Statistical Methods and Analysis
Techniques in Experimental Physics
ETHZ/UNIZH, FS09

# Introduction to ROOT

Andrea Rizzi, rizzi@phys.ethz.ch
HPK/F-28

# Outline

- What is ROOT
- ROOT interactive console
- Important C++ remarks
- Reading data with ROOT
- Histograms
- Style, options, legend, canvas

# What is ROOT?

- ROOT is an object oriented framework for data analysis
  - read data from some source
  - write data (persistent objects)
  - selected data with some criteria
  - produce results as plots, numbers, fits, ...
- Supports “interactive” (C/C++ like, Python) and “compiled” (C++) usage
- Integrates several tools like random number generations, fit methods (Minuit), Neural Network framework
- Developed and supported by High Energy Ph. community
  - homepage with documentation and tutorials: root.cern.ch

# ROOT interactive console

- Prepare your shell environment

```
sh# export ROOTSYS=/path/to/root/installation
sh# export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
sh# export PATH=$ROOTSYS/bin:$PATH
```

- Launch ROOT interactive console (CINT interpreter)

```
sh# root
   ****************************************
   *                                      *
   *        W E L C O M E  to  R O O T    *
   *                                      *
   *   Version   5.10/00      1 March 2006  *
   *                                      *
   *  You are welcome to visit our Web site  *
   *          http://root.cern.ch         *
   *                                      *
   ****************************************

FreeType Engine v2.1.9 used to render TrueType fonts.
Compiled on 2 March 2006 for linux with thread support.

CINT/ROOT C/C++ Interpreter version 5.16.8, February 9,
2006
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

# ROOT interactive console

- First, how to quit? type **.q**
- Some useful commands
- Some useful tips
- Some names:
  - **CINT**: is the C/C++ interpreter of ROOT. C++ is not meant to be an interpreted language, so CINT has some limitations!
  - **Aclic**: ROOT C/C++ compiler, invoked when you ask ROOT to compile something

Load code from external file
**.L fileName.C**
Load code and *execute* myfunction()
**.x myfunction.C**
you can append "**++**" to the filename to have code compiled

•You can use "TAB" key to complete names in ROOT or to get help about the argument of a funcion

```
root[0] TH1F histo(
```

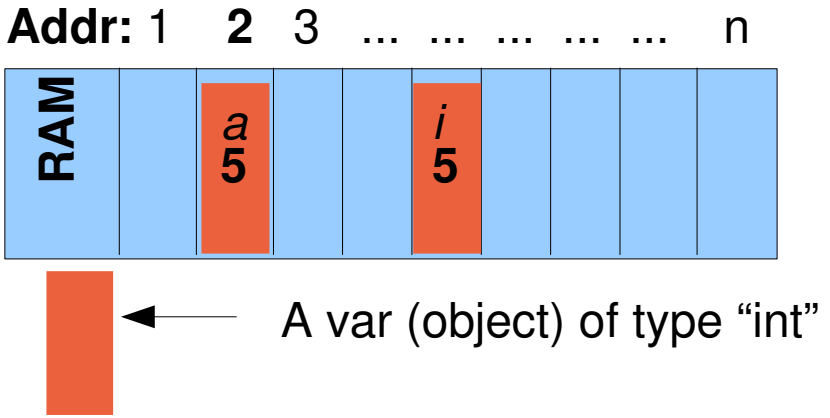
**TAB**

```
TH1F TH1F()
TH1F TH1F(const char* name,...
TH1F TH1F(const char* name,...
TH1F TH1F(const char* name,...
TH1F TH1F(const TVectorF& v)
TH1F TH1F(const TH1F& h1f)
```

•The history of your recent commands is kept in a file ~/.root_hist
**#sh** cat ~/.root_hist

# Some C++ (Pointers, references, instances)

- Pointers are variables that knows where another variable is stored in RAM




```
int a = 5;
int i = a;   // create another object and copy “a”
int * b = &a; //give to b the address of a
int & r = a;  // r is a reference to a
const & c = a;  // c is a const reference to a
```

```
cout << "a value is : " << a << endl;      //  b value is : 5
cout << "b value is : " << b << endl;      //  b  value is : 2
cout << "value pointed by b : " << *b << endl; // value pointed by b : 5

//references are not copies, they refer to the same address
r=6;
cout << a << endl;    // 6
cout << i << endl;    // 5
cout << &r << endl;   // 2

//constant reference cannot be modified (are often used to pass objects
// to functions that should not modify the passed object
cout << c << endl;    // 6
c=7; //this doesn't compile!
```

- the operator * return the value pointed (of a ptr)
- the operator **&** return the pointer of a variable

# Allocation, scope of objects

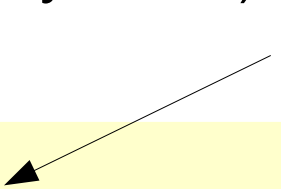
- new objects can be created in two ways
  - objects created by the user with “new” should be deleted by the user with “delete”
  - objects declared in a block are deleted automatically when they go out of scope

```
{
My2DPoint  a(3.12,2.22);
My2DPoint * b = new MyObject(3.12,2.22);
} // here “a” is deleted, b is not deleted (up to you!)
```

- two common problems
  - memory leaks when “b” are not deleted
  - invalid pointers when the address of “a” is taken
    - My2DPoint * c = &a;  (cannot be used after a is deleted)

# ROOT vs C++ memory management

- ROOT objects (Histograms, Canvas, ...) are managed in memory (and disk) by root using “names”
- ROOT define a hierarchical structure of directories
- In the same directory you cannot have two objects with the same name (ROOT will complain about memory leaks)
- ROOT does not like the following:




```
TH1F * histos[10];
for(int i = 0; i < 10 ; i++) histos[i]= new TH1F(“hist”,”hist”,1,2,3);
```

- Interactive ROOT fixes for you wrong usage of pointer vs reference (but when you compile you MUST use correct syntax)
  - objects member functions can be accessed with “.” (for instances and reference ) or “->” (for pointers) root “understand” both: histogram->GetMean(); or histogram.GetMean();

# Standard Template Library

- Recent version of ROOT also support STD containers, e.g.
  - std::vector<double> , std::vector<MyObject>
  - std::map<std::string, double>
  - ....
- std::string can be used but should be converted to “C string” when ROOT needs a “const char *”

```
std::string histogramName;
histogramName = prefix+"_EnergyHistogram";
TH1F his(histogramName.c_str(),"Title",10,1,10);
```

# Reading data

- ROOT can read data from different sources such as files, network, databases
- In ROOT framework the data is usually stored in **TTree** (or the simplified version TNtuple)
  - Trees/Ntuples are like “tables”, each raw represent usually an “event”, each column is a given quantity
  - Single cells can also be “complex” objects instead of simple numbers
- Ntuple and Trees can be read from “ROOT files” in which they are stored, can be created and filled from an ASCII file, can be created and saved by the user

# Reading from ASCII file

- Ex: text file with 3 columns space separated
- We can create an “NTuple” with three columns and read it

```
sh# head -n4 calls.txt
#cost   time     type
1.46    127      2
2.25    124      11
0.82    71       1
```

```
root [0] TNtuple calls("calls","calls","cost:time:type")
root [1] calls.ReadFile("calls.txt")
(Long64_t)192
```

**number of row read**

**name and title**

**Declaration of columns**

The list of variables to print can be specified in the parenthesis as “var1:var2:var3...”

```
root [2] calls->Scan()
************************************
*    Row   *       cost *       time *       type *
************************************
*        0 * 1.4600000 *        127 *          2 *
*        1 *      2.25 *        124 *         11 *
*        2 * 0.8199999 *         71 *          1 *
```

# Saving/reading ROOT file

- We can save the TNtuple in a file

```
root[2] TFile f("rootfile.root","CREATE")
root[3] f.cd()
root[4] calls.Write()
root[5] f.Close()
```

- And read it back from a new ROOT console

```
root[0] TFile f("rootfile.root")
root[1] TNtuple * calls = f.Get("calls")
```

- When you read back, the pointer to the NTuple is *owned* by root, you should not delete it
- the “Get” method identify the objects with their “name”
- you can list the name and type of objects in a file

```
root [2] f.ls()
TFile**          rootfile.root
 TFile*          rootfile.root
  KEY: TNtuple   calls;1 calls
```

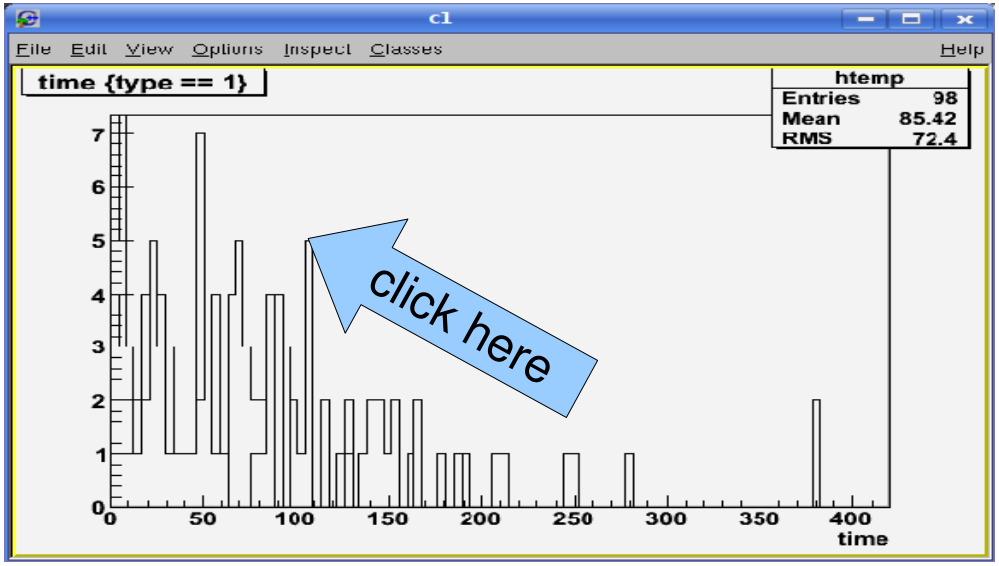
# TTree/TNtuple drawing

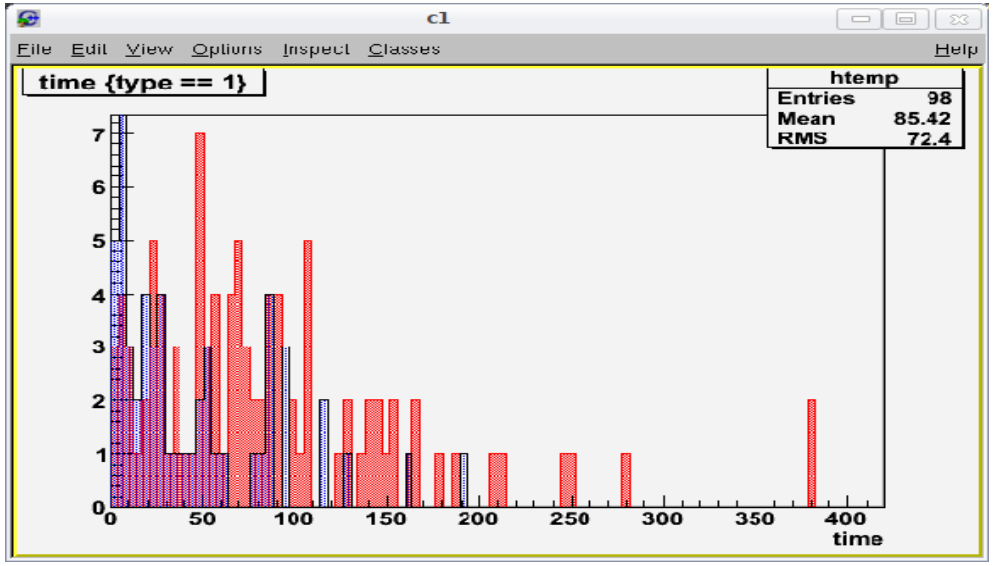
- You can make an histogram of the distribution of a variable in a TTree

```
root[5] calls->Draw("time")
root[6] calls->Draw("time","type == 1")
root[7] calls->Draw("time","type == 2","same")
root[8] calls->Draw(
```

**TAB**

- Variable to plot
- Cut to apply
- Options for drawing
- To know more...




- Properties of drawn objects can be changed with right click on the object (Right click on the **top of a bin** of an histogram and chose SetFillAttributes)

# Booking histograms

- It is possible (and is better!) to (*user)* define histograms: dimension, axis range, number of bins, name, title

**Dimension**

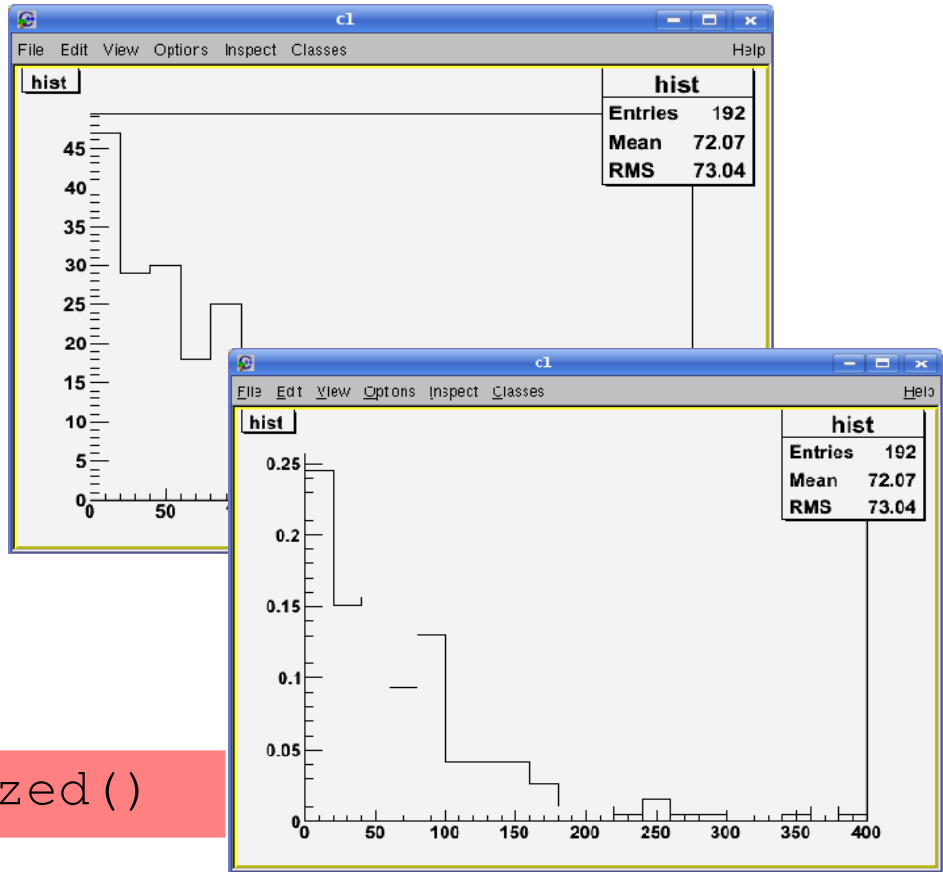
- Histgrams objects are called TH1F, TH2F,TH3F, TH1D,...

**Float/Double**

- To create a new histogram with 20 bins, in range [0,400]:

```
root[2] TH1F hist("hist","hist",20,0,400);
root[3] calls->Draw("time>>hist")
```

- Now we can do a lot of things on the histogram (changing properties, fitting, asking integrals, value of a bin, overflow, underflow, scaling, drawing normalized,.....)
- More info: google "TH1F" or http://root.cern.ch/root/html512/TH1F.html




```
root[4] hist.DrawNormalized()
```

# Some histogram properties

- Accessing histogram information:

| | |
|---|---|
| `hist.GetMean();` | mean |
| `hist.GetRMS();` | root of variance |
| `hist.GetMaximum();` | maximum bin content |
| `hist.GetMaximumBin(int bin_number);` | location of max. |
| `hist.GetBinCenter(int bin_number);` | center of bin |
| `hist.GetBinLowEdge(int bin_number);` | lower edge of bin |
| `hist.GetBinContent(int bin_number);` | content of bin |

- Bin 0 is the underflow bin
- Bin 1 the first (visible) bin
- Bin n+1 is the overflow bin

**Color/Fill/Style:**

SetLineColor() / SetLineStyle()
http://root.cern.ch/root/html512/TAttLine.html
SetMarkerColor() / SetMarkerStyle()
http://root.cern.ch/root/html512/TAttMarker.html
SetFillColor() / SetFillStyle()
http://root.cern.ch/root/html512/TAttFill.html

# Manual filling of histograms

- We have already seen how to fill an histogram from Ttree/TNtuple::Draw (using “>>histoname”)
- An histogram can be filled by calling TH1F::Fill function

```
//the syntax for 1D histo is: hist.Fill(value, weight)
root[2] for(int i=0;i < 10;i++) hist.Fill(i);
```

- Fill() function can be useful if in your program/macro you do “by hand” the loop on the events:

```
TFile f("rootfile.root")
TNtuple* calls = f->Get("calls");
TH1F hist("hist","hist",20,0,10);
.L loop.C
loop(calls, &hist)
hist->Draw()
```

load the file loop.C

**loop.C**

```
loop(TNtuple * nt,TH1F * histo) {
 Float_t time,cost,type;
 nt->SetBranchAddress("time", &time);
 nt->SetBranchAddress("cost", &cost);
 nt->SetBranchAddress("type", &type);

Int_t nevent = nt->GetEntries();

 for (Int_t i=0;i<nevent;i++) {
  nt->GetEntry(i);
  if(type == 1)
     histo->Fill(cost,2.); //weight 2
    else
     histo->Fill(cost,1.); //weight 1
 }
}
```

# Canvas, style, options

- If no Canvas is available ROOT create one when you “draw”
- Canvas can be created with:

```
root[0] c1 = new TCanvas
```

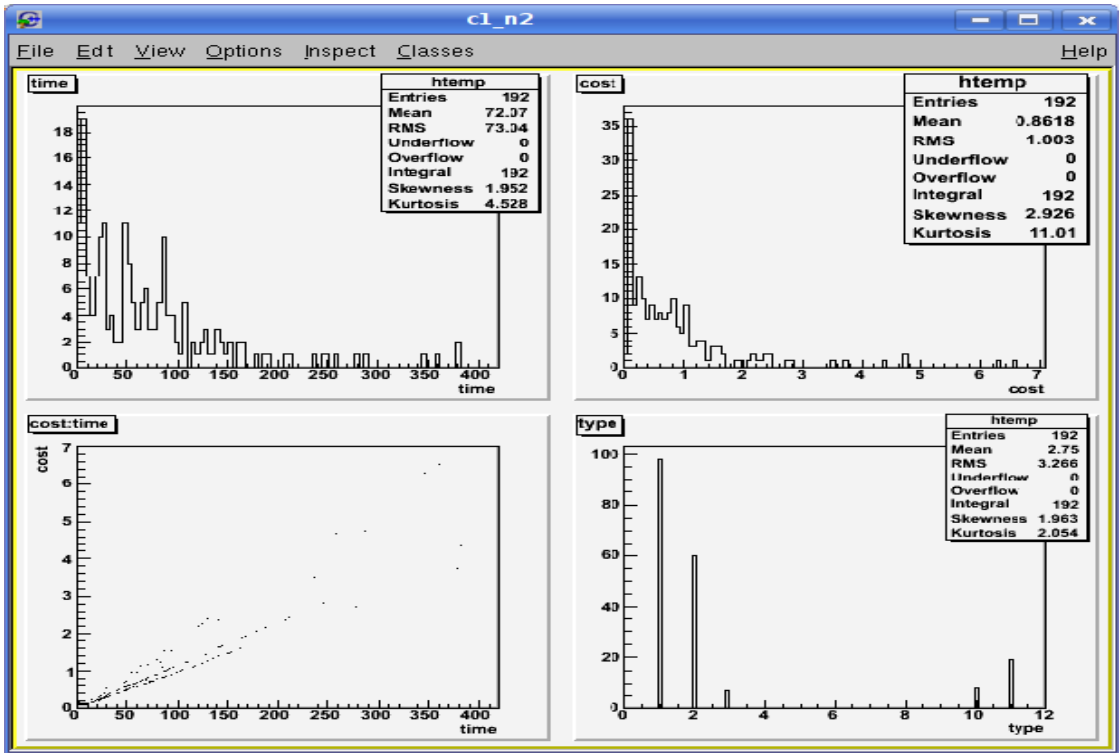
- Canvas can be splitted

```
root[1] c1->Divide(2,2); c1->cd(3);
```

- Using canvas you can set log scale or draw a grid

```
root[1] c1->SetGridx(); c1->SetGridy();
root[2] c1->SetLogy();
```

- The information shown in top right box in a plot can be customized with **`gStyle->SetOptStat(1111111);`** (before drawing the histogram!)

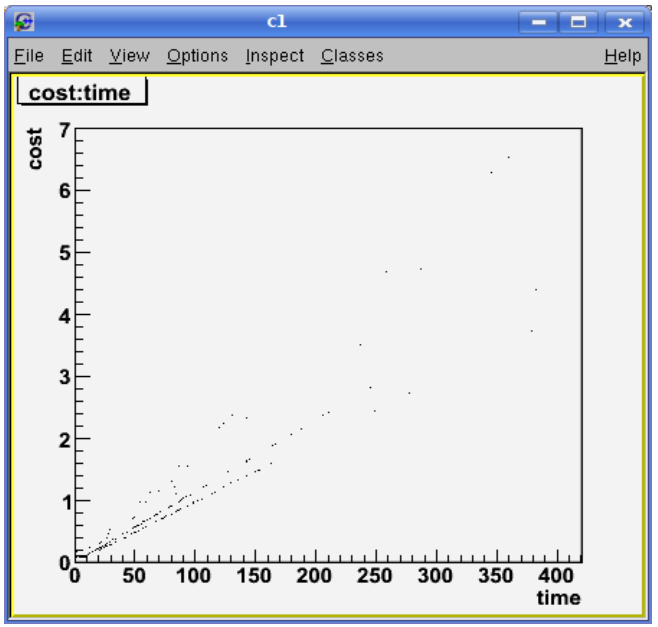


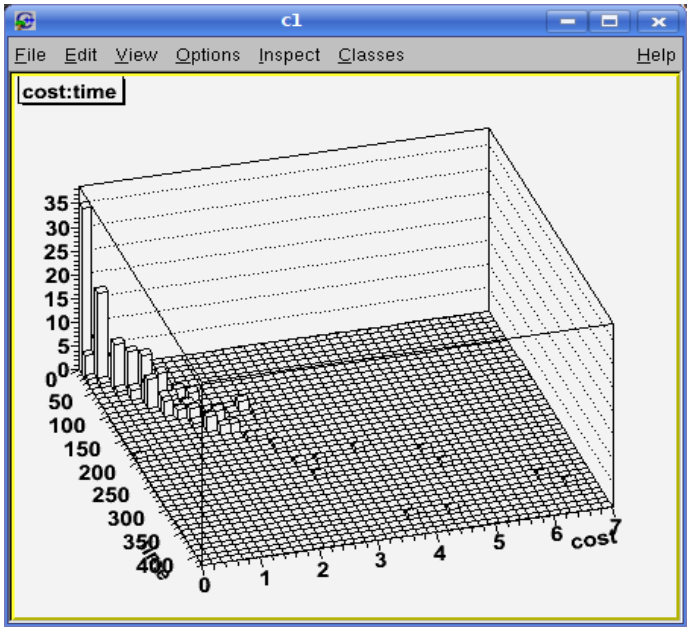
# 2D histograms

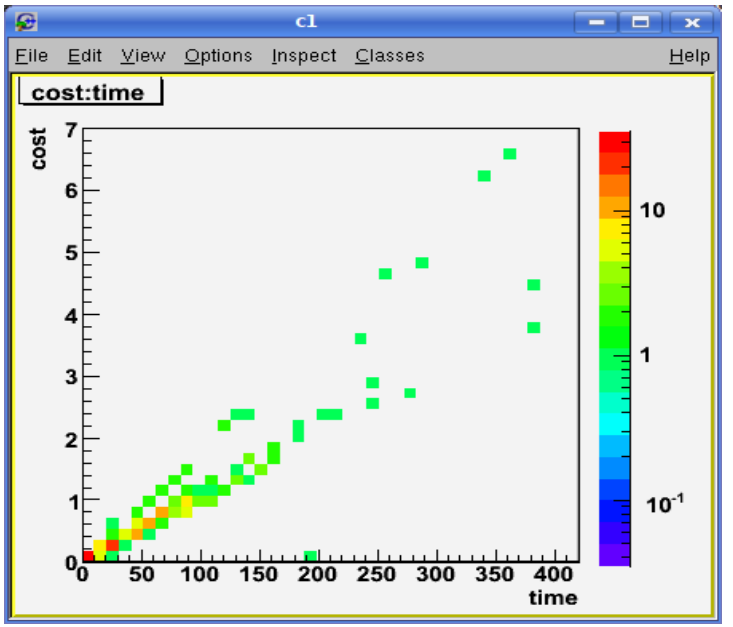
- 2D histograms can be drawn with many different styles

```
root[2] calls->Draw("cost:time")    //default:scatter plot
root[3] calls->Draw("cost:time","","lego")
root[4] gStyle->SetPalette(1)  //set nice palette colors
root[5] calls->Draw("cost:time","","COLZ")
```

- It is possible to rotate with mouse 3D graphics (e.g. lego plot)
- SetLogz can be used to set log scale for the histogram bins



# Fitting histograms

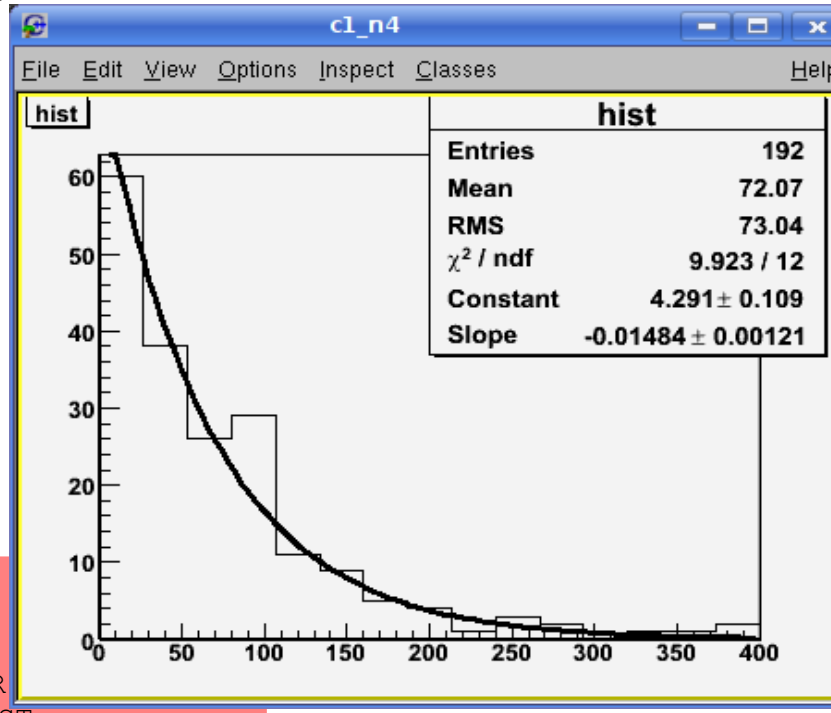
- ROOT provides predefined fittable functions for polynomials, exponential, gaussian, landaua
- User defined functions can be defined

x range

```
TF1 *f1 = new TF1("fun1","x*[0]*sin(x+[1])",-5,5);
```

parameters to be fitted

- Histograms can be fitted with TH1F::Fit(name of the function)




```
root [4] hist.Fit("expo")
 FCN=9.92324 FROM MIGRAD    STATUS=CONVERGED      62 CALLS
                    EDM=1.57791e-09    STRATEGY= 1      ERROR MATR
  EXT PARAMETER                                          STEP          FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     4.29051e+00   1.09210e-01   1.19606e-04  -4.90897e-04
   2  Slope       -1.48356e-02   1.21109e-03   1.32615e-06   2.83027e-03
(Int_t)0
```

# Plot options and additional info

- axis labeling:

```
hist->SetXTitle("#sqrt{s}");
```

- center title:

```
hist->GetXaxis()->CenterTitle(1);
```

- Legends:

```
leg = new TLegend(0.1,0.5,0.3,0.8);
leg->AddEntry(hist1,"description 1");
leg->AddEntry(hist2,"description 2");
leg->Draw("FLP");
```

**F = show the "Fill" color/style**
**L = show the "Line" color/style**
**P = show the "Point" color/marker style**

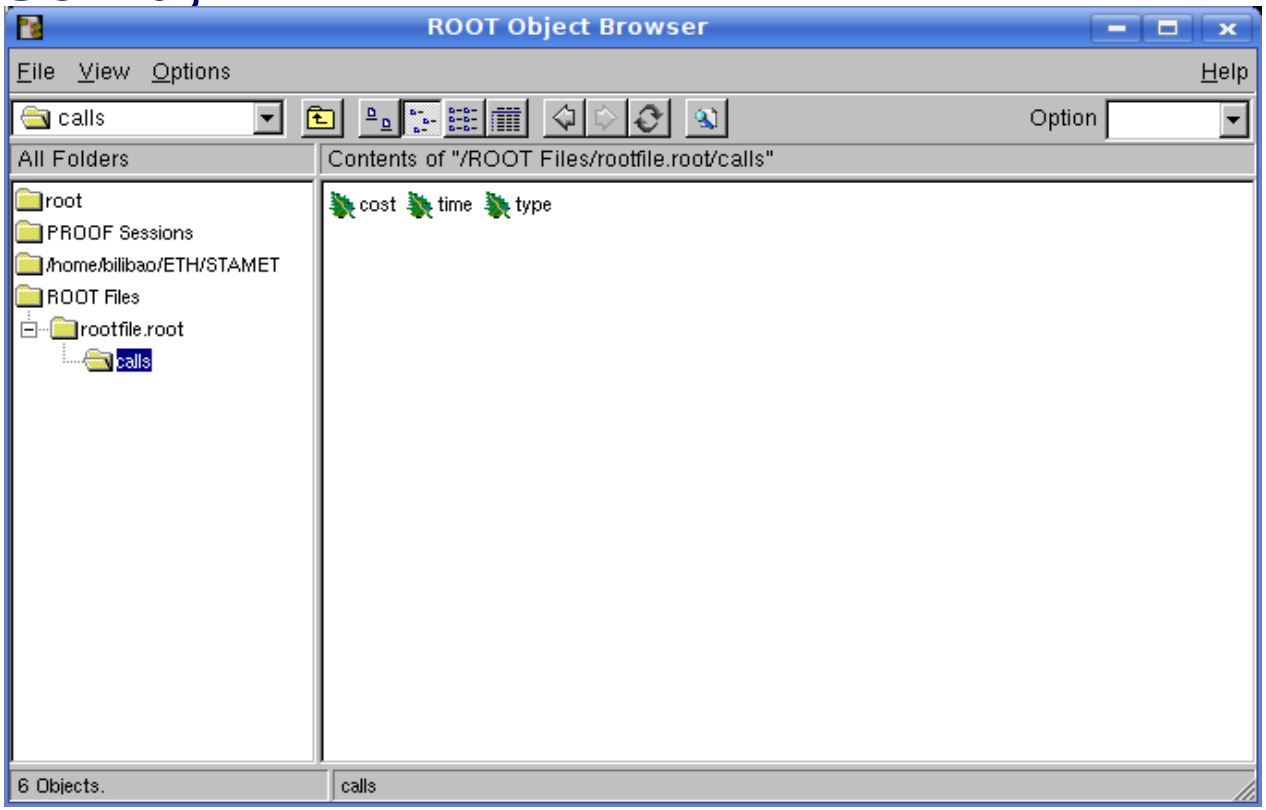
- Printing:

```
gPad->SaveAs("test.ps");
```

Can also be saved as .**eps**, .**gif**, root binary file, root macro and other graphic formats

# TBrowser

- You can open a new TBrowser in a ROOT session
  - **TBrowser b;**




- **Can be useful to interactively browse the content of root files, available histograms, TTree structure, ...**

# Example standalone application

- The program (**myapp.cc**):

ROOT init

only needed for user interaction

```
#include <TROOT.h>
#include <TApplication.h>
#include <TH1.h>

int main(int argc, char **argv)
{
  TROOT my_root_app("myapp","myapp");
  TH1F histo("myhisto","myhisto",20,0.5,20.5);
  histo.Draw();
  gApplication->Run();
}
```

- To compile

```
g++ -I$ROOTSYS/include `root-config –glibs`
  myapp.cc -o myapp
```

# For Stamet09:

- This introduction:

  http://ihp-lx.ethz.ch/CompMethPP/lectureNotes/exercises/rootintro.pdf

- ROOT installation is in:

  ROOTSYS=/h1/cern/RooT/5.04.00

  http://ihp-lx.ethz.ch/CompMethPP/lectureNotes/exercises/setroot

- Some ROOT & C++ examples, calls.txt, loop.C available at

  http://ihp-lx.ethz.ch/CompMethPP/lectureNotes/exercises/RootExamples.tar.gz

  - To unpack the archive (.tar.gz)

  # wget http://ihp-lx.ethz.ch/CompMethPP/lectureNotes/exercises/RootExamples.tar.gz

  # tar -xzvf RootExamples.tar.gz