ROOT

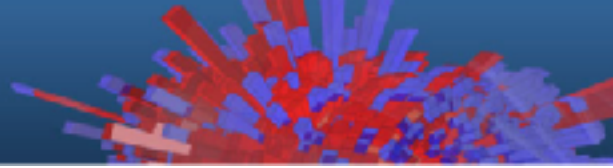
Data Analysis Framework

Sanofi Mini-Course 2017

Sergei Gleyzer, Lorenzo Moneta

CERN EP-SFT

This Course



Objectives for today:

- Become familiar with the ROOT toolkit
- Be able to use the ROOT Python interface
- Learn about SWAN and Jupyter notebooks
- Plot and analyse data
- Perform basic I/O operations

Format:

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

This Tutorial

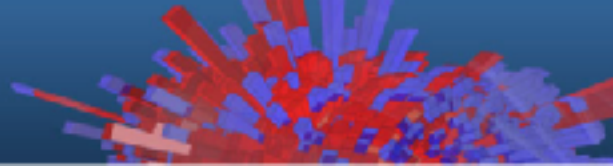
These slides are supported by the **“ROOT Primer”**

- 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 6.08

How Can I Run ROOT?



You can choose between three options:

1. Local ROOT installation
   - https://root.cern.ch/downloading-root
2. 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
3. ROOT Virtual Machine (for training course)
   - https://github.com/root-mirror/training

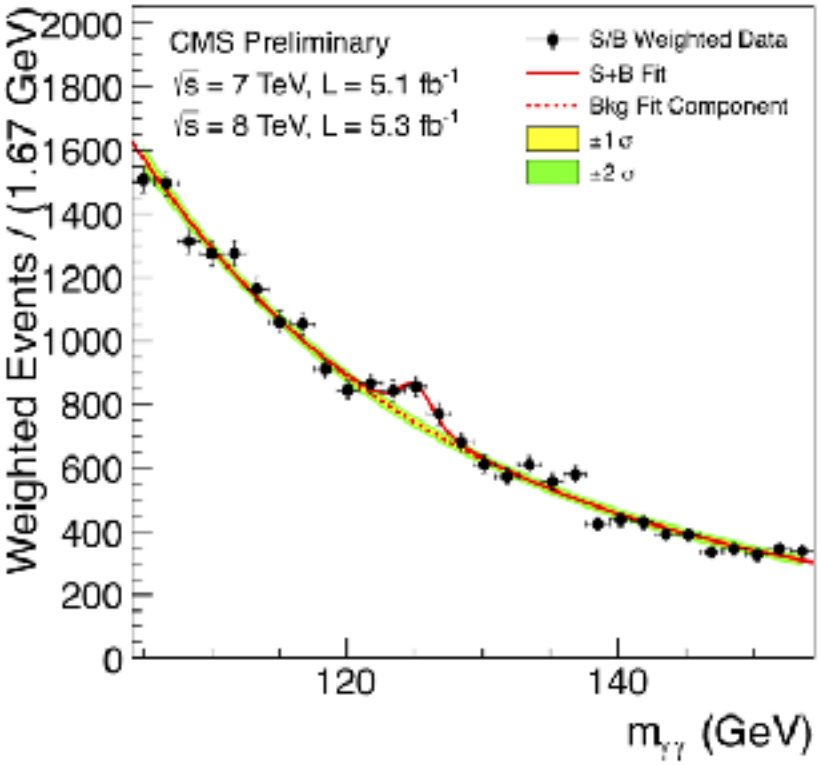
A “Quick Tour” Of ROOT

ROOT: Open Source Project

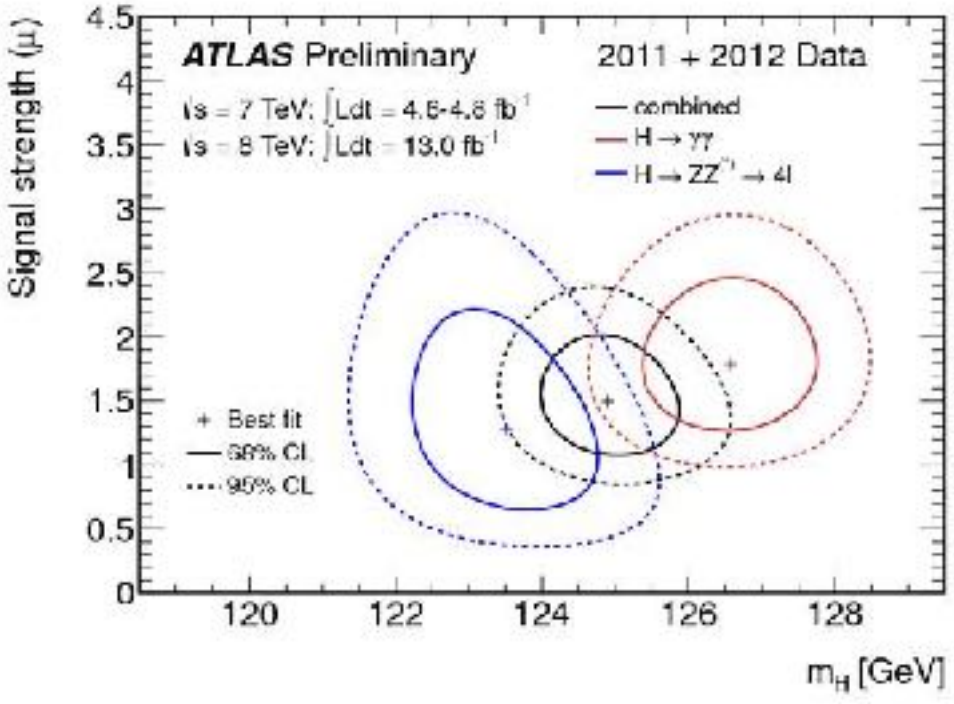
ROOT is an open source project started at CERN in 1995

- Available under the GNU LGPL license
- 5-10 full time developers at CERN plus 1 at Fermilab (Chicago)
- Many contributors from high energy physics experiments which uses ROOT as base for their software frameworks.
- Large number of part-time developers.
- Several thousands of users (not only form High Energy physics community) giving feedback and a very long list of small contributions.

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!

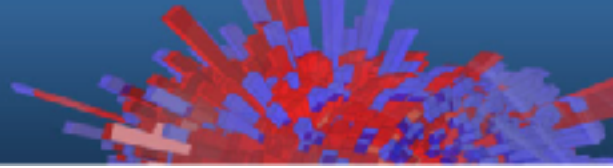
ROOT is written mainly in C++ (C++11 standard)

- Bindings for Python are 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

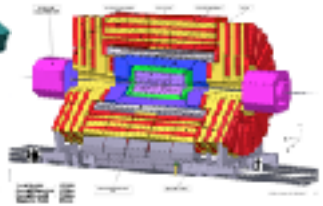


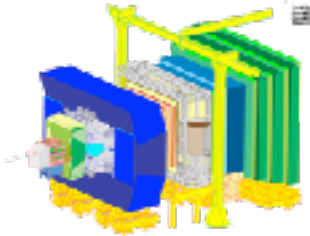
ROOT can be seen as a collection of building blocks for various activities, like:

- 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

Analysis

Offline Processing

Reconstruction

Further processing, skimming

Event Selection, statistical treatment …

Raw

Reco

…

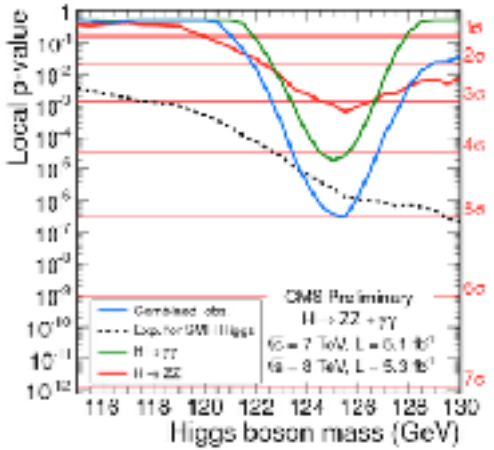
Analysis Formats

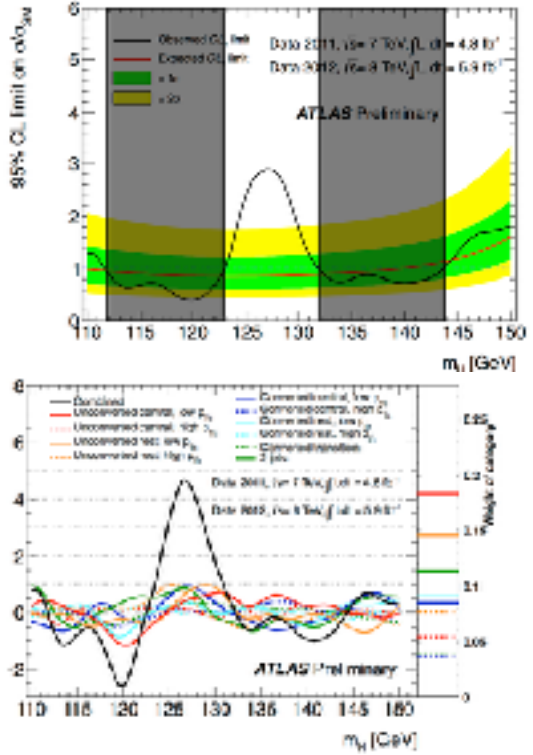
Images

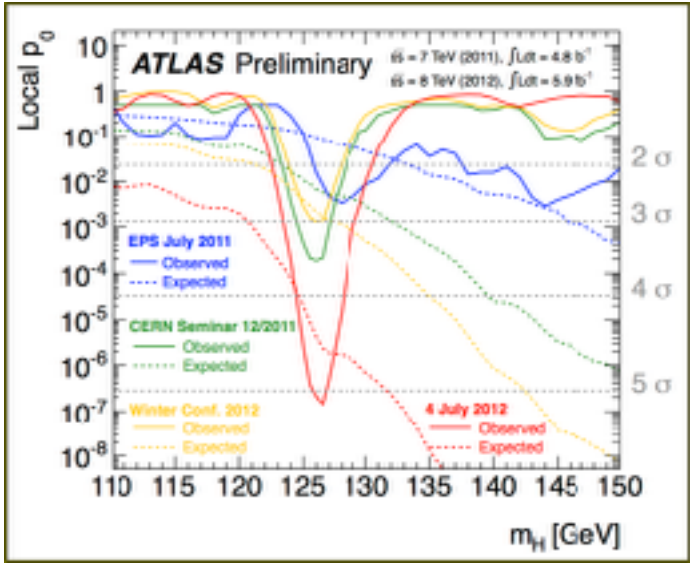
Data Storage: Local, Network

Higgs boson discovery

On July 4th 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.

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

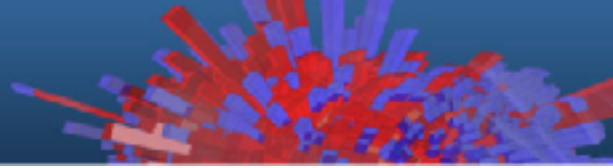
Can interpret “macros” (non compiled programs)

- Rapid prototyping possible

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

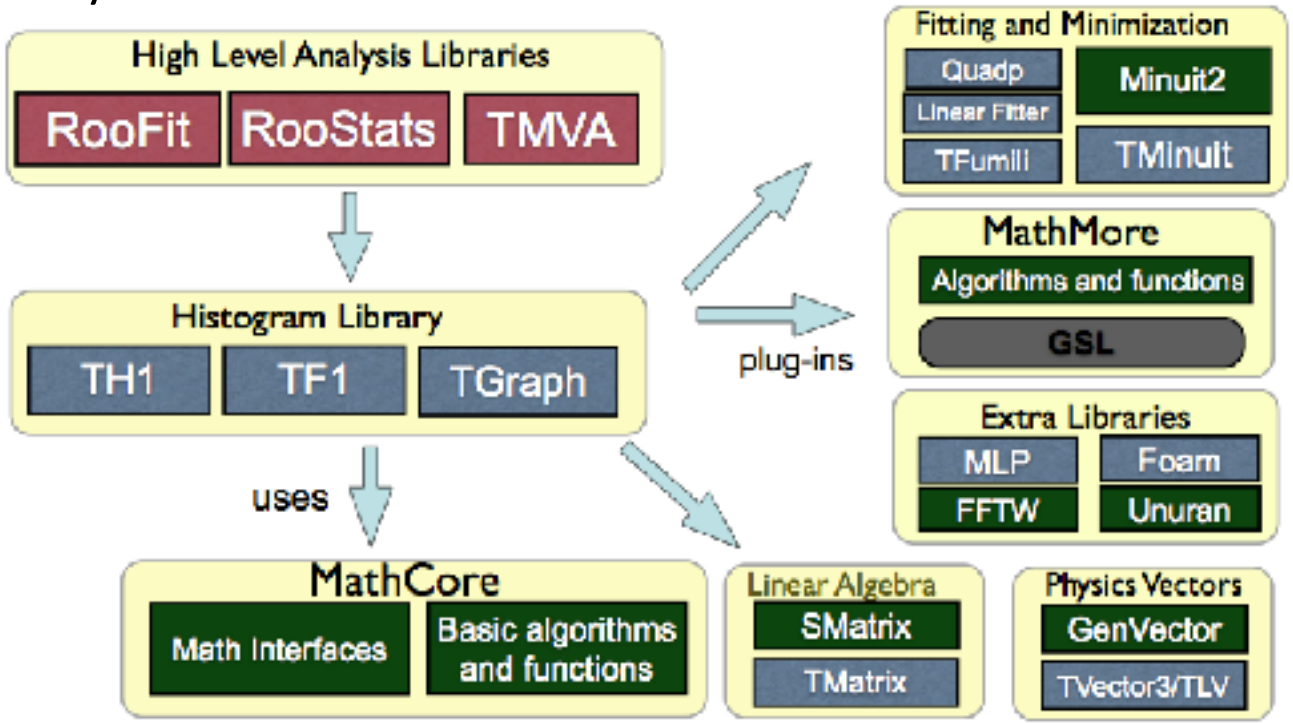
- Raw and column-wise streaming

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

Cornerstone for storage of experimental data

ROOT Math/Stats Libraries

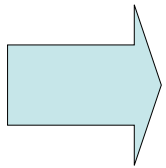
ROOT provides a reach set of mathematical libraries and tools needed for sophisticated statistical data analysis

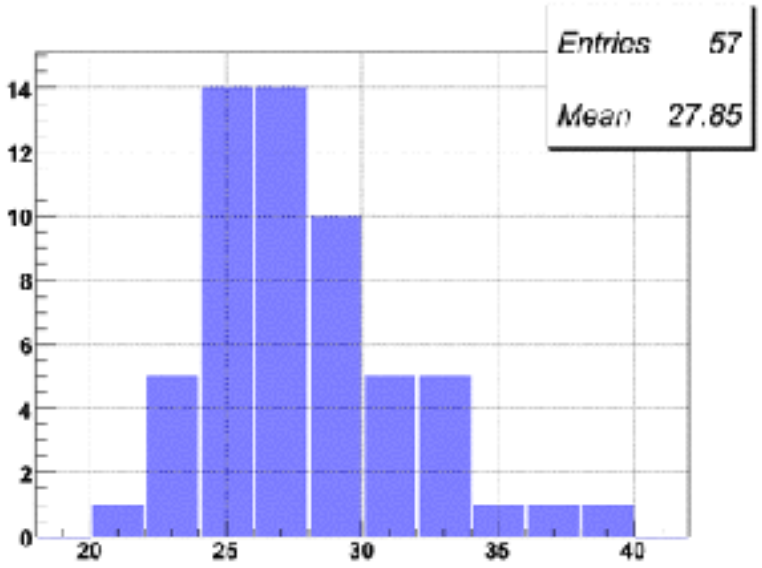


Histograms

Table of Ages (binned)

Age	Number
20-22	1
22-24	5
24-26	11
26-28	11
28-30	10
30-32	5
32-34	5
34-36	1
36-38	1
38-40	1

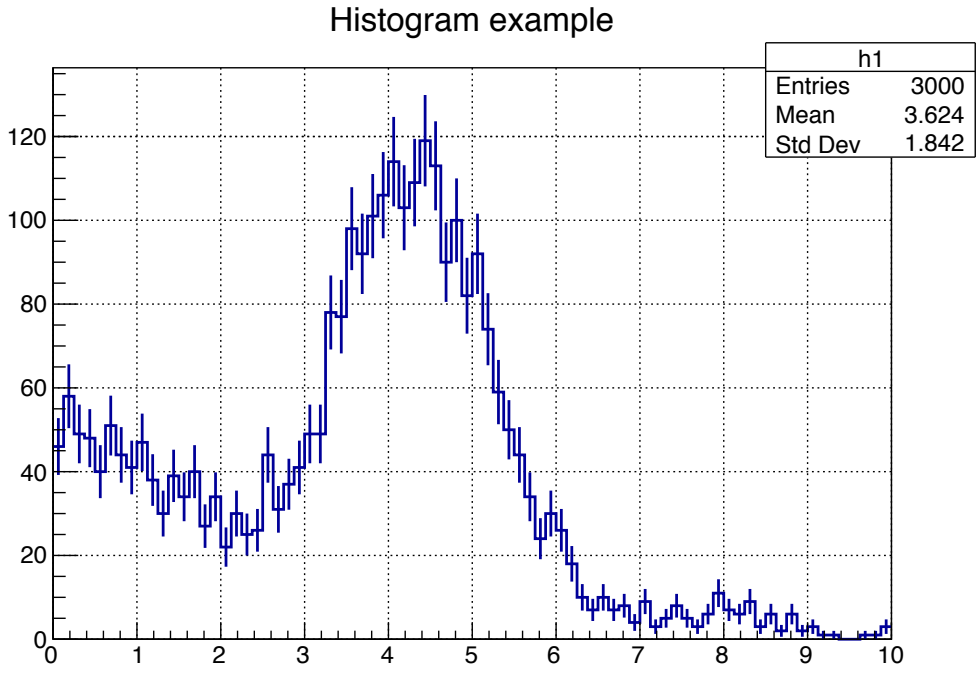




Shows distribution of ages, total number of entries (57 participants) and average: 27 years 10 months 6 days…

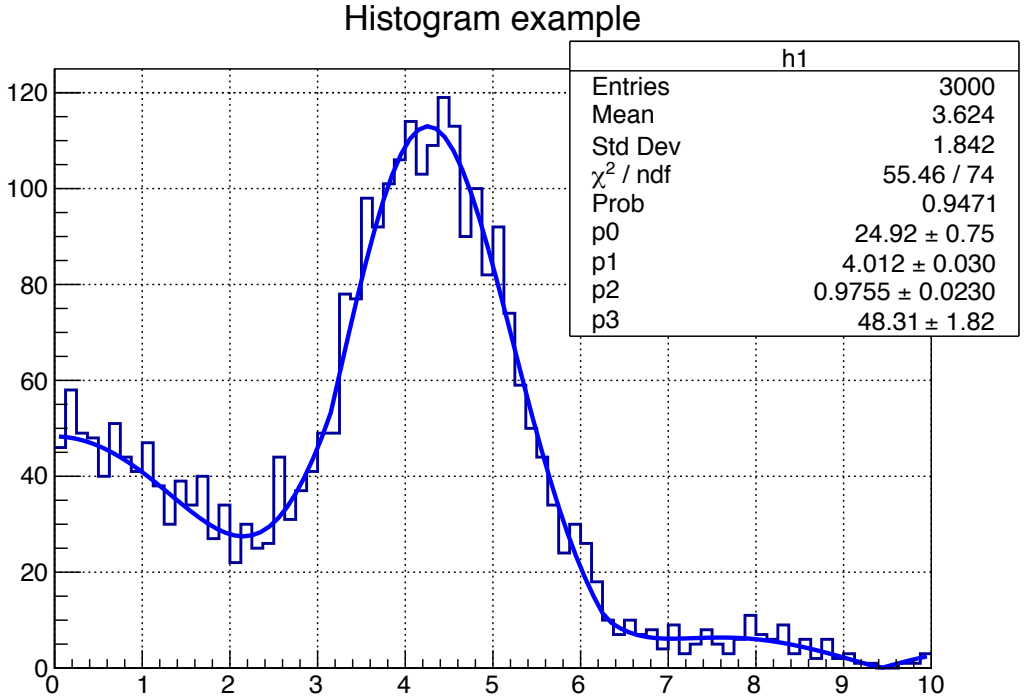
Histograms

Analysis result: often a histogram



Fitting

Analysis result: often a **FIT** of a histogram

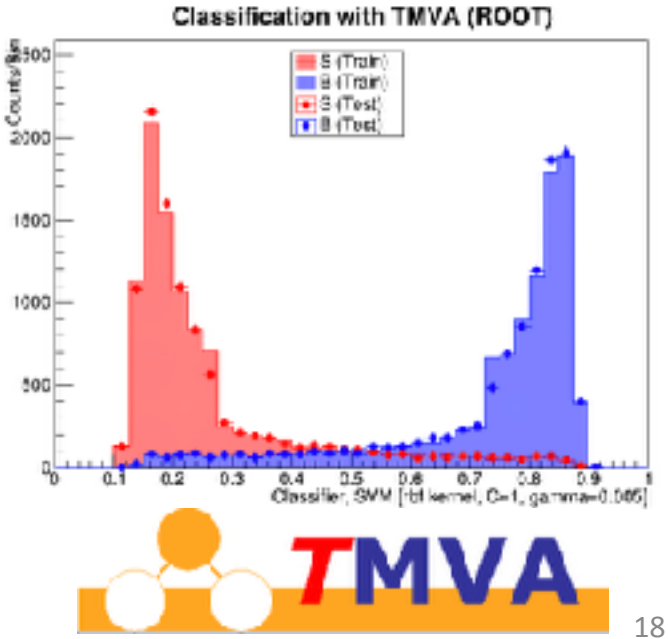


Fit Panel GUI

Machine Learning: TMVA

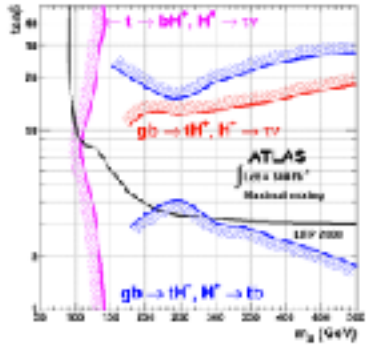
TMVA (Toolkit for Multi-Variate data Analysis in ROOT)

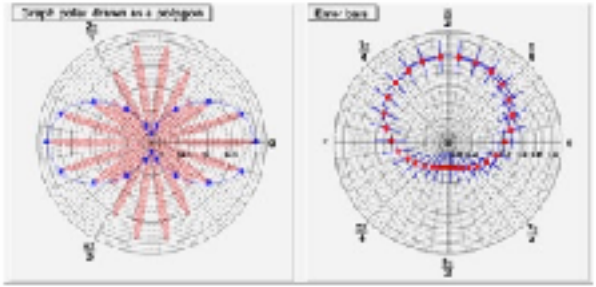
- provides several built-in ML methods including
  - Boosted decision trees
  - Deep Neural Network
  - Support Vector Machine
- and interfaces to external tools
  - scikit-learn, R, Keras (Theano)

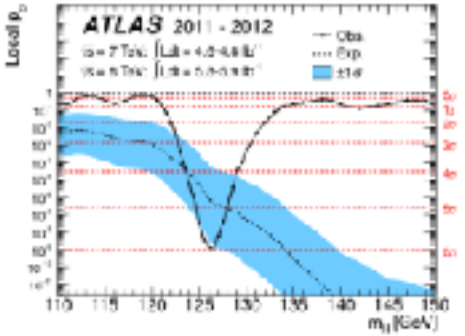


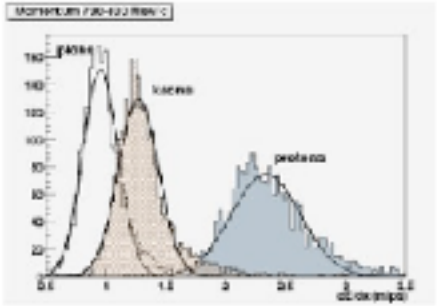
Graphics In ROOT

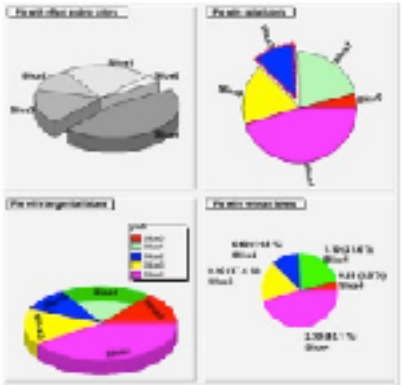
Many formats for data analysis, and not only, plots

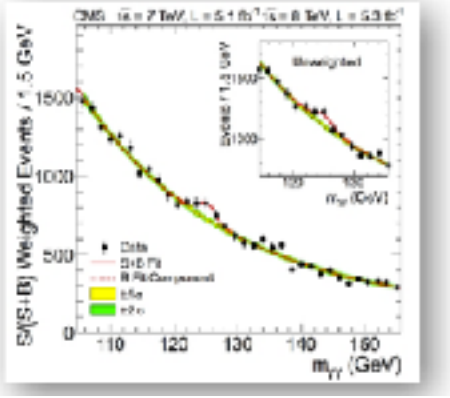










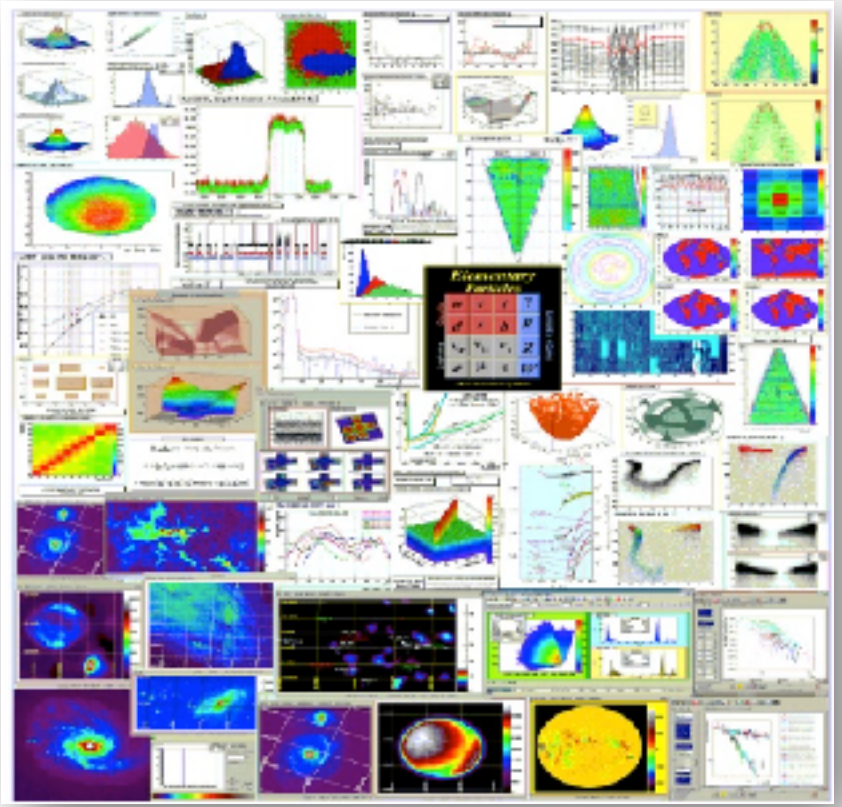


2D Graphics

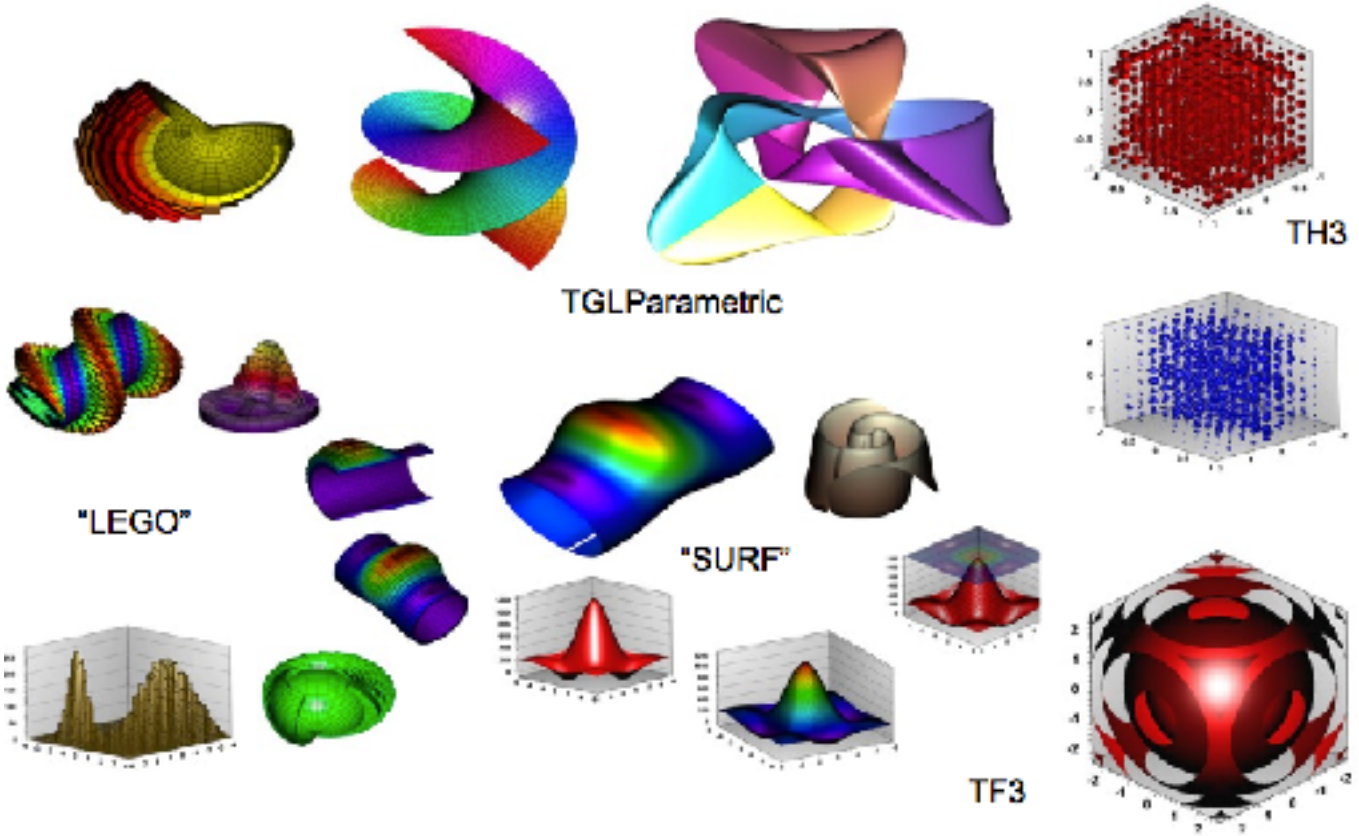
New functionalities added at every new release

Always requests for new style of plots

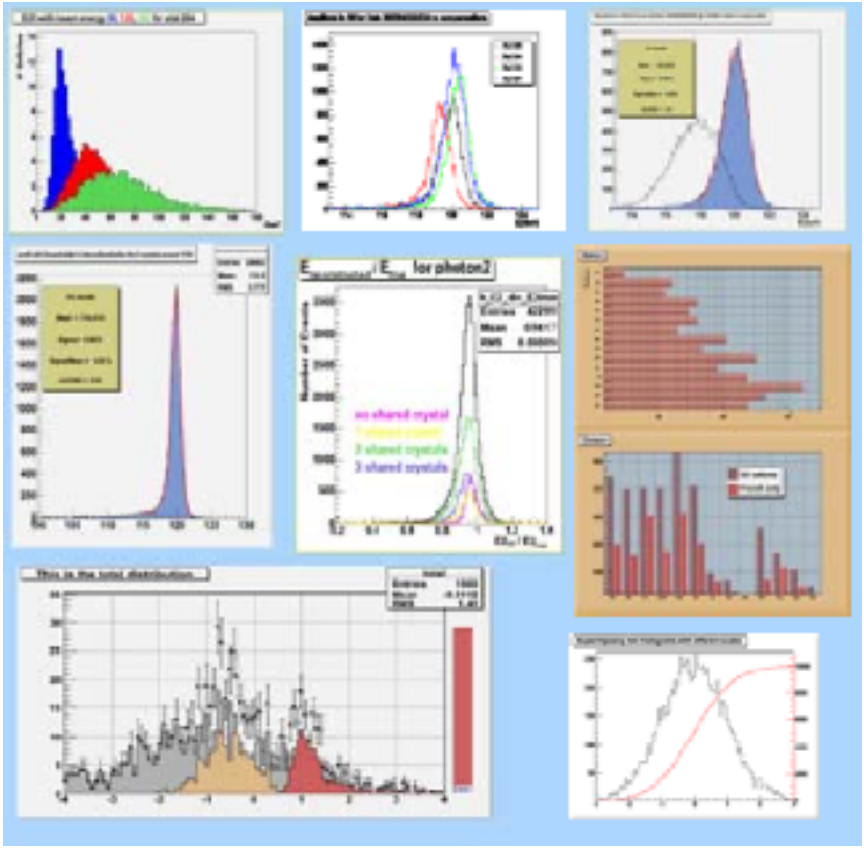
Can save graphics in many formats: *ps, pdf, svg, jpeg, LaTex, png, c, root …*



3D Graphics

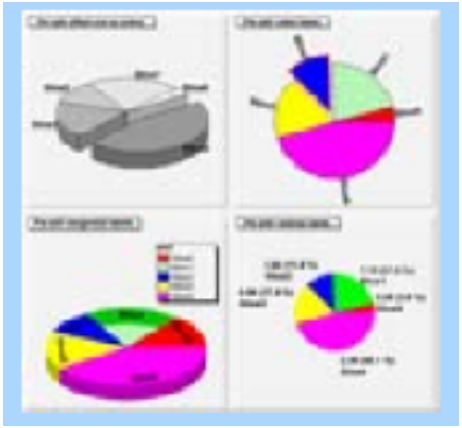


2 Variables Techniques: Histograms



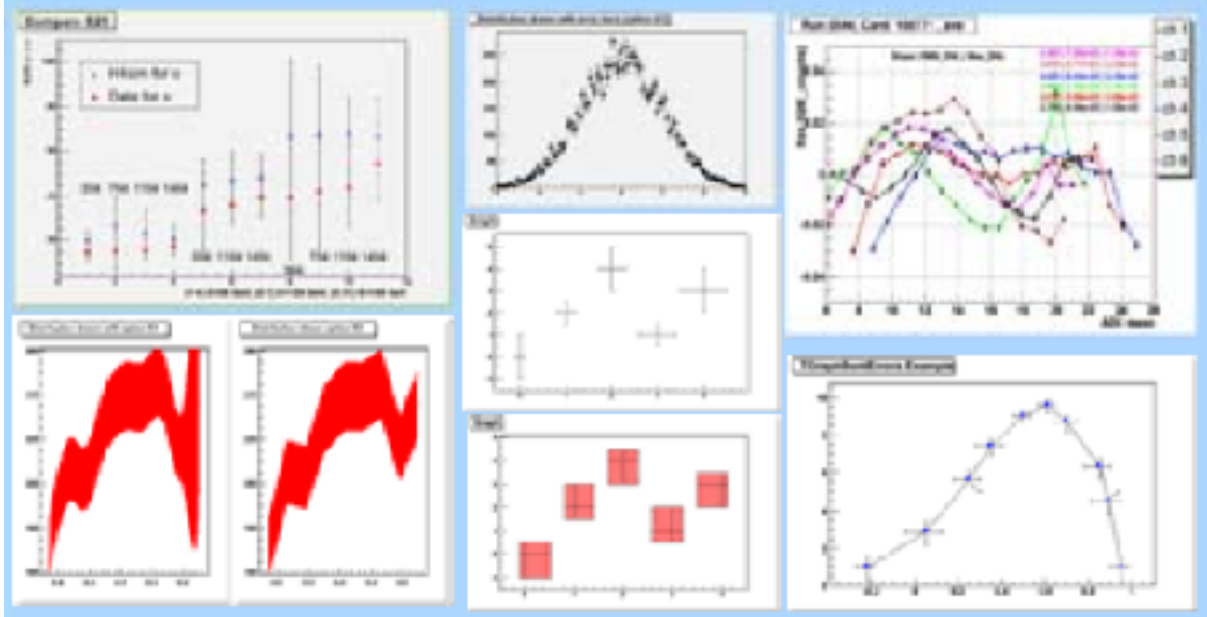
Bar charts and lines are a common way to represent 1D histograms.

Pie charts can be also used to visualize 1D histograms.



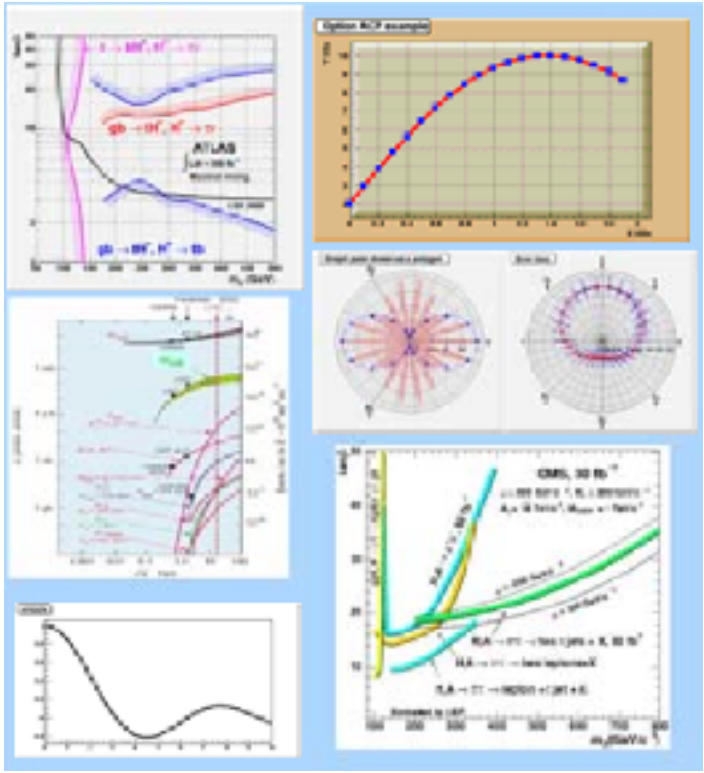
2 Variables Techniques: Errors

Visualization of Variables with Errors (1D Histograms and Graphs)



Errors can be represented as bars, band, rectangles. They can be symmetric, asymmetric or bent.

2 Variables Techniques: Graphs



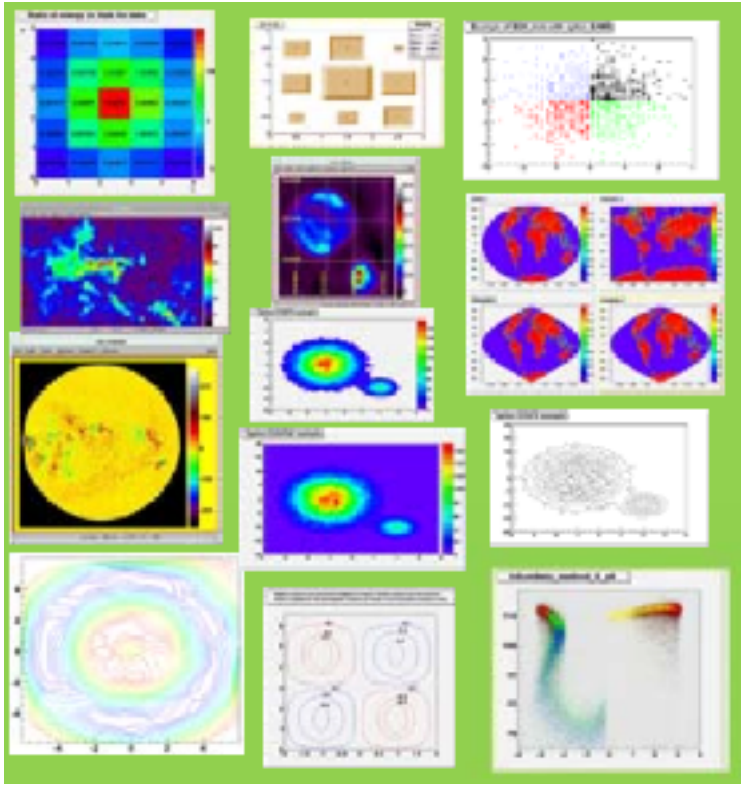
Graphs can be drawn as simple lines, like functions.
Can also visualize exclusion zones.
Can be plotted in polar coordinates.

3 Variables Techniques in 2D

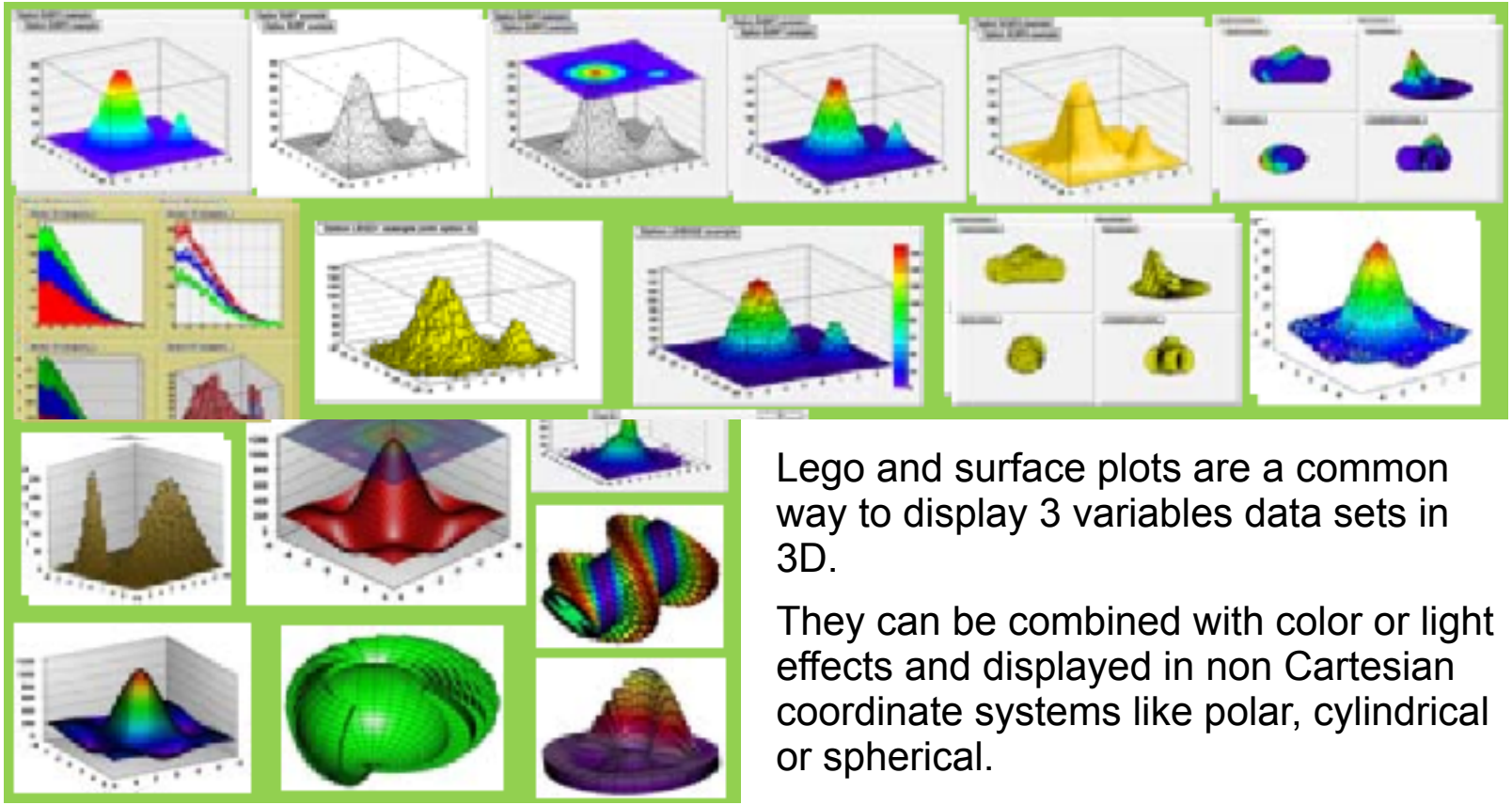
Visualization of 3 variables data sets (TH2, TGraph2D) in 2 dimensions.

Two variables are mapped on the X and Y axis and the 3rd one on some graphical attributes:

- the color or the size of a box;
- a density of points (scatter plot);
- writing the value of the bin content.
- Using contour plots. Some special projections (like Aitoff) are available to display such contours.



3 Variables Techniques in 3D



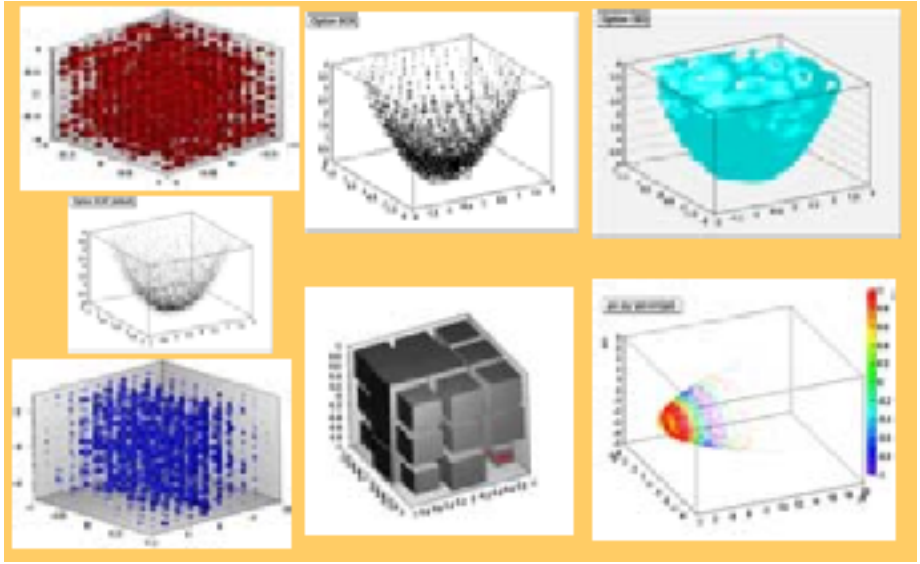
Lego and surface plots are a common way to display 3 variables data sets in 3D.

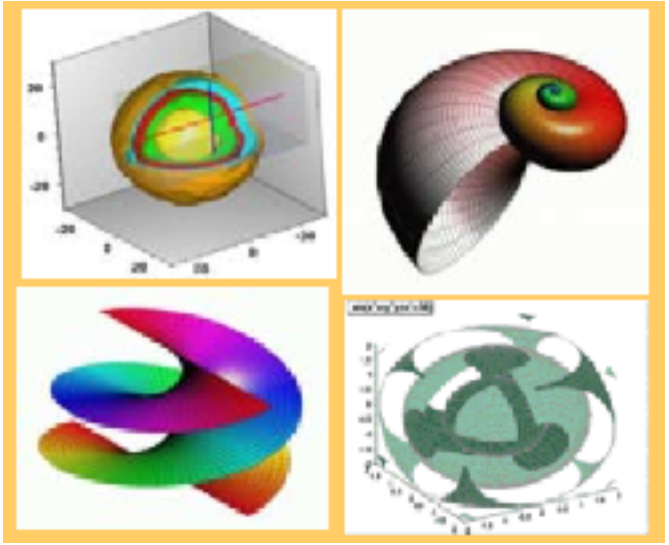
They can be combined with color or light effects and displayed in non Cartesian coordinate systems like polar, cylindrical or spherical.

4 Variables Techniques

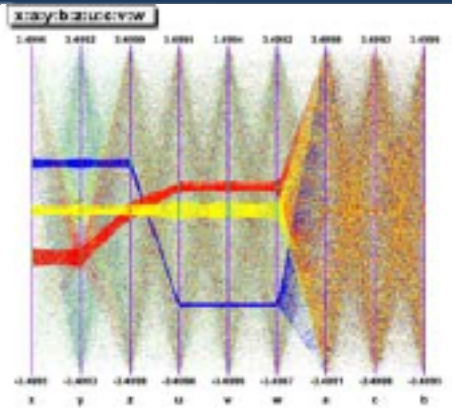
4 variables data set representations (e.g. TH3, TF3)

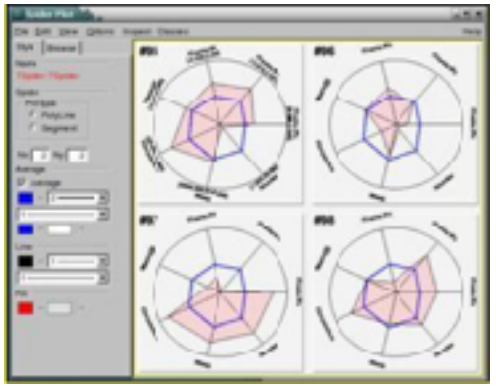
Rectangles become boxes or spheres, contour plots become iso-surfaces. The scatter plots (density plots) are drawn in boxes instead of rectangles. The 4th variable can also be mapped on colors. The use of OpenGL allows to enhance the plots' quality and the interactivity.





N Variables Techniques

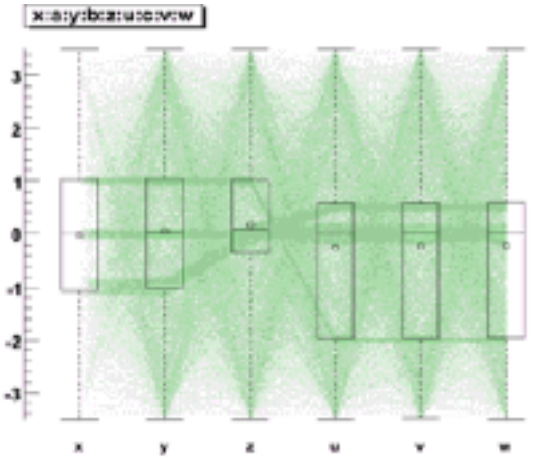




Specific visualization techniques are required for N > 4. ROOT provides:

- The parallel coordinates (top)
- the candle plots (right) which can be combined with the parallel coordinates.
- The spider plot (top right).

These three techniques, and in particular the parallel coordinates, require a high level of interactivity to be fully efficient.



Other ROOT Features

Geometry Toolkit

- Represent geometries as complex as LHC detectors

Event Display (EVE)

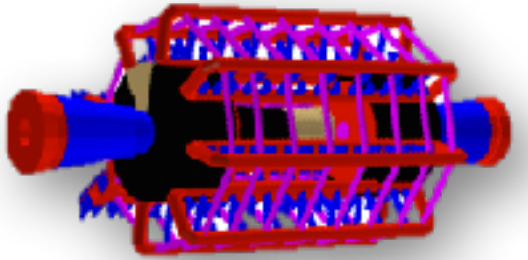
- Visualise particles collisions within detectors

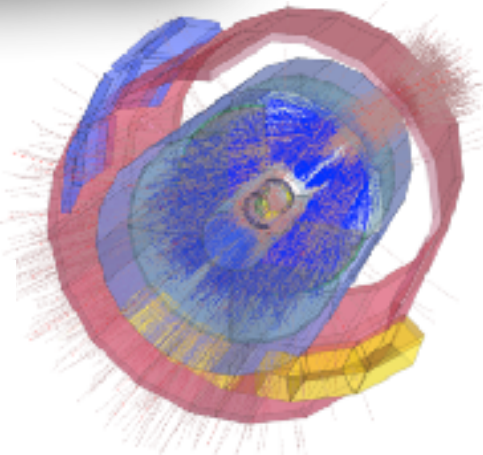
Multiproc

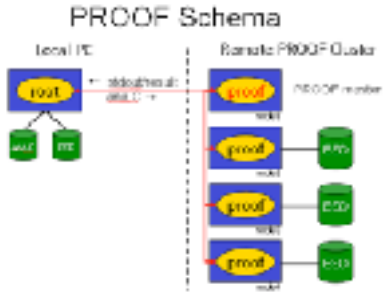
- Parallel shared-memory Map-Reduce

PROOF: Parallel ROOT Facility

- Multi-process approach to parallelism
- A system to run ROOT queries in parallel on a number of distributed computers
- Proof-lite: does not need a farm, uses all the on a desktop machine

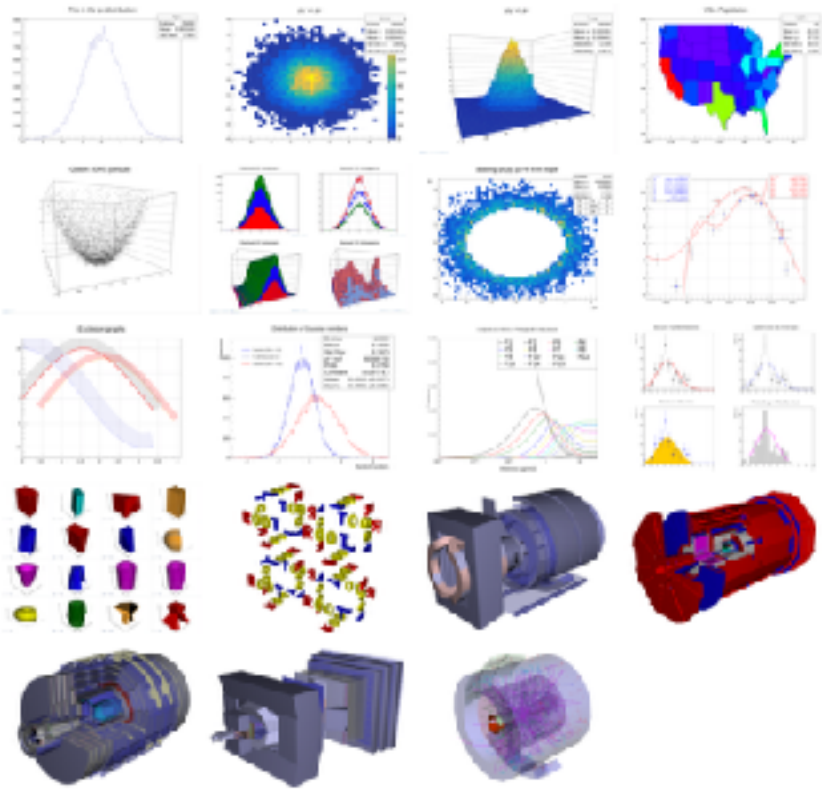






JavaScript Visualization

- ROOT supports also visualization using Javascript
  - Interactive ROOT graphics display in a Web browser
  - available in SWAN



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 CERNBox
- Simplify teaching of data processing and programming

http://swan.cern.ch

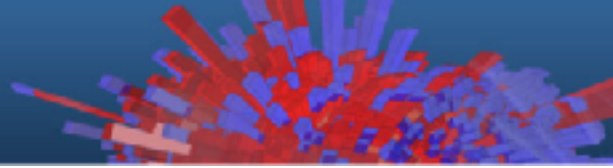
root.cern

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

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

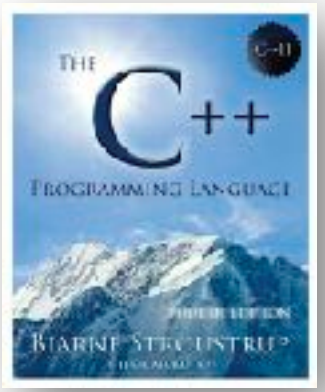
? *Why prompt and libraries instead of a GUI ?*

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

C++ From 10.000 Km

C++ From 10.000 Km

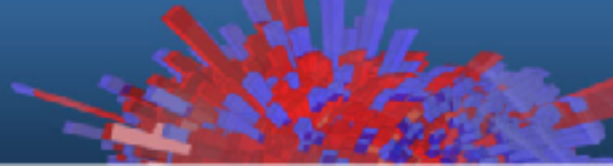
- Compiled, strongly typed language, allows to get the best performance from the hardware
- Allows object orientation
- Generic programming: Templates
- Explicit memory management (pointers)



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

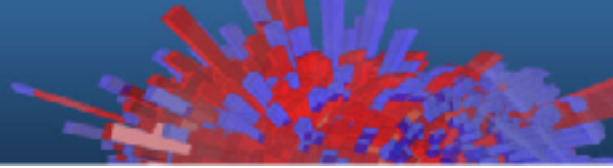
Example:

```
MyClass myClassInstance(“myName”);
myClassInstance.GetName();

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

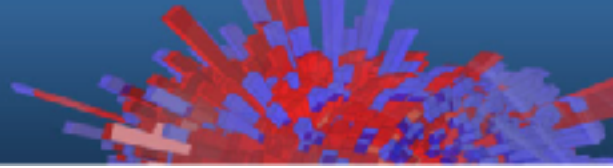
Note: the language is somehow approximate but certainly ok for this lecture

Python



- Python is a general-purpose programming language
  - easy to understand and it is fun to use it
  - very flexible (dynamically typed language)
  - very large community using it (one of th emost used languages and lots of libraries available)
    - in particular in the big data and machine learning community
  - not easy to maintain
  - slow (it is a dynamically typed language)

Python Interactive shell



```
% python
Python 2.7.10 (default, Jul 30 2016, 19:40:32)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You can type things directly into a running Python session

```
>>> 2+3*4
14
>>> name = "Andrew"
>>> name
'Andrew'
>>> print "Hello", name
Hello Andrew
>>>
```

ROOT Interface to Phython : 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

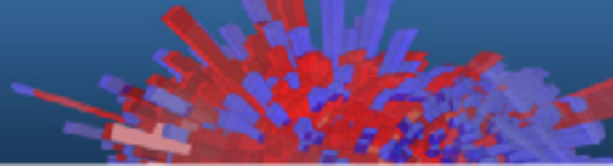
Entry point to use ROOT from within Python:

```
import ROOT
```

All classes you now know can be accessed like ROOT.TH1D, ROOT.TGraph, …

```
h1 = ROOT.TH1D("h1","h1",100,-3,3)
```

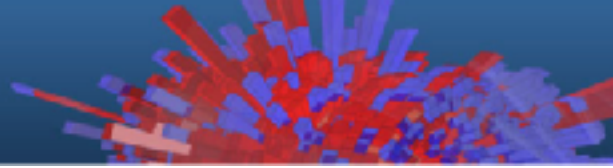
ROOT Basics



- Interactive ROOT Section
  - we will work in Jupyter notebooks in SWAN
- ROOT as a Calculator
- ROOT as Function Plotter
- Plotting Measurements
- Histograms

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

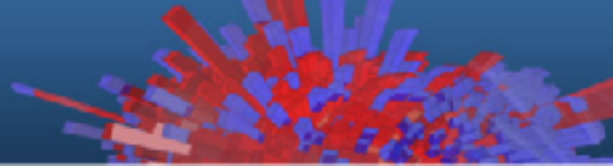
ROOT interactive prompt can be used as an advanced calculator !

```
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
```

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
```

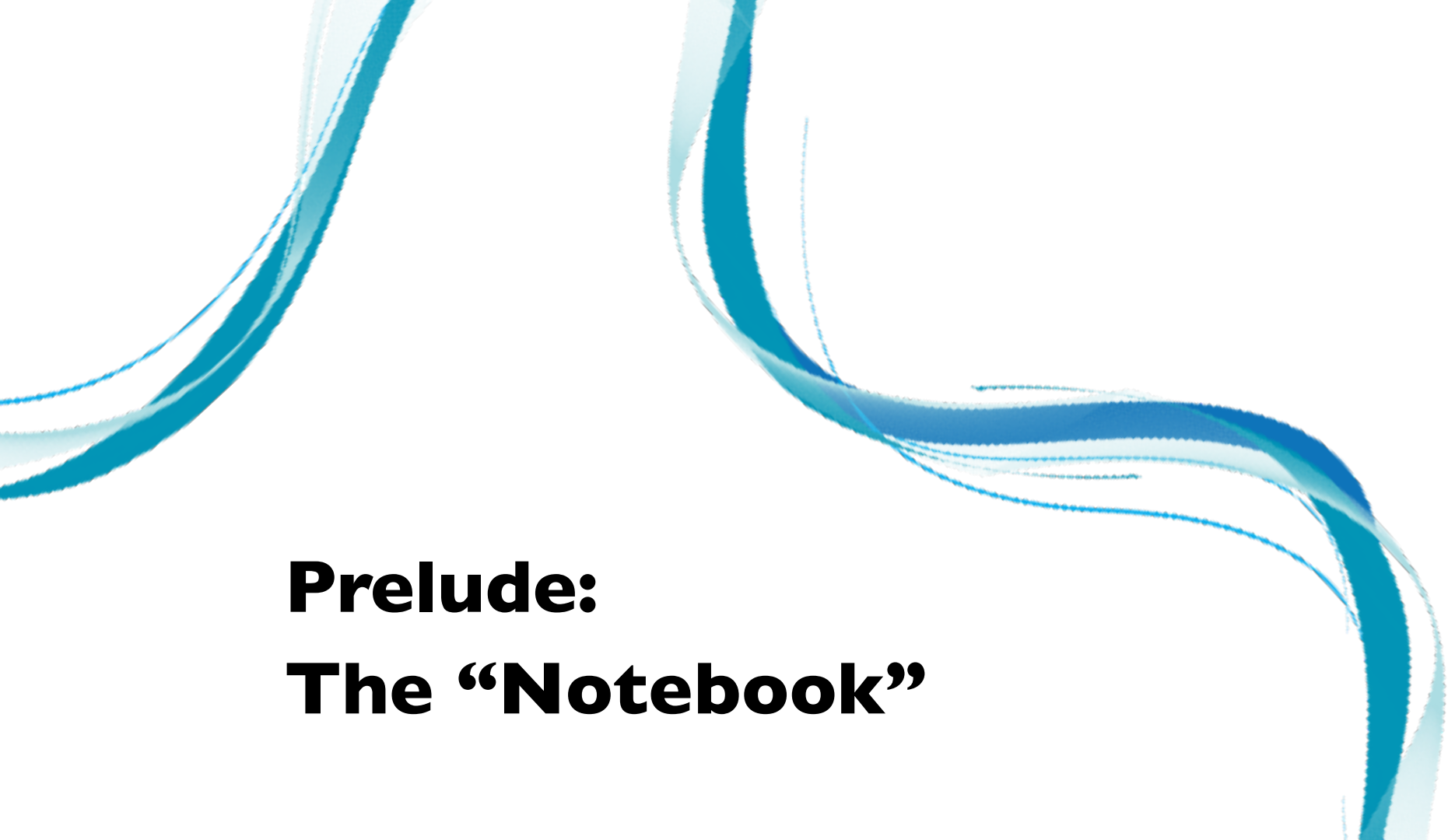
Interlude: Controlling ROOT

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



Prelude:
The “Notebook”

The Notebook

A web-based interactive computing interface and platform that combines code, equations, text and visualisations.

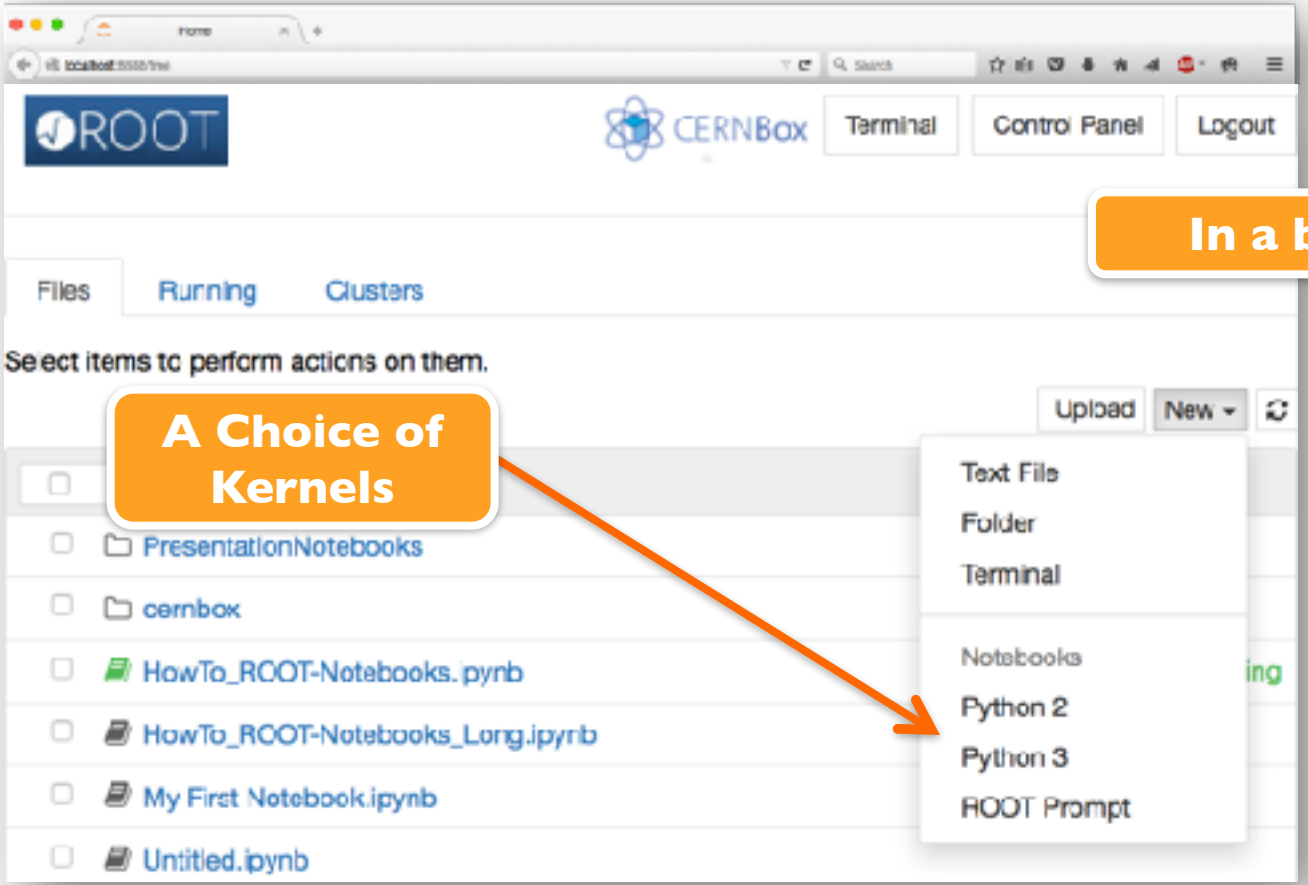
http://www.jupyter.org

Many supported languages: Python, Haskell, Julia, R … One generally speaks about a "kernel" for a specific language

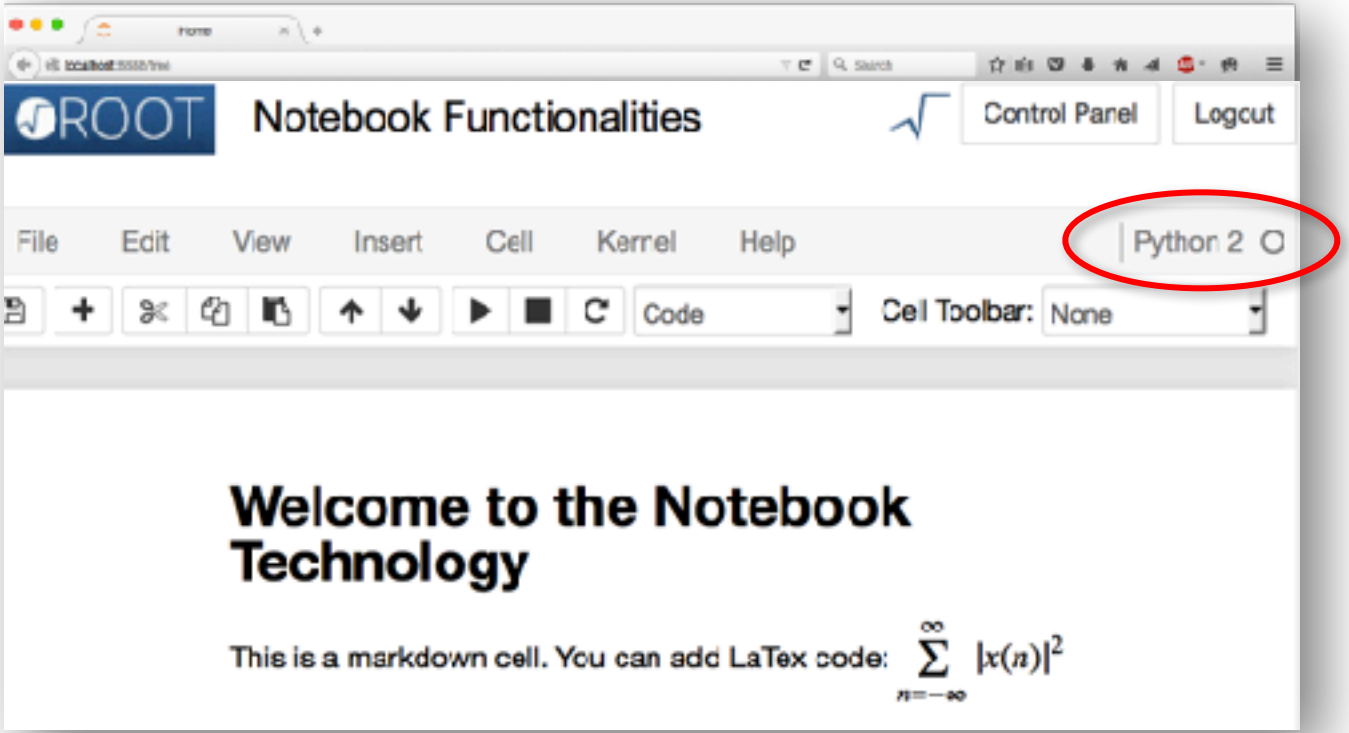
No excuses possible when it comes to describe all steps in an analysis!

In a nutshell: an "interactive shell opened within the browser"

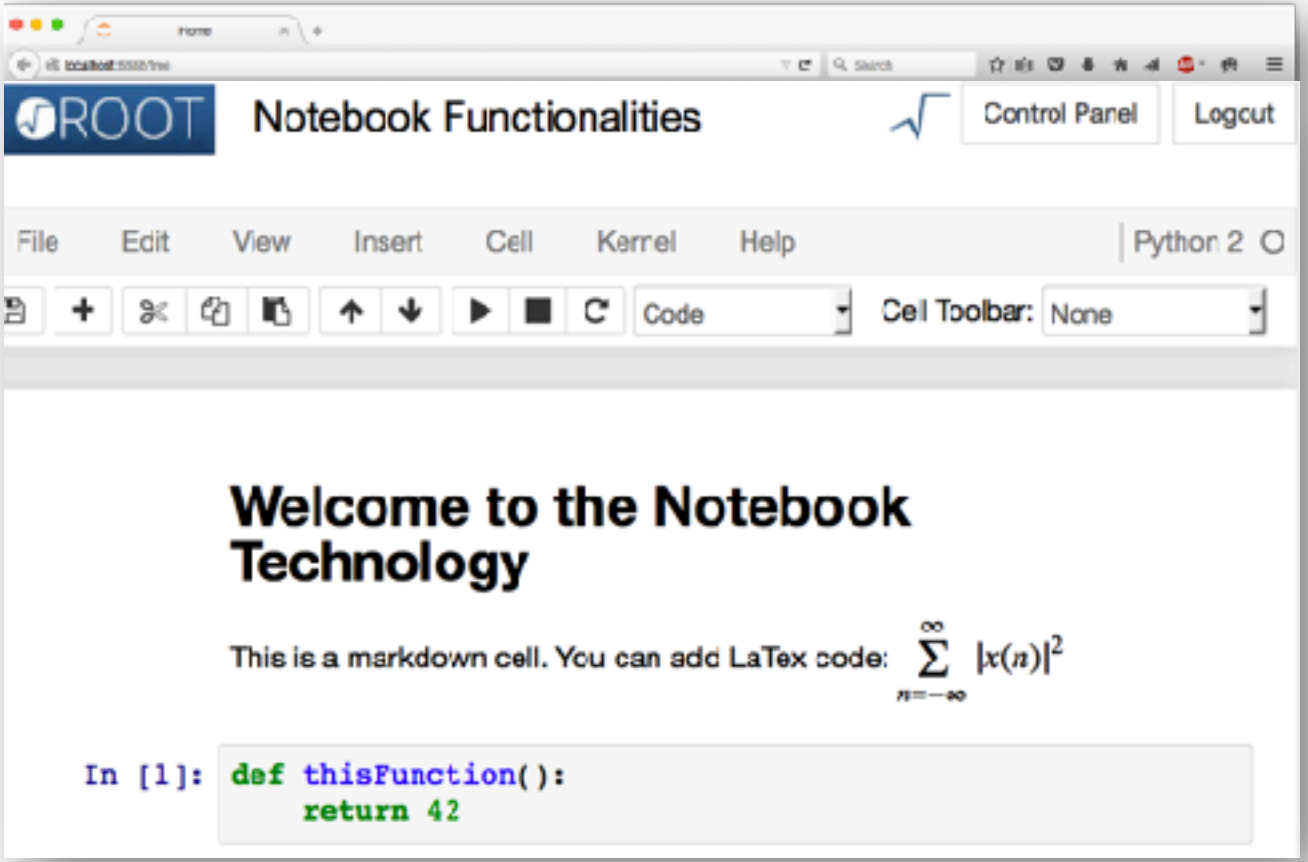
Also called:
"Jupyter Notebook" or "IPython Notebook"



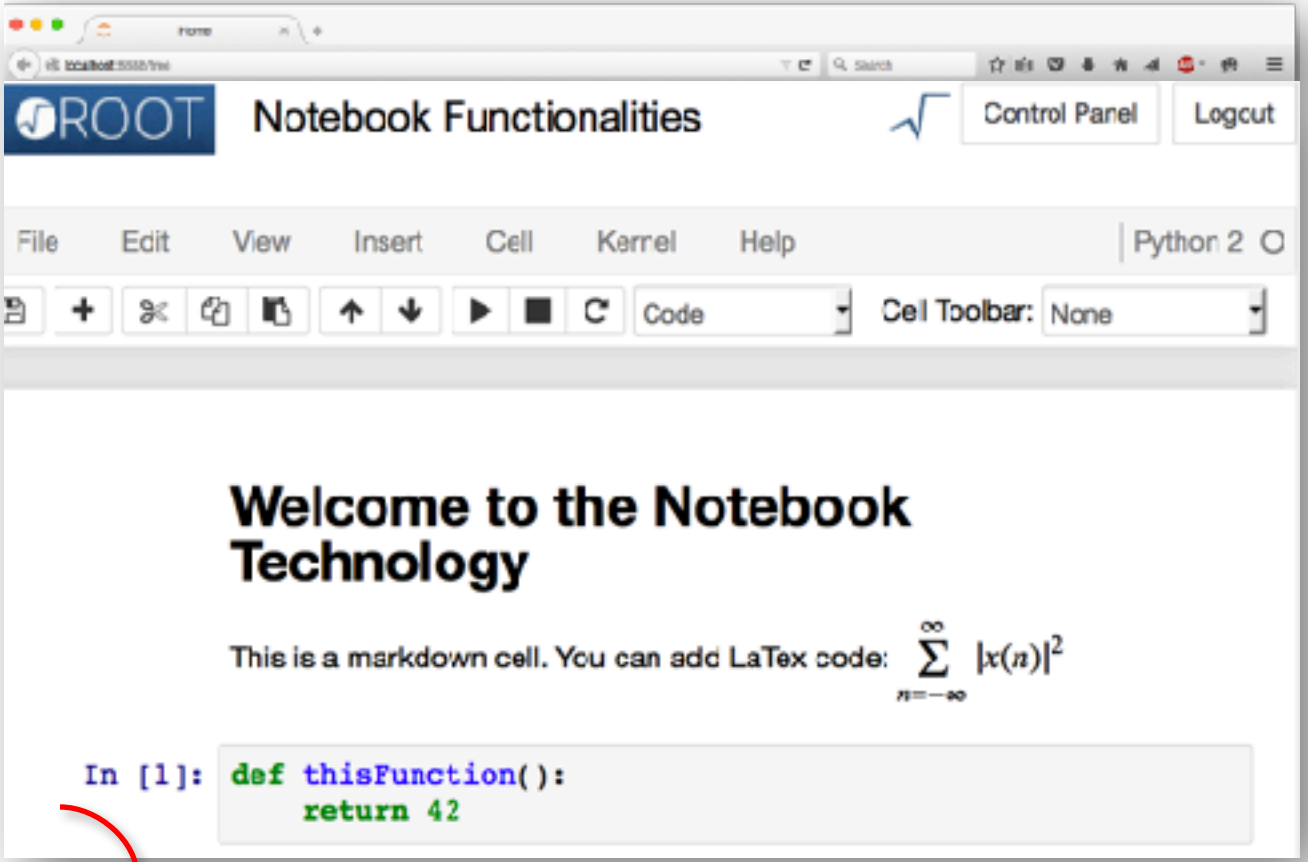
Kernels are processes that run interactive code in a particular programming language and return output to the user. Kernels also respond to tab completion and introspection requests.



Text and Formulas


ROOT
Notebook Functionalities
Control Panel
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Python 2
Code
Cell Toolbar:
None
Welcome to the Notebook Technology
This is a markdown cell. You can add LaTex code: $\sum_{n=-\infty}^{\infty} |x(n)|^2$
In [1]:
def thisFunction():
return 42

Code



This is a notebook in Python

Code

```
In [1]: def thisFunction():
            return 42

In [2]: thisFunction()

Out[2]: 42

In [3]: %%bash
        curl rootaasdemo.web.cern.ch/rootaasdemo/SaasFee.jpg \
        > SF.jpg
```

We can invoke commands in the shell…

Shell Commands

```
In [1]: def thisFunction():
            return 42

In [2]: thisFunction()

Out[2]: 42

In [3]: %%bash
        curl rootaasdemo.web.cern.ch/rootaasdemo/SaasFee.jpg \
        > SF.jpg
```

```
  % Total    % Received % Xferd  Average Speed   Time
  Time     Time  Current
                                 Dload  Upload   Total
  Spent    Left  Speed
100  128k  100  128k    0     0  2731k      0 --:--:--
--:--:-- --:--:-- 2787k
```

... And capture their output

Shell Commands

In [1]:
```
def thisFunction():
    return 42
```

In [2]:
```
thisFunction()
```

Out[2]:
```
42
```

In [3]:
```
%%bash
curl rootaasdemo.web.cern.ch/rootaasdemo/SaasFee.jpg \
> SF.jpg
```

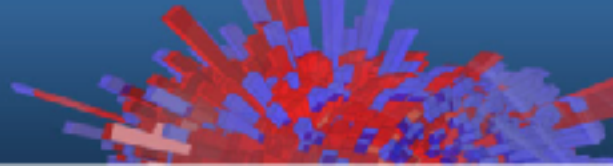
```
  % Total    % Received % Xferd  Average Speed   Time
  Time     Time  Current
                                 Dload  Upload   Total
  Spent    Left  Speed
100  128k  100  128k    0     0  2731k      0 --:--:--
--:--:-- --:--:-- 2787k
```

In [4]:
```
from IPython.display import Image
Image(filename="./SF.jpg",width=225)
```

Out[4]:

Images

Notebooks: Summary



- Working in a browser. No need to download any software
- Capable of executing code step by step
- Display of text and formulae (using Mardown)
- Possible to run operating system commands (e.g. shell commands)

ROOT Integration With Notebooks

Full integration of ROOT with Jupyter Notebooks

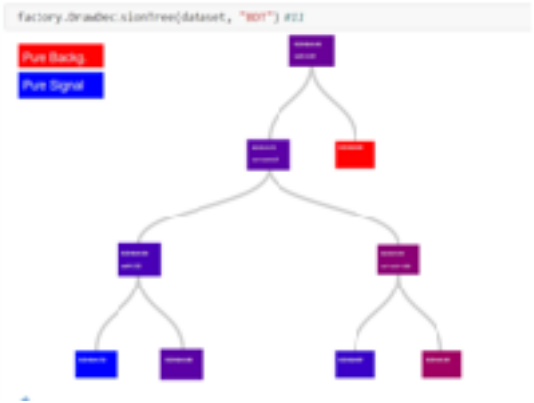
- "import ROOT": only action required to activate all goodies!
- A C++ Kernel 
- Inlining of plots as images or JavaScript interactive graphics
- *Magics* to JIT or compile C++ code for acceleration
  - Immediately usable in Python!
- Tab completion for name and methods of classes known to ROOT
- Capturing of output from C++ libraries

"Like before, but better"

ROOT – TMVA Integration

Full integration of ROOT Machine Learning package (TMVA) with Jupyter Notebooks

- Enhanced JSROOT plots
- Interactive training
- neural network and decision tree visualization

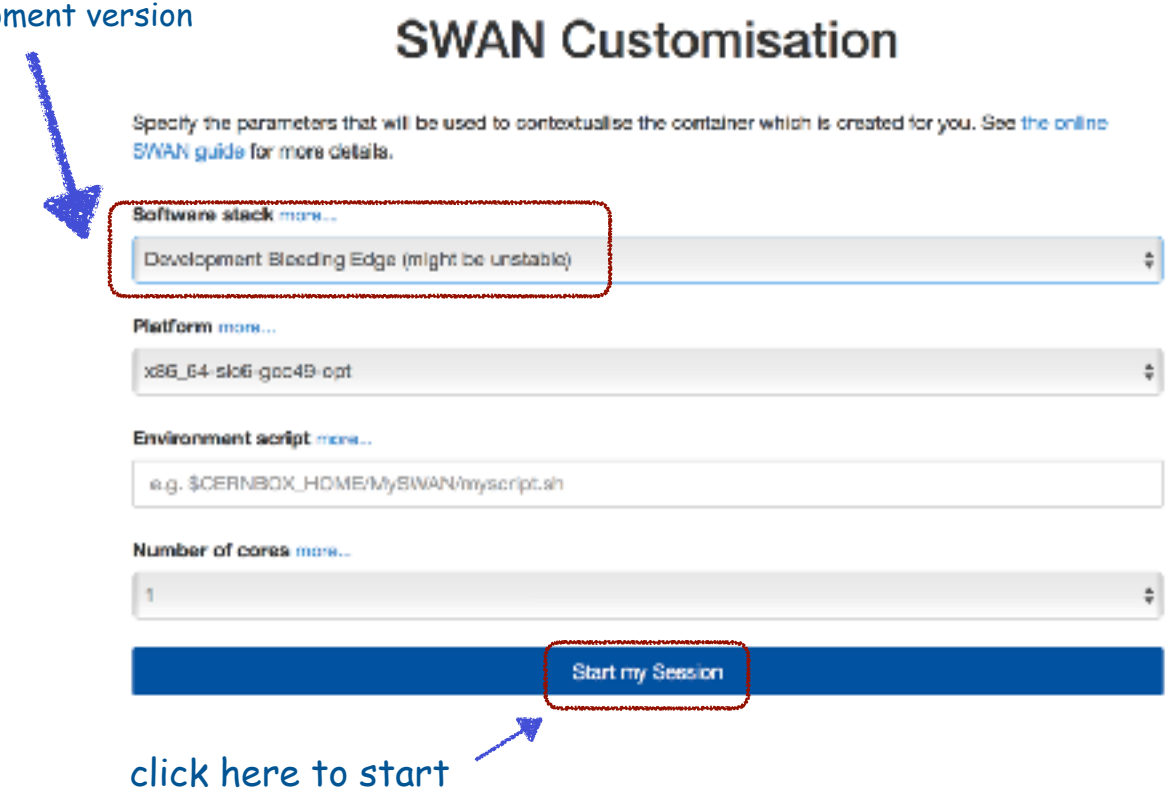


SWAN

- go to swan.cern.ch

Starting SWAN

make sure to have selected the development version



click here to start

Starting a Terminal in SWAN

After login cernbox home directory will be visible

Start a terminal window

Downloading the Notebooks

- After starting the terminal, download the notebooks.
- Download from my Web page using *wget*:

wget http://www.cern.ch/moneta/Sanofi_notebooks.tar.gz

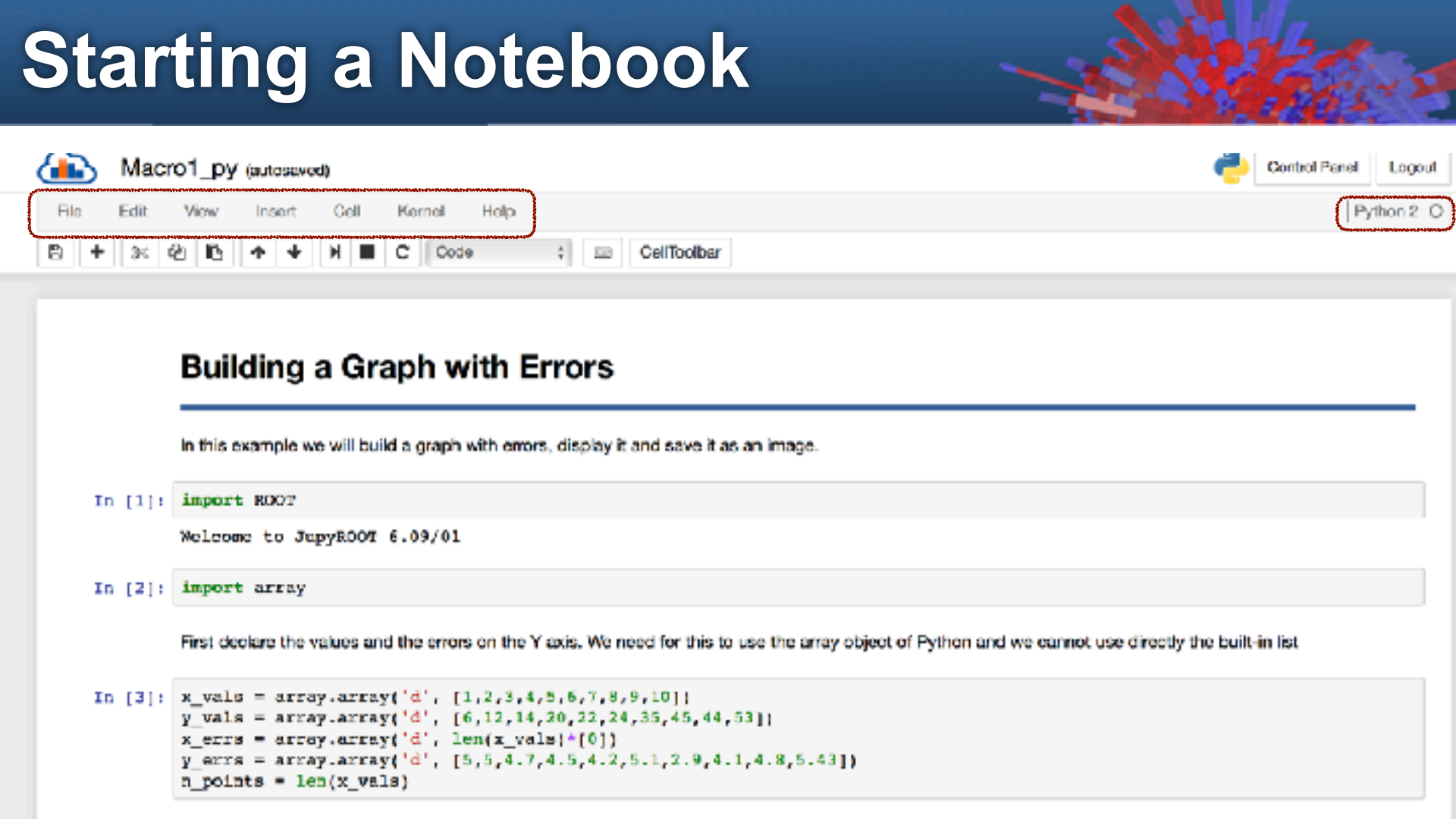
```
bash-4.1$ wget -nv http://www.cern.ch/moneta/Sanofi_notebooks.tar.gz
2017-03-04 22:09:15 URL:http://moneta.web.cern.ch/moneta/Sanofi_notebooks.tar.gz [166741/166741] -> "Sanofi_notebooks.tar.gz.1" [1]
bash-4.1$
```

tar -zxvf Sanofi_notebooks.tar.gz

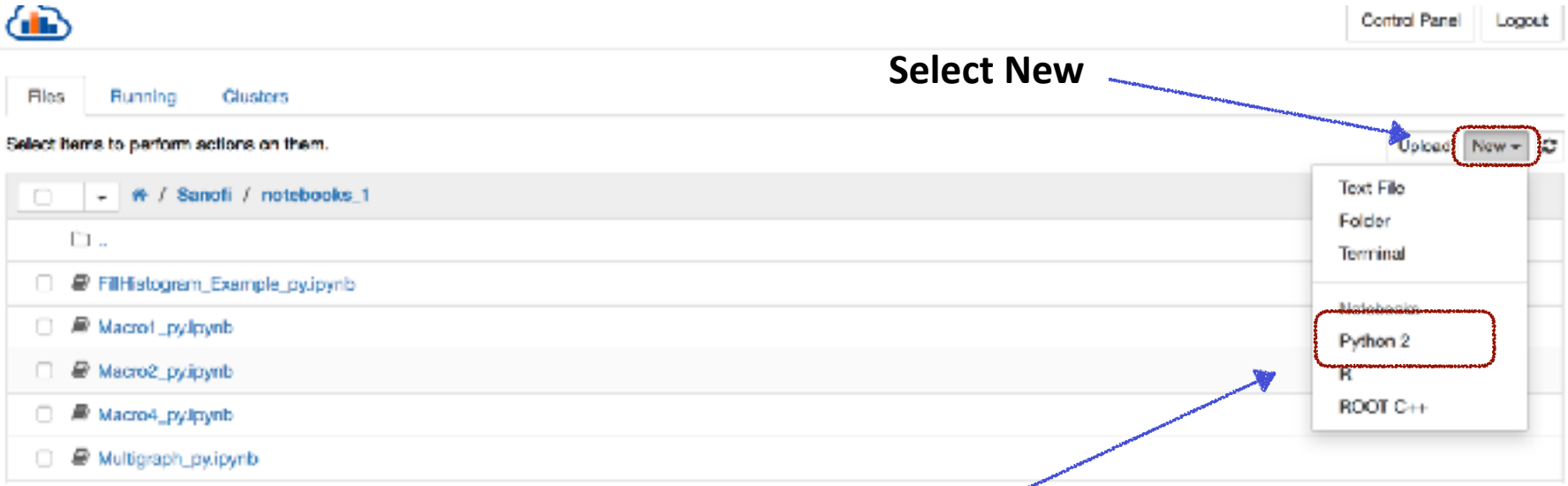
Starting a Notebook

- After downloaded and extracted the notebooks, go back to SWAN home page and select the desired notebook

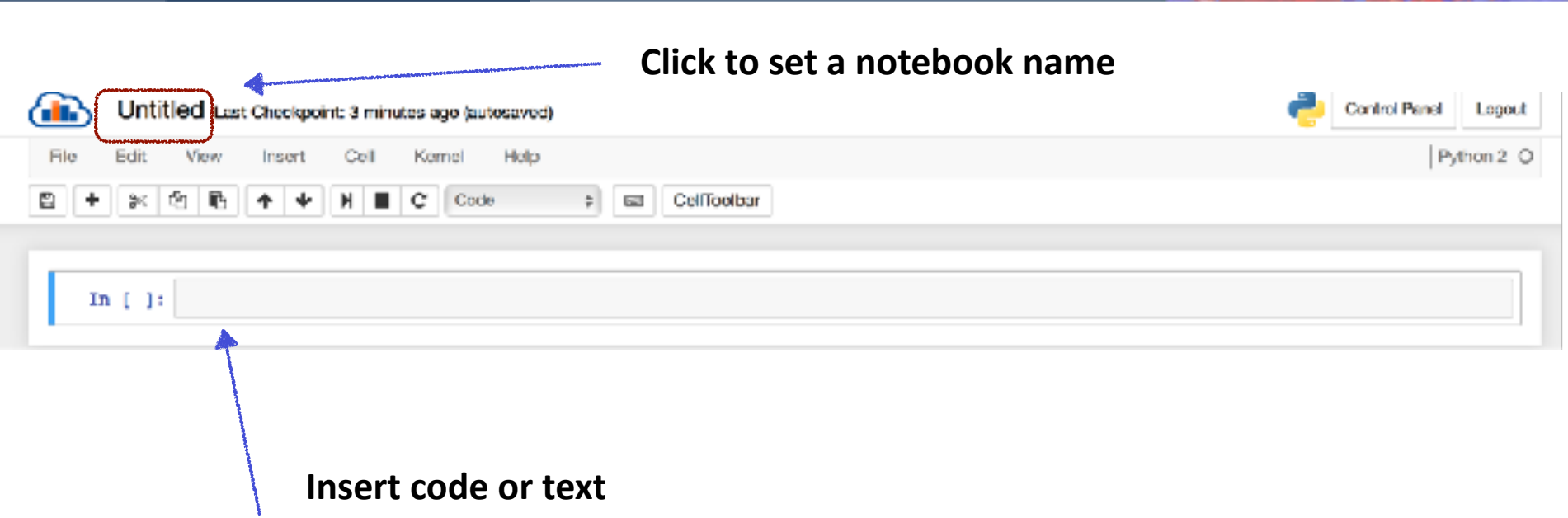
Starting a Notebook



Creating a new Notebook



Creating a new Notebook



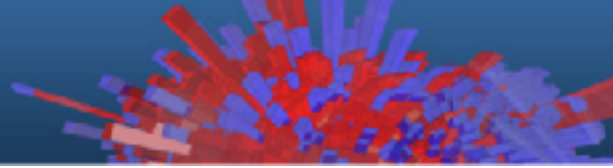
Exercise

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.

For one number

```
import math
import ROOT
x=1
math.exp(-x*x*.5) - ROOT.TMath.Gaus(x)
[…]
```

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.

```
import math
import ROOT
x=1
math.exp(-x*x*.5) - ROOT.TMath.Gaus(x)
```

Many possible ways of solving this! E.g:

```
v=[0,1,10,20]
for x in v:
     print x,math.exp(-x*x*.5) - ROOT.TMath.Gaus(x)
```

ROOT As a Function Plotter

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

```
import ROOT
f1 = ROOT.TF1(“f1”,”sin(x)/x",0.,10.) # name,formula, min, max
f1.Draw()
ROOT.gPad.Draw()     #  needed to Draw in Jupyter
```

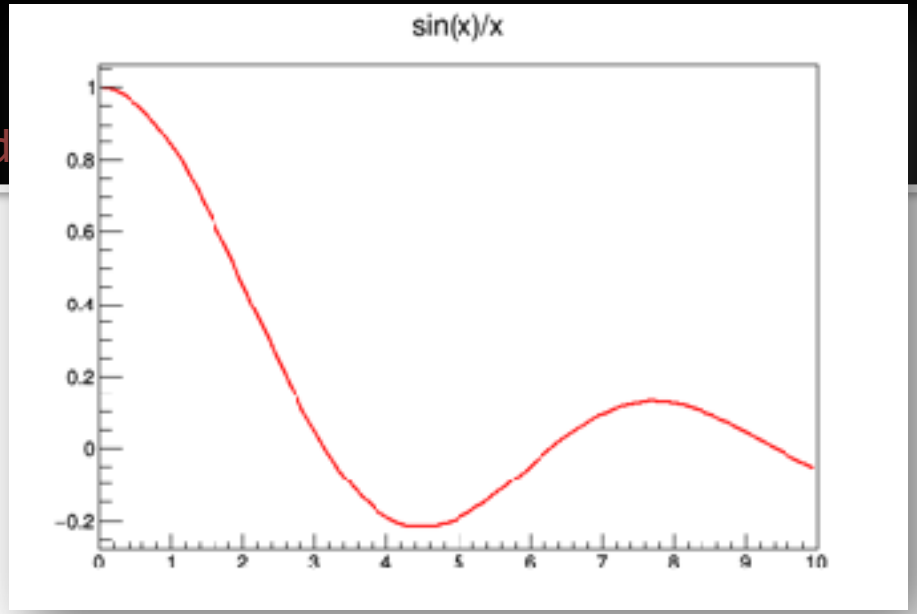
An extended version of this example is the definition of a function with parameters:

```
f2 = ROOT.TF1(“f2","[0]*sin([1]*x)/x",0.,10.)
f2.SetParameters(1,1)
f2.Draw()
ROOT.gPad.Draw()     #  needed to Draw in Jupyter
```

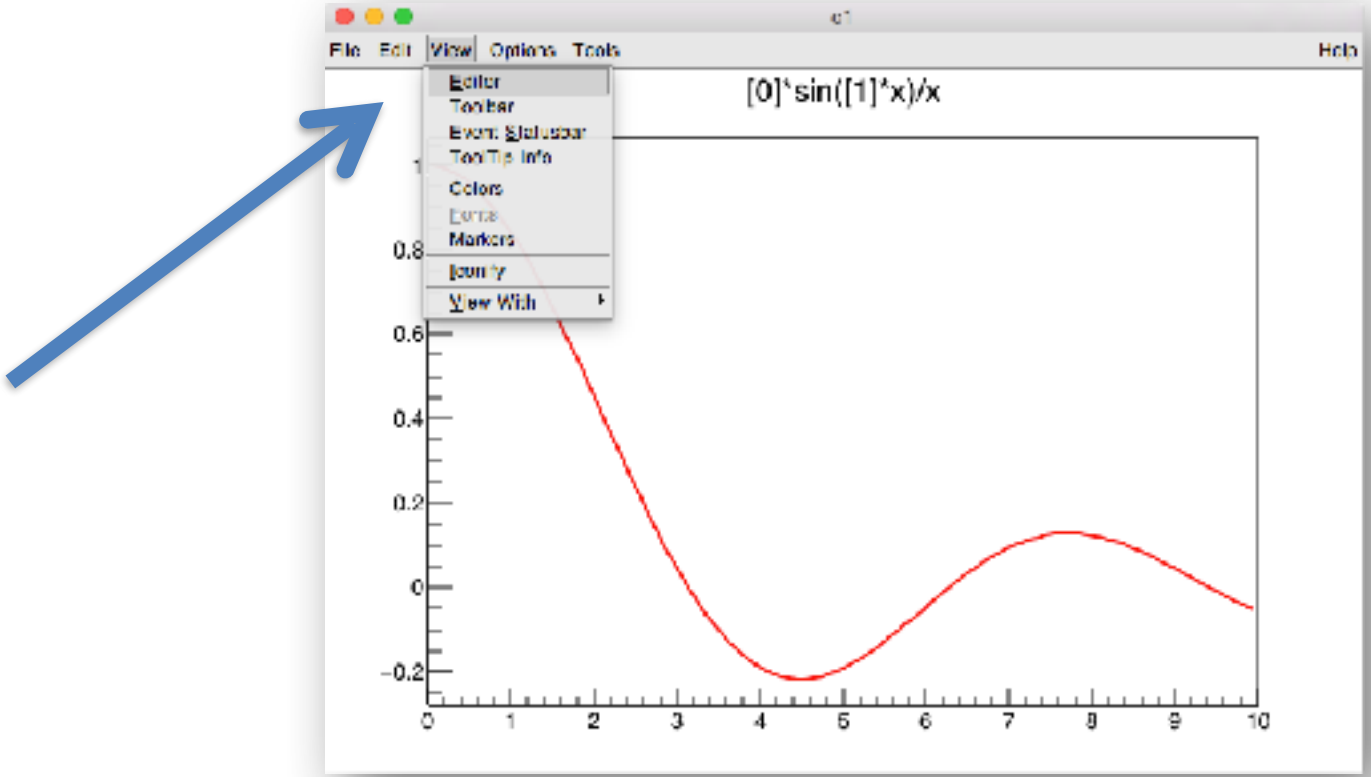
ROOT As a Function Plotter

Example: the definition of a function with parameters:

```
import ROOT
f2 = ROOT.TF1("f2","[0]*sin([1]*x)/x",0.,10.)
f2.SetParameters(1,1)
f2.Draw()
ROOT.gPad.Draw()    #  need
```

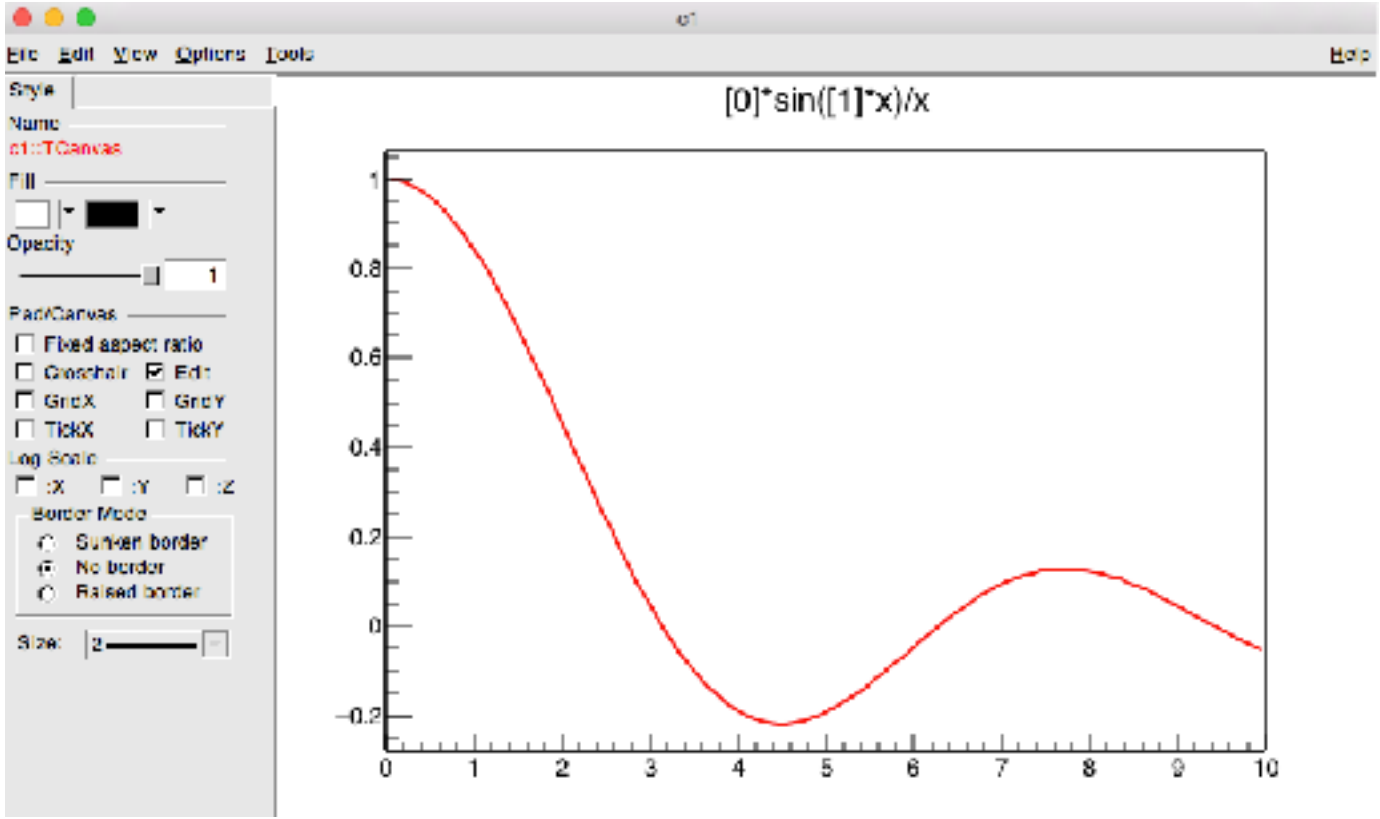


Exercise: Interaction With The Plot

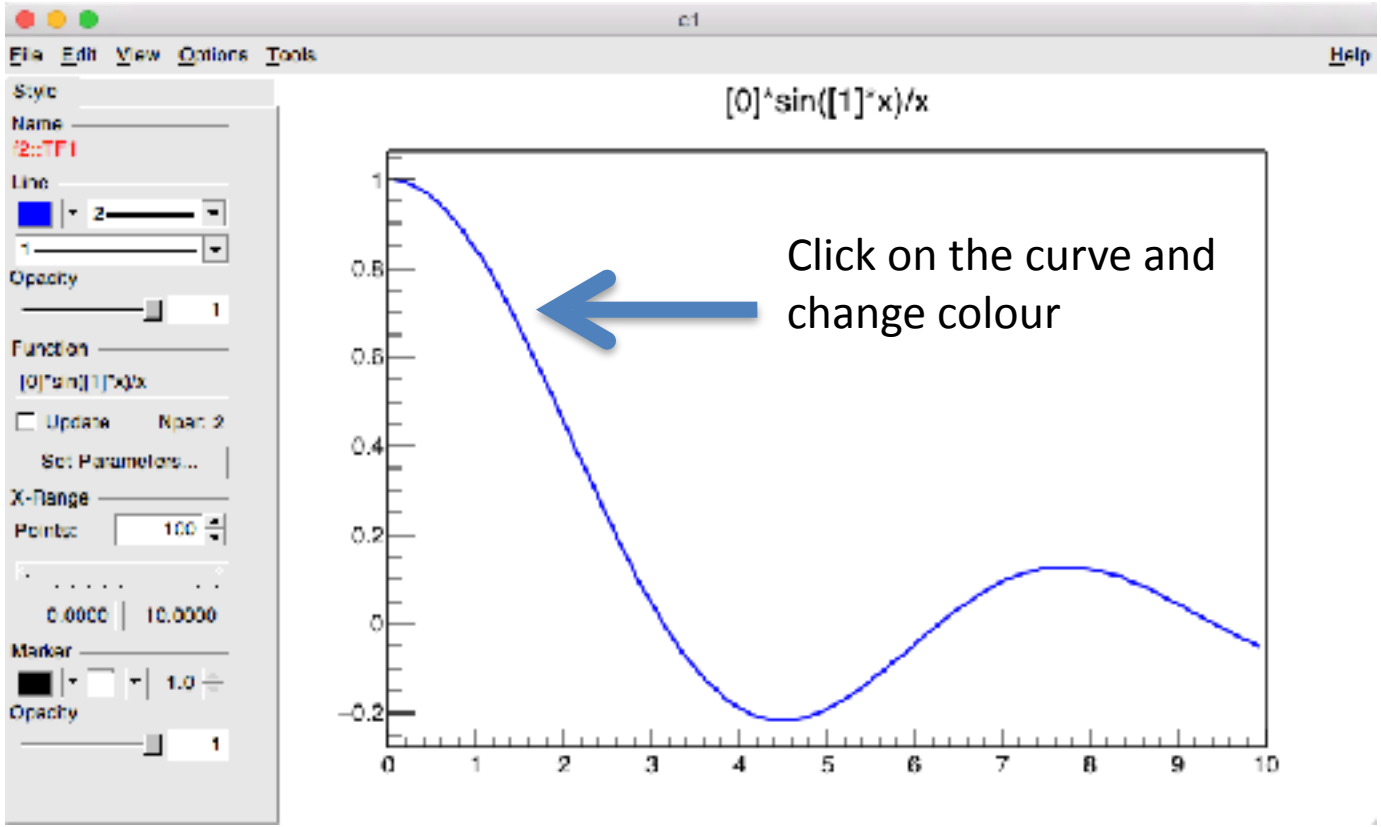


This available when working from ROOT prompt or from Python env

Exercise: Interaction With The Plot



Exercise: Interaction With The Plot

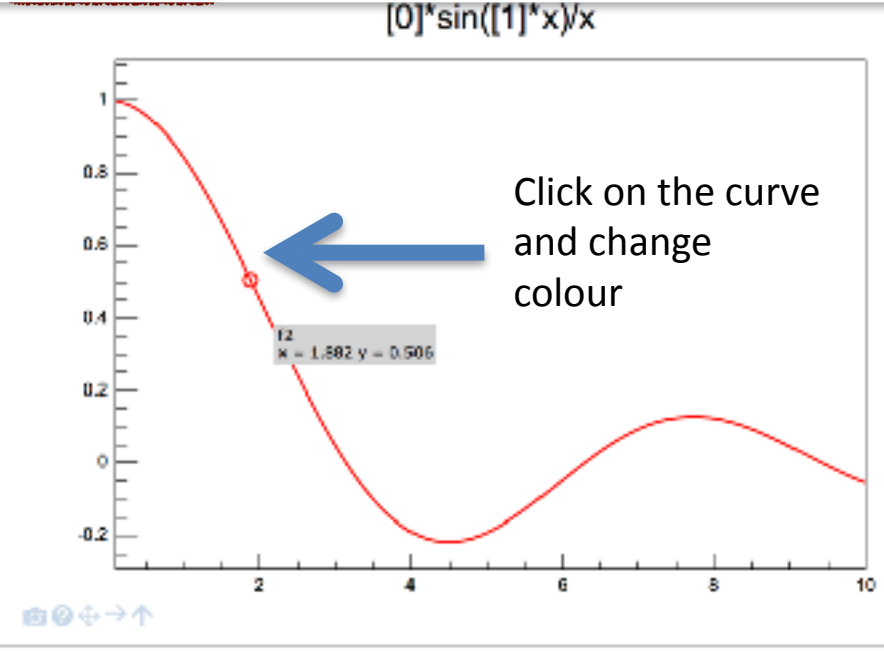


Exercise: Interaction With The Plot

From a notebook one can enable Javascript plotting (JSROOT)

```
jsroot on
ROOT.gPad.Draw()     #  needed to Draw in Jupyter
```

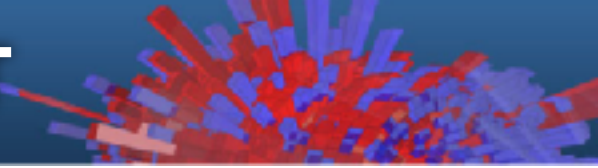
Notebook requires to draw the ROOT canvas explicitly



Click on the curve and change colour

Javascript menu

Mathematical Functions in ROOT



- **`TMath`**: a namespace providing the following functionality:
  - Numerical constants.
  - Trigonometric and elementary mathematical functions.
  - Statistic Functions (e.g. Gauss)
  - Special Mathematical Functions (e.g. Bessel functions)
  - For more details, see the reference documentation of TMath.

```
ROOT.TMath.Gaus( x, mean, sigma)
```

- Functions provided in **`ROOT::Math`** namespace
  - special functions (many implemented using the Gnu Scientific Library)
  - all major statistical functions
  - For more details, see the reference documentation of ROOT::Math functions

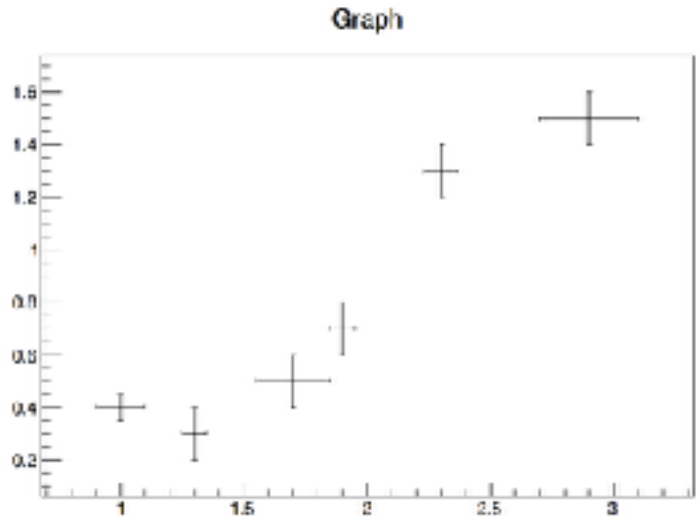
```
ROOT.Math.chisquared_pdf(x,ndf)
```

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:

```
gr  = ROOT.TGraphErrors ("ExampleData.txt")
gr.Draw("AP")
ROOT.gPad.Draw()
```

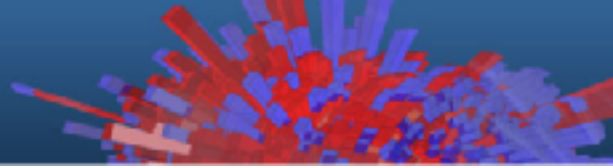
Tells ROOT to draw the **A**xis and the **P**oints



run it in SWAN!

ExampleGraph_py

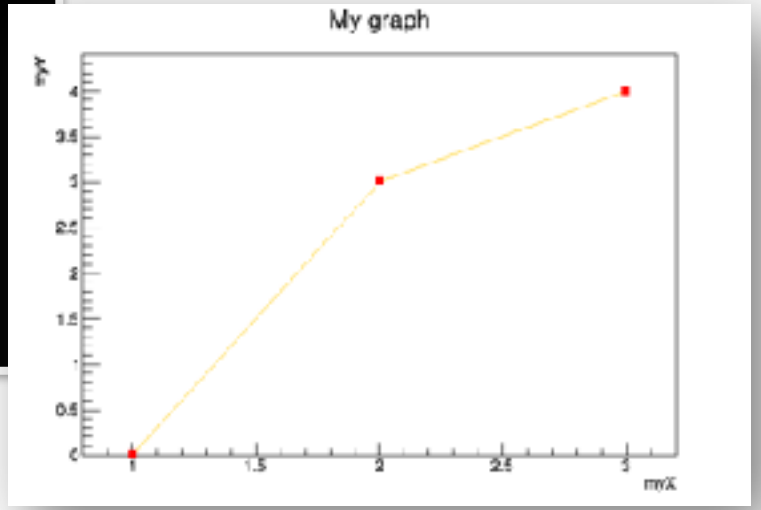
Exercise: TGraph



- 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

```
import ROOT
g = ROOT.TGraph()
g.SetTitle("My graph;myX;myY")
g.SetPoint(0,1,0)
g.SetPoint(1,2,3)
g.SetPoint(2,3,4)
g.SetMarkerStyle(ROOT.kFullSquare)
g.SetMarkerColor(ROOT.kRed)
g.SetLineColor(ROOT.kOrange)
g.Draw("APL")
ROOT.gPad.Draw()
```



Histograms

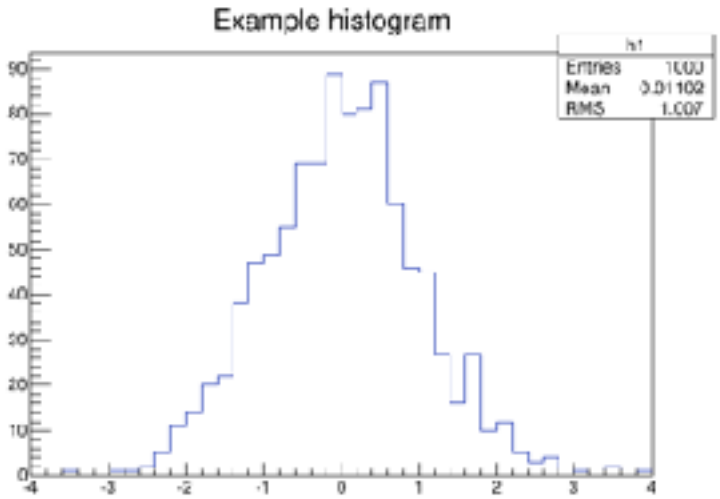
- What is a histogram?

–from Wikipedia:

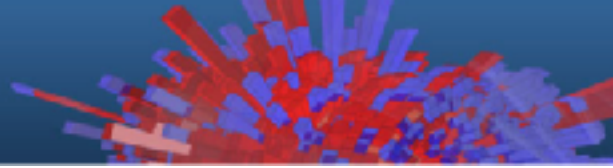
– a **histogram** is a graphical representation showing a visual impression of the distribution of data. It is an estimate of the probability distribution of a continuous variable and was first introduced by Karl Pearson. [1]

– A histogram consists of tabular frequencies, shown as adjacent rectangles, erected over discrete intervals (bins), with an area equal to the frequency of the observations in the interval.

– The height of a rectangle is also equal to the frequency density of the interval, i.e., the frequency divided by the width of the interval.
The total area of the histogram is equal to the number of data entries.



Histograms in ROOT



- Used to display and estimate the distribution of a variable (e.g. observed energy spectrum)
  - –visualize number of events in a certain range, called *bin*
  - –*bins* typically have equal widths, but not always
    - •ROOT supports histograms with equal and variable bins
- Histograms can be used for further analysis
  - –e.g to understand the underlying parent distribution
- ROOT provides various types of histograms depending on:
  - –contained data type (double, float, integer, char)
  - –choice of uniform or variable bins
  - –dimension (1,2 or 3)

How to use ROOT Histograms

- Example of creating a one-dim. histogram:

```
h1 = ROOT.TH1D("h1","Example histogram",40,-4.,4.)
```

- Filling histogram:

```
h1.Fill(x)
```

Fill the histogram with one observation "

```
for i in range(0,1000):
    x = ROOT.gRandom.Gaus(0,1)
    h1.Fill(x)
```

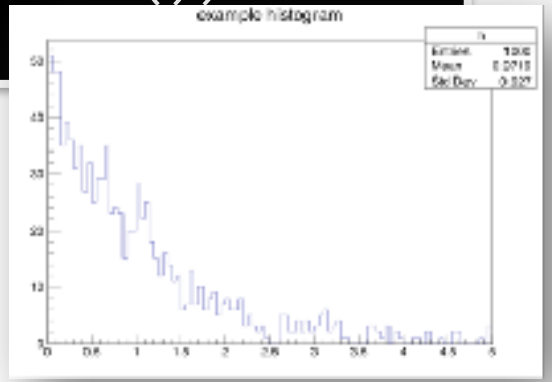
Fill the histogram with
1000 gaussian distributed
random numbers

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 …

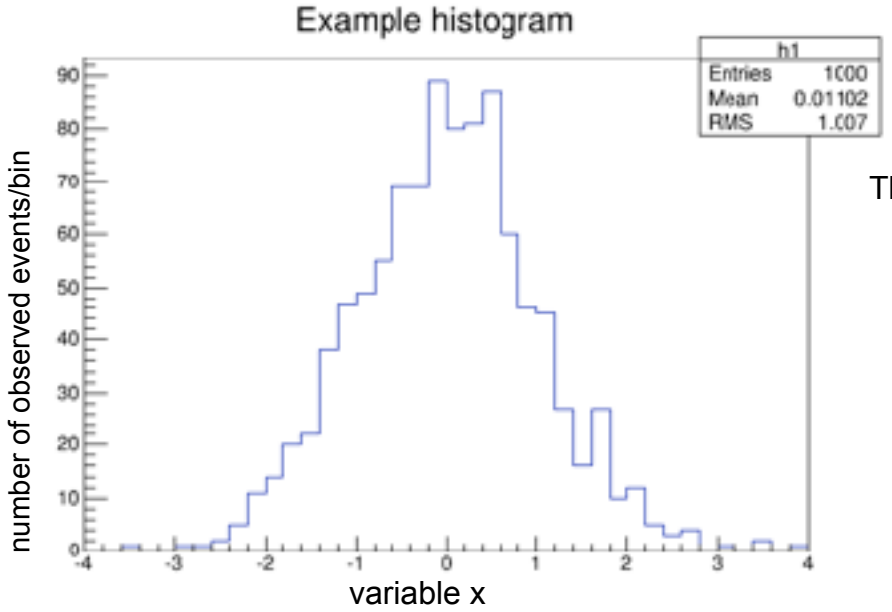
```
efunc = ROOT.TF1(“efunc","exp([0]+[1]*x)",0.,5.)
efunc.SetParameters(1,-1)
h = ROOT.TH1F(“h","hist",100,0.,5.)
for i in range(0,1000): h.Fill(efunc.GetRandom())
h.Draw()
```



Displaying ROOT histograms

- Drawing histograms in a ROOT canvas:

```
h1.Draw();
```

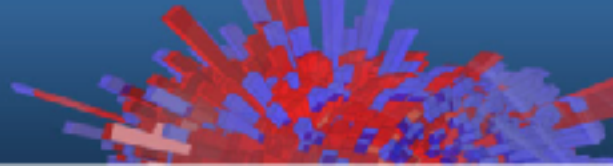


histogram name

The histogram statistics:

- Total number of entries
- Sample Mean
- Sample Standard Deviation (RMS)

Histogram Statistics



- To extract statistics information from an histogram:

```
h1.GetEntries()
1000
h1.Integral()
1000
h1.GetMean()
0.026801411040800516
h1.GetMeanError()
0.03281977394226118
h1.GetStdDev()
1.0378523794938883
h1.GetSkewness()
-0.0010784297300549448
h1.GetKurtosis()
0.004886407306863294
```

Histogram Drawing Options

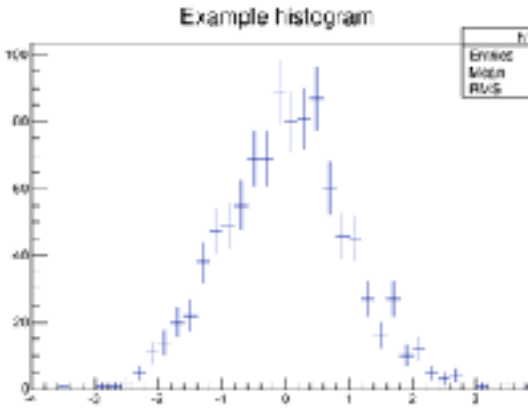
- Various drawing options are available:
  - draw error bars on every bin

```
h1.Draw("E")
```

  - "SAME"

```
h1.Draw("SAME")
```

    - draw the histogram on the canvas without replacing what is already there.
      Plot one histogram on top of another



- The default drawing option is "HIST" for histograms without errors (unweighted histograms) and "E" for weighted histograms
- For displaying the histogram in log scale in one axis, e.g. the y axis:

```
ROOT.gPad.SetLogy()
```

1D Histogram Drawing Options

- Almost all drawing options for 1D histograms:

"AXIS"	Draw only axis
"AH"	Draw histogram, but not the axis labels and tick marks
"]["	When this option is selected the first and last vertical lines of the histogram are not drawn.
"B"	Bar chart option
"C"	Draw a smooth Curve through the histogram bins
"E"	Draw error bars
"E0"	Draw error bars including bins with 0 contents
"E1"	Draw error bars with perpendicular lines at the edges
"E2"	Draw error bars with rectangles
"E3"	Draw a fill area through the end points of the vertical error bars
"E4"	Draw a smoothed filled area through the end points of the error bars
"L"	Draw a line through the bin contents
"P"	Draw current marker at each bin except empty bins
"P0"	Draw current marker at each bin including empty bins
"*H"	Draw histogram with a * at each bin
"LF2"	Draw histogram like with option "L" but with a fill area. Note that "L" draws also a fill area if the hist fillcolor is set, but the fill area corresponds to the histogram contour.

Histogram Drawing Options

- Histogram drawing is handled internally by the THistPainter class.
- The documentation for all the drawing options can be found in the class reference page
  - http://root.cern.ch/root/html/THistPainter.html


THistPainter Class Reference

The histogram painter class.

Implements all histograms' drawing's options.

- Introduction
- Histograms' painting options
  - Options supported for 1D and 2D histograms
  - Options supported for 1D histograms
  - Options supported for 2D histograms
  - Options supported for 3D histograms
  - Options supported for histograms' stacks (THStack)
- Setting the Style

Exercise

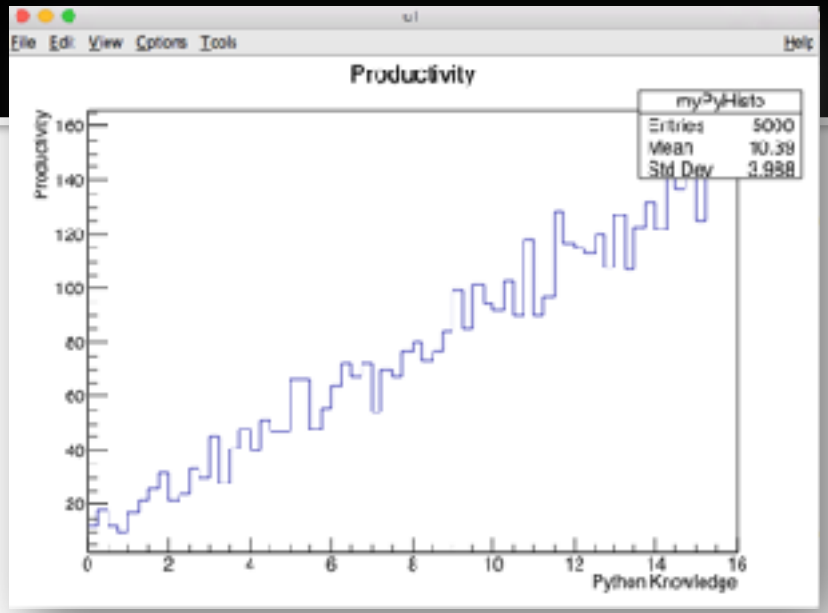
- Create a histogram with 64 bins and an x axis ranging from 0 to 16
- Fill it with random numbers distributed according to a linear function (“pol1”)
- Change its line width with a thicker one
- Draw it!

run it in SWAN!

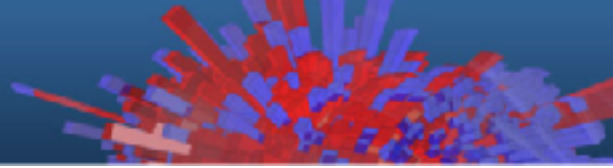
https://swan.cern.ch

Exercise Solution

```
import ROOT
h = ROOT.TH1F(“myPyHisto","Productivity;Python Knowledge;Productivity",64,0,16)
h.FillRandom(“pol1”)
h.SetLineStyle(2)
h.Draw()
```

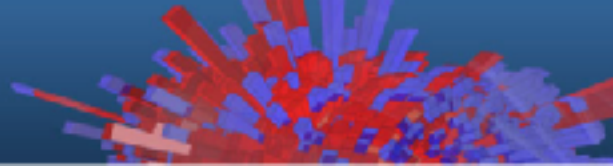


ROOT Macros



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

General Remarks

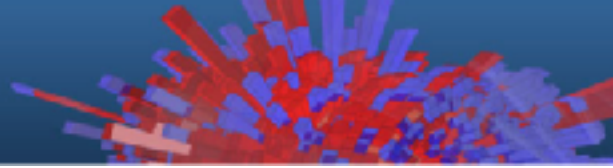


We have seen how to interactively type lines at the prompt.
The next step is to write “ROOT Macros” – lightweight programs
The general structure for a macro stored in file *MacroName.C* is:

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
```

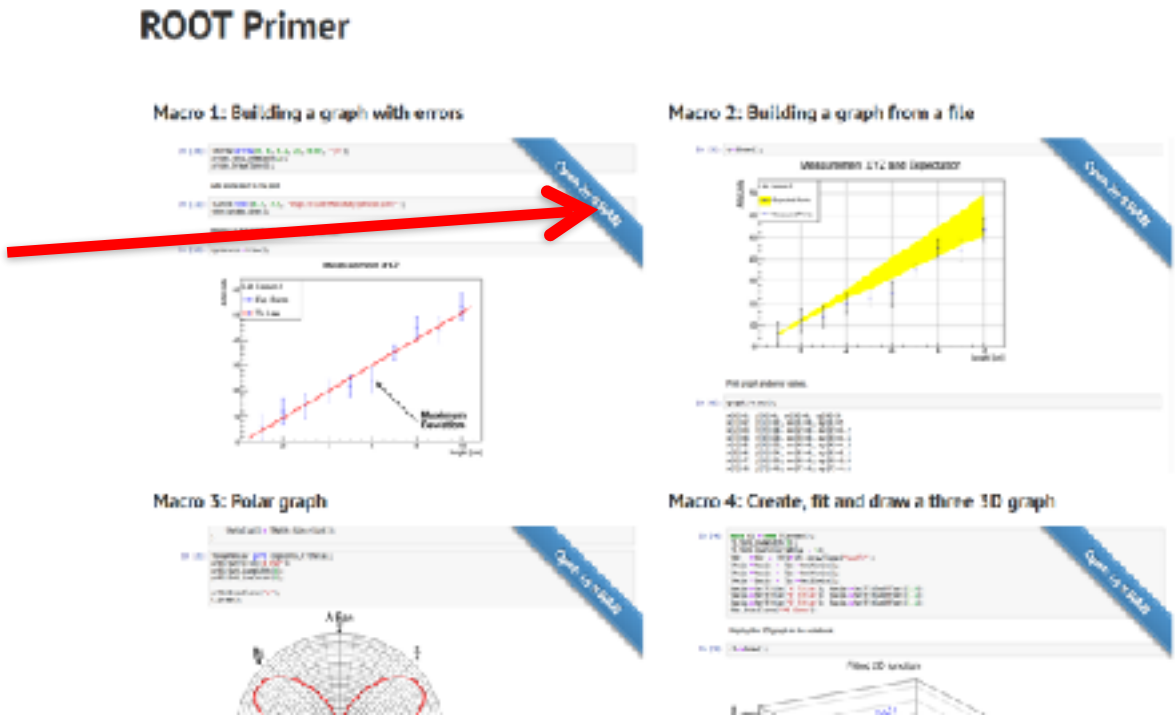
or it can be loaded into a ROOT session and then be executed by typing:

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

Macros available in SWAN

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

Click to open the Macro 1 notebook in SWAN

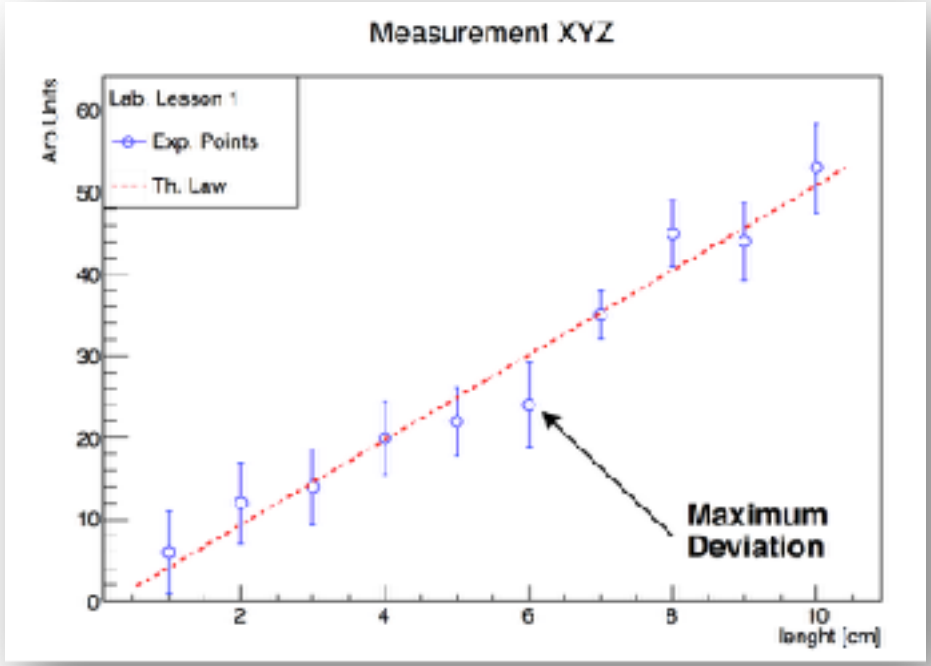


Macro1_py

A More Complex Example

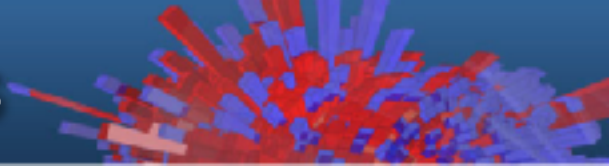
The example in section 3.2 of the ROOT primer, is a typical task in data analysis, a macro that constructs a graph with errors, fits a (linear) model to it and saves it as an image.

Let’s inspect it together and run it



Macro1_py

Summary of Visual Effects

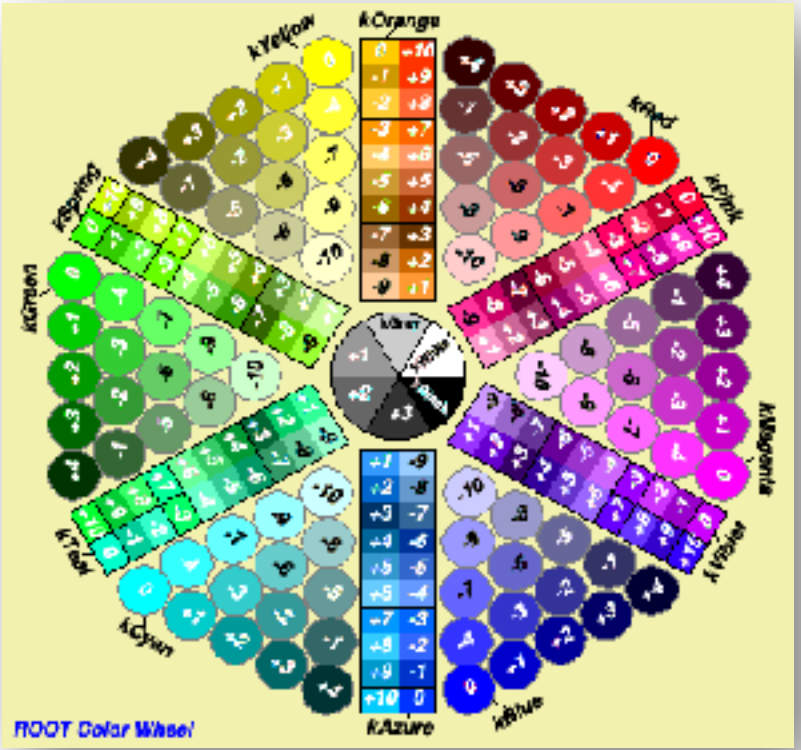


- **Colours and Graph Markers:** To specify a colour, some identifiers like kWhite, kRed or kBlue can be used for markers, lines, arrows etc. The complete summary of colours is represented by the ROOT “colour wheel”. ROOT provides several graphics markers like triangles, crosses or stars.

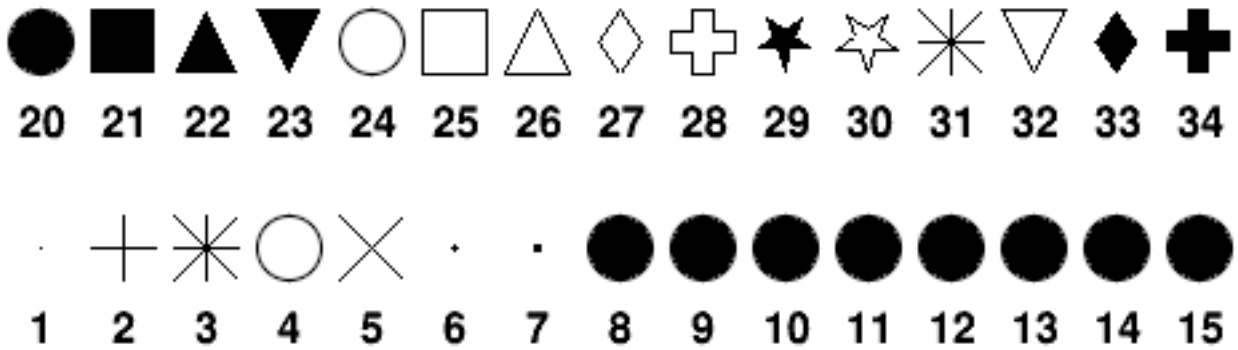
- **Arrows and Lines:** The class representing arrows is TArrow, which inherits from TLine. The constructors of lines and arrows always contain the coordinates of the endpoints.

- **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

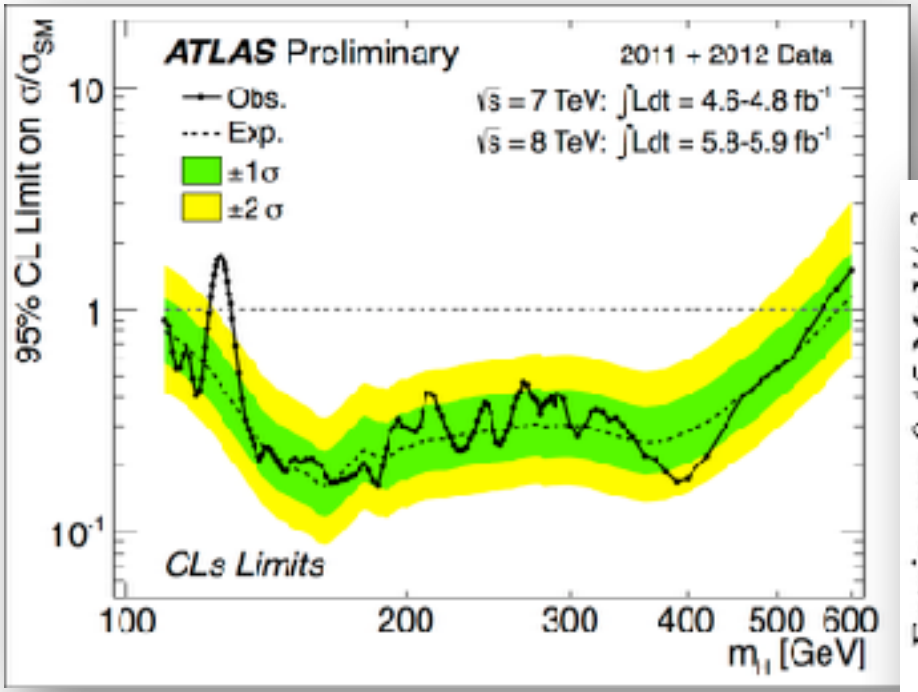
ROOT libraries can be also used to produce standalone, compiled applications:

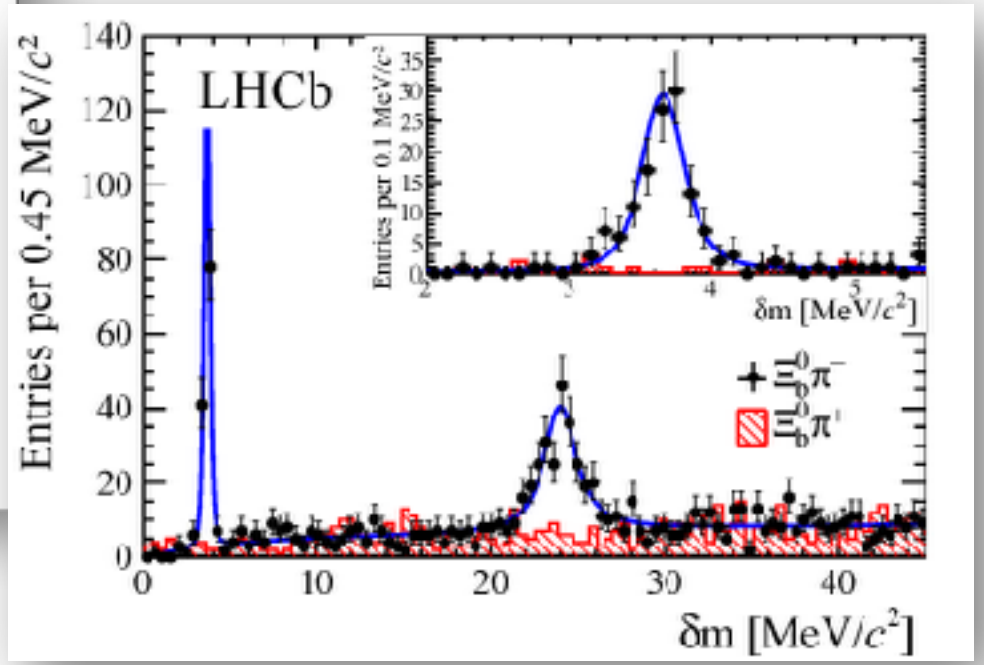
ExampleMacro.C

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

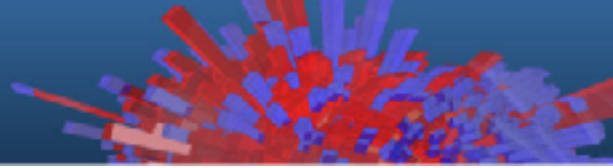
```
> g++ -o ExampleMacro ExampleMacro.C `root-config --cflags --libs`
> ./ExampleMacro
```

More about Graphs and Histograms





Graphs



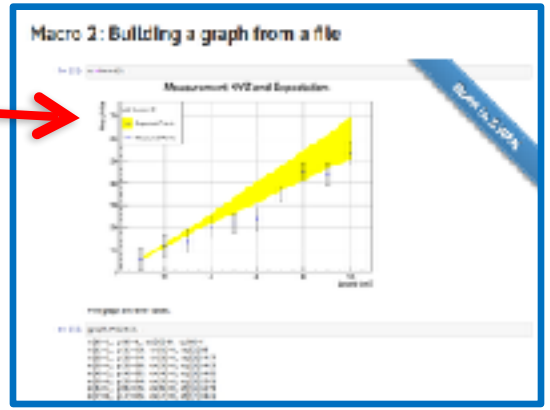
- Read Graph Points from File
- Polar Graphs
- 2D Graphs
- Multiple graphs

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="");
```

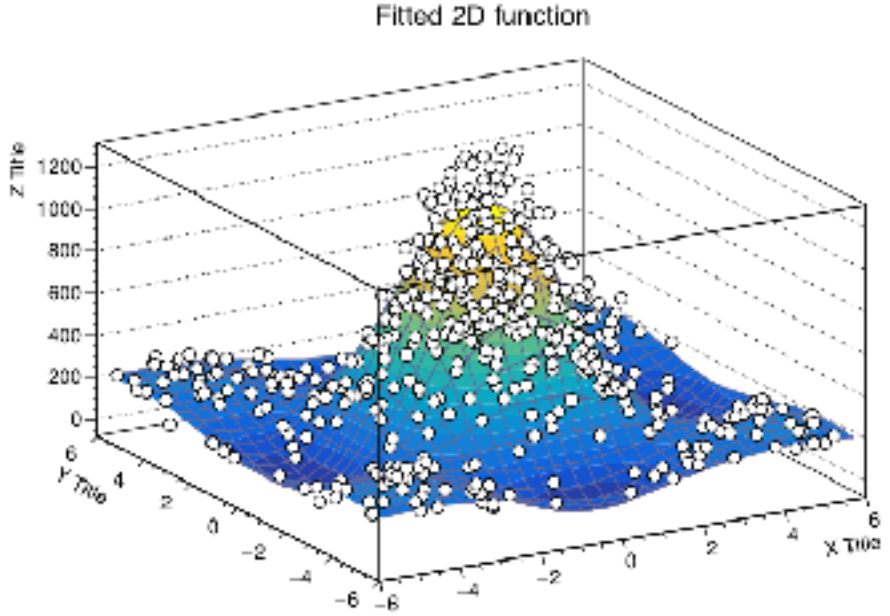
Let's have a look at macro2.C in a notebook (also in section 4.1 of the Primer).

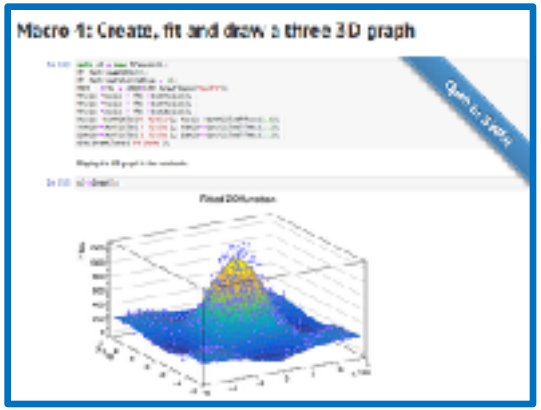


Macro2_py

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:

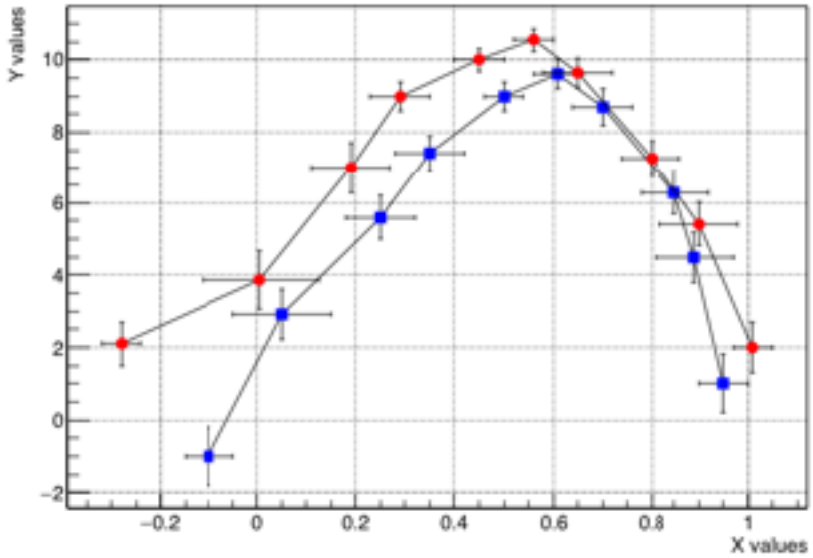


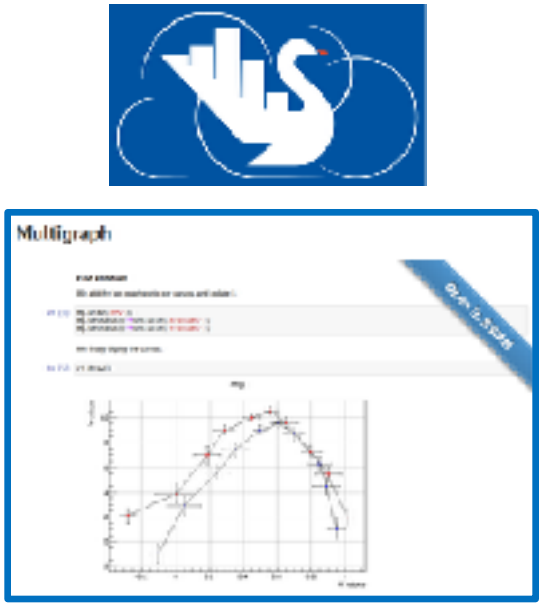


Macro4_py

Multiple Graphs

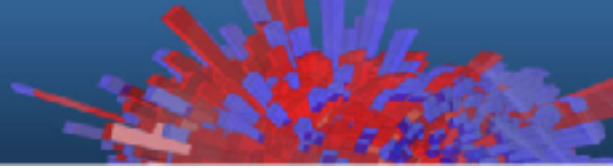
It is sometimes useful to group graphs in a single entity, for instance to compute a common axis system. The class *TMultiGraph* described in section 4.4 of the Primer allows that.





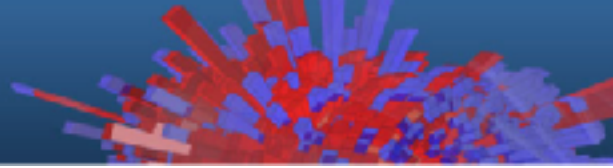
Multigraph_py

Histograms



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

Exercise

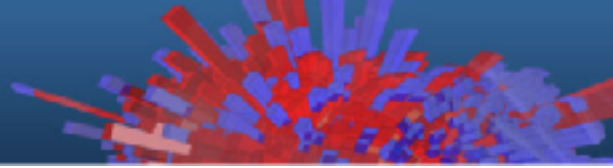


Write a macro to visualise a Poisson distribution in a histogram

- 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 but we will do it using Python in SWAN

Exercise - Optional



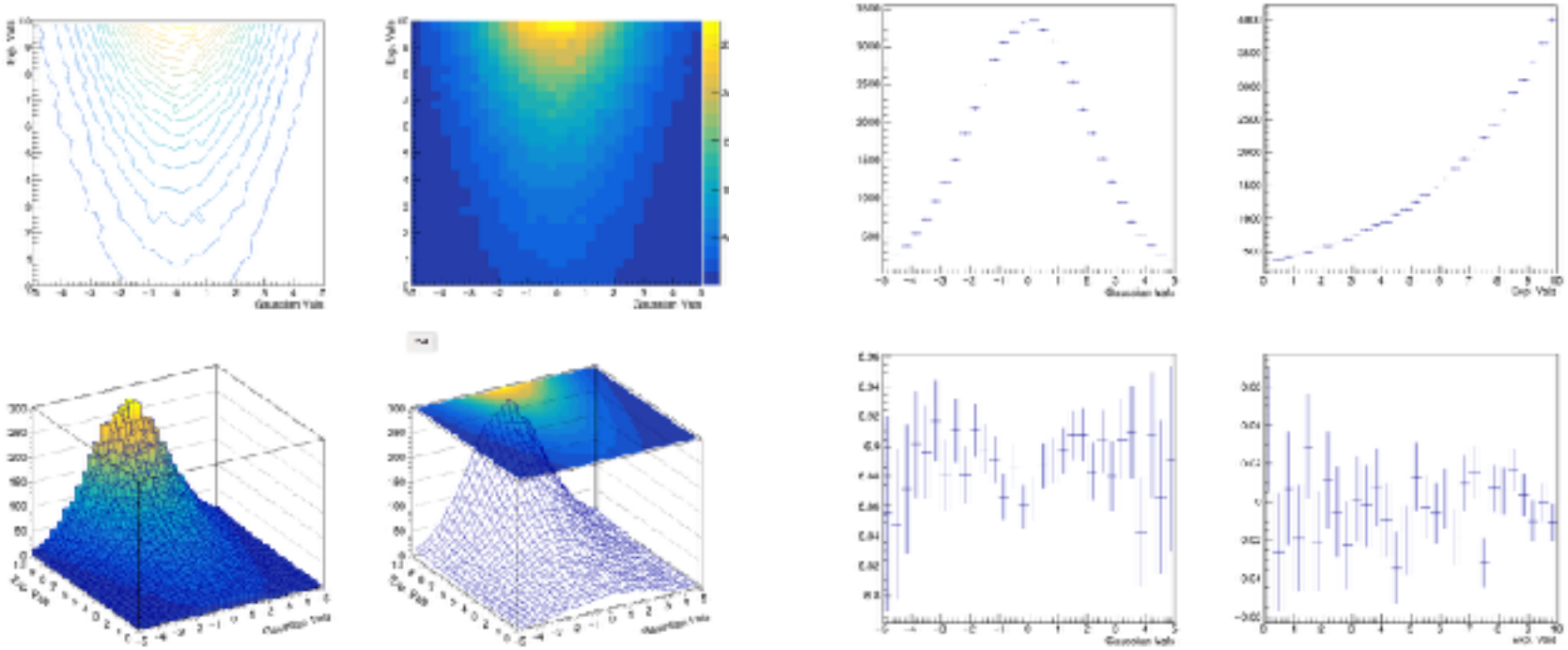
Create a macro that draws the sum, difference and ratio of two histograms

- 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

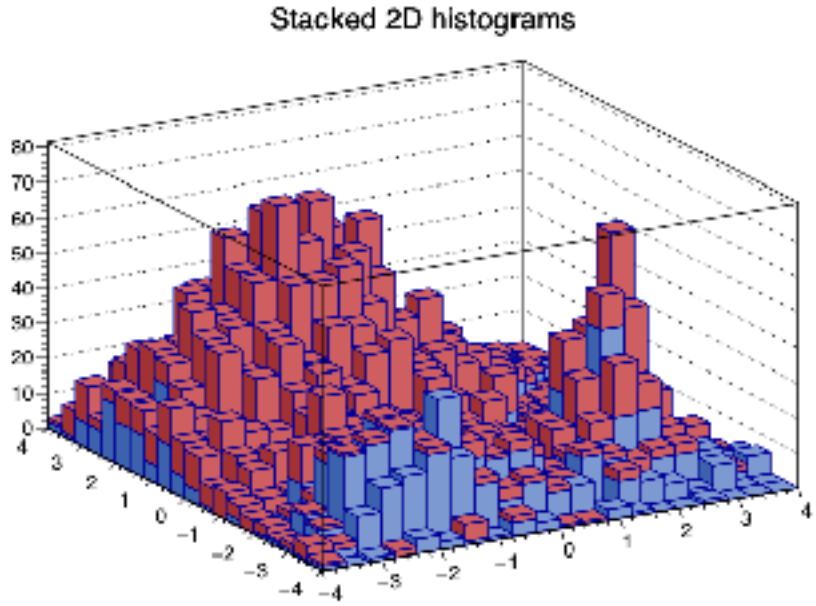
Two-dimensional histograms are a very useful tool, for example to inspect correlations between variables, as in the example in section 5.3 of the Primer (macro7.C):

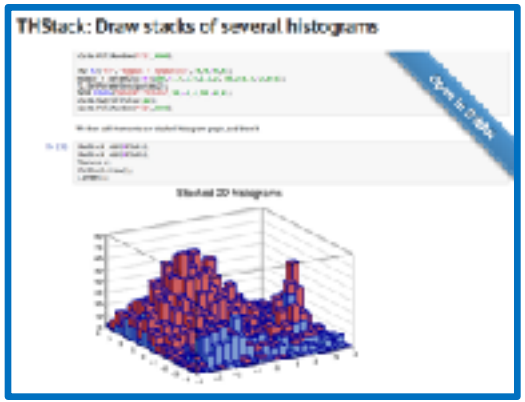


Multiple Histograms

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

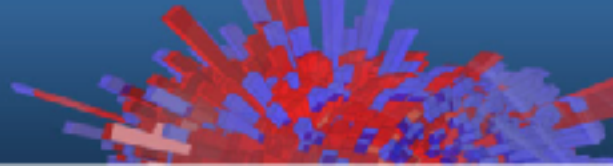
Class THStack





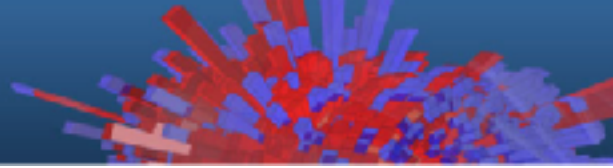
THStack_py

Input and Output



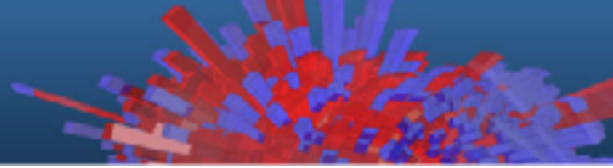
- Storing Objects
- N-tuples

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

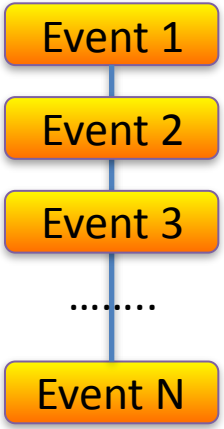


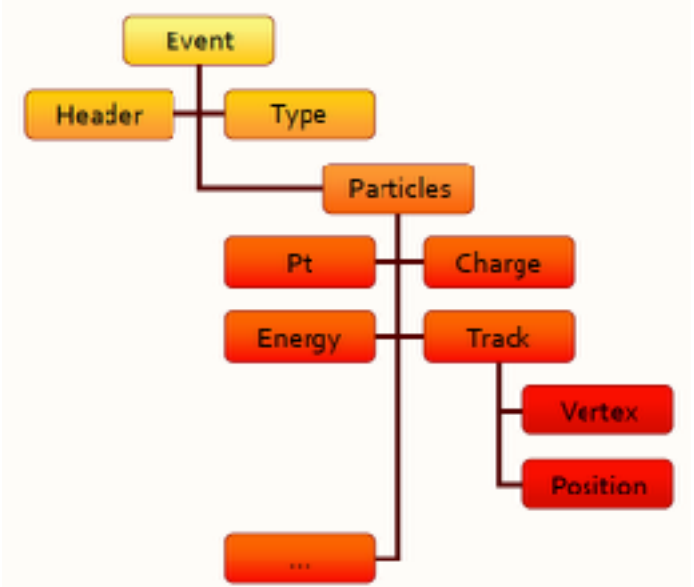
```
# Open a TFile
out_file = ROOT.TFile(“my_rootfile.root”,"RECREATE")
h = ROOT.TH1F("my_histogram","My Title;X;# of entries",100,-5,5)
h.FillRandom("gaus")
h.Write(); // Write the histogram in the file
out_file.Close(); // Close the file
```

Trees

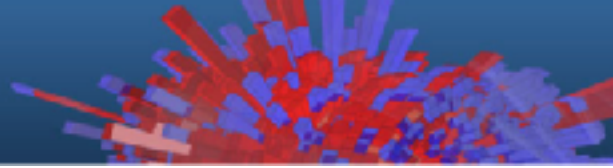
- The TTree is the data structure ROOT provides to store large quantities of same types objects
- Organised in branches, each one holding objects
- Organised in independent events, e.g. collision events
- Efficient disk space usage, optimised I/O runtime

old 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

```
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();
```

Exercise: Potential of the Tree

- Run the `write_ntuple_to_file_py` notebook
- Run the code in SWAN
  - Check your CERNBox for the result file!

Write_ntuple_to_file_py

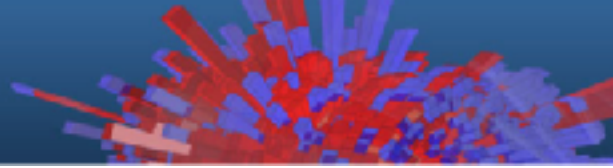
Creting TTree from csv files

- It is possible to fill a TTree from a csv file (e.g. from Excel)
- Use function TTree::ReadFile or TTree::ReadStream from a file or from a stream
  - need to specify a string describing the branches (the columns headers)

```
branchDescription = “named/C:x/F:y/F
tree.ReadFile(“myfile.root”, branchDescription)
```

ConvertData

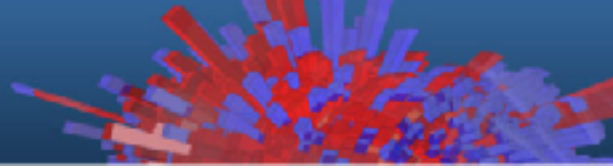
Accessing Complex Trees



- Class to access TTree from C++
- 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");
```

Reading TTree in Pyton



- Reading and analysing TTree in Python is easy
- One can iterate on a single entry in a TTree and access directly the branches

```
file = ROOT.TFile(“my_tree.root”)
t = file.tree    # retrieve the tree from the file
for event in t :
    print t.x, t.y  # print the content of the branches x and y
```

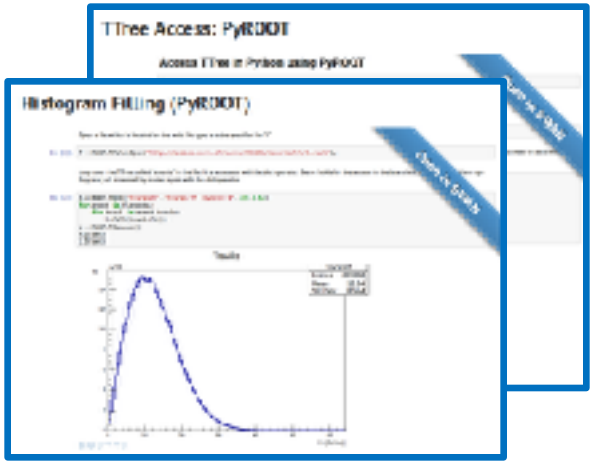
Exercise: Reading TTree from Python

Example of reading and examine a TTree from a file using Python

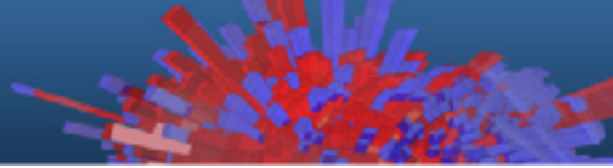
TTreeAccess_Example_py

Example of reading a TTree from a file and filling an histogram

FillHistogram_Example_py



Summary



Objectives reached:

- Become familiar with the ROOT toolkit
- Be able to use the Python interface and learn the Jupyter notebooks
- Plot data and play with the Histograms
- Perform basic I/O operations
- Working with the TTree class