ROOT TUTORIAL

Claudia Seitz

April 20th, 2015

Introduction to the Terascale

ROOT

- ROOT is an analysis software that is used extensively in particle physics
- The three main aspects are:
  - Graphic/Plotting
    - Various 1-dim up to n-dim histogram formats
    - Graphs and functions
  - Data analysis
    - Math libraries
    - Statistical libraries such as RooFit /RooStat
    - TMVA (neural network, boosted decision trees, etc.)
  - Data storage
    - Data structures for event-based data analysis
- C++ and python (PyRoot) can both be used

Some technical details

- Connect to either eduroam or the school network:
  - Name: terascale
  - WPA/WPA2-PSK: XxPWjNH7
- Code examples throughout the talk with colors

```
Execute this
```

```
Some example code
```

- All will get school accounts for naf
  - Example: ssh -X -Y school30@naf-school02.desy.de
- Setup the needed software

```
module avail
module load root/5.34
```

Installation on your laptop (maybe for later)

- **Installation**
  - A recent version of ROOT 5 can be obtained from http://root.cern.ch/drupal/content/production-version-534 as binaries for Linux, Windows and Mac OS X and as source code.

- **Linux - Ubuntu**
  - Ready-to-use packages of ROOT are available for Ubuntu. They can be installed with:

```
sudo apt-get install root-system
```

- **Windows**
  - For Windows the following software needs to be downloaded and installed: ROOT 5.34: ftp://root.cern.ch/root/root_v5.34.10.win32.vc10.msi
  - Python: https://www.python.org/downloads/

Getting started: C++

- ROOT is prompt based and speaks C++

```
$ root -l
root [0] gROOT->GetVersion()
(const char* 0x1094c4221)"5.34/18 »
root [1] sqrt(9) + 4
(const double)7.000000000000000e+00
```

- Quit the root session

```
root [5] .q
```

- External macros

```
root [2] .x Example.C(2)
```

or

```
root [3] .L Example.C
root [4] Example(2)
```

Create Example.C

```
float Example(float x) {
  float x2 = x*x;
  return x2;
}
```

From command line (quotation marks needed if function takes argument):

```
$ root -l "Example.C(2)"
```

Getting started: PyROOT

- Start the python environment and load ROOT

```
$ python
>>> from ROOT import *
>>> gROOT.GetVersion()
'5.34/18'
>>> sqrt(9) + 4
7.0
>>> from Example import *
>>> Example(2)
4
>>>
```

Create Example.py (function)

```
def Example( x ):
    x2 = x*x
    return x2
```

Create Example2.py (plain macro)

```
from ROOT import *
print "Hello World"
for i in range(0,5):
    print i
```

```
$ python -i Example2.py
            or
>>> from Example import *
```

-i keeps the python prompt open

- Quit the session

```
>>> quit() (or Ctrl + d)
```

Comparison: Python vs. C++

- Both languages have their pros and cons

python	C/C++
interpreted	compiled
slower execution of python code	fast
dynamic typing /checks at runtime	strict type checking at compile time
automatic memory management	manual memory management
blocks separated by indentation	code blocks separated by {}

- I often mix and match depending on the task
  - Python wrappers for defining inputs, reading parameters, plotting, calling C++ code, etc…
  - C++ code for calculations, fitting, etc…

Python

C++


```python
#defining a variable
a = 1
b = 1.5
#printing things to the screen
print a, "is not equal", b

#importing functions/classes
from ROOT import TH1F

#Indentation defines commands
#loops/statement

#For loop
for i in range(0,10):
    print i
#if/else statements
if b == c:
    print "they are equal"
elif b >  c:
    print "b is bigger"
else:
    print "c is bigger"
```

```cpp
//defining a variable
int a = 1;
float b = 1.5;
//printing output
cout<<a<<" is not equal "<<b<<endl;

//importing packages
#include "TH1F.h"

//{} define the commands inside
//loops/statement

//For loop
for (int i =0; i < 10; i++){
     cout << i << endl;}
//if/else statements
 if (b == c){
   cout<<"they are equal"<<endl;}
 else if ( b > c){
   cout<<"b is bigger"<<endl;}
 else{
   cout<<"c is bigger"<<endl;}
```

Basic classes in ROOT

- **TObject:** base class for all ROOT objects
- **TH1**: base class for 1-, 2-, 3-D Histograms
- **TStyle**: class for style of histograms, axis, title, markers, etc…
- **TCanvas**: class for graphical display
- **TGraph**: class of graphic object based on x and y arrays
- **TF1**: base class for functions
- **TFile**: class for reading/writing root files
- **TTree**: basic storage format in ROOT
- **TMath**: class for math routines
- **TRandom3**: random generator class

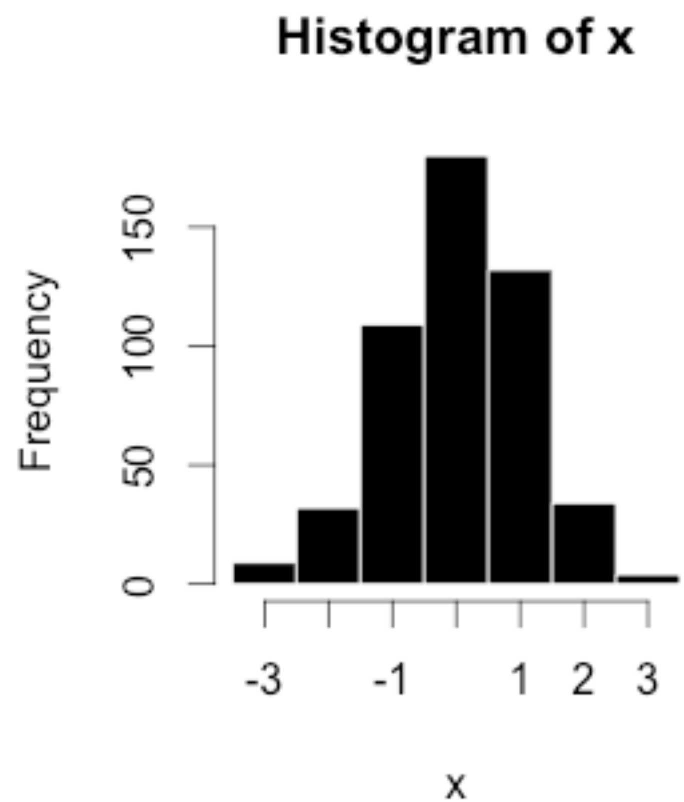
Complete list: http://root.cern.ch/root/html/ClassIndex.html

Histograms

- A histogram is just occurrence counting, i.e. how often a certain outcome appears

-3
-3.3
2
2.5
-1
1.4
3.4
-2.9
3.3
3.2
3.4
-2.9
2
2.5
-1
….

Bin	Count
[-3.5, -2.5]	9
[-2.5, -1.5]	32
[-1.5, -0.5]	109
[-0.5, 0.5]	180
[0.5, 1.5]	132
[1.5, 2.5]	34
[2.5, 3.5]	4



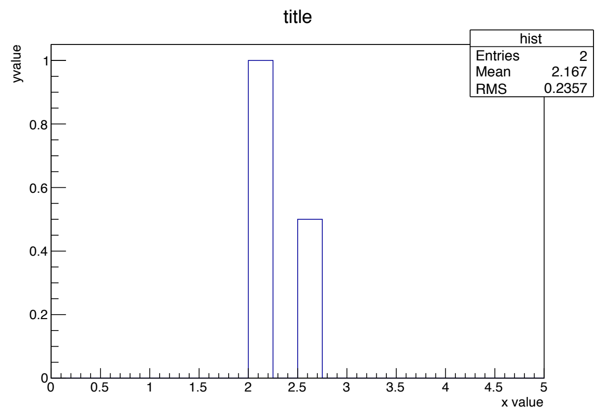
Histograms in ROOT

- Histograms can be:
  - Standard classes: 1D (TH1), 2D (TH2), 3D(TH3)
  - Special class: n-D (THn or THnSparse)
  - Content: integers (TH1I), floats (TH1F), double (TH1D)

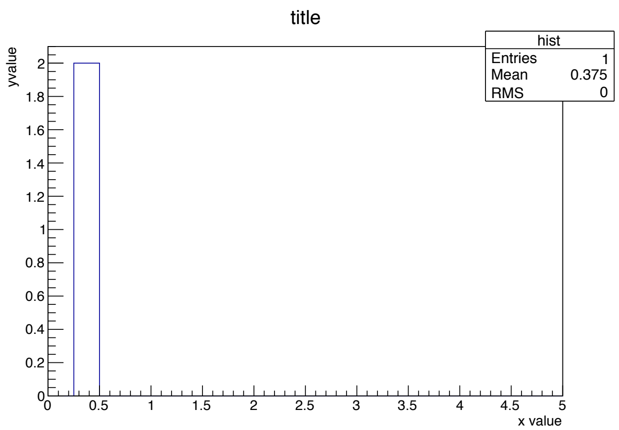
```
>>> from ROOT import *
>>> hist = TH1F("hist", "title; x value; y value", 20, 0, 5)
```

```
>>> hist.Fill(2)
>>> hist.Fill(2.5,0.5)
```

```
>>> hist.SetBinContent(2,2)
```



Increase bin at x value by 1 (default) (or 0.5 “weight”)



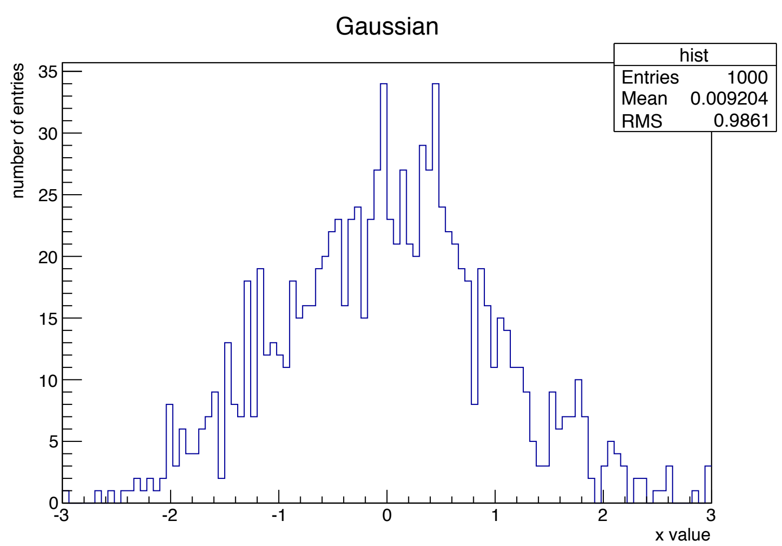
Set content of bin 2, which corresponds to values 0.25 < x < 0.5, to 2

Histograms in ROOT

- Fill histogram according to Gaussian distribution with 1000 entries and extract mean and RMS

```
>>> from ROOT import *
>>> hist = TH1F("hist", "Gaussian; x value; number of entries", 100, -3, 3)
>>> hist.FillRandom("gaus", 10000)
>>> hist.Draw()
```

```
>>> hist.GetBinContent(58)
34.0
>>> hist.GetMean()
0.0092044895591l6142
>>> hist.GetRMS()
0.98606676284414O
```




```
>>> #Change binning of histogram
>>> hist.Rebin(2)
>>> #Multiply each bin by factor
>>> hist.Scale(2)
```

One can always combine bins (rebin) but not the other way around

Histograms styles


```
>>> hist.Draw("OPTION")
```

https://root.cern.ch/root/html/THistPainter.html

Option	Explanation
"E"	Draw error bars.
"HIST"	When an histogram has errors it is visualized by default with error bars. To visualize it without errors use the option "HIST".
"SAME"	Superimpose on previous picture in the same pad.
"TEXT"	Draw bin contents as text.
Options just for TH1	
"C"	Draw a smooth Curve through the histogram bins.
"E0"	Draw error bars. Markers are drawn for bins with 0 contents.
"E1"	Draw error bars with perpendicular lines at the edges.
"E2"	Draw error bars with rectangles.
"E3"	Draw a fill area through the end points of the vertical error bars.
"E4"	Draw a smoothed filled area through the end points of the error bars.
Options just for TH2	
"COL"	A box is drawn for each cell with a color scale varying with contents.
"COLZ"	Same as "COL". In addition the color palette is also drawn.
"CONT"	Draw a contour plot (same as CONT0).
"SURF"	Draw a surface plot with hidden line removal.

Exercise: Histograms

Write a python macro ExerciseHist.py

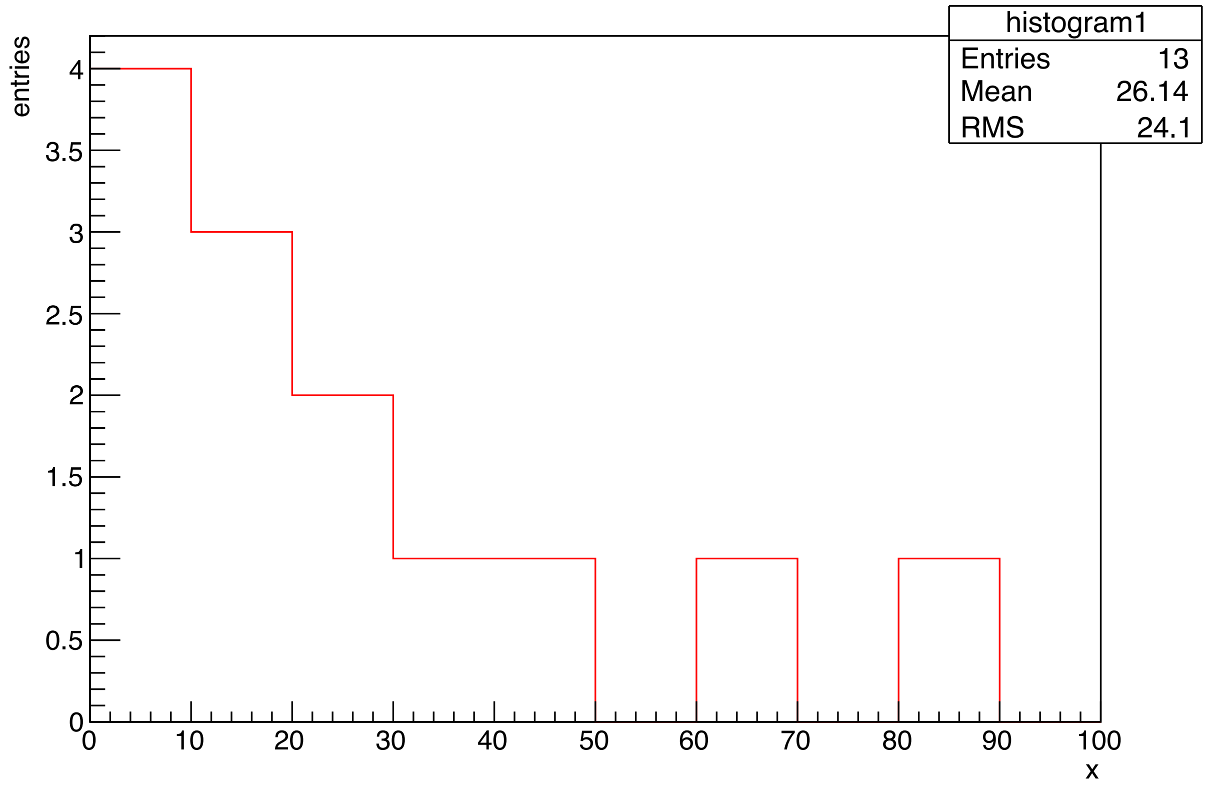
1. Create a histogram with 10 bins ranging from 0. to 100. with title/x-axis label "x"
2. Fill the histogram at the following numbers: 11.3, 25.4, 18.1
3. Fill the histogram with the square of all integers from 0. to 9.
   (Hint: A simple loop will save you from typing several lines of code)
4. Draw the histogram.
5. Calculate the mean value and the rms and show it on the screen.

   ```
   print mean, rms
   ```
6. Calculate the integral of the histogram.
7. Identify the bin with the maximum number of entries.
8. Calculate the maximum bin content.
9. Set the y-axis label to "entries".
10. Set the line color of the histogram to red.
11. Run with

    ```
    python -i ExerciseHist.py
    ```

- One dimensional histogram TH1F.
- Constructor of a histogram: TH1F::TH1F(const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup).
- Fill a histogram: Int_t TH1F::Fill(Double_t x)
- Draw a histogram: void TH1F::Draw(Option_t* option = "")
- Mean of a histogram: Double_t TH1F::GetMean(Int_t axis = 1) const
- RMS of a histogram: Double_t TH1F::GetRMS(Int_t axis = 1) const
- Mode of a histogram: Int_t TH1F::GetMaximumBin() const
- Get the bin content of a histogram: Double_t TH1F::GetBinContent(Int_t bin) const
- Integral of a histogram: Double_t TH1F::Integral(Option_t* option = "") const
- Y-axis used to draw the histogram: TAxis* TH1F::GetYaxis() const
- Access axis and set label void TAxis::SetTitle(char*)
- Change line color of the histogram: void TAttLine::SetLineColor(Color_t lcolor).
  The color index for red is named kRed.

Exercise: Histograms



Canvas and Legends in ROOT

- ROOT distinguishes between a histogram and a "canvas" where is histogram is drawn on
- Multiple histograms (and other objects) can be drawn on the same canvas with Draw("same")
- Legends can be added to the canvas

```
>>> from ROOT import *
>>> c = TCanvas("canvas", "canvas", 800 , 600)
...
...
>>> legend = TLegend(0.16, 0.63, 0.45, 0.91)
>>> legend.AddEntry(hist1, "Gaussian", "l")
>>> legend.AddEntry(hist2, "Polynomial", "l")
>>> legend.Draw()
```

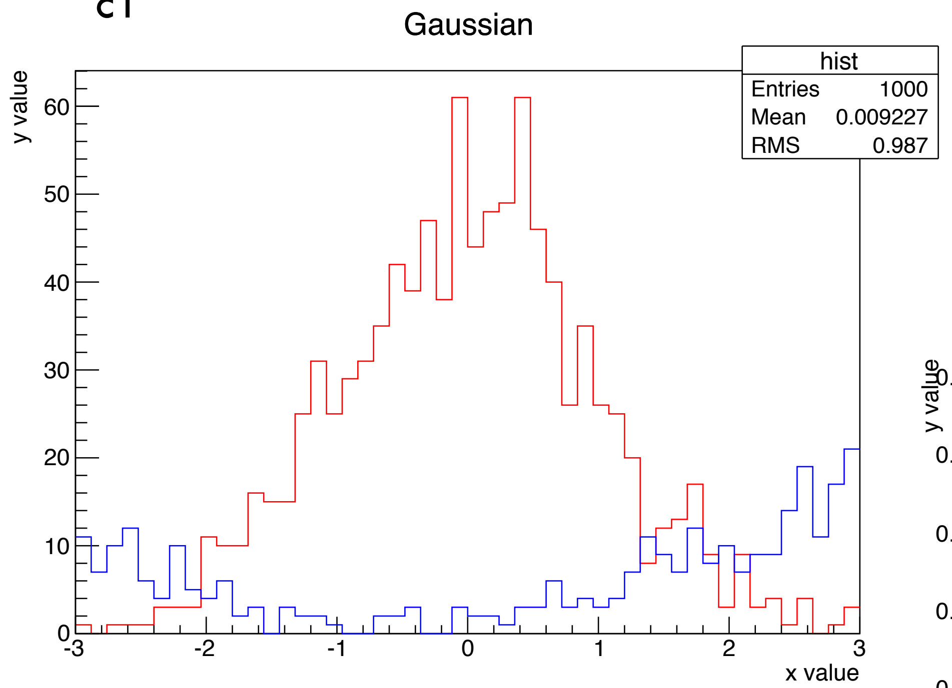
Exercise: Canvas and Legends

Write a python macro ExerciseCanvas.py:

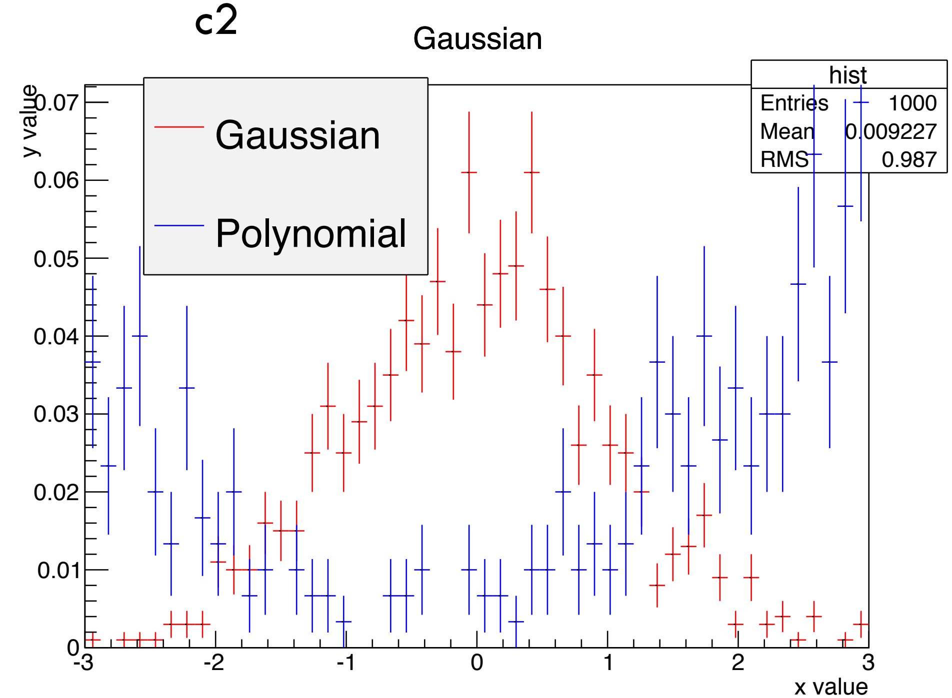
- Create two histograms with 50 bins ranging from -3. to 3. with two different names
- Fill first histogram with Gaussian distribution with 1000 entries
- Fill second histogram with a second order polynomial and 500 entries
  - `hist2.FillRandom("pol2", 500)`
- Create a TCanvas c1 and draw both histograms (option "same")
- Set the line color of the first histogram to kRed and the second to kBlue
- Clone both histograms
  - `hist1b = hist1.Clone()`
- Scale both cloned histograms by the inverse of their respective integral, i.e. normalise them to unit area.
- Create a TCanvas c2 and draw both cloned histograms
- Create a legend at position (0.16, 0.63, 0.45, 0.91) and add entries for both histograms to it. Draw the legend.
- Save both canvases as pdf files and as root file
  - `c.Print("filename.pdf")`
  - `c.SaveAs("filename.root")`

Exercise: Canvas and Legends

c1



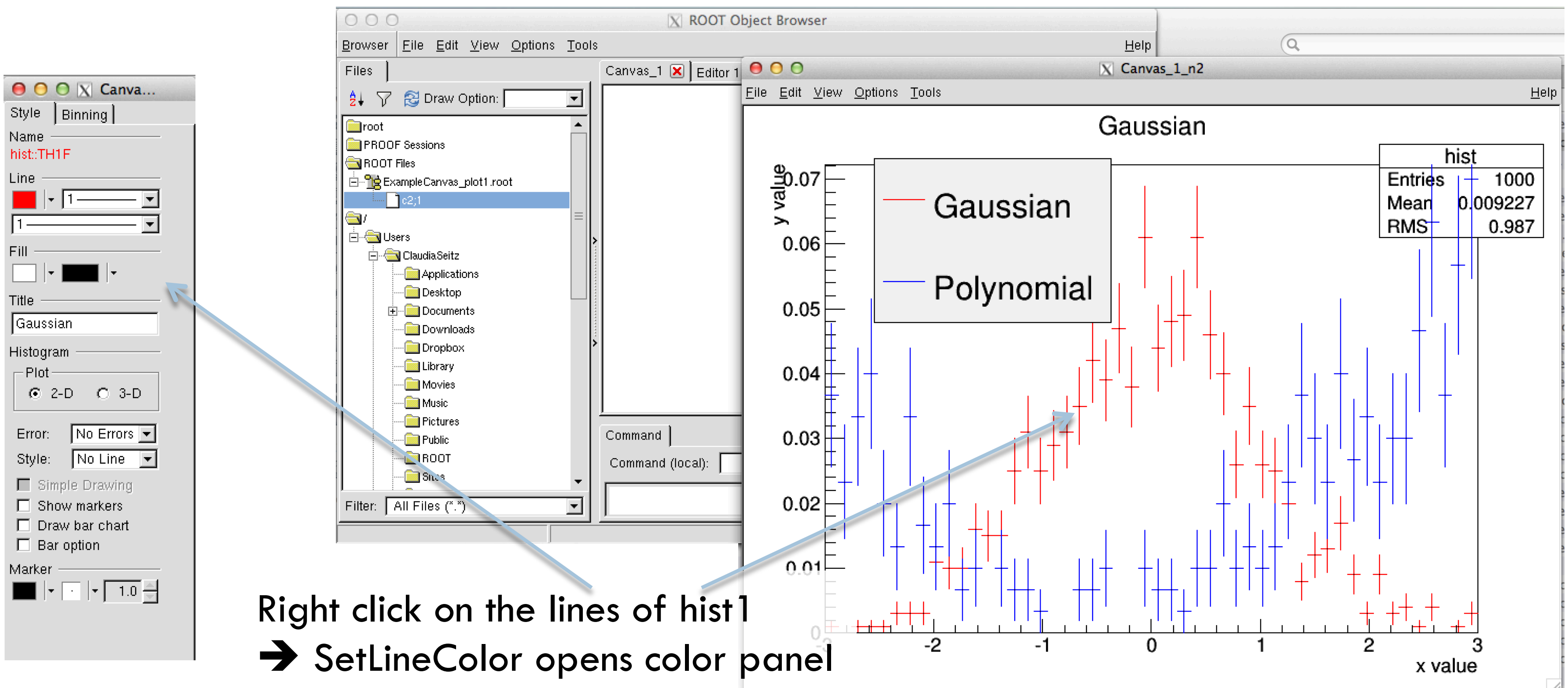
c2



Graphical User Interface (GUI)

- GUI can be used for visualization and adjustment of styles or plotting on the fly

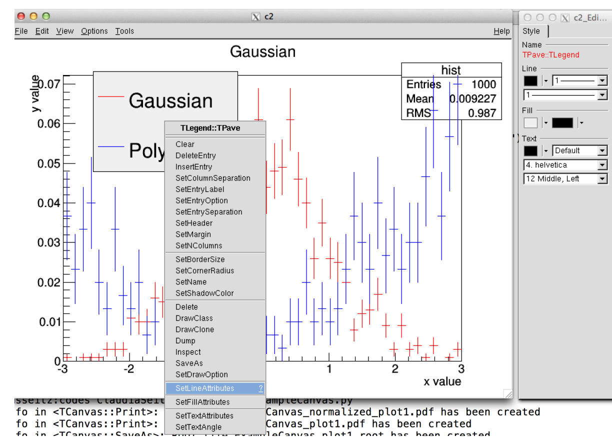
```
>>> from ROOT import *
>>> b = TBrowser()
>>> f = TFile("filename.root")
```



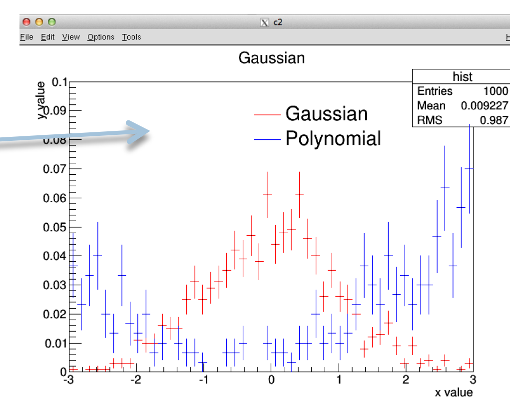
Right click on the lines of hist1
➔ SetLineColor opens color panel

Graphical User Interface (GUI)

- Sometimes changing things by hand are much easier
  - Position of legends (coordinates are given as percentage with respect to the boundaries of the plot)
  - Font sizes of axis labels, offset of lables
- Make the change manually
- Save the canvas as a .C file
- Find the code, import the settings back




```
TLegend *leg = new TLegend(0.4560302,0.7062937,0.7462312,0.8426573,NULL,"brNDC");
leg->SetBorderSize(1);
leg->SetLineColor(0);
leg->SetLineStyle(1);
leg->SetLineWidth(1);
leg->SetFillColor(0);
leg->SetFillStyle(1001);
```

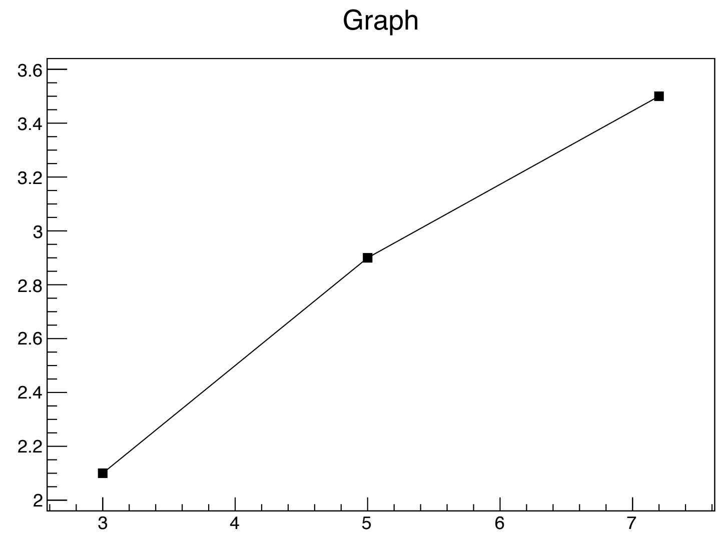


New legend position and settings: white bkg and line color

Graphs in ROOT

- Three main classes for graphs `TGraph`, `TGraphErrors`, `TGraphAsymmetricErrors`
- Graphs are used to display value pairs, errors can be defined to be either symmetric or antisymmetric

```
>>> from ROOT import *
>>> #create graph with 3 points
>>> graph = TGraph(3)
>>> #set three points of the graph
>>> graph.SetPoint(0, 3.0, 2.1)
>>> graph.SetPoint(1, 5.0, 2.9)
>>> graph.SetPoint(2, 7.2, 3.5)
>>> #set styles
>>> graph.SetMarkerStyle(21)
>>> graph.SetMarkerSize(1)
>>> #Draw axis (A), points (P), and line (L)
>>> graph.Draw("APL")
```

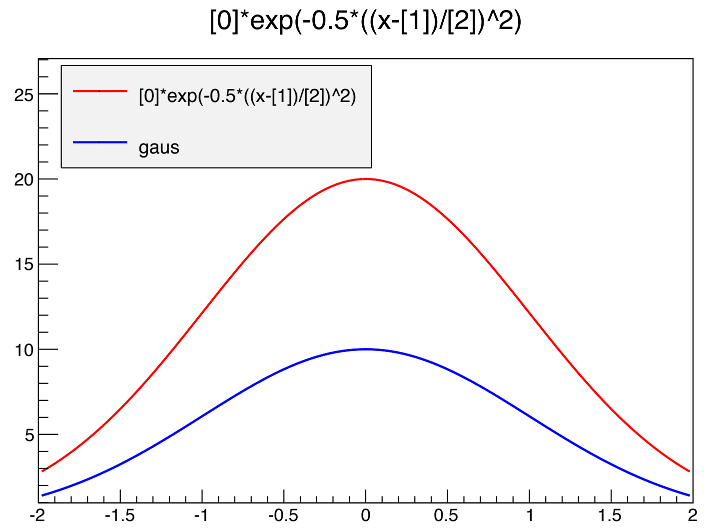


Functions in ROOT

- Classes for TF1, TF2, TF3 for 1 to 3 dimensional functions

```
>>> from ROOT import *
>>> #Use of predefined functions "gaus", "pol1","pol3", etc.
>>> fGaus = TF1("fGaus", "gaus", -2, 2)

>>> #Use of custom user functions
>>> f = TF1("f","[0]*exp(-0.5*((x-[1])/[2])^2)", -2, 2)
```

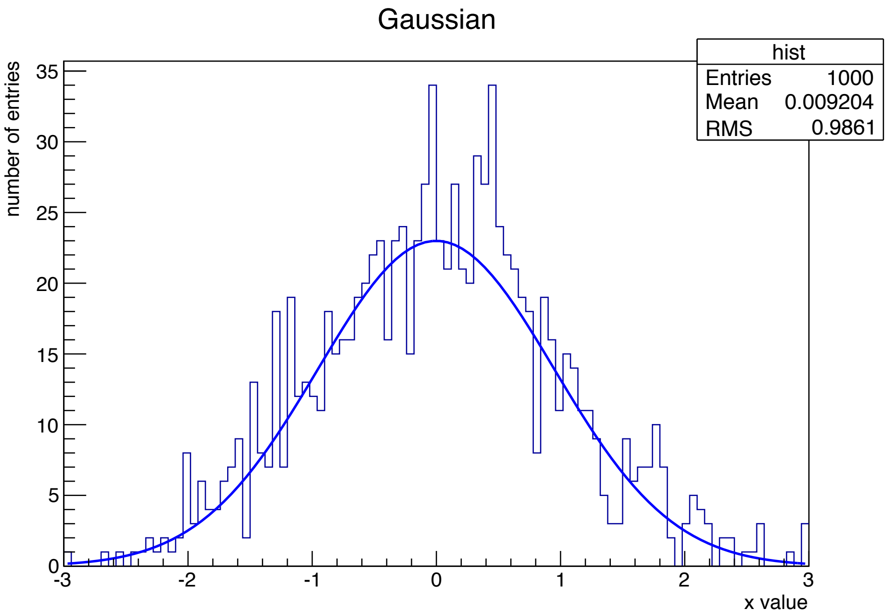
```
>>> #Setting the parameters
>>> f.SetParameter(0,20)
>>> f.SetParameter(1,0)
>>> f.SetParameter(2,1)

>>> fGaus.SetParameter(0,10)
>>> fGaus.SetParameter(1,0)
>>> fGaus.SetParameter(2,1)
```



Fitting in ROOT


```
>>> hist.Fit("fGaus")
 FCN=97.4876 FROM MIGRAD    STATUS=CONVERGED      67 CALLS          68 TOTAL
                     EDM=3.44445e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     2.29946e+01   1.02159e+00   3.70880e-03   2.59473e-04
   2  Mean        -2.11506e-03   3.28869e-02   1.58874e-04   5.12360e-03
   3  Sigma        9.50152e-01   3.00472e-02   3.74233e-05   1.80927e-02
<ROOT.TFitResultPtr object at 0x7fa0db5b9e70>
```

```
>>> hist.Draw()
>>> fGaus.Draw("same")
```



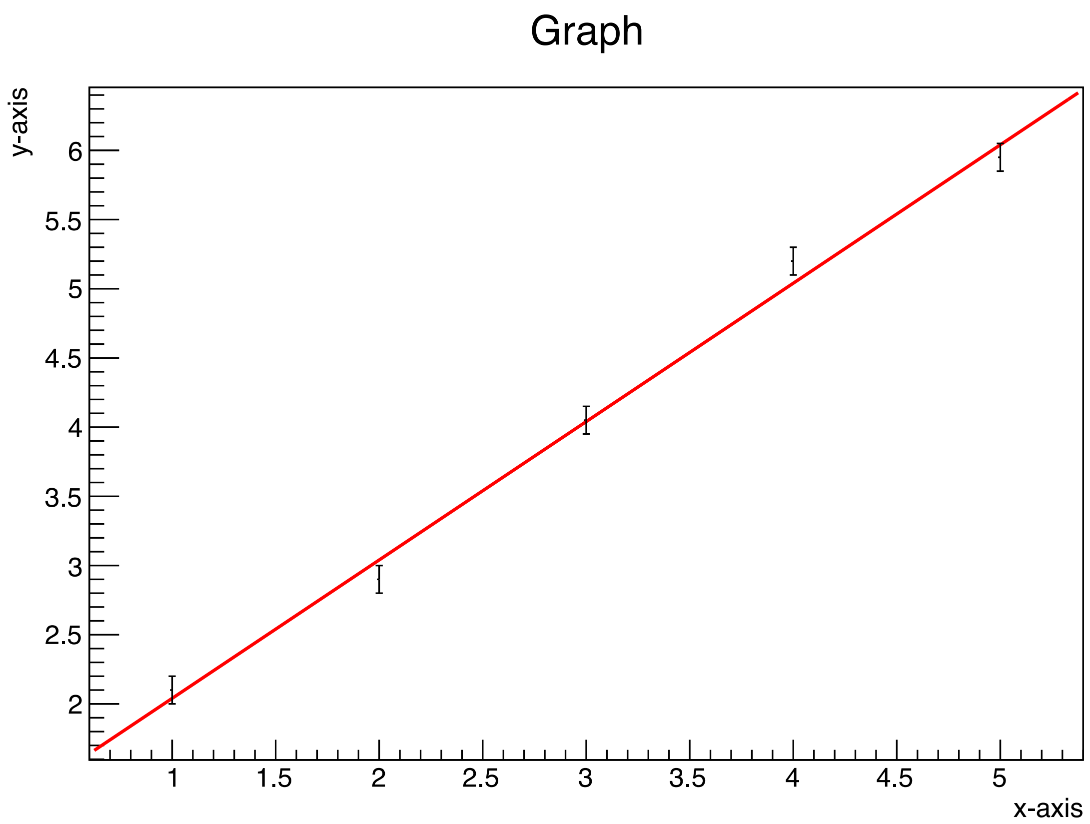
Exercise: Graphs and Fits

Write a python macro ExerciseGraph.py:

- Create a graph with symmetric errors and 5 points.
- Set the following points (0-4): (1.0, 2.1), (2.0, 2.9), (3.0, 4.05), (4.0, 5.2), (5.0, 5.95)
- Set the errors on x to 0.0 and the errors on y to 0.1.
- Draw the graph including the axes and error bars.
- Create a one dimensional function f(x)=mx + b and fit it to the graph.
- Obtain the two parameters a and b from the function and their estimated uncertainties.

- A one dimensional graph TGraphErrors.
- A constructor of a graph: TGraphErrors::TGraphErrors(Int_t n).
- A method to set the points of a graph: void TGraphErrors::SetPoint(Int_t i, Double_t x, Double_t y).
- A method to set the errors of a graph: void TGraphErrors::SetPointError(Double_t ex, Double_t ey).
- A method to fit a graph with a function: TFitResultPtr TGraphErrors::Fit(const char *fname, Option_t *option, Option_t *, Axis_t xmin, Axis_t xmax).
- A method to return the parameters of a function: Double_t TF1::GetParameter(Int_t ipar).
- A method to return the errors on the parameters of a function: Double_t TF1:GetParError(Int_t ipar) const .

Exercise: Graphs and Fits



Classes: TFile and TTree

- TFile is basic I/O format in root
  - Open an existing file (read only)
    - `InFile = TFile("myfile.root", "OPTION")`
      - `OPTION = leave blank (read only), "RECREATE" (replace file), "UPDATE" (append to file)`
    - Files can contain directories, histograms and trees (ntuples) etc.
- ROOT stores data in TTree format
  - Tree has "entries" (e.g. collision events) each with identical data structure
  - Can contain floats, integers, or more complex objects (whole classes, vectors, etc…)
  - TNtuple is a tree that contains only floats

Creating a TTree from text file

- Copy the following text file
  - http://www.desy.de/~clseitz/School/TeraScale/
  - cp /afs/desy.de/user/c/clseitz/public/Schools/TeraScale/basic.dat .

```
>>> from ROOT import *
>>> f = TFile("ntuple.root","RECREATE")
>>> t = TTree("ntuple","reading data from ascii file")
>>> t.ReadFile("basic.dat","x:y:z")
>>> t.Write()
```

```
clseitz@naf-hh: $ more basic.dat
-1.102279 -1.799389 4.452822
1.867178 -0.596622 3.842313
-0.524181 1.868521 3.766139
-0.380611 0.969128 1.084074
0.552454 -0.212309 0.350281
-0.184954 1.187305 1.443902
0.205643 -0.770148 0.635417
```

Working with TTrees

- Get the following root file (or use from previous page)
  - `cp /afs/desy.de/user/c/clseitz/public/Schools/TeraScale/basic.root .`

```
>>> from ROOT import *
>>> f = TFile("basic.root")
>>> t = f.Get("ntuple")
```

```
>>> t.Show(2)
======> EVENT:2
 x                = -0.524181
 y                = 1.86852
 z                = 3.76614
```

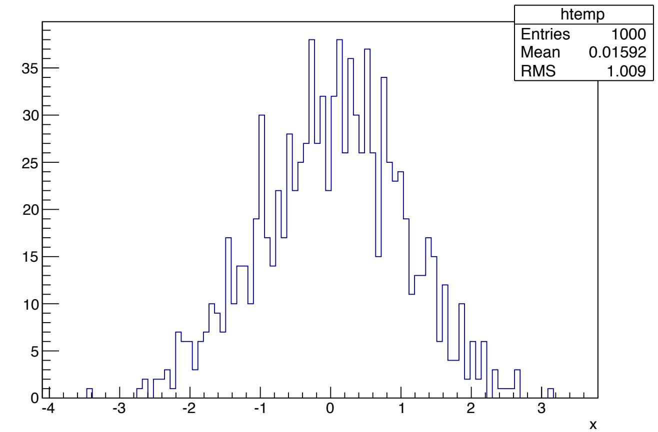
Shows the content and structure of the tree for one entry

```
>>> t.Scan()
************************************************
*    Row   *         x *         y *         z *
************************************************
*        0 * -1.102278 * -1.799389 * 4.4528222 *
*        1 * 1.8671779 * -0.596621 * 3.8423130 *
*        2 * -0.524181 * 1.8685209 * 3.7661390 *
*        3 * -0.380611 * 0.9691280 * 1.0840740 *
```

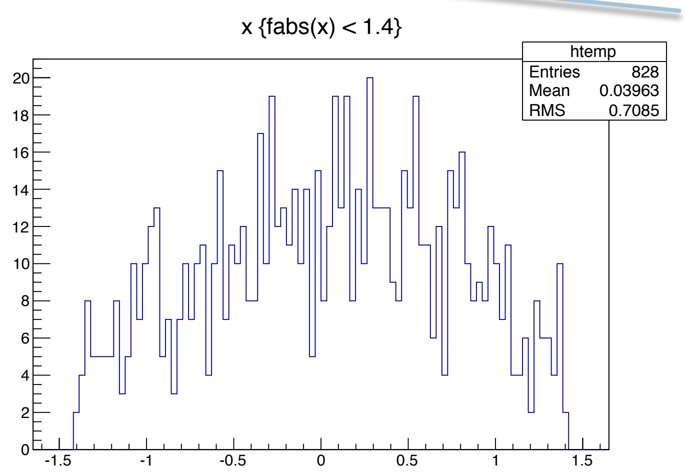
Shows one or multiple variables for all entries

Plotting quantities directly from TTrees


```
>>> t.Draw("x")
```



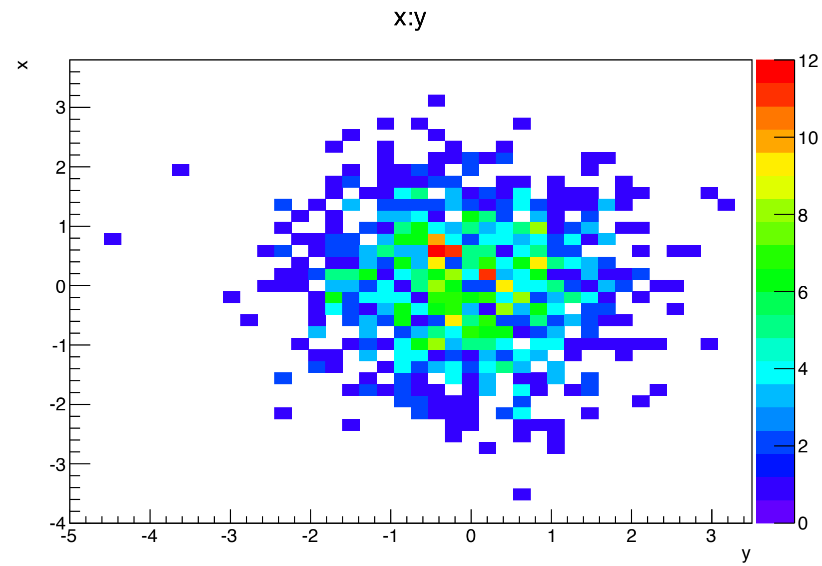

```
>>> t.Draw("x","fabs(y) < 1.4","")
829L
```



number tells you how many entries passed condition

Scatter plot shows the correlation between variables

```
>>> T.Draw("x:y","","colz”)
```



TTree functions (very useful for quick checks)

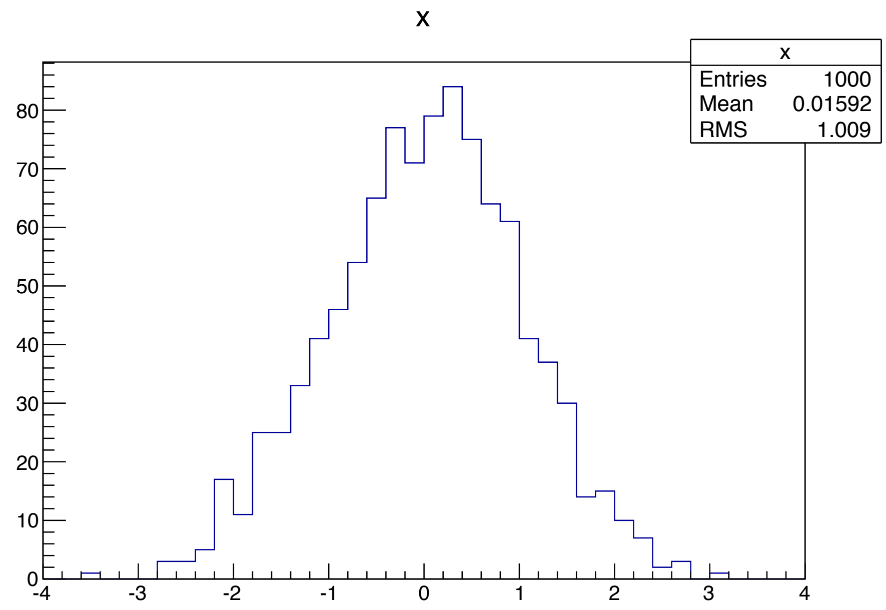
Command	Action
`t.Print()`	Prints the content of the tree
`t.Scan()`	Scans the rows and columns
`t.Draw("x")`	Draw a branch of tree

How to apply cuts:
`t.Draw("x", "x>0")`
`t.Draw("x", "x>0 && y>0")`
	
	Draw “x” when “x>0”
	Draw “x” when both x >0 and y >0

`t.Draw("y", "", "same")`	Superimpose “y” on “x”
`t.Draw("y:x")`	Make “y vs x” 2d scatter plot
`t.Draw("z:y:x")`	Make “z:y:x” 3d plot
`t.Draw("sqrt(x*x+y*y)")`	Plot calculated quantity
`t.Draw("x>>h1")`	Dump a root branch to a histogram

Looping through entries of a TTree


```
>>> from ROOT import *
>>> f = TFile("basic.root")
>>> t = f.Get("ntuple")
>>> nEntries = t.GetEntries()
>>> hist = TH1D("x", "x",40,-4,4)
>>> for i in range(0,nEntries):
...     entry = t.GetEntry(i)
...     hist.Fill(t.x)
...
>>> hist.Draw()
```



The End

Thank you for your attention

Any more questions?