

Part 1

Luciano Pandola
INFN, LNS

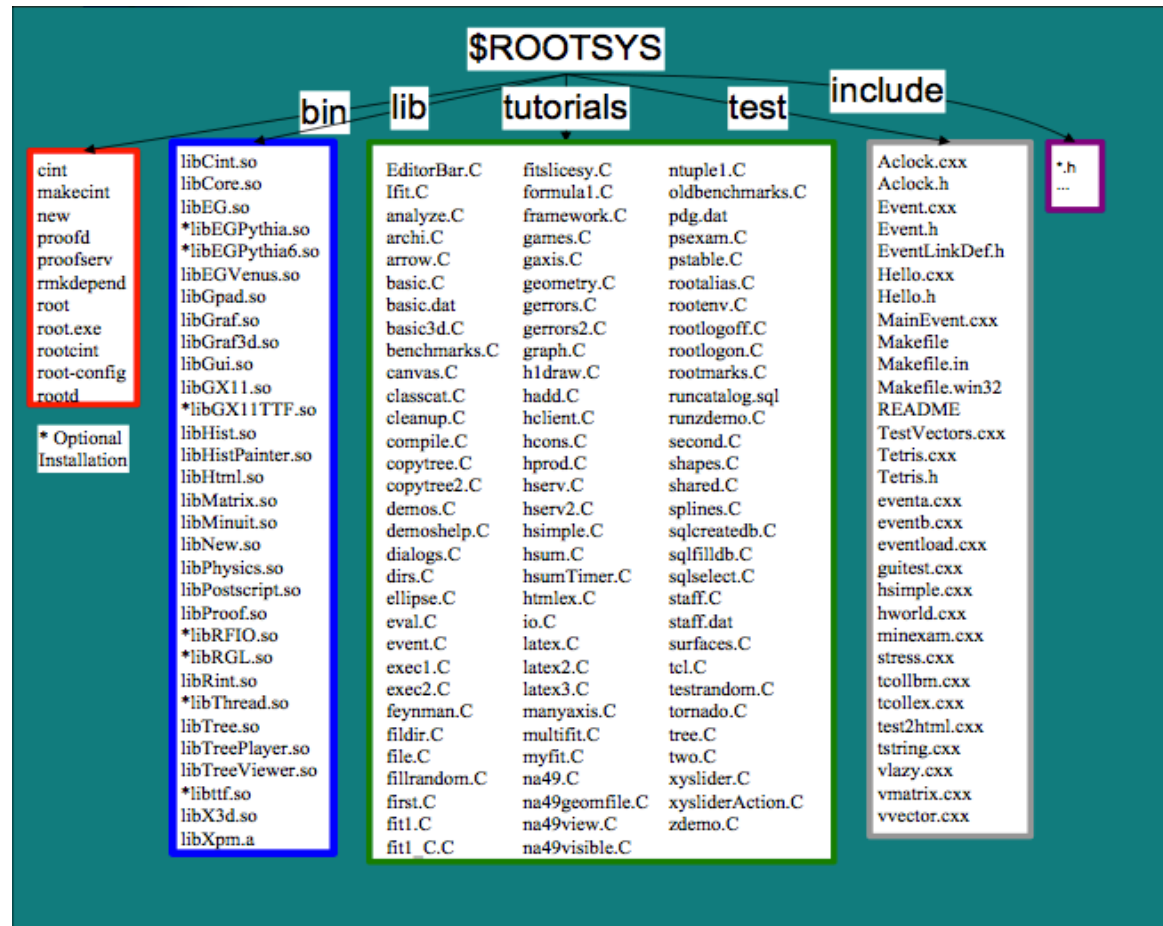
Thanks to: N. Di Marco, S. Panacek and A. Tramontana

What are the capabilities of ROOT?

- Histograms and fit
- Graphic and scatter plot (2D, 3D)
- **I/O on file**
 - Specialized for histograms and **Ntuple** (TTrees)
- Support for data analysis
- User interface
 - GUI: Browsers, Panels, Tree Viewer
 - Command line interface: C++ interpreter (CINT) → upgrade to **cling** in ROOT 6
- Processor for **scripts** (**compiled** C++ \Leftrightarrow **interpreted** C++)
- Possibility to use ROOT classes in **external programs**

The framework organization

- Everything is controlled from the **\$ROOTSYS** environment variable
- Directories for *binaries*, *includes* and *compiled libraries*
- Specific tutorials for different user applications





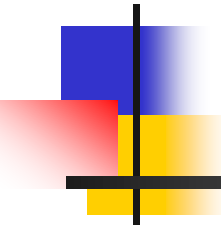
ROOT 5 vs. ROOT 6

- **Major change** between ROOT 5.x series and ROOT 6.x series
- ROOT 6 features
 - Makes use of the new standard **C++11** of C++ (requires gcc > 4.8)
 - Additional features coming from C++11
 - Old C++ interpreter (CINT) replaced by an **integrated compiler (Cling)** which can work also as an interpreter
 - More powerful and flexible
 - Better support of **templated classes**
 - Can support **python**
- ROOT 5.x series still **maintained** and **supported** (patches of root 5.34)



ROOT 5 vs. ROOT 6

- These lectures are designed and meant to cover **ROOT 5**
 - ROOT 5.32 installed on the virtual machine
- **All exercises** and examples of this course will (should) **work in ROOT 6** as-they-are
 - The difference is visible only for **advanced tasks**
- A few "**deprecated**" **features** of CINT will not work any longer with cling
 - E.g. `->` and `.` **cannot** be used interchangeably
 - `->` for pointers and `.` for objects, as in proper C++
 - Enforce **proper usage** of C++
 - More appropriate **management** of **memory**



Installation (quick overview)

ROOT installation (location independent)



- Step 0: download
- Step 1: unpack the tarball and enter directory
 - `tar -xzf root_v5.34.28.source.tar.gz`
 - `cd root_v5.34.28`
- Step 2: configure
 - `./configure --enable-xxxx --enable-yyyy`
 - Many additional packages available which can be enabled/disabled by command line
 - Some of them require external libraries
 - See `./configure --help` for documentation

ROOT installation (location independent)

- Step 3: compile
 - **make**
 - (will take a while)
- Step 4: Add **bin/** to **PATH** and **lib/** to **(DY)LD_LIBRARY_PATH**.
 - **. bin/thisroot.sh** (sh family)
 - **source bin/thisroot.csh** (csh family)
 - Can add the sourcing of the script in your **profile**, so that it is executed at **every login**
- ROOT can also be build using **cmake**
 - Not shown in detail here, but **cmake/make/make install**

ROOT installation (location dependent)

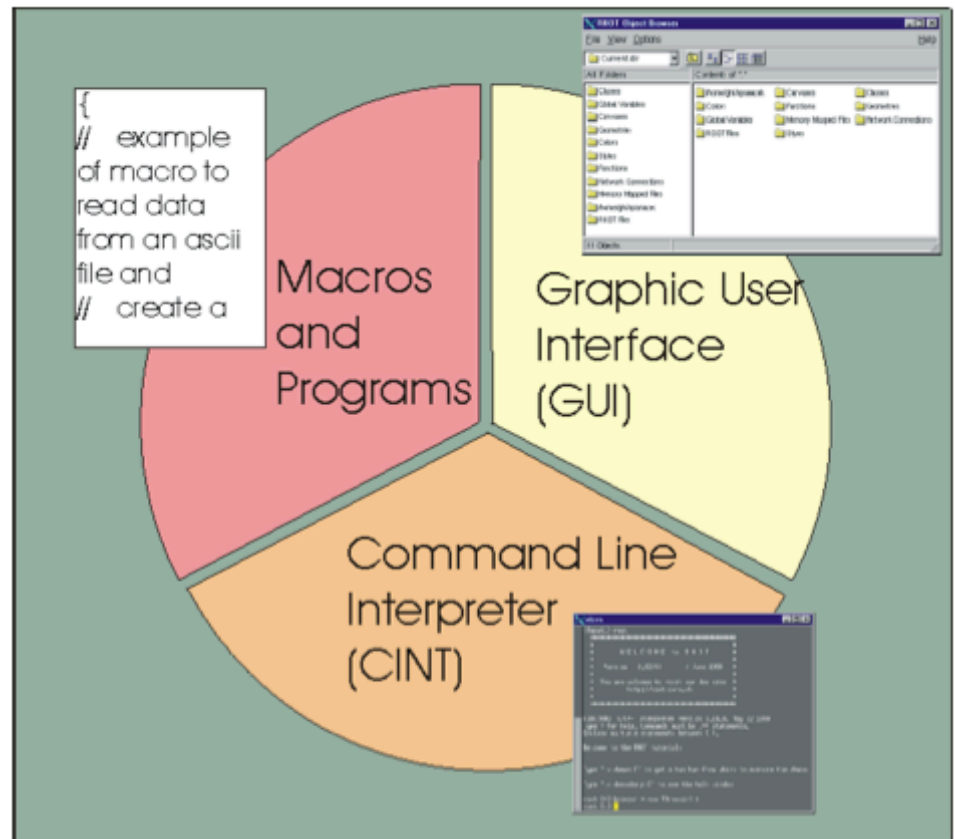
- Copies all binaries and libraries in a **given location** (typically /usr/local, but it can be specified)
 - Convenient if you are **system administrator**
 - Likely get PATH and LD_LIBRARY_PATH for free
- Step 2 (v2) : configuration
 - `./configure -prefix=/usr/local --enable-xxxx --enable-yyyy`
- Step 3 (v2) : compilation and installation
 - `make`
 - `make install` (copies things into the target dir)
 - `ldconfig`
 - (no need to source the script)



Basic and commands

How can we talk to ROOT?

- Interactive **GUI**
 - buttons, graphic menus, etc.
- ROOT interactive **command line** through CINT/cling (C++ interpreter)
- **Macro**, external **applications** with ROOT classes, libraries (C++ compiler)
 - buttons, graphic menus, etc.





What can we do with GUI?

- ROOT files **search** and **opening**
- **Histogram** drawing
- **Contextual menus** with 3 mouse clicks
- **Draw** panel:
 - Parameters choice, colors, etc.
- **Fit** panel:
 - Parameters choice, limits, etc.
- Add **text**, **legend** or other objects
- Window **separation** (**TCanvas**)
 - Selection of lin/log scales

ROOT: command line and GUI

Enter in ROOT

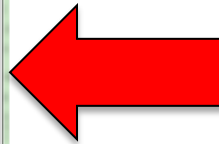
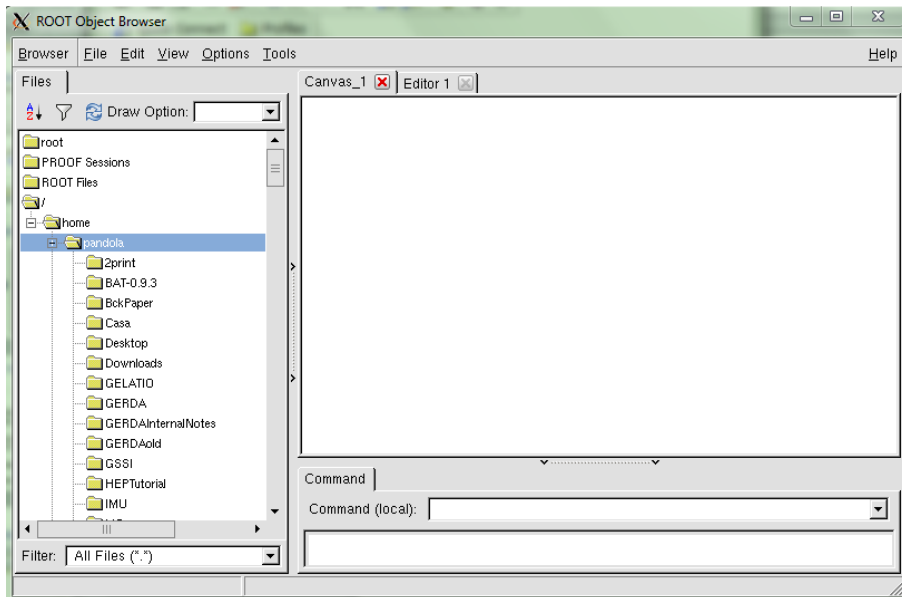
> **root**

Exit from ROOT

root [0] .q



```
*****  
*                                     *  
*      W E L C O M E  t o  R O O T      *  
*                                     *  
*   Version   5.34/03   27 October 2012   *  
*                                     *  
*   You are welcome to visit our Web site *  
*           http://root.cern.ch          *  
*                                     *  
*****  
  
ROOT 5.34/03 (tags/v5-34-03@46856, Oct 27 2012, 23:19:27 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]
```



Browser opening
TBrowser b;



GUI: basic navigation by clicking

- Left Click
 - select the object
 - drag the object
 - resize the object
- Right Click
 - context menu
 - class::name
 - methods
- Middle Click
 - activate canvas
 - Freezes event status bar



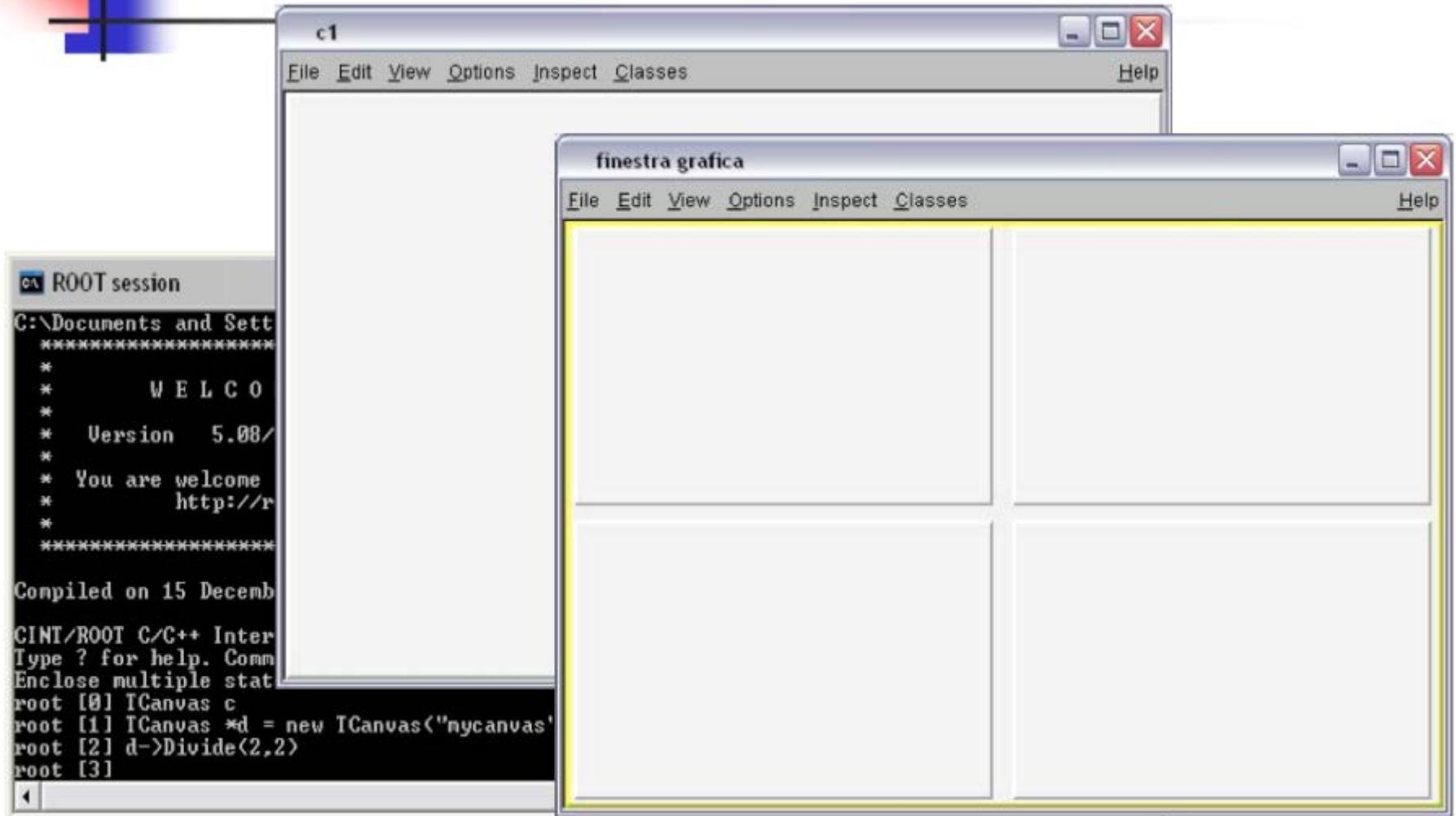
TH1 F::htemp
DrawPanel
Fit
FitPanel
SetMaximum
SetMinimum
SetName
SetTitle
Delete
DrawClass
DrawClone
Dump
Inspect
SetDrawOption
SetLineAttributes
SetFillAttributes
SetMarkerAttributes



GUI / interactive / macros: when and what

- GUI
 - quick tasks using "standard" features
 - fine-tuning of plots, adjustments, labels, etc.
- Command line
 - quick "real-life" analysis, check outputs
 - Retrieve numbers (e.g. counts in a histogram)
- Macro
 - All real-life analysis tasks will required dedicated (compiled) macros
 - Full control and flexibility for input and output

Basic elements of ROOT





Three types of command

1. **CINT/cling** commands start with "."

```
root[0] .?
```

- List of all CINT commands

```
root[1] .x [filename]
```

- Loads [filename] and runs the function [filename] (same name of the file)

```
root[2] .L [filename]
```

- Load [filename]

2. **Shell** commands start with "!", ex.

```
root[3] .! ls
```



Three types of command

3. (Kind of) **C++-like syntax**

```
root [0] TBrowser *b = new TBrowser()
```

or

```
root [0] TBrowser *b = new TBrowser();
```

“:” is optional:

If it is non given, ROOT shows the **return value** (if any) of the command :

```
root [0] 23+5 // shows the return value
```

```
(int)28
```

```
root [1] 23+5; // no return value
```

```
root [2]
```

Getting help from the command line

- Start writing and **press TAB** for auto-completion

- **Classes**

- [] **TCa** [TAB]
Completed to **TCanvas**

- **List of methods**

- [] **TH1D::GetMe** [TAB]
 - **GetMean()**
 - **GetMeanError()**

- **Parameters and return types** of methods

- [] **TH1D* h1 = new TH1D(** [TAB]
 - **TH1D TH1D(const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)**
 - **TH1D TH1D(const char* name, const char* title, Int_t nbinsx, const Float_t* xbins)**
 - ...



Functions and histograms



1-dim functions (class TF1)

- **TF1** -> 1D function $f(x)$
 - Each function is identified by a **name** (string)
 - There are some **predefined functions**
 - 'gaus', 'expo', 'pol0', ..., 'polN'
 - Already known by the command line
- **Custom user functions** can be defined
 - Starting from existing functions, ex. '**gaus+expo**'
 - Writing **explicit formulas** in text format, ex. 'sin(x)/x'
 - For more complicated functions, writing a **function $f(x)$** from a given **x** value (and free parameters)
- Classes TF2 and TF3 for **2- and 3-dim functions**

Define and plot a function

name formula range

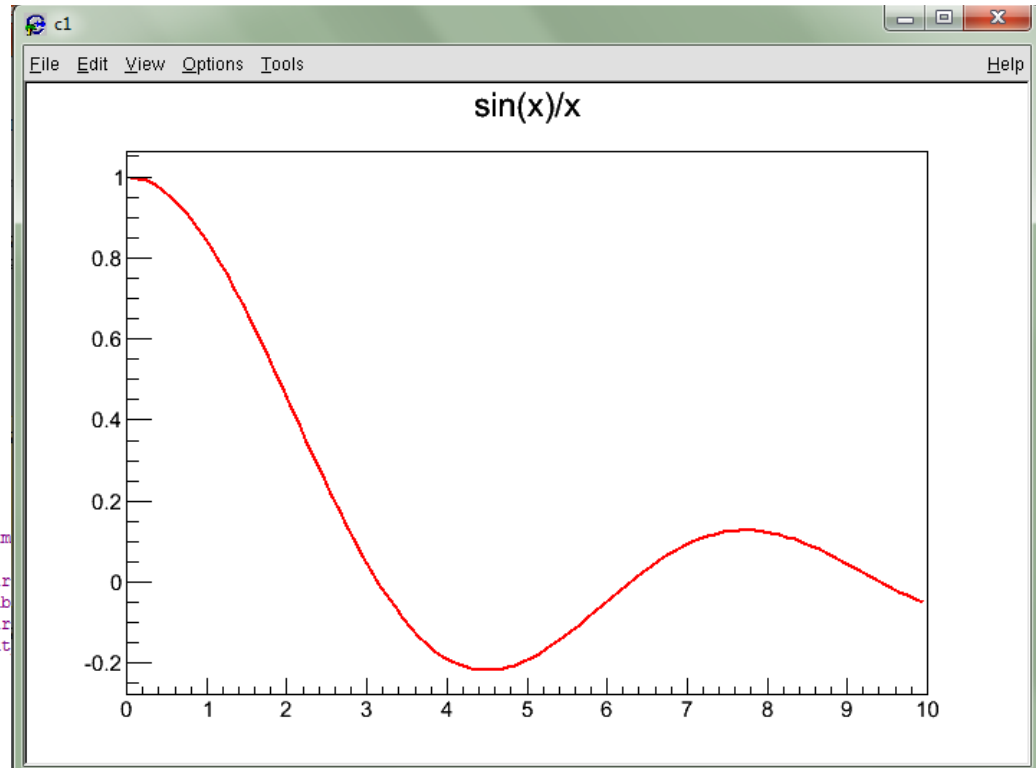
```
[ ] TF1 f1("func1", "sin(x)/x", 0, 10)
```

```
[ ] f1.Draw()
```

```
*    Version    5.34/03    27 October 2012    *
*                                                                 *
*    You are welcome to visit our Web site    *
*                 http://root.cern.ch         *
*                                                                 *
*****
ROOT 5.34/03 (tags/v5-34-03@46856, Oct 27 2012, 23:19:27 on linuxx8664gcc)

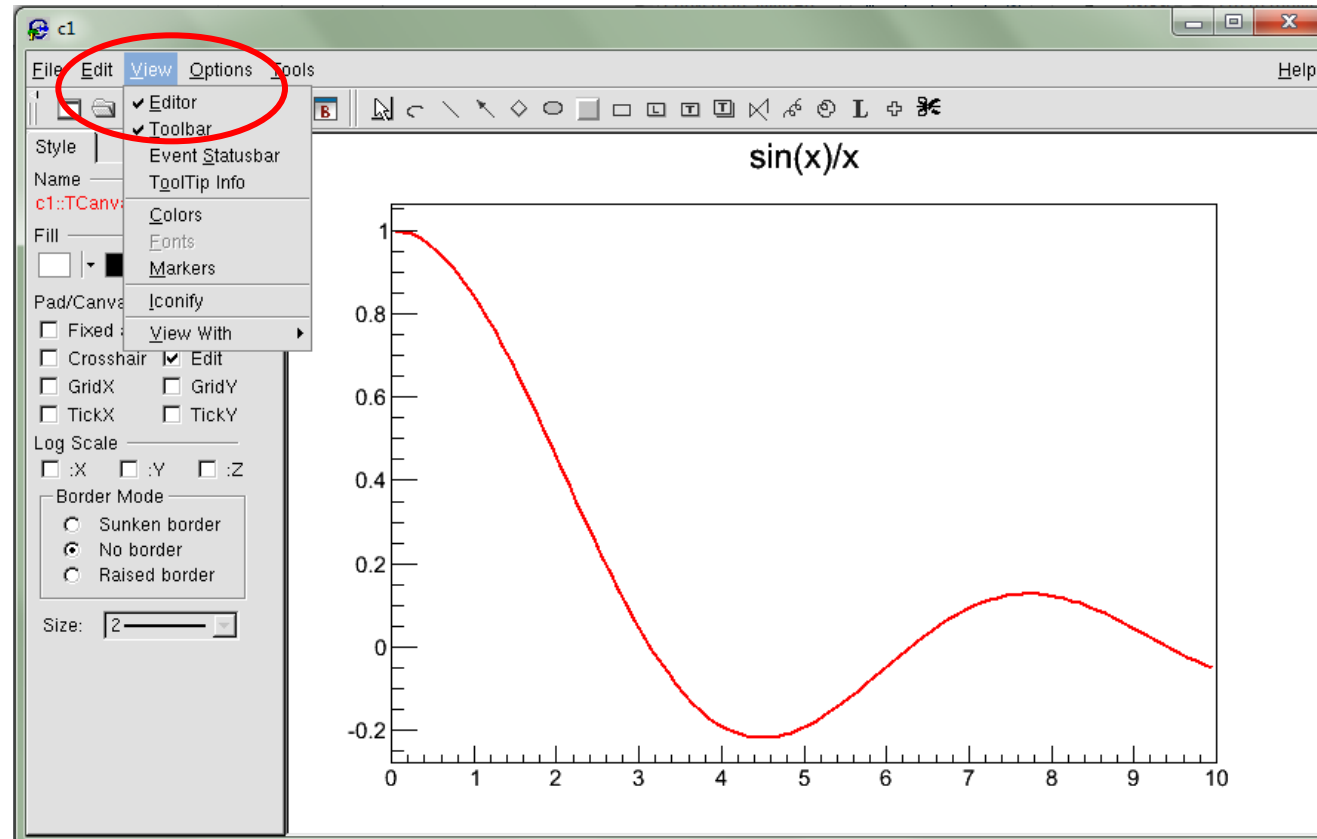
CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] TF1 f1(

TF1 TF1()
TF1 TF1(const char* name, const char* formula, Double_t xmin = 0, Double_t xm
TF1 TF1(const char* name, Double_t xmin, Double_t xmax, Int_t npar)
TF1 TF1(const char* name, void* fcn, Double_t xmin, Double_t xmax, Int_t npar
TF1 TF1(const char* name, ROOT::Math::ParamFuncor f, Double_t xmin = 0, Doub
TF1 TF1(const char* name, void* ptr, Double_t xmin, Double_t xmax, Int_t npar
TF1 TF1(const char* name, void* ptr, void*, Double_t xmin, Double_t xmax, Int
TF1 TF1(const TF1& f1)
root [0] TF1 f1("func1", "sin(x)/x", 0, 10);
root [1] f1.Draw()
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [2]
```



Adjust an existing plot

- Interactive selection of:
 - Axis scales, titles and labels
 - Colors
 - Line width
 - Style
 - Weight
 - Log scale
 - Grids





Histograms

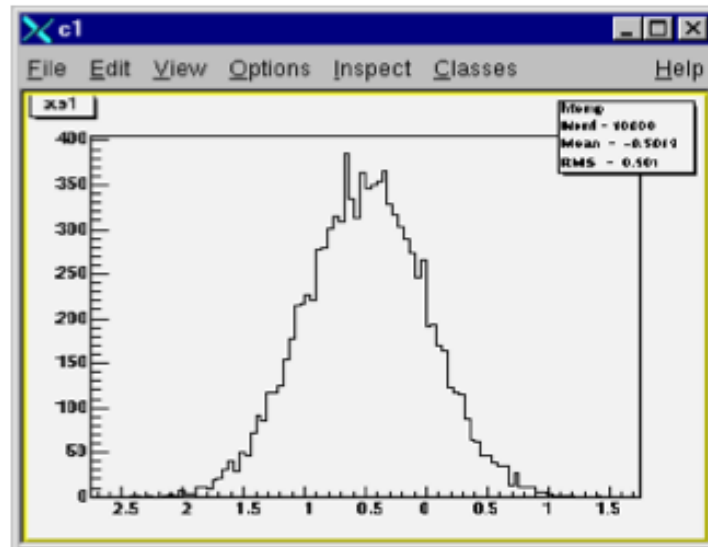
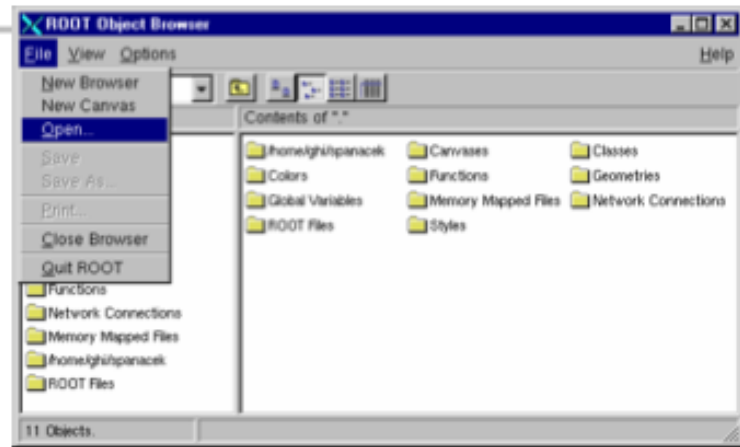
- **TH1D, TH1F, TH1I** -> 1D histograms in ROOT
 - D = double, F = float, I = int
 - **Parameters** are: name (string), title (string), number of bins and range
- TH2* and TH3* for **multi-dimension** histograms
 - Support **advanced plotting**: colour codes, LEGO plots, contour plots, etc
- All histogram classes derive from the **same base interface**, so they have the same basic commands
- Histogram **handling** supported:
 - **operations** between histograms (sum, multiplication)
 - operation between histogram and function
 - **scaling, rebinning, ...**

Plot of an histogram

Open a ROOT file with a **Browser**, search of the histogram from a file

The **double click** on the histogram opens automatically the canvas

Use **Draw()** from the command line



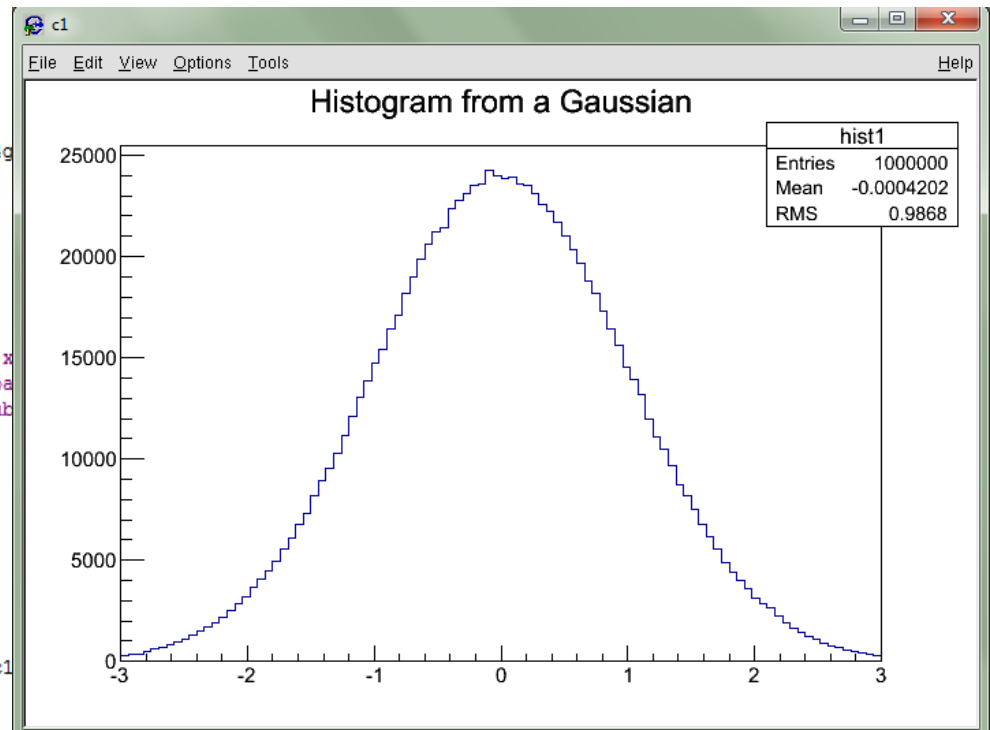
Creation of an histogram

```
name title bins range
[ ] TH1D h1("hist","Histogram from a gaussian",100,-3,3);
[ ] h1.FillRandom("gaus",100000);
[ ] h1.Draw() ← Fill with 105 values from "gaus"
```

```
* You are welcome to visit our Web site *
* http://root.cern.ch *
* *
*****
ROOT 5.34/03 (tags/v5-34-03@46856, Oct 27 2012, 23:19:27 on linuxx8664g)
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] TH1D h1(

TH1D TH1D()
TH1D TH1D(const char* name, const char* title, Int_t nbinsx, Double_t x
TH1D TH1D(const char* name, const char* title, Int_t nbinsx, const Floa
TH1D TH1D(const char* name, const char* title, Int_t nbinsx, const Doub
TH1D TH1D(const TVectorDs v)
TH1D TH1D(const TH1Ds h1d)
root [0] TH1D h1("hist1","Histogram from a Gaussian",100,-3,3);
root [1] h1.FillRandom(

void FillRandom(const char* fname, Int_t ntimes = 5000)
void FillRandom(TH1* h, Int_t ntimes = 5000)
root [1] h1.FillRandom("gaus",1000000)
root [2] h1.Draw()
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [3] █
```



Histograms: how to handle the content



```
TH1D* h2 = new TH1D("hist","Title",100,-3,3);
```

- Add **one value** at time

```
h2->Fill(myvalue);
```

- Set the **content** of each **bin**

```
for (Int_t i=1;i<=h2->GetNbinsX();i++)  
    h2->SetBinContent(i,value[i]);
```

Notice: bin 0 contains the **underflows**, bin N+1 the **overflows**

- **Retrieve** the content of each bin

```
Double_t val[i] = h2->GetBinContent(i);
```

- Total **entries**

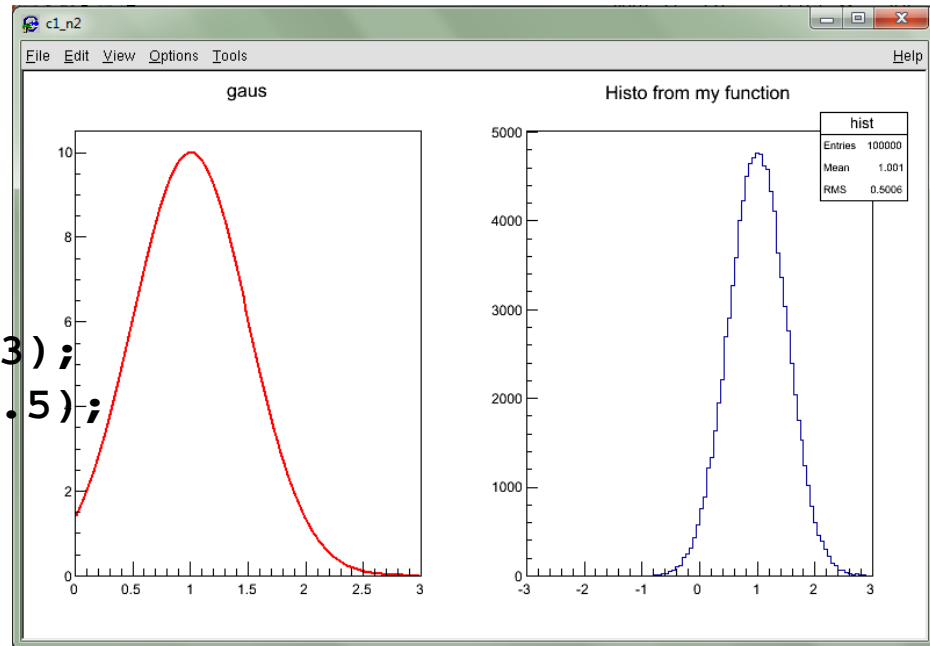
```
Int_t entries = h2->GetEntries();
```

Example

Fill an histogram with 100000 random numbers from the function:

$$par(0)e^{-0.5\left(\frac{x-par(1)}{par(2)}\right)^2}$$

```
[ ] TF1 myfunc("myfunc","gaus",0,3);  
[ ] myfunc.SetParameters(10.,1.,0.5);  
[ ] TCanvas c;  
[ ] c.Divide(2,1);  
[ ] c.cd(1);  
[ ] myfunc.Draw();  
[ ] TH1D h2("hist","Histo from my function",100,-3,3);  
[ ] h2.FillRandom("myfunc",100000);  
[ ] c.cd(2);  
[ ] h2.Draw();
```

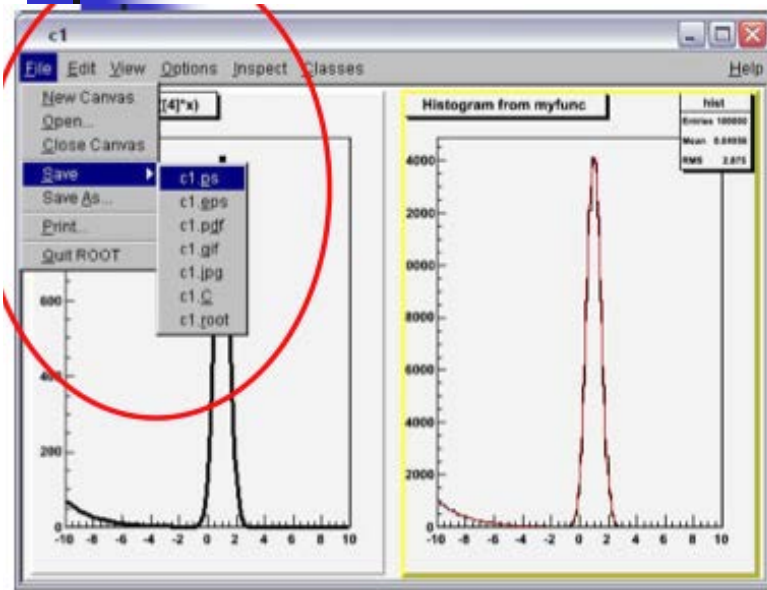




Histogram, some more extras

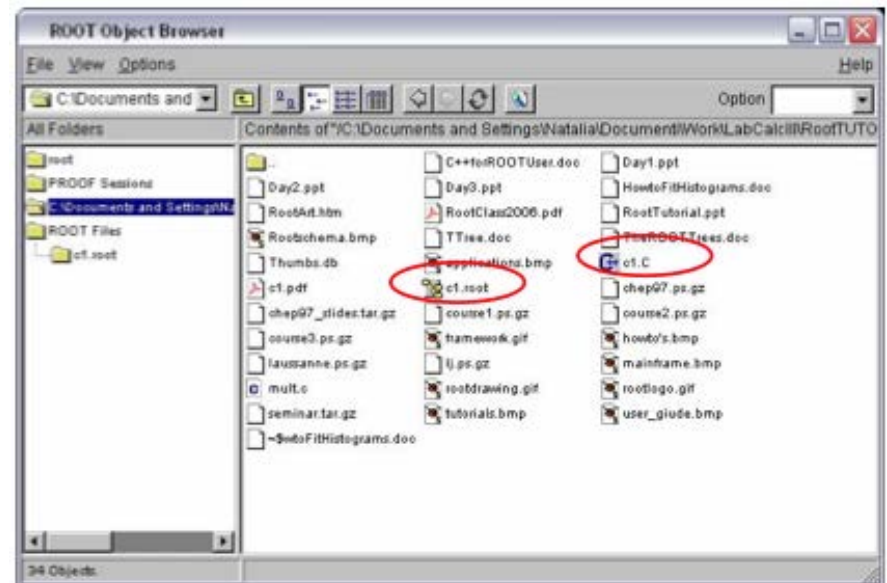
- Draw with **error bars**
 - `h1->Draw("E");`
 - Notice: by default, the error given to each bin is the **square root of its content**. This is **wrong** if the histogram is normalized and/or the y scale is not "counts". Can use `SetBinError(i,error)`
- **Superimpose** in the current Canvas
 - `h1->Draw("same")`
 - Applies also **to all other ROOT objects** (functions, graphs, etc.)
- Change title axis
 - `h1->GetXaxis()->SetTitle("Energy (keV)");`
- Operations on histogram(s)
 - `h1->Add(h2,1);`
 - `h1->Scale(0.01);`
- Most operations can also be **done** with the **GUI**

Save the plot with TBrowser



- 1) As a ROOT script **c1.C**
Retrieve with
`[] .x c1.C`
- 2) As a **graphic file**
pdf, gif, png, ...

- 3) As a ROOT file **c1.root**
It contains the ROOT objects.
Can be retrieved from a user
macro, or interactively by
`[] TBrowser tb;`



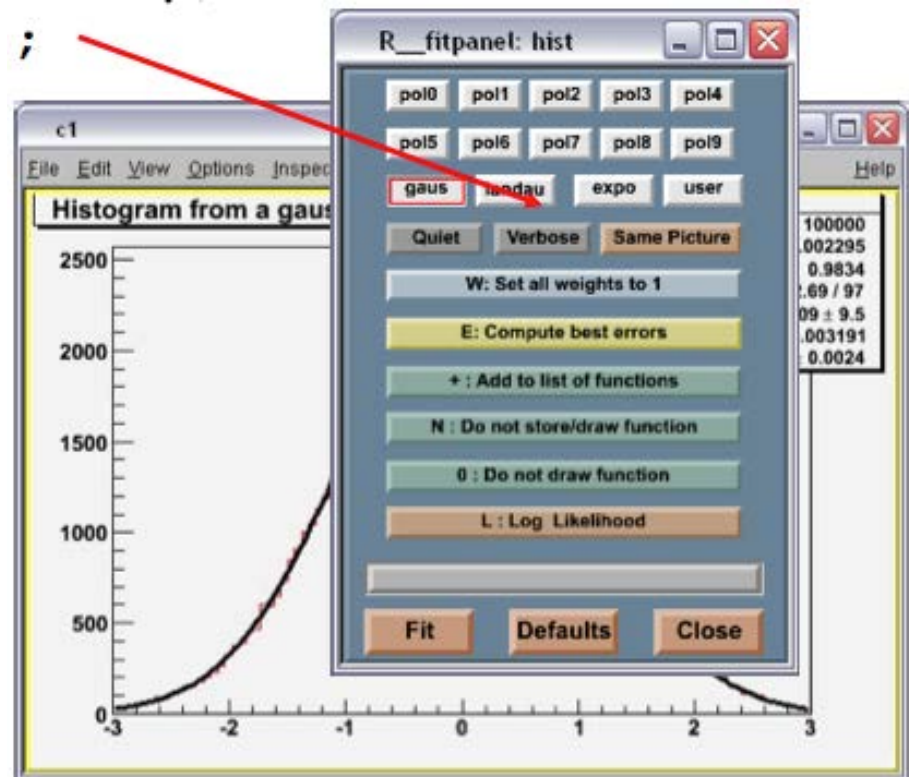
Fit of a histogram - 1

```
[ ] TH1F h1("hist","Histogram from a gaussian",100,-3,3);  
[ ] h1.FillRandom("gaus",100000);  
[ ] gStyle->SetOptFit(111);  
[ ] h1.Fit("gaus");
```

Or

```
[ ] h1.FitPanel();
```

The **FitPanel** allows to choose parameters and functions in an interactive way

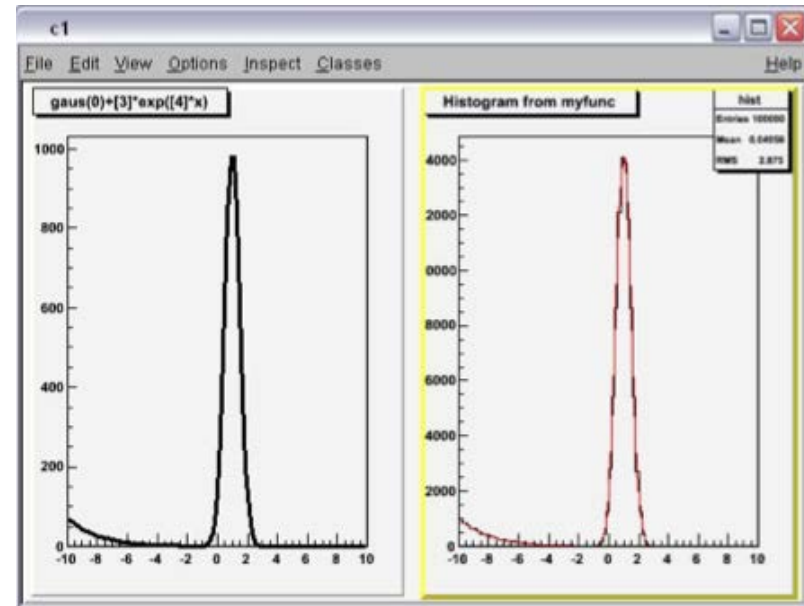


Fit of a histogram - 2

To fit an histogram with : $par(0)e^{-\left(\frac{x-par(1)}{par(2)}\right)^2} + par(3)e^{par(4)x}$

```
[ ] TF1 f1("myfunc","gaus(0)+[3]*exp([4]*x)",-10.,10.);  
[ ] f1.SetParameters(1000.,1.,0.5,0.5,-0.5);  
[ ] TH1F h1("hist","Histogram from myfunc",100,-10,10);  
[ ] h1.FillRandom("myfunc",100000);  
[ ] h1.Fit("myfunc");
```

```
root [10] h1.Fit("myfunc")  
PCN=58.0027 FROM MIGRAD      STATUS=CONVERGED      5  
EDM=1.40487e-008      STRATEGY=  
EXT PARAMETER  
NO.  NAME      VALUE      ERROR  
 1  p0      1.43217e+004  5.83272e+001  
 2  p1      9.98341e-001  1.66154e-003  
 3  p2      4.98132e-001  1.17738e-003  
 4  p3      7.32169e+000  3.23833e-001  
 5  p4      -4.96196e-001  5.34834e-003  
<Int_t>0  
root [11]
```





Fit of a histogram – 3

- Fit on a sub-range

- Define the **range** in the **TF1 constructor**

```
TF1 *g1 = new TF1("g1", "gaus", 85, 95);
```

- By default, TH1::Fit on the defined **histogram range**. Use **"R"** option in the Fit() method.

```
h->Fit("g1", "R");
```

- Can **"merge"** different **functions** in different sub-ranges

- Retrieve fit function and parameters

- Fit function attached to the **histogram**

```
TF1* fun = h->GetFunction("g1");
```

```
cout << fun->GetParameter(0) << " +/- " <<
```

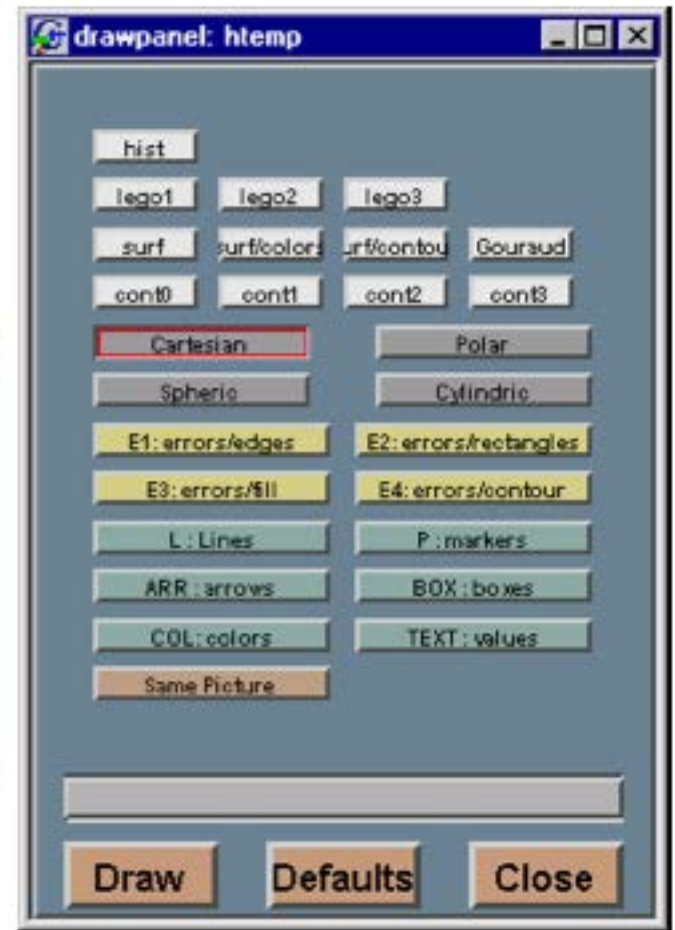
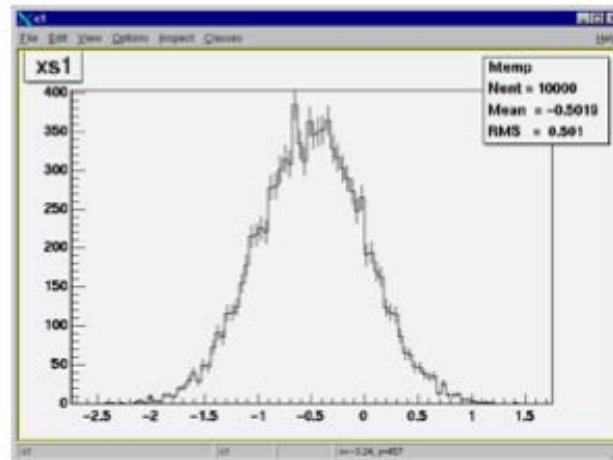
```
    fun->GetParError(0) << endl;
```

```
fun->SetParLimits(1, -10, 10);
```

The DrawPanel()

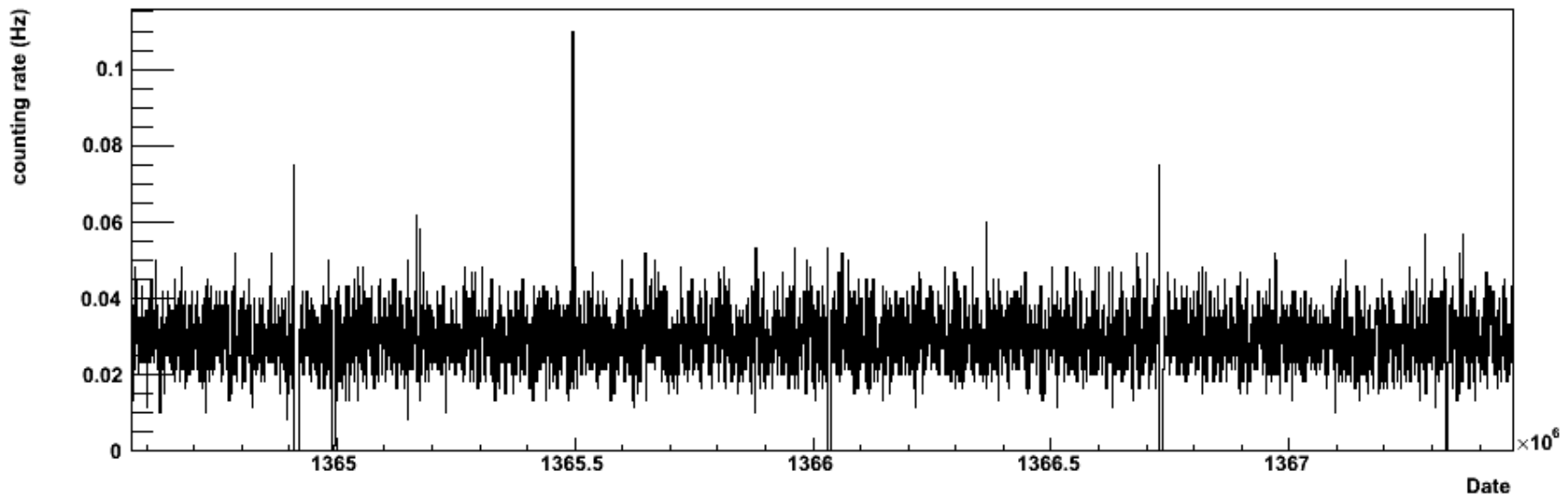
- The interactive panel:
 - `h1 -> DrawPanel();`

It allows to change parameters/type of histogram, coordinates, errors, colors, etc.



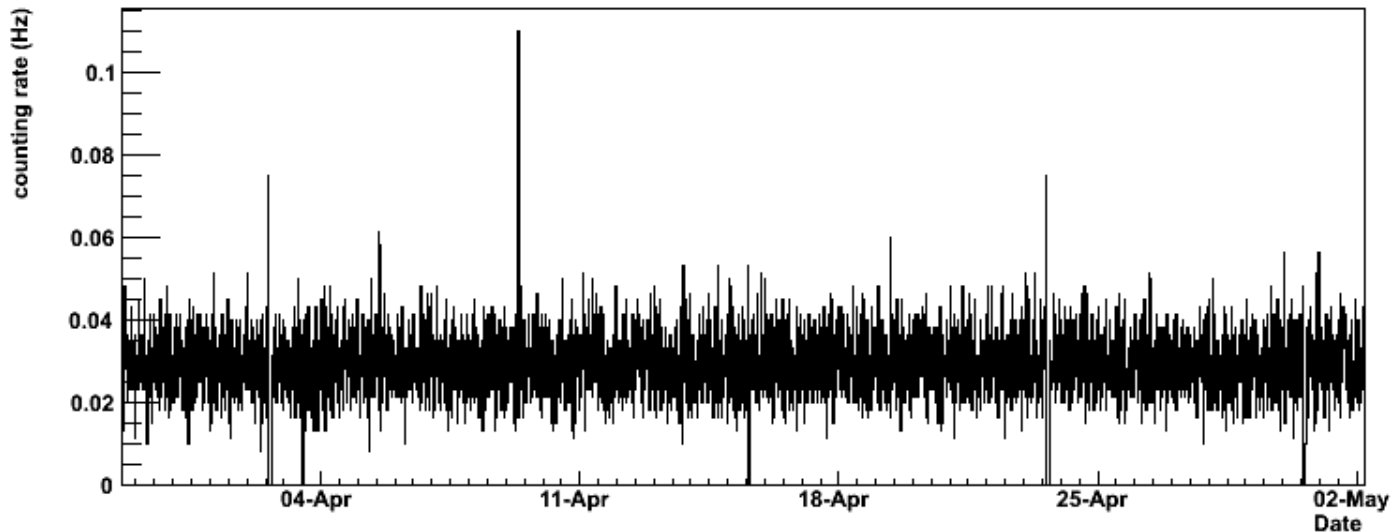
Time axis - 1

- In some cases, the **quantity** of one axis is **date/time**
 - Typical case: **UNIX time stamp** (number of seconds elapsed since 1/1/1970)
 - E.g. Jun 1st 2014 00:00:00 GMT → 1401580800
- Want to have a *more meaningful* x-axis scale!



Time axis - 2

- Use time axis option
 - `h1->GetXaxis()->SetTimeDisplay()`
- Set the **format** and the time **offset**
 - Default: time elapsed since **1/1/1995**
 - `h1->GetXaxis()->SetTimeFormat("%d-%b %F1970-01-01 00:00:00");`





Time axis - 3

- Check the documentation online for the possible formats of date/time
- Same applies for offsets

- for date :
 - %a abbreviated weekday name
 - %b abbreviated month name
 - %d day of the month (01-31)
 - %m month (01-12)
 - %y year without century
 - %Y year with century
- for time :
 - %H hour (24-hour clock)
 - %I hour (12-hour clock)
 - %p local equivalent of AM or PM
 - %M minute (00-59)
 - %S seconds (00-61)
 - %% %

1. By setting the global default time offset:

```
TDatetime da(2003,02,28,12,00,00);  
gStyle->SetTimeOffset(da.Convert());
```

If no time offset is defined for a particular axis, the default time offset will be used. In the example above, notice the usage of [TDatetime](#) to translate an explicit date into the time in seconds required by [SetTimeFormat](#).

2. By setting a time offset to a particular axis:

```
TDatetime dh(2001,09,23,15,00,00);  
h->GetXAxis()->SetTimeOffset(dh.Convert());
```

3. Together with the time format using [SetTimeFormat](#):

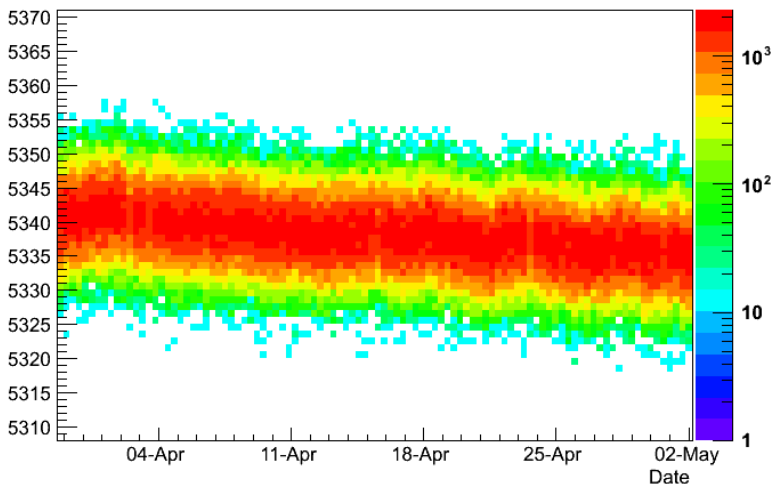
The time offset can be specified using the control character %F after the normal time format. %F is followed by the date in the format: yyyy-mm-dd hh:mm:ss.

Example:

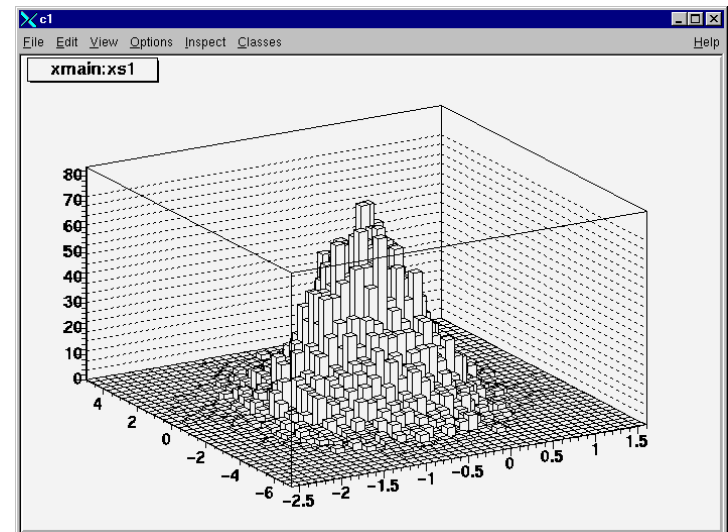
```
h->GetXAxis()->SetTimeFormat("%d\/%m\/%y%F2000-02-28 13:00:01");
```

2-dimensional histograms

- Class TH2* (TH2D, TH2F, ...)
 - Very **same methods** inherited from the basic TH1 interface (plots, fits, etc.)
 - **Extra options** for plotting (contour, lego, colour code, ...)
 - Have a **z-axis** now!
 - `c1->SetLogz()`



Rotate with the mouse





Histogram drawing options

- Check the **THistPainter** documentation for all available options
- Most options also available from the **GUI**

Options supported for 2D histograms

" "	Default (scatter plot).
"ARR"	Arrow mode. Shows gradient between adjacent cells.
"BOX"	A box is drawn for each cell with surface proportional to the content's absolute value. A negative content is marked with a X.
"BOX1"	A button is drawn for each cell with surface proportional to content's absolute value. A sunken button is drawn for negative values a raised one for positive.
"COL"	A box is drawn for each cell with a color scale varying with contents. All the none empty bins are painted. Empty bins are not painted unless some bins have a negative content because in that case the null bins might be not empty. TProfile2D histograms are handled differently because, for this type of 2D histograms, it is possible to know if an empty bin has been filled or not. So even if all the bins' contents are positive some empty bins might be painted. And vice versa, if some bins have a negative content some empty bins might be not painted.
"COLZ"	Same as "COL". In addition the color palette is also drawn.
"CANDLE"	Draw a candle plot along X axis.
"CANDLEX"	Same as "CANDLE".
"CANDLEY"	Draw a candle plot along Y axis.
"CONT"	Draw a contour plot (same as CONTO).
"CONTO"	Draw a contour plot using surface colors to distinguish contours.
"CONT1"	Draw a contour plot using line styles to distinguish contours.
"CONT2"	Draw a contour plot using the same line style for all contours.

Random generators and math libraries





TMMath

- Large library of **mathematical tools**
 - Numerical **constants**
 - $\pi = \text{TMMath}::\text{Pi}()$, $1/\pi = \text{TMMath}::\text{InvPi}()$, ...
 - **Trigonometric** and elementary math functions
 - Sin, cos, log(s), power, asin, ...
 - Functions and **tools** to work with **arrays** and **collections**
 - Sort, binary search, max, min, ...
 - **Statistical** and special functions
 - Breit-Wigner, Bessel, KolmogorovProb, χ^2 quantiles
- Can be used by **command line** and in **scripts**

```
TF1
```

```
fun1("f1", "TMMath::ASin(x)*TMMath::Log10(x)",  
1,10);
```

TRandom - 1

- Random number generators **embedded** in ROOT
 - Can be used by **command line** and (mostly) compiled/interpreted **scripts**
- A **few generators available**, differing for the "quality" of the random numbers and for the CPU time

Not to be used for real statistical studies

TRandom	34	ns/call	(BAD Generator)
TRandom1	242	ns/call	CERN RANLUX
TRandom2	37	ns/call	
TRandom3	45	ns/call	Mersenne Twister generator



TRandom - 2

- A few **commonly-used** distributions provided

```
TRandom3 rd;  
Double_t val = rd.Exp(tau);  
Int_t n = rd.Poisson(mean);  
    ■ Integer(imax), Gaus(mean,sigma), Rndm(),  
      Uniform(), Landau(mpv,sigma),  
      Binomial(ntot,prob)  
    ■ Able to handle/change random seed
```
- Can generate **random numbers** according to a given **histogram** or **function**

```
TF1 f1("f1","abs(sin(x))*sqrt(x)",0,10);  
Double_t val = f1.GetRandom()
```