



ROOT  
Data Analysis Framework

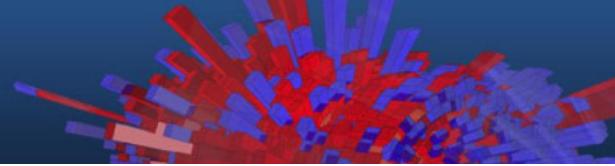
# Summer Students Course 2016

*Olivier Couet, Enric Tejedor Saavedra*

*CERN PH-SFT*



# This Course



This is an introductory ROOT Workshop.

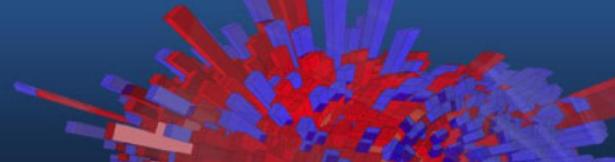
## Objectives:

- Become familiar with the ROOT toolkit
- Be able to use the C++ prompt
- Plot data
- Fit data
- Perform basic I/O operations
- Learn about SWAN and Jupyter notebooks

## Format:

- Slides treating the most important concepts
- Hands on exercises proposed during the exposition

# This Tutorial

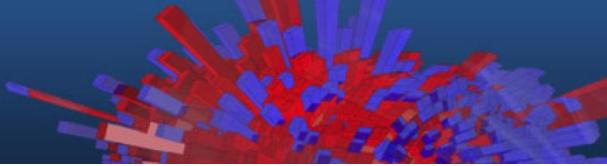


These slides are supported by the “**ROOT Primer**”

- Introductory booklet (~60 pages)
- Available on the ROOT website (html, epub, pdf): <https://root.cern.ch/guides/primer>
- Code examples can be visualised with the Jupyter Notebooks available at:  
<http://swan.web.cern.ch/content/root-primer>
- Signaled with name and the sign:  followed by the notebook’s name

Two release series of ROOT are available: ROOT5 and ROOT6  
**This lecture refers to ROOT6, version 6.06**

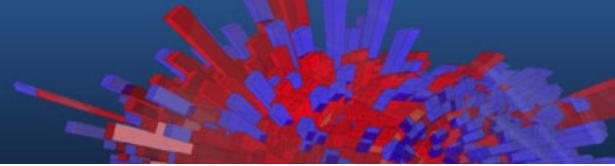
# How Can I Run ROOT?



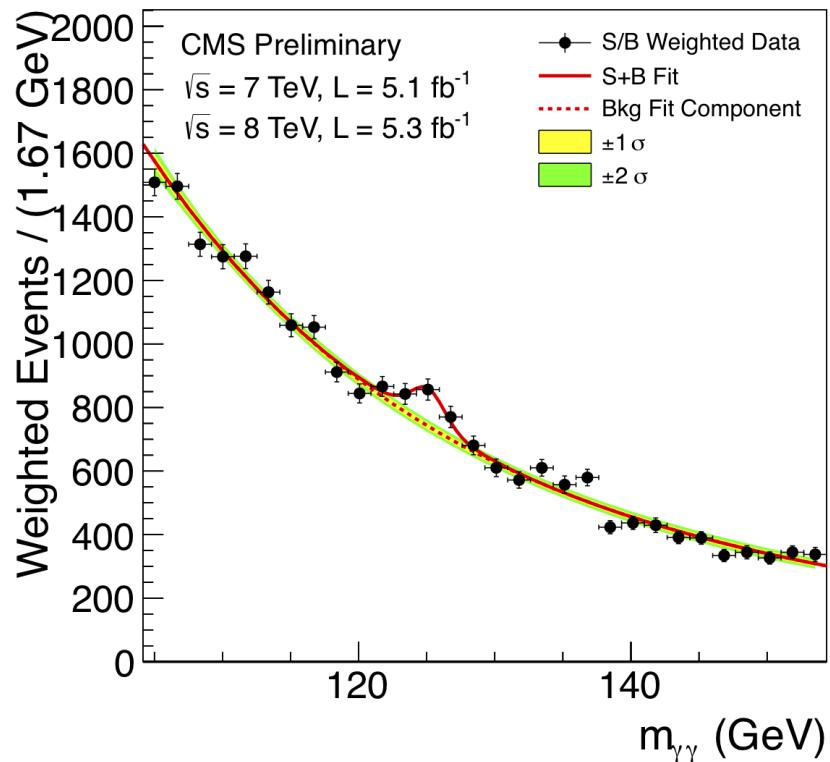
You can choose between three options:

1. Local ROOT installation
  - <https://root.cern.ch/downloading-root>
2. ROOT Virtual Machine
  - <https://github.com/root-mirror/training>
3. SWAN
  - No installation needed: only a web browser
  - Preparation: log into CERNBox at <https://cernbox.cern.ch>
  - Access SWAN service at <https://swan.cern.ch>

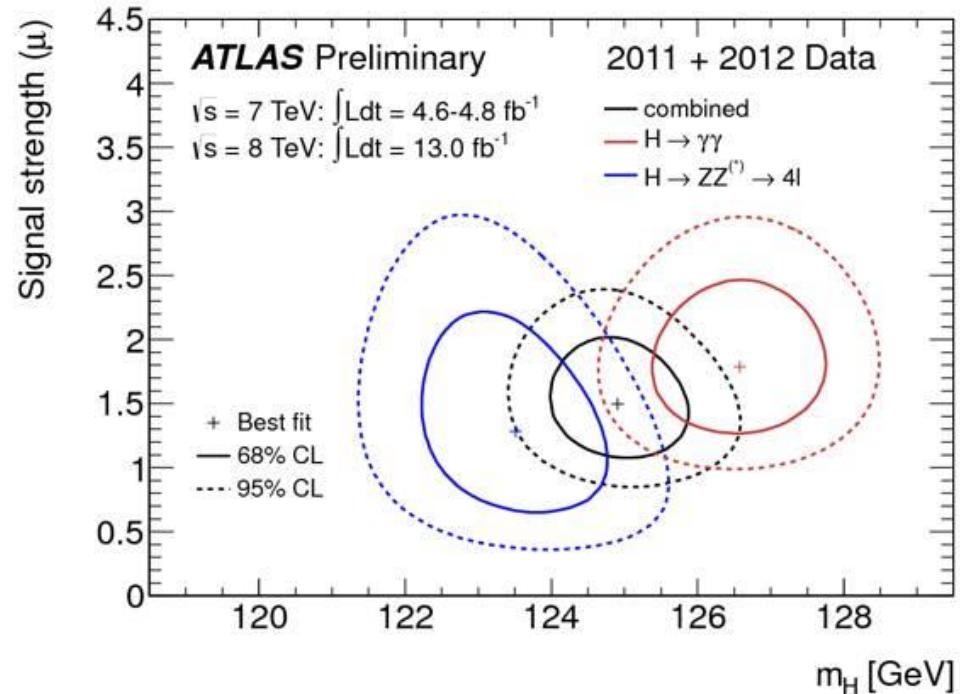
# A “Quick Tour” Of ROOT



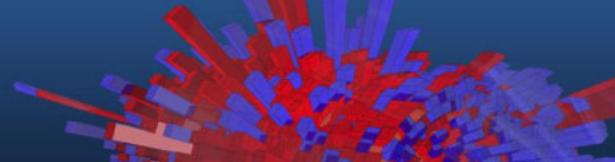
# What can you do with ROOT?



LHC collision in CMS:  
event display, also done with ROOT!



# ROOT in a Nutshell



ROOT is a software toolkit which provides building blocks for:

- Data processing
- Data analysis
- Data visualisation
- Data storage

An Open Source Project

*All contributions are warmly welcome!*



ROOT is written mainly in C++ (C++11 standard)

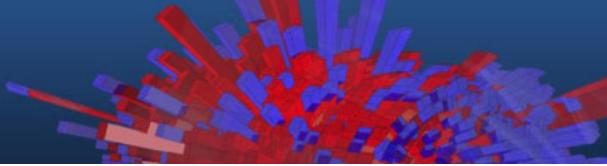
- Bindings for Python is provided.



Adopted in High Energy Physics and other sciences (but also industry)

- ~250 PetaBytes of data in ROOT format on the LHC Computing Grid
- Fits and parameters' estimations for discoveries (e.g. the Higgs)
- Thousands of ROOT plots in scientific publications

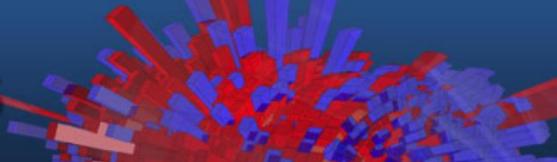
# ROOT in a Nutshell



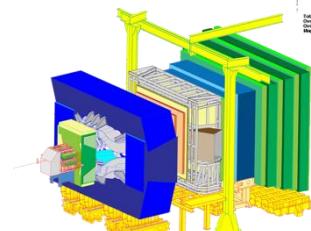
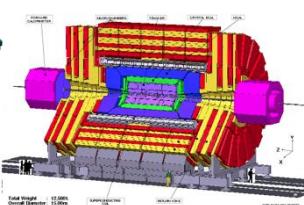
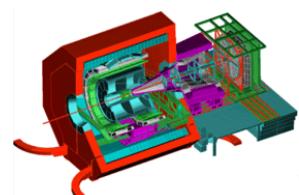
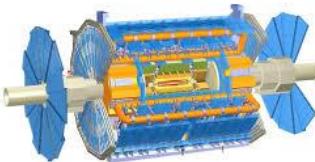
ROOT can be seen as a collection of building blocks for various activities, like:

- Data analysis: [histograms, graphs, trees](#)
- [I/O](#): row-wise, column-wise storage of **any** C++ object
- Statistical tools ([RooFit/RooStats](#)): rich modeling and statistical inference
- [Math](#): non trivial functions (e.g. Erf, Bessel), optimised math functions ([VDT](#))
- [C++ interpretation](#): fully C++11 compliant
- [Multivariate Analysis](#) (TMVA): e.g. Boosted decision trees, neural networks
- Advanced graphics (2D, 3D, event display).
- PROOF: [parallel analysis facility](#)
- And more: [HTTP servering](#), [JavaScript visualisation](#).

# ROOT Application Domains



A selection of the experiments adopting ROOT



Event Filtering

Data

Offline Processing

Reconstruction

Further processing,  
skimming

Analysis

Event Selection,  
statistical treatment ...

Raw

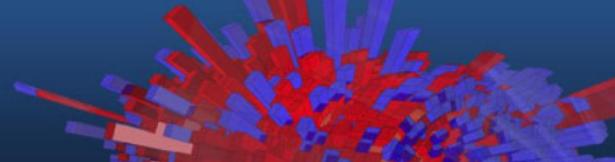
Reco

Analysis  
Formats

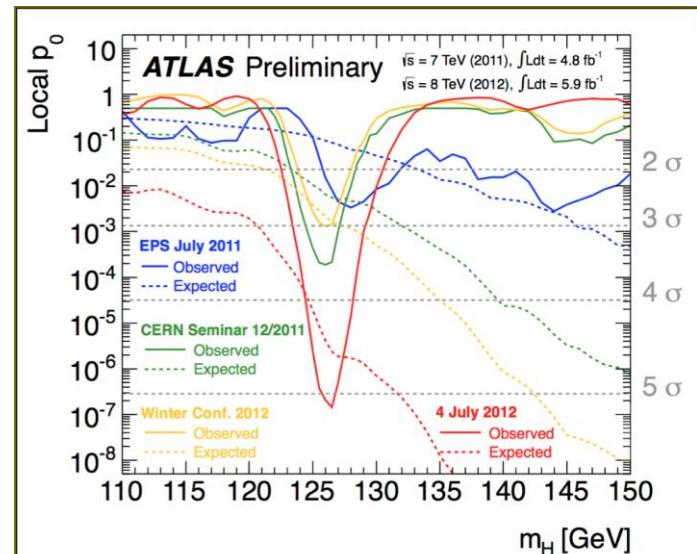
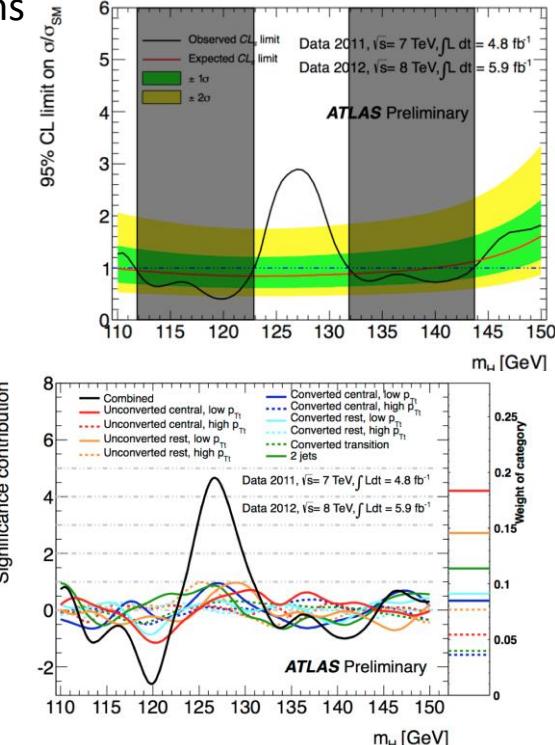
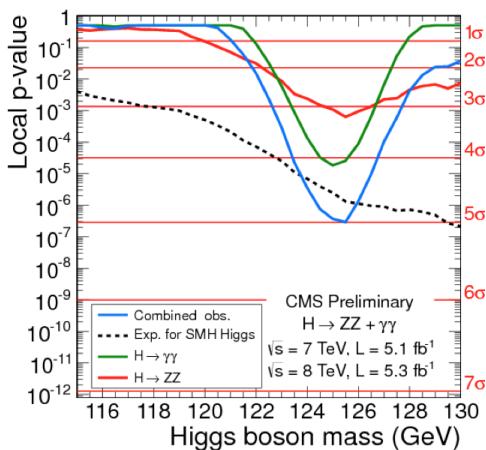
Images

Data Storage: Local, Network

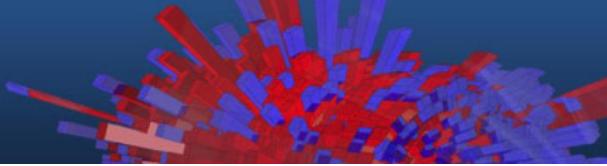
# Higgs boson discovery



On July 4<sup>th</sup> 2012, the plots presented by the ATLAS and CMS collaborations where all produced with ROOT !



# Interpreter



ROOT has a built-in interpreter : CLING

- **C++ interpretation:** highly non trivial and not foreseen by the language !
- One of its kind: Just In Time (JIT) compilation
- A C++ interactive shell.

Can interpret “macros” (non compiled programs)

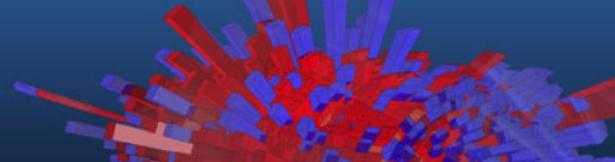
- Rapid prototyping possible

```
$ root -b  
root [0] 3 * 3  
(const int)9
```

ROOT provides also **Python bindings**:

- Can use Python interpreter directly after a simple *import ROOT*
- Possible to “mix” the two languages (see more in the following slides)

# Persistency (I/O)



ROOT offers the possibility to write C++ objects into files

- This is impossible with C++ alone.
- Used the LHC detectors to write several petabytes per year.

Achieved with serialization of the objects using the reflection capabilities, ultimately provided by the interpreter

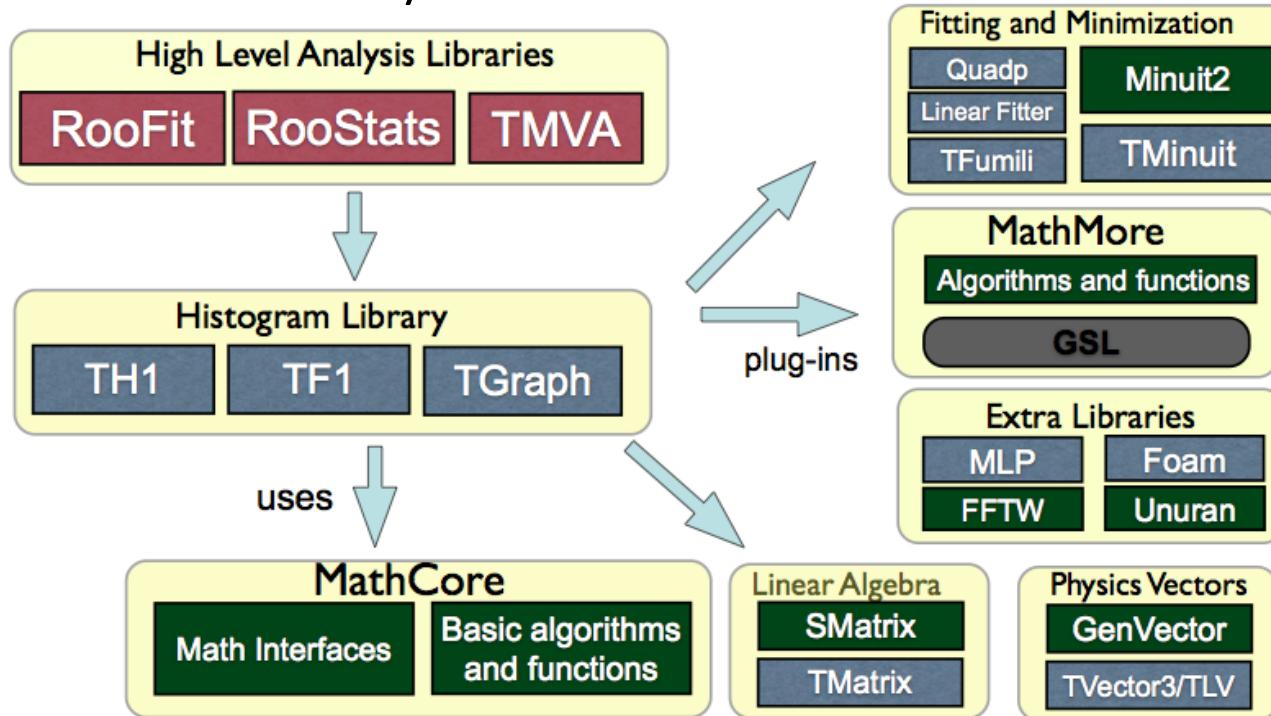
- Raw and column-wise streaming

As simple as this for ROOT objects: one method - *TObject::Write*

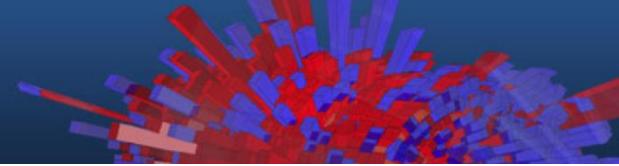
Cornerstone for storage  
of experimental data

# ROOT Math/Stats Libraries

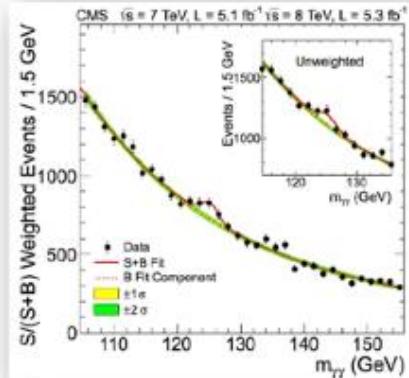
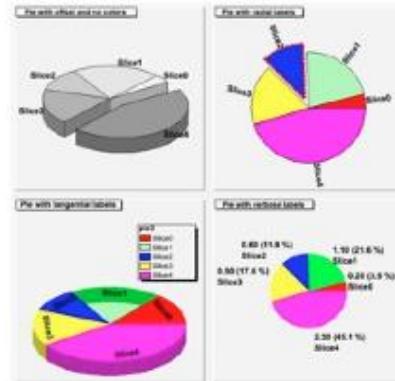
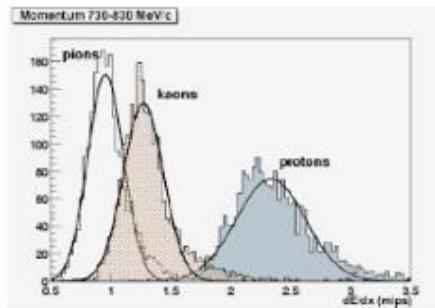
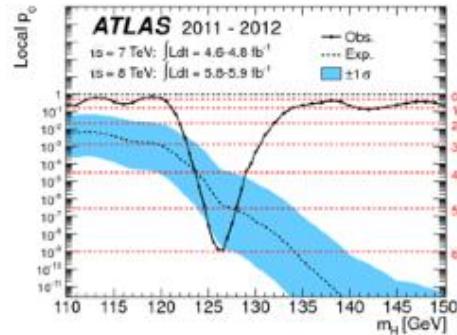
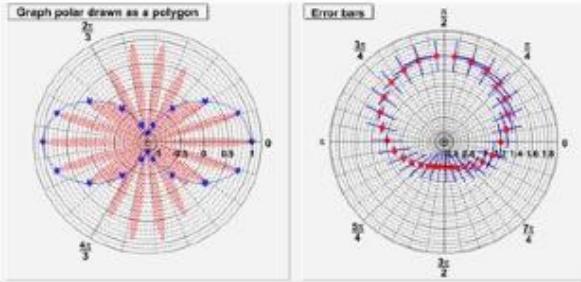
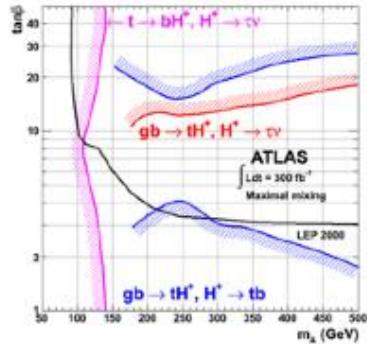
ROOT provides a reach set of mathematical libraries and tools needed for sophisticated statistical data analysis



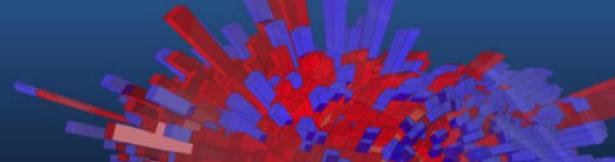
# Graphics In ROOT



Many formats for data analysis, and not only, plots



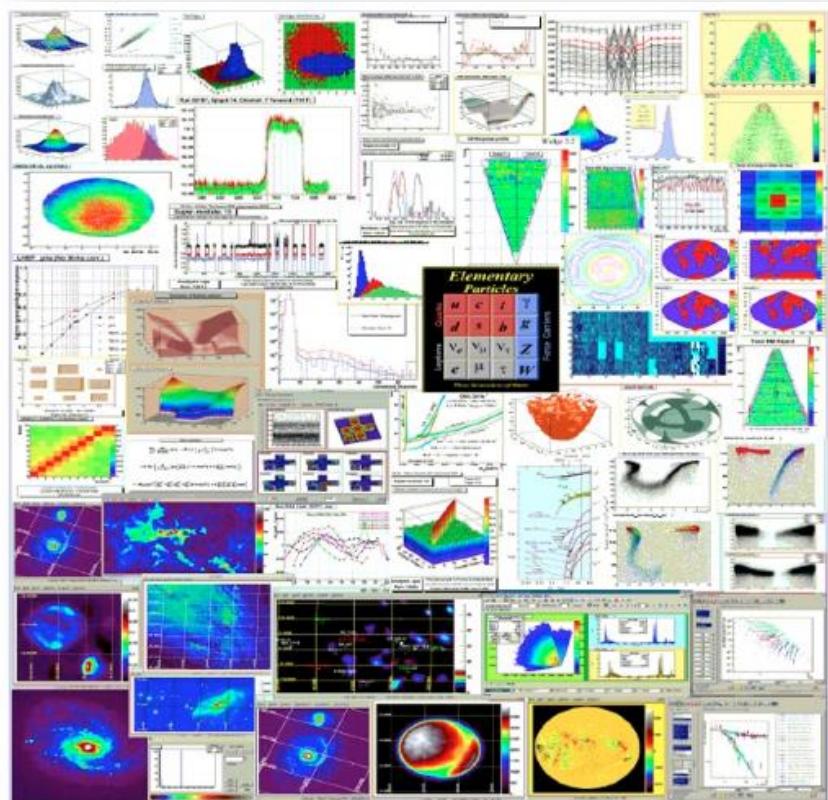
# 2D Graphics



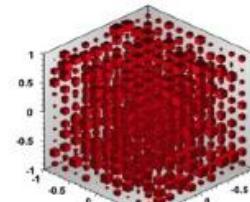
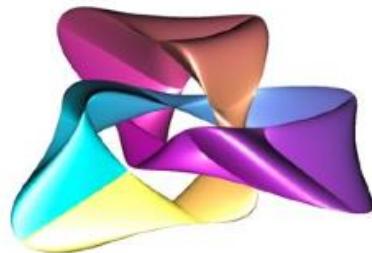
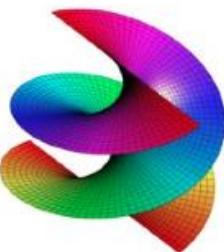
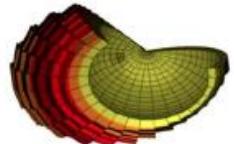
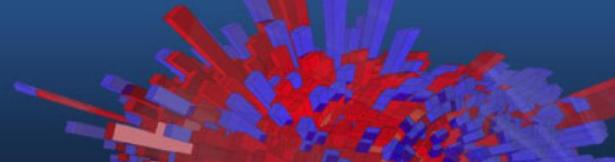
New functionalities added at every new release

Always requests for new style of plots

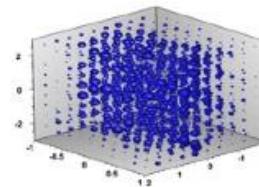
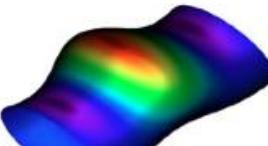
Can save graphics in many formats: *ps*, *pdf*,  
*svg*, *jpeg*, *LaTex*, *png*, *c*, *root* ...



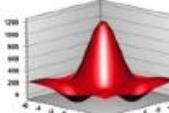
# 3D Graphics



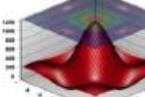
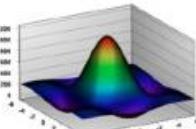
TGLParametric



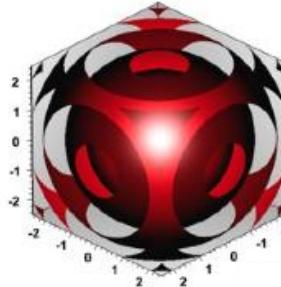
"LEGO"



"SURF"



TF3



# Other ROOT Features

## Geometry Toolkit

- Represent geometries as complex as LHC detectors

## Event Display (EVE)

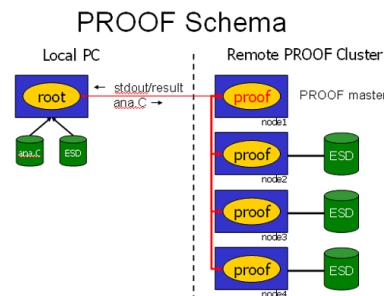
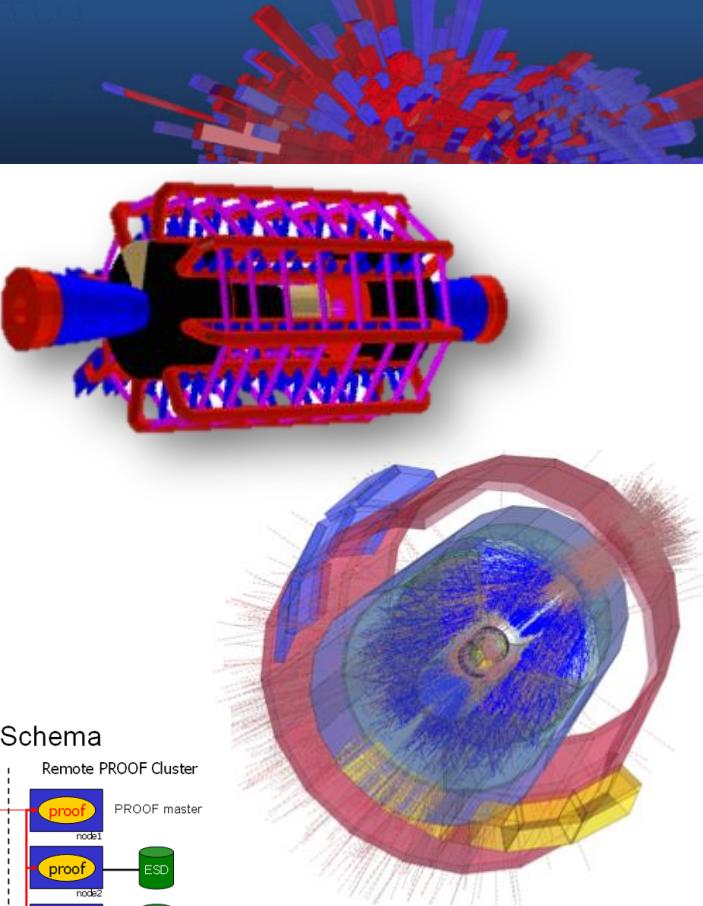
- Visualise particles collisions within detectors

## Multiproc

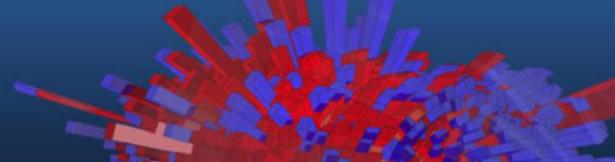
- Parallel shared-memory Map-Reduce

## 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



# The SWAN Service



**Data analysis with ROOT “as a service”**

*Interface:* Jupyter Notebooks

*Goals:*

- Use ROOT only with a web browser
  - Platform independent ROOT-based data analysis
  - Calculations, input and results “in the Cloud”
- Allow easy sharing of scientific results: plots, data, code
  - Through your CERNBox
- Simplify teaching of data processing and programming



<http://swan.cern.ch>

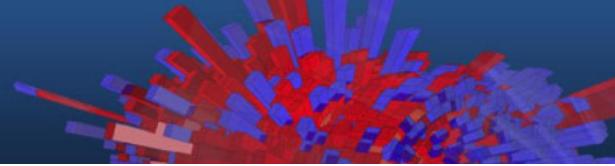
## ROOT web site: **the** source of information and help for ROOT users

- For beginners and experts
- Downloads, installation instructions
- Documentation of all ROOT classes
- Manuals, tutorials, presentations and more
- Forum
- ...

We propose to do a quick tour of the web site  
Don't hesitate to use it, even today!

The screenshot shows the official ROOT website at [root.cern](http://root.cern). The header features the ROOT logo and navigation links for Download, Documentation, News, Support, About, Development, and Contribute. Below the header are four icons: Getting Started, Reference Guide, Forum, and Gallery. A central section titled "ROOT is ..." provides a brief overview of the software. A "Download" button is highlighted with a red circle. The "Under the Spotlight" section highlights recent news items. The footer contains a sitemap and links to various project components like Documentation, News, Support, and Development.

# A Few Q/A



**? What could be the advantage of learning this software technology ?**

**! 1.** You have all the tools to process, store, analyse and visualise data in one single kit.

**! 2.** You join a huge users' community, and a very supportive team of core developers

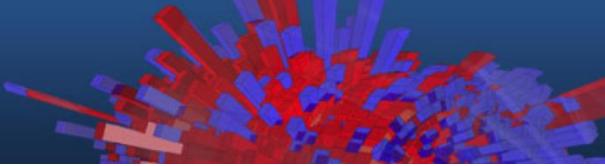
**? Why C++ and not a scripting language ?**

**! Performance.** Support for languages like Python

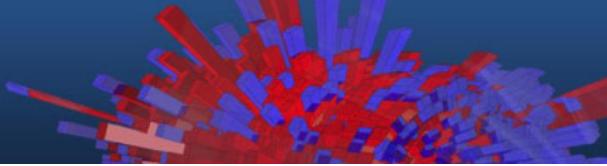
**? Why prompt and libraries instead of a GUI ?**

**! ROOT** is a programming framework, not an office suite.

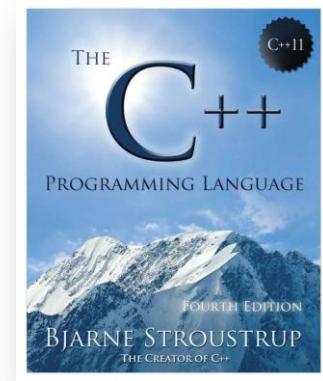
# C++ From 10.000 Km



# C++ From 10.000 Km



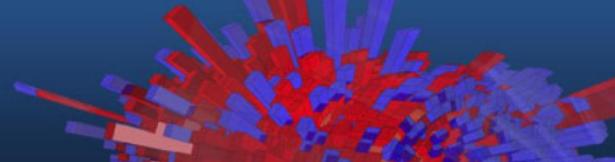
- Compiled, strongly typed language, allows to get the best performance from the hardware
- Allows object orientation
- Generic programming: Templates
- Explicit memory management (pointers)



Main language of HEP (together with Python)

- In the 90s nearly all legacy FORTRAN HEP code has been migrated to C++
- Reduce costs of management of large codebases (millions of lines of code)
- Allow groups of hundreds of active developers

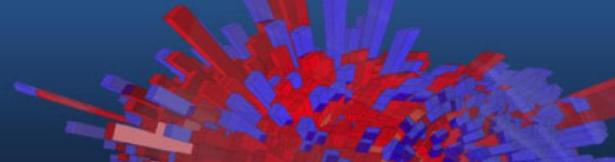
# Some Useful Terms



- A “class” is an entity which encapsulate “data” and “actions” on it
- The “data” is represented by the *data members* (“variables of the class”)
- The “actions” are expressed by the *class methods* (“functions of the class”)
- One *calls/invokes* a method which can have zero or more arguments
- An *object* is an instance of a *class*
- An object is created by a special method, the *constructor*. There can be more than one constructor, e.g.:
  - `TH1F histo = TH1F(); // default constructor`
  - `TH1F histo = TH1F("histName", "HistTitle", 64, 0, 64); // with params`

Note: the language is somehow approximate but certainly ok for this lecture

# -> and .



The *dot* and *arrow operators* are used to access methods and members of objects and pointers to objects

- *Dot*: to access methods and members of objects
- *Arrow*: to access methods and members of pointers to objects

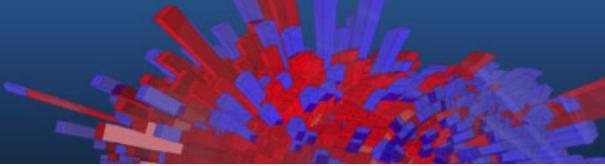
Example:

```
MyClass myClassInstance("myName");  
myClassInstance.GetName();
```

```
MyClass *myClassInstancePtr = new MyClass ("myName");  
myClassInstancePtr->GetName();
```

Note: the language is somehow approximate but certainly ok for this lecture

# ROOT Basics

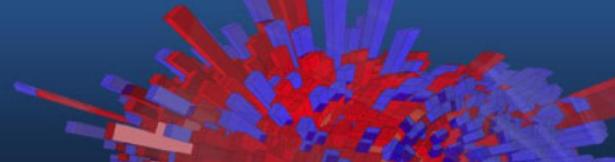


- ROOT as a Calculator
- ROOT as Function Plotter
- Plotting Measurements
- Histograms
- Interactive ROOT Section

# Let's Fire Up ROOT



# The ROOT Prompt



C++ is a compiled language

- A compiler is used to translate source code into machine instructions

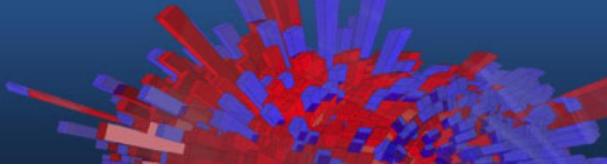
ROOT provides a C++ **interpreter**

- Interactive C++, without the need of a compiler, like Python, Ruby, Haskell ...
- Allows reflection (inspect at runtime layout of classes)
- Is started with the command:

`root`

- The interactive shell is also called “ROOT prompt” or “ROOT interactive prompt”

# ROOT As a Calculator



ROOT interactive prompt can be used as an advanced calculator !

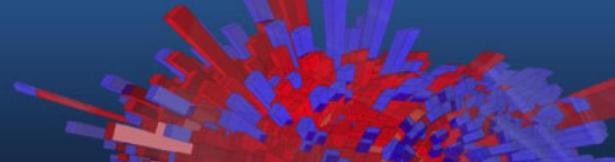
```
root [0] 1+1  
(int)2  
root [1] 2*(4+2)/12.  
(double) 1.00000  
root [2] sqrt(3.)  
(double) 1.73205  
root [3] 1 > 2  
(bool) false
```

Try it!

ROOT allows not only to type in **C++ statements**, but also advanced **mathematical functions**, which live in the TMath namespace.

```
root [4] TMath::Pi()  
(Double_t) 3.14159  
root [5] TMath::Erf(.2)  
(Double_t) 0.222703
```

# ROOT As a Calculator++



Here we make a step forward.

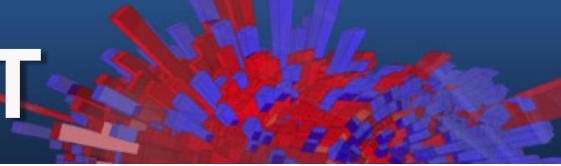
We Declare **variables** and used a *for* control structure.

Tab-completion is available. Try it.

```
root [6] double x=.5
(double) 5.00000
root [7] int N=30
(int) 30
root [8] double gs=0
(double) 0.00000
```

```
root [9] for (int i=0;i<N;++i) gs += TMath::Power(x,i)
root [10] TMath::Abs(gs - (1-TMath::Power(x,N-1))/(1-x))
(Double_t) 1.862645e-09
```

# Interlude: Controlling ROOT



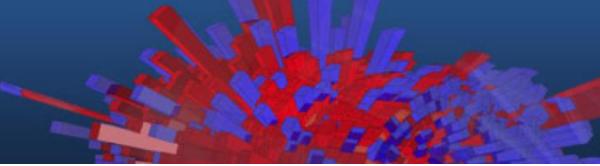
Special commands which are not C++ can be typed at the prompt, they start with a “.”

```
root [1] .<command>
```

For example:

- To quit root use `.q`
- To issue a shell command use `.!<OS_command>`
- To load a macro use `.L <file_name>` (see following slides about macros)
- `.help` or `.?` gives the full list

# Exercise

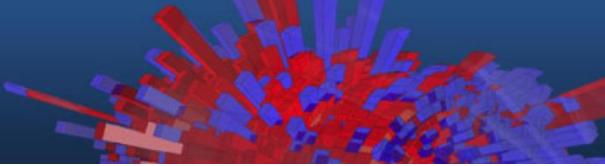


For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine.

```
root [0] double x=0
root [1] exp(-x*x*.5) - TMath::Gaus(x)
[...]
```

For one number

# Exercise Solution



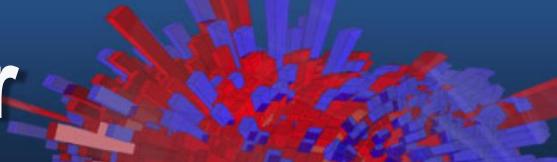
For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine.

```
root [0] double x=0
root [1] exp(-x*x*.5) - TMath::Gaus(x)
[...]
```

Many possible ways of solving this! E.g:

```
root [0] for (auto v : {0.,1.,10.,20.}) cout << v << " " << exp(-
v*v*.5) - TMath::Gaus(v) << endl
```

# ROOT As a Function Plotter



The class TF1 represents one dimensional functions (e.g.  $f(x)$  ):

```
root [0] TF1 f1("f1","sin(x)/x",0.,10.); //name, formula, min, max  
root [1] f1.Draw();
```

An extended version of this example is the definition of a function with parameters:

```
root [2] TF1 f2("f2","[0]*sin([1]*x)/x",0.,10.);  
root [3] f2.SetParameters(1,1);  
root [4] f2.Draw();
```

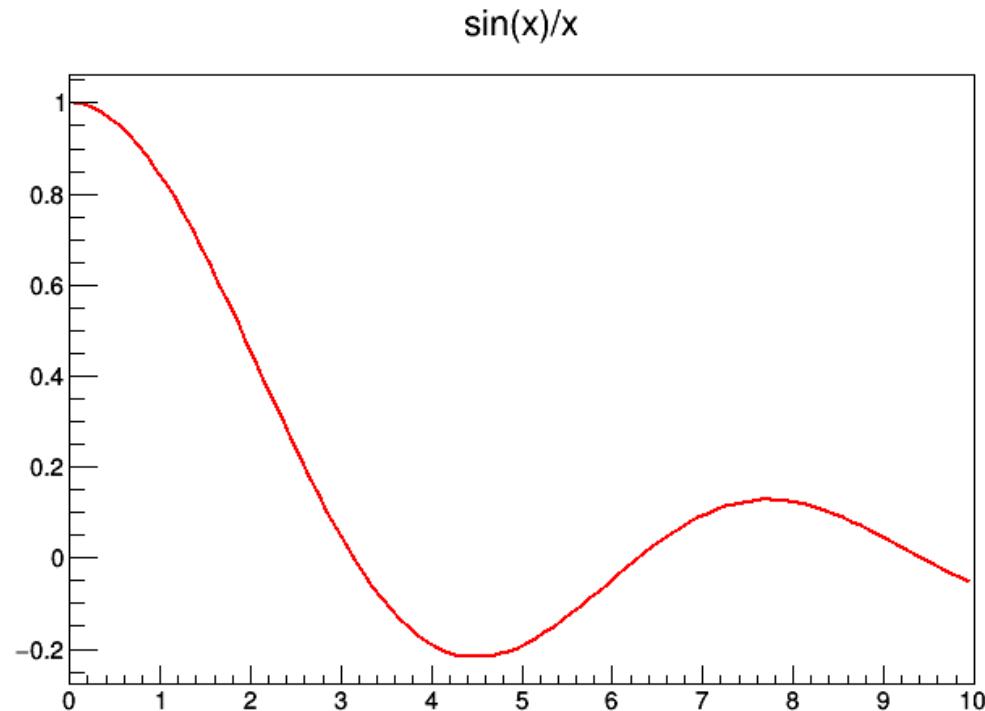
# ROOT As a Function Plotter

The class TF1

```
root [0] TF1  
root [1] f1.D
```

An extended version  
with parameters

```
root [2] TF1  
root [3] f2.  
root [4] f2.
```

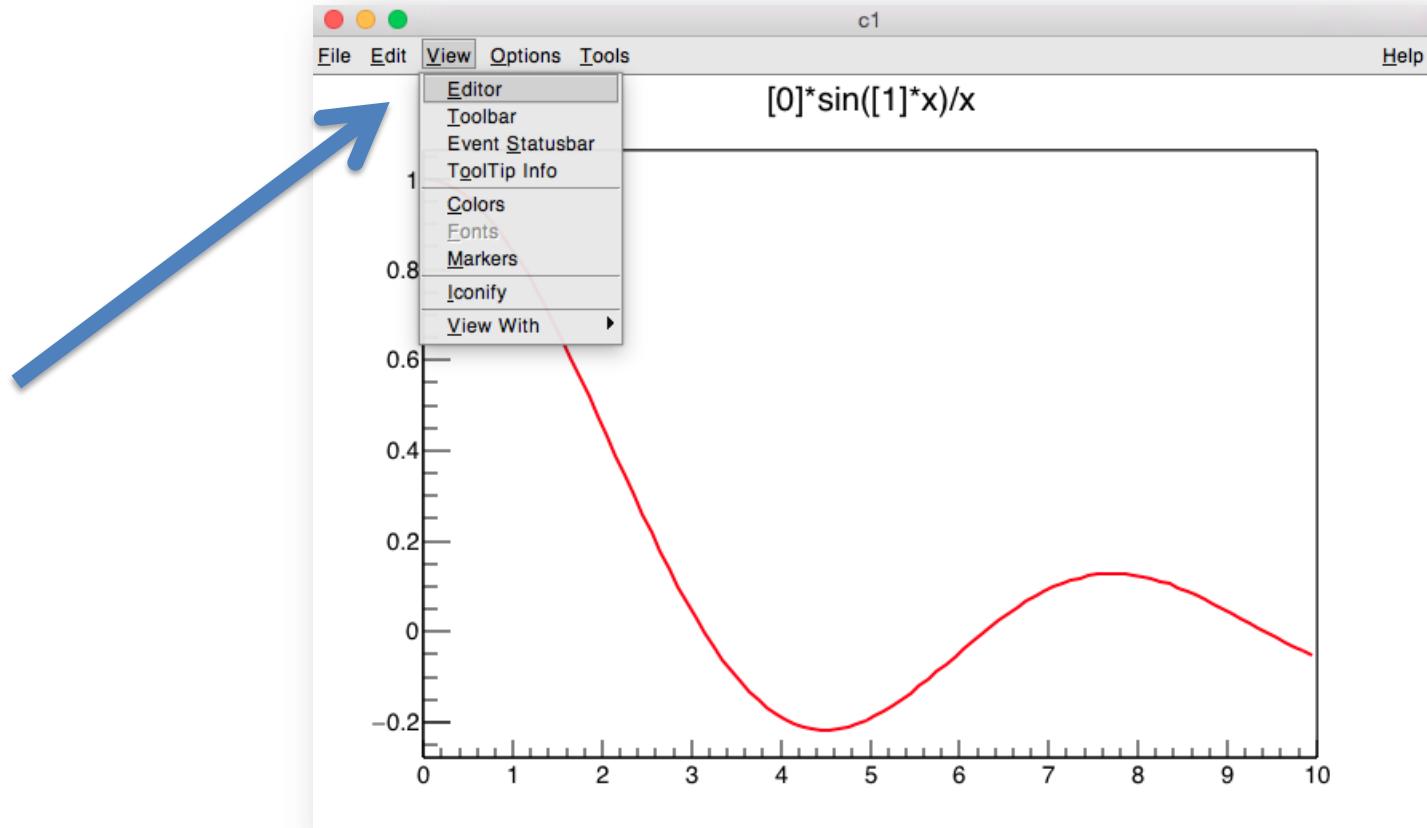


:

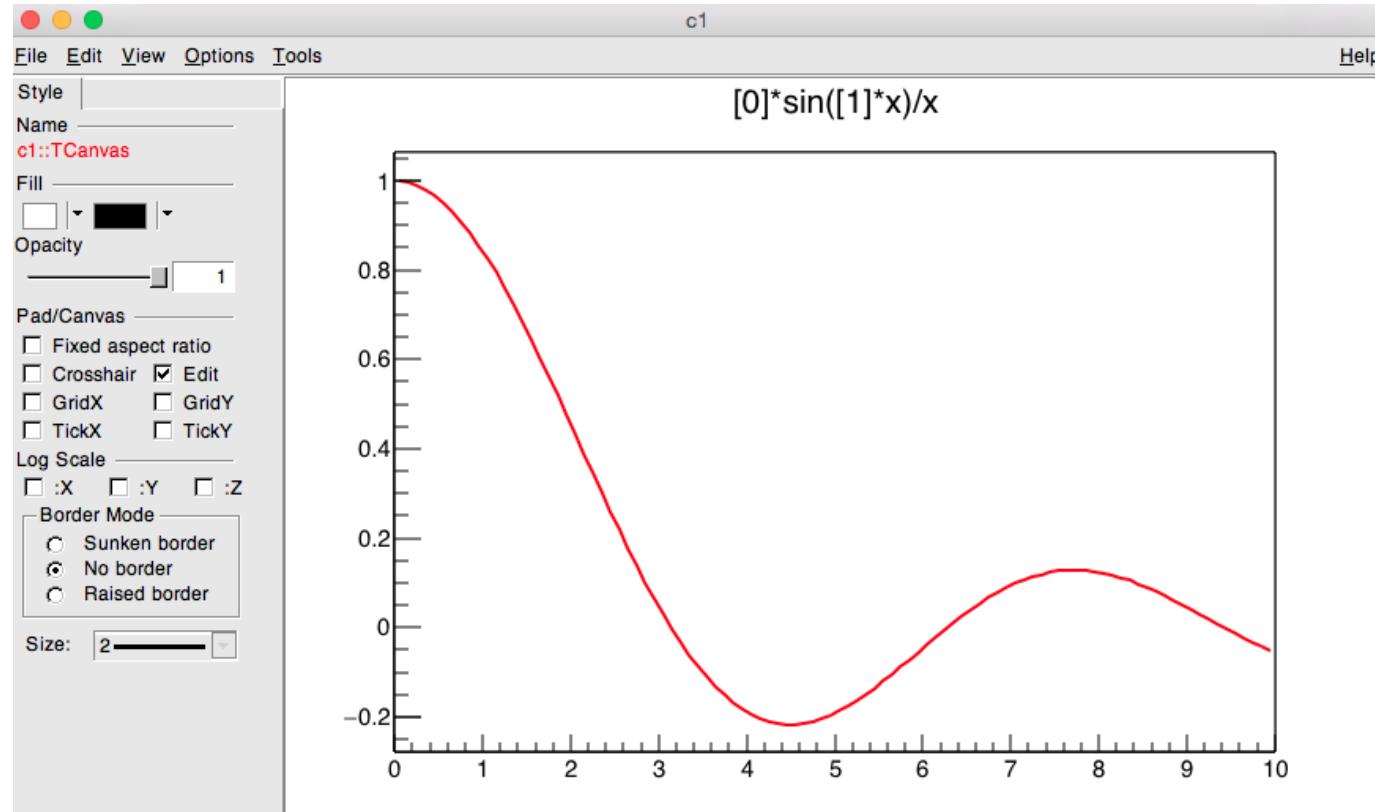
```
min, max
```

on

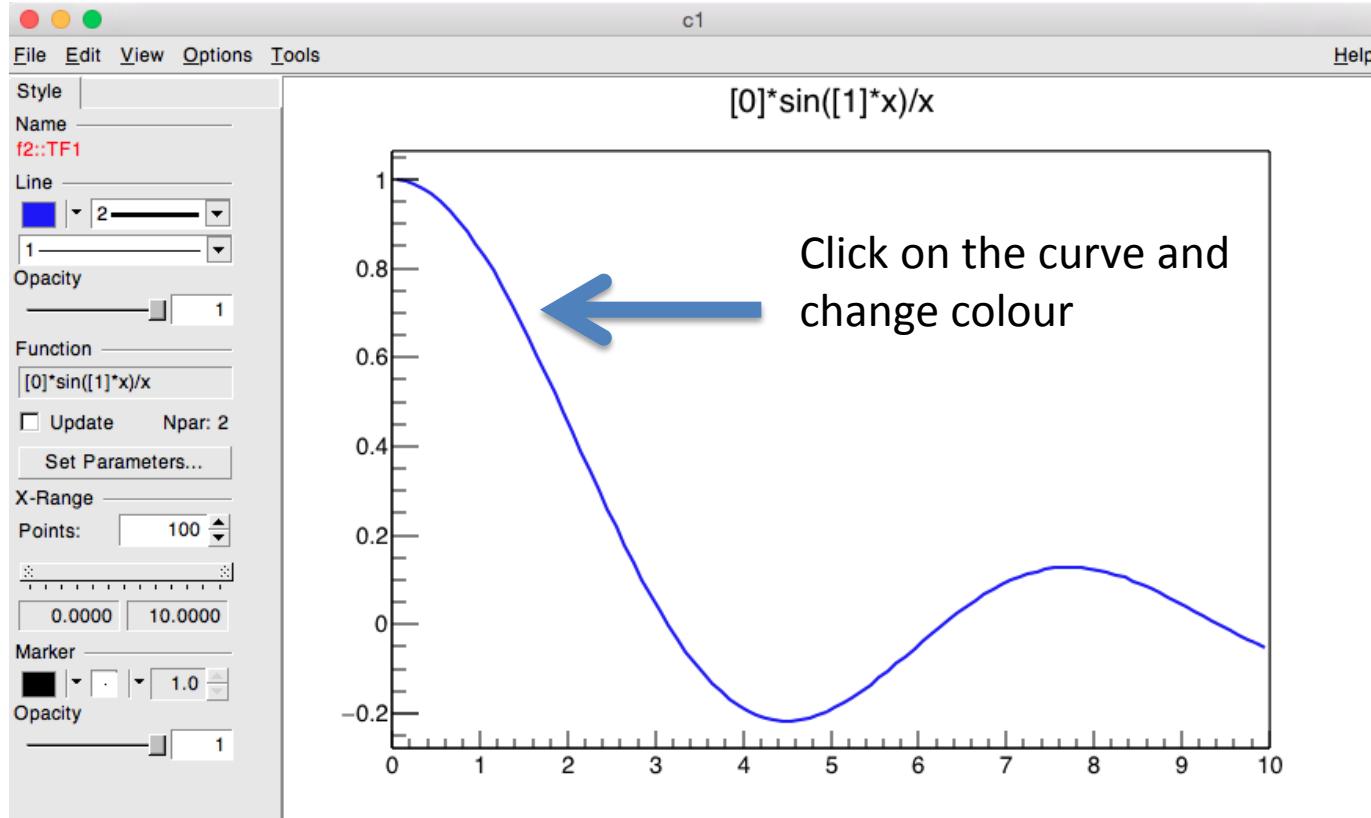
# Exercise: Interaction With The Plot



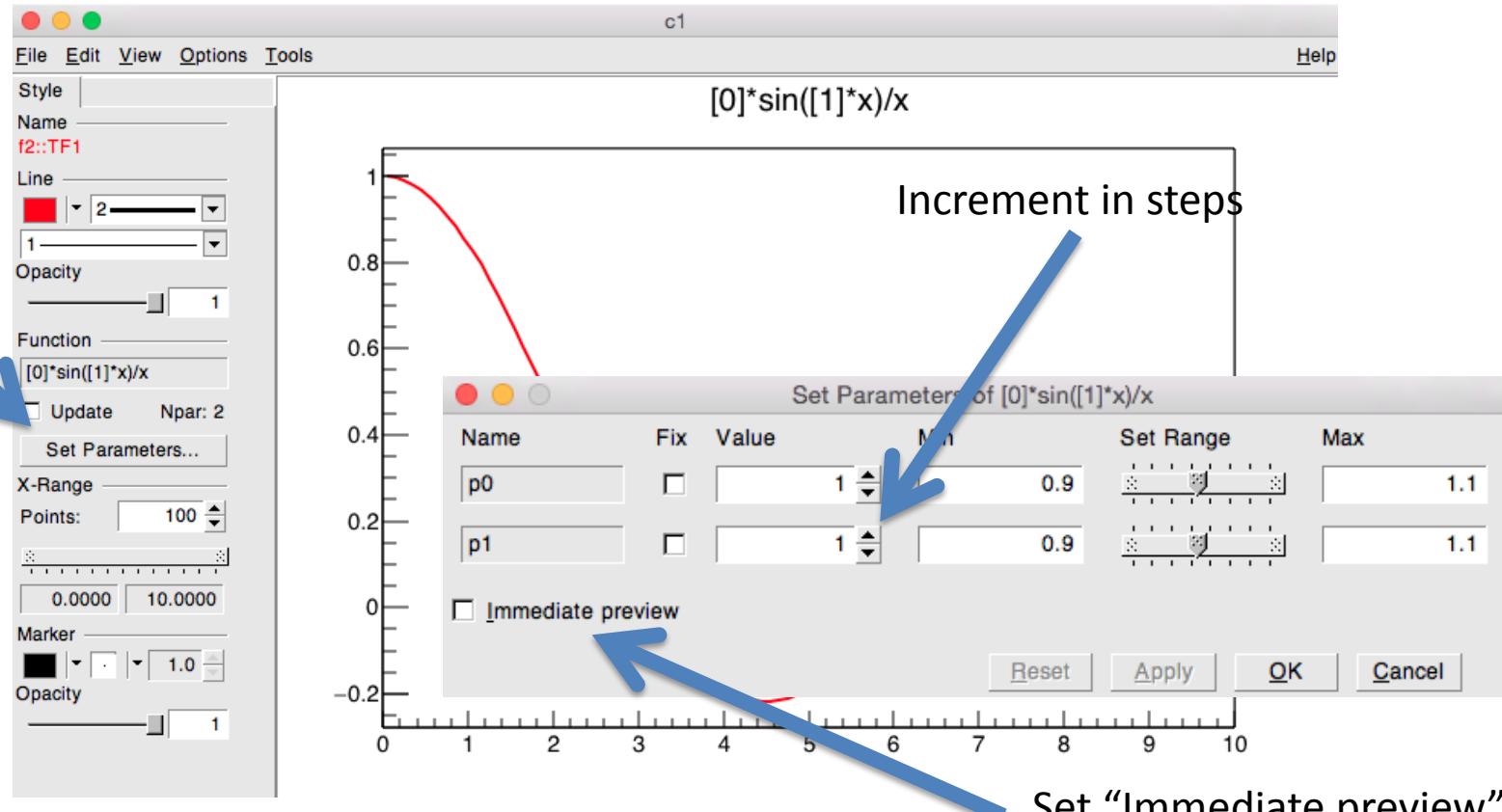
# Exercise: Interaction With The Plot



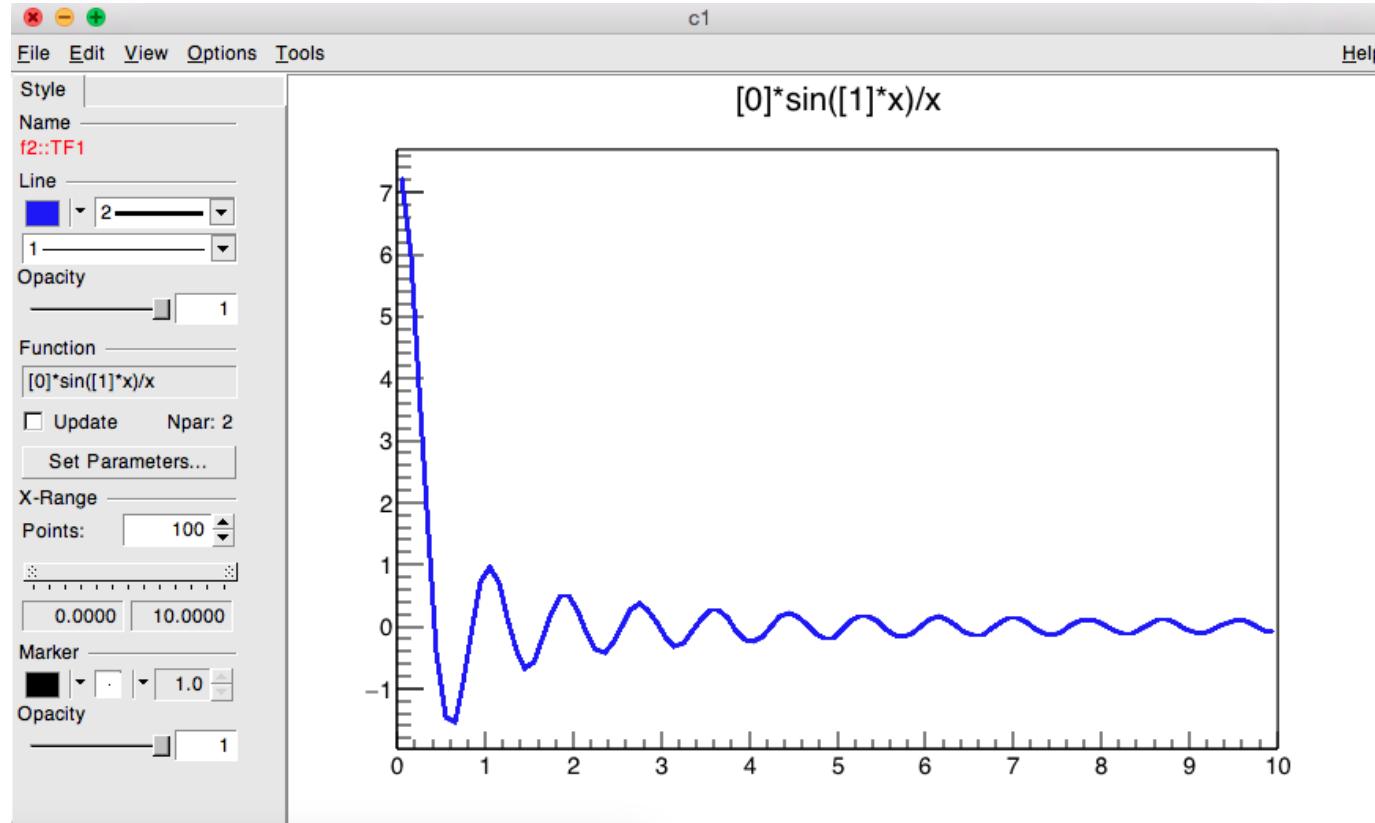
# Exercise: Interaction With The Plot



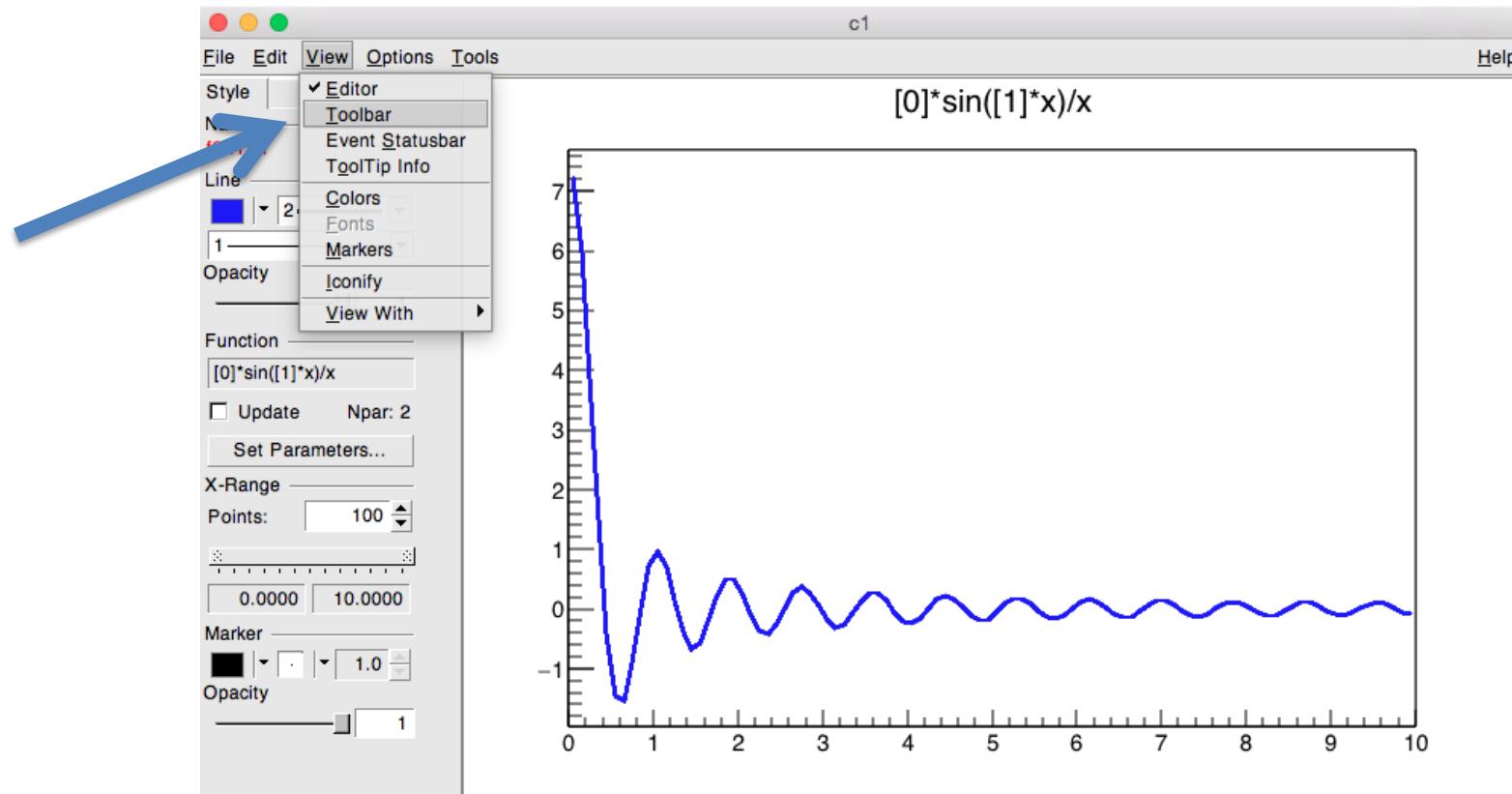
# Exercise: Interaction With The Plot



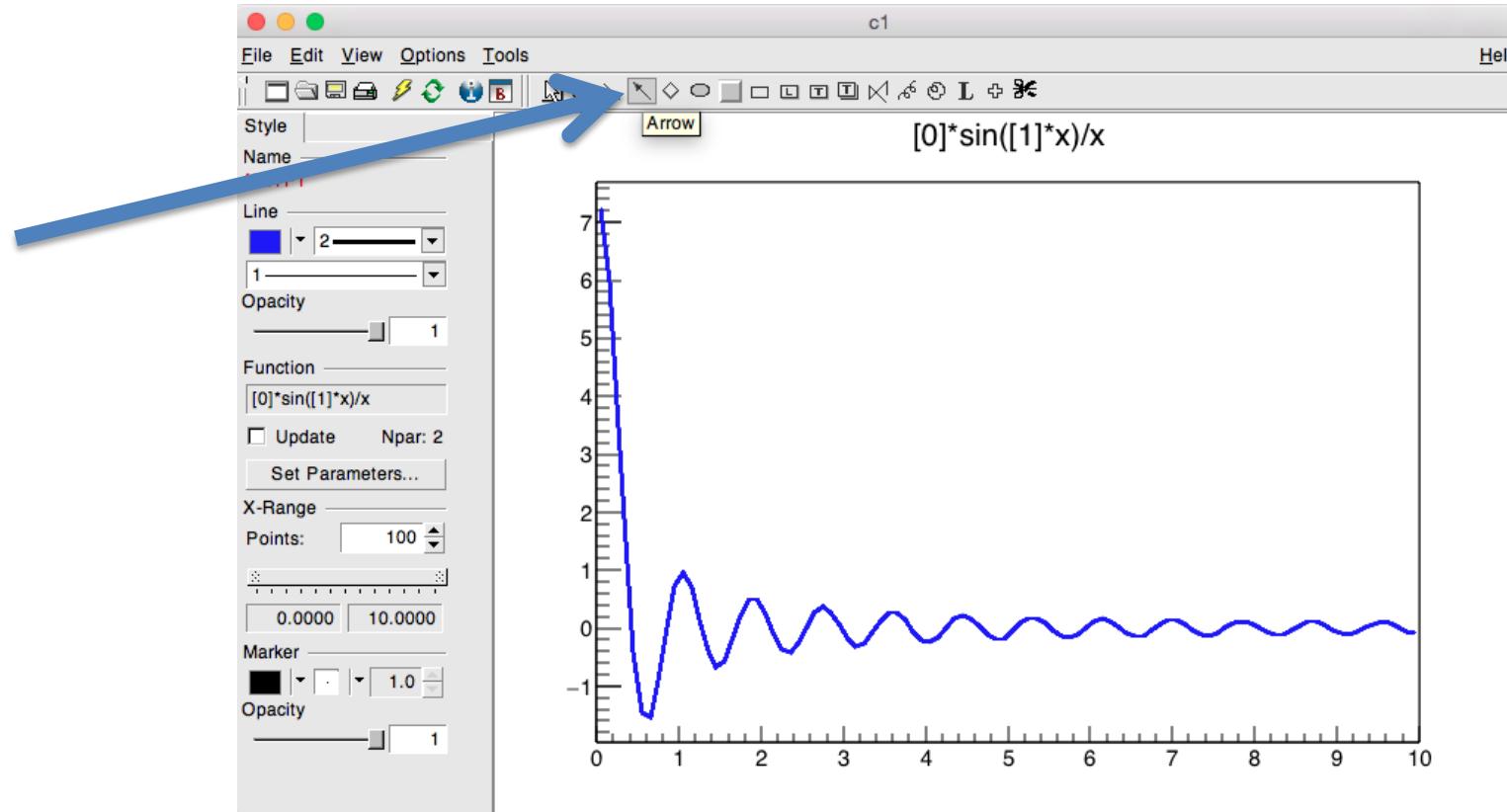
# Exercise: Interaction With The Plot



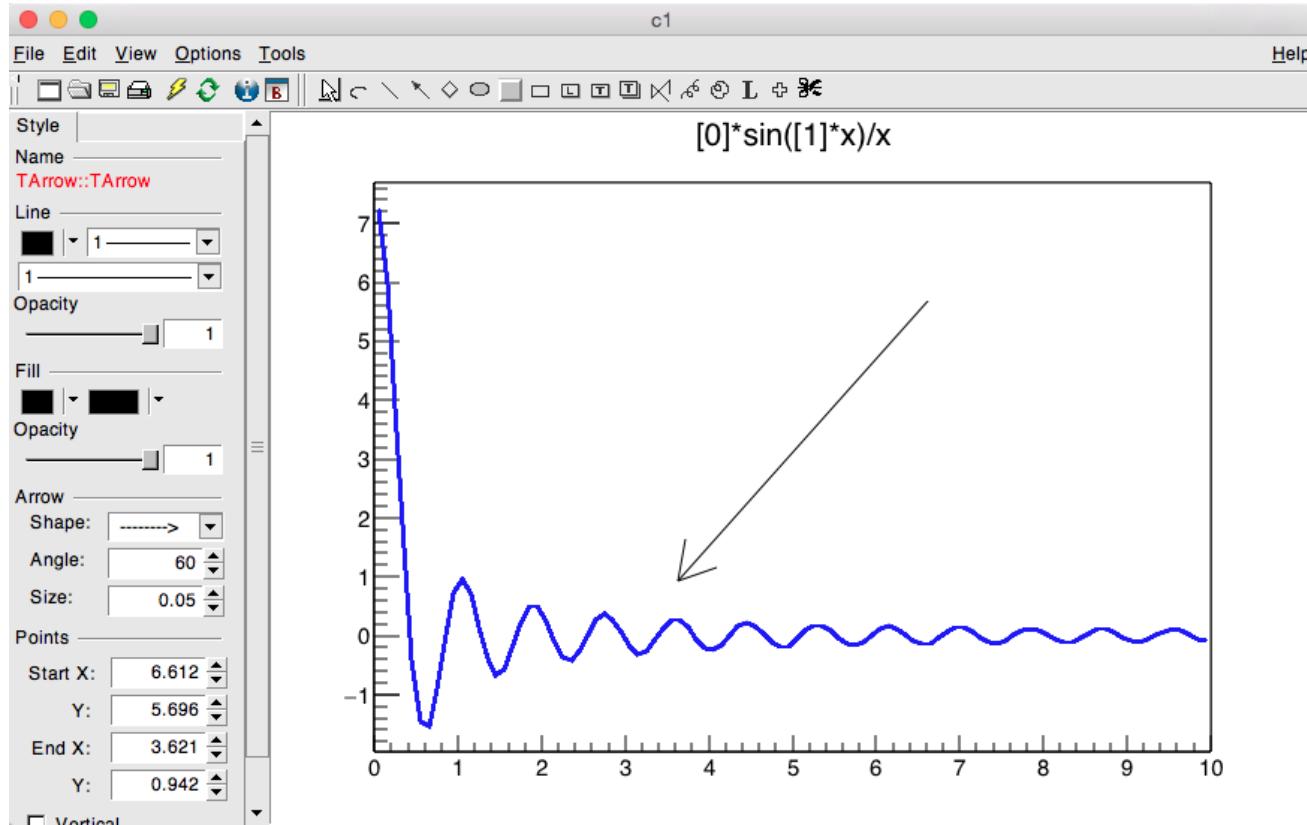
# Exercise: Interaction With The Plot



# Exercise: Interaction With The Plot

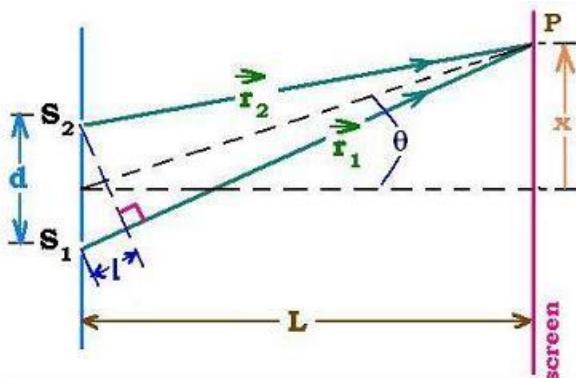


# Exercise: Interaction With The Plot

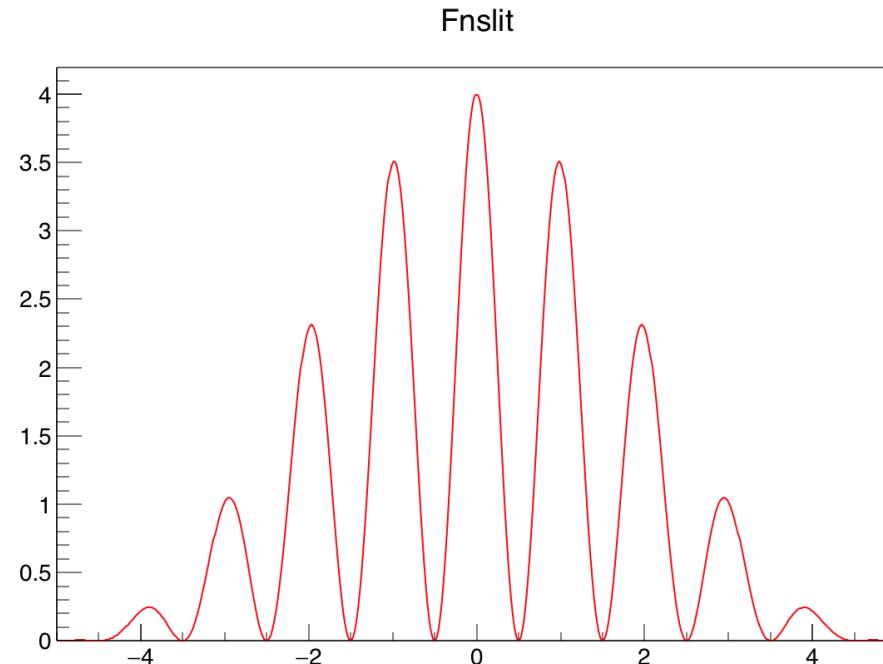


# ROOT As a Function Plotter

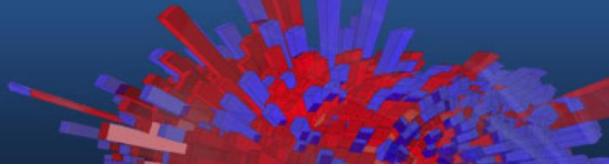
The example **slits.C**, detailed in the Primer, is a more complex C++ program calculating and displaying the interference pattern produced by light falling on a multiple slit.



$$L \gg d \Rightarrow \text{Lines from each slit to } P \text{ are parallel}$$
$$\Rightarrow \sin \theta = \frac{x}{L} = \frac{1}{d}$$



# Now Try It in SWAN!



<http://swan.web.cern.ch/content/root-primer>



Click to open the  
Slits notebook in  
SWAN

Slits: Double slits diffraction figure

```
In [5]: TF1 Fnslit("Fnslit",&slit,-5.001,5.,2);
Fnslit.SetN(500);

Here you can see that we are setting the parameters for the code, so that we can apply them as fixed variables that we can reuse in the rest of our code.
This saves us the trouble of entering long numbers over and over again every time we want to run the code.

In [6]: Fnslit.Draw();
```

Now that we've got our code nicely written out, we can use the "Draw" command so that we can see the graph.

```
In [7]: TCanvas c;
Fnslit.Draw();
c.Draw();
```

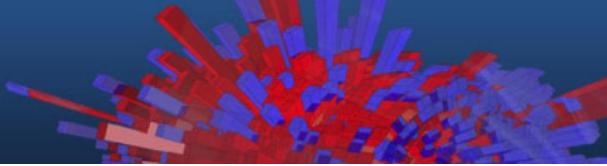
Fnslt

Open in SWAN



Slits\_cpp

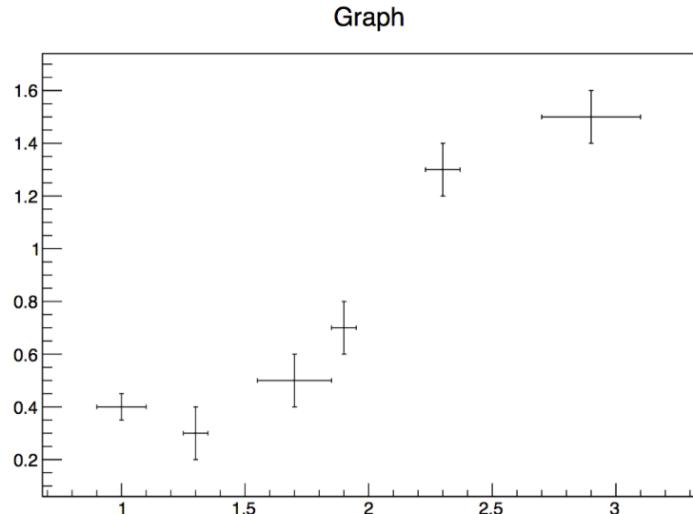
# Plotting Measurements



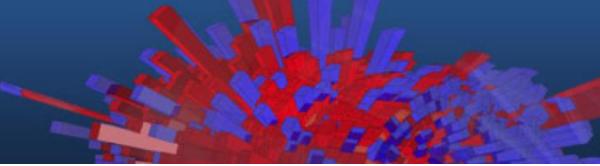
The class `TGraphErrors` allows to display measurements, including errors, with different types of constructors. In the following example, data are taken from the file `ExampleData.txt`:

```
root [0] TGraphErrors gr("ExampleData.txt");
root [1] gr.Draw("AP");
```

Tells ROOT to draw the **AxIs** and the **PoInts**

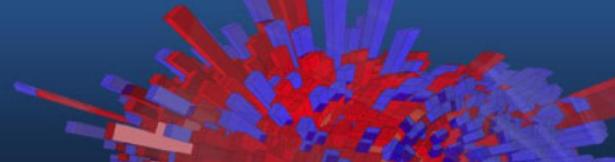


# Exercise: TGraph

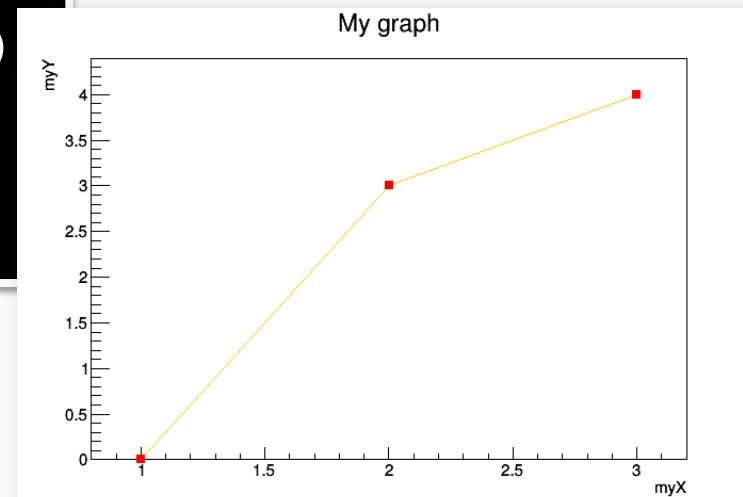


- Create a graph (TGraph)
- Set its title to “My graph”, its X axis title to “myX” and Y axis title to “myY”
- Fill it with three points: (1,0), (2,3), (3,4)
- Set a red full square marker
- Draw an orange line between points

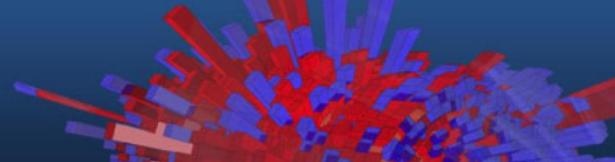
# Exercise Solution



```
root [0] TGraph g
root [1] g.SetTitle("My graph;myX;myY")
root [2] g.SetPoint(0,1,0)
root [3] g.SetPoint(1,2,3)
root [4] g.SetPoint(2,3,4)
root [5] g.SetMarkerStyle(kFullSquare)
root [6] g.SetMarkerColor(kRed)
root [7] g.SetLineColor(kOrange)
root [8] g.Draw("APL")
```

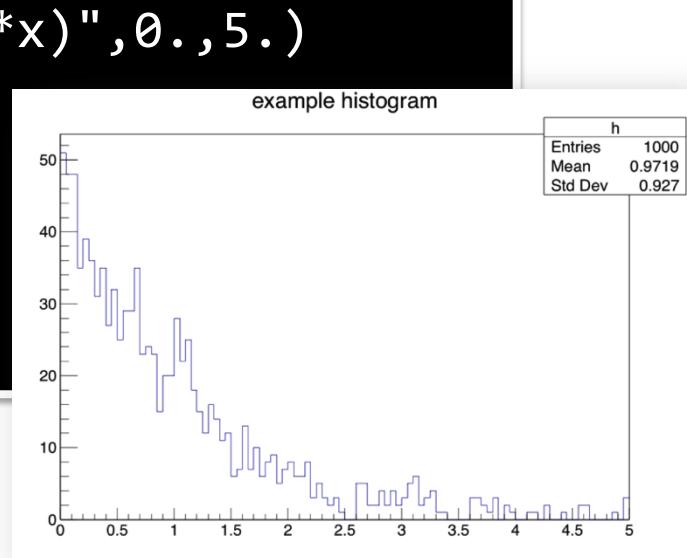


# Histograms

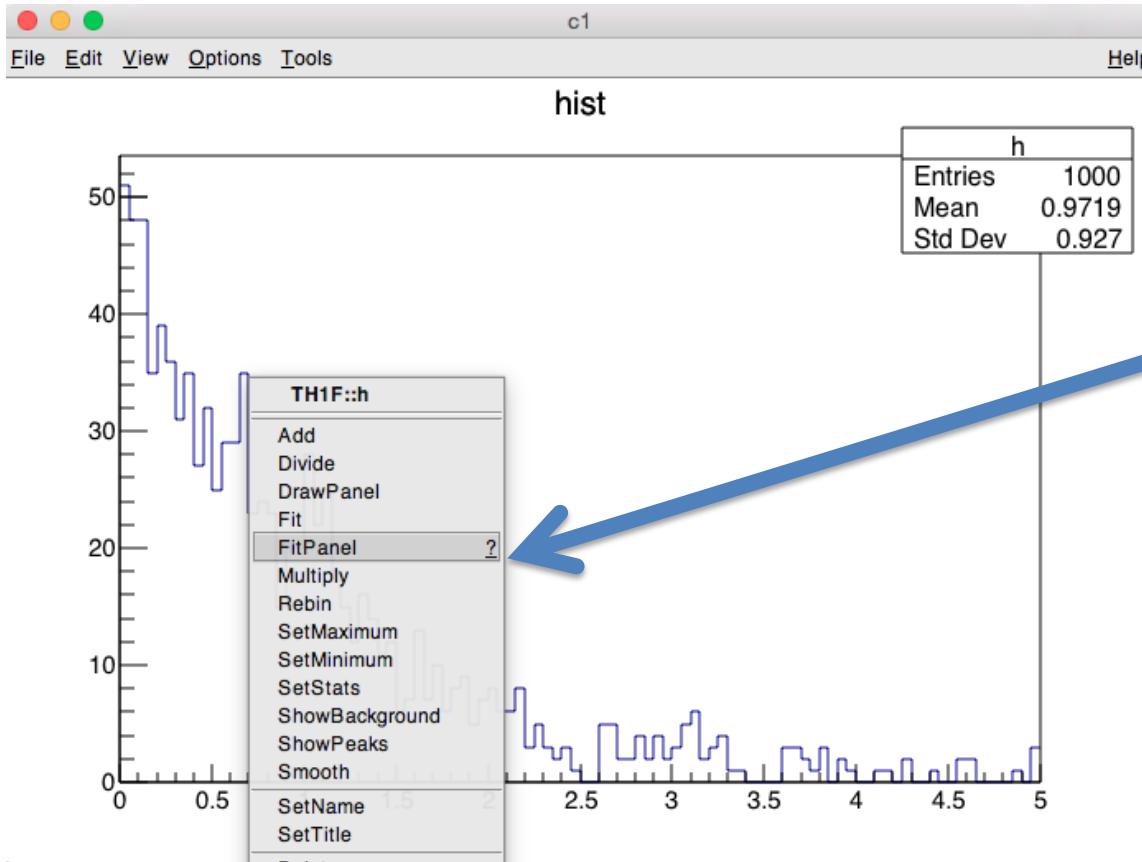
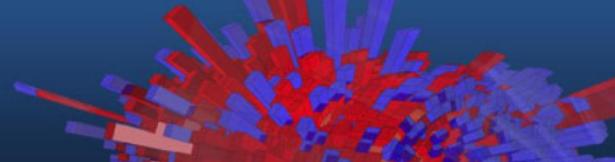


- The TH\* classes represent histograms
- TH1\* are monodimensional, TH2\* are bidimensional ...
- The final letter describes the type stored in each bin:  
A double in TH1D, a float in TH1F ...

```
root [0] TF1 efunc("efunc","exp([0]+[1]*x)",0.,5.)
root [1] efunc.SetParameters(1,-1)
root [2] TH1F h("h","hist",100,0.,5.)
root [3] for (int i=0;i<1000;i++)
h.Fill(efunc.GetRandom())
root [4] h.Draw()
```

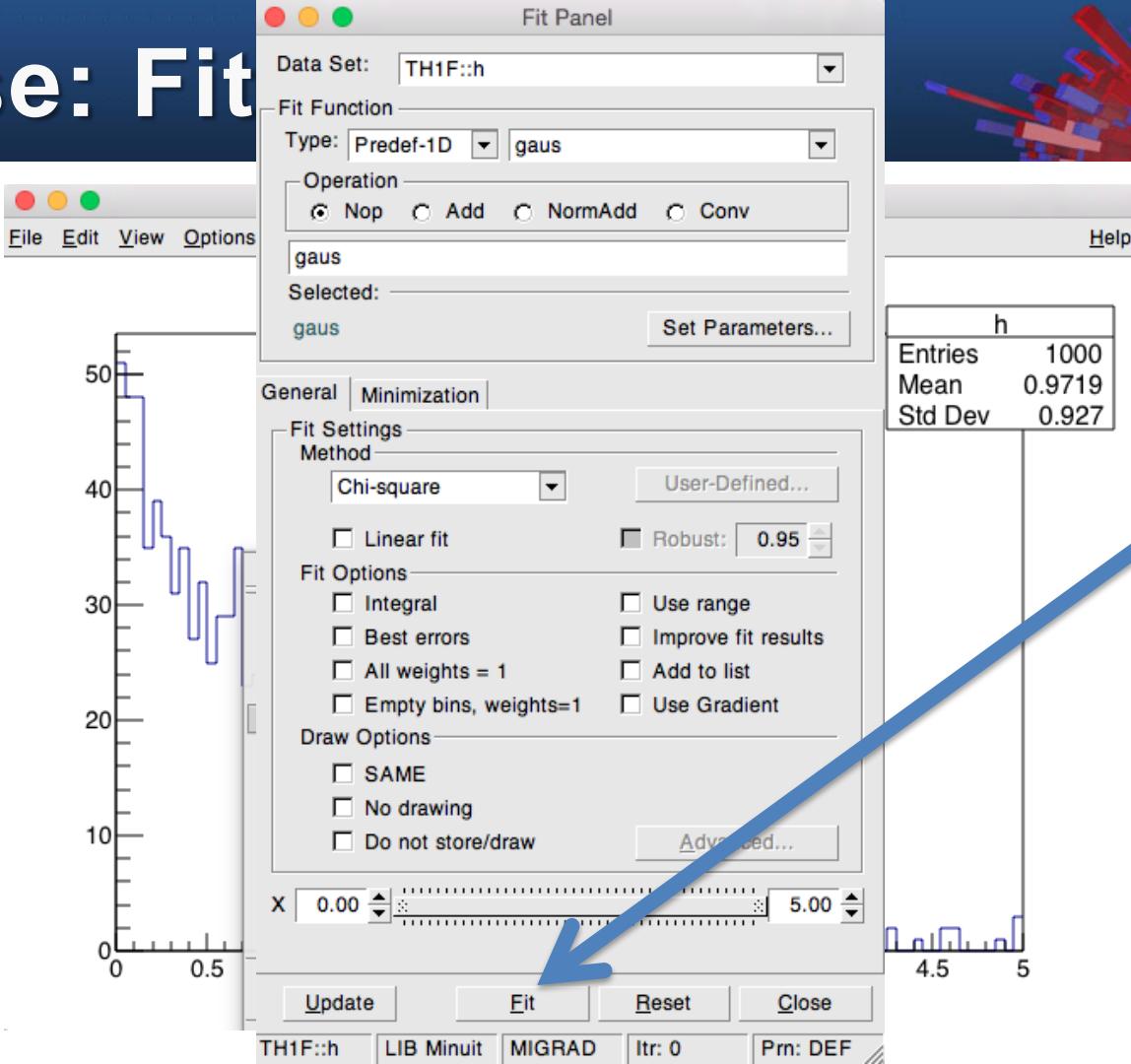


# Exercise: Fitpanel

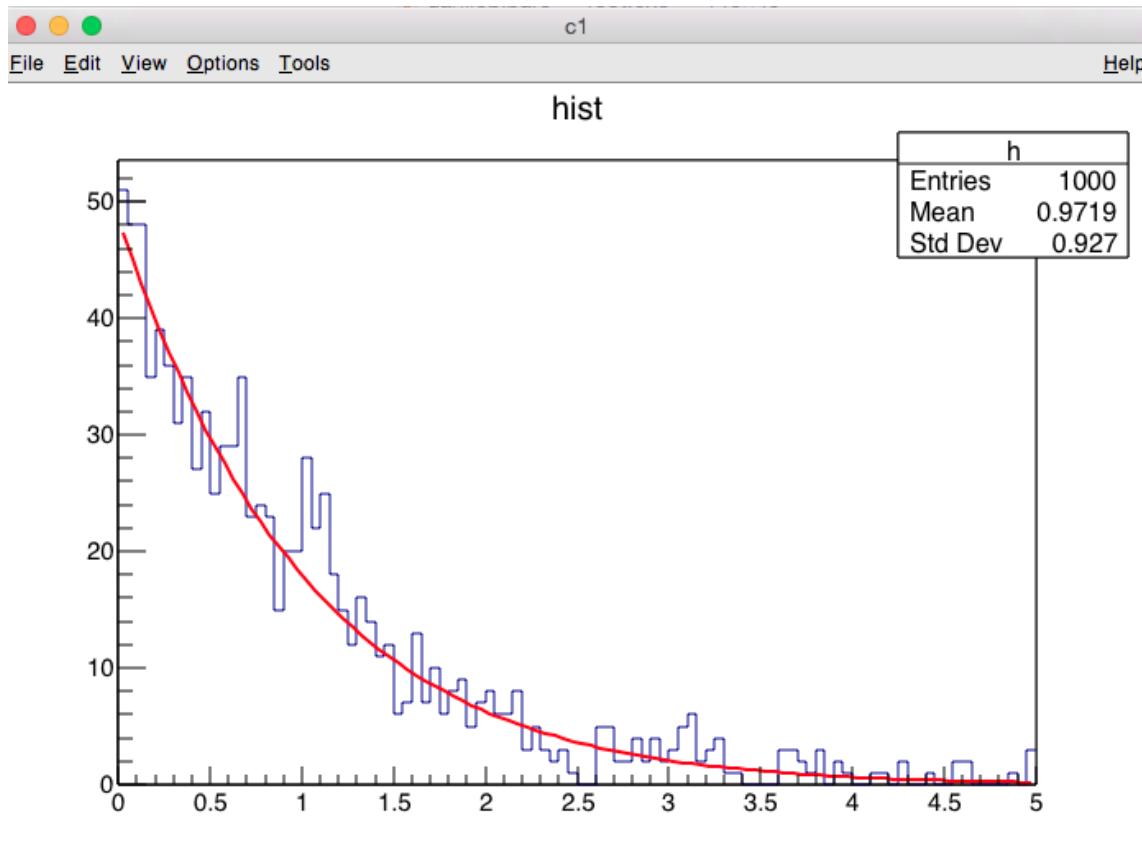
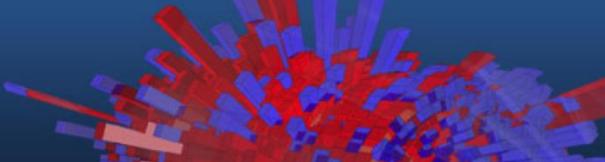


Click on the  
histogram “line”

# Exercise: Fit



# Exercise: Fitpanel

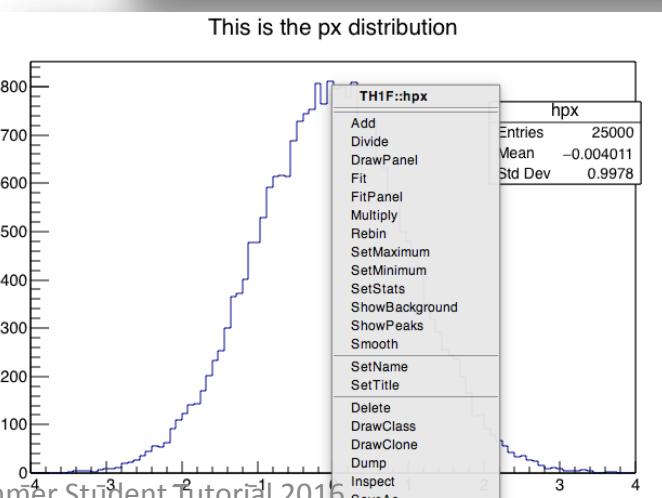
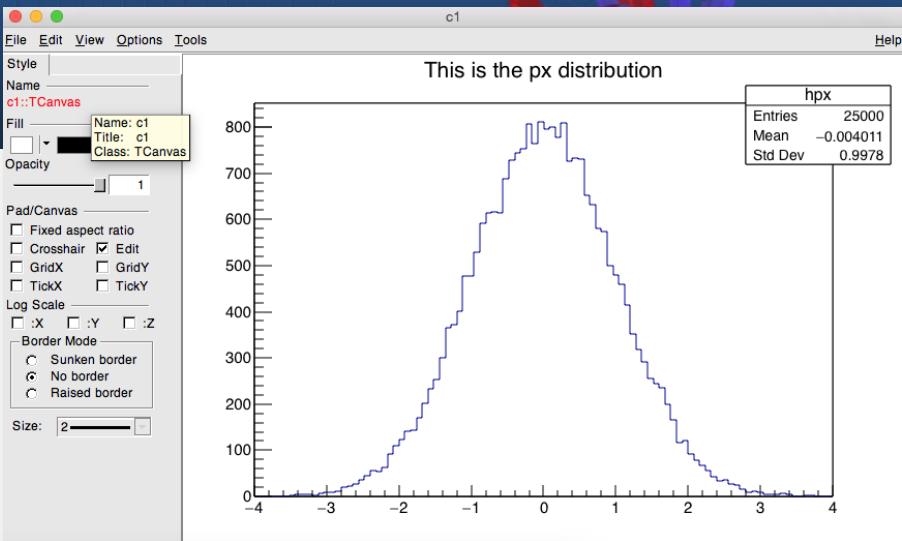
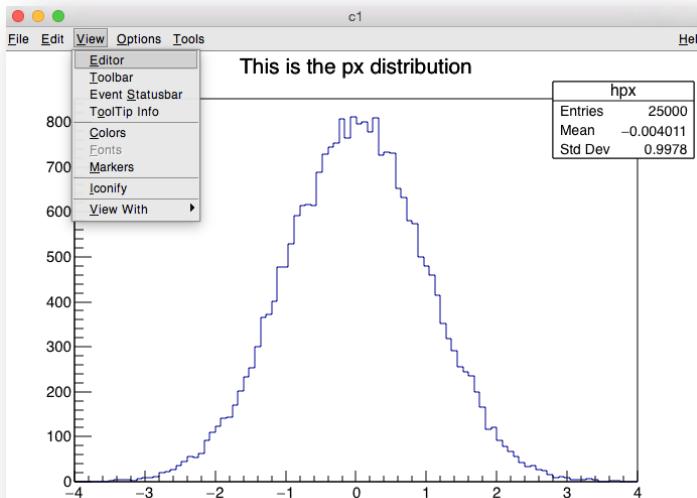


# Interactive ROOT

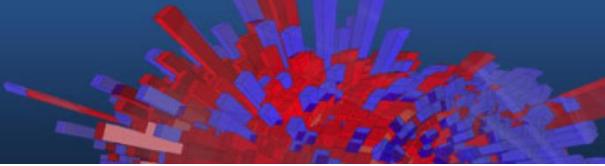
Look at one of your plots again and move the mouse across.

You will notice that this is much more than a static picture.

Try to interact with objects and manipulate them.

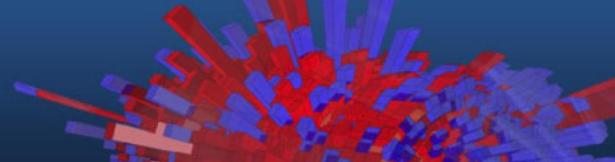


# ROOT Macros



- General Remarks
- A more complete example
- Summary of Visual effects
- Interpretation and Compilation

# General Remarks



We have seen how to interactively type lines at the prompt.

The next step is to write “ROOT Macros” – lightweight programs

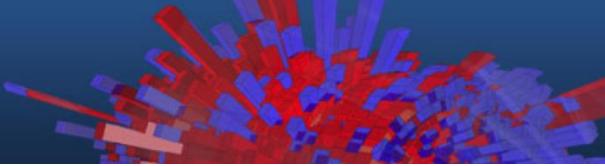
The general structure for a macro stored in file *MacroName.C* is:

**Function, no main, same  
name as the file**

```
void MacroName() {  
    <           ...  
    your lines of C++ code  
    ...           >  
}
```



# Running a Macro



The macro is executed at the system prompt by typing:

```
> root MacroName.C
```

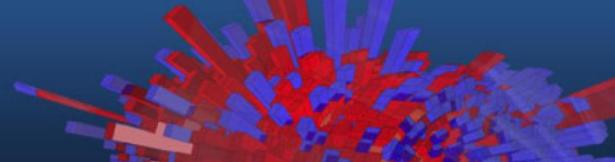
or executed at the ROOT prompt using .x:

```
> root  
root [0] .x MacroName.C
```

or it can be loaded into a ROOT session and then be executed by typing:

```
root [0].L MacroName.C  
root [1] MacroName();
```

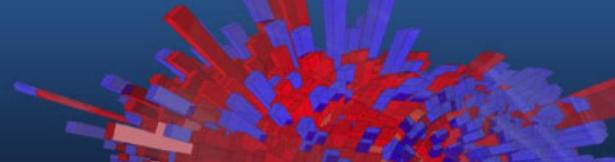
# A More Complex Example



The example in section 3.2 of the ROOT primer, is a typical task in data analysis, a macro that constructs a graph with errors, fits a (linear) model to it and saves it as an image.

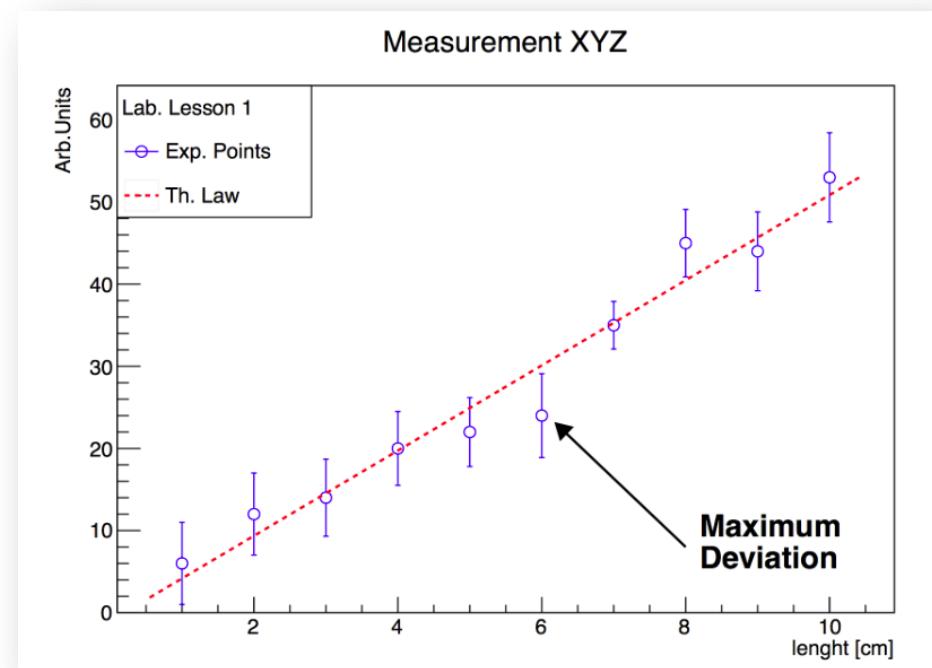
Let's inspect it together.

# A More Complex Example

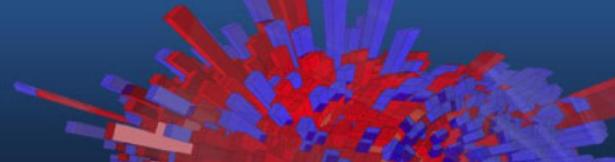


And Run it!

```
> root macro1.C
```



# Macros available in SWAN



<http://swan.web.cern.ch/content/root-primer>

## ROOT Primer

Click to open the Macro 1 notebook in SWAN



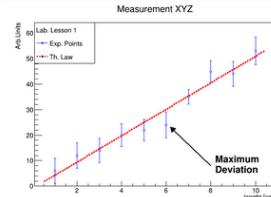
Macro1\_cpp

### Macro 1: Building a graph with errors

```
In [10]: TArrow arrow9, 8, 6.2, 23, 0.02, ">";  
arrow9.SetAngle(23);  
arrow9.DrawLine();  
  
Add some text to the plot.
```

```
In [11]: TLatex text(2.2, 7.5, "#exp{line}(Maximum)(Deviation)");  
text.DrawClose();
```

```
In [12]: g->Canvas->Draw();
```

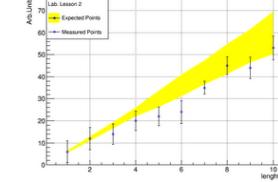


Open In SWAN

### Macro 2: Building a graph from a file

```
In [5]: c->Draw();
```

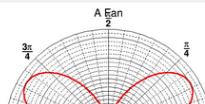
Measurement XYZ and Expectation



Open In SWAN

### Macro 3: Polar graph

```
In [2]: theta[pt] = TMath::Sin((1.pt));  
  
TGraphPolar grP1(inputs,r,theta);  
grP1.SetTitle("A. Fan");  
grP1.SetXTitle("r");  
grP1.SetYTitle("theta");  
grP1.DrawClone("l");  
c->Draw();
```



Open In SWAN

### Macro 4: Create, fit and draw a three 3D graph

```
In [4]: auto cl = new TCanvas();  
f2.SetTitle("f2");  
T3D T3D; T3D.SetColor(1);  
T3D.SetLineColor(1);  
Taxis Maxxis = T3D->GetXaxis();  
Taxis Minxis = T3D->GetXaxis();  
Maxxis->SetRange(-1.5, 1.5);  
Minxis->SetRange(-1.5, 1.5);  
Yaxis Maxyis = T3D->GetYaxis();  
Yaxis Minyis = T3D->GetYaxis();  
Maxyis->SetTitle("z");  
Minyis->SetTitle("z");  
Zaxis Maxzis = T3D->GetZaxis();  
Zaxis Minzis = T3D->GetZaxis();  
Maxzis->SetTitle("r");  
Minzis->SetTitle("r");  
dts->DrawClone("p");
```

Display the 2D graph in the notebook.

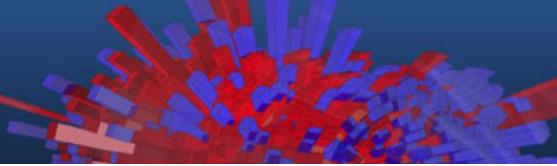
```
In [5]: cl->Draw();
```

Fitted 2D function



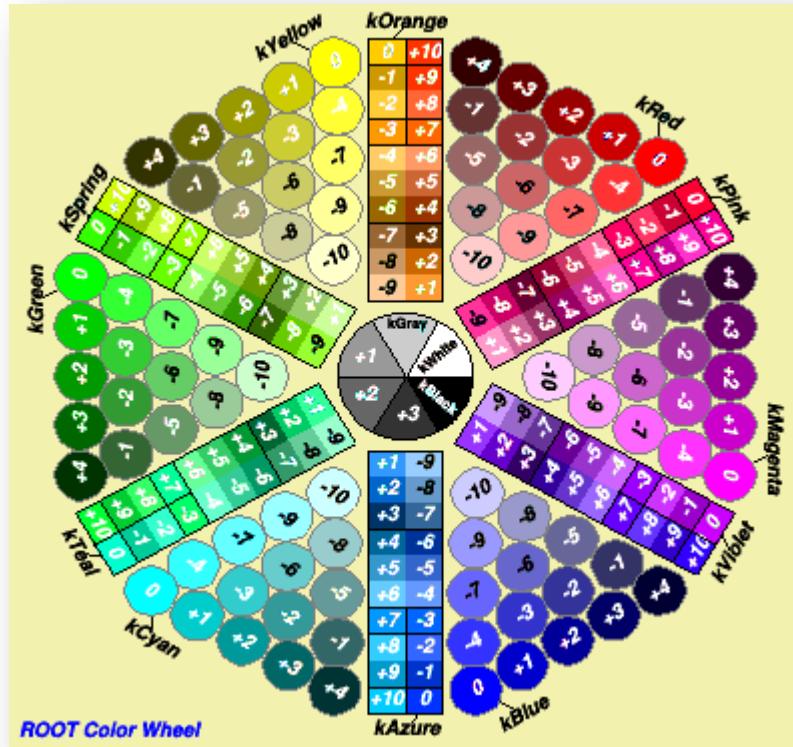
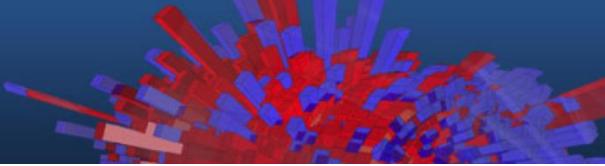
Open In SWAN

# Summary of Visual Effects

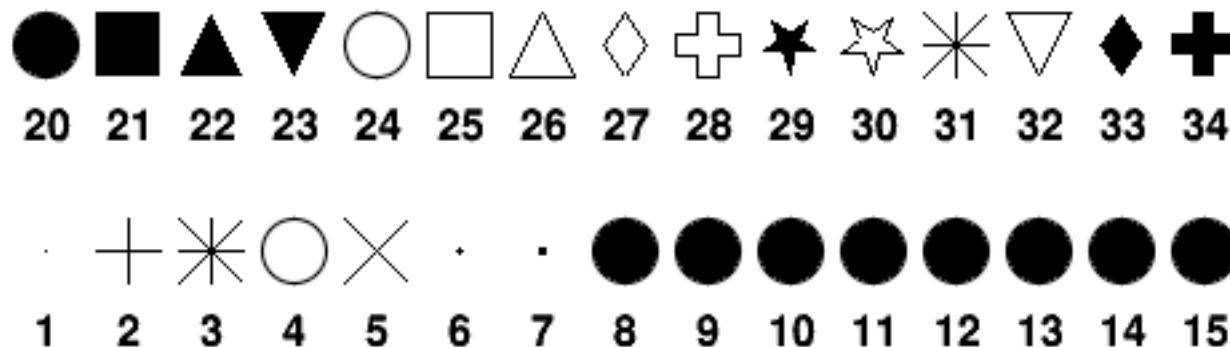
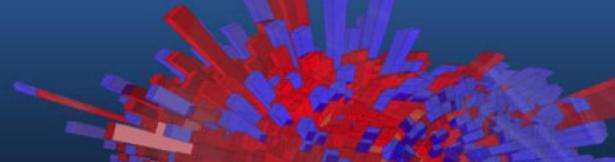


- **Colours and Graph Markers:** To specify a colour, some identifiers like kWhite, kRed or kBlue can be used for markers, lines, arrows etc. The complete summary of colours is represented by the ROOT “colour wheel”. ROOT provides several graphics markers like triangles, crosses or stars.
- **Arrows and Lines:** The class representing arrows is TArrow, which inherits from TLine. The constructors of lines and arrows always contain the coordinates of the endpoints.
- **Text:** A possibility to add text in plots is provided by the TLatex class. Latex mathematical symbols are automatically interpreted, you just need to replace the “\” by a “#”.

# TColorWheel



# The Family of Markers



kDot=1, kPlus, kStar, kCircle=4, kMultiply=5,  
kFullDotSmall=6, kFullDotMedium=7, kFullDotLarge=8,  
kFullCircle=20, kFullSquare=21, kFullTriangleUp=22,  
kFullTriangleDown=23, kOpenCircle=24, kOpenSquare=25,  
kOpenTriangleUp=26, kOpenDiamond=27, kOpenCross=28,  
kFullStar=29, kOpenStar=30, kOpenTriangleDown=32,  
kFullDiamond=33, kFullCross=34

Also available  
through more  
friendly names ☺

# Interpretation and Compilation

We have seen how ROOT interprets and “just in time compiles” code. ROOT also allows to compile code “traditionally”. At the ROOT prompt:

```
root [1] .L macro1.C+
root [2] macro1()
```

Generate shared library and execute function

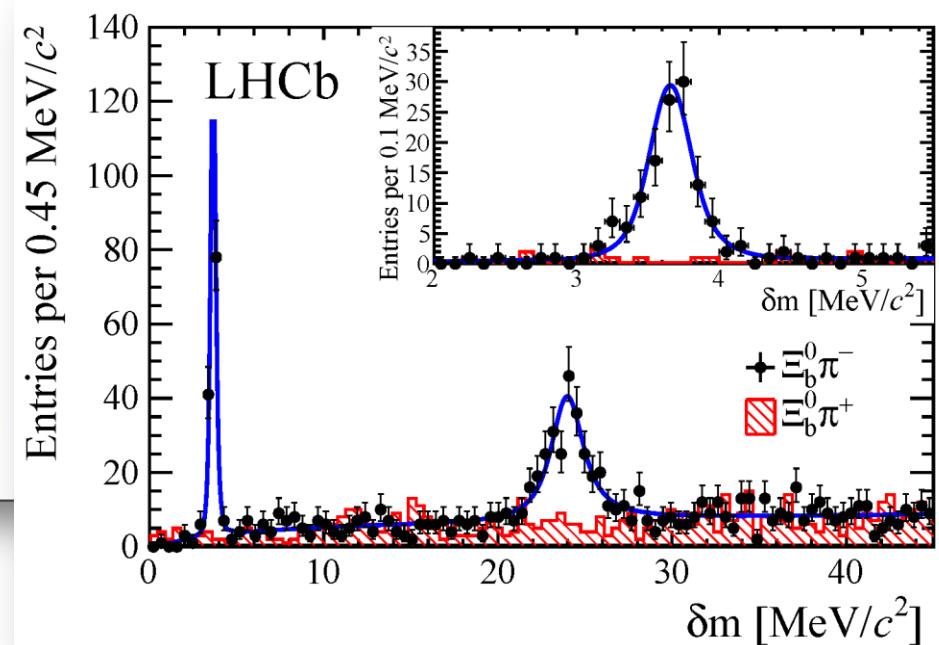
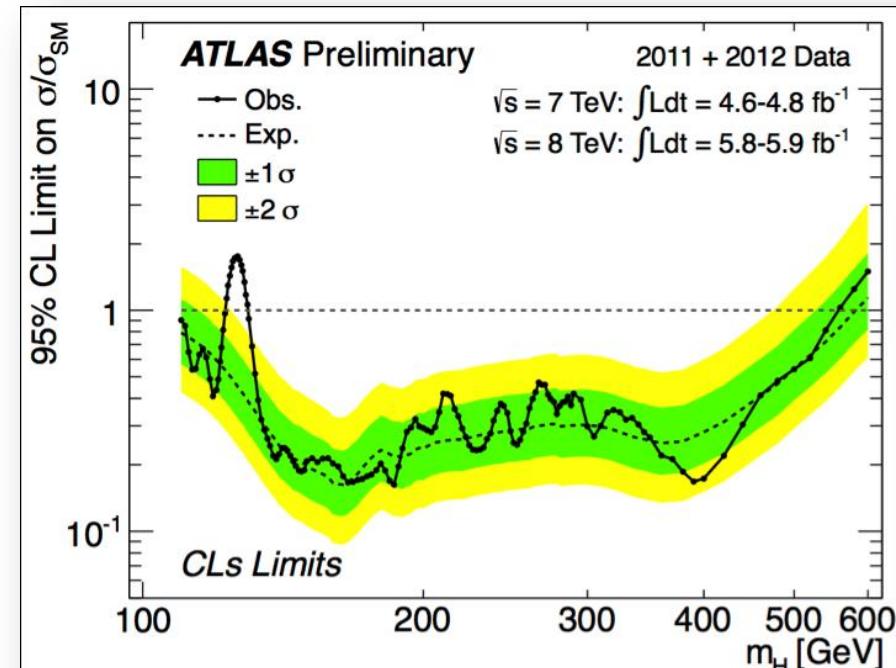
ExampleMacro.C

```
int main() {
    ExampleMacro();
    return 0;
}
```

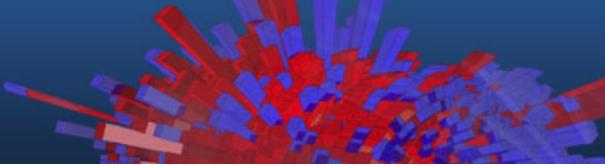
ROOT libraries can be also used to produce standalone, compiled applications:

```
> g++ -o ExampleMacro ExampleMacro.C `root-config --cflags --libs`  
> ./ExampleMacro
```

# More about Graphs and Histograms

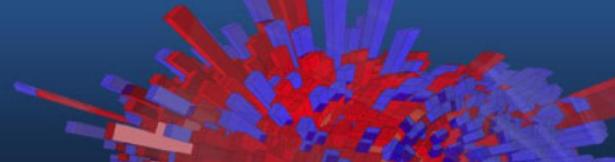


# Graphs



- Read Graph Points from File
- Polar Graphs
- 2D Graphs
- Multiple graphs

# From an ASCII File



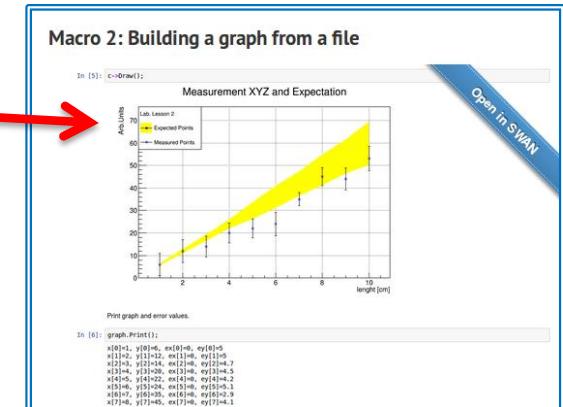
To build a graph, experimental data can be read from an ASCII file (i.e. standard text) using this constructor:

```
TGraphErrors(const char *filename,  
             const char *format="%lg %lg %lg %lg",  
             Option_t *option="");
```

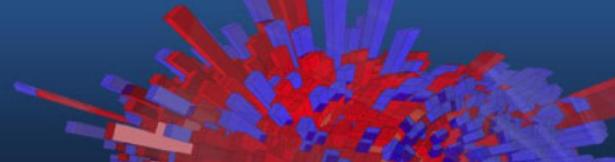
Let's have a look at macro2.C in a notebook  
(also in section 4.1 of the Primer).



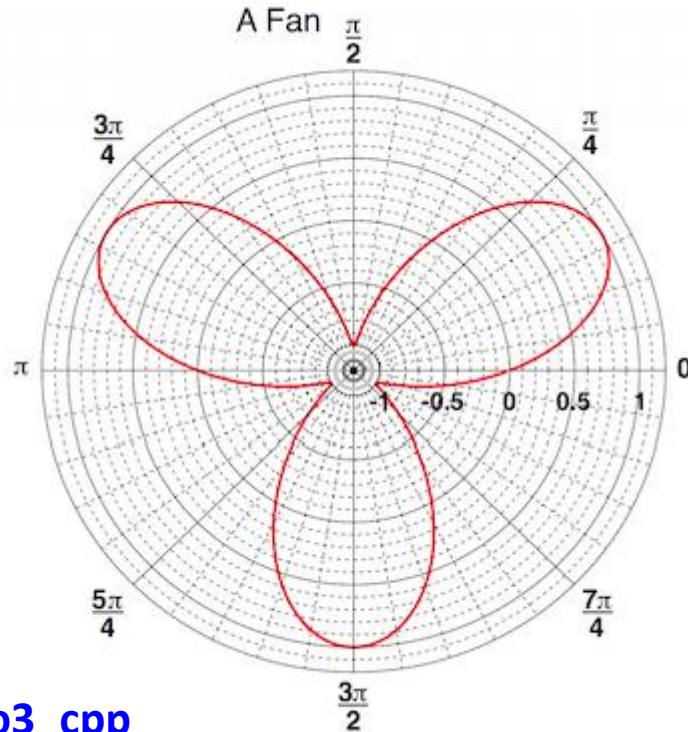
Macro2\_cpp



# Polar Graphs



Graphs can also be displayed in polar coordinate like in *macro3.C* (section 4.2 in the Primer):

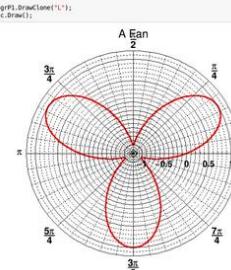


Macro3\_cpp



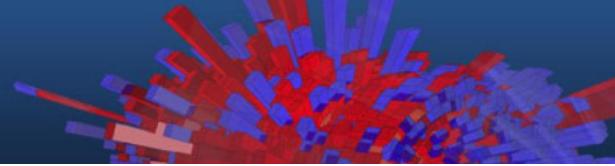
Macro 3: Polar graph

```
theta(pt) = TMath::Pi()*(r*pt);  
grP1.SetTitle("A Fan");  
grP1.SetLineWidth(2);  
grP1.SetLineColor(2);  
grP1.DrawLine("l");  
(drawn);
```

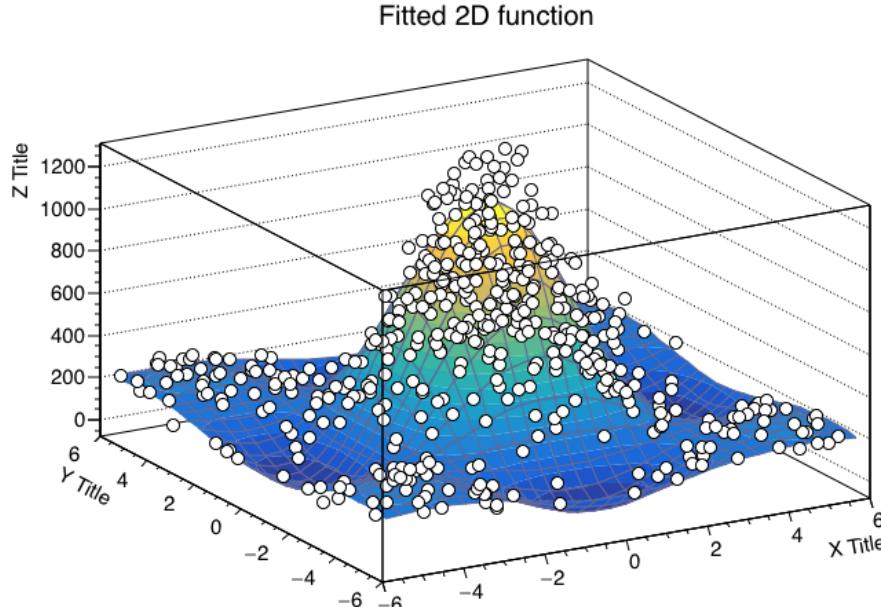


Open in SWAN

# 2D Graphs



Bi-dimensional graphs can be created in ROOT with the *TGraph2DErrors* class. *macro4.C*, described in Primer's section 4.3, gives a nice example:



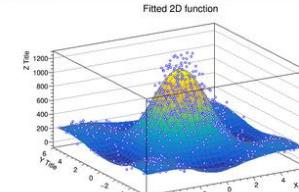
Macro4\_cpp



Macro 4: Create, fit and draw a three 3D graph

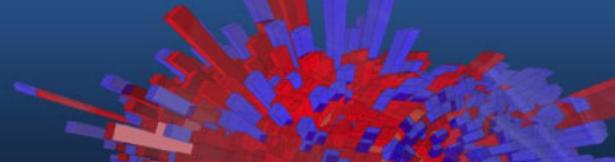
```
In [4]: note cl = new TCanvas();
T2.SetTitle("Depth1");
T2->SetZRange(0,1200);
T2->Draw();
TF2 = TF2 + (TF2*TF2).Clone("Surf1");
TAxis *xaxis = T2->GetXaxis();
TAxis *yaxis = T2->GetYaxis();
TAxis *zaxis = T2->GetZaxis();
zaxis->SetTitle("Z Title");
zaxis->SetTitleOffset(1.5);
yaxis->SetTitle("Y Title");
yaxis->SetTitleOffset(1.5);
zaxis->SetTitle("Z Title");
zaxis->SetTitleOffset(1.5);
note->Draw();
Display the 2D graph in the notebook.
```

```
In [5]: cl->draw();
```

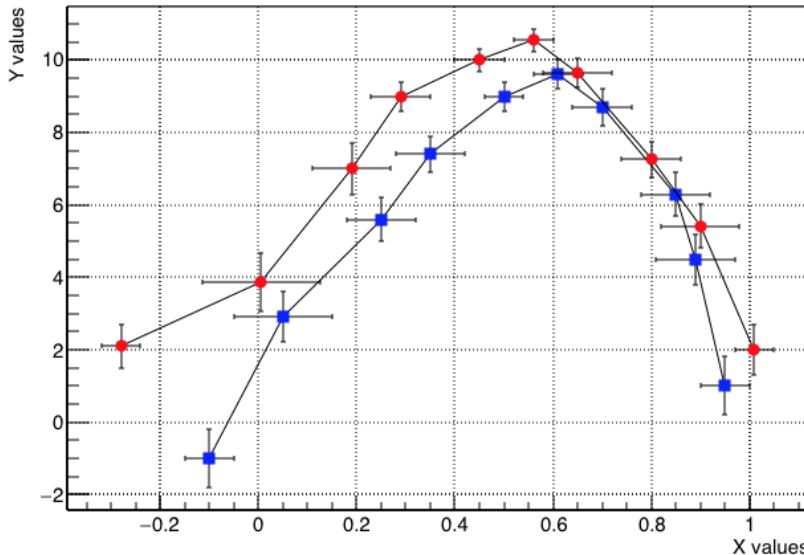


Open in SWAN

# Multiple Graphs



It is sometimes useful to group graphs in a single entity, for instance to compute a common axis system. The class *TMultiGraph* described in section 4.4 of the Primer allows that.



Multigraph\_cpp



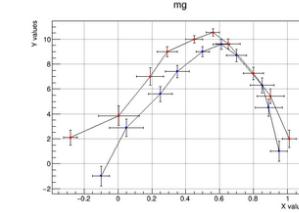
## Multigraph

**End Product**  
We add the two graphs onto our canvas, and update it.

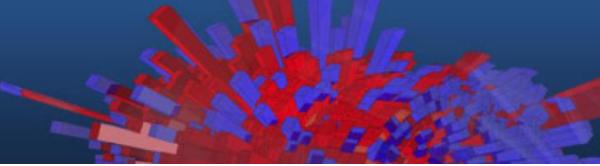
```
In [4]: mg.Draw("APL");
mg.GetXaxis()->SetTitle("x values");
mg.GetYaxis()->SetTitle("y values");

We finally display the canvas.
```

In [5]: c1.Draw()

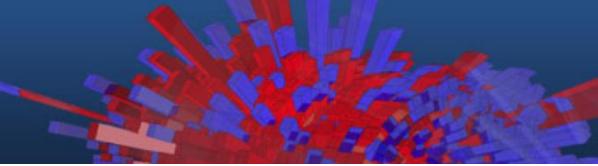


# Histograms



- Your First (in fact second) Histogram
- Add and Divide Histograms
- Two-dimensional Histograms
- Multiple Histograms

# Exercise

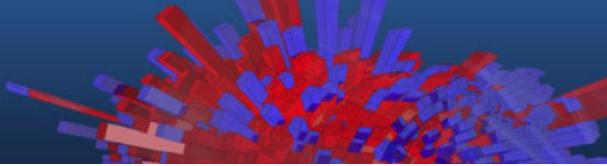


Write a macro to visualise a Poisson distribution in a histogram

- Create a 1D histogram the bins of which are double precision numbers
- The max number of counts collected is 15 (max value on the x axis)
- Use a random generator to generate 1000 Poissonian counts,  $\mu=4$
- Properly set the title and axes names, fill the histogram in blue
- Fit it, programmatically or with the fit panel (right click on the histogram)

The solution of this exercise is macro5.C shown in section 5.1 in the Primer

# Exercise - Optional

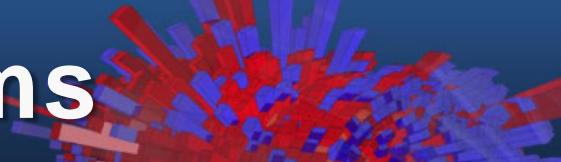


Create a macro that draws the sum, difference and ratio of two histograms

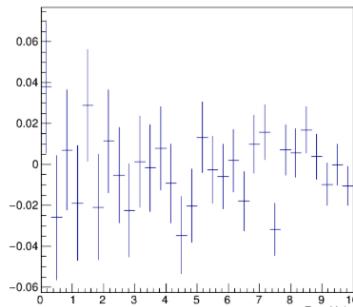
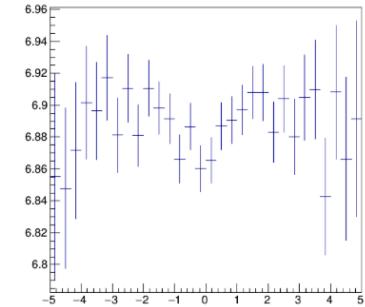
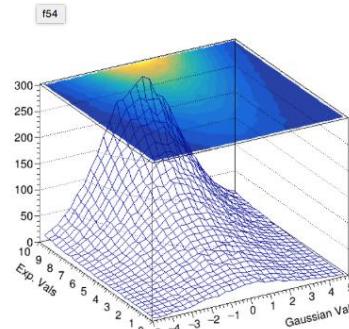
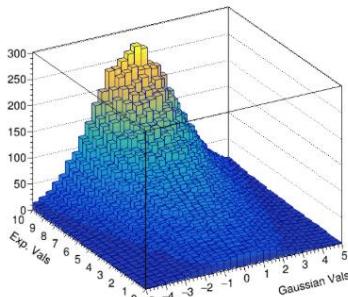
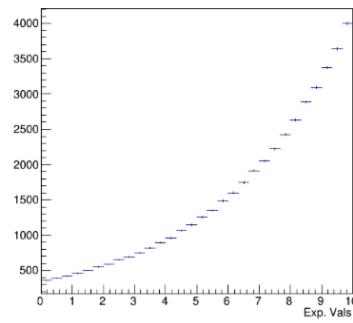
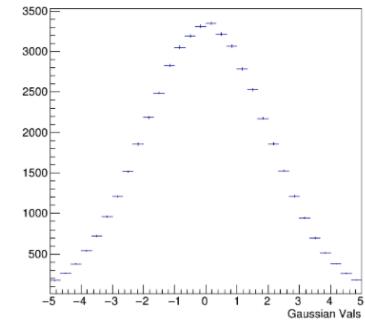
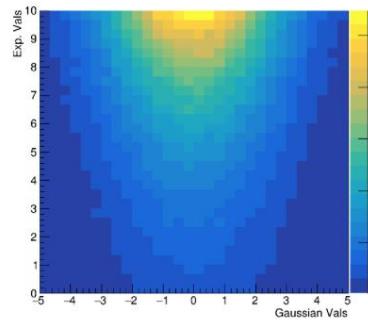
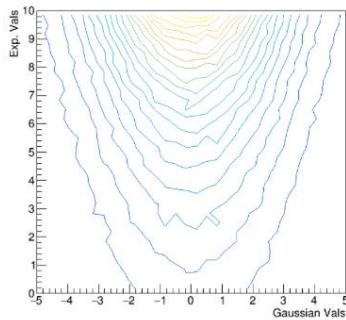
- Create three pairs of histograms, fill them randomly with normally distributed numbers (`TH1::FillRandom("gaus")`)
- Divide, sum and subtract them
  - Useful methods:  
`TH1::Divide(const TH1*),`  
`TH1::Add(const TH1*, Double_t)` the second parameter is a weight
- Note: for every plot a different canvas has to be created and before drawing, one has to "cd" into it
  - `TCanvas c; c.cd();`

The solution of this exercise is `macro6.C` shown in section 5.2 in the Primer

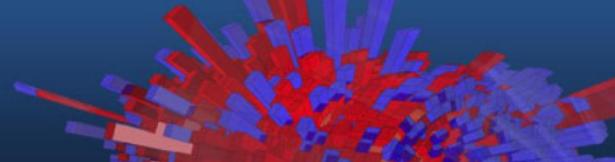
# Two Dimensional Histograms



Two-dimensional histograms are a very useful tool, for example to inspect correlations between variables, as in the example in section 5.3 of the Primer (macro7.C):

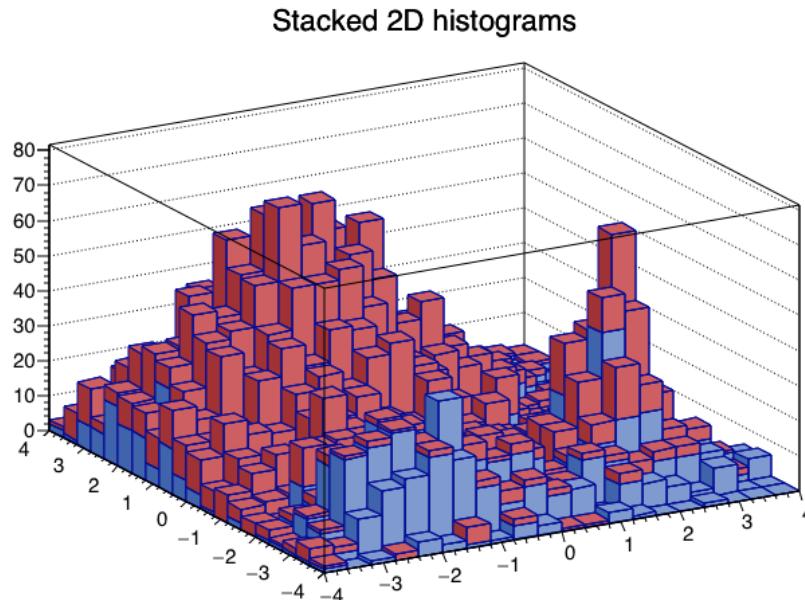


# Multiple Histograms



The example in section 5.4 (hstack.C) shows how to group histograms in a single entity call a “stack”.

**Class  
THStack**



**Hstack\_cpp**



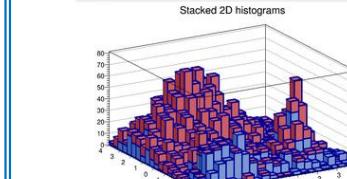
**THStack: Draw stacks of several histograms**

```
h2sta.F1FillRandom("t1", 4000);  
TF2 f2f("f2f", "x,y=gaus(51,-0.4,-0.4);  
double *params2[1] = {100,-1,-4,1.9,1.1,1.2,10,2,0.7,-2,0.5};  
f2f.SetParameters(params2);  
TH3 h3bst(h3bst15,"h3bst15",-0.4,4,20,-4,4,20,-4,4);  
h2sta.F1FillRandom("t2", 4000);  
h2stb.F1FillRandom("t3", 4000);
```

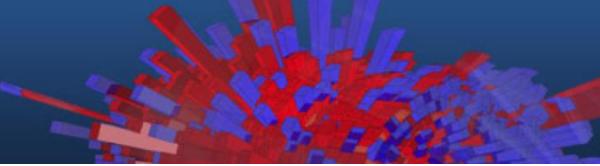
We then add them onto our stacked histogram graph, and draw it.

```
In [3]: theStack.Add(h2sta1);  
theStack.Add(h2stb1);  
TCanvas c1;  
theStack.Draw();  
c1.Draw();
```

[Open In SWAN](#)

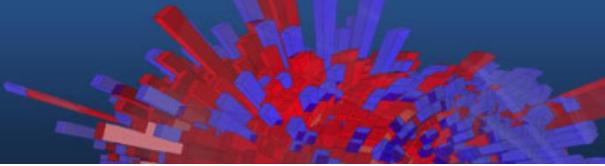


# Input and Output



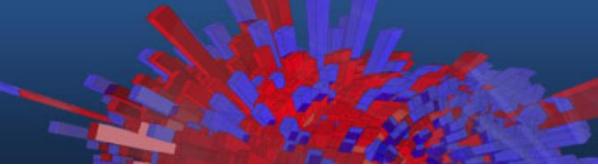
- Storing Objects
- N-tuples

# Storing Objects in a File



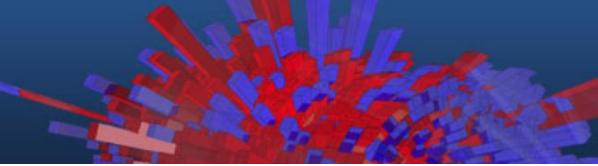
- ROOT allows to store C++ objects on disk (natively the language cannot)
- All ROOT objects (inheriting from TObject) can be written on disk via the Write method.
- Two ways of storing: row wise (single object dump ) and column wise (N-tuple like storage ).
- Feature widely used, e.g. by all LHC experiments

# An Example



```
TFile out_file("my_rootfile.root","RECREATE"); // Open a Tfile
TH1F h("my_histogram","My Title;X;# of entries",100,-5,5);
h.FillRandom("gaus");
h.Write(); // Write the histogram in the file
out_file.Close(); // Close the file
```

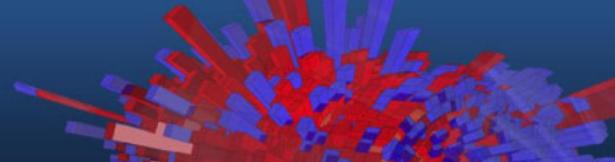
# Exercise



Inspect the content of a file with the TBrowser

- Create a file copying the lines of the previous slide at the prompt
- Quit the command line interpreter
- Boot ROOT opening the file: root my\_rootfile.root
- Type: TBrowser myBrowser
- Inspect the content of the file

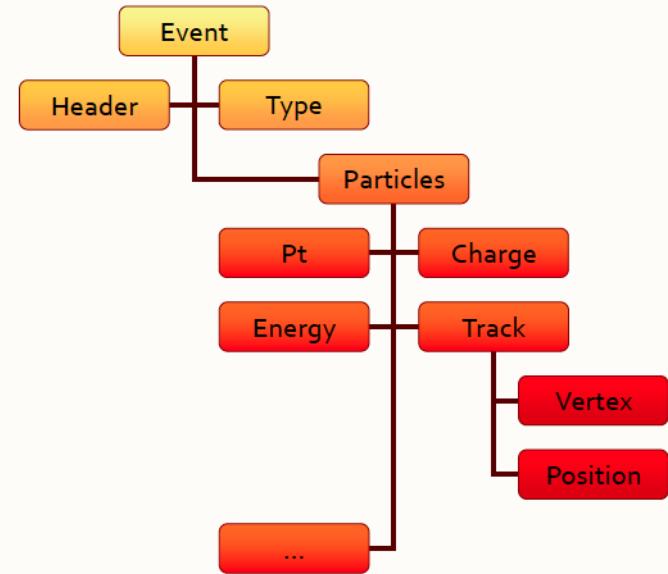
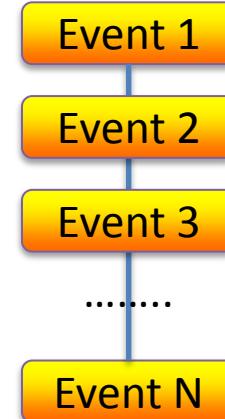
# Trees



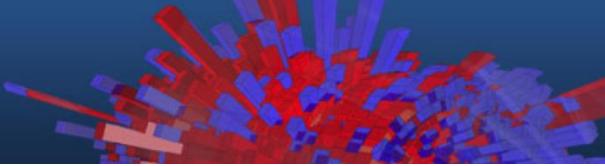
- The TTree is the data structure ROOT provides to store large quantities of same types objects
- Organised in branches, each one holding objects
- Organised in independent events, e.g. collision events
- Efficient disk space usage, optimised I/O runtime

x	y	z
-1.10228	-1.79939	4.452822
1.867178	-0.59662	3.842313
-0.52418	1.868521	3.766139
-0.38061	0.969128	1.084074
0.552454	-0.21231	0.360281
-0.18495	1.187305	1.443902
0.205643	-0.77015	0.635417
1.079222	-0.32739	1.271904
-0.27492	-1.72143	3.038899
2.047779	-0.06268	4.197329
-0.45868	-1.44322	2.293266
0.304731	-0.88464	0.875442
-0.71234	-0.22239	0.556881
-0.27187	1.181767	1.470484
0.866202	-0.65411	1.213209
-2.03555	0.527648	4.421883
-1.45905	-0.464	2.344113
1.230661	-0.00565	1.514559
		3.562347

LEP style flat n-tuples  
evolved in more efficient  
trees (fast access, read  
ahead)

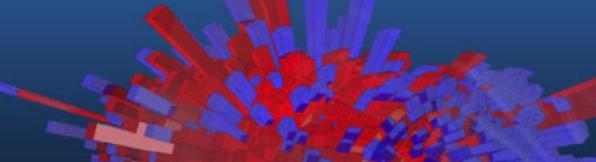


# Ntuples



- The TNtuple is a simplified version of the TTree: store floating point numbers
- As powerful for analysis

# Example



```
TFile ofile("conductivity_experiment.root","RECREATE");
TNtuple cond_data("cond_data",
                  "Example N-Tuple",
                  "Potential:Current:Temperature:Pressure");
TRandom3 rndm; // We'll fill random values
float pot,cur,temp,pres;
for (int i=0;i<10000;++i) {
    pot = rndm.Uniform(0.,10.);      // get voltage
    temp = rndm.Uniform(250.,350.);  // get temperature
    pres = rndm.Uniform(0.5,1.5);    // get pressure
    cur = pot/(10.+0.05*(temp-300.)-0.2*(pres-1.));// current
    // add some random smearing (measurement errors)
    pot* = rndm.Gaus(1.,0.01); temp+=rndm.Gaus(0.,0.3);
    pres*= rndm.Gaus(1.,0.02); cur*=rndm.Gaus(1.,0.01);
    // write to ntuple
    cond_data.Fill(pot,cur,temp,pres);
}
// Save the ntuple and close the file
cond_data.Write(); ofile.Close();
```

**Primer macro (section 7.2.1)**  
***write\_ntuple\_to\_file.C***

# Exercise: Potential of the Tree

- Run the `write_ntuple_to_file.C` macro
- Open the file in the TBrowser
- Create plots clicking on the leaves
- Run the code in SWAN
  - Check your CERNBox for the result file!



## Writing an N-tuple to a File

Filling an n-tuple (simulating the conductivity of a material at different conditions of pressure and temperature) and writing it to a file.

Posted in Notebook by Théo Hansen

To use the ROOT notebook, we need to import ROOT onto our Notebook, which we also set to C++

[Open in SWAN](#)

### The Tuple

We create a file which will contain our ntuple and the tuple itself.

```
In [1]: Title::Title("conductivity_experiments.root","RECREATE");
TFile cond_data("cond_data", "Example N-Tuple", "Potential:Current:Temperature:Pressure");

Then we fill it randomly to simulate acquired data using the TRandom3 random generator. We are also applying some "smearing" (measurement errors): 1% error on voltage [pot], pressure and current and 1.0 absolute error on temperature. At the end of the loop body we fill the ntuple.

In [2]: TH1D* pot;
float pot,cur,temp,press;
for (int i=0; i<100000; i++) {
    pot=rnd3.Uniform(0.,10.);
    temp=rnd3.Uniform(-10.,10.);
    press=rnd3.Uniform(0.5,1.5);
    cur= pot*(10.-40.*temp*300.)-0.2*(pres-1);
    pot+=rnd3.Gaus(0.,0.01);
    temp+=rnd3.Gaus(0.,0.3);
    pres+=rnd3.Gaus(1.,0.01);
    cur+=rnd3.Gaus(0.,0.1);
    cond_data.Fill(pot,cur,temp,press);
}
```

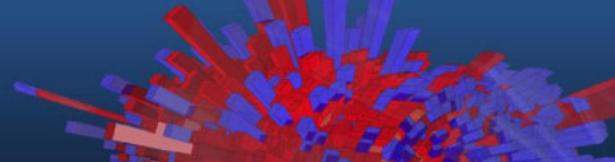
Save the TH1D and close the file.

```
In [3]: cond_data.Close();
ofile.Close();
```



## [Write\\_ntuple\\_to\\_file\\_cpp](#)

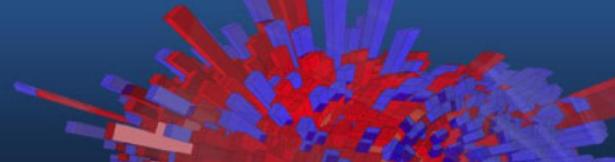
# Accessing Complex Trees



- TTreeReader class: tool to access complex trees in a type-safe manner
  - Not only floating point numbers as in TNtuple, but all objects!

```
// Access a TTree called "MyTree" in the file:  
TTreeReader reader("MyTree", file);  
// Establish links with two of the branches  
TTreeReaderValue<float> rvMissingET(reader, "missingET");  
TTreeReaderValue<std::vector<Muon>> rvMuons(reader, "muons");
```

# Accessing the Data



```
// Loop through all the TTree's entries
// It behaves like an iterator...
while (reader.Next()) {
    float missingET = *rvMissingET;
    ...
    for (auto&& mu: rvMuons) {
        hist->Fill(pT);
    }
}
```



**TTree Access: C++ with TTreeReader**

Access a TTree with the TTreeReader

Open a file which is located on the web. Build a TTreeReader and two TTreeReader values, one for the tracks and one for the events.

```
In [1]: auto f = TFile::Open("http://indico.cern.ch/event/395980/material/8/0.root");
TTreeReader myReader("events", f);
TTreeReaderValue<vector<ROOT::Math::PolyVector3>> trackInv(myReader, "tracks");
TTreeReaderValue<int> eventNum(myReader, "eventnum");

Loop over the events stored in the tree. Analyse the transverse momentum of tracks and identify the maximum one. Print the result every one hundred events.
```

```
In [2]: do {
    double eventPt;
    while (myReader.Next()) {
        auto eventNum = (*eventInv);
        auto tracks = *trackInv;
        auto pTmax = 0;
        for (auto& track : tracks) {
            auto pt = track.Pt();
            if (pt > pTmax) pTmax = pt;
        }
        if (eventNum >= 0) {
            std::cout << "Processing event number " << eventNum << std::endl;
            std::cout << "Max pt is " << pTmax << std::endl;
        }
    }
} while (true);

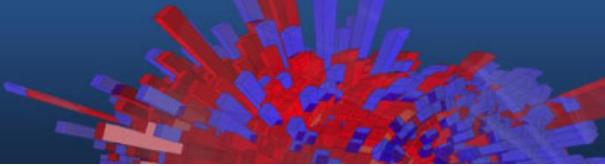
Processing event number 0
Max pt is 37.3882
Processing event number 100
Max pt is 37.3882
Processing event number 200
Max pt is 37.3882
Processing event number 300
Max pt is 37.3882
Processing event number 400
Max pt is 37.3882
Processing event number 500
```

*Open in SWAN*



[TTreeReader\\_Example\\_cpp](#)

# PyROOT



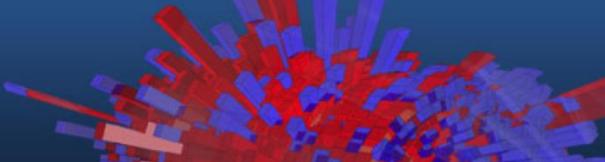
- ROOT offers the possibility to interface to Python via a set of bindings called PyROOT
- Mix the power of C++ (compiled libraries) and flexibility of Python
- Killer application: JIT of C++ code from within Python
  - Real mix of the two languages
- See Primer's section 8 for more details

Entry point to use ROOT from within Python:

```
import ROOT
```

All classes you now know can be accessed like `ROOT.TH1F`, `ROOT.TGraph`, ...

# Exercise



- Open the Python interpreter (type `python`)
- Import the ROOT module
- Create a histogram with 64 bins and an x axis ranging from 0 to 16
- Fill it with random numbers distributed according to a linear function (“`pol0`”)
- Change its line width with a thicker one
- Draw it!

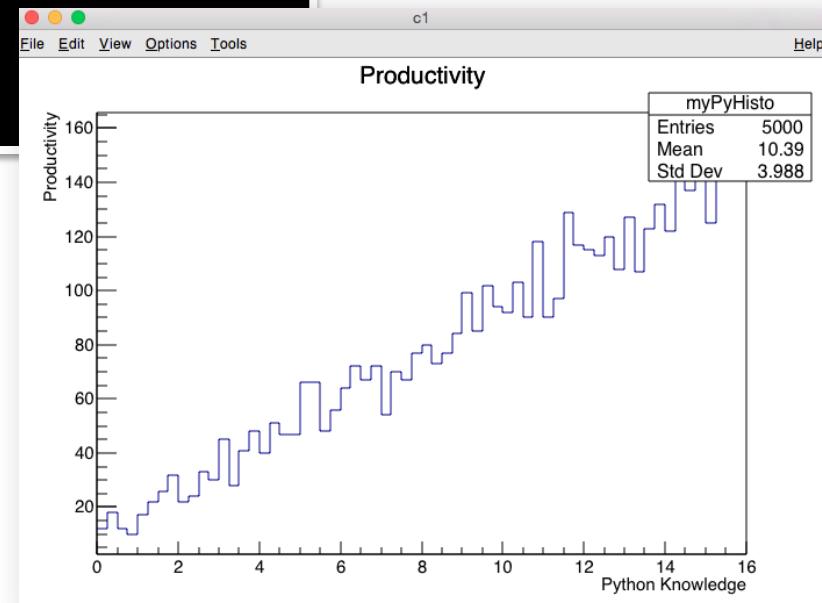
...or try it in **SWAN!**



<https://swan.cern.ch>

# Exercise

```
~> python  
>>> import ROOT  
>>> h = ROOT.TH1F("myPyHisto","Productivity;Python  
Knowledge;Productivity",64,0,16)  
>>> h.FillRandom("pol1")  
>>> h.Draw()
```

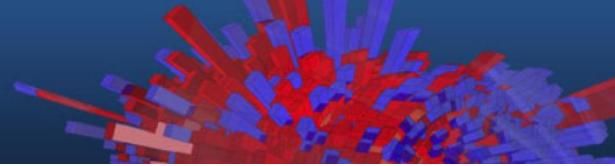


# Exercise

```
~> python  
>>> import ROOT  
>>> h = ROOT.TH1F("myPyHisto","Productivity;Python  
Knowledge;Productivity",64,0,16)  
>>> h.FillRandom("pol1")  
>>> h.Draw()
```



[FillHistogram\\_Example\\_py](#)



**TTree Access: PyROOT**

Access TTree in Python using PyROOT

**Histogram Filling (PyROOT)**

Open a file which is located on the web. No type is to be specified for 'T'.

```
In [13]: f = ROOT.TFile.open("http://root.cern.ch/event/395190/material/0/0.root");  
  
In [14]: h = ROOT.TH1F("TracksPt","Tracks;Pt [GeV/c]",128,0,64)  
for event in f.events:  
    for track in event.tracks:  
        h.Fill(track.Pt)  
c = ROOT.TCanvas()  
h.Draw()  
c.Draw()
```

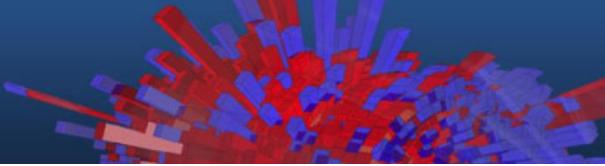
**Tracks**

A histogram titled "Tracks" showing the distribution of particle tracks based on their transverse momentum (Pt) in GeV/c. The x-axis ranges from 0 to 60 GeV/c, and the y-axis ranges from 0 to 15. The distribution is roughly Gaussian, peaking around 15-20 GeV/c. A legend box in the upper right corner provides summary statistics: Entries 499307, Mean 17.54, and Std Dev 6.554.

Open In SWAN

Open In SWAN

# Review of the objectives



## Objectives:

- Become familiar with the ROOT toolkit
- Be able to use the C++ prompt
- Plot data
- Fit data
- Perform basic I/O operations
- Learn about SWAN and Jupyter notebooks