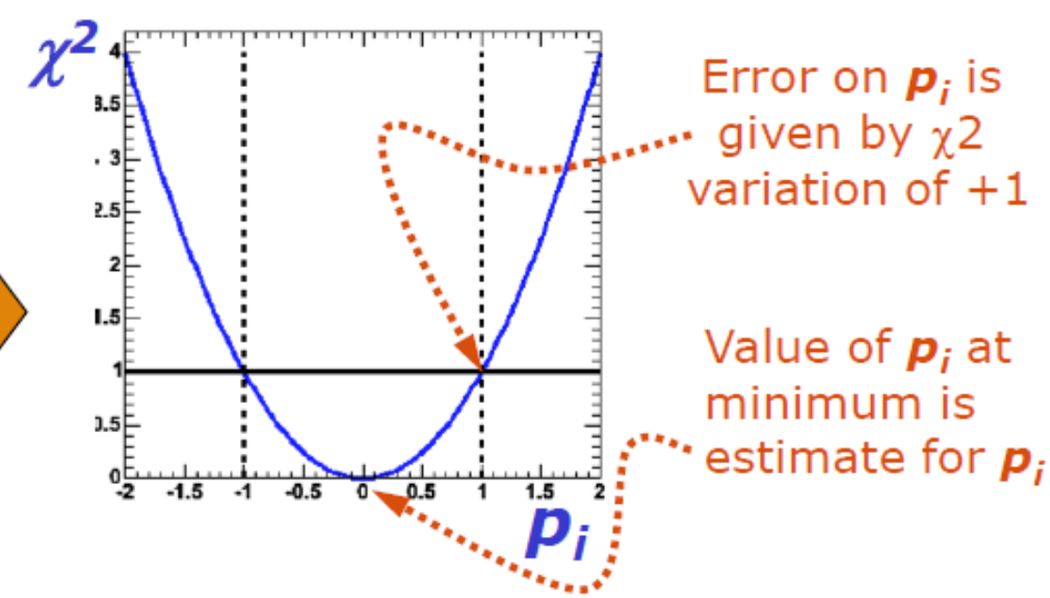
- Given a set of points $\{(\vec{x}_i, y_i, \sigma_i)\}$ and a function ***f(x,p)*** define the χ^2



$$\chi^2(\vec{p}) = \sum_i \frac{(y_i - f(\vec{x};\vec{p}))^2}{\sigma_y^2}$$

- Estimate parameters by minimizing the $\chi^2(\boldsymbol{p})$ with respect to all parameters p_i
  - *In practice, look for*

$$\frac{d\chi^2(p_i)}{dp_i} = 0$$



- Well known: but why does it work? Is it always right? Does it always give the best possible error?

ML Fit of an Histogram

- **Maximum Likelihood (ML) Fit**:
  - The parameters are estimated by finding the maximum of the likelihood function (or minimum of the negative log-likelihood function).
  - Likelihood: $$L(x|\theta) = \prod_i P(x_i|\theta)$$
- The Likelihood for a histogram is obtained by assuming a Poisson distribution in every bin:
  - `Poisson(` n_{obs} `|` n_{exp} `)`
    - n_{obs} is the observed bin content.
    - n_{exp} is the expected bin content, which can be obtained from the fit model function (the underlying distribution of the histogram)
      - $n_{exp} = f(x_c|\theta)$, where x_c is the bin center, assuming a linear function within the bin. Otherwise it is obtained from the integral of the function in the bin.
- The least-square fit and the maximum likelihood fit are equivalent when the distribution of observed events in each bin is normal.
  - This is true only for large histogram statistics (large bin contents).
- **For low histogram statistics the ML method is the correct one !**

The Likelihood estimator

- Definition of Likelihood
  - given **D(x)** and **F(x;p)**

Functions used in likelihoods must be Probability Density Functions:

$$\int F(\vec{x};\vec{p})d\vec{x} \equiv 1, \quad F(\vec{x};\vec{p}) > 0$$

$$L(\vec{p}) = \prod_i F(\vec{x}_i;\vec{p}), \quad \text{i.e.} \quad L(\vec{p}) = F(x_0;\vec{p}) \cdot F(x_1;\vec{p}) \cdot F(x_2;\vec{p}) \ldots$$

  - For convenience the *negative log* of the Likelihood is often used

$$-\ln L(\vec{p}) = -\sum_i \ln F(\vec{x}_i;\vec{p})$$

- Parameters are estimated by minimizing –log(L): $\left.\frac{d \ln L(\vec{p})}{d\vec{p}}\right|_{p_i=\hat{p}_i} = 0$

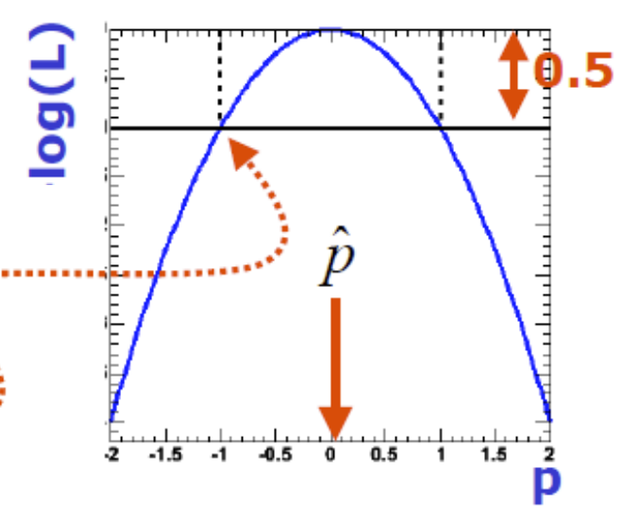
- Estimator for the parameter variance is $\hat{\sigma}(p)^2 = \hat{V}(p) = \left(\frac{d^2 \ln L}{d^2 p}\right)^{-1}$

- Visual interpretation of variance estimate
  - Taylor expand –log(L) around minimum

$$\ln L(p) = \ln L(\hat{p}) + \left.\frac{d \ln L}{dp}\right|_{p=\hat{p}} (p-\hat{p}) + \tfrac{1}{2}\left.\frac{d^2 \ln L}{d^2 p}\right|_{p=\hat{p}} (p-\hat{p})^2$$

$$= \ln L_{\max} + \left.\frac{d^2 \ln L}{d^2 p}\right|_{p=\hat{p}} \frac{(p-\hat{p})^2}{2}$$

$$= \ln L_{\max} + \frac{(p-\hat{p})^2}{2\hat{\sigma}_p^2} \quad \Rightarrow \quad \ln L(p \pm \sigma) = \ln L_{\max} - \frac{1}{2}$$



ML or χ^2 – What should you use?

- χ^2 fit is fastest, easiest
  - Works fine at high statistics
  - Gives absolute goodness-of-fit indication
  - Make (incorrect) Gaussian error assumption on low statistics bins
  - Has bias proportional to 1/N
  - Misses information with feature size < bin size

- Full Maximum Likelihood estimators most robust
  - No Gaussian assumption made at low statistics
  - No information lost due to binning
  - Gives best error of all methods (especially at low statistics)
  - No intrinsic goodness-of-fit measure, i.e. no way to tell if 'best' is actually 'pretty bad'
  - Has bias proportional to 1/N
  - Can be computationally expensive for large N

- Binned Maximum Likelihood in between

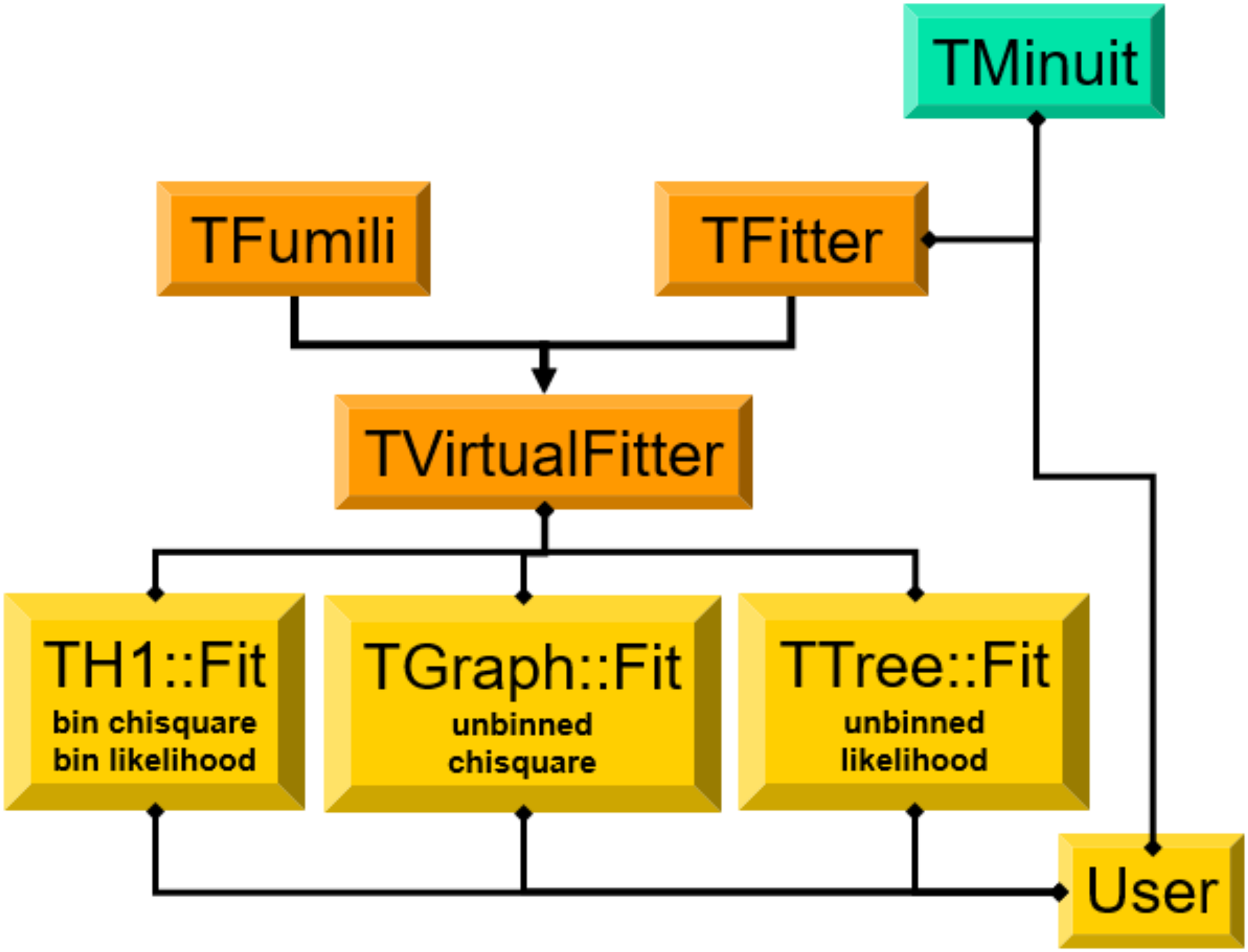
$$-\ln L(p)_{\text{binned}} = \sum_{\text{bins}} n_{\text{bin}} \ln F(\vec{x}_{\text{bin-center}}; \vec{p})$$

  - Much faster than full Maximum Likihood
  - Correct Poisson treatment of low statistics bins
  - Misses information with feature size < bin size
  - Has bias proportional to 1/N

Fitting in ROOT

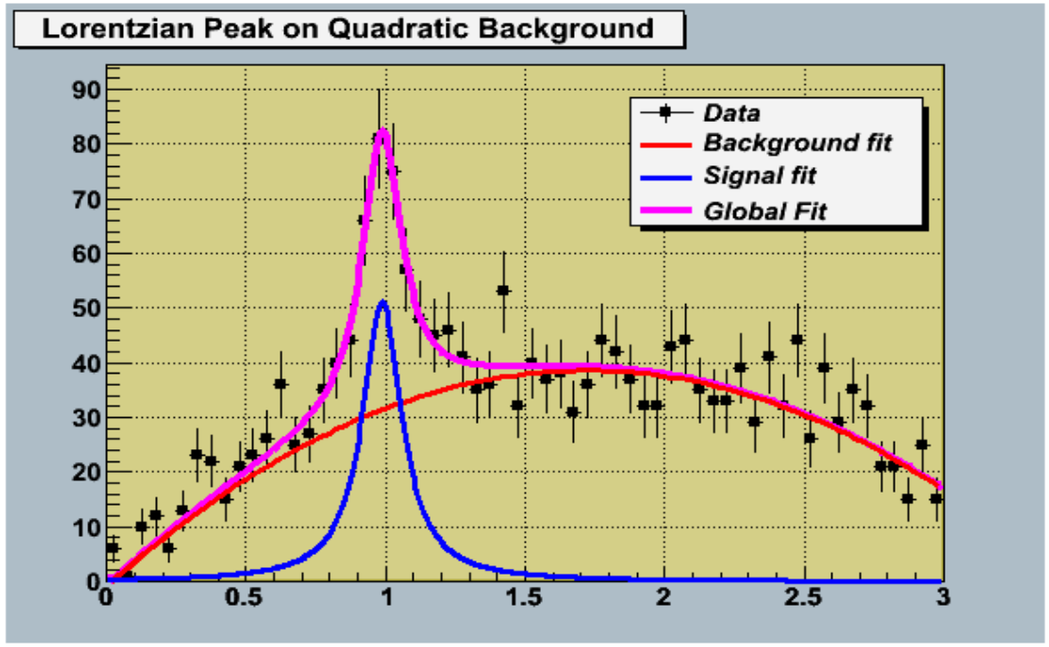
- **How do we do fit in ROOT:**
  - Create first a parametric function object, `TF1`, which represents our model, *i.e.* the fit function.
  - Set the initial values of the function parameters.
  - Fit the data object (Histogram or Graph):
    - call the `Fit` method on the Histogram or Graphs passing the function object as parameter
      - various options are possibles (see the `TH1::Fit` documentation)
        - » e.g select type of fit : least-square (default) or likelihood (option "L")
      - the resulting fit function is then drawn on top of the Histogram or the Graph.
  - **Examine result:**
    - get parameter values;
    - get parameter errors (e.g. their confidence level);
    - get parameter correlation;
    - get fit quality.

Fitting


TMinuit
TFumili
TFitter
TVirtualFitter
TH1::Fit
bin chisquare
bin likelihood
TGraph::Fit
unbinned
chisquare
TTree::Fit
unbinned
likelihood
User

Fitting - Interface

- **Minimization packages: Minuit and Fumili**
- **Fitting can be done through the general interface of**
  - **TH1::Fit (binned data)**
    Chisquare and Loglikelihood methods
  - **TGraph::Fit**
    unbinned data
  - **TGraphErrors::Fit**
    data with errors
  - **TGraphAsymmErrors::Fit**
    taking into account asymmetry of errors
  - **TTree::Fit and TTree::UnbinnedFit**



- **RooFit** for object-oriented data modeling (下节课介绍).
  ---- Distributed with ROOT starting from version 5.02-00

the Fit Method

- Use the **TH1::Fit** method

```
void Fit(const char *fname, Option_t *option, Option_t *goption,
 Axis_t xxmin, Axis_t xxmax)
```

- *fname - the name of the fitted function. This name may be one of ROOT pre-defined function names or a user-defined function. Predefined functions:
  - ✓ **gaus**: Gaussian function with 3 parameters: p0*exp(-0.5*((x-p1)/p2)^2))
  - ✓ **expo**: an Exponential with 2 parameters:exp(p0+p1*x)
  - ✓ **polN**: a polynomial of degree N:p0 + p1*x + p2*x2 +...
  - ✓ **landau**: Landau function with mean and sigma.
- *option - fitting option
- *goption - graphics option which is the same as in the **TH1::Draw()**
- xxmin, xxmax - specify the range over which to apply the fit.

the Fitting Option

```
"W"   Set all weights to 1 for non empty bins; ignore error bars
"WW"  Set all weights to 1 including empty bins; ignore error bars
"I"   Use integral of function in bin instead of value at bin center
"L"   Use log likelihood method (default is chi-square method)
"U"   Use a user specified fitting algorithm
"Q"   Quiet mode (minimum printing)
"V"   Verbose mode (default is between Q and V)
"E"   Perform better errors estimation using the Minos technique
"M"   Improve fit results
"R"   Use the range specified in the function range
"N"   Do not store the graphics function, do not draw
"0"   Do not plot the result of the fit. By default the fitted function
      is drawn unless the option "N" above is specified.
"+"   Add the new fitted function to the list of fitted functions (by
      default, the previous function is deleted and only the last one is kept)
"B"   Use this option when you want to fix one or more parameters and the
      fitting function is like polN, expo, landau, gaus.
"LL"  An improved Log Likelihood fit in case of very low statistics and when
      bin contents are not integers.
"C"   In case of linear fitting, don't calculate the chisquare (saves time).
"F"   If fitting a polN, switch to Minuit fitter (by default, polN functions
      are fitted by the linear fitter).
```

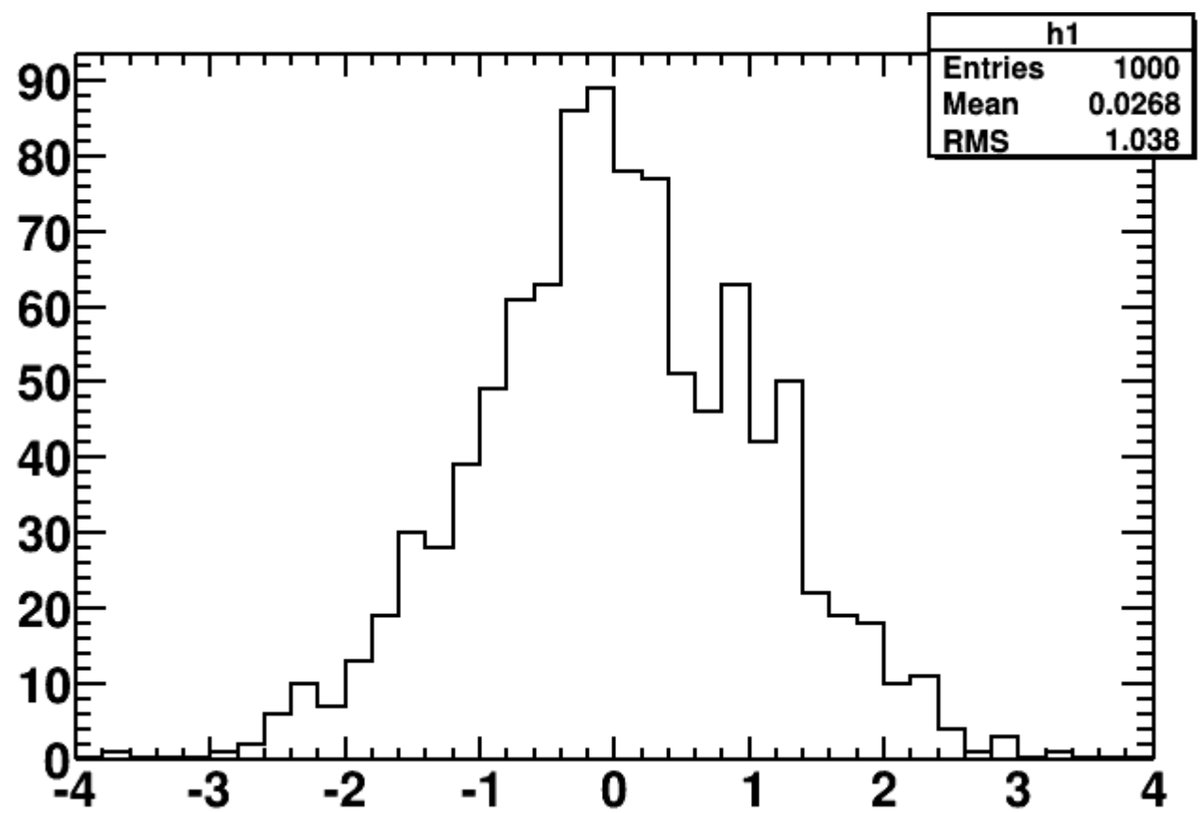
- Empty bins are excluded in the fit when using the Chi-square fit method. When fitting the histogram with the low statistics, it is recommended to use the **Log-Likelihood** method (**RooFit** is recommended!!!).

Simple Gaussian Fitting

- Recalling our previous histogram:
  - suppose we do not know how it was generated;
  - we want to estimate the mean and sigma of the underlying gaussian distribution.

root draw.C

```
#include "TH1.h"
#include "TF1.h"
#include "TRandom3.h"
void p(){
 TH1D * h1 = new TH1D("h1","Example",40,-4.,4.);
  for (int i = 0; i < 1000; ++i) {
   double x = gRandom->Gaus(0,1);
   h1->Fill(x);
 }
 h1->Draw();
}
```



Creating the Fit Function

- To create a parametric function object (a `TF1`) :
  - we can use the available functions in ROOT library

```
TF1 * f1 = new TF1("f1","[0]*TMath::Gaus(x,[1],[2])");
```

  - or we can use pre-defined functions defined in `TFormula` (see `TFormula` documentation for the list of them):

```
TF1 * f1 = new TF1("f1","gaus");
```

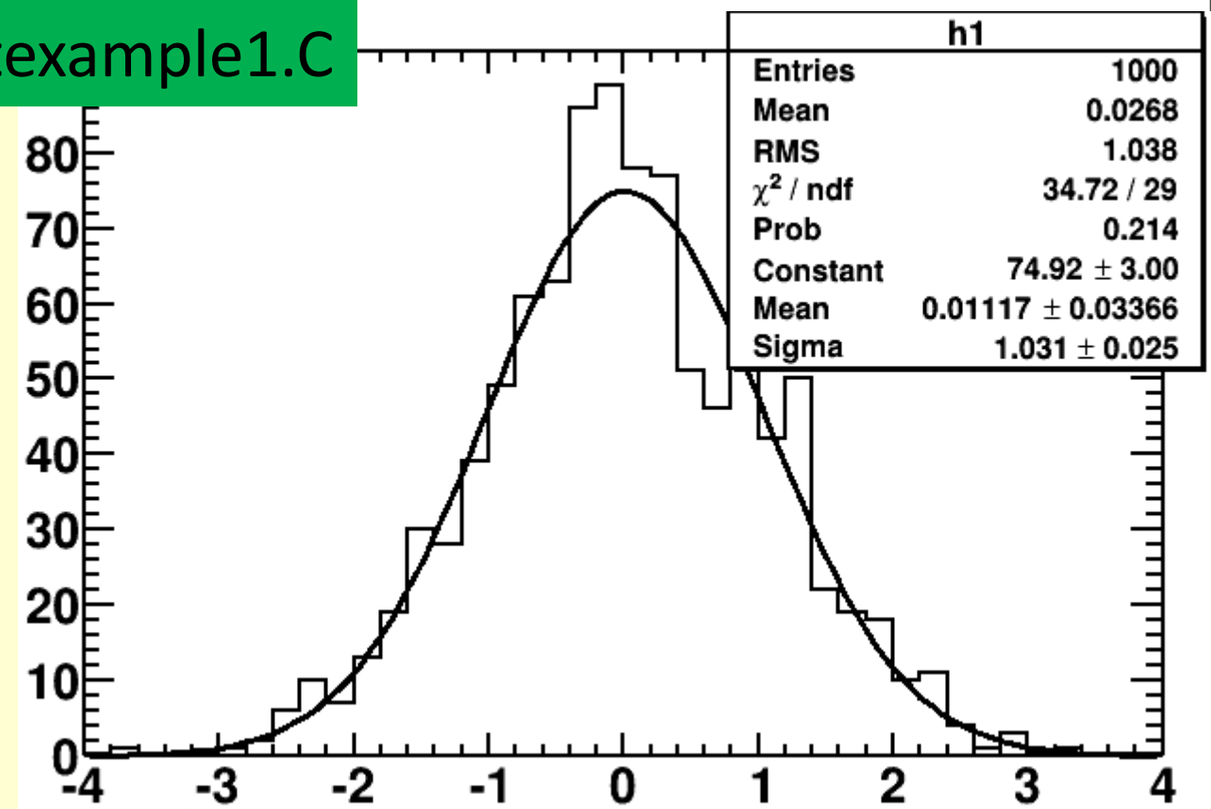
    - using pre-defined functions we have the parameter name automatically set to meaningful values.
    - initial parameter values are estimated whenever possible.
  - We will see later in general how to build a more complex function objects
    - e.g. by using other functions

Fitting Histogram

- How to fit the histogram (use a predefined function):
  - after creating the function one needs to set the initial value of the parameters
  - then we can call the `Fit` method of the histogram class

```
#include "TMath.h"
#include "TH1.h"
#include "TF1.h"
#include "TRandom3.h"
#include "TStyle.h"

void fitexample1() {
  TH1D * h1 = new TH1D("h1","Example",40,-4.,4.);
  for (int i = 0; i < 1000; ++i) {
    double x = gRandom->Gaus(0,1);
    h1->Fill(x);
  }

  TF1 *func= new TF1("func","gaus");
  func->SetParameters(1,0,1);
  // set the parameters to 1, 0, 1
  gStyle->SetOptFit(1111);
  h1->Fit("func");
}
```

运行：
root fitexample1.C



For displaying the fit parameters:

```
gStyle->SetOptFit(1111);
```

可用下面的命令编译，以检查错误：
root [1] .L fitexample1.C+

$ROOTSYS/tutorials/fit/myfit.C

Fitting Histogram (2)

- How to fit the histogram (use a user-defined Function):

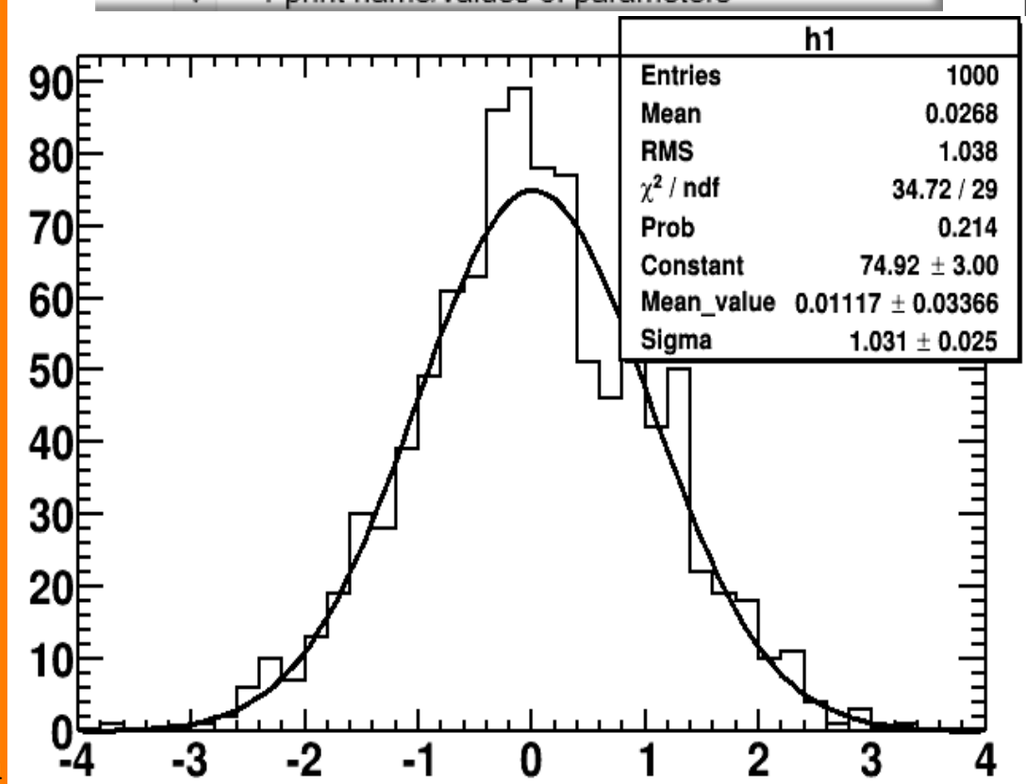
```
#include "TMath.h"
#include "TH1.h"
#include "TF1.h"
#include "TRandom3.h"
#include "TStyle.h"

// define a function with 3 parameters
Double_t fitf(Double_t *x,Double_t *par){
  Double_t arg = 0;
  if (par[2] != 0) arg = (x[0] - par[1])/par[2];
  Double_t fitval = par[0]*TMath::Exp(-0.5*arg*arg);
  return fitval;
}

void fitexample2() {
  TH1D * h1 = new TH1D("h1","Example",40,-4.,4.);
  for (int i = 0; i < 1000; ++i) {
    double x = gRandom->Gaus(0,1);
    h1->Fill(x);
  }
  TF1 *func= new TF1("func",fitf,-3,3,3);
  // set the parameters to mean and RMS of the histogram
  func->SetParameters(500,h1->GetMean(),h1->GetRMS());
  // give the parameters meaningful names
  func->SetParNames ("Constant","Mean_value","Sigma");
  gStyle->SetOptFit(1111);
  h1->Fit("func");
}
```

For displaying the fit parameters:

```
gStyle->SetOptFit(1111);
```

four digits: `mode = pcev` (`default = 0111`)

- `p` = 1 print probability
- `c` = 1 print Chi-square/number of degrees of freedom
- `e` = 1 print errors (if `e=1` , `v` must be 1)
- `v` = 1 print name/values of parameters



TF1 * f = new TF1("f",fobj,xmin,xmax,npar);
// create TF1 class with n-parameters and range [xmin,xmax]

$ROOTSYS/tutorials/fit/myfit.C

Fixing and Setting Parameters

- Parameters must be initialized to some value as close as possible to the expected values before invoking the Fit method.
  - To set bounds for one parameter, use **TF1**::SetParLimits:

```
func->SetParLimits(0, -1, 1);  //parameter 0 varies from -1 to 1
func->SetParameter(4, 10);     //initialize parameter 4 to 10
func->SetParLimits(4, 10, 10); //parameter 4 is fixed
```

  - To fix a parameter to 0, one must call the FixParameter function:

```
func->SetParameters(0, 3.1, 1.e-6, -1.5, 0, 100);
func->SetParLimits(3, -10, 4);
func->FixParameter(4, 0);
```

Retrieving The Fit Result

- The main results from the fit are stored in the fit function, which is attached to the histogram; it can be saved in a file (except for customized C/C++ functions).
- The fit function can be retrieved using its name:

```
TF1 * fitFunc = h1->GetFunction("f1");
```

- The parameter values using their indices (or their names):

```
fitFunc->GetParameter(par_index);
```

- The parameter errors:

```
fitFunc->GetParError(par_index);
```

- The Chisquare:

```
fitFunc->GetChisquare();;
```

- It is also possible to access the `TFitResult` class which has all information about the fit, if we use the fit option "S":

```
Minimizer is Minuit / Migrad
Chi2                      =     34.7164
NDf                       =          29
Edm                       =  1.94352e-08
NCalls                    =          61
Constant                  =     74.9237   +/-   3.00474
Mean                      =   0.0111669   +/-   0.0336562
Sigma                     =     1.03102   +/-   0.0253614
root [18]
```

```
TFitResultPtr r = h1->Fit(f1,"S");
r->Print();
TMatrixDSym C = r->GetCorrelationMatrix();
```

1 Introduction & Overview

- *Introduction*
- *Some basics statistics*
- *RooFit design philosophy*

RooFit: Your toolkit for data modeling

What is it?

- A powerful toolkit for modeling the expected distribution(s) of events in a physics analysis
- Primarily targeted to high-energy physicists using ROOT
- Originally developed for the BaBar collaboration by Wouter Verkerke and David Kirkby.
- Included with ROOT v5.xx

Documentation:

- http://root.cern.ch/root/Reference.html – for latest class descriptions. RooFit classes start with "Roo".
- http://roofit.sourceforge.net – for documentation and tutorials

Tutorials:

- Dig $ROOTSYS/tutorials/rootfit

RooFit purpose - Data Modeling for Physics Analysis

Distribution of observables $\vec{x}$

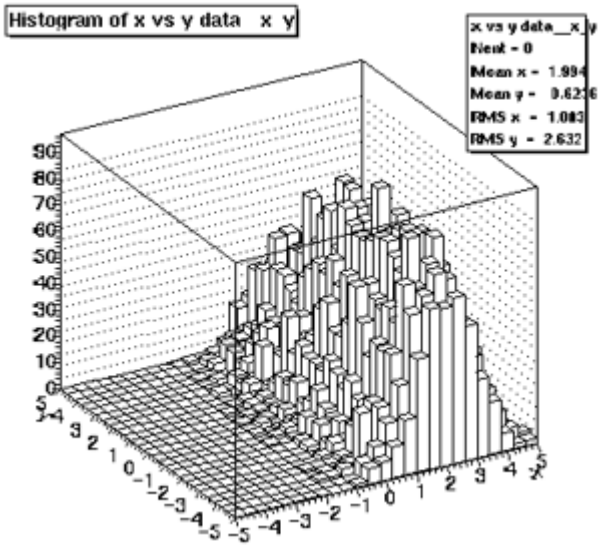
Define data model

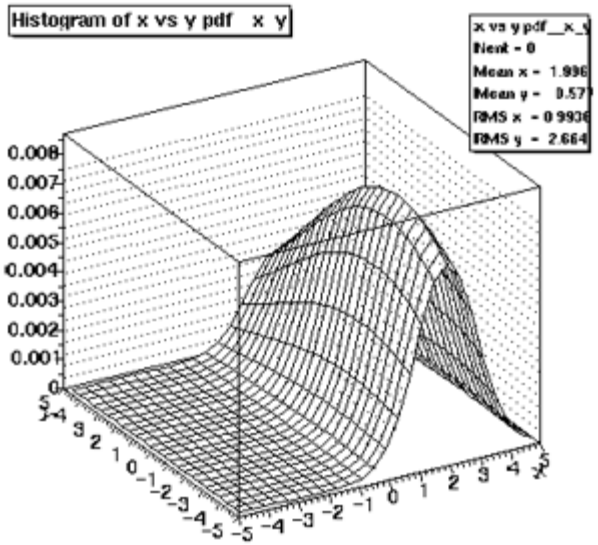
Probability Density Function $F(\vec{x}; \vec{p}, \vec{q})$

- Physical parameters of interest $\vec{p}$
- Other parameters $\vec{q}$ to describe detector effect (resolution,efficiency,…)
- Normalized over allowed range of the observables $\vec{x}$ w.r.t the parameters $\vec{p}$ and $\vec{q}$

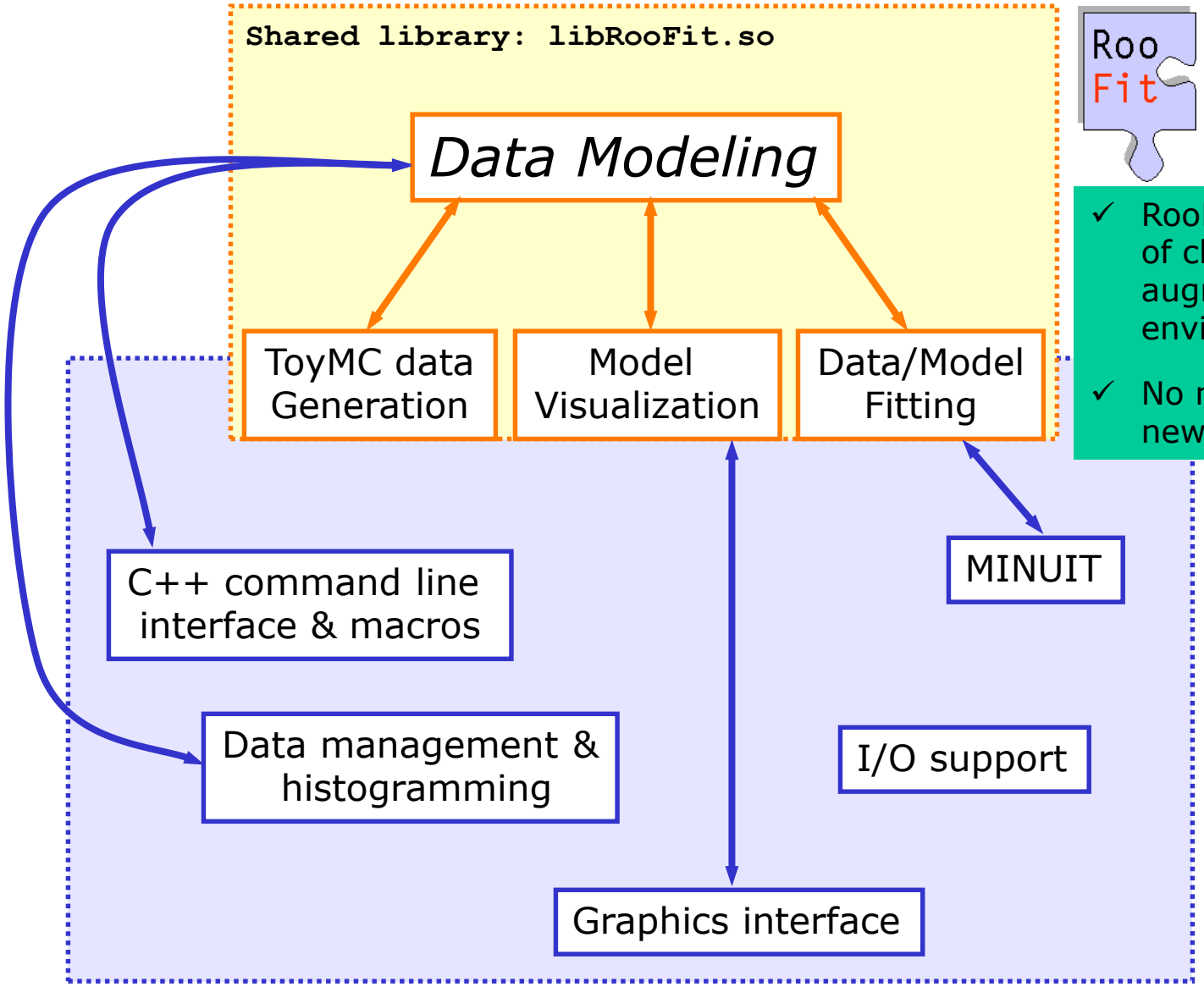
Fit model to data

Determination of $\vec{p},\vec{q}$





Implementation – Add-on package to ROOT



- ✓ RooFit is collection of classes that augment the ROOT environment.
- ✓ No need to learn new language.

Data modeling - Desired functionality

Analysis cycle

Building/Adjusting Models

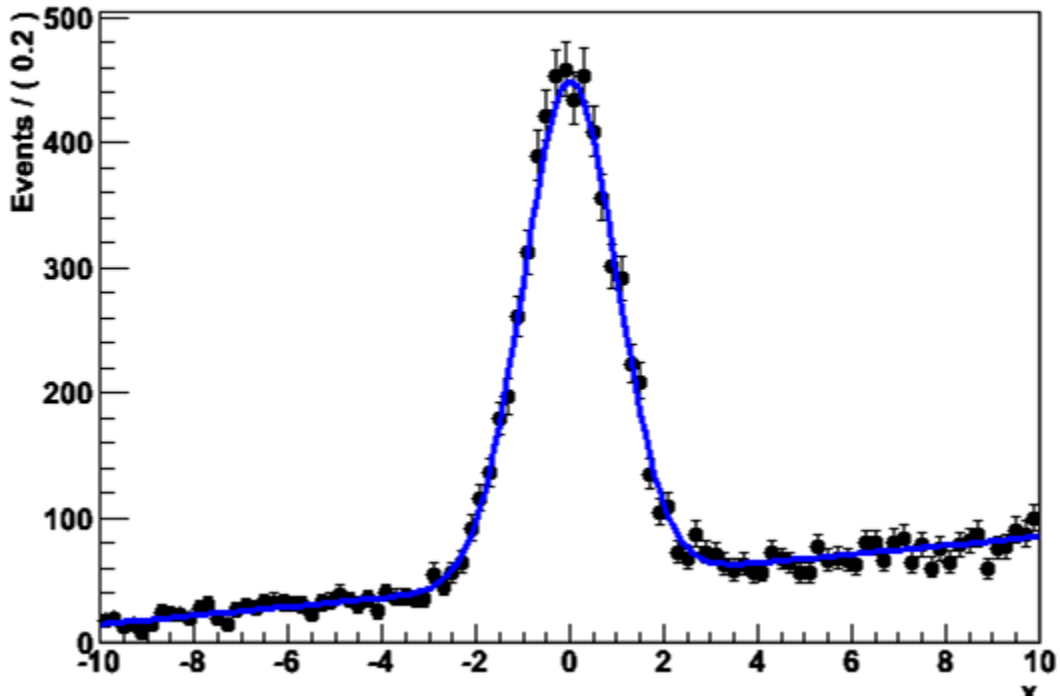
- ✓ *Easy to write* basic PDFs (→ normalization)
- ✓ Easy to *compose complex models* (modular design)
- ✓ *Reuse* of existing functions
- ✓ *Flexibility* – No arbitrary implementation-related restrictions

Using Models

- ✓ Fitting : Binned/Unbinned (extended) MLL fits, Chi^2 fits
- ✓ Toy MC generation: Generate MC datasets from *any* model
- ✓ Visualization: Slice/project model & data in *any possible way*
- ✓ Speed – Should be *as fast or faster* than hand-coded model

Introduction -- Focus: coding a probability density function

- Focus on one practical aspect of many data analysis in HEP: How do you formulate your p.d.f. in ROOT
  - For 'simple' problems (gauss, polynomial), ROOT built-in models well sufficient

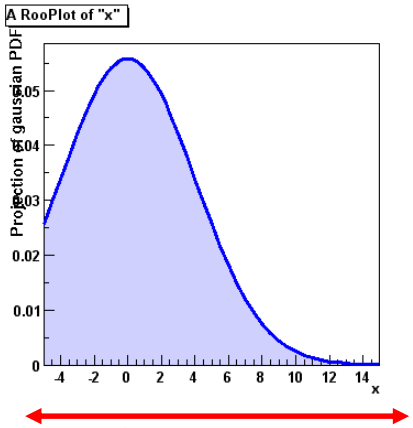


  - But if you want to do unbinned ML fits, use non-trivial functions, or work with multidimensional functions you are quickly running into trouble

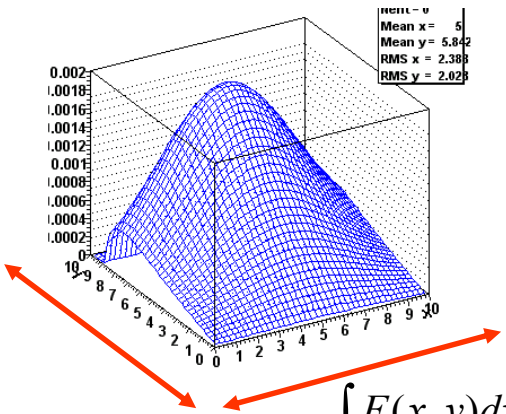
Mathematic – Probability density functions

- Probability Density Functions describe probabilities, thus
  - All values most be >0
  - The total probability must be 1 *for each p*, i.e.
  - Can have any number of dimensions

$$\int_{\vec{x}_{\min}}^{\vec{x}_{\max}} g(\vec{x}, \vec{p}) d\vec{x} \equiv 1$$



$$\int F(x) dx \equiv 1$$

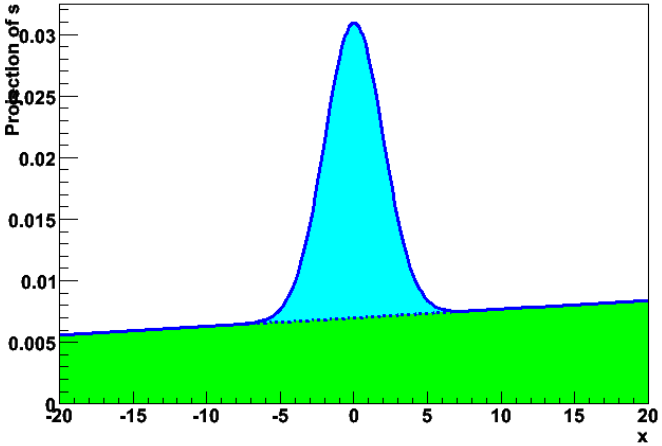


$$\int F(x, y) dx dy \equiv 1$$

- Note distinction in role between *parameters* (p) and *observables* (x)
  - Observables are measured quantities
  - Parameters are degrees of freedom in your model

Math – Functions vs probability density functions

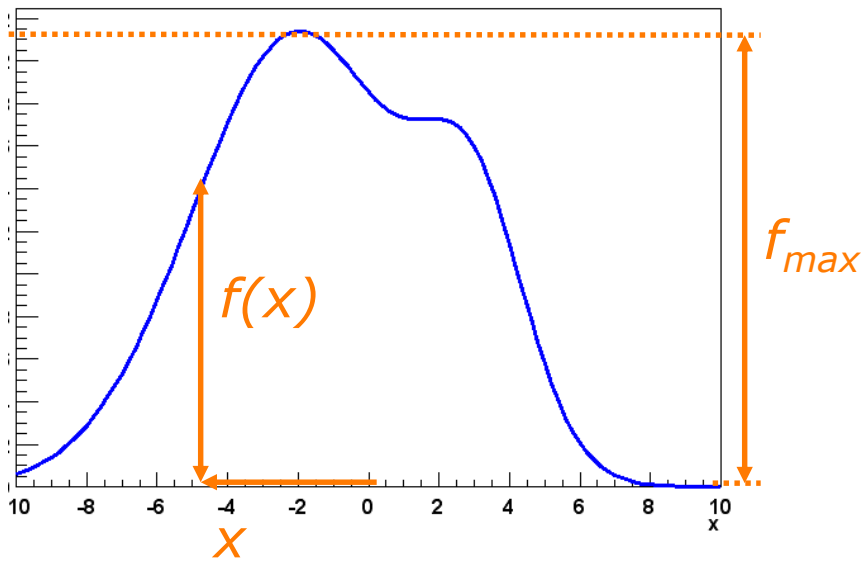
- Why use *probability density* functions rather than 'plain' functions to describe your data?
  - *Easier to interpret your models*. If Blue and Green pdf are each guaranteed to be normalized to 1, then fractions of Blue,Green can be cleanly interpreted as #events
  - Many statistical techniques only function properly with PDFs (e.g maximum likelihood)
  - Can sample 'toy Monte Carlo' events from p.d.f because value is always guaranteed to be >=0



- So why is not everybody always using them
  - The normalization can be hard to calculate (e.g. it can be different for each set of parameter values p)
  - In >1 dimension (numeric) integration can be particularly hard
  - RooFit aims to simplify these tasks

Math – Event generation

- For every p.d.f, can generate 'toy' event sample as follows
  - Determine maximum PDF value by repeated random sample
  - Throw a uniform random value (x) for the observable to be generated
  - Throw another uniform random number between 0 and fmax
    If $ran*f_{max} < f(x)$ accept x as generated event
  - *More efficient techniques exist*



Math – What is an estimator?

- An ***estimator*** is a ***procedure*** giving a value for a parameter or a property of a distribution as a function of the actual data values, i.e.

$$\hat{\mu}(x) = \frac{1}{N}\sum_i x_i$$ ← **Estimator of the mean**

$$\hat{V}(x) = \frac{1}{N}\sum_i (x_i - \vec{\mu})^2$$ ← **Estimator of the variance**

- A perfect estimator is （一致性，无偏性，有效性）
  - Consistent: $\lim_{n\to\infty}(\hat{a}) = a$
  - Unbiased – *With finite statistics you get the right answer on average*
  - Efficient $V(\hat{a}) = \left\langle (\hat{a} - \langle \hat{a} \rangle)^2 \right\rangle$ ← This is called the 最小方差界 **Minimum Variance Bound**
  - ***There are no perfect estimators for real-life problems***

Math – The Likelihood estimator

- Definition of Likelihood
  - given **D($\vec{x}$)** and **F($\vec{x}$;$\vec{p}$)**

Functions used in likelihoods must be Probability Density Functions:

$$\int F(\vec{x};\vec{p})d\vec{x} \equiv 1, \quad F(\vec{x};\vec{p}) > 0$$

$$L(\vec{p}) = \prod_i F(\vec{x}_i;\vec{p}), \quad \text{i.e.} \quad L(\vec{p}) = F(x_0;\vec{p}) \cdot F(x_1;\vec{p}) \cdot F(x_2;\vec{p}) \ldots$$

  - For convenience the *negative log* of the Likelihood is often used

$$-\ln L(\vec{p}) = -\sum_i \ln F(\vec{x}_i;\vec{p})$$

- Parameters are estimated by maximizing the Likelihood, or equivalently minimizing –log(L)

$$\left.\frac{d \ln L(\vec{p})}{d\vec{p}}\right|_{p_i = \hat{p}_i} = 0$$

Math – Variance on ML parameter estimates

- **Estimator** for the **parameter variance** is

$$\hat{\sigma}(p)^2 = \hat{V}(p) = \left(\frac{d^2 \ln L}{d^2 p} \right)^{-1}$$

  - I.e. variance is estimated from 2nd derivative of –log(L) at minimum
  - Valid if estimator is **efficient** and **unbiased**!

From Rao-Cramer-Frechet inequality

$$V(\hat{p}) \geq \frac{1 + \frac{db}{dp}}{\left(\frac{d^2 \ln L}{d^2 p} \right)}$$

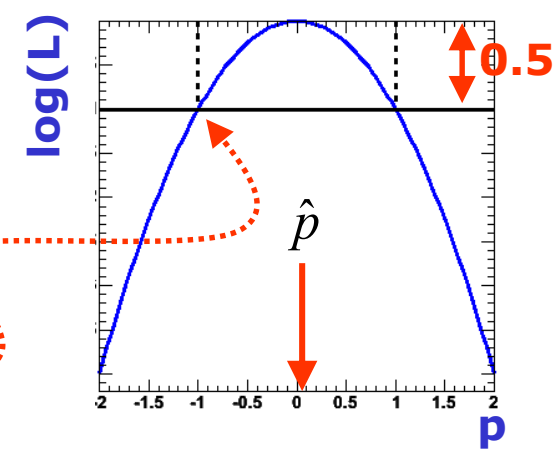
b = bias as function of p, inequality becomes equality in limit of efficient estimator

- **Visual interpretation** of variance estimate
  - Taylor expand –log(L) around minimum

$$\ln L(p) = \ln L(\hat{p}) + \left. \frac{d \ln L}{dp} \right|_{p=\hat{p}} (p - \hat{p}) + \tfrac{1}{2} \left. \frac{d^2 \ln L}{d^2 p} \right|_{p=\hat{p}} (p - \hat{p})^2$$

$$= \ln L_{\max} + \left. \frac{d^2 \ln L}{d^2 p} \right|_{p=\hat{p}} \frac{(p - \hat{p})^2}{2}$$

$$= \ln L_{\max} + \frac{(p - \hat{p})^2}{2\hat{\sigma}_p^2} \quad \Rightarrow \quad \ln L(p \pm \sigma) = \ln L_{\max} - \frac{1}{2}$$



Math – Properties of Maximum Likelihood estimators

- In general, Maximum Likelihood estimators are
  - **Consistent** (gives right answer for N→∞)
  - **Mostly unbiased** (bias ∝1/N, may need to worry at small N)
  - **Efficient for large N** (you get the smallest possible error)
  - **Invariant**: (a transformation of parameters will Not change your answer, e.g $(\hat{p})^2 = \widehat{(p^2)}$

Use of 2nd derivative of –log(L) for variance estimate is usually OK

- MLE efficiency theorem: the MLE will be unbiased and efficient if an unbiased efficient estimator exists

Math – Extended Maximum Likelihood

- Maximum likelihood information only parameterizes *shape* of distribution
  - I.e. one can determine *fraction* of signal events from ML fit, but not *number* of signal events

$$L(\vec{p}) = \prod_i F(\vec{x}_i;\vec{p}), \quad \text{i.e.} \quad L(\vec{p}) = F(x_0;\vec{p})\cdot F(x_1;\vec{p})\cdot F(x_2;\vec{p})\ldots$$

- Extended Maximum likelihood add extra term

The information can be incorporated by combining the standard maximum likelihood with the knowledge that a particular $Q(x;a)$ predicts ν events in the observed range, and accordingly multiplies the likelihood of a given data sample of N events by the Poisson probability of obtaining N events from a mean of ν: → $e^{-\nu}\frac{\nu^N}{N!}$.

$$-\log(L(\vec{p})) = -\sum_D \log(g(\vec{x}_i,\vec{p})) + \boxed{N_{\exp} - N_{obs}\log(N_{\exp})}$$

Log of Poisson(Nexp,Nobs) (modulo a constant)

  - Clever choice of parameters will allows us to extract N_{sig} and N_{bkg} in one pass ($N_{exp}=N_{sig}+N_{bkg}$, $f_{sig}=N_{sig}/(N_{sig}+N_{bkg})$)

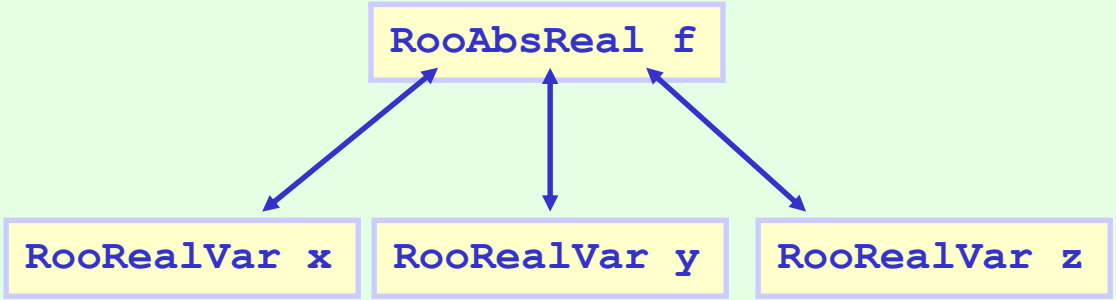
RooFit core design philosophy

- Mathematical objects are represented as C++ objects

Mathematical concept		RooFit class
variable	x	**RooRealVar**
function	$f(x)$	**RooAbsReal**
PDF	$f(x)$	**RooAbsPdf**
space point	$\vec{x}$	**RooArgSet**
integral	$\int_{x_{\min}}^{x_{\max}} f(x)dx$	**RooRealIntegral**
list of space points		**RooAbsData**

RooFit core design philosophy

- Represent relations between variables and functions as client/server links between objects

Math	$f(x,y,z)$
RooFit diagram	

RooFit code	`RooRealVar x("x","x",5) ;`
	`RooRealVar y("y","y",5) ;`
	`RooRealVar z("z","z",5) ;`
	`RooBogusFunction f("f","f",x,y,z) ;`

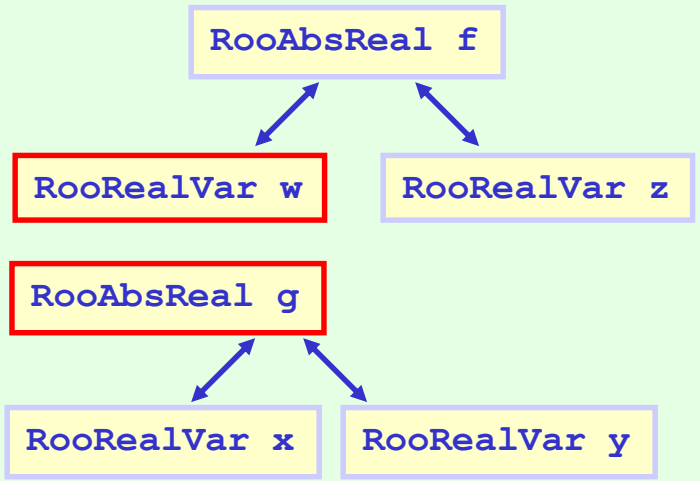
RooFit core design philosophy

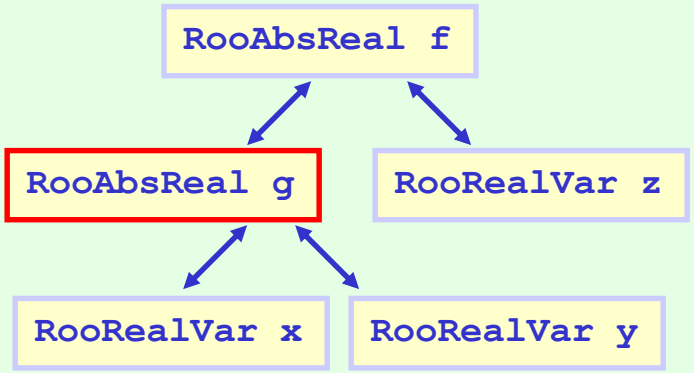
- Composite functions → Composite objects

Math

$$f(w,z) \qquad g(x,y) \quad \longrightarrow \quad f(g(x,y),z) = f(x,y,z)$$

RooFit diagram





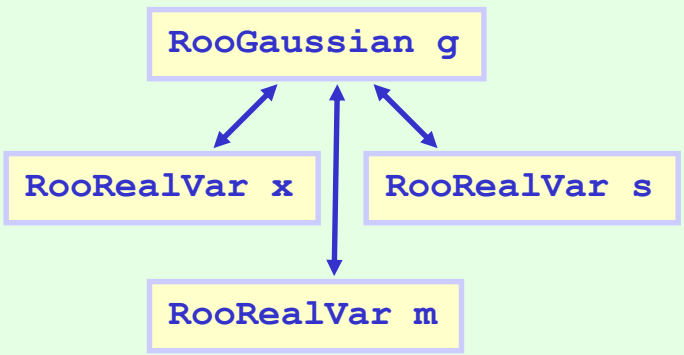
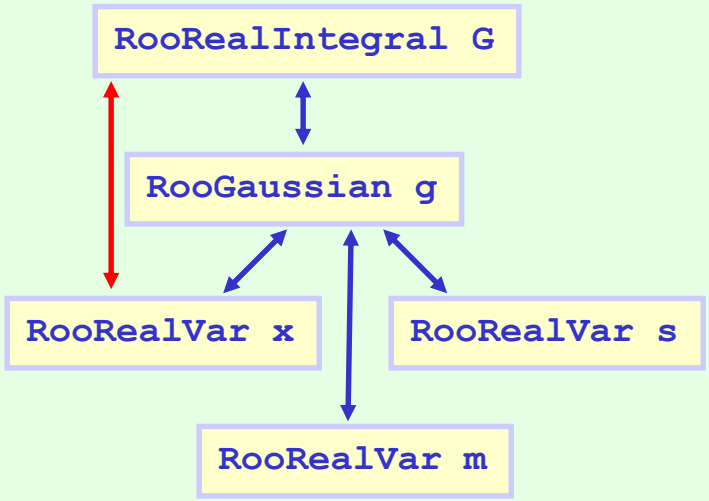
RooFit code

```
RooRealVar x("x","x",2) ;
RooRealVar y("y","y",3) ;
RooGooFunc g("g","g",x,y) ;

RooRealVar w("w","w",0) ;
RooRealVar z("z","z",5) ;
RooFooFunc f("f","f",w,z) ;
```

```
RooRealVar x("x","x",2) ;
RooRealVar y("y","y",3) ;
RooGooFunc g("g","g",x,y) ;

RooRealVar z("z","z",5) ;
RooFooFunc f("f","f",g,z) ;
```

RooFit core design philosophy

- Represent integral as an object,
  instead of representing integration as an action

Math	$g(x,m,s)$	$\int_{x_{min}}^{x_{max}} g(x,m,s)dx = G(m,s,x_{min},x_{max})$
RooFit diagram		

RooFit code	`RooRealVar x("x","x",2,-10,10)`
	`RooRealVar s("s","s",3) ;`
	`RooRealVar m("m","m",0) ;`
	`RooGaussian g("g","g",x,m,s)`
		`RooAbsReal *G =`
		`g.createIntegral(x) ;`

Object-oriented data modeling

- In RooFit every variable, data point, function, PDF represented in a C++ object
  - Objects classified by data/function type they represent, not by their role in a particular setup
  - All objects are **self documenting**
    - **Name** - Unique identifier of object
    - **Title** – More elaborate description of object

Initial range

Objects representing a 'real' value.

```
RooRealVar mass("mass","Invariant mass",5.20,5.30) ;
RooRealVar width("width","B0 mass width",0.00027,"GeV");
RooRealVar mb0("mb0","B0 mass",5.2794,"GeV") ;
```

Initial value Optional unit

PDF object

```
RooGaussian b0sig("b0sig","B0 sig PDF",mass,mb0,width);
```

References to variables

Object-oriented data modeling

- Elementary operations on value holder objects

Print value and attributes

```
mass.Print()
RooRealVar::mass:  5.2500 L(5.2 - 5.3)
```

Assign new value

```
mass = 5.27 ;
mass.setVal(5.27) ;
mass = 9.0  ;
RooAbsRealLValue::inFitRange(mass):
    value 9 rounded down to max limit 5.3
```

Error: new value out of allowed range

Retrieve contents

```
Double_t massVal = mass.getVal();
```

Print works for all RooFit objects

```
b0sig.Print()
RooGaussian::b0sig(mass,mb0,width) = 0
```

getVal() works for all real-valued objects (variables and functions)

```
Double_t val = b0sig.getVal()
```

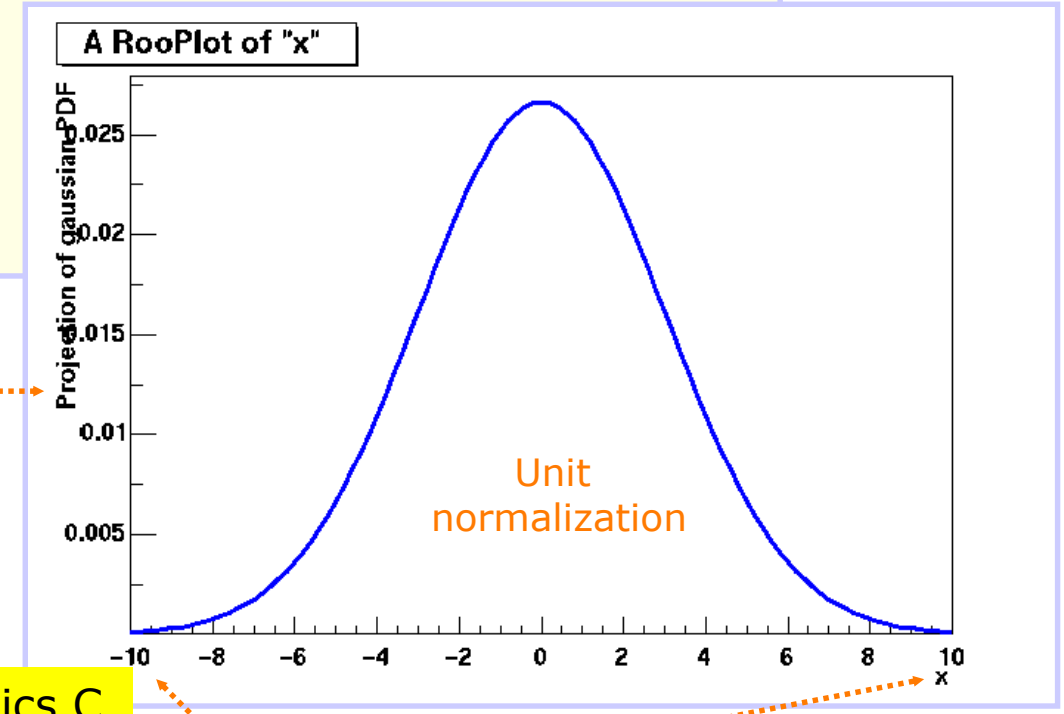
2 Basic Functionality

- *Creating a p.d.f*
- *Basic fitting, plotting, event generation*
- *Some details on normalization, event generation*
- *Library of basic shapes (including non-parametric shapes)*

Basics – Creating and plotting a Gaussian p.d.f

Setup gaussian PDF and plot

```
// Build Gaussian PDF
RooRealVar x("x","x",-10,10) ;
RooRealVar mean("mean","mean of gaussian",0,-10,10) ;
RooRealVar sigma("sigma","width of gaussian",3) ;

RooGaussian gauss("gauss","gaussian PDF",x,mean,sigma) ;

// Plot PDF
RooPlot* xframe = x.frame()
gauss.plotOn(xframe) ;
xframe->Draw() ;
```



Axis label from `gauss` title

A `RooPlot` is an empty frame capable of holding anything plotted versus it variable

$ROOTSYS/tutorials/roofit/rf101_basics.C

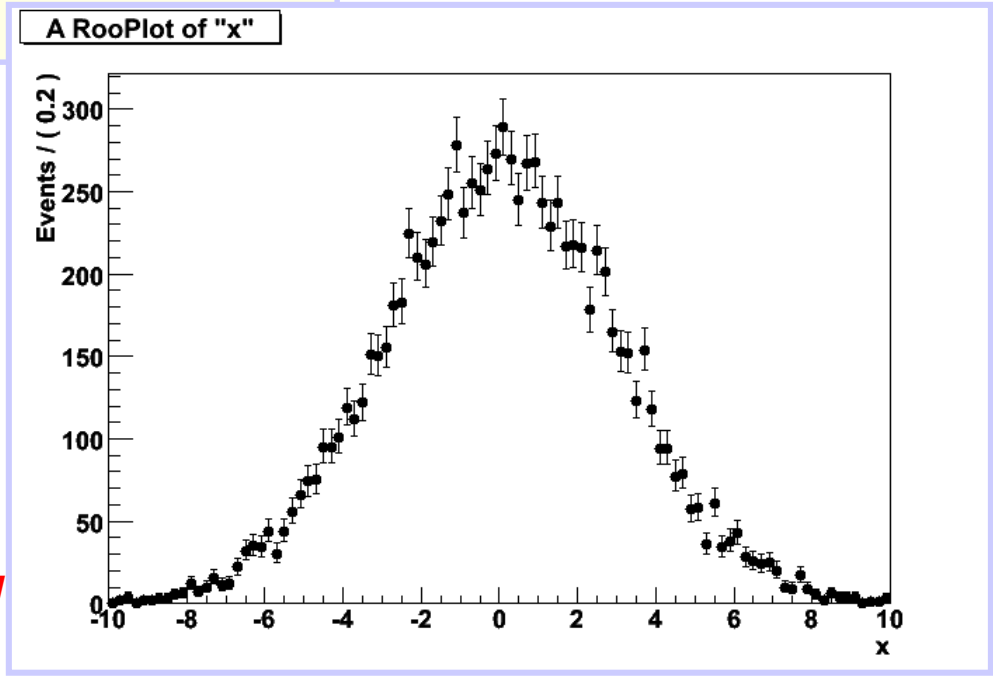
Plot range taken from limits of `x`

Basics – Generating toy MC events

demo1.cc

Generate 10000 events from Gaussian p.d.f and show distribution

```
// Generate a toy MC set
RooDataSet* data = gauss.generate(x,10000) ;

// Plot PDF
RooPlot* xframe = x.frame() ;
data->plotOn(xframe) ;
xframe->Draw() ;
```



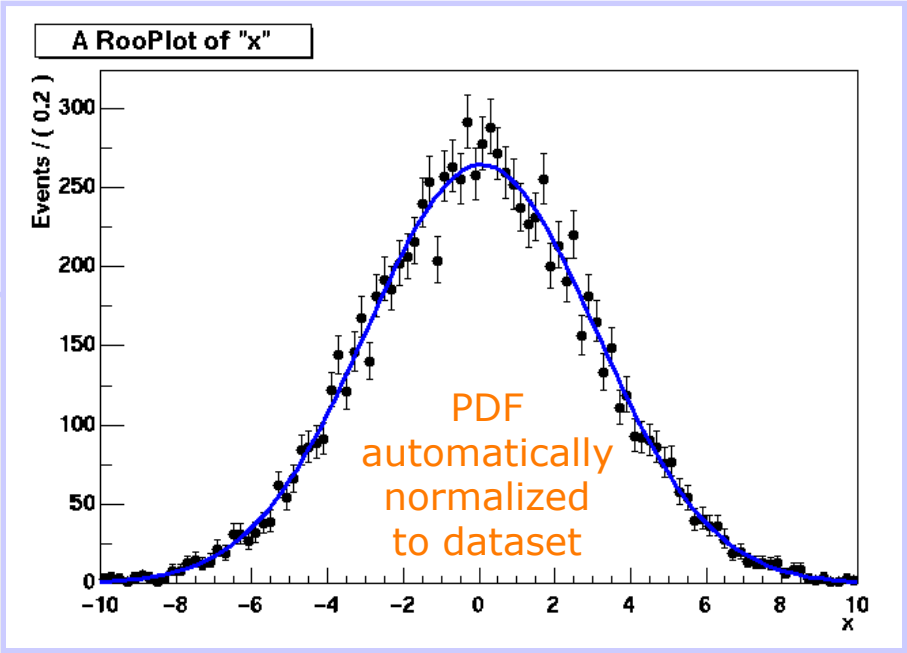
Returned dataset is ***unbinned*** dataset

Binning into histogram is performed in `data->plotOn()` call

Once the model is built, Generating ToyMC, fitting, plotting are mostly one-line operations!

Basics – ML fit of p.d.f to *unbinned* data

demo1.cc



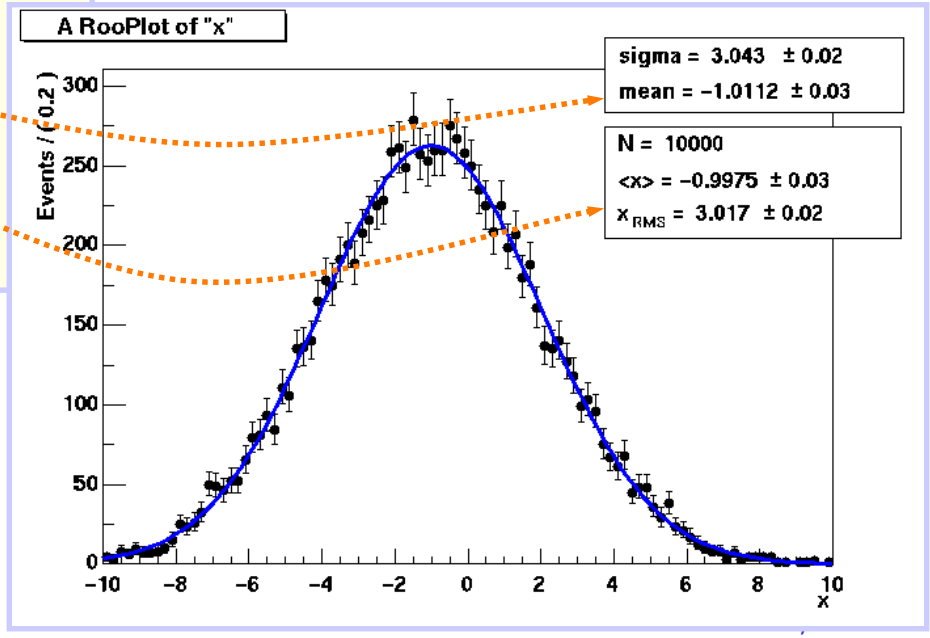

```
// ML fit of gauss to data
gauss.fitTo(*data) ;
(MINUIT printout omitted)

// Parameters if gauss now
// reflect fitted values
mean.Print()
RooRealVar::mean = 0.0172335 +/- 0.0299542
sigma.Print()
RooRealVar::sigma = 2.98094  +/- 0.0217306

// Plot fitted PDF and toy data overlaid
RooPlot* xframe2 = x.frame() ;
data->plotOn(xframe2) ;
gauss.plotOn(xframe2) ;
xframe2->Draw() ;
```

Basics – RooPlot Decoration

- A RooPlot is an empty frame that can contain
  - RooDataSet projections
  - PDF and generic real-valued function projections
  - Any ROOT drawable object (arrows, text boxes etc)

- Adding a dataset statistics box / PDF parameter box

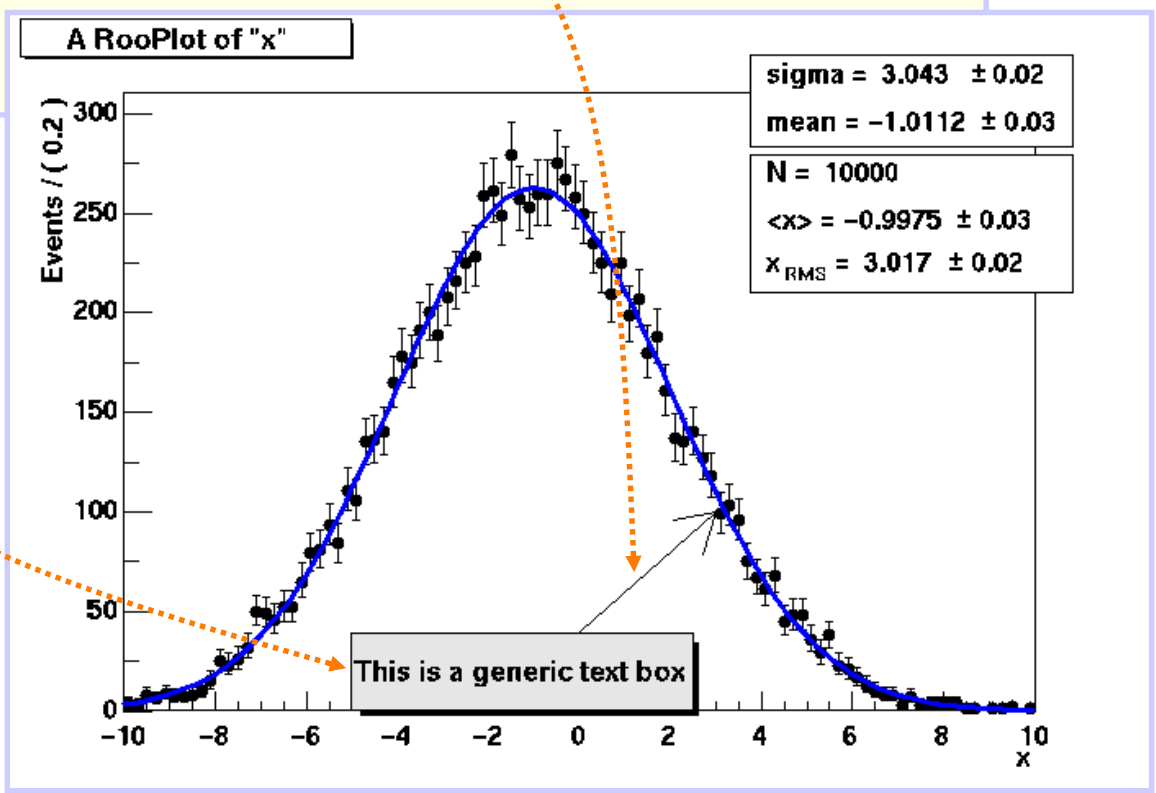
```
RooPlot* frame = x.frame() ;
data.plotOn(xframe) ;
pdf.plotOn(xframe) ;
pdf.paramOn(xframe,data) ;
data.statOn(xframe) ;
xframe->Draw() ;
```



Basics – RooPlot decoration

- Adding generic ROOT text boxes, arrows etc.

```
TPaveText* tbox = new TPaveText(0.3,0.1,0.6,0.2,"BRNDC");
tbox->AddText("This is a generic text box") ;
TArrow* arr = new TArrow(0,40,3,100) ;

xframe2->addObject(arr) ;
xframe2->addObject(tbox) ;
```



You can save a RooPlot with all its decorations in a ROOT file

Basics – Observables and parameters of Gauss

- Class **`RooGaussian`** has *no intrinsic notion* of distinction between observables and parameters

- Distinction always implicit in use context with dataset
  - **x** = observable (as it is a variable in the dataset)
  - **mean,sigma** = parameters

- Choice of observables (for unit normalization) always passed to **`gauss.getVal()`**

```
gauss.getVal();      // Not normalized (i.e. this is _not_ a pdf)
gauss.getVal(x);     // Guarantees Int[xmin,xmax] Gauss(x,m,s)dx==1
gauss.getVal(sigma);// Guarantees Int[smin,smax] Gauss(x,m,s)ds==1
```

How does it work – Normalization

- Flexible choice of normalization facilitated by explicit normalization step in RooFit p.d.f.s

`gauss.getVal(x)`

$$g(\mathbf{x};m,s)=\frac{g(x,m,s)}{\int_{x_{\min}}^{x_{\max}} g(x,m,s)dx}$$

`gauss.getVal(s)`

$$g(\mathbf{s};m,x)=\frac{g(x,m,s)}{\int_{s_{\min}}^{s_{\max}} g(x,m,s)ds}$$

- Supporting class `RooRealIntegral` responsible for calculation of any

$$\int_{\vec{x}_{\min}}^{\vec{x}_{\max}} g(\vec{x};\vec{p})d\vec{x}$$

  - Negotiation with p.d.f on which (partial) integrals it can internally perform analytically
  - Missing parted are supplemented with numerical integration
  - Class `RooRealIntegral` can in principle integrate *everything.*

How does it work – Normalization

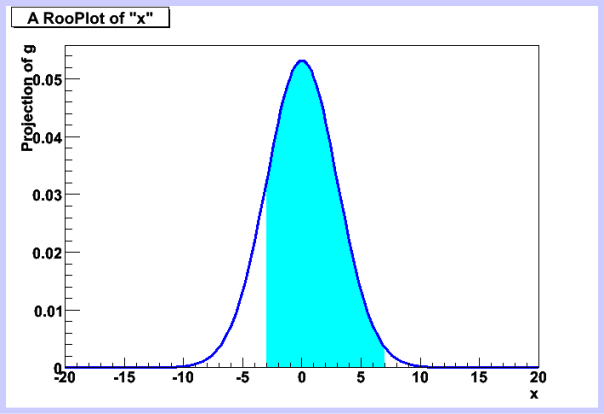
- A peak in the code of class **RooGaussian**

```
// Raw (unnormalized value) of Gaussian
Double_t RooGaussian::evaluate() const {
  Double_t arg= x - mean;
  return exp(-0.5*arg*arg/(sigma*sigma)) ;
}

// Advertise that x can be integrated internally
Int_t RooGaussian::getAnalyticalIntegral(RooArgSet& allVars,
      RooArgSet& analVars, const char* /*rangeName*/) const {
  if (matchArgs(allVars,analVars,x)) return 1 ;
  return 0 ;
}

// Implementation of analytical integral over x
Double_t RooGaussian::analyticalIntegral(Int_t code,
                                         const char* rname) const {
  static const Double_t root2 = sqrt(2.) ;
  static const Double_t rootPiBy2 = sqrt(atan2(0.0,-1.0)/2.0);
  Double_t xscale = root2*sigma;
  return rootPiBy2*sigma*(RooMath::erf((x.max(rname)-mean)/xscale)
                          -RooMath::erf((x.min(rname)-mean)/xscale));
}
```

Basics – Integrals over p.d.f.s

- It is easy to create an object representing integral over a normalized p.d.f in a sub-range

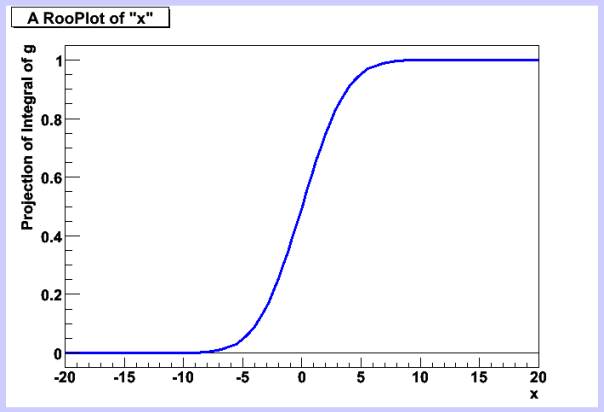
```
x.setRange("sig",-3,7) ;
RooAbsReal* ig = g.createIntegral(x,NormSet(x),Range("sig")) ;
cout << ig.getVal() ;
0.832519
mean=-1
cout << ig.getVal() ;
0.743677
```



- Similarly, one can also request the *cumulative distribution function*

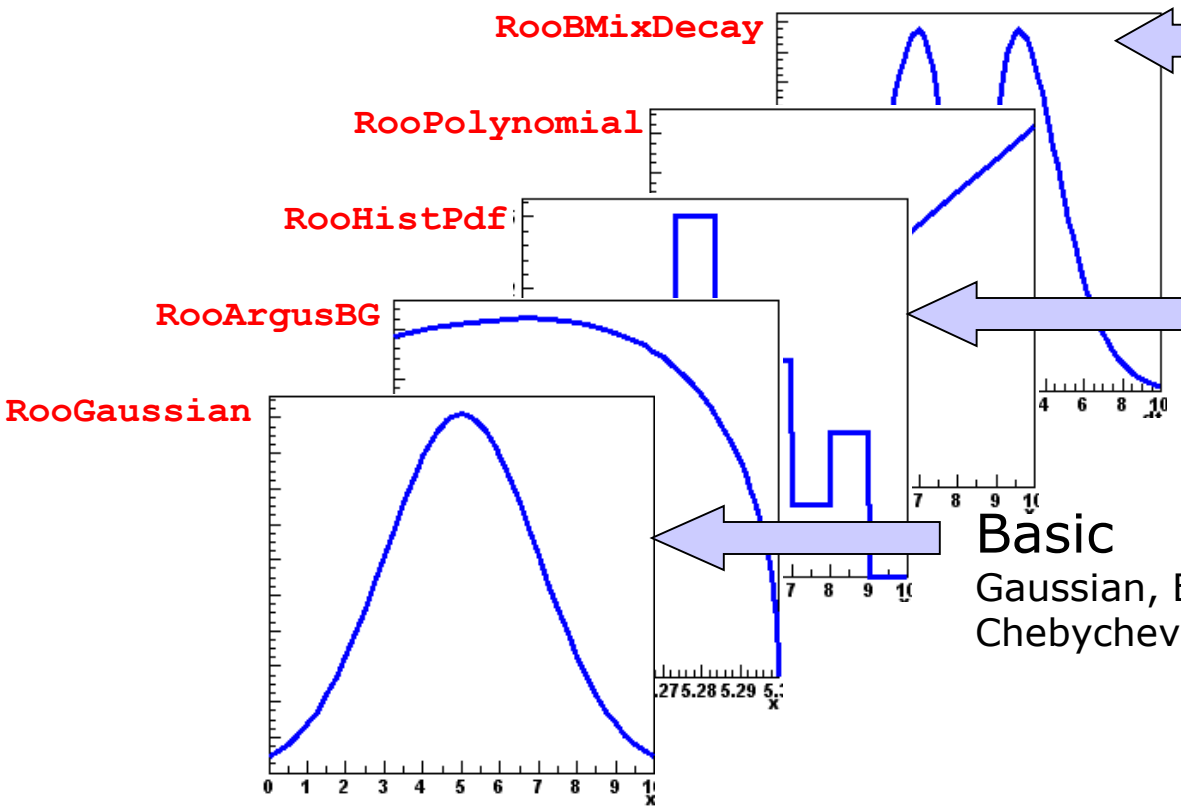
$$C(x) = \int_{x_{\min}}^{x} F(x')dx'$$

```
RooAbsReal* cdf = gauss.createCdf(x) ;
RooPlot* frame = x.frame() ;
cdf->plotOn(frame)->Draw() ;
```



Model building – (Re)using standard components

- RooFit provides a collection of compiled standard PDF classes



Physics inspired
ARGUS,Crystal Ball,
Breit-Wigner, Voigtian,
B/D-Decay,….

Non-parametric
Histogram, **KEYS**

Basic
Gaussian, Exponential, Polynomial,…
Chebychev polynomial

Easy to extend the library: each p.d.f. is a separate C++ class

The building blocks

- RooFitModels provides a collection of 'building block' PDFs

RooArgusBG	- Argus background shape
RooBCPEffDecay	- B0 decay with CP violation
RooBMixDecay	-B0 decay with mixing
RooBifurGauss	-Bifurcated Gaussian
RooBreitWigner	-Breit-Wigner shape
RooCBShape	-Crystal Ball function
RooChebychev	-Chebychev polynomial
RooDecay	-Simple decay function
RooDircPdf	-DIRC resolution description
RooDstD0BG	-D* background description
RooExponential	- Exponential function
RooGaussian	-Gaussian function
RooKeysPdf	-Non-parametric data description
Roo2DKeysPdf	-Non-parametric data description
RooPolynomial	-Generic polynomial PDF
RooVoigtian	-Breit-Wigner (X) Gaussian

- More will PDFs will follow
  - Easy to for users to write/contribute new PDFs

以上源程序都在 roofit/src 中

Model building – Generic expression-based PDFs

- If your favorite PDF isn't there
  and you don't want to code a PDF class right away
  → use **RooGenericPdf**

- Just write down the PDFs expression as a C++ formula

```
// PDF variables
RooRealVar x("x","x",-10,10) ;
RooRealVar y("y","y",0,5) ;
RooRealVar a("a","a",3.0) ;
RooRealVar b("b","b",-2.0) ;

// Generic PDF
RooGenericPdf gp("gp","Generic PDF","exp(x*y+a)-b*x",
                 RooArgSet(x,y,a,b)) ;
```

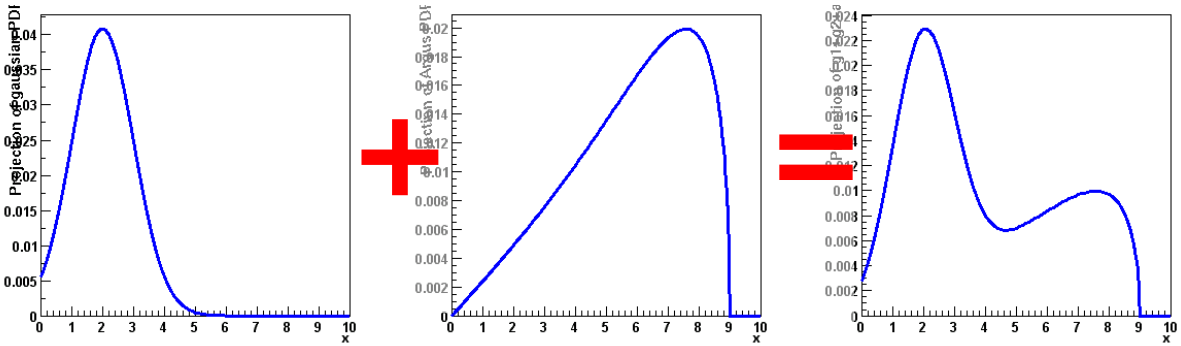
- Numeric normalization automatically provided

3 P.d.f. addition & convolution

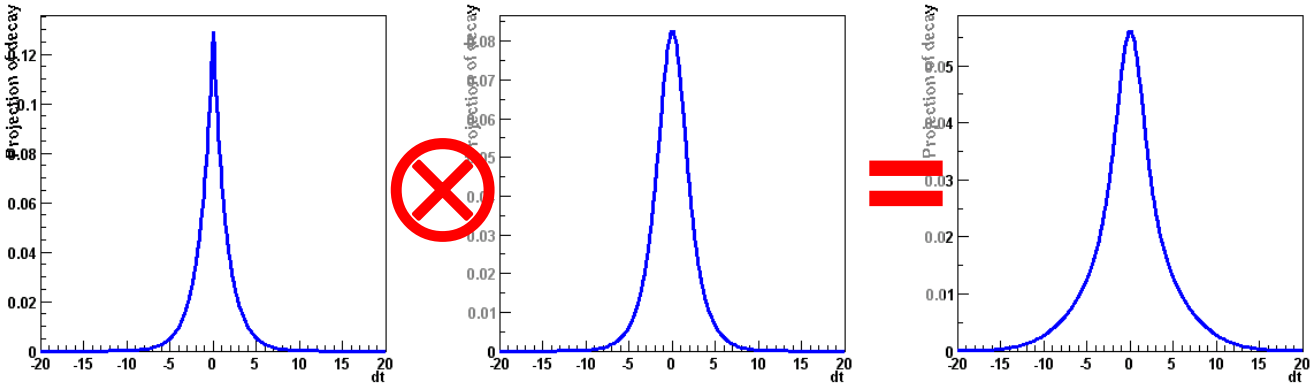
- *Using the addition operator p.d.f*
- *Using the convolution operator p.d.f.*

Building realistic models

- Complex PDFs be can be trivially composed using operator classes
  - Addition

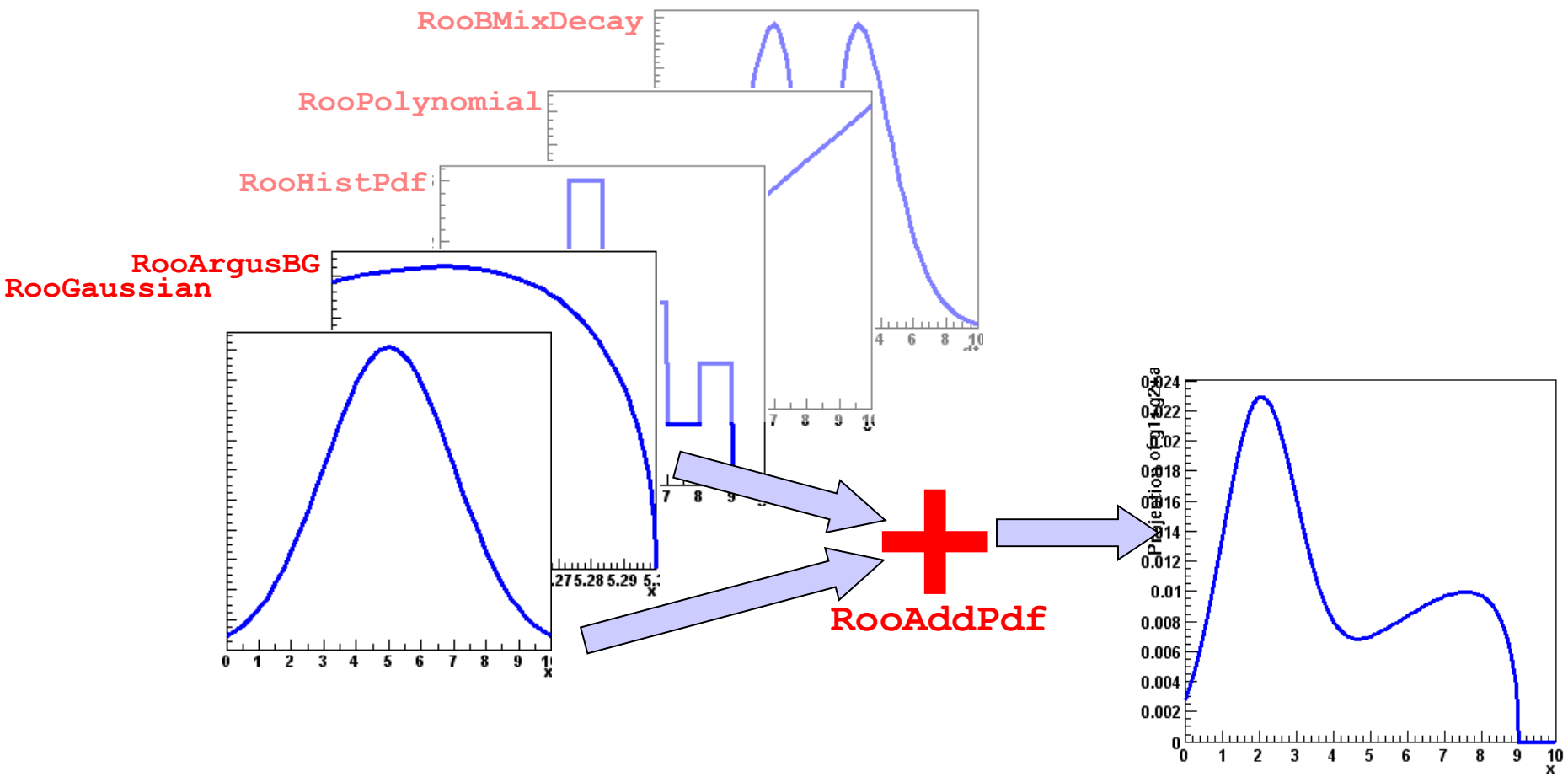


  - Convolution



Model building – (Re)using standard components

- Most realistic models are constructed as the sum of one or more p.d.f.s (e.g. signal and background)
- Facilitated through operator p.d.f **RooAddPdf**



Adding p.d.f.s – Mathematical side

- From math point of view adding p.d.f is simple
  - Two components *F, G*

$$S(x) = fF(x) + (1-f)G(x)$$

  - Generically for *N* components P_0-P_N

$$S(x) = c_0 P_0(x) + c_1 P_1(x) + \ldots + c_{n-1} P_{n-1}(x) + \left(1 - \sum_{i=0,n-1} c_i\right) P_n(x)$$

- For *N* p.d.f.s, there are *N-1* fraction coefficients that should sum to less 1
  - The remainder is by construction 1 minus the sum of all other coefficients

Constructing a sum of p.d.f.s

RooAddPdf constructs the sum of N PDFs with N-1 coefficients:

$$S = c_0 P_0 + c_1 P_1 + c_2 P_2 + \ldots + c_{n-1} P_{n-1} + \left(1 - \sum_{i=0,n-1} c_i\right) P_n$$

Build 2 Gaussian PDFs

Build ArgusBG PDF

```
// Build two Gaussian PDFs
RooRealVar x("x","x",0,10) ;
RooRealVar mean1("mean1","mean of gaussian 1",2) ;
RooRealVar mean2("mean2","mean of gaussian 2",3) ;
RooRealVar sigma("sigma","width of gaussians",1) ;
RooGaussian gauss1("gauss1","gaussian PDF",x,mean1,sigma) ;
RooGaussian gauss2("gauss2","gaussian PDF",x,mean2,sigma) ;

// Build Argus background PDF
RooRealVar argpar("argpar","argus shape parameter",-1.0) ;
RooRealVar cutoff("cutoff","argus cutoff",9.0) ;
RooArgusBG argus("argus","Argus PDF",x,cutoff,argpar) ;

// Add the components
RooRealVar g1frac("g1frac","fraction of gauss1",0.5) ;
RooRealVar g2frac("g2frac","fraction of gauss2",0.1) ;
RooAddPdf  sum("sum","g1+g2+a",RooArgList(gauss1,gauss2,argus),
                               RooArgList(g1frac,g2frac)) ;
```

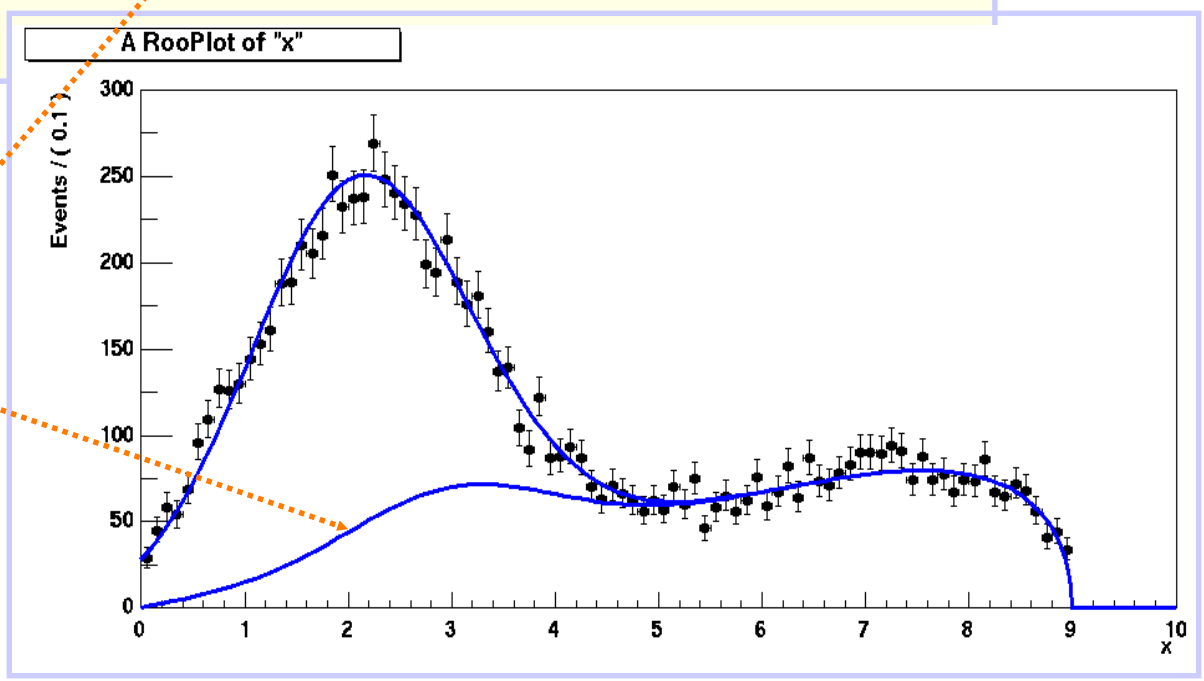
List of PDFs

List of coefficients

Plotting a sum of p.d.f.s, and its components

```
// Generate a toyMC sample
RooDataSet *data =
     sum.generate(x,10000) ;

// Plot data and PDF overlaid
RooPlot* xframe = x.frame() ;
data->plotOn(xframe) ;
sum->plotOn(xframe) ;

// Plot only argus and gauss2
sum->plotOn(xframe,Components(RooArgSet(argus,gauss2))) ;
xframe->Draw() ;
```

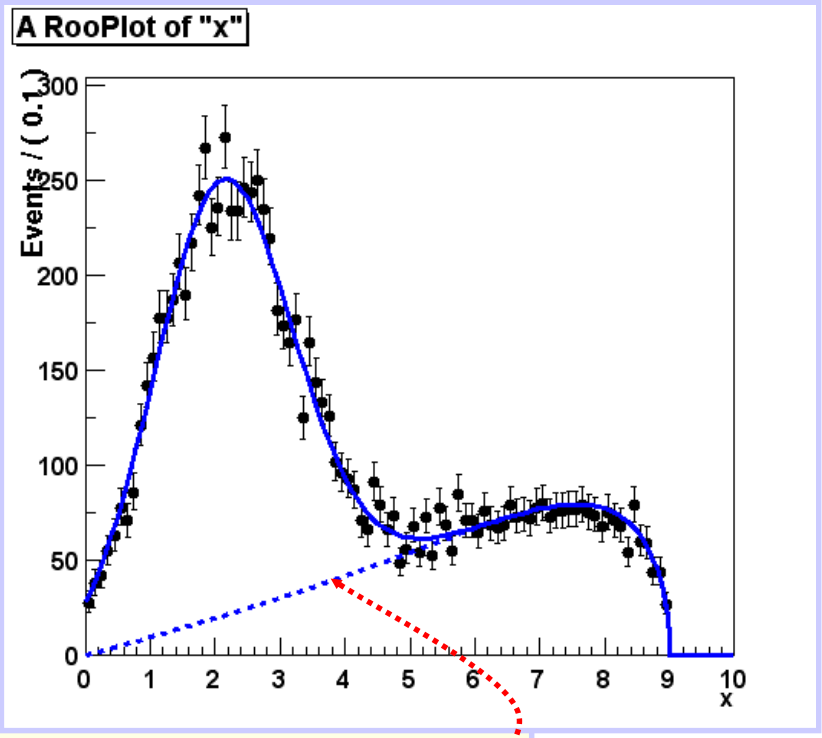
Plot selected components of a **RooAddPdf**



Component plotting - Introduction

- Also special tools for plotting of components in RooPlots
  - Use Method `Components()`

- Example:
  Argus + Gaussian PDF




```
// Plot data and full PDF first

// Now plot only argus component
sum->plotOn(xframe,
            Components(argus), LineStyle(kDashed)) ;
```

Component plotting – Selecting components

There are various ways to select single or multiple components to plot

Can refer to components either by name or reference

```
// Single component selection
pdf->plotOn(frame,Components(argus)) ;
pdf->plotOn(frame,Components("gauss")) ;


// Multiple component selection
pdf->plotOn(frame,Components(RooArgSet(pdfA,pdfB)) ;
pdf->plotOn(frame,Components("pdfA,pdfB")) ;



// Wild card expression allowed
pdf->plotOn(frame,Components("bkgA*,bkgB*")) ;
```

Recursive fraction form of RooAddPdf

- Fitting a sum of >2 p.d.f.s can pose some problems as the sum of the coefficients $f_1 \ldots f_{N-1}$ may become >1
  - This results in a negative remainder component ($\equiv 1-\Sigma_i f_i$)
  - Composite p.d.f may still be positive definite, but interpretation less clear
  - Could set limits on fractions f_i to avoid $\Sigma f_i > 1$ scenario, but where to put limits?
- Viable alternative to write as sum of *recursive* fractions

$$S_2(x) = f_1 P_1(x) + (1-f_1) P_2(x)$$
$$S_3(x) = f_1 P_1(x) + (1-f_1)\big(f_2 P_2(x) + (1-f_2) P_3(x)\big)$$
$$S_4(x) = f_1 P_1(x) + (1-f_1)\big(f_2 P_2(x) + (1-f_2)\big(f_3 P_3(x) + (1-f_3) P_4(x)\big)\big)$$

```
// Add the components with recursive fractions
RooAddPdf  sum("sum",“fA*a+(fG*g1+g2)",RooArgList(a,g1,g2),
                                       RooArgList(afrac,gfrac),kTRUE) ;
```

参考 tutorials/roofit/rf201_composite.C

Extended p.d.f form of RooAddPdf

- If extended ML term is introduced, we can fit expected number of events (N_{exp}) in addition to shape parameters
- In case of sum of p.d.f.s it is convenient to *re-parameterize* sum of p.d.f.s.

$$\begin{pmatrix} f_{sig} \\ N_{\exp} \end{pmatrix} \Rightarrow \begin{pmatrix} N_{sig} \equiv f_{sig} N_{\exp} \\ N_{bkg} \equiv (1 - f_{sig}) N_{\exp} \end{pmatrix}$$

- This transformation is applied automatically in **RooAddPdf** if equal number of p.d.f.s and coefs are given

```
RooRealVar nsig("nsig","number of signal events",100,0,10000) ;
RooRealVar nbkg("nbkg","number of backgnd events",100,0,10000) ;
RooAddPdf sume("sume","extended sum pdf",RooArgList(gauss,argus),
                                          RooArgList(nsig,nbkg)) ;
```

General features of extended p.d.f.s

- Extended term –log(Poisson(N_{obs},N_{exp})) is not added by default to likelihood
  - Use the `Extended()` argument to fit to have it added

```
// Regular maximum likelihood fit
pdf.fitTo(*data) ;

// Extended maximum likelihood fit
pdf.fitTo(*data,Extended(kTRUE)) ;
```

- If p.d.f. is extended, N_{exp} is default number of events to generate

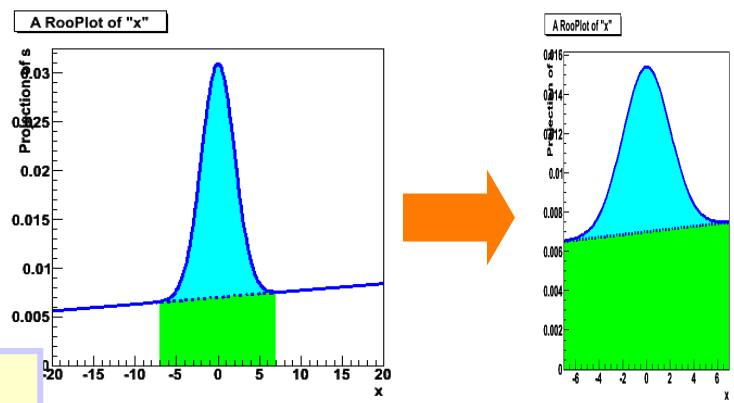
```
// Generate pdf.expectedEvents() events
RooDataSet* data = pdf.generate(x) ;

// Generate 1000 events
RooDataSet* data = pdf.generate(x,1000) ;
```

How it works – Normalization of **RooAddPdfs**

- Since all component p.d.f.s are normalized, resulting sum of p.d.f.s is automatically normalized
  - As long as sum of coefficients is 1, which is automatically enforced

$$S(x) = c_0 P_0(x) + c_1 P_1(x) + \ldots + c_{n-1} P_{n-1}(x) + \left(1 - \sum_{i=0,n-1} c_i\right) P_n(x)$$

  - But note that fraction parameter multiplies *normalized* p.d.f.s
- Interpretation of fraction depends on range of observables (and number of observables for >1D)
  - If range of observable is changed and fraction parameter is same, the shape effectively different
  - Can mitigate this by specifying a fixed reference range for fraction interpretation



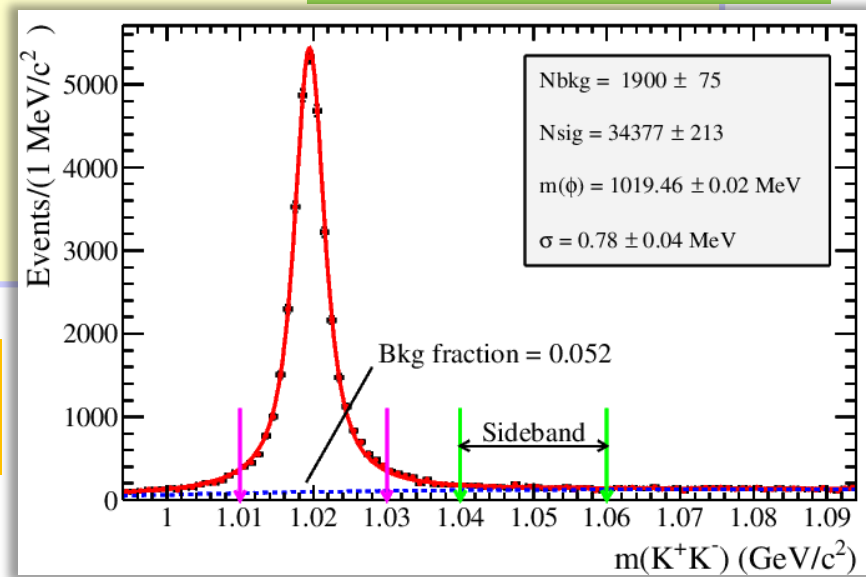

```
x.setRange("ref",-20,20) ;
pdf->setAddCoefRange("ref") ;
```

参考 tutorials/roofit/rf203_ranges.C，rf204_extrangefit.C，rf312_multirangefit.C

Extended ML fit with range definition

```
RooRealVar x("x", "m(K^{+}K^{-})", 0.994,1.094);
RooRealVar mass("Xmass", "Tmass", 1.02, 1.01 , 1.03);
RooRealVar width("Xwidth", "Twidth", 0.00426, 0.00 , 0.00);
RooRealVar sigma("Xsigma", "Tsigma", 0.00, 0.00 , 0.10);
RooVoigtian sig("Voigtian", "VTp.d.f", x, mass, width, sigma);
RooChebychev bkg("bkg","bkg",m34,RooArgList(c0,c1,c2));
double nmax = mkk->numEntries()+100;
RooRealVar nsig("nsig","#signal events",    nmax*0.4,0,nmax);
RooRealVar nbkg("nbkg","#background events",nmax*0.6,0,nmax);
m34.setRange("cut",1.01,1.03);
RooExtendPdf sige1 ("sige1","sige1",sig, nsig,"cut");
RooExtendPdf bkge1 ("bkge1","bkge1",bkg, nbkg,"cut");
RooAddPdf sum("sum","g+b",RooArgList(sige1,bkge1));
RooFitResult* r =sum.fitTo(*mkk,
RooFit::Extended(kTRUE),RooFit::Save(kTRUE));
r->Print("v");
RooPlot* phiplot = x.frame(100);
phiplot->Draw();
```

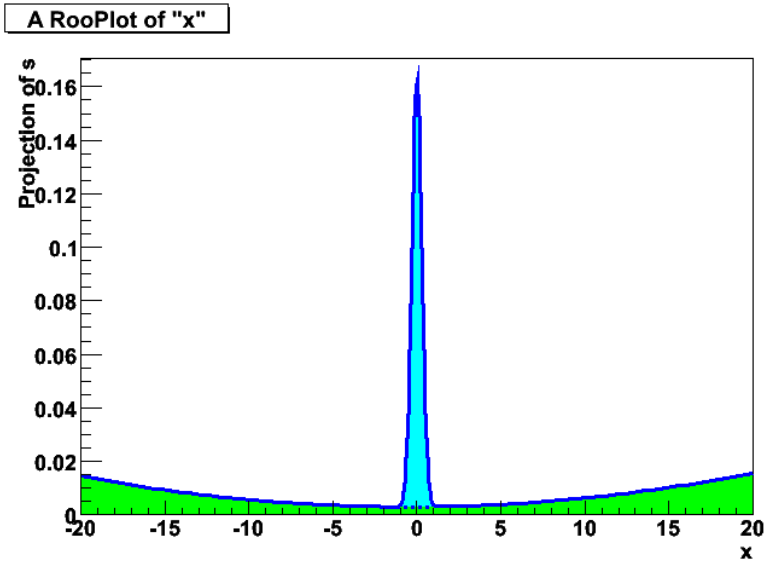
拟合得到的Nsig和Nbkg为信号区间1.01-1.03的事例数



类似的拟合脚本，参考
$ROOTSYS/tutorials/roofit/rf204_extrangefit.C

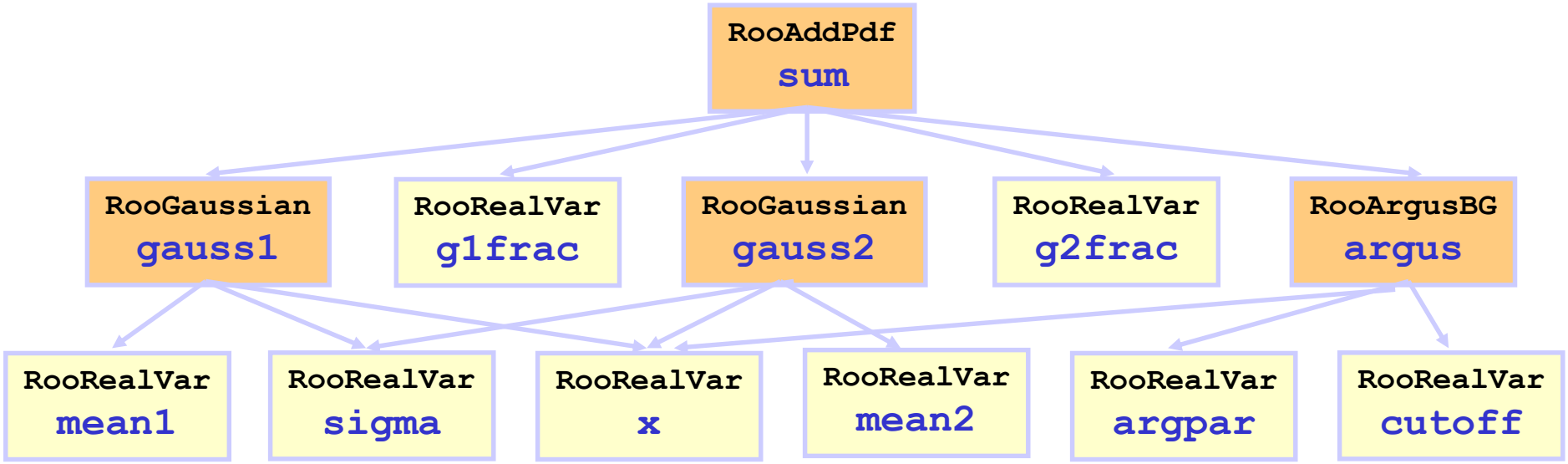
How it works – event generation of RooAddPdf

- Composite event generation algorithm of **`RooAddPdf`**
  - Choose randomly a component to generate (probability proportional to coefficient fractions)
  - Delegate generation of observable to algorithm of component p.d.f.
- Allows to efficiently handle sum of p.d.f with very different shapes in most cases
  - Example:
    Blue Gaussian
    (internal generator)
    plus Green Polynomial
    (accept/reject)



Dealing with composite p.d.f.s

- A RooAddPdf is an example of a composite p.d.f
  - The value of the sum is represented by a *tree* of components



  - The compositeness of a p.d.f. is completely transparent to most high-level operations
  - Can e.g. do `sum->fitTo(*data)` or `sum->generate(x,1000)` without being aware of composite nature of p.d.f.

Dealing with composite p.d.f.s

- The observables reported by a composite p.d.f and the 'leaf' of the expression tree
  - For example, request for list of parameters of composite sum, will return parameters of components of sum

```
RooArgSet *paramList = sum.getParameters(data) ;
paramList->Print("v") ;
RooArgSet::parameters:
  1) RooRealVar::argpar : -1.00000 C
  2) RooRealVar::cutoff :  9.0000 C
  3) RooRealVar::g1frac :  0.50000 C
  4) RooRealVar::g2frac :  0.10000 C
  5) RooRealVar::mean1  :  2.0000 C
  6) RooRealVar::mean2  :  3.0000 C
  7) RooRealVar::sigma  :  1.0000 C
```

- In general, composite p.d.f.s work *exactly the same* as basic p.d.f.s.

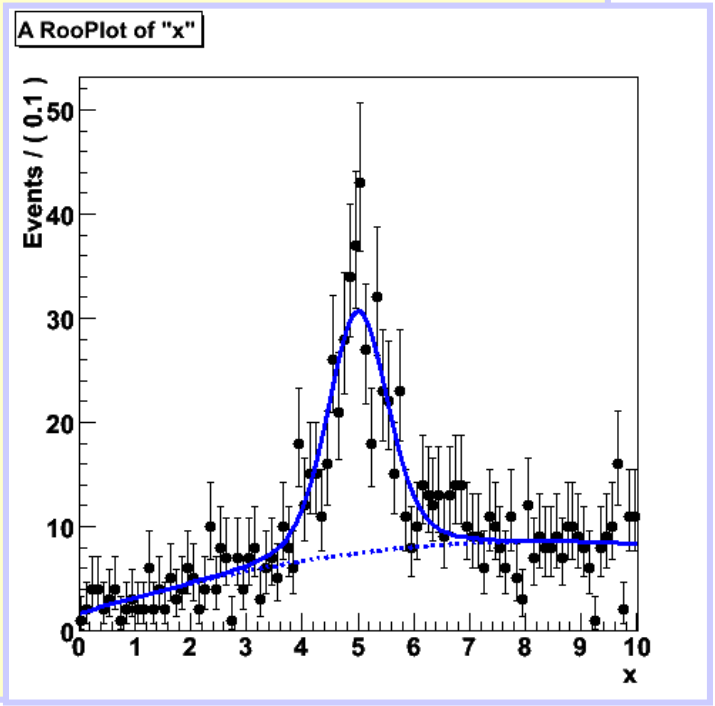
Visualization tools for composite objects

- Special tools exist to visualize the tree structure of composite objects
  - On the command line

```
Root> sum.Print("t") ;
0x927b8d0 RooAddPdf::sum (g1+g2+a)  [Auto]
  0x9254008 RooGaussian::gauss1 (gaussian PDF)  [Auto] V
    0x9249360 RooRealVar::x (x) V
    0x924a080 RooRealVar::mean1 (mean of gaussian 1) V
    0x924d2d0 RooRealVar::sigma (width of gaussians) V
  0x9267b70 RooRealVar::g1frac (fraction of gauss1) V
  0x9259dc0 RooGaussian::gauss2 (gaussian PDF)  [Auto] V
    0x9249360 RooRealVar::x (x) V
    0x924cde0 RooRealVar::mean2 (mean of gaussian 2) V
    0x924d2d0 RooRealVar::sigma (width of gaussians) V
  0x92680e8 RooRealVar::g2frac (fraction of gauss2) V
  0x9261760 RooArgusBG::argus (Argus PDF)  [Auto] V
    0x9249360 RooRealVar::x (x) V
    0x925fe80 RooRealVar::cutoff (argus cutoff) V
    0x925f900 RooRealVar::argpar (argus shape parameter) V
    0x9267288 RooConstVar::0.500000 (0.500000) V
```

Putting it all together – Extended unbinned ML Fit to signal and background

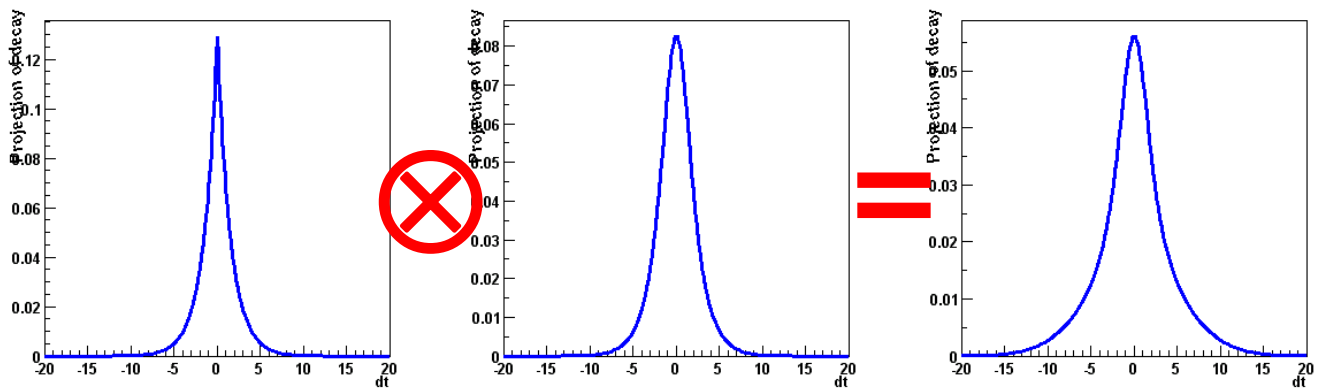
```
// Declare observable x
RooRealVar x("x","x",0,10) ;

// Creation of 'sig', 'bkg' component p.d.f.s omitted for clarity

// Model = Nsig*sig + Nbkg*bkg (extended form)
RooRealVar nsig("nsig","#signal events",300,0.,2000.) ;
RooRealVar nbkg("nbkg","#background events",700,0,2000.) ;
RooAddPdf  model("model","sig+bkg",RooArgList(sig,bkg),RooArgList(nsig,nbkg)) ;

// Generate a data sample of Nexpected events
RooDataSet *data = model.generate(x) ;

// Fit model to data
model.fitTo(*data, Extended(kTRUE)) ;

// Plot data and PDF overlaid
RooPlot* xframe = x.frame() ;
data->plotOn(xframe) ;
model.plotOn(xframe) ;
model.plotOn(xframe,Components(bkg),
             LineStyle(kDashed)) ;
xframe->Draw() ;
```



参考 tutorials/roofit/rf202_extendedmlfit.C

Building models – Convolutions

- Many experimental observable quantities are well described by convolutions
  - Typically physics distribution smeared with experimental resolution (e.g. for B0 → J/ψ K_S exponential decay distribution smeared with Gaussian)



  - By explicitly describing observed distribution with a convolution p.d.f can disentangle detector and physics
    - To the extent that enough information is in the data to make this possible

Mathematical introduction & Numeric issues

- Mathematical form of convolution
  - Convolution of two functions

$$f(x) \otimes g(x) = \int_{-\infty}^{+\infty} f(x) g(x - x') dx'$$

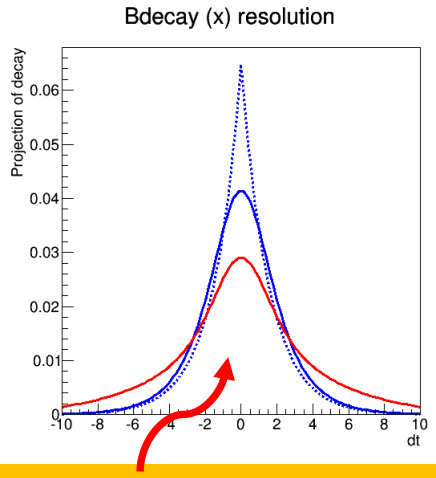
  - Convolution of two normalized p.d.f.s itself is *not* automatically normalized, so expression for convolution p.d.f is

$$F(x) \otimes G(x) = \frac{\int_{-\infty}^{+\infty} F(x) G(x - x') dx'}{\int_{x_{\min}}^{x_{\max}} \int_{-\infty}^{+\infty} F(x) G(x - x') dx' dx}$$

  - Because of (multiple) integrations required convolution are difficult to calculate
  - Convolution integrals are best done analytically, but often not possible

Convolution operation in RooFit

- RooFit has several options to construct convolution p.d.f.s
  - Class **RooNumConvPdf** – 'Brute force' numeric calculation of convolution (and normalization integrals)
  - Class **RooFFTConvPdf** – Calculate convolution integral using discrete FFT technology in fourier-transformed space.
  - Bases classes **RooAbsAnaConvPdf**, **RooResolutionModel**. Framework to construct analytical convolutions (with implementations mostly for B physics)
  - Class **RooVoigtian** – Analytical convolution of non-relativistic Breit-Wigner shape with a Gaussian
- All convolution in one dimension so far
  - N-dim extension of **RooFFTConvPdf** foreseen in future



参考 tutorials/roofit/rf209_anaconv.C
(分别卷积delta function、 Gaussian和double Gaussian)