



ROOT

Data Analysis Framework

Summer Students Course

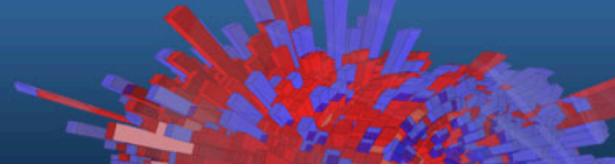
2017

Danilo Piparo, Enric Tejedor, Olivier Couet

CERN EP-SFT



This Course



This is an introductory ROOT Workshop.

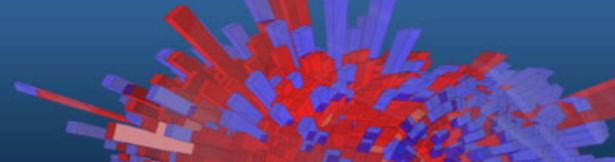
Objectives:

- Become familiar with the ROOT toolkit
- Be able to use the C++ prompt and run ROOT macros
- Plot and fit data
- Perform basic I/O operations
- Go parallel with ROOT

Format:

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

This Tutorial



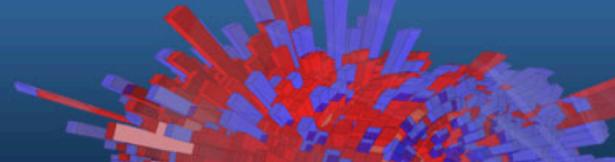
These slides are supported by the “**ROOT Primer**”

- 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:



Two release series of ROOT are available: ROOT5 and ROOT6
This lecture refers to ROOT6, version 6.10

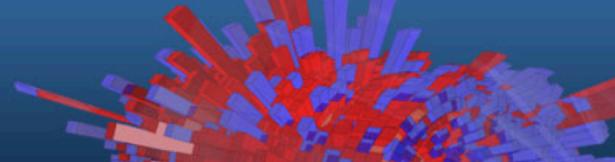
How Can I Run ROOT?



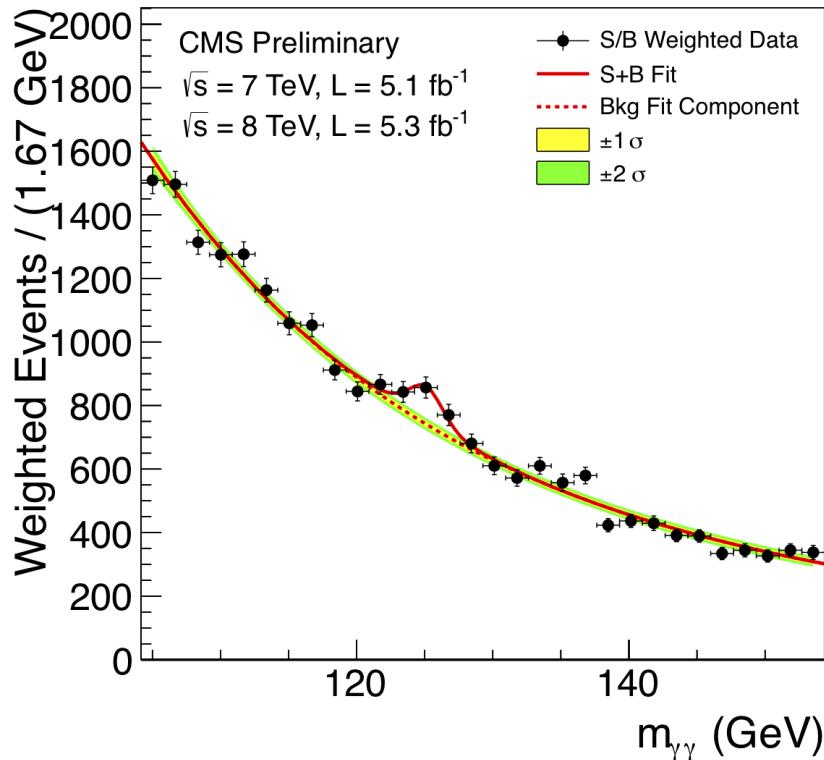
You can choose between three options:

1. Local ROOT installation
 - <https://root.cern.ch/downloading-root>
2. Lxplus node
 - <https://github.com/root-project/training/blob/master/2017/lxplus.md>
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>

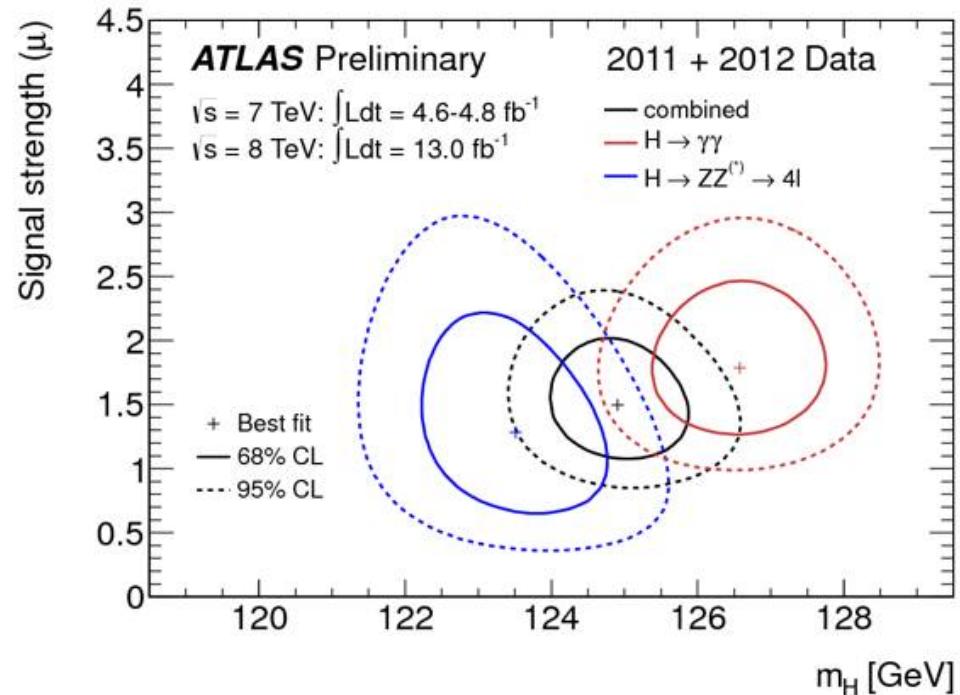
A “Quick Tour” Of ROOT



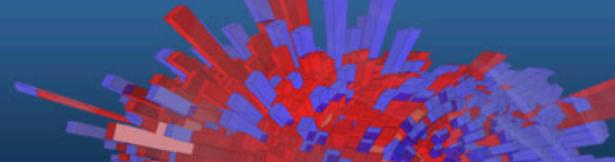
What can you do with ROOT?



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

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

- Bindings for Python is provided.



An Open Source Project

We are on github

github.com/root-project

All contributions are warmly welcome!

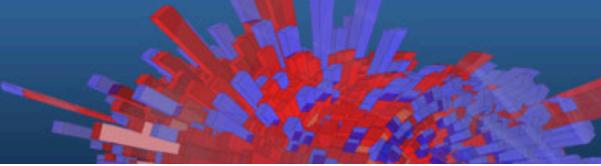


Adopted in High Energy Physics and other sciences (but also industry)

- ~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



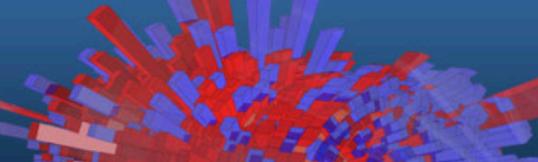
ROOT in a Nutshell



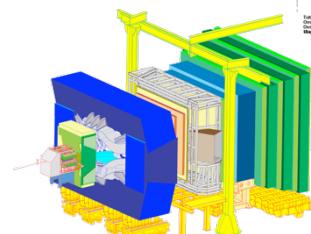
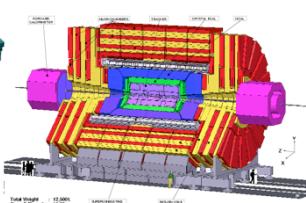
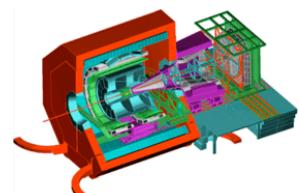
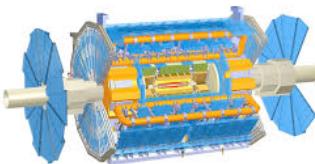
ROOT can be seen as a collection of building blocks for various activities, like:

- 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](#).

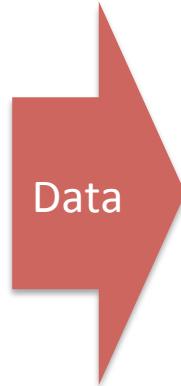
ROOT Application Domains



A selection of the experiments adopting ROOT



Event Filtering



Data

Offline Processing

Reconstruction

Further processing,
skimming

Analysis

Event Selection,
statistical treatment ...

Raw

Reco

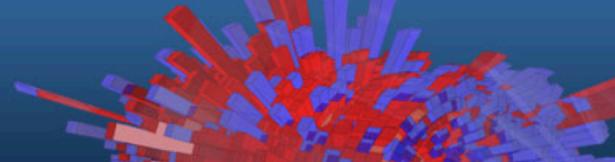
Analysis
Formats

Images

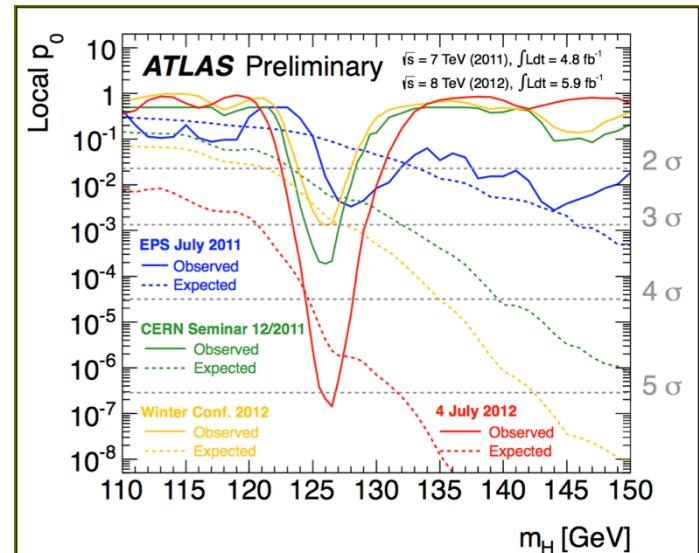
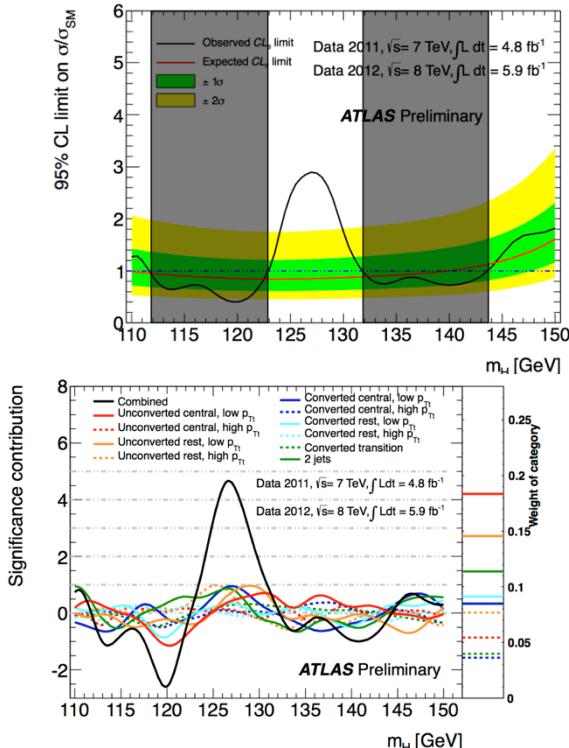
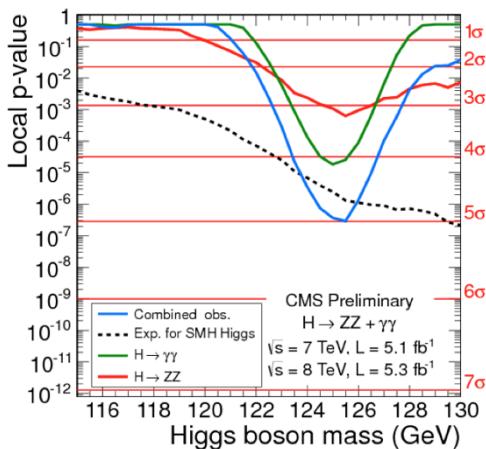
...

Data Storage: Local, Network

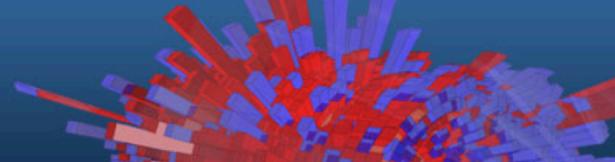
Higgs boson discovery



ATLAS and CMS collaborations were 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)

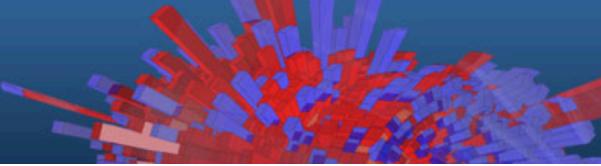
- Rapid prototyping possible

```
$ 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.

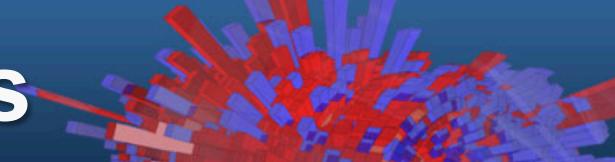
Achieved with serialization of the objects using the reflection capabilities, ultimately provided by the interpreter

- Raw and column-wise streaming

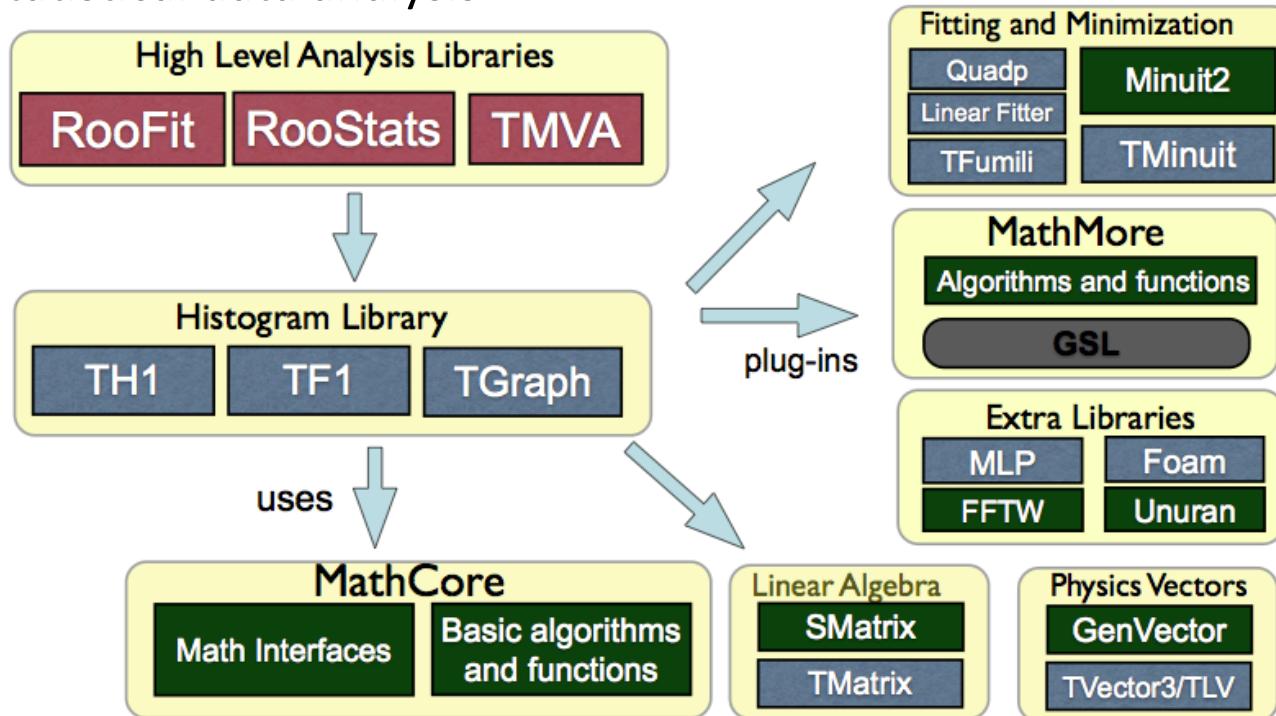
As simple as this for ROOT objects: one method - *TObject::Write*

Cornerstone for storage
of experimental data

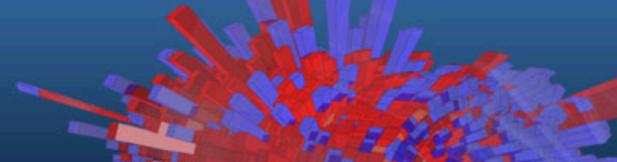
ROOT Math/Stats Libraries



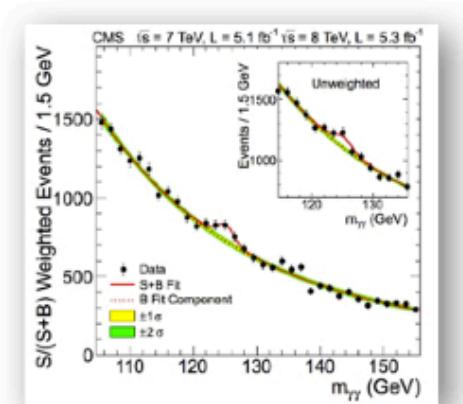
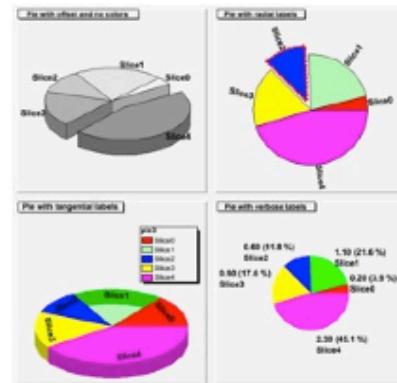
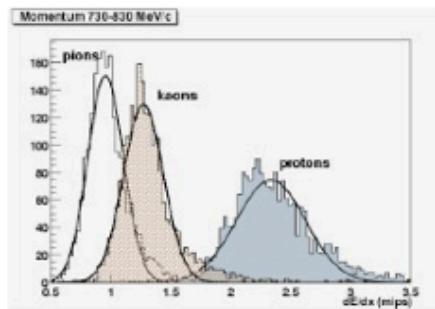
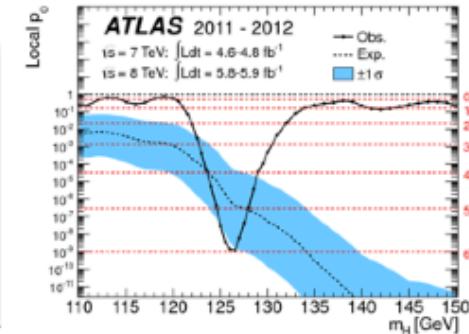
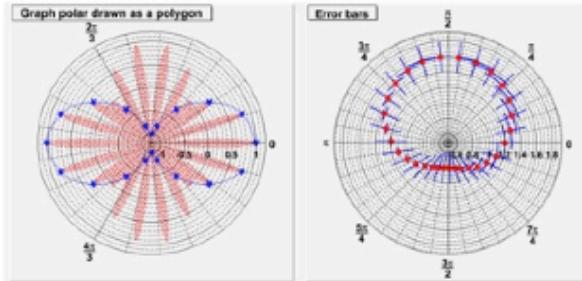
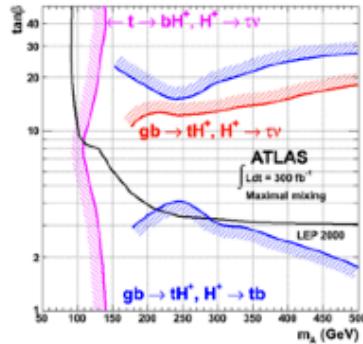
ROOT provides a reach set of mathematical libraries and tools needed for sophisticated statistical data analysis



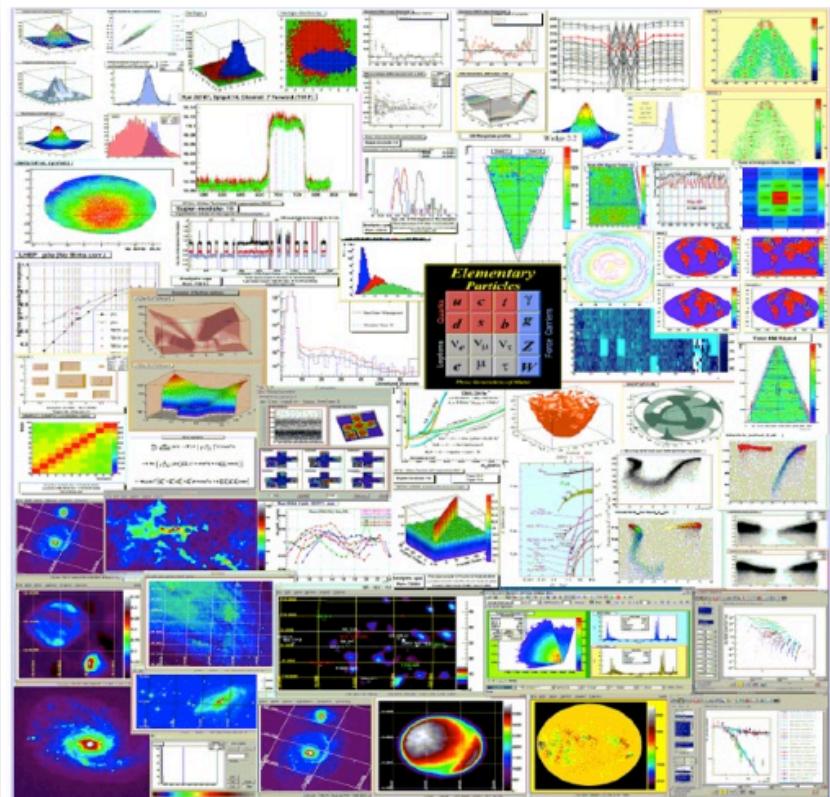
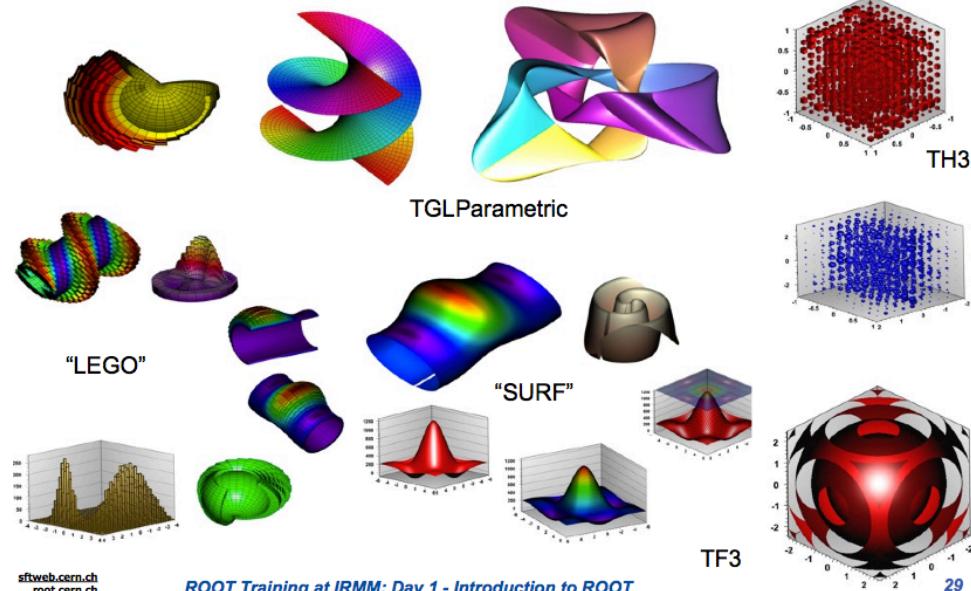
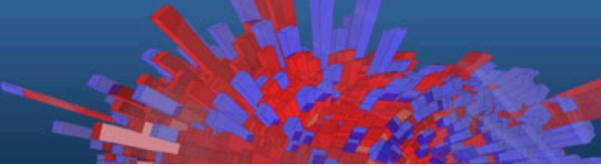
Graphics In ROOT



Many formats for data analysis, and not only, plots

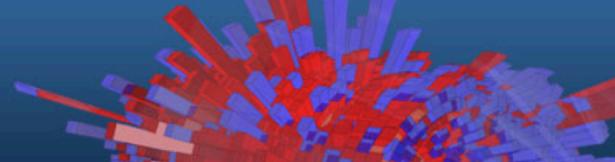


2D and 3D Graphics



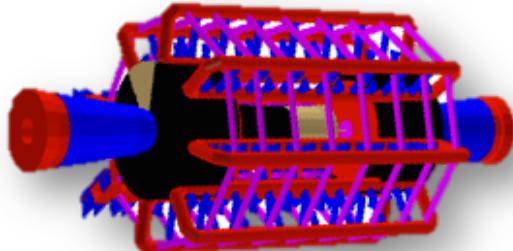
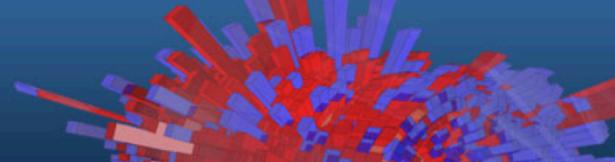
Can save graphics in many formats: *ps*, *pdf*,
svg, *jpeg*, *LaTex*, *png*, *c*, *root* ...

Parallelism



- Hot topic: many ongoing efforts to provide means for parallelisation in ROOT
- Explicit parallelism
 - TThreadExecutor and TProcessExecutor
 - Protection of resources
- Implicit parallelism
 - **TDataFrame**: functional chains
 - TTreeProcessor: process tree events in parallel
 - TTree::GetEntry: process of tree branches in parallel

Other ROOT Features

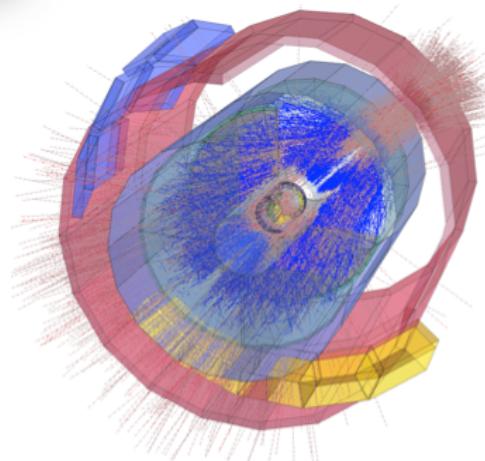


Geometry Toolkit

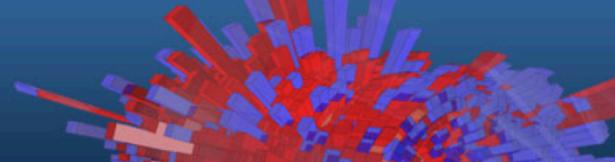
- Represent geometries as complex as LHC detectors

Event Display (EVE)

- Visualise particles collisions within detectors



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



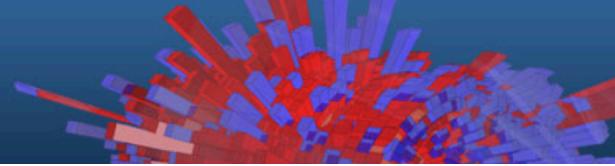
<http://swan.web.cern.ch>

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

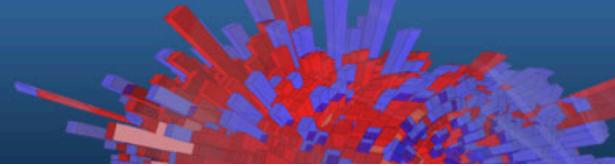
- For beginners and experts
- Downloads, installation instructions
- Documentation of all ROOT classes
- Manuals, tutorials, presentations and more
- Forum
- ...

The screenshot shows the official ROOT website at www.root.cern. The page features a dark blue header with the "root" logo and the text "Data Analysis Framework". Below the header is a navigation bar with links: Download, Documentation, News, Support, About, Development, and Contribute. To the right of the navigation bar is a "Google Custom Search" bar. The main content area includes four icons: Getting Started, Reference Guide, Forum, and Gallery. A large, complex plot titled "CMS and LHCb (LHC run I)" is displayed, showing particle tracks and background distributions. Below the plot are links for "Previous", "Pause", and "Next". A red circle highlights the "Download" button, which is located under the "Under the Spotlight" section. This section also includes news items such as "Try the new ROOTbooks on Binder (beta)", "ROOT has its Jupyter Kernel!", and "ROOT Users' Workshop 2015". The bottom of the page contains a "SITEMAP" with links to various sections like Documentation, News, Support, and Development, along with a "Contributors" link.

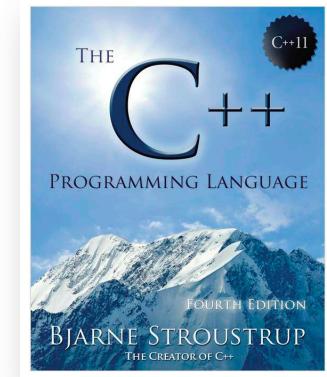
C++ From 10.000 Km



C++ From 10.000 Km



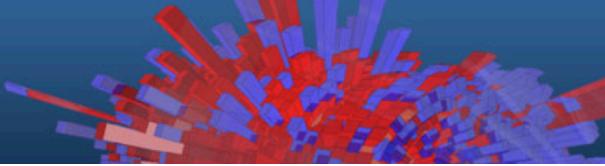
- Compiled, strongly typed language, allows to get the best performances from the hardware
- Allows object orientation
- Templates
- Explicit memory management (pointers)



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

ROOT Basics

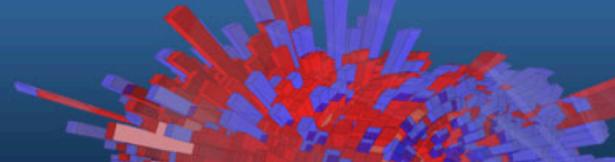


- ROOT as a Calculator
- ROOT as Function Plotter
- Plotting Measurements
- Histograms
- Interactive ROOT Section

Let's Fire Up ROOT



The ROOT Prompt



C++ is a compiled language

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

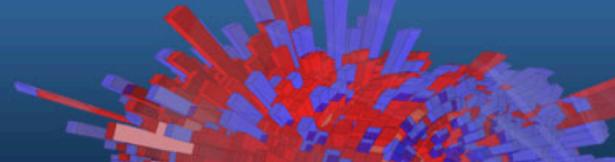
ROOT provides a C++ **interpreter**

- 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 !

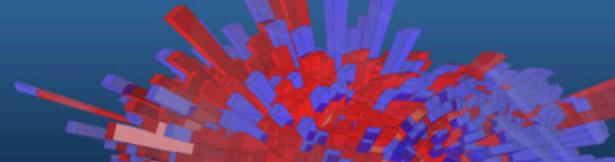
```
root [0] 1+1  
(int)2  
root [1] 2*(4+2)/12.  
(double) 1.00000  
root [2] sqrt(3.)  
(double) 1.73205  
root [3] 1 > 2  
(bool) false
```

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.

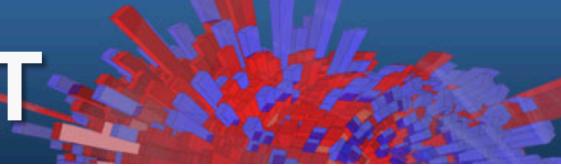
We Declare **variables** and used a *for* control structure.

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



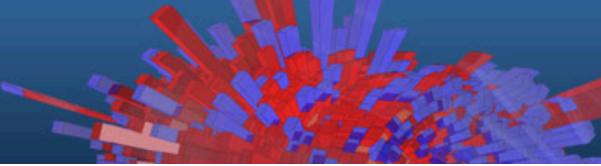
Special commands which are not C++ can be typed at the prompt, they start with a “.”

```
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

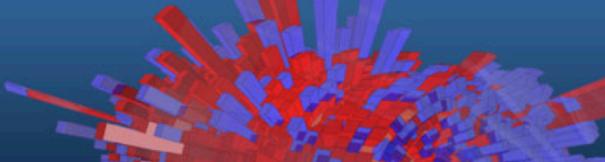


For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine.

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

For one number

Exercise Solution



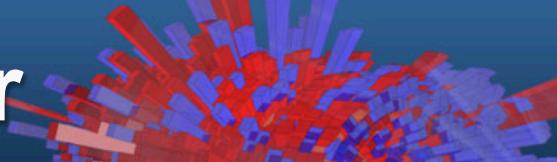
For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine.

```
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();
```

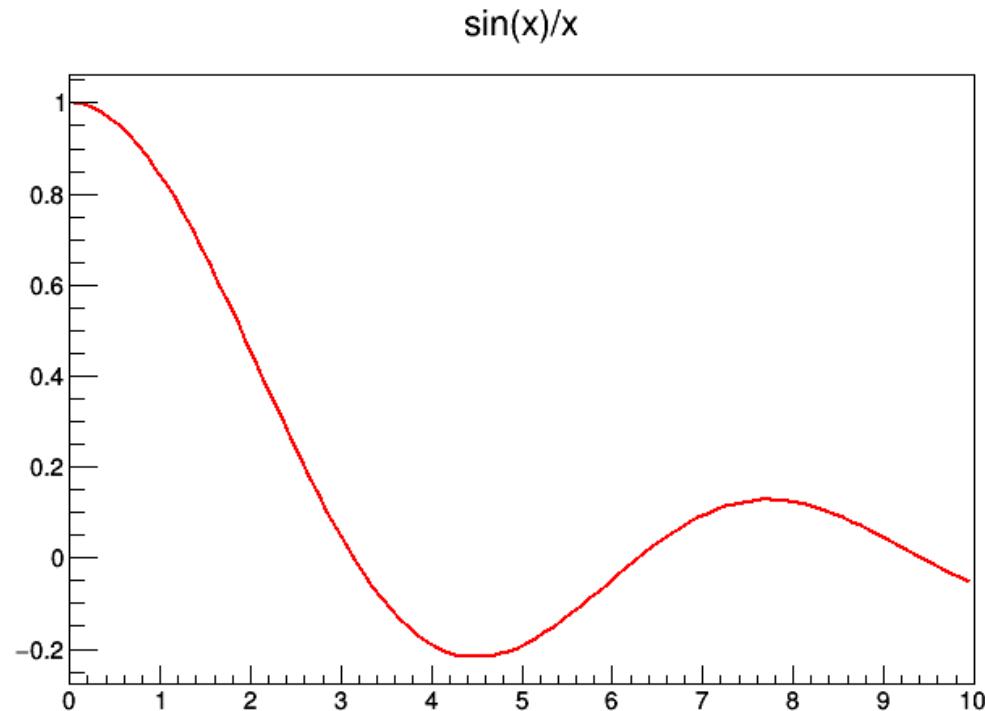
ROOT As a Function Plotter

The class TF1 re

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

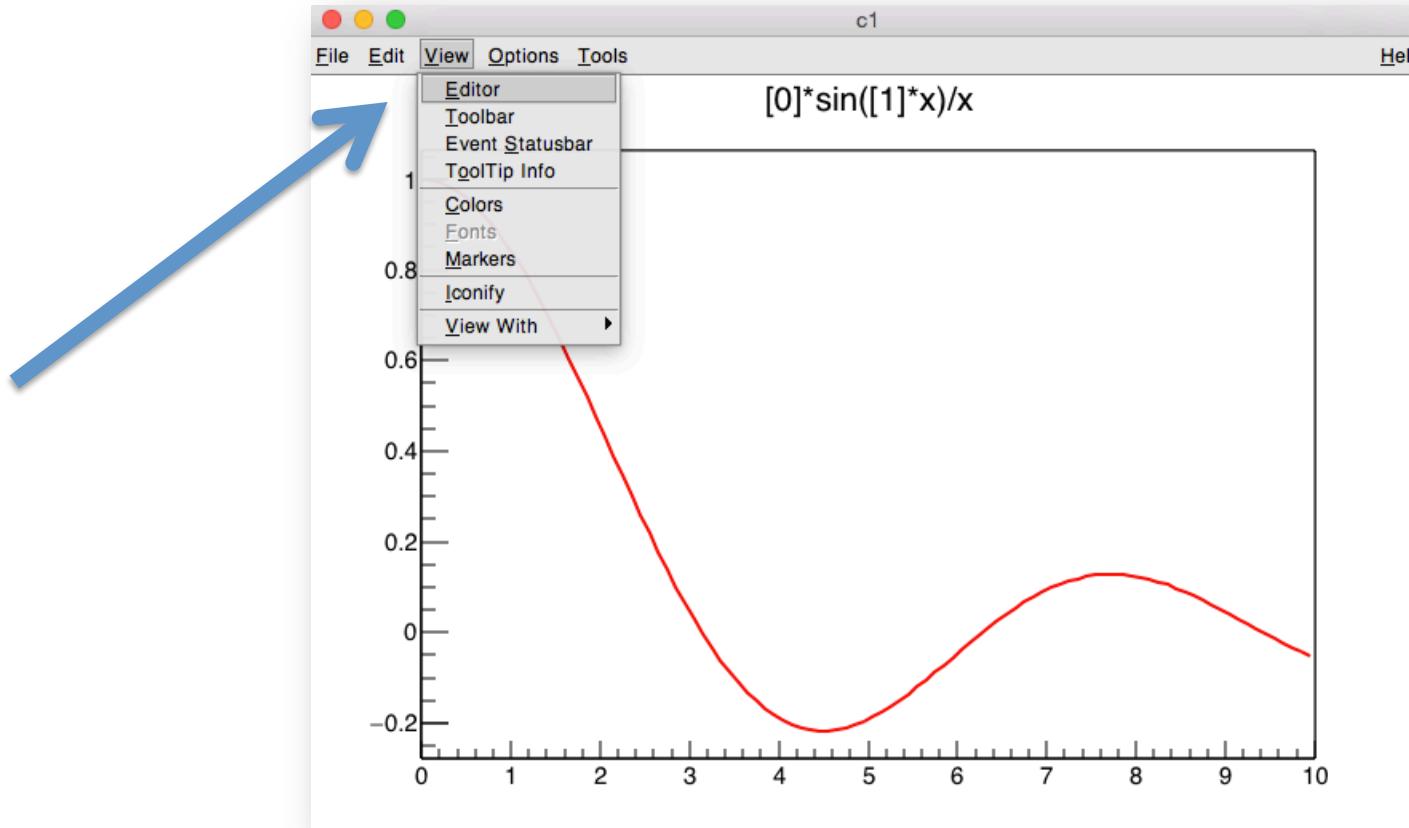
An extended ve
with parameters

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

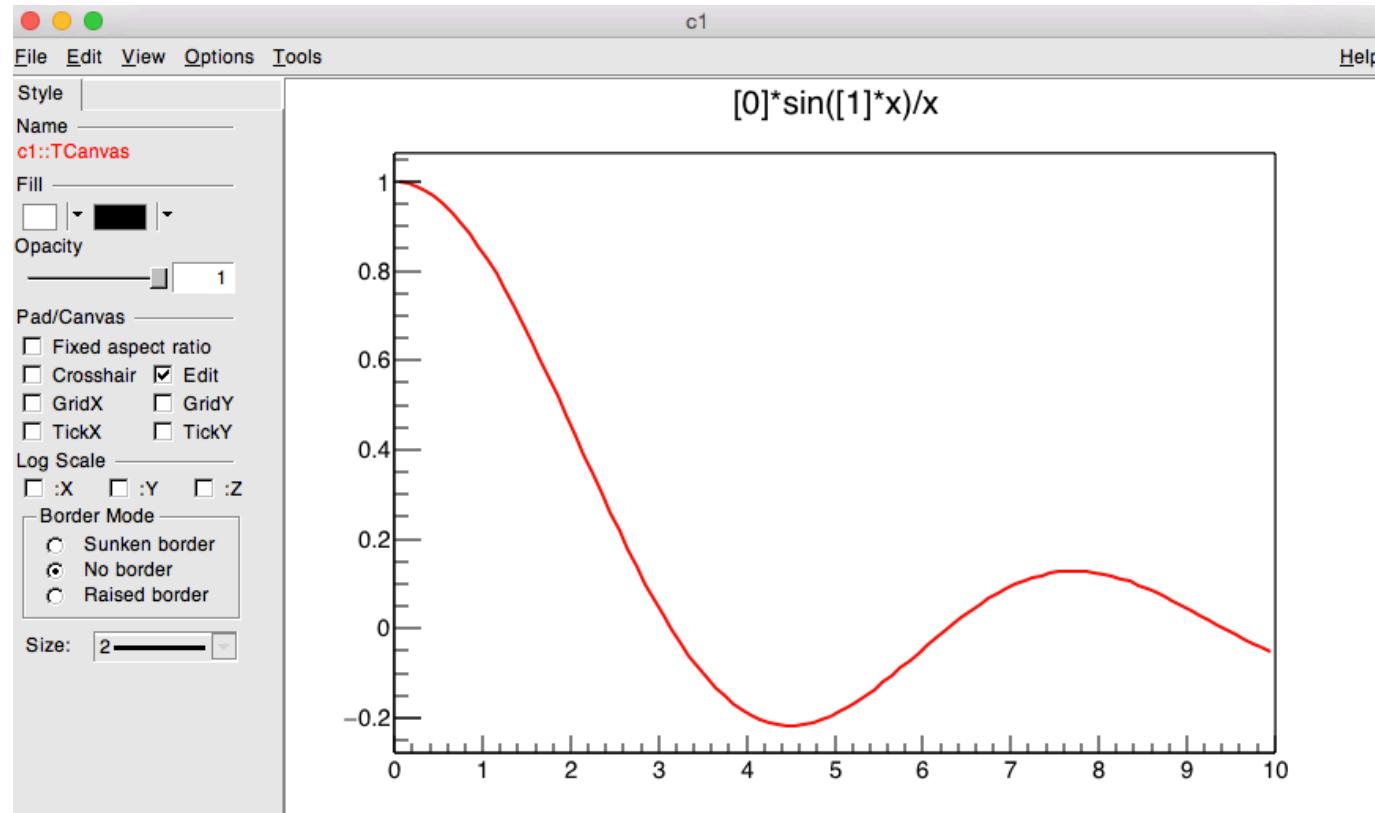


min, max

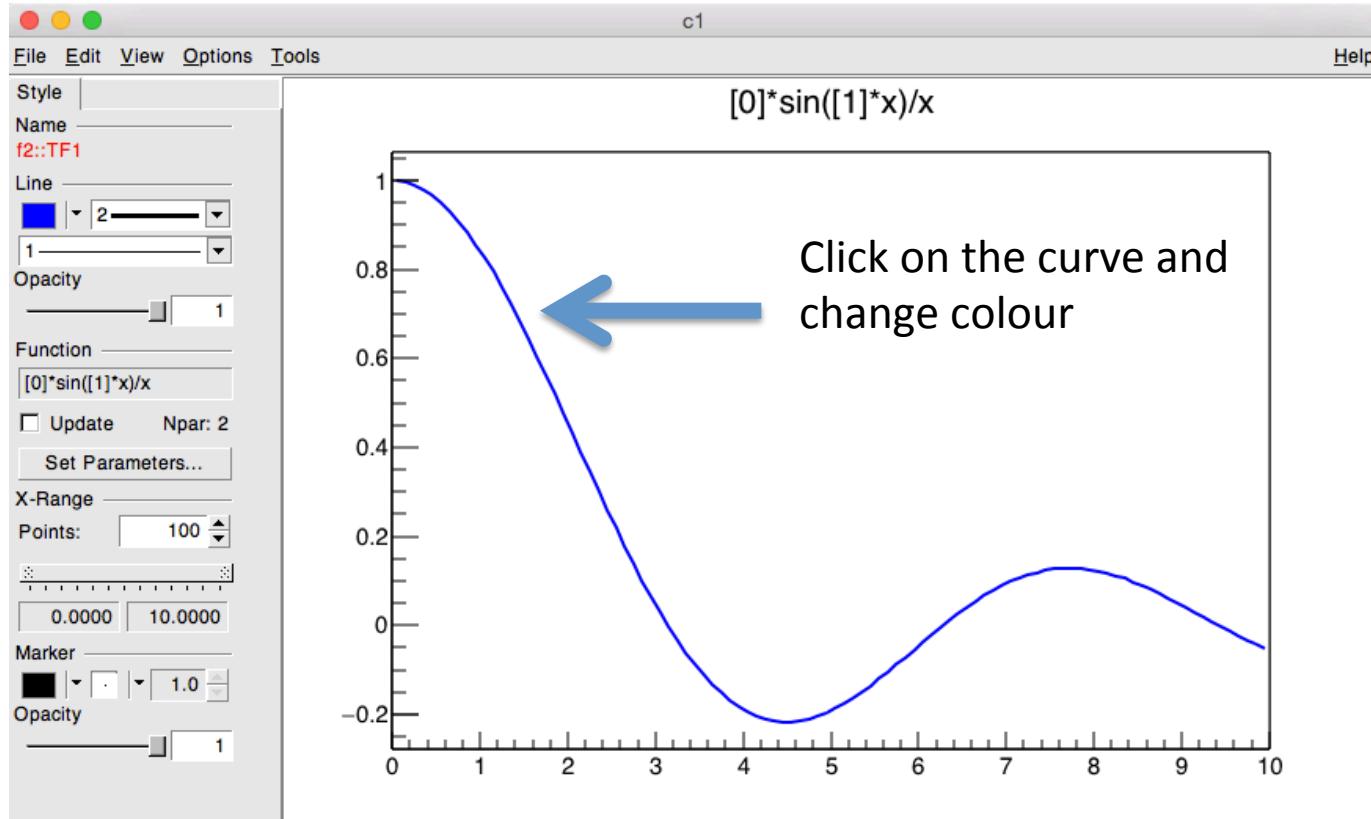
Exercise: Interaction With The Plot



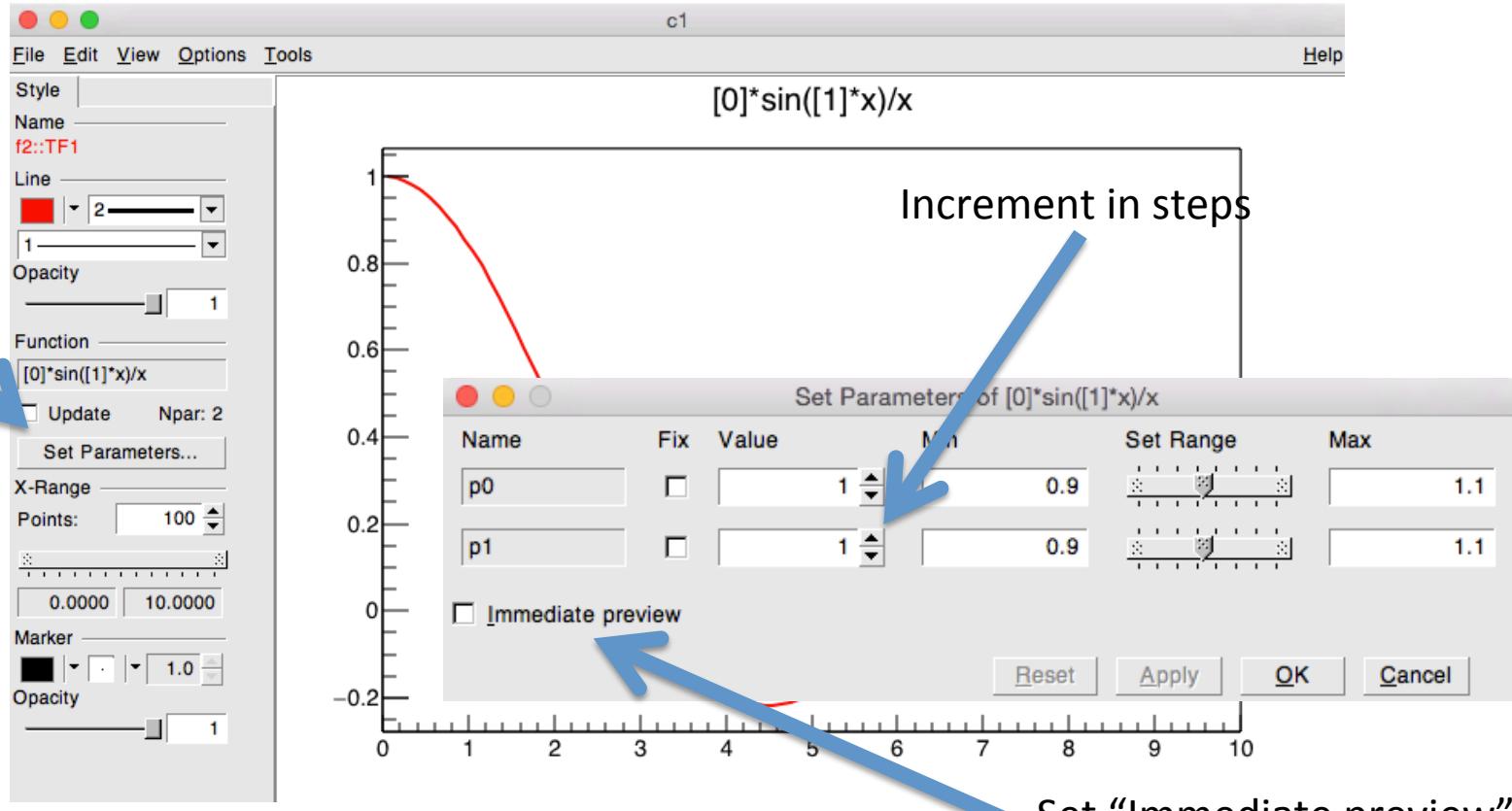
Exercise: Interaction With The Plot



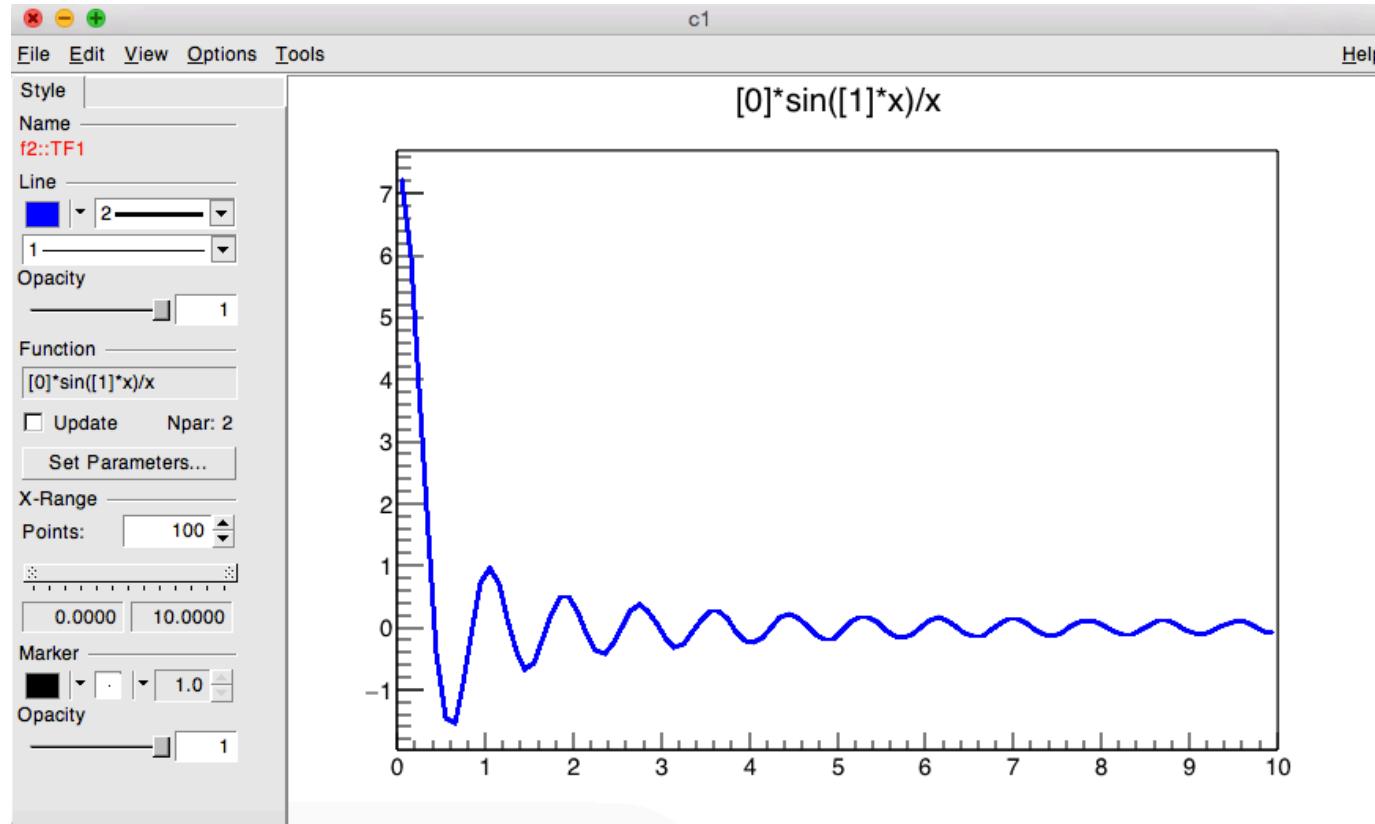
Exercise: Interaction With The Plot



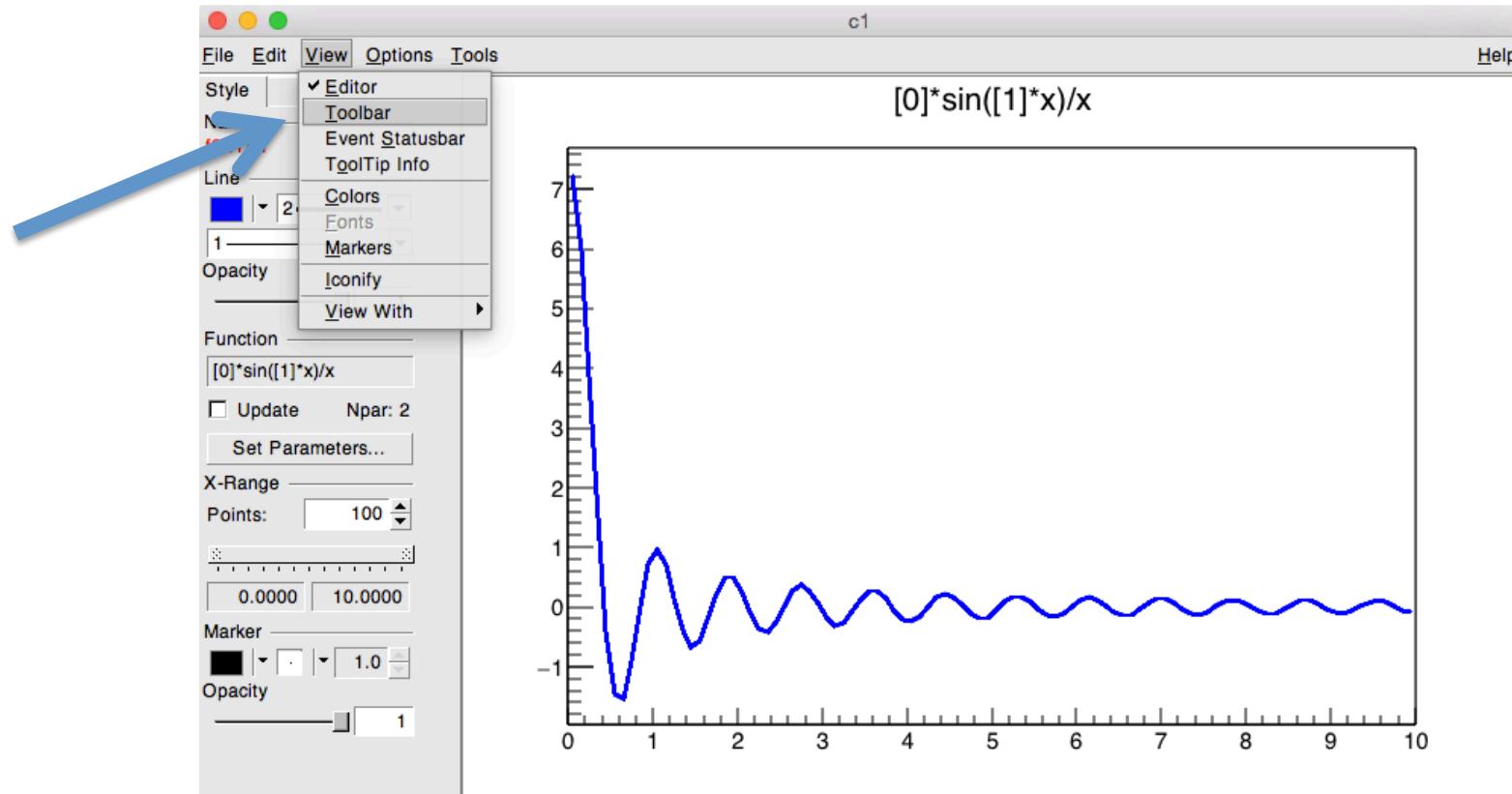
Exercise: Interaction With The Plot



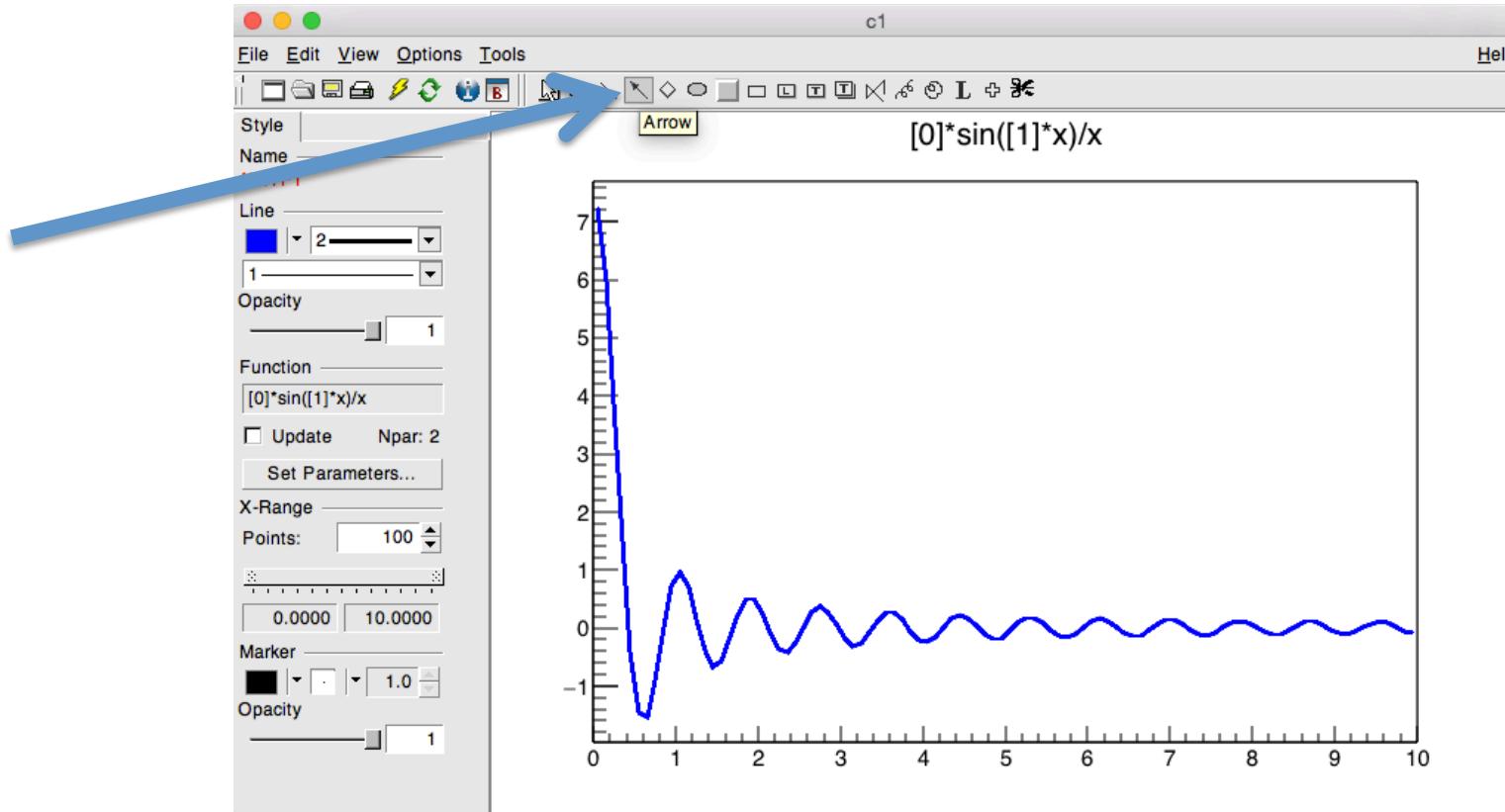
Exercise: Interaction With The Plot



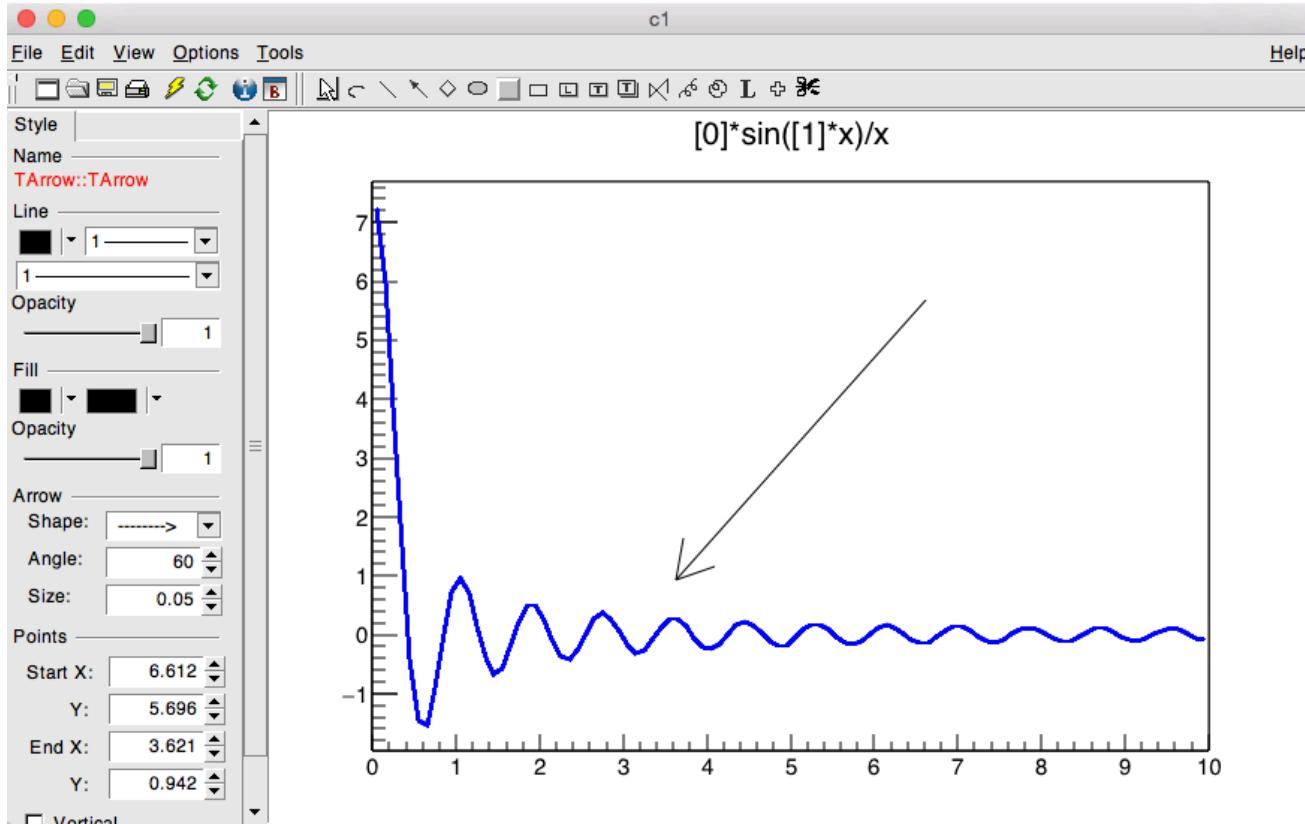
Exercise: Interaction With The Plot



Exercise: Interaction With The Plot

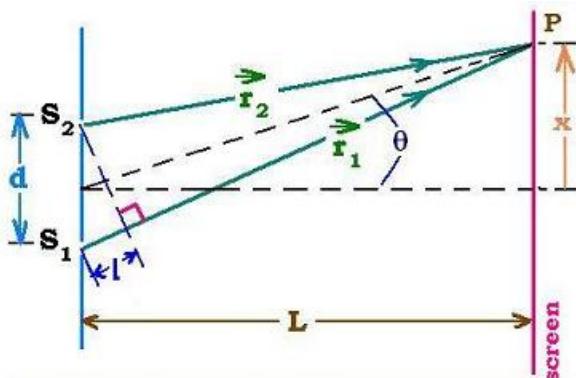


Exercise: Interaction With The Plot

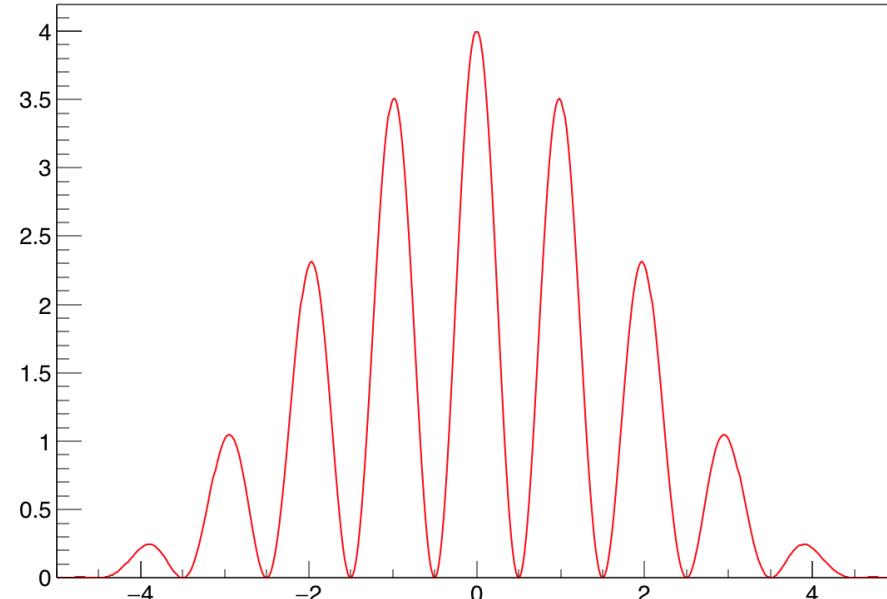


ROOT As a Function Plotter

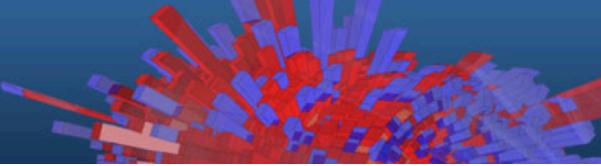
The example **slits.C**, detailed in the Primer, is a more complex C++ program calculating and displaying the interference pattern produced by light falling on a multiple slit.



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



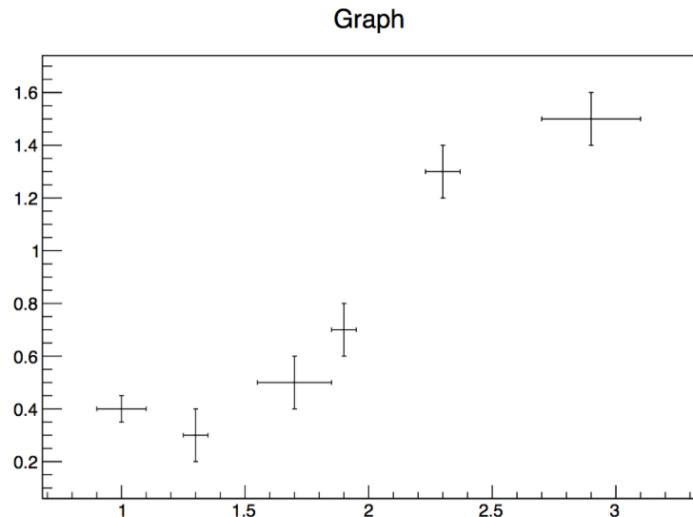
Plotting Measurements



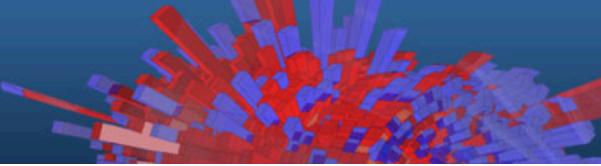
The class `TGraphErrors` allows to display measurements, including errors, with different types of constructors. In the following example, data are taken from the file `ExampleData.txt`:

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

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



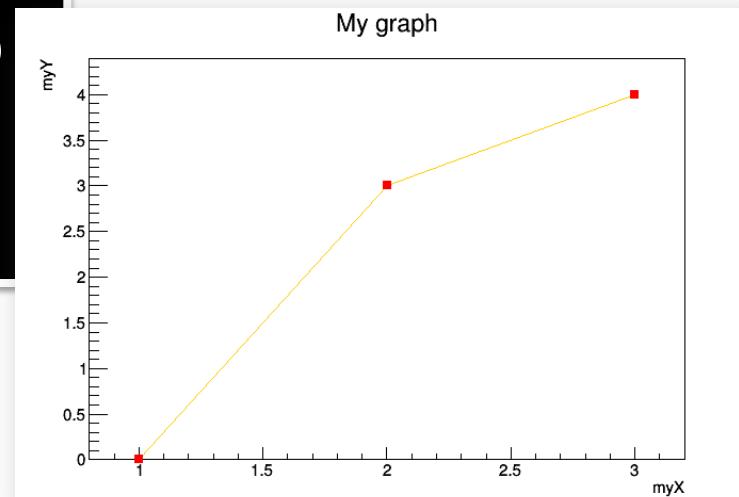
Exercise: TGraph



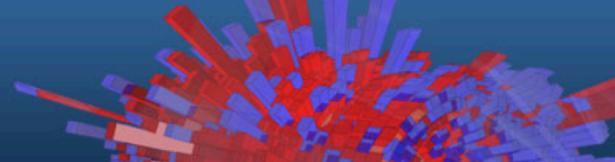
- Create a graph (TGraph)
- Set its title to “My graph”, its X axis title to “myX” and Y axis title to “myY”
- Fill it with three points: (1,0), (2,3), (3,4)
- Set a red full square marker
- Draw an orange line between points

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")
```

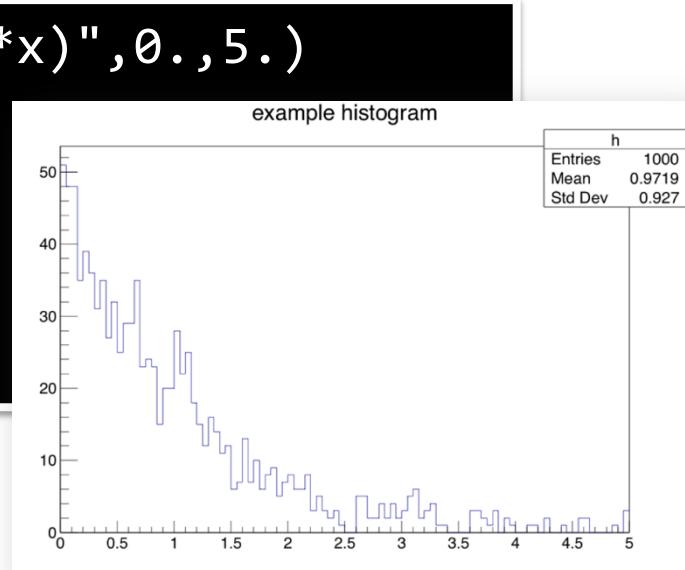


Histograms

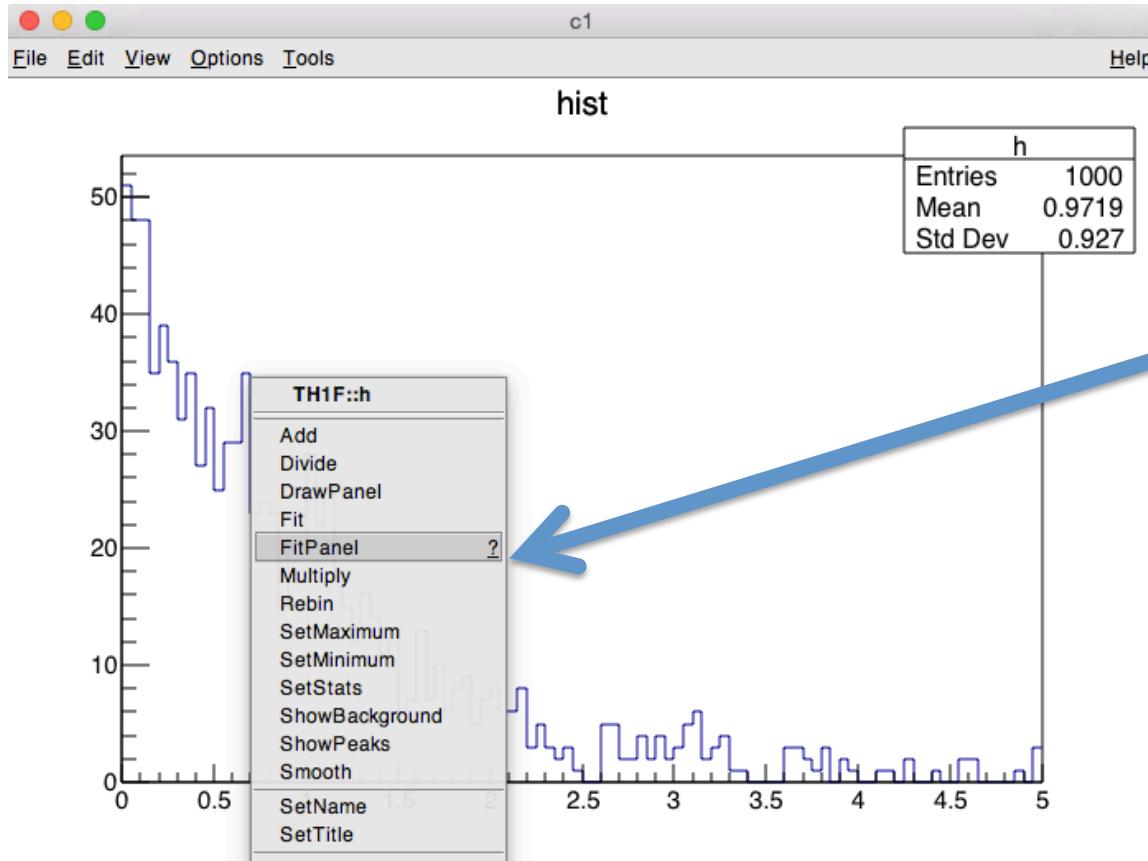
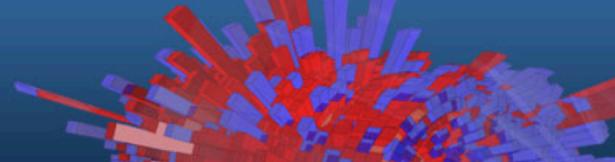


- The TH* classes represent histograms
- TH1* are monodimensional, TH2* are bidimensional ...
- The final letter describes the type stored in each bin:
A double in TH1D, a float in TH1F ...

```
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()
```

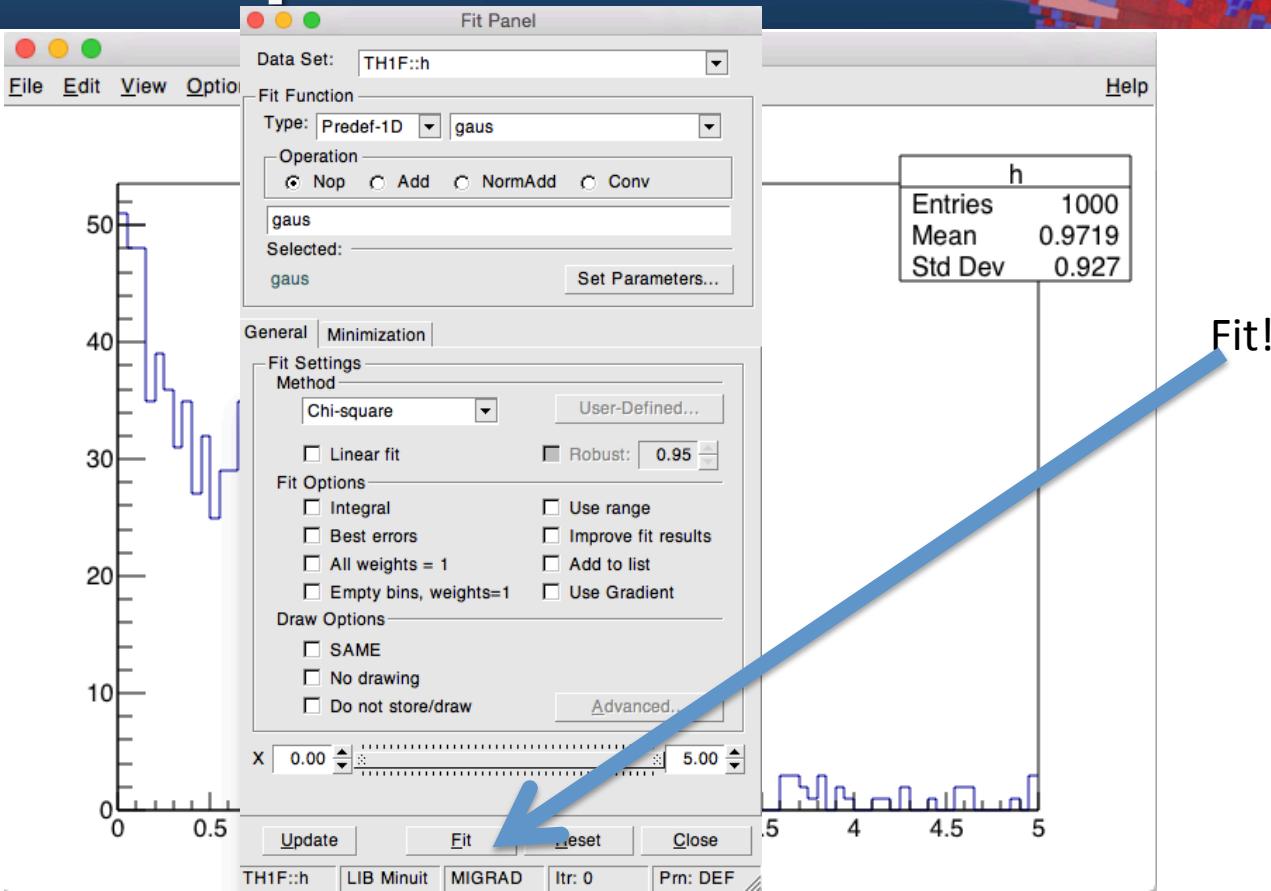


Exercise: Fitpanel

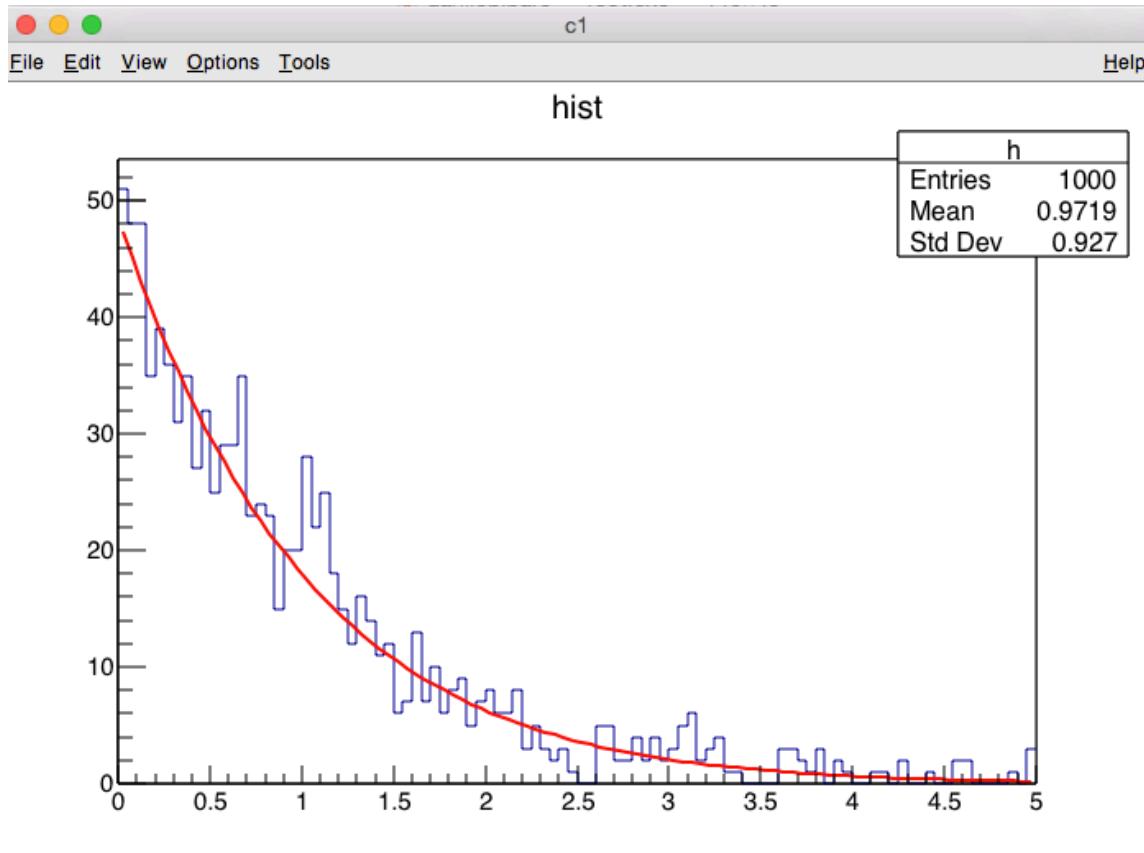
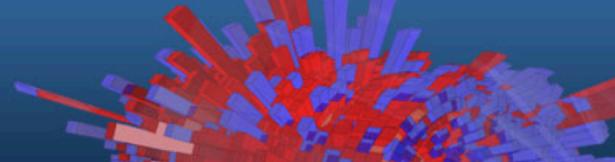


Click on the histogram “line”

Exercise: Fitpanel



Exercise: Fitpanel

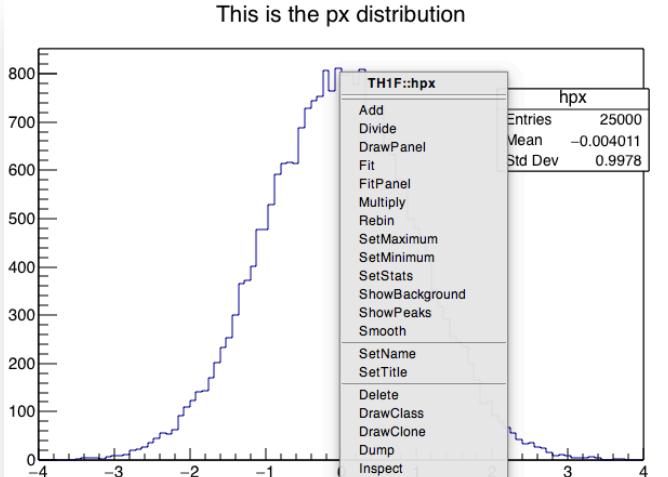
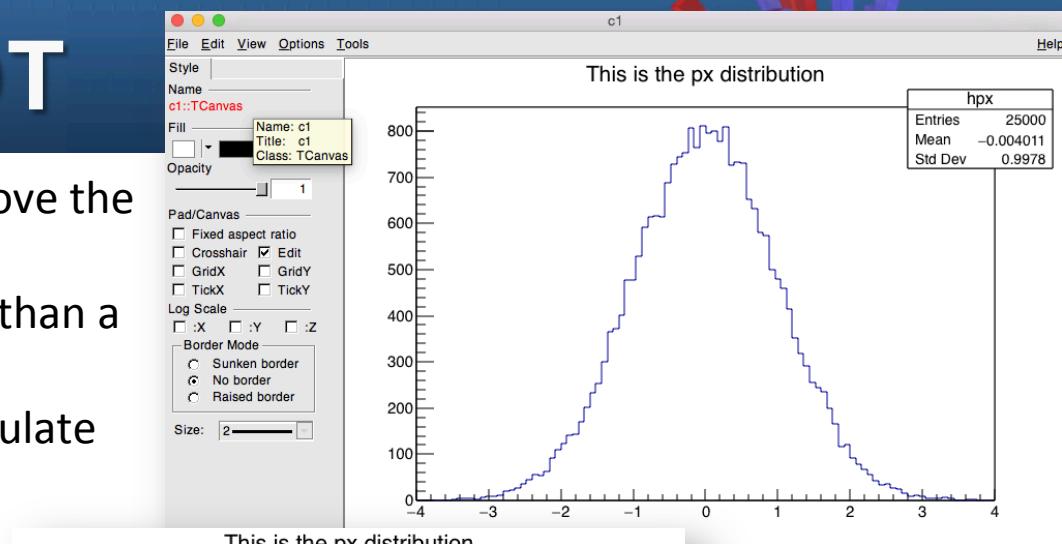
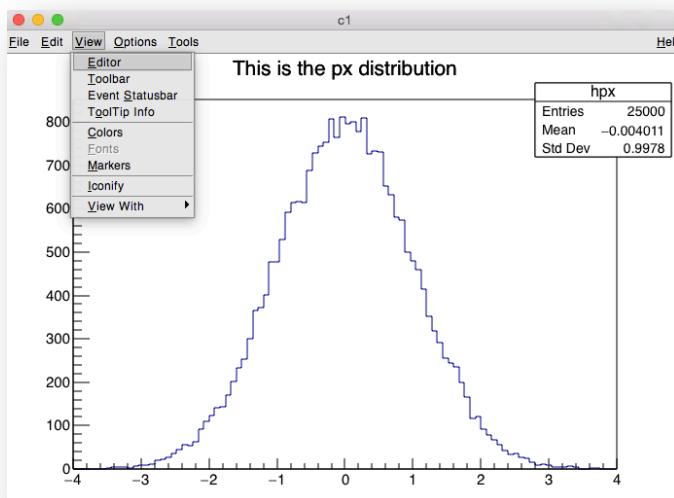


Interactive ROOT

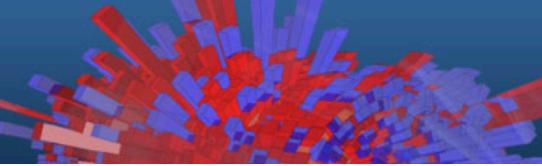
Look at one of your plots again and move the mouse across.

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

Try to interact with objects and manipulate them.

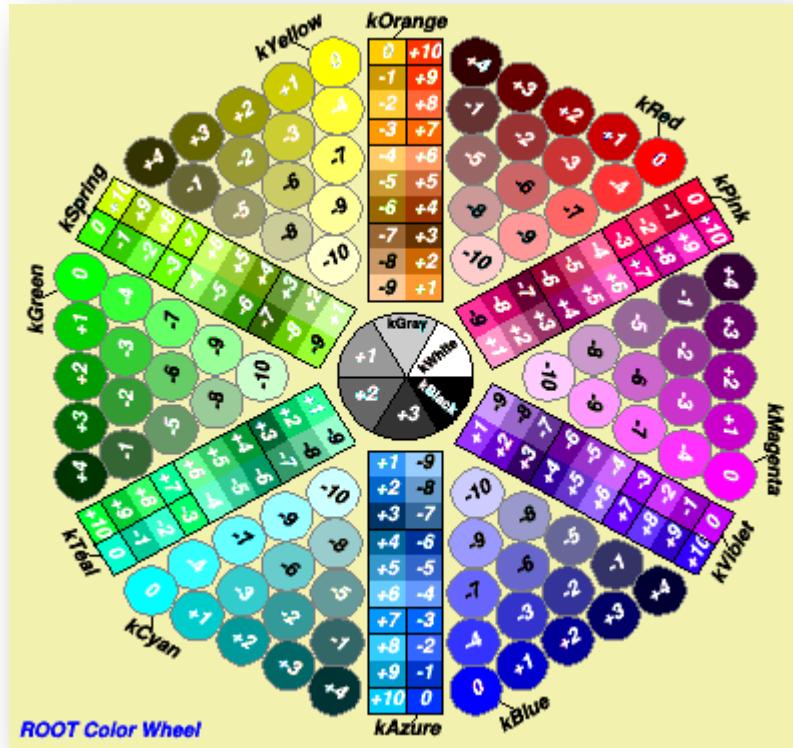
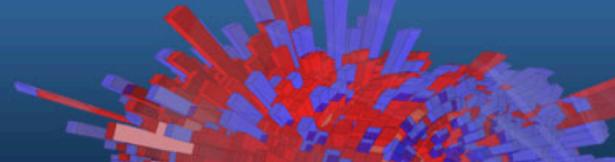


Summary of Visual Effects

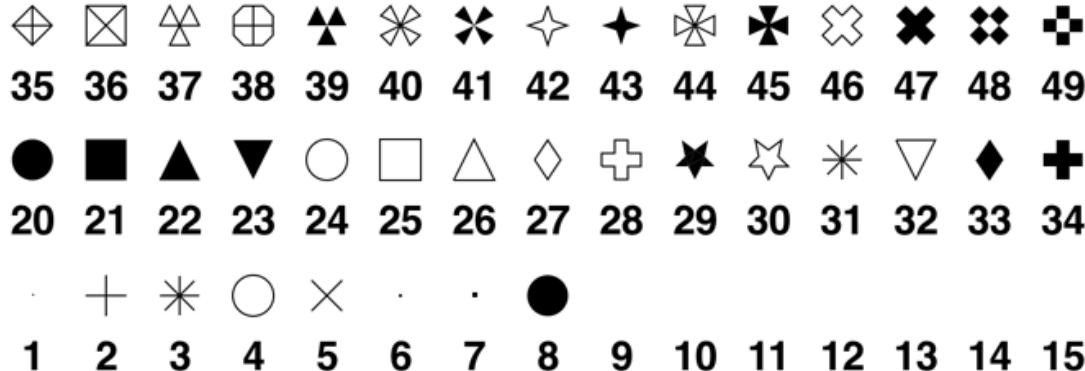
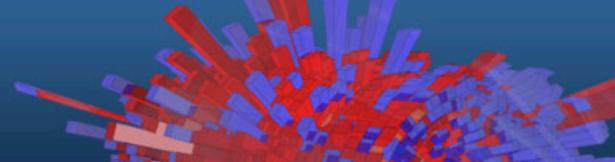


- **Colours and Graph Markers:** To specify a colour, some identifiers like kWhite, kRed or kBlue can be used for markers, lines, arrows etc. The complete summary of colours is represented by the ROOT “colour wheel”. ROOT provides several graphics markers like triangles, crosses or stars.
- **Arrows 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



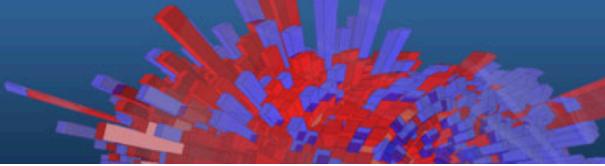
The Family of Markers



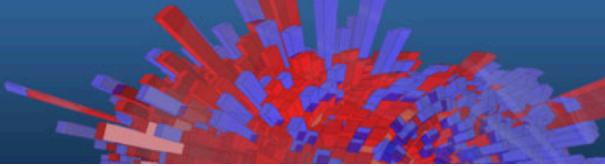
```
kDot=1, kPlus, kStar, kCircle=4, kMultiply=5,  
kFullDotSmall=6, kFullDotMedium=7, kFullDotLarge=8,  
kFullCircle=20, kFullSquare=21, kFullTriangleUp=22,  
kFullTriangleDown=23, kOpenCircle=24, kOpenSquare=25,  
kOpenTriangleUp=26, kOpenDiamond=27, kOpenCross=28,  
kFullStar=29, kOpenStar=30, kOpenTriangleDown=32,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

Also available
through more
friendly names ☺

Break

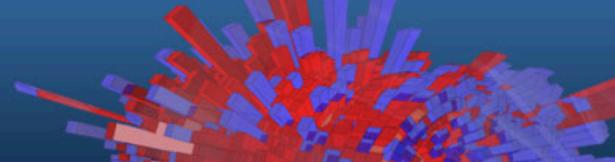


ROOT Macros



- 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

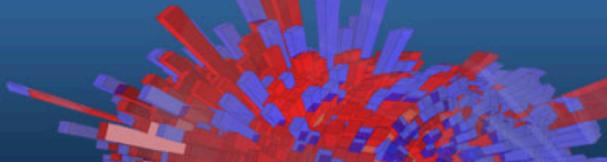
The general structure for a macro stored in file *MacroName.C* is:

**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();
```

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

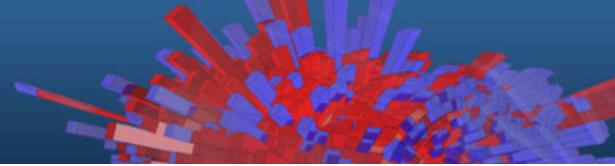
ExampleMacro.c

```
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
```

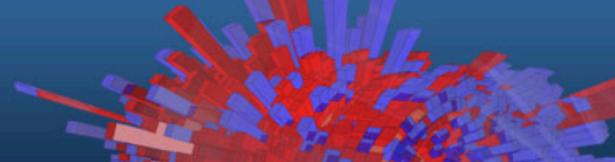
A More Complex Example



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.

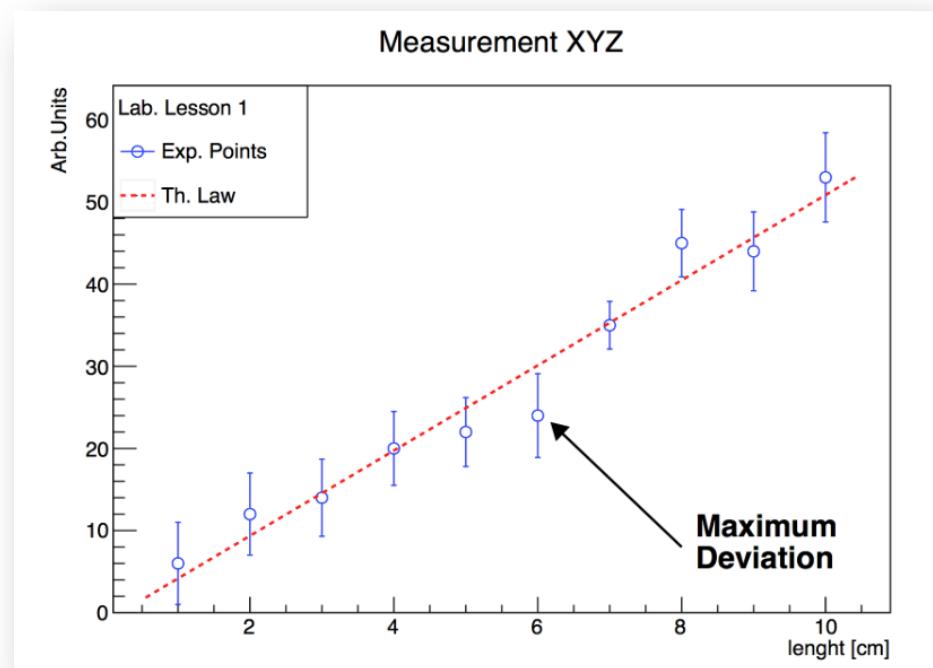
Let's inspect it together.

A More Complex Example



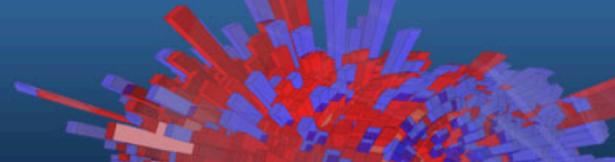
And Run it!

```
> root macro1.C
```



Macro1.cpp

Now Try It in SWAN!



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

ROOT Primer

Click to open the
Macro 1 notebook
in SWAN

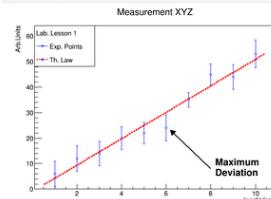


Macro 1: Building a graph with errors

```
In [10]: Storage arrow(9, 8, 6.2, 23, 0.02, ">");  
arrow.DrawLine();  
arrow.DrawClose();  
  
Add some text to the plot.
```

```
In [11]: TLatex text(2, 7.5, "#exp{line}(Maximum)(Deviation)");  
text.DrawClose();
```

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

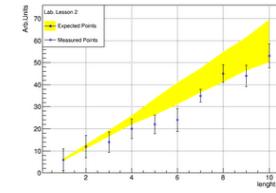


Open In SWAN

Macro 2: Building a graph from a file

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

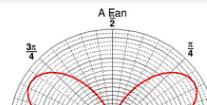
Measurement XYZ and Expectation



Open In SWAN

Macro 3: Polar graph

```
theta(pt) = TMath::Sin((1pt));  
  
In [2]: TGraphPolar gr1(inputs,r,theta);  
gr1.SetTitle("A Fan");  
gr1.SetXTitle("X");  
gr1.SetYTitle("Y");  
gr1.DrawClone("L");  
c.Draw();
```



Open In SWAN

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

```
auto cl = new TCanvas();  
f2.SetName("f2");  
T3D f2.SetLineColor(1);  
T3D f2.SetLineStyle(1);  
TAxis Xaxis = f2->GetXaxis();  
TAxis Yaxis = f2->GetYaxis();  
TAxis Zaxis = f2->GetZaxis();  
Xaxis->SetTitle("X"); Xaxis->SetTitleOffset(1.5);  
Yaxis->SetTitle("Y"); Yaxis->SetTitleOffset(1.5);  
Zaxis->SetTitle("Z"); Zaxis->SetTitleOffset(1.5);  
dts.DrawClone("F", "f2");
```

Display the 2D graph in the notebook.

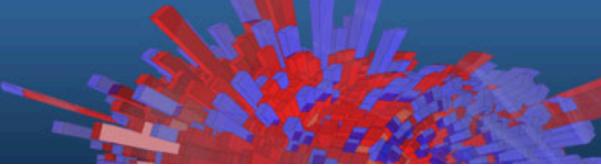
```
In [5]: cl->Draw();
```

Fitted 2D function



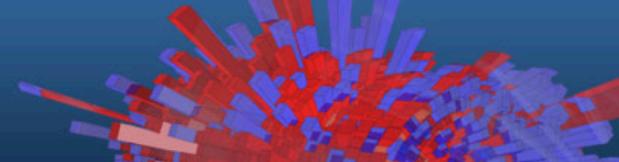
Open In SWAN

Graphs



- Read Graph Points from File
- 2D Graphs
- Also in ROOT primer
 - Polar Graphs
 - Multiple graphs

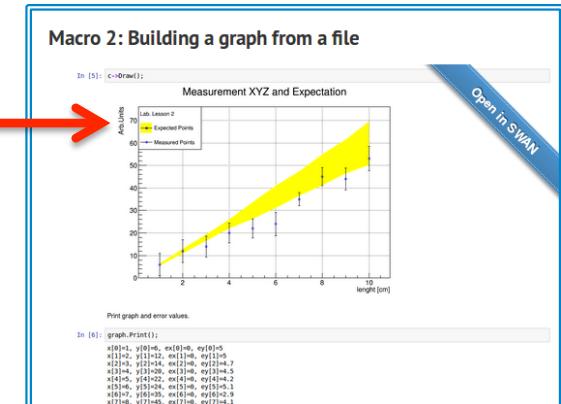
From an ASCII File



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

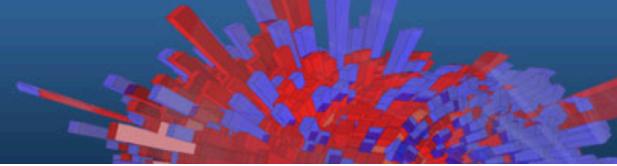
```
TGraphErrors(const char *filename,  
             const char *format="%lg %lg %lg %lg",  
             Option_t *option="");
```

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

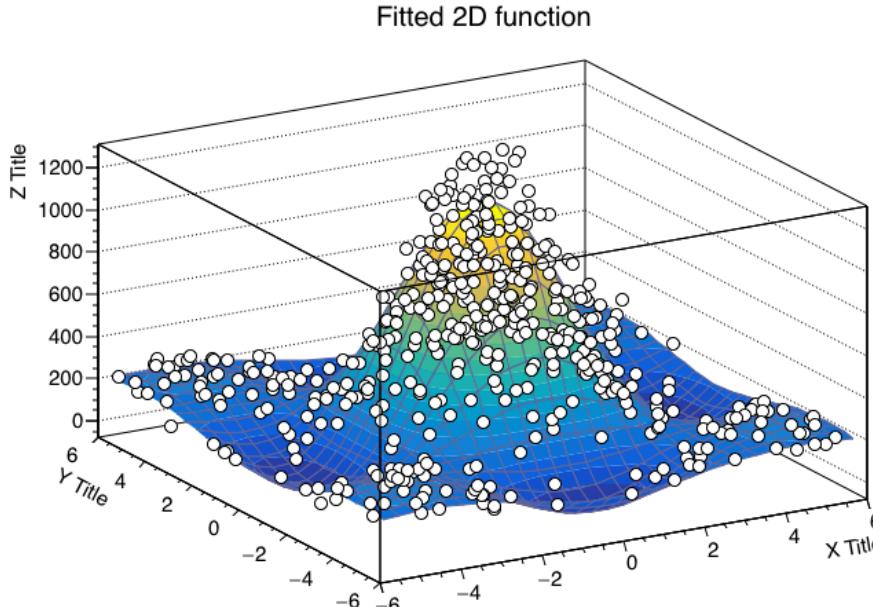


Macro2_cpp

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

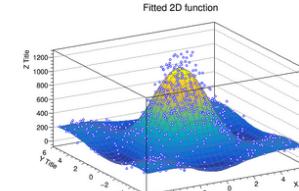


Macro 4: Create, fit and draw a three 3D graph

```
In [4]: note cl = new TCanvas();
T2->SetLineColor(1);
T2->SetLineWidth(1);
T2->SetTitle("Fitted 2D function");
TAxis *xaxis = T2->GetXaxis();
TAxis *yaxis = T2->GetYaxis();
TAxis *zaxis = T2->GetZaxis();
xaxis->SetTitle("X Title");
yaxis->SetTitle("Y Title");
zaxis->SetTitle("Z Title");
xaxis->SetTitleOffset(1.5);
yaxis->SetTitleOffset(1.5);
zaxis->SetTitleOffset(1.5);
xaxis->SetLabelOffset(1.5);
yaxis->SetLabelOffset(1.5);
zaxis->SetLabelOffset(1.5);
cl->DrawClone("No Border");
```

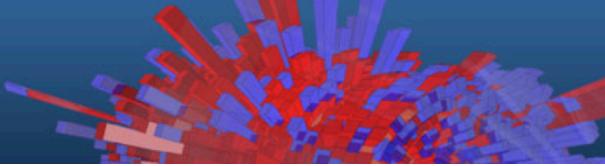
Display the 2D graph in the notebook.

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



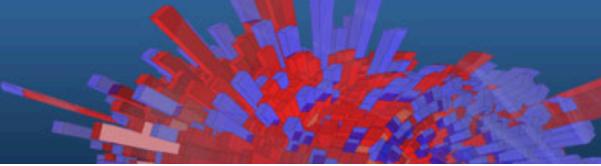
Open in SWAN

Histograms



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

Exercise

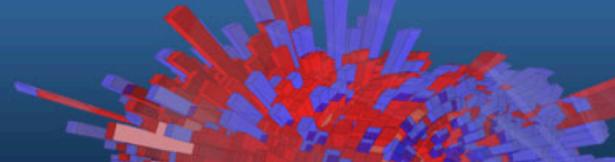


Write a macro to visualise a Poisson distribution in a histogram

- 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

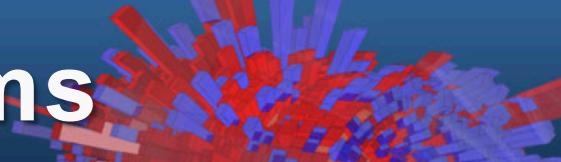


Create a macro that draws the sum, difference and ratio of two histograms

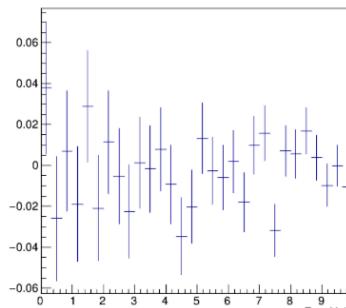
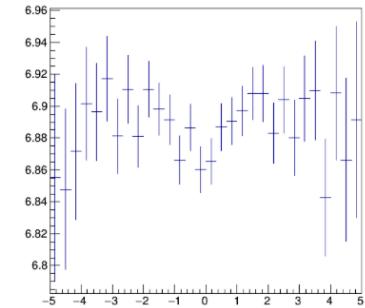
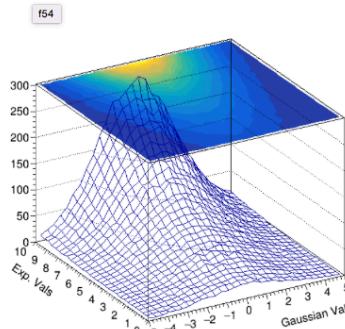
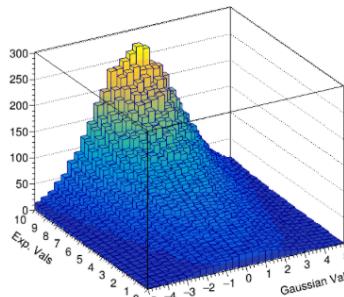
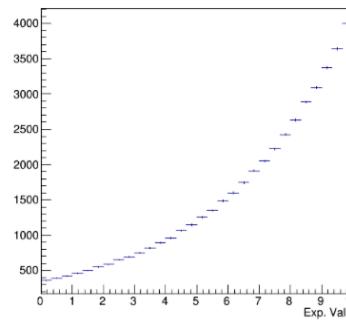
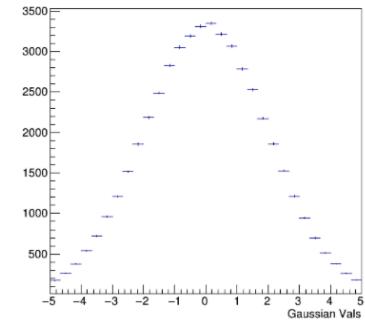
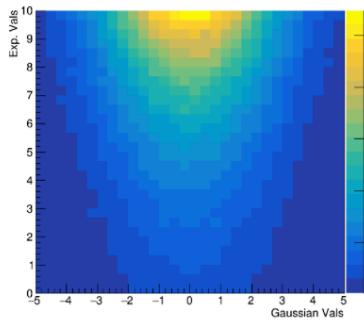
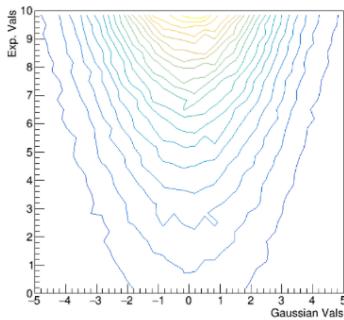
- Create three pairs of histograms, fill them randomly with normally distributed numbers (`TH1::FillRandom("gaus")`)
- Divide, sum and subtract them
 - Useful methods:
`TH1::Divide(const TH1*)`,
`TH1::Add(const TH1*, Double_t)` the second parameter is a weight
- Note: for every plot a different canvas 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

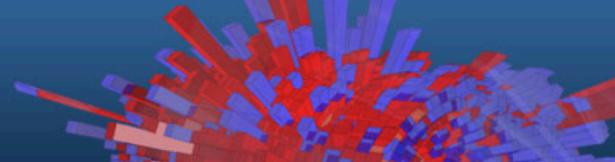
Two Dimensional Histograms



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):

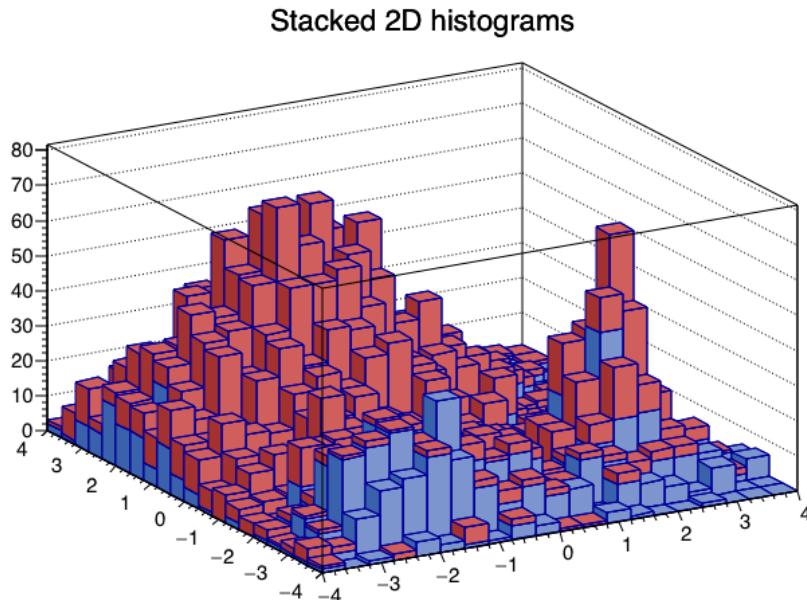


Multiple Histograms



- The example in section 5.4 (hstack.C) shows how to group histograms in a single entity call a “stack”
- Useful when plotting several backgrounds and a signal

**Class
THStack**



Hstack_cpp



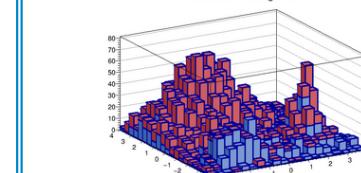
THStack: Draw stacks of several histograms

```
h2ta.FillRandom("t1", 4000);  
TF2 f2("f2", "exp(a0 + exp(b0*x) * (c0*x^2 + d0*x^3 + e0*x^4))";  
double p[params2] = {100, -1, 1.5, 1.1, 1.2, 90, 2, 0.7, -2, 0.5};  
TGraph g2(params2, p);  
TH1 h2tb("h2tb", "h2tb"; -20, -4, 4, 20, -4, 4);  
h2tb.FillRandom("t2", 3000);
```

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

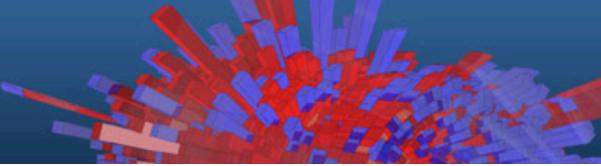
```
In [3]: theStack.Add(h2ta1);  
theStack.Add(h2tb1);  
TCubes c;  
theStack.Draw();  
c.Draw();
```

Stacked 2D histograms



Open In SWAN

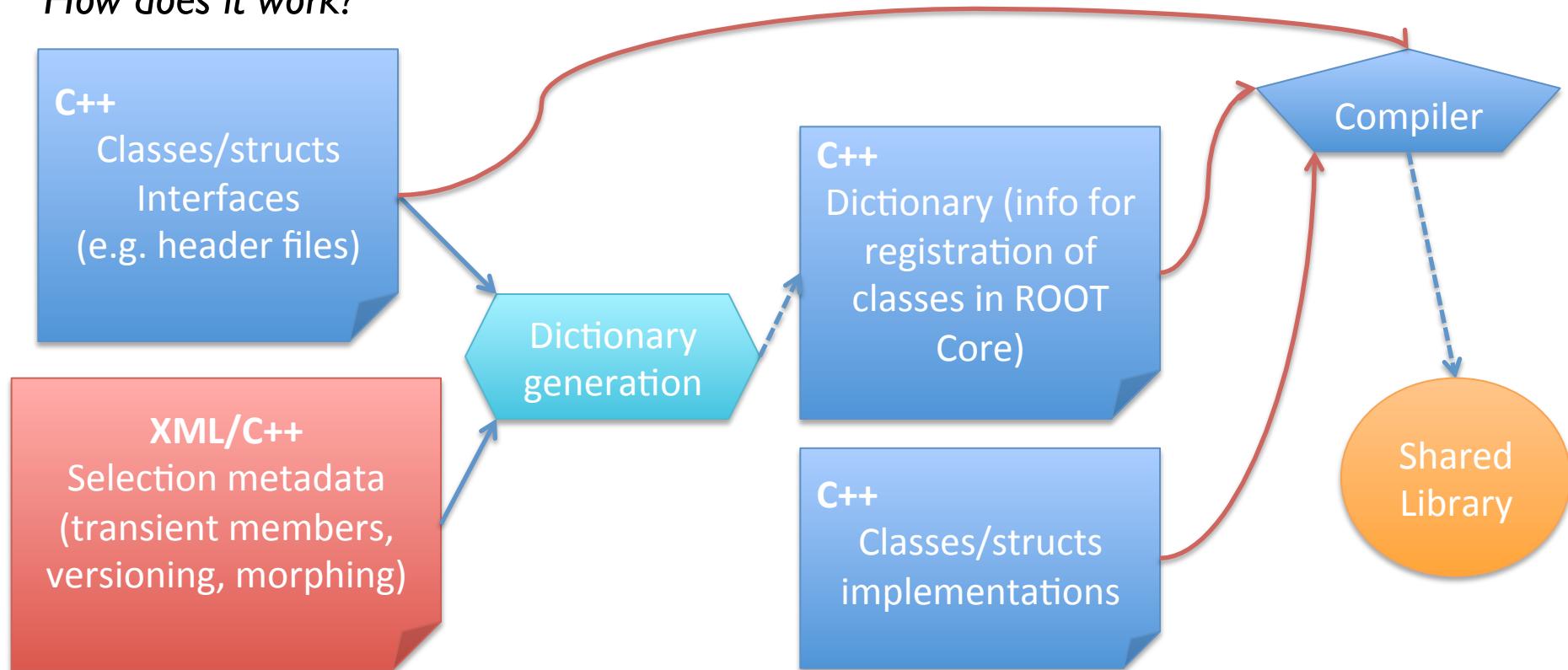
Input and Output



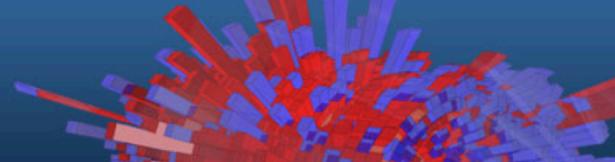
- Storing Objects
- TTrees and N-tuples

Persistency (I/O)

How does it work?

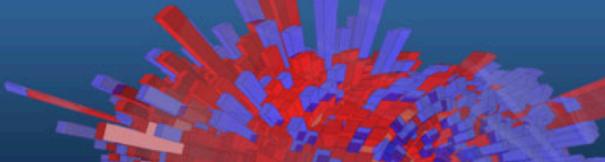


Storing Objects in a File



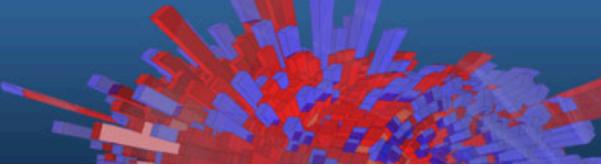
- ROOT allows to store C++ objects on disk (natively the language cannot)
- All ROOT objects (inheriting from TObject) can be written on disk via the Write method.
- Two ways of storing: row wise (single object dump) and column wise (N-tuple like storage).
- Feature widely used, e.g. by all LHC experiments

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
```

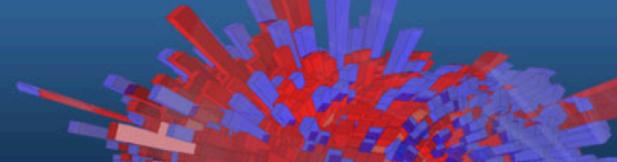
Exercise



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

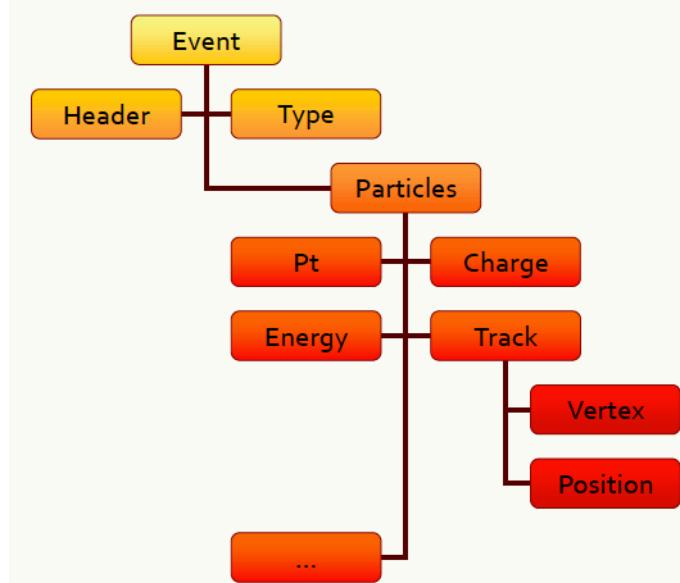
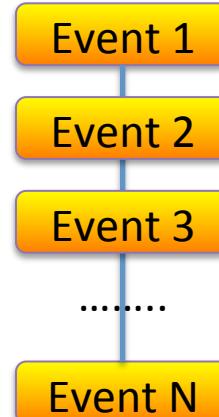
Trees



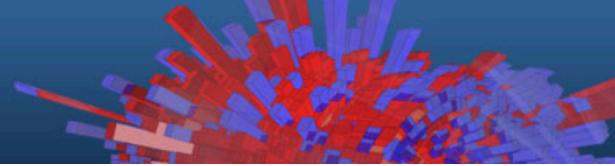
- The TTree is the data structure ROOT provides to store large quantities of same types objects
- Organised in branches, each one holding objects
- Organised in independent events, e.g. collision events
- Efficient disk space usage, optimised I/O runtime

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)

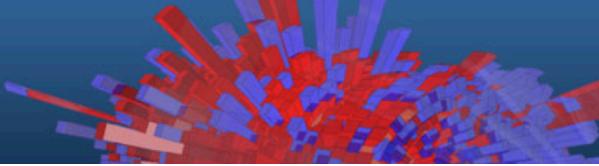


Ntuples



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

Example



Primer macro (section 7.2.1)
write_ntuple_to_file.C



write_ntuple_to_file.cpp

```
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;++) {
    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();
```

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 in different conditions of pressure and temperature) and writing it to a file.

Ported to Notebook by: Thies Hansen

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

The Tuple

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

In [1]:

```
TH1*file("conductivity.root","RECREATE");
TTree*cond_data("cond_data","Example N-tuple","Potential:Current:Temperature:Pressure");
```

Then we fill randomly 100 simulated data using the Fibonaci random generator. We are also applying some "smearing" (measurement errors): 1% error on voltage (volt), pressure and current and 1.3 absolute error on temperature. At the end of the loop body we fill the ntuple.

In [2]:

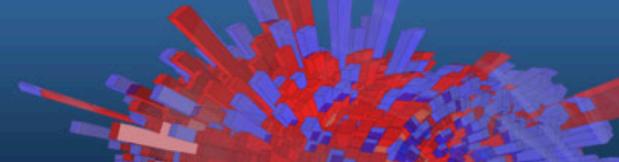
```
#Random numbers
float volt,curr,temp,press;
for (int i=0;i<100;i++){
    volt=randf.Uniform(0.,10.);
    curr=randf.Uniform(0.,1000000000.);
    temp=randf.Uniform(0.5,1.5);
    press=randf.Uniform(0.5,1.5);
    curpot=(temp-0.05)*(temp-300.)-0.2*(pres-1);
    pot=volt*curpot;
    temperr=(temp-0.5)*0.01;
    preserr=(pres-0.5)*0.01;
    currerr=(curr-1000000000.)*0.01;
    cond_data.Fill(pot,cur,temp,press);
}
```

Save the TH1file and close the file.

In [3]:

```
cond_data.Close();
ofile.Close();
```

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/395198/material/Bin.root");
TTreeReader myHeader("events",f);
TTreeReaderValue<vector<float>> muEta(myHeader, "muEta");
TTreeReaderValue<vector<float>> muPhi(myHeader, "muPhi");
TTreeReaderValue<vector<float>> muPt(myHeader, "muPt");

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]: #include <iostream>
#include <vector>
#include <algorithm>

auto myHeader.Next();
auto myHeader.EventNum();

auto tracks = &trackN;
for (auto& track : tracks){
    if (track.pt() > maxPt) maxPt = track.pt();
}
if (eventNum % 100 == 0) {
    std::cout << "Processing event number " << eventNum << std::endl;
    std::cout << "Max pt is " << maxPt << std::endl;
}

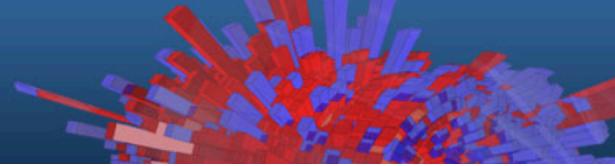
Processing event number 0
Max pt is 37.5852
processing event number 100
Max pt is 32.400
processing event number 200
Max pt is 32.400
processing event number 300
Max pt is 32.400
processing event number 400
Max pt is 35.2018
processing event number 500
```

Open in SWAN!



[TTreeReader_Example_cpp](#)

TDataFrame



- Modern way to interact with columnar data (e.g. TTrees) with a **functional approach**: avoid all boilerplate code!
- Deal with transformations and actions, e.g. filters, creation of new columns and histograms, profiles
 - Inspired by tools such as Spark and Pandas
- Output data in a new file
- One line to enable **implicit parallelism!**

New in version 6.10!

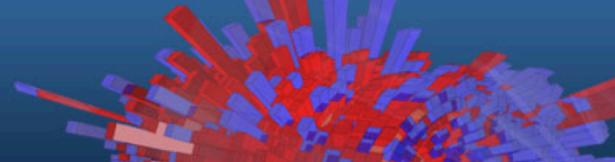
TDataFrame documentation:

https://root.cern/doc/master/classROOT_1_1Experimental_1_1TDataFrame.html

TDataFrame tutorials:

https://root.cern/doc/master/group__tutorial__tdataframe.html

TDataFrame



```
TTreeReader data(tree);
TTreeReaderValue<A> x(data, "x");
TTreeReaderValue<B> y(data, "y");
TTreeReaderValue<C> z(data, "z");

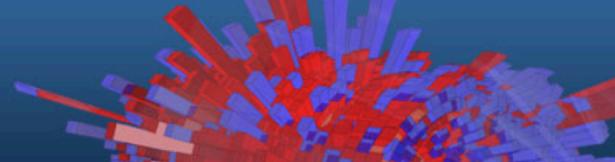
while(data.Next()) {
    if (IsGoodEvent(*x,*y,*z)) {
        DoStuff(*x,*y,*z);
    }
}
```



```
TDataFrame(tree, {"x", "y", "z"});
data.Filter(IsGoodEvent)
    .Foreach(DoStuff);
```

- users have full control over the event-loop
- ✓ needs some boilerplate
- ✓ running the event-loop in parallel is not trivial
- ✓ users implement trivial operations again and again

One Line to Go Parallel



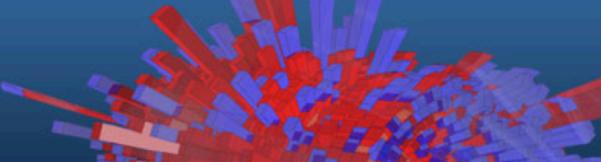
```
ROOT::EnableImplicitMT();  
TDataFrame(tree, {"x", "y", "z"});  
data.Filter(IsGoodEvent)  
    .Foreach(DoStuff);
```

ROOT will parallelise
the operations which
come afterwards
automatically



The runtime ROOT presently leverages
to achieve multithreading is TBB.

Cut and Fill Example



```
bool IsPos(double x) { return x > 0.; }
bool IsNeg(double x) { return x < 0.; }
TDataFrame d("tree", "file.root");
auto h1 = d.Filter(IsPos, {"theta"}).Histo1D("pt");
auto h2 = d.Filter(IsNeg, {"theta"}).Histo1D("pt");
h1->Draw();           // event loop is run once here
h2->Draw("SAME");    // no need to run loop again here
```

Create Columns and Write to Disk

```
// We read the tree from the file and create a TDataFrame
// The tree has one single branch, b1
ROOT::Experimental::TDataFrame d("mytree", "myfile.root");

// ## Select entries
// We now select some entries in the dataset
auto d_cut = d.Filter("b1 % 2 == 0");

// ## Enrich the dataset
// Build some temporary columns: we'll write them out
// Either a string or a function (or lambda or functor) can be used
auto d2 = d_cut.Define("b1_square", "b1 * b1")
.Define("b2_vector",
    [](float b2) {
        std::vector<float> v;
        for (int i = 0; i < 3; i++) v.push_back(b2*i);
        return v;
    },
    {"b2"});
// ## Write it to disk in ROOT format
// We now write to disk a new dataset with one of the variables originally
// present in the tree and the new variables.
// The user can explicitly specify the types of the columns as template
// arguments of the Snapshot method, otherwise they will be automatically
// inferred.
d2.Snapshot(treeName, outFile, {"b1", "b1_square", "b2_vector"});
```

You can Start from Scratch too!

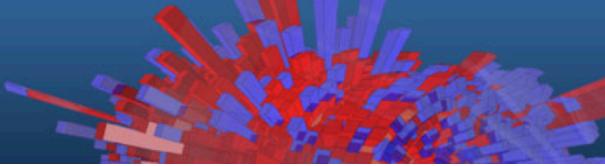
```
// We create an empty data frame of 100 entries
ROOT::Experimental::TDataFrame tdf(100);

// We now fill it with random numbers
TRandom3 rnd(1);
auto tdf_1 = tdf.Define("rnd", [&rnd](){return rnd.Gaus();});

// We plot these numbers
auto hgaus = tdf_1.Histo1D("rnd");

// And we write out the dataset on disk
tdf_1.Snapshot("randomNumbers", "tdf008_createDataSetFromScratch.root");
```

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 and TDataFrame tutorials (both in C++ and Python!)

Entry point to use ROOT from within Python:

```
import ROOT
```

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

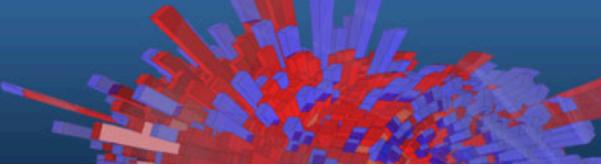
Universal Python Bindings

```
import ROOT
classCode = """
int f(int i) {return i*i;}
class A{
public:
    A(){cout << "Hello PyROOT!" << endl;}
};

# We inject the code in the interpreter
ROOT.gInterpreter.Declare(classCode);

# We find all the C++ entities in Python!
a = ROOT.A()
# this prints Hello PyROOT!
ROOT.f(3)
# this returns 9
```

Exercise



- Open the Python interpreter (type `python`)
- Import the ROOT module
- Create a histogram with 64 bins and a x axis ranging from 0 to 16
- Fill it with random numbers distributed according to a linear function (“`pol1`”)
- Change its line width with a thicker one
- Draw it!

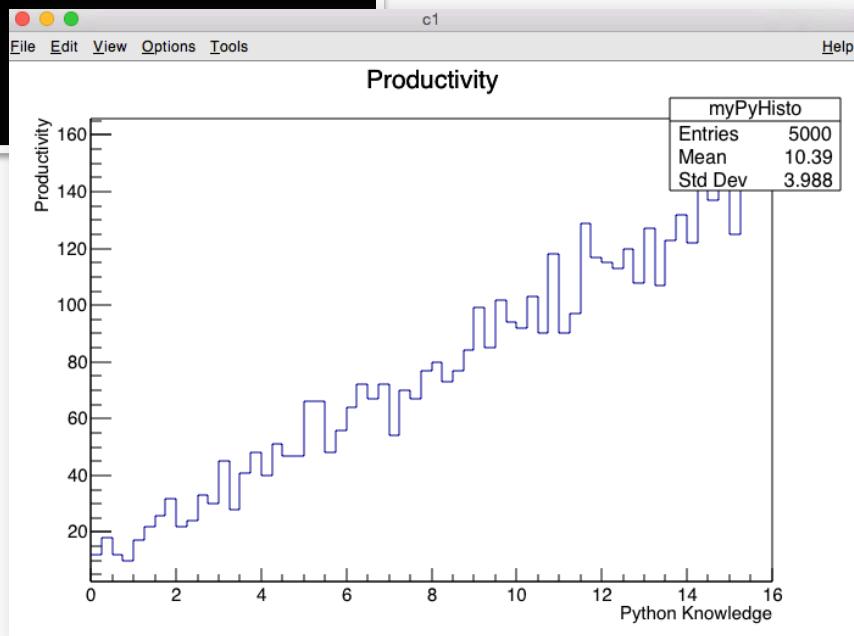
...or try it in **SWAN!**

<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 and examples

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



FillHistogram_Example_py

TTreeAccess_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 [3]: t = ROOT.TFile.Open("http://edie.cern.ch/events/399596/materials/0/0.root");  
In [4]: h = ROOT.TH1F("track", "Tracks;#eta", 128, -64, 64)  
for event in t.Events:  
    for track in event.tracks:  
        track.SetMarkerSize(1)  
        h.Fill()  
c = ROOT.TCanvas()  
c.Draw()
```

Tracks

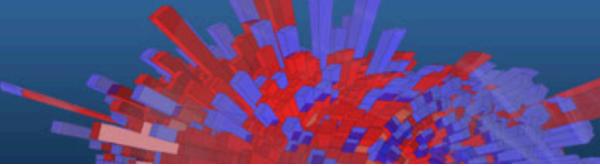
Events: 496932 Mean: 12.54 Std Dev: 6.554

The histogram shows a peak at approximately 12.54, with a standard deviation of about 6.554. The x-axis is labeled "#eta" and ranges from 0 to 60, while the y-axis ranges from 0 to 16.

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 and fit data
- Perform basic I/O operations
- Go parallel with ROOT