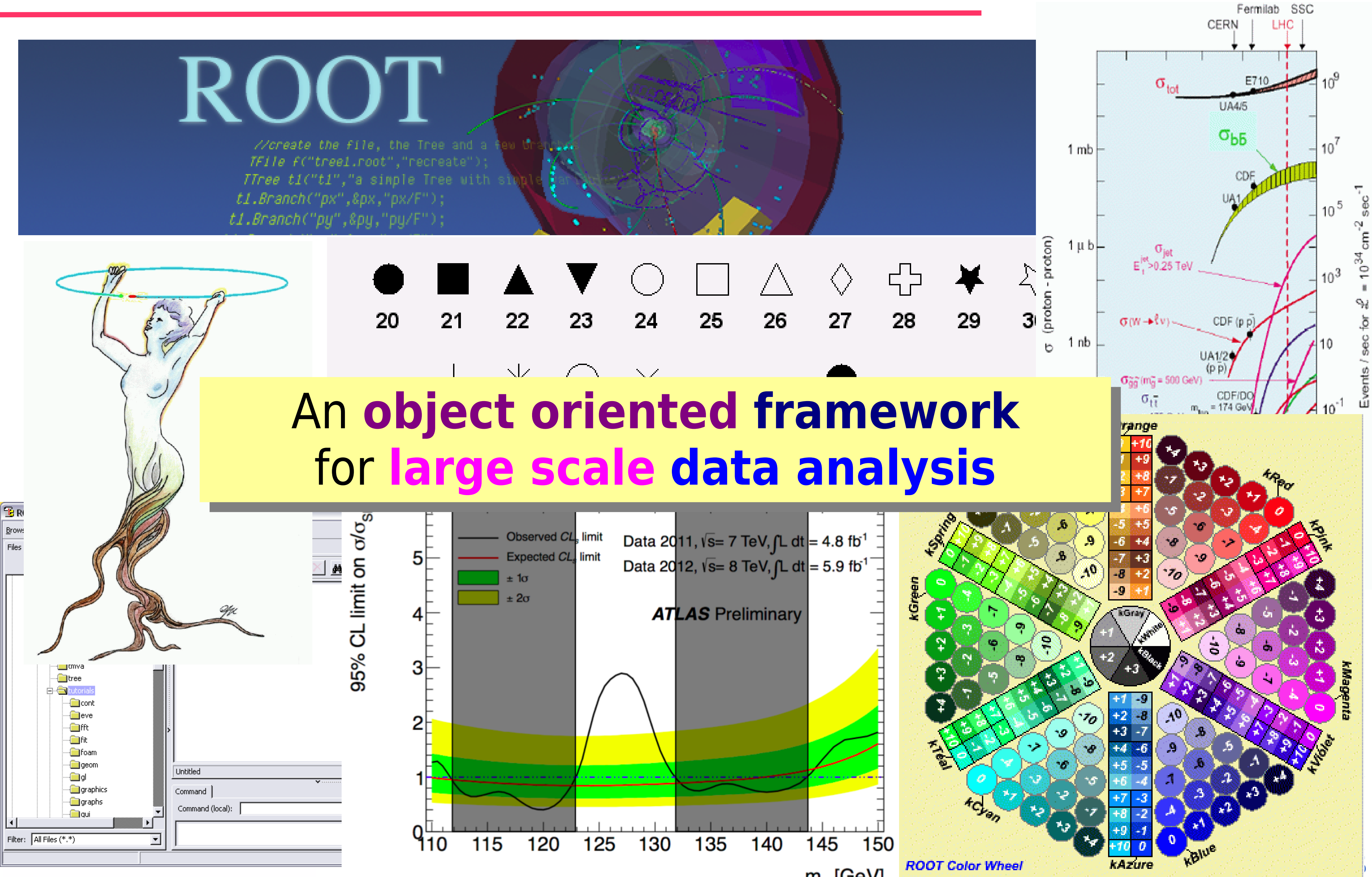
ROOT tutorial, part 1

Attilio Andreazza Università di Milano
Caterina Doglioni Université de Genève

HASCO school – 17/07/2012

What is ROOT? http://root.cern.ch/



An **object oriented framework**
for **large scale data analysis**

Object oriented…

What is "Object-Oriented Programming"? (1991 revised version)

Bjarne Stroustrup

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ROOT objects:
C++ classes with
data members,
member functions
inheritance relationships

WIKIPEDIA
The Free Encyclopedia

Main page
Contents

Article Talk | Read Edit View

Object-oriented programming

From Wikipedia, the free encyclopedia

Object-oriented programming (**OOP**) is a programming paradigm using "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs. Programming techniques may include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance. Many modern programming languages now support OOP, at least as an option.

...framework...

ROOT: a set of reusable **classes** and **libraries**

ROOT in **interactive mode**

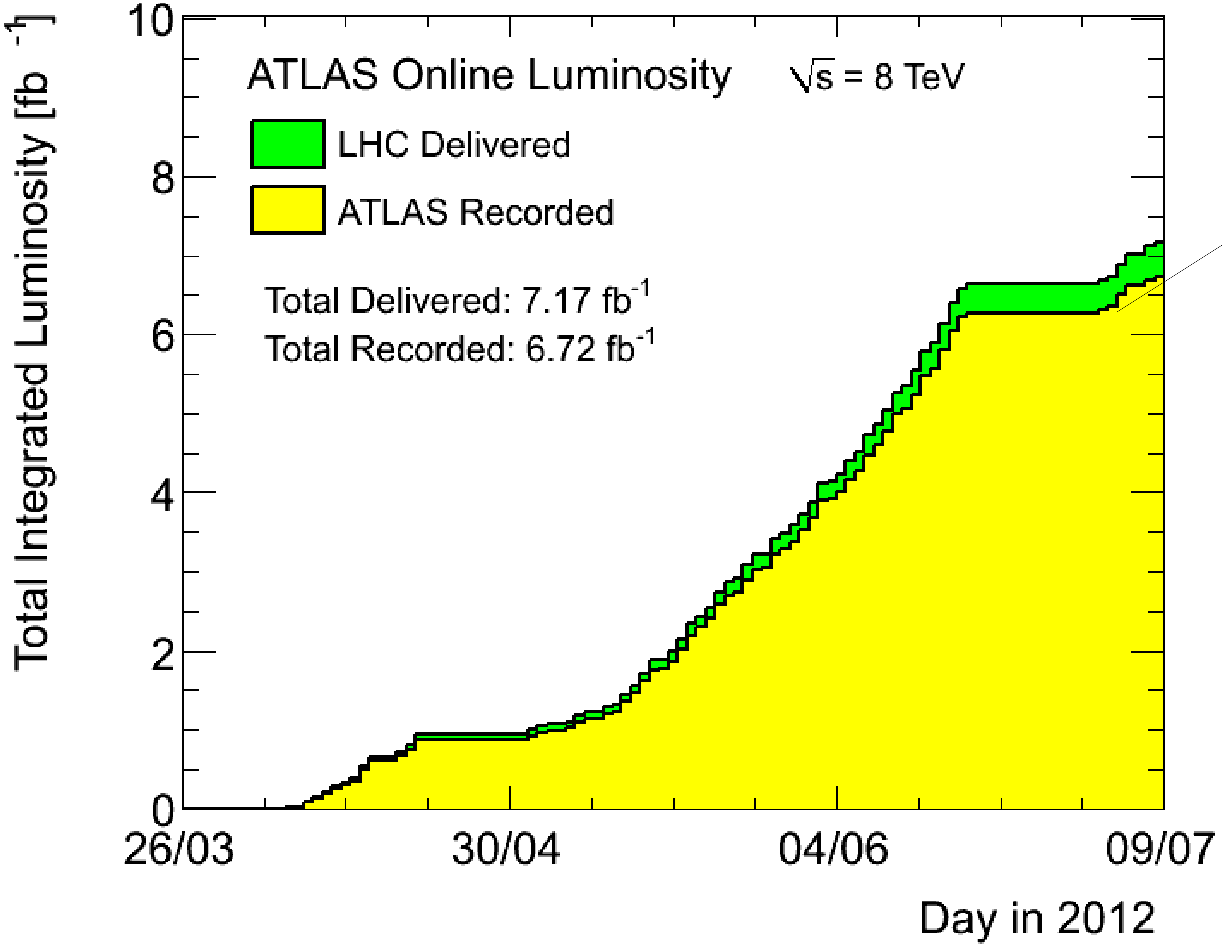
```
cate@catelenovolinux:~$ root -l
root [0]  TF1 *myFunction = new TF1("myFunction
","[0]+[1]*x",0,10);
root [1]
```

cate - root

ROOT in **compiled code**

```
#include "TF1.h"

int main() {

  TF1 *myFunction =
  new TF1("myFunction","[0]+[1]*x",0,10);

  delete myFunction;

  return(0);

}
```

...for large scale...

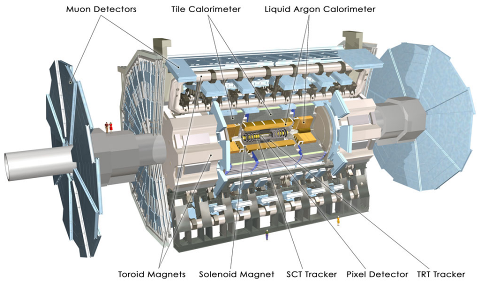


Enormous amount of data recorded by e.g. the LHC:

Need efficient data formats and tools to:
store the data
read data out (I/O)
extract information from data

(this plot has been made with ROOT)

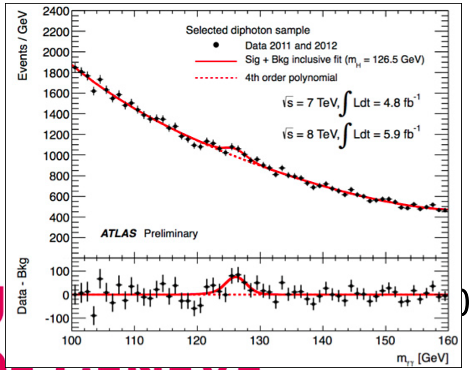
...data analysis



Raw data

Reconstructed/calibrated Physics objects

Ntuples



...very roughly...

Analysing data involves:

- Recording and storage of data/MC
- Reconstruction of physics objects
- Discrimination of signal from background (e.g. using cuts)
- Quantitative comparison of predictions to experimental results
- Presentation results (usually using plots)

...and much more:
ROOT is used to do all of this

this tutorial: **final data analysis**

Documentation and links

[Class reference]

[ROOTTalk (forum)]

[ROOT manual]

Tutorial disclaimer: partial / personal view of all that ROOT can do...

ROOT » HIST » HIST » TF1

class TF1: public TFormula, public TAttLine, public TAttFill, public TAttMarker

TF1: 1-Dim function class

A TF1 object is a 1-Dim function defined between a lower and upper limit.
The function may be a simple function (see `TFormula`) or a precompiled user function.
The function may have associated parameters.
TF1 graphics function is via the `TH1/TGraph` drawing functions.

The following types of functions can be created:

- A - Expression using variable x and no parameters
- B - Expression using variable x with parameters
- C - A general C function with parameters
- D - A general C++ function object (functor) with parameters
- E - A member function with parameters of a general C++ class

A - Expression using variable x and no parameters

ROOT Support

Moderator: rootdev

NEWTOPIC 11874 topics • Page 1 of 238 • 1 2 3 4 5 ...

TOPICS

ROOT has moved to Subversion...
by rdm » Wed Oct 10, 2007 11:54

Root in Cygwin on Win7 doesn't respond after s
by willfischer » Tue Jul 10, 2012 20:42

qtcint.so.5.31 cannot be built in trunk@41669
by Pepe Le Pew » Tue Nov 01, 2011 18:02

TTree::Draw and TVector Branches
by jfcaron » Thu Jul 05, 2012 22:22

User's Guide

The ROOT User's Guide has been translated into DocBook (Xml). The corrections and updates are now made in this new format. The new version is still under development, therefore we will continue to provide, for a limited duration, the old version (see below on this page).

Latest User's Guide (A4 format)
Latest User's Guide (HTML version)

Old version:

We will appreciate your comments on this edition. If you would like to contribute to a chapter, section, or even a paragraph, do not hesitate to contact us and send your comments to: rootdoc@root.cern.ch. You can also post your comments or questions in the section Documentation of the ROOT Forum.

Files available for download:	User's Guide v5.26 1 page per sheet ~11MB (with Hyper-links)	User's Guide v5.26 TwoInOne 2 pages per sheet ~7MB	User's Guide v5.26 MSWord Doc ~13MB

User's Guide v5.26
Preface, Table of Contents and Table of Figures

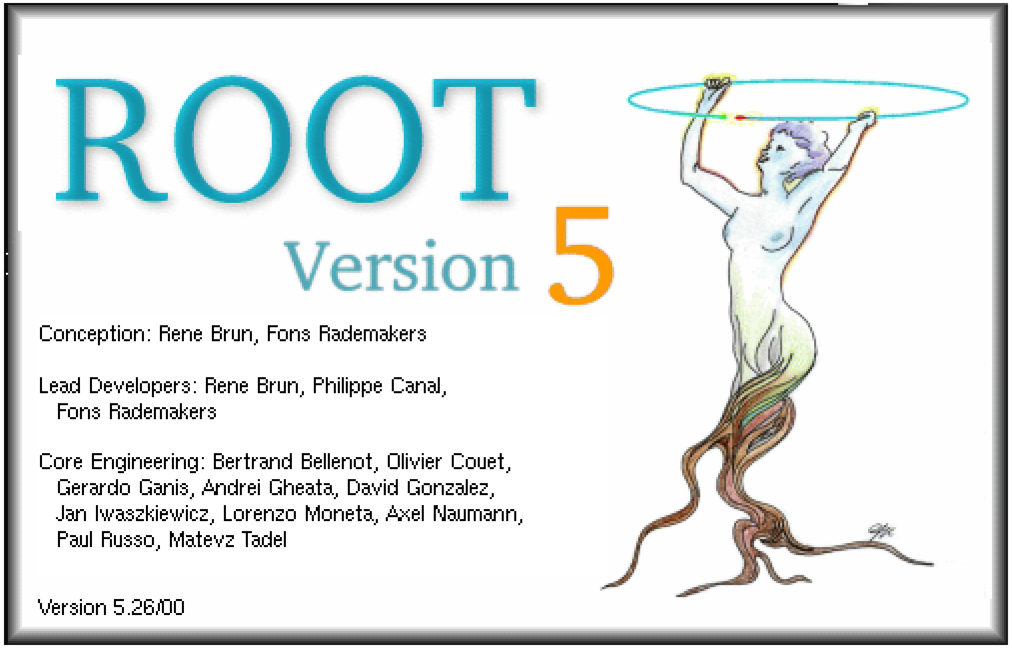
Using ROOT: interactive (CINT), ACLiC

From now on, raise your hand
if you want any of the lines of code
written out & demonstrated live!

ROOT Tutorial
HASCO school – 17/07/2012

Start and quit ROOT

```
cate@catelenovolinux:~$ root
  ******************************************
  *                                        *
  *        W E L C O M E  to  R O O T      *
  *                                        *
  *   Version   5.32/01  29 February 2012  *
  *                                        *
  *  You are welcome to visit our Web site *
  *          http://root.cern.ch           *
  *                                        *
  ******************************************

ROOT 5.32/01 (tags/v5-32-01@43181, Feb 29 2012,
x8664gcc)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```



No splash screen

```
cate@catelenovolinux:~$ root -l
root [0]
```

To quit

```
cate@catelenovolinux:~$ root -l
root [0] .q
cate@catelenovolinux:~$
```

To quit a stubborn session

```
root [0] .qqqq
Info in <TRint::ProcessLine>: Bye... (try '.qqqq
qqq' if still running)
cate@catelenovolinux:~$
```

CINT: necessary health warning

For most of this tutorial, we will use CINT

CINT is an **interpreter**, **not** a **compiler**

```
~ : root
File  Edit  View  Bookmarks  Settings  Help
cate@catelenovolinux:~$ root -l
root [0] TF1 f
root [1] f.SetName("Function, object")
root [2] f.GetName()
(const char* 0x22c9850)"Function, object"
root [3] f->GetName()
(const char* 0x22c9850)"Function, object"
root [4]
```

A compiler would complain about this liberal use of pointer operators on objects...

CINT has limitations, but it is easy to use on command line
and works reasonably for quick plotting purposes

E.g. one advantage: CINT will look for objects in the current directory and save you some typing

However, **bad idea** to learn C++ via CINT...

Macros in CINT

Unnamed macros

```
{

  TF1 myFunction;
  myFunction.SetName("myFunction");
  cout << myFunction.GetName() << endl;

}
```

```
cate@catelenovolinux:~$ root -l
root [0] .x m

missingRuns
mozilla.pdf
myFirstMacro.C
massResolution_J5.eps
root [0] .x myFirstMacro.C
myFunction
root [1]
```

Tab completion

Named macros

```
void MyFirstMacro(string textToSayHelloToTheFunction) {

  TF1 myFunction;
  myFunction.SetName("myFunction");

  cout << textToSayHelloToTheFunction << " "
       << myFunction.GetName() << endl;

}
```

Function argument

```
root [0] .L MyFirstMacro.C
root [1] M

MemInfo_t
MyFirstMacro
root [1] MyFirstMacro(

void MyFirstMacro(string textToSayHelloToTheFunction)
root [1] MyFirstMacro(

void MyFirstMacro(string textToSayHelloToTheFunction)
root [1] MyFirstMacro("Hello function named")
Hello function named myFunction
root [2]
```

Loads the macro so function can be executed

Macros in ACLiC

More info [on this link]

Compiled macros

Let's try with the named macro

```
~ : root
File  Edit  View  Bookmarks  Settings  Help
void MyFirstMacro(string textToSayHelloToTheFunction) {

  TF1 myFunction;
  myFunction.SetName("myFunction");

  cout << textToSayHelloToTheFunction << " "
       << myFunction.GetName() << endl;

}
```

```
root [0] .L MyFirstMacro.C+
Info in <TUnixSystem::ACLiC>: creating shared library /hom
e/cate/./MyFirstMacro_C.so
In file included from /home/cate/MyFirstMacro_C_ACLiC_dict
.h:34:0,
                 from /home/cate/MyFirstMacro_C_ACLiC_dict
.cxx:17:
/home/cate/./MyFirstMacro.C: In function ‘void MyFirstMacr
o(std::string)’:
/home/cate/./MyFirstMacro.C:3:3: error: ‘TF1’ was not decl
ared in this scope
/home/cate/./MyFirstMacro.C:3:7: error: expected ‘;’ befor
e ‘myFunction’
/home/cate/./MyFirstMacro.C:4:3: error: ‘myFunction’ was n
ot declared in this scope
/home/cate/./MyFirstMacro.C:6:3: error: ‘cout’ was not dec
lared in this scope
/home/cate/./MyFirstMacro.C:7:35: error: ‘endl’ was not de
clared in this scope
g++: error: /home/cate/MyFirstMacro_C_ACLiC_dict.o: No suc
h file or directory
Error in <ACLiC>: Compilation failed!
root [1]
```

Needs a bit more work...

Macros in ACLiC

More info on this link

Indicates a macro that you can try out in the tarball attached to the agenda

Compiled macros

```
~ : vim
File  Edit  View  Bookmarks  Settings  Help
#include <iostream>
#include <string>
#include "TF1.h"

using std::cout;
using std::endl;
using std::string;

void MyFirstMacro(string textToSayHelloToTheFunction) {

  TF1 myFunction;
  myFunction.SetName("myFunction");

  cout << textToSayHelloToTheFunction << " "
       << myFunction.GetName() << endl;

}
```

#includes (for each class used)

namespaces Standard library objects

```
root [0] .L MyFirstMacro.C+
Info in <TUnixSystem::ACLiC>: creating shared library /home
/cate/./MyFirstMacro_C.so
root [1] M

MemInfo_t
Mult
Mult
MyFirstMacro
root [1] MyFirstMacro(

void MyFirstMacro(string textToSayHelloToTheFunction)
root [1] MyFirstMacro("hello again function")
hello again function myFunction
root [2]
```

myFirstMacro.C

Compiled macros are **faster**!

Worth thinking about if e.g. reading events from file

Mini-introduction to OO in ROOT

ROOT Tutorial
HASCO school – 17/07/2012

An object in ROOT: TF1

A (mathematical) function ***TF1*** is an **object**: has **data members/methods**

Constructor:
makes an **instance** of the object

```
root [0] TF1 f("myFunction", "sin(x)/x", 0, 10)
root [1] f
(class TF1)40879392
```

Name — Formula — Range (min/max)

Methods:
ask for/modify **properties** of the object

```
root [3] cout << f.GetName() << endl
myFunction
root [4]
root [4] f.SetName("myFunctionWithNewName")
root [5]
root [5] cout << f.GetName() << endl
myFunctionWithNewName
```

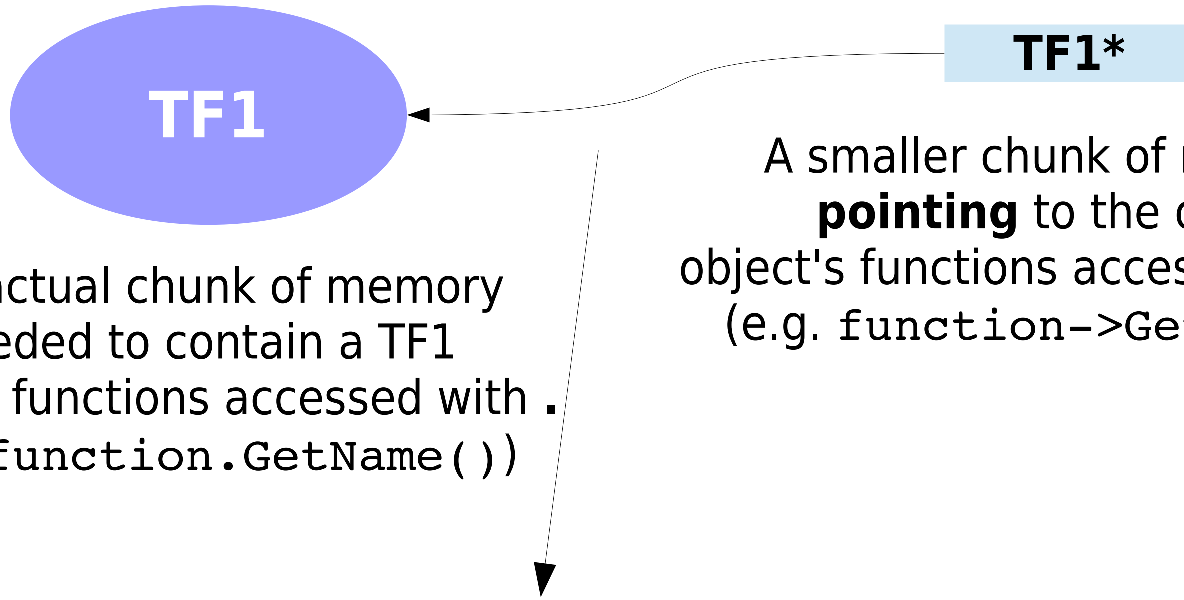
Data members:
properties of the object
generally inaccessible to us (encapsulation)
can be modified with **setters/getters**

From the **TF1.h** class

```
Double_t     fXmin;        //Lower bounds for the range
Double_t     fXmax;        //Upper bounds for the range
```

An object in memory: TF1*

What is the difference between an **object** and a **pointer to an object**?



The actual chunk of memory needed to contain a TF1 object's functions accessed with **.** (e.g. `function.GetName()`)

A smaller chunk of memory **pointing** to the object object's functions accessed with **->** (e.g. `function->GetName()`)

Nasty things can happen if this link is broken (e.g. pointer doesn't point anywhere anymore...)

```
root [1] invalidpf->GetName()
Error: illegal pointer to class object invalidpf 0x0 743
tmpfile):1:
*** Interpreter error recovered ***
```

Good practice to **check the pointer**: a broken link will show up as a **null pointer**

```
root [2] if (invalidpf == 0) cout << "Invalid pointer!" <<
endl
Invalid pointer!
```

An object in memory: TF1*

What is the difference between an **object** and a **pointer to an object**?

Main difference (to me): **persistency**

```
root [8] TF1 of("myFunction","sin(x)/x",0,10)
```

Object lives in the memory **stack**
→ **memory gets freed** automatically
when object goes **out of scope**

```
for (unsigned int i=0; i<100000; i++) {
  TF1 pf("myFunction", "sin(x)/x", 0,100);
} //at every step, memory is freed
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM
10483	cate	20	0	74100	18m	8828	S	24.6	0.2

An object in memory: TF1*

What is the difference between an **object** and a **pointer to an object**?

Main difference (to me): **persistency**

```
root [6] TF1 * pf = new TF1("myFunction","sin(x)/x", 0, 10)
```

MemoryLeak.C

```
for (unsigned int i=0; i<100000; i++) {
  TF1 * pf = new TF1("myFunction", "sin(x)/x", 0,100);
}
```

Associated object lives in the memory **heap**
→ **memory does not get freed** automatically
when it goes **out of scope**

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM
10256	cate	20	0	3627m	3.5g	8868	S	56.6	45.3

2 GB of functions!

```
root [9] delete pf
```

especially in compiled code,
every **new** needs a **delete to free the memory...**
otherwise **memory leak**

Another object in ROOT: TH1

Most famous object in ROOT: **histogram (TH1...)**

Various types of histograms depending on type of content:
e.g. **TH1D**: bins filled with doubles
TH1I: bins filled with integers

- 1-D histograms:
  - TH1C : histograms with one byte per channel. Maximum bin content = 127
  - TH1S : histograms with one short per channel. Maximum bin content = 32767
  - TH1I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH1F : histograms with one float per channel. Maximum precision 7 digits
  - TH1D : histograms with one double per channel. Maximum precision 14 digits

```
TH1
 ^
 |
 |
 |
-----------------------------------
 |        |      |      |     |     |
 |      TH1C   TH1S   TH1I  TH1F  TH1D
```

Many properties and functionalities in common
→ **inheritance** from common class **TH1**
~ all functions of TH1 will be **inherited** by **derived classes**

Most ROOT objects inherit from **TNamed** class → all have a **SetName** function

Interlude: naming conventions

How does ROOT call its classes and functions?

- Class names start with capital T, e.g. **TF1**
- Class data members start with f, e.g. **fXmin**
- Names of non-class data types end with _t: e.g. **Int_t**
- Class methods start with _t: e.g. **GetName()**
- Global variable names start with _t: e.g. **gPad**
- Constant (or enumerator) names start with k: e.g. **kTrue**
- Words in names are capitalized: e.g. **GetLineColor()**
- Two subsequent capital letters are avoided: e.g. **GetXaxis()**

Objects in files

ROOT Tutorial
HASCO school – 17/07/2012

TFile: opening for reading

TFile: how to persistify ROOT's objects

Reading objects from a file

```
root [0] TFile * myFile = TFile::Open("example.root", "READ")
```

Returns a pointer to a Tfile

Opening option: will not modify the file

```
root [3] myFile->ls()
TFile**         example.root
 TFile*         example.root
  KEY: TH1F     cut_flow;1      cut_flow
  KEY: TH1F     averageIntPerXing;1     averageIntPerXing
  KEY: TH1F     delta_eta;1     delta_eta
  KEY: TH1F     delta_phi;1     delta_phi
  KEY: TH1F     mjj;1   mjj
```

Like unix's ls fuction, list the file content

A bit of pointer gymnastic: Get() returns a TObject, need to cast it to the correct object in order to access the pointer later

```
root [6] TTree *myTree = (TTree*)myFile->Get("highestMjjEvents")
```

TFile: writing objects

TFile: how to persistify ROOT's objects

Writing objects on a new file

```
root [0] TFile * myFile = TFile::Open("myNewFile.root", "RECREATE")
root [1] myFile.ls()
TFile**         myNewFile.root
 TFile*         myNewFile.root
```

Opening option: will overwrite any existing file with the same name (alternative: `UPDATE`)

```
root [2] TF1 * myFunction = new TF1("myFunction", "sin(x)/x", 0, 10)
root [3] myFunction->Write()
root [4] myFile->ls()
TFile**         myNewFile.root
 TFile*         myNewFile.root
  KEY: TF1      myFunction;1    sin(x)/x
```

Simply write the function(object) to the file

```
root [5] myFunction->Write("theCopyOfMyFunction")
(Int_t)212
root [6]
root [6] myFile->ls()
TFile**         myNewFile.root
 TFile*         myNewFile.root
  KEY: TF1      myFunction;1    sin(x)/x
  KEY: TF1      theCopyOfMyFunction;1   sin(x)/x
```

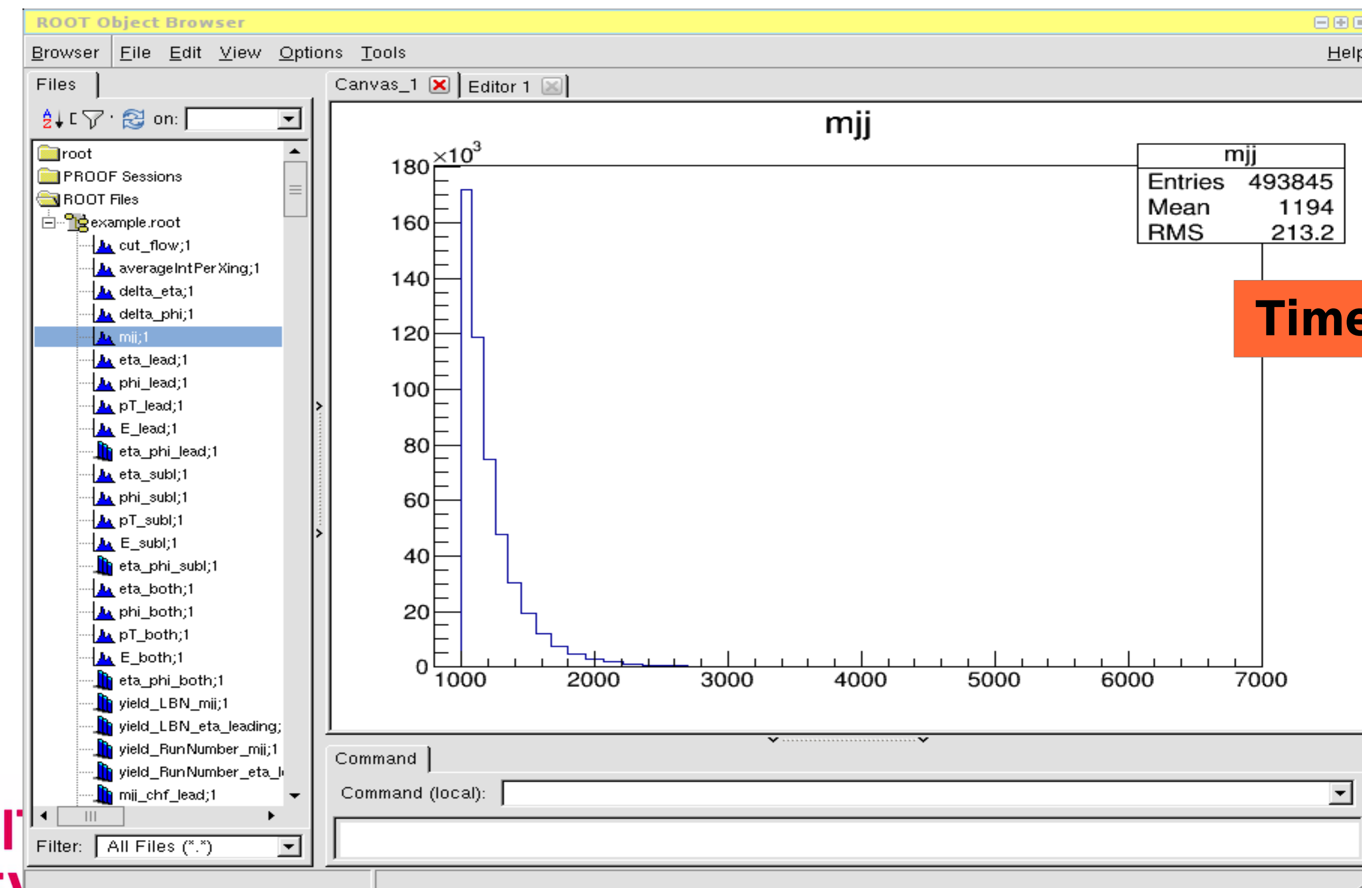
Write the function to the file with a different name

TBrowser: ROOT's GUI

TBrowser: convenient way of accessing objects quickly

```
cate@catelenovolinux:~/Work/HASCO$ root -l example.root
root [0]
Attaching file example.root as _file0...
root [1] TBrowser b
```

List of filenames to be opened by ROOT and put in current directory



Time for a demo

Functions: TF1s

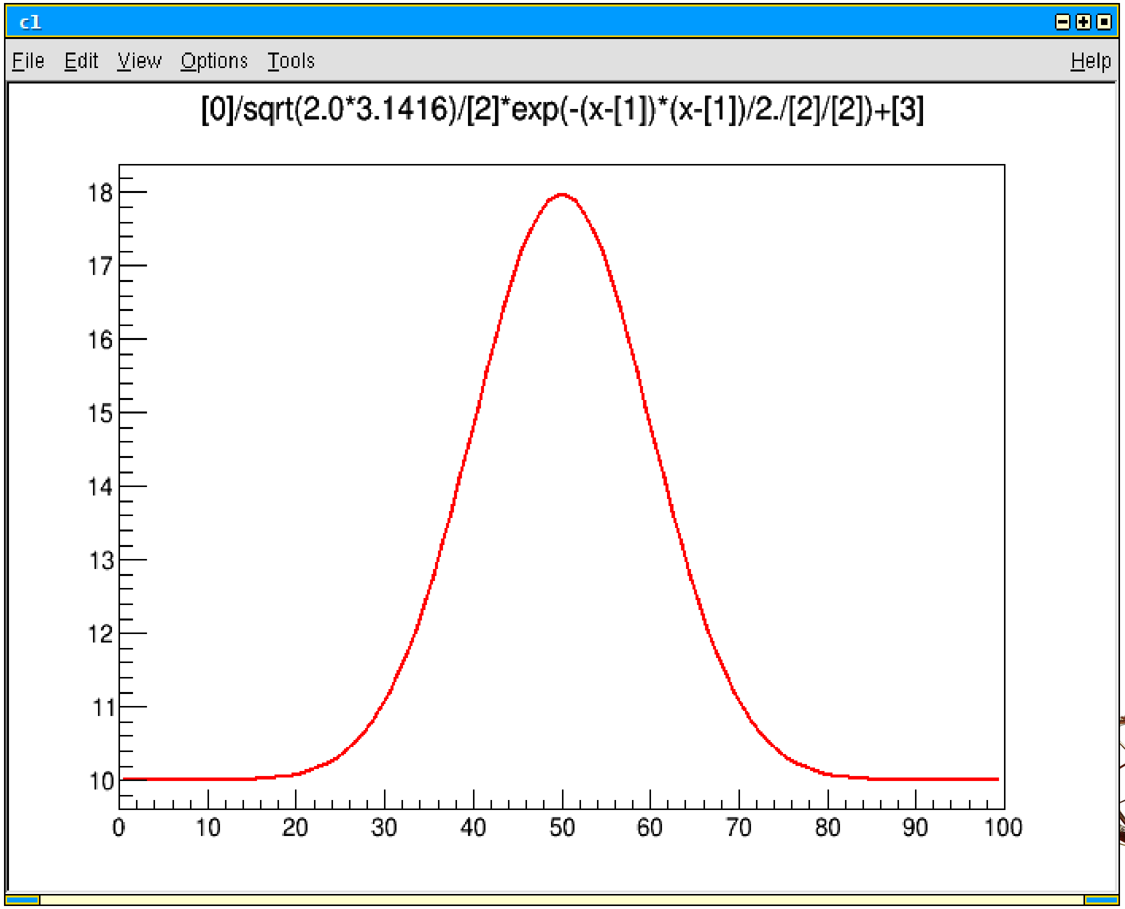
ROOT Tutorial
HASCO school – 17/07/2012

TF1 with parameters

A function can have **parameters** (e.g. floating parameters for fits...)

```
{

  TF1 * f1 = new TF1("f1",
                     "[0] / sqrt(2.0 * 3.1416) / [2] * exp(-(x-[1])*(x-[1])/2./[2]/[2]) + [3]",
                     0., 100.);
  f1->SetParameter(0, 200.0);
  f1->SetParameter(1, 50.0);
  f1->SetParameter(2,  10);
  f1->SetParameter(3,  10);
  f1->Draw();

}
```

GaussianWithOffset.C

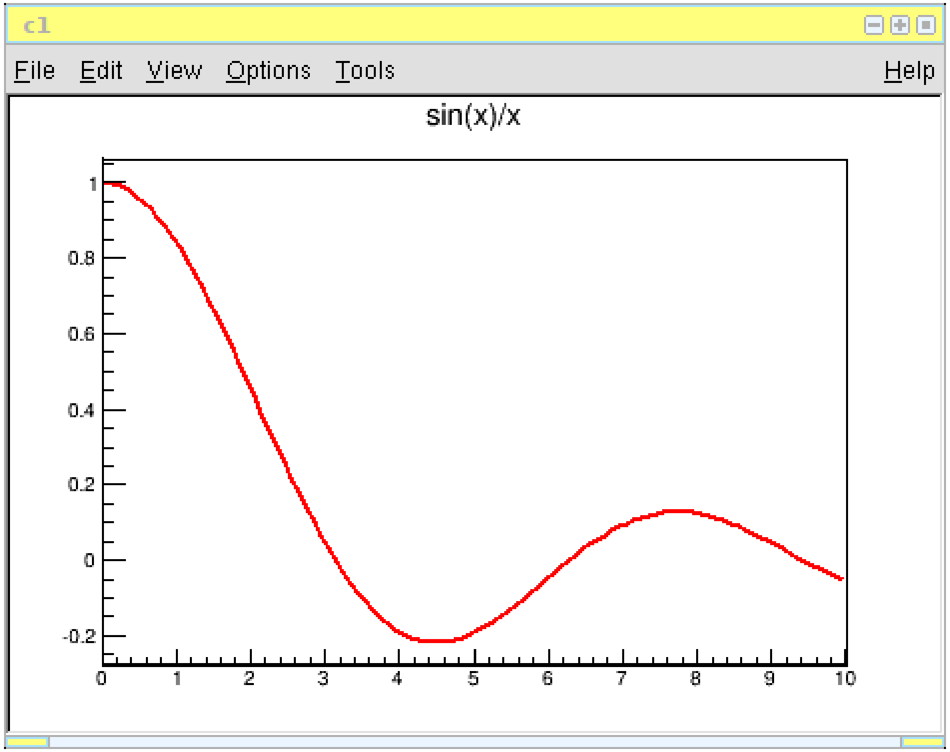



```
root [6] f1->GetParameter(0)
(const Double_t)2.00000000000000000e+02
root [7] f1->GetParameter(1)
(const Double_t)5.00000000000000000e+01
root [8] f1->GetParameter(2)
(const Double_t)1.00000000000000000e+01
root [9] f1->GetParameter(3)
(const Double_t)1.00000000000000000e+01
```

Let's draw a TF1 on a TCanvas

Like most objects in ROOT, functions can be **drawn** on a **canvas**

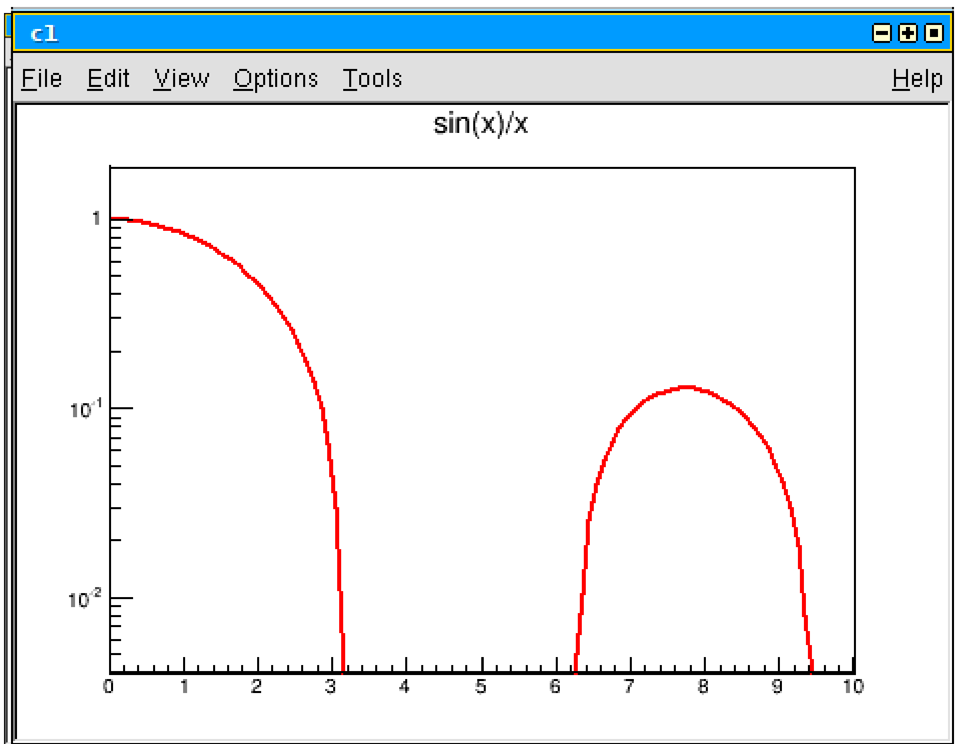
```
root [3] TF1 of("myFunction","sin(x)/x",0,10)
root [4] of.Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas
with name_c1
```



Let's draw a TF1 on a TCanvas

Like most objects in ROOT, functions can be **drawn** on a **canvas**

```
root [3] TF1 of("myFunction","sin(x)/x",0,10)
root [4] of.Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas
with name_c1
```



gPad: global variable pointing to current canvas

```
root [7] gPad->GetName()
(const char* 0x24905d9)"c1"
```

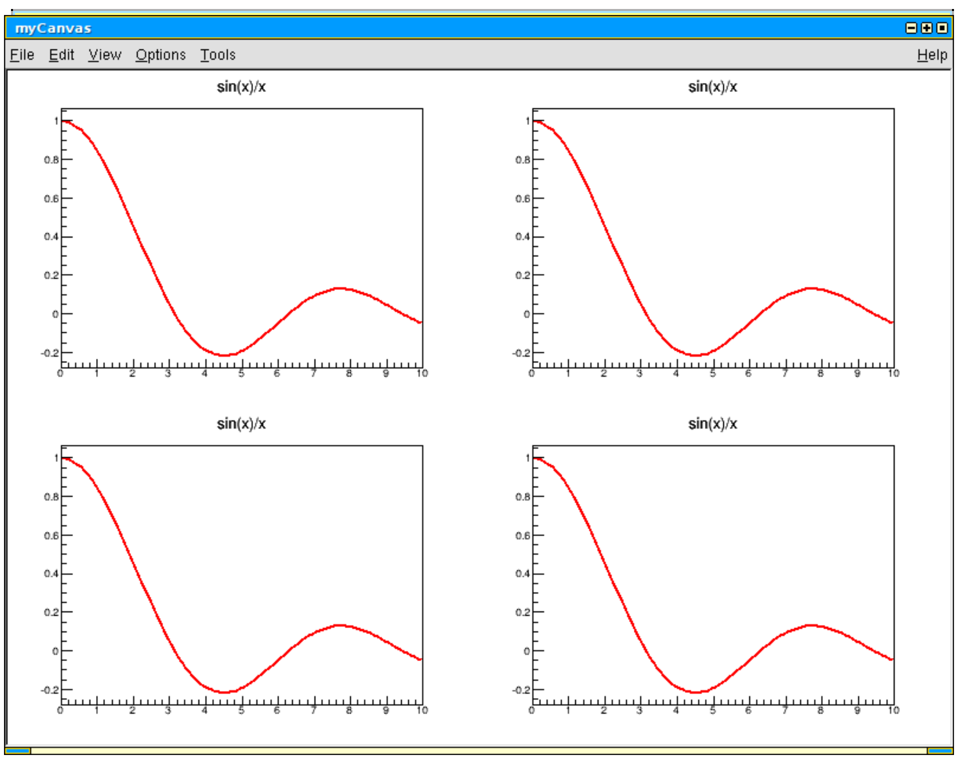
gPad controls properties of current canvas, e.g. log scale

```
root [8] gPad->SetLogy()
```

Let's draw a TF1 on a TCanvas

Like most objects in ROOT, functions can be **drawn** on a **canvas**

```
root [3] TF1 of("myFunction","sin(x)/x",0,10)
root [4] of.Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas
with name_c1
```



a **TCanvas** is an object too...

```
root [10] TCanvas c("myCanvas", "myCanvas"
, 800, 600)
```

...it can be divided in **TPads**

```
root [1] c.Divide(2,2)
root [3] c.cd(2)
(class TVirtualPad*)0x242a890
root [4] of.Draw()
root [5] c.cd(1)
(class TVirtualPad*)0x242a510
root [6] of.Draw()
root [7] c.cd(3)
(class TVirtualPad*)0x242ac30
root [8] of.Draw()
root [9] c.cd(4)
(class TVirtualPad*)0x242afb0
root [10] of.Draw()
```

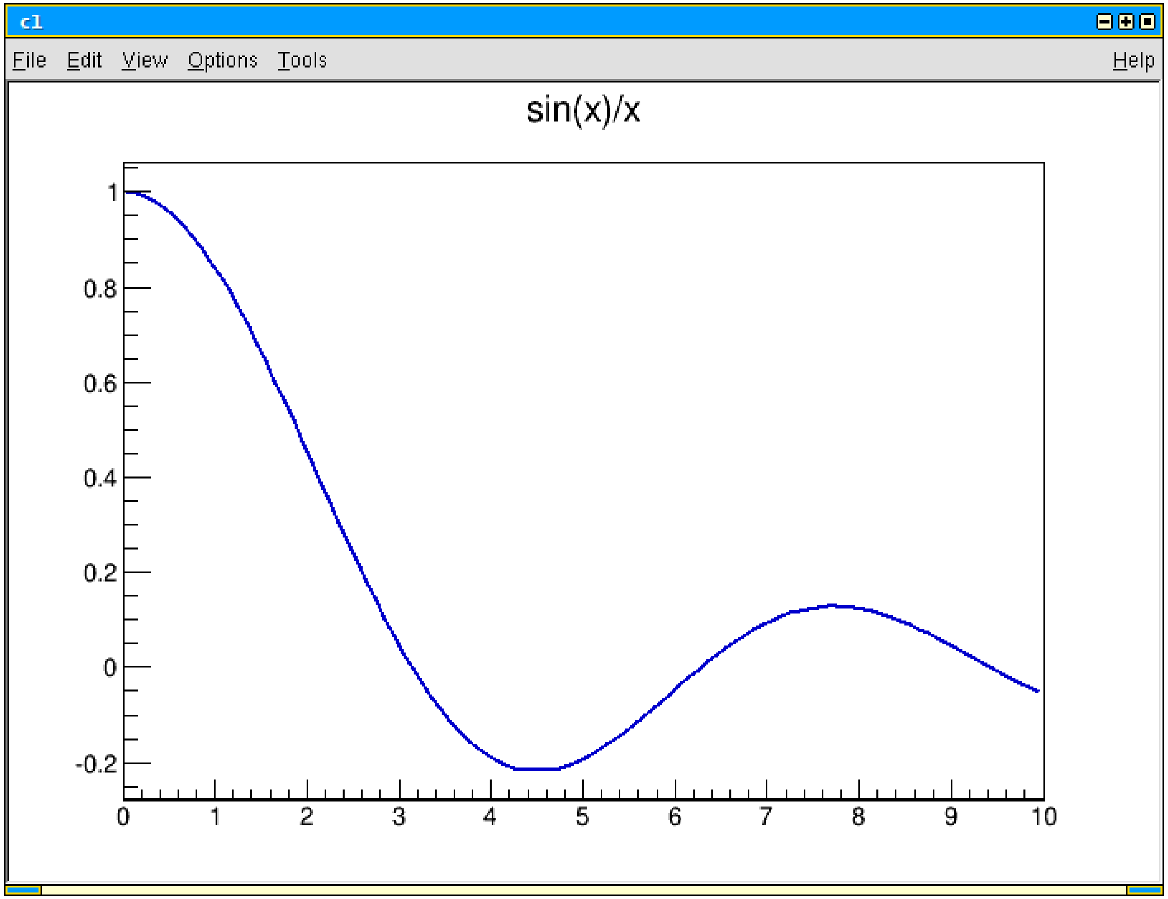
...and saved as an image

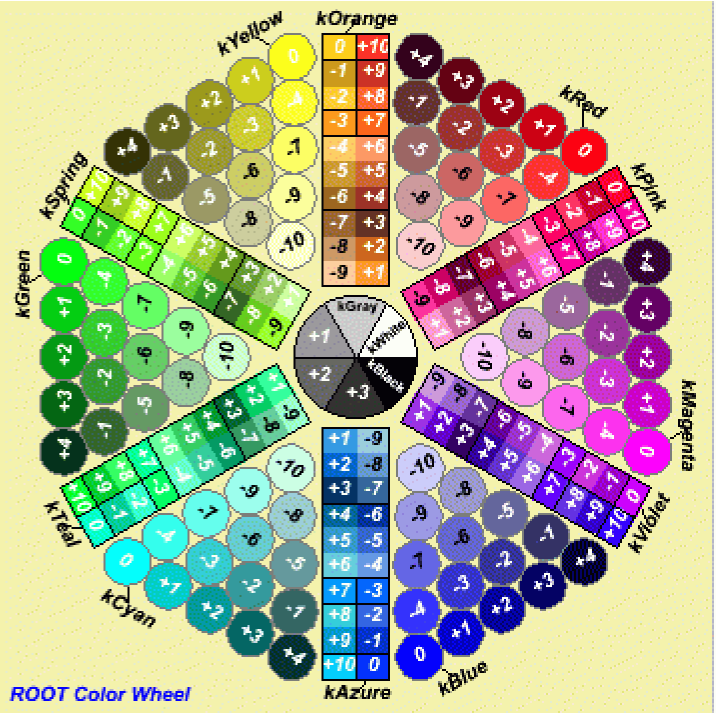
```
root [6] c.SaveAs("myFunction.png")
Info in <TCanvas::Print>: png file myFunction.png has
 been created
```

Formatting TF1s

Graphical properties of TF1 can be changed

```
root [2] of.SetLineColor(kBlue+1)
root [3] of.Draw()
```



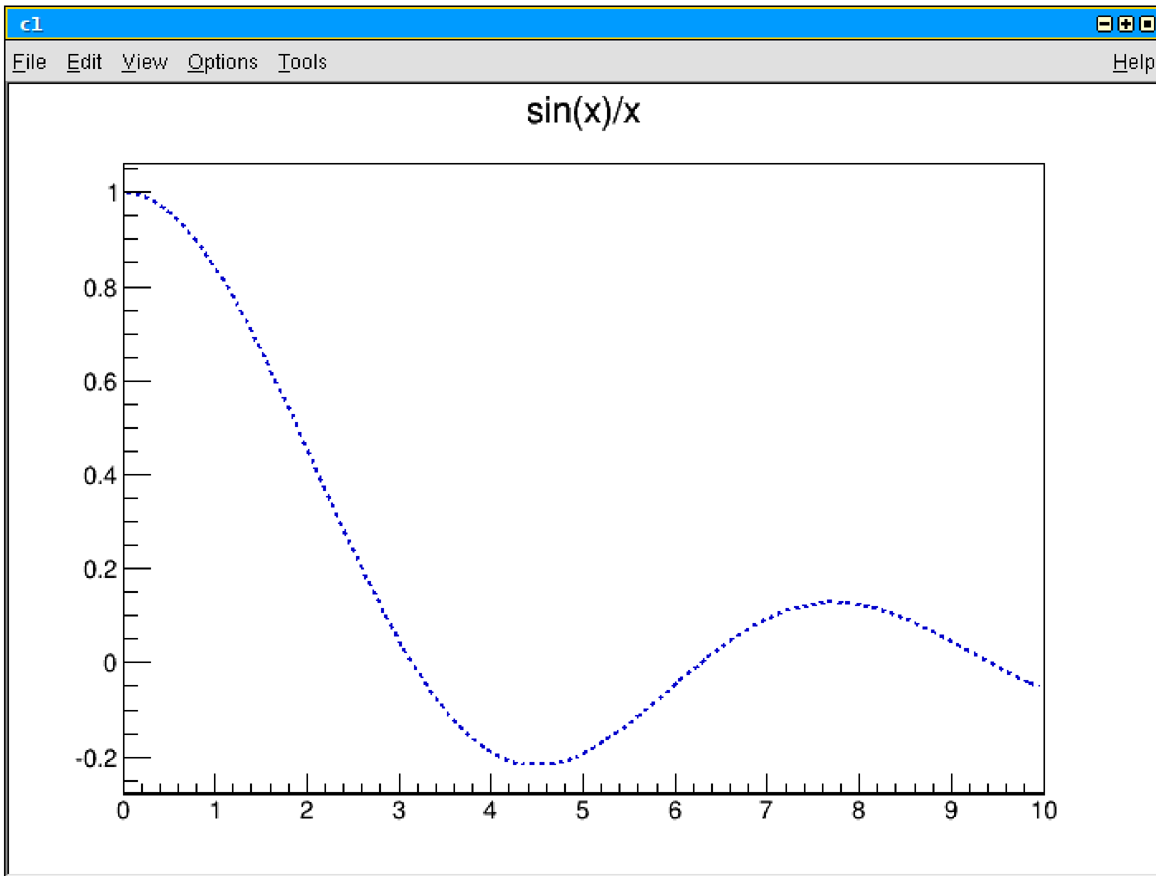


This will work for histograms too!

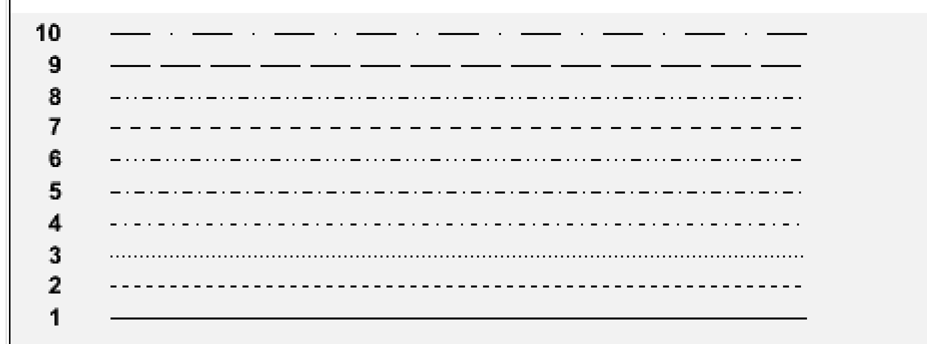
Formatting TF1s

Graphical properties of TF1 can be changed

```
root [4] of.SetLineStyle(2)
root [5] of.Draw()
```



Some available line styles



This will work for histograms too!

Histograms: TH1/TH2s

ROOT Tutorial
HASCO school – 17/07/2012

1-dimensional histograms (1)

1-D histograms can be instantiated in various ways

With fixed bin size

TH1D (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)

```
TH1F *myHistogram = new TH1F("myHistogram", "My histogram title", 100, 0, 4.4);
```

With variable bin size

TH1D (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins)

C array with low edges for each bin + high edge of last bin

```
root [7] Double_t Bins[4] = {0,2,5,8}
root [8] TH1F *myHistogram_varBinSize = new TH1F("myHistogram_varBinSize", "My histogram title", 3, Bins);
```

The number of bins is equal to the number of elements in the vector of bins **minus one**

1-dimensional histograms (2)

Filling a histogram, getting information from a histogram

```
{ TH1Basic.C

Double_t Bins[4] = {0,2,5,8};
TH1F *myHistogram_varBinSize = new TH1F("myHistogram_varBinSize", "My histogram
title", 3, Bins);

myHistogram_varBinSize->Fill(1);

cout << "Bin 1 now has "
     << myHistogram_varBinSize->GetBinContent(1)
     << " entries"
     << endl;

myHistogram_varBinSize->Print("all");

}
```

Can also:

- fill with **weights**: call `Fill(xEntry, weight)` and `TH1::SetSumw2` for calculating errors correctly
- Set **entire bin content**: call `setBinContent(iBin, binContent)`

```
root [0] .x TH1Basic.C
Bin 1 now has 1 entries
TH1.Print Name  = myHistogram_varBinSize, Entries= 1, Total sum= 1
 fSumw[0]=0, x=-1.33333
 fSumw[1]=1, x=1
 fSumw[2]=0, x=3.5
 fSumw[3]=0, x=6.5
 fSumw[4]=0, x=9.33333
```

Useful when no graphic session

1-dimensional histograms (3)

Useful information on **bin conventions**

Overflows and underflows

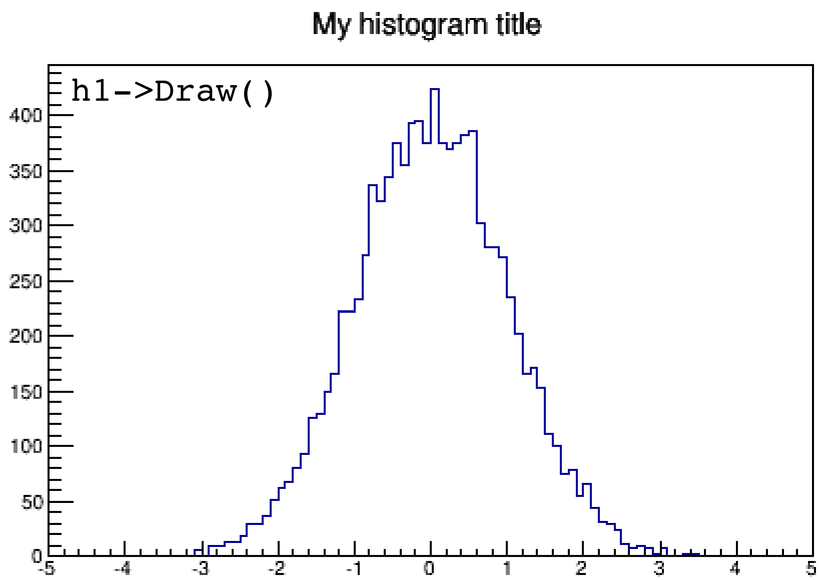
Every ROOT histogram has:
overflow bin → where entries beyond the upper edge of the last bin go
Underflow bin → where entries beyond the low edge of the first bin go

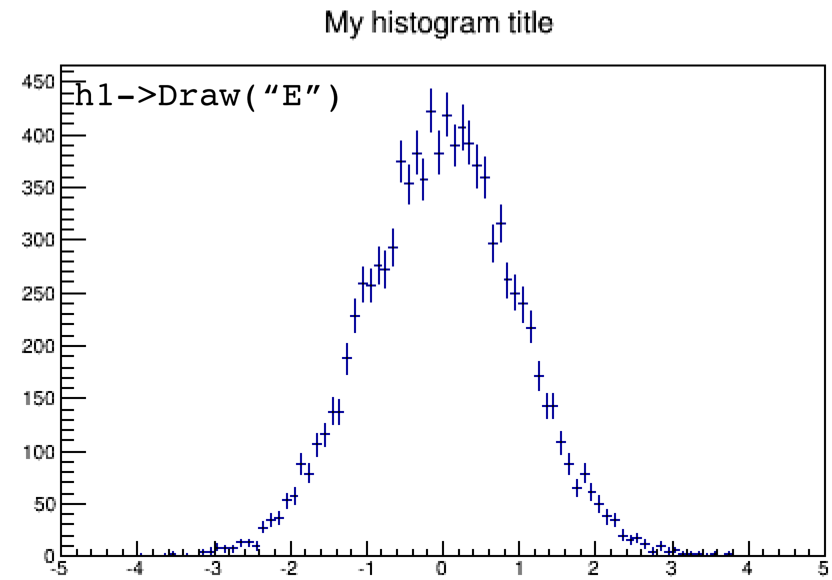
Bin numbering conventions

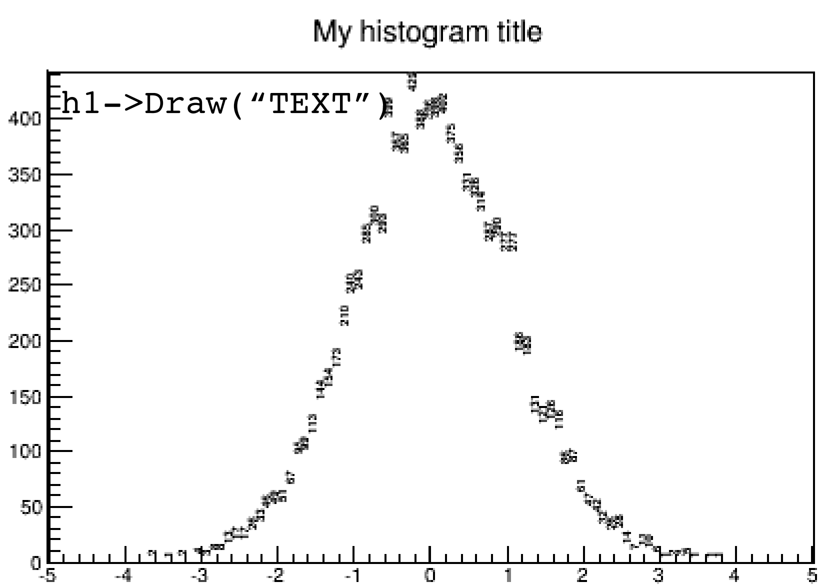
bin = 0; underflow bin
bin = 1; first bin with low-edge included
bin = nbins; last bin with upper-edge excluded
bin = nbins+1; overflow bin

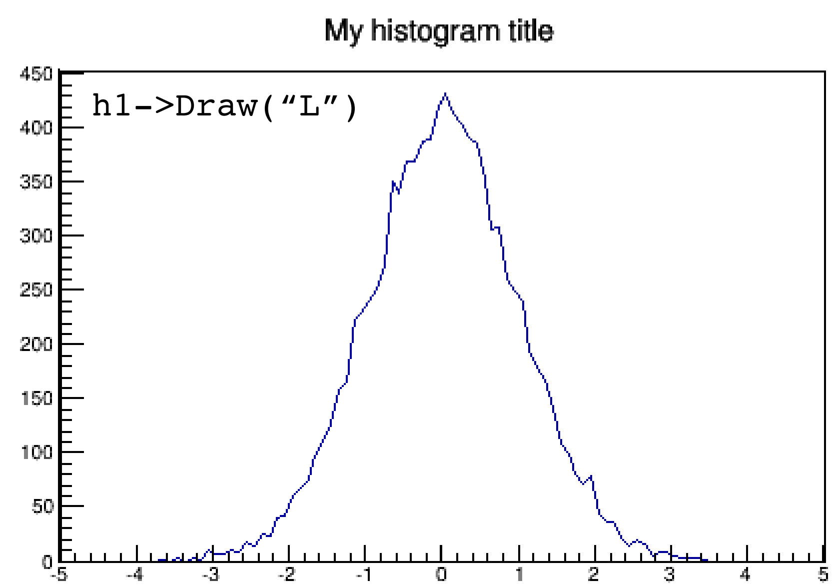
Drawing histograms

Many options to **draw** a histogram (see THistPainter)



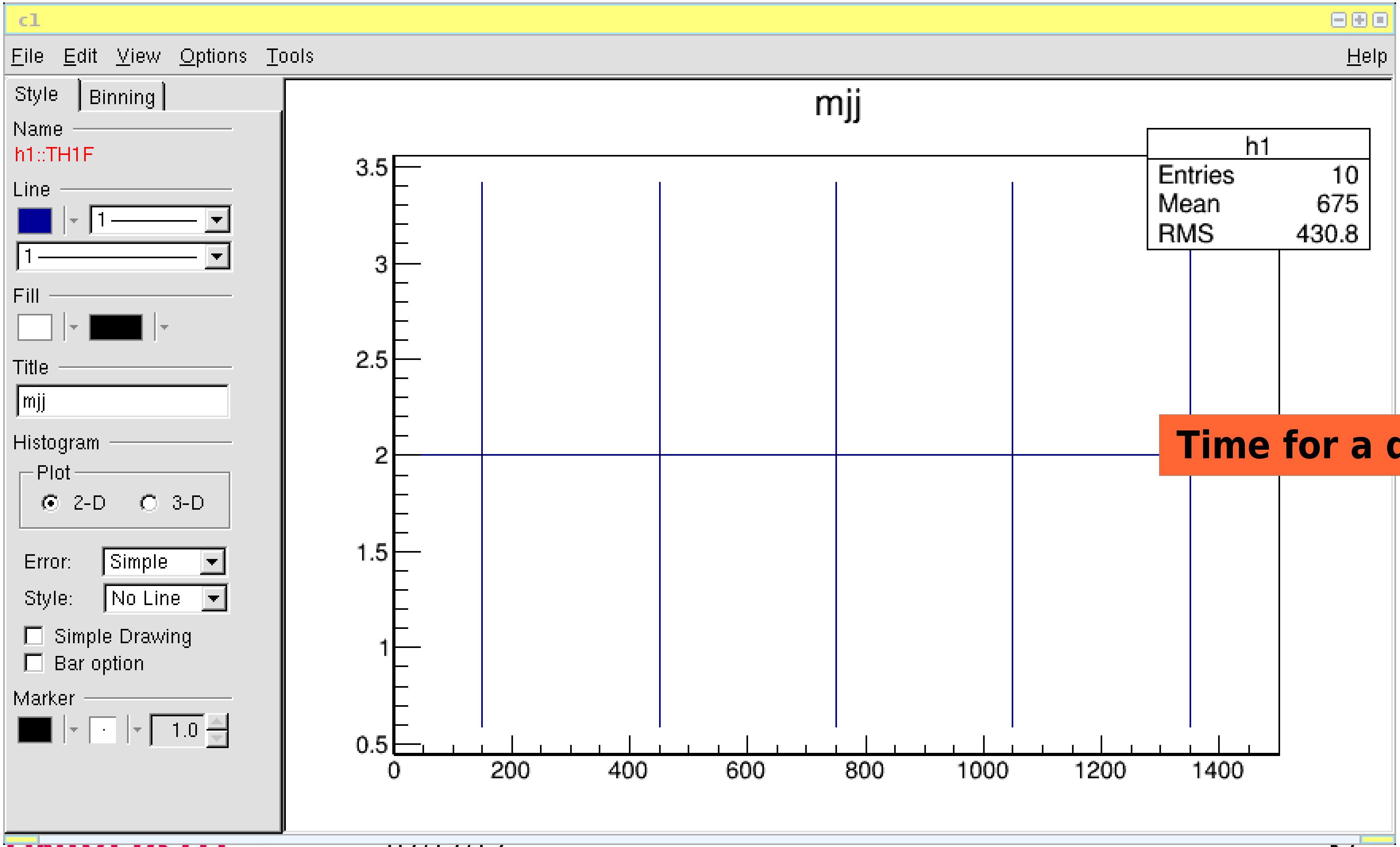






The TBrowser editor

Let's click our way through **editing a histogram**...



Time for a demo

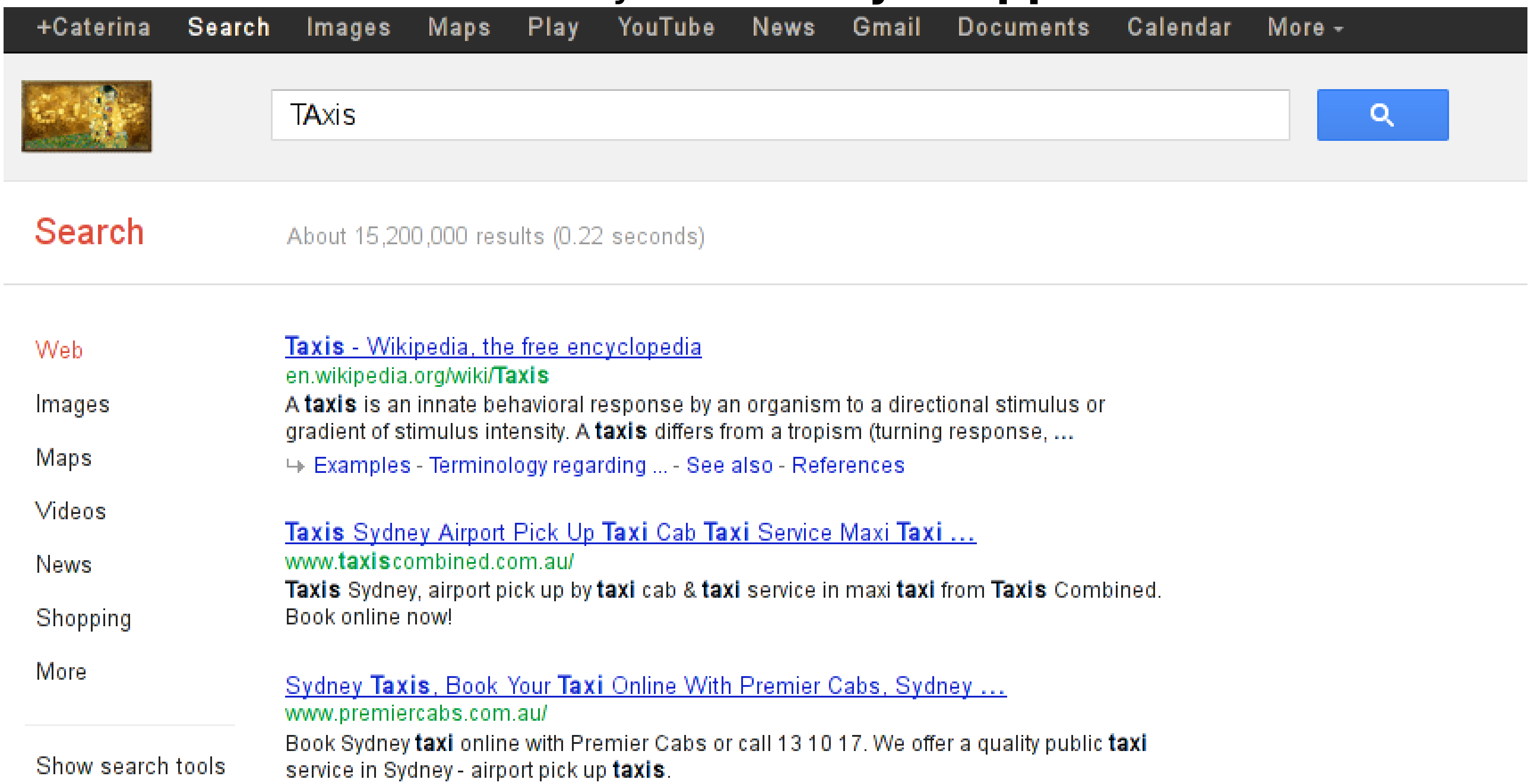
UNIVERSITÉ DE GENÈVE

ATLAS

Don't forget the axis labels (1)

TAxis: class controlling x and y axes

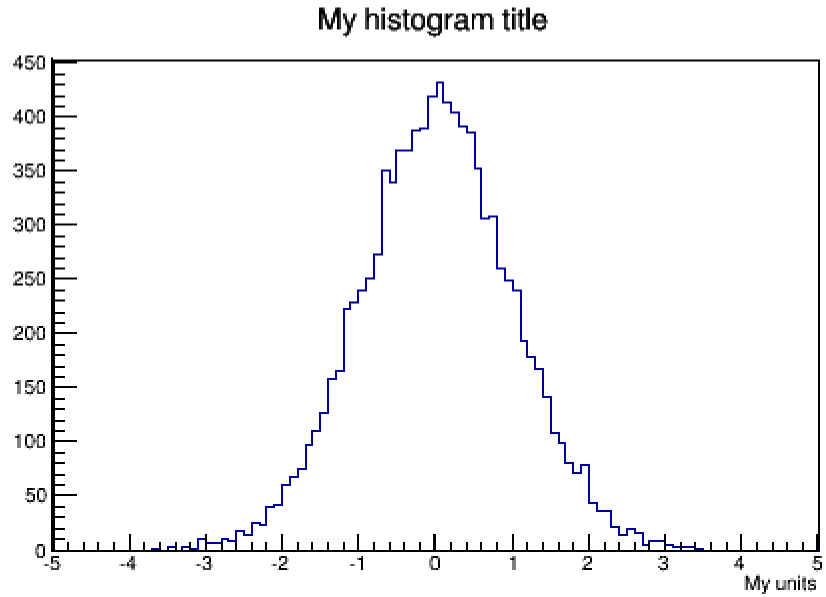
Incidentally, this **always happens**



Don't forget the axis labels (2)

TAxis: class controlling x and y axes

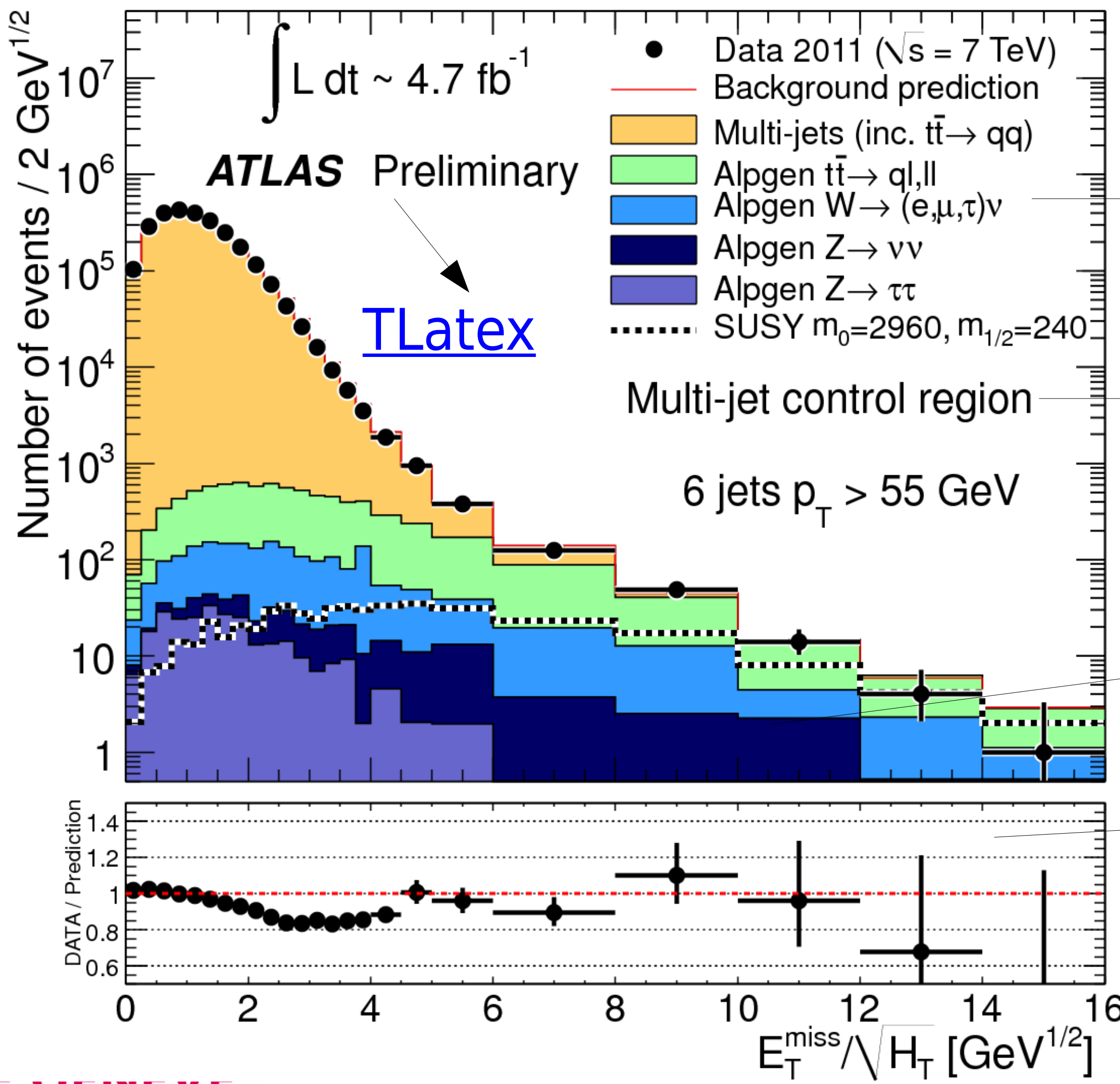
```
root [13] TAxis * xAxis = myHistogram->GetXaxis()
root [14] xAxis->SetTitle("My units")
```



Axis title

Many 1-dimensional histograms (1)

How to plot **many histograms** at once?



TLatex

TLegend

TPaveText

THStack

TPad

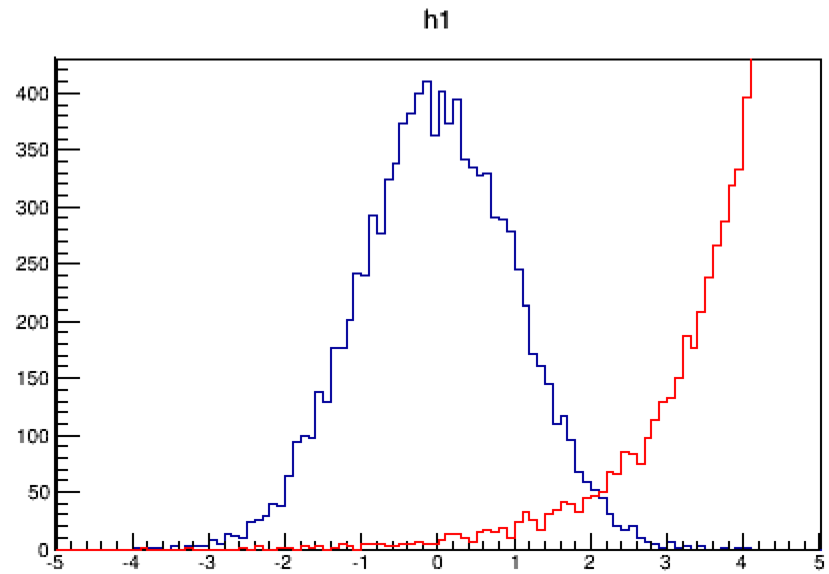
Many 1-dimensional histograms (2)

How to plot many histograms at once, the **easy** way

```
{

TH1F *h1= new TH1F("h1", "h1", 100, -5,5);
h1->FillRandom("gaus",10000);
h1->Draw("");

TH1F *h2= new TH1F("h2", "h2", 100, -5,5);
h2->FillRandom("expo",10000);
h2->SetLineColor(kRed);
h2->Draw("same");

}
```

TH1Draw.C

Draw histogram (or anything else) on the same canvas



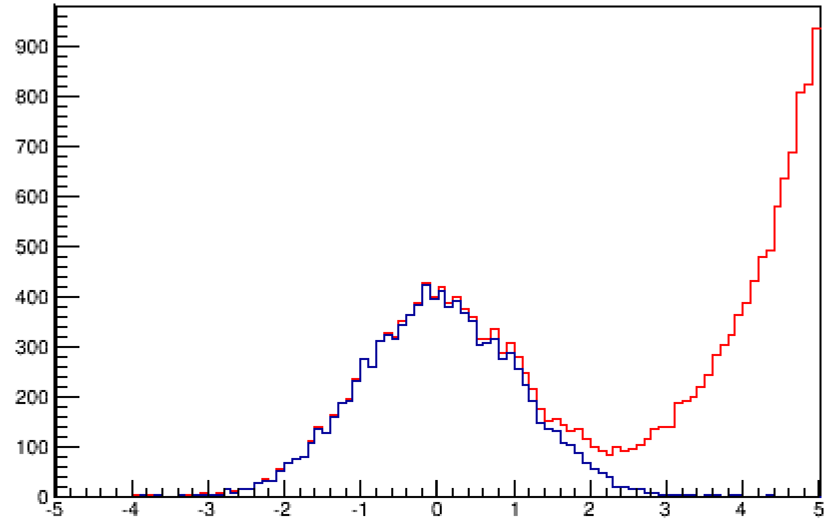
Disadvantage: any formatting of axes, title etc will be tied to first histogram

THStack (1)

How to plot many histograms at once and stack them as well

More on random number generators later...

```
TH1F *h1= new TH1F("h1", "h1", 100, -5,5);
h1->FillRandom("gaus",10000);

TH1F *h2= new TH1F("h2", "h2", 100, -5,5);
h2->FillRandom("expo",10000);
h2->SetLineColor(kRed);

THStack *hStack = new THStack();
hStack.Add(h1);
hStack.Add(h2);

hStack.Draw();
```



Stacked histograms:
Total bin content displayed
= sum of bin contents
of individual histograms

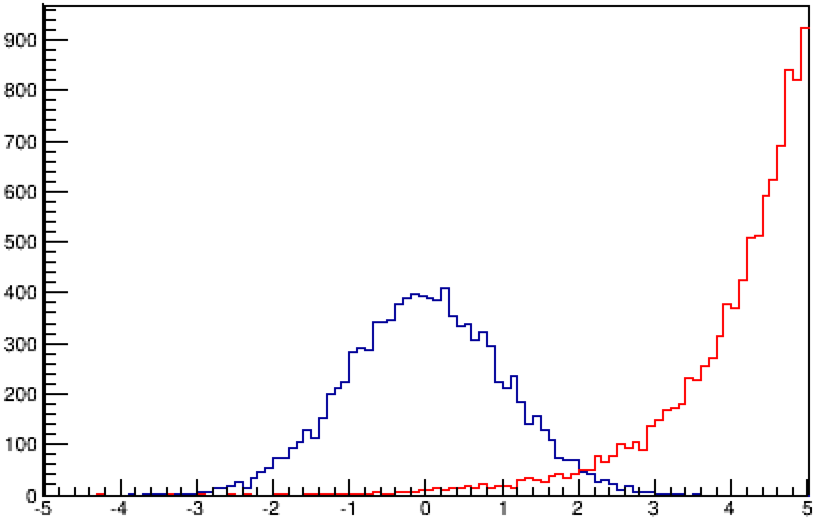
THStack (2)

How to plot many histograms at once and **stack** them as well

```
TH1F *h1= new TH1F("h1", "h1", 100, -5,5);
h1->FillRandom("gaus",10000);

TH1F *h2= new TH1F("h2", "h2", 100, -5,5);
h2->FillRandom("expo",10000);
h2->SetLineColor(kRed);

THStack *hStack = new THStack();
hStack.Add(h1);
hStack.Add(h2);

hStack.Draw("A");
hStack.GetXaxis().SetTitle("my units");
hStack.Draw("nostack");
```

Needed to 'create' the axis

TH1Stack.C



nostack option:
Equivalent to drawing with "same"
Advantage: control global drawing properties (axes etc) using THStack only

TPad

```
//___________________________________________________________________________
//  The Pad class is the most important graphics class in the ROOT system.
```

How to have e.g. a data/MC inset on the bottom of your plot

TPad: contained in a TCanvas, can contain other TPads

```
////**Making the pads

//Set the coordinates of the current pad
//xLow, yLow, xHigh, yHigh
pad1 = new TPad("pad1","pad1",0.05,0.30,1,1);
pad2 = new TPad("pad2","pad2",0.05,0.05,1,0.30);
pad1->SetTopMargin(0.02);
pad1->SetLogy();
pad2->SetTopMargin(0.0);
pad1->SetBottomMargin(0.0);
pad2->SetBottomMargin(0.20);
pad1->Draw();
pad2->Draw();
pad1->cd();

//now draw the histograms
stack.Draw("nostack");
//Update() is used to make the
//canvas realise something has happened
cv->Update();
pad2->cd();
ratio->Draw();
cv->Update();
```

Parameters: xLow, yLow, xHigh, yHigh
Coordinates are relative to the canvas: (x,y)=(0,0) is bottom left

If plots share the same x axis, cover axis for first plot

From now on, everything will be Draw()n on pad1

From now on, everything will be Draw()n on pad2

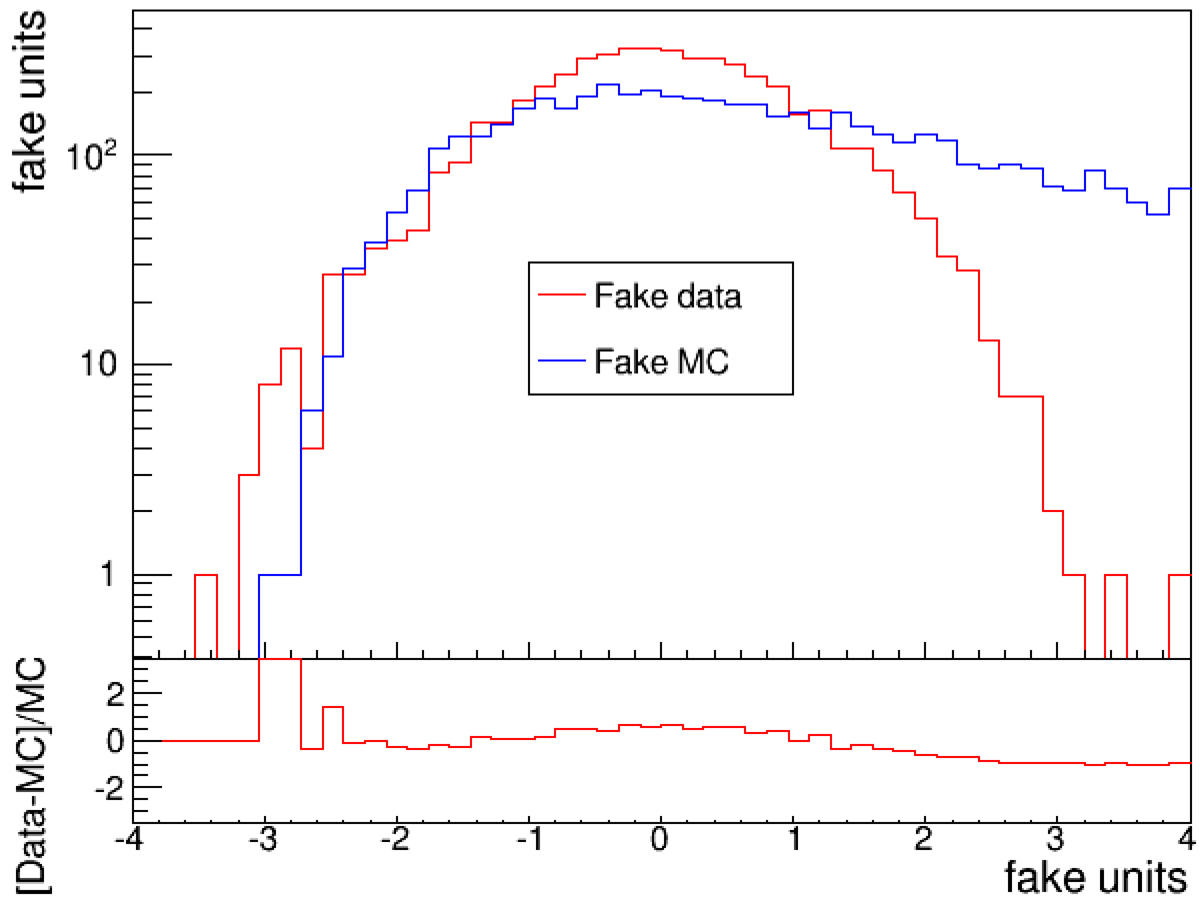
TPadExample.C

TPad

```
//___________________________________________________________________________
//  The Pad class is the most important graphics class in the ROOT system.
```

How to have e.g. a data/MC inset on the bottom of your plot

Final result (with some more formatting + a TLegend needed...)

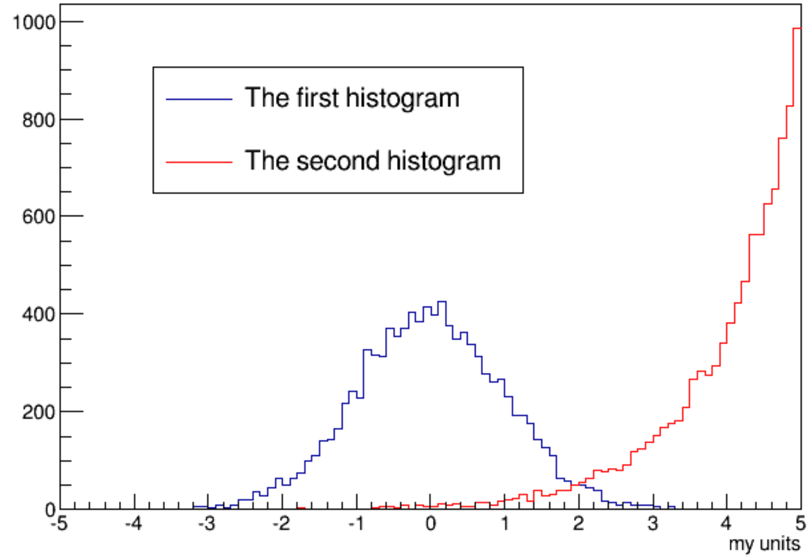


TLegend

How to draw a **legend** for multiple histograms

```
//constructor takes normalized coordinates within the pad
//with x=0, y=0 being the bottom left corner
TLegend *l = new TLegend(0.2, 0.6, 0.6, 0.8);
//let's make the legend background white
l.SetFillColor(kWhite);
//arguments: pointer to histogram, text, options: draw line (L)
l.AddEntry(h1, "The first histogram", "L");
l.AddEntry(h2, "The second histogram", "L");
l.Draw("same");
```

TH1Stack.C



2-dimensional histograms

2-D histogram can be instantiated in a similar way as 1-D ones, with one dimension more (there are also 3D histograms...)

With fixed bin size

TH2 (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup, Int_t nbinsy, Double_t ylow, Double_t yup)

```
root [0] TH2D * h2 = new TH2D("h2", "h2", 100, 0, 100, 200, 0, 200)
```

With variable bin size

TH2 (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins, Int_t nbinsy, const Double_t* ybins)

C arrays with low edges for each bin + high edge of last bin

```
root [2] Double_t binsX[4] = {1,2,4,6}
root [3] Double_t binsY[4] = {10,20,40,60}
root [4] TH2D * h2_varBinSize = new TH2D("h2_varBinSize", "h2_varBinSize", 3, binsX, 3, binsY)
```

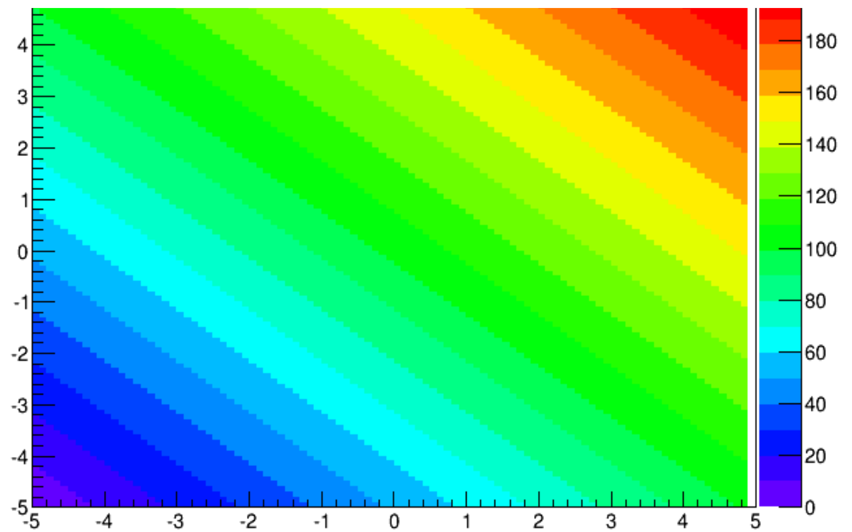
The number of bins is equal to the number of elements in the vector of bins **minus one**

2-dimensional histograms

Filling a 2-D histogram

```
TH2D * h2 = new TH2D("h2", "h2", 1000, -5, 5, 1000, -5, 5);
//avoid underflow and overflow
for (unsigned int iBinX = 1; iBinX < h2.GetNbinsX()+1; iBinX++) {
  for (unsigned int iBinY = 1; iBinY < h2.GetNbinsY()+1; iBinY++) {
    //same syntax as TH1s, with one dimension more
    h2->SetBinContent(iBinX, iBinY, iBinX+iBinY);
  }
}
```

TH2Basic.C



2-dimensional histograms

Getting information from a 2-D histogram

```
//finding the identifier of a bin
cout << "In the TH2 bin numbering scheme"
     << "x=4.5, y=4.5 is located in bin: "
     <<  h2->FindBin(4.5,4.5)
     << endl;

//this is particularly useful for 2D histograms
//as the function to find the bin content uses this
cout << "The bin content for the "
     << "x=4.5, y=4.5 bin is: "
     <<  h2->GetBinContent(h2->FindBin(4.5,4.5))
     << endl;
```

TH2Basic.C

```
root [0] .x TH2Basic.C
In the TH2 bin numbering schemex=4.5, y=4.5 is located in bin: 953853
The bin content for the x=4.5, y=4.5 bin is: 1902
```

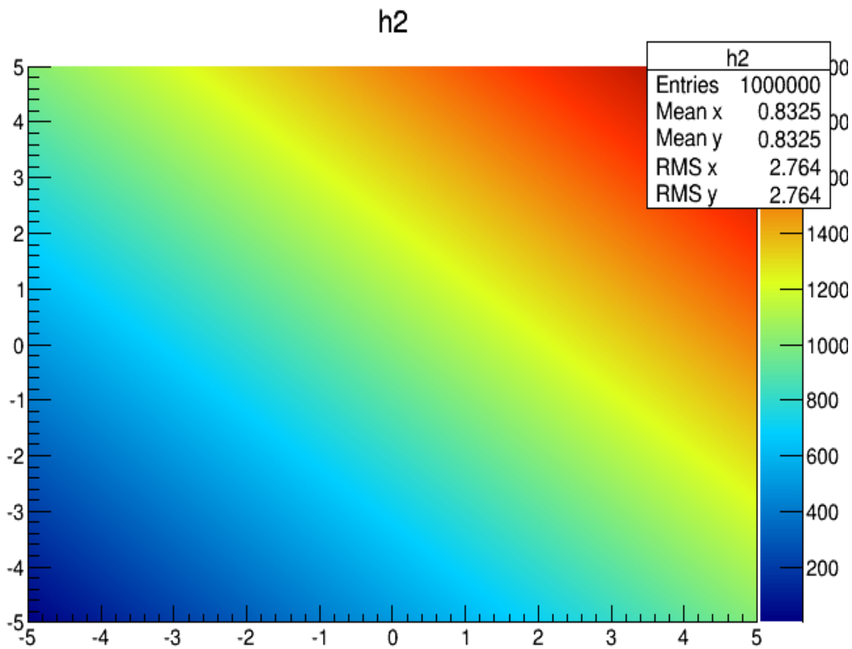
Pretty 2-dimensional histograms

How to set a new **palette** (credits to this website)

```
void set_plot_style() {
    const Int_t NRGBs = 5;
    const Int_t NCont = 255;
    Double_t stops[NRGBs] = { 0.00, 0.34, 0.61, 0.84, 1.00 };
    Double_t red[NRGBs]   = { 0.00, 0.00, 0.87, 1.00, 0.51 };
    Double_t green[NRGBs] = { 0.00, 0.81, 1.00, 0.20, 0.00 };
    Double_t blue[NRGBs]  = { 0.51, 1.00, 0.12, 0.00, 0.00 };
    TColor::CreateGradientColorTable(NRGBs, stops, red, green, blue, NCont);
    gStyle->SetNumberContours(NCont);
}
```

SetPlotStyle.C

```
root [0] .L SetPlotStyle.C
root [1] set_plot_style()
root [2] .x TH2Basic.C
```



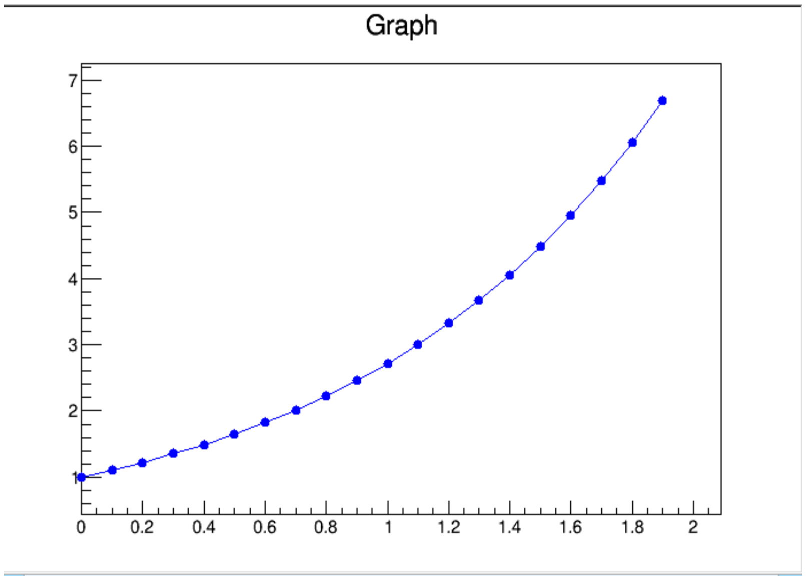
Graphs with errors

ROOT Tutorial
HASCO school – 17/07/2012

TGraph

TGraph: two arrays of points representing x and y coordinates
TGraphErrors: TGraph with symmetric errors on x and y points
TGraphAsymmErrors: TgraphErrors, with asymmetric errors

```
{
   Double_t x[100], y[100];
   Int_t n = 20;
   for (Int_t i=0;i<n;i++) {
     x[i] = i*0.1;
     y[i] = exp(x[i]);
   }
   TGraph * g = new TGraph(n,x,y);
   //set marker style and size
   g->SetMarkerStyle(kFullCircle);
   g->SetMarkerSize(1.0);
   g->SetMarkerColor(kBlue);
   g->SetLineColor(kBlue);
   //in TGraph, need to draw Axis (A)
   //want to draw markers (P) and line (L)
   g->Draw("APL");
}
```

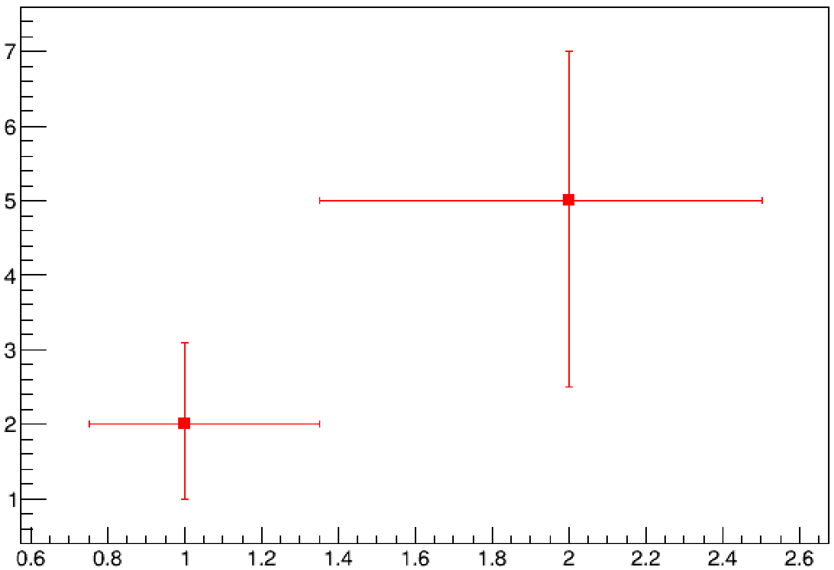


TGraph.C

TGraphAsymmErrors

TGraph: two arrays of points representing x and y coordinates
TGraphErrors: TGraph with symmetric errors on x and y points
TGraphAsymmErrors: TGraphErrors, with asymmetric errors

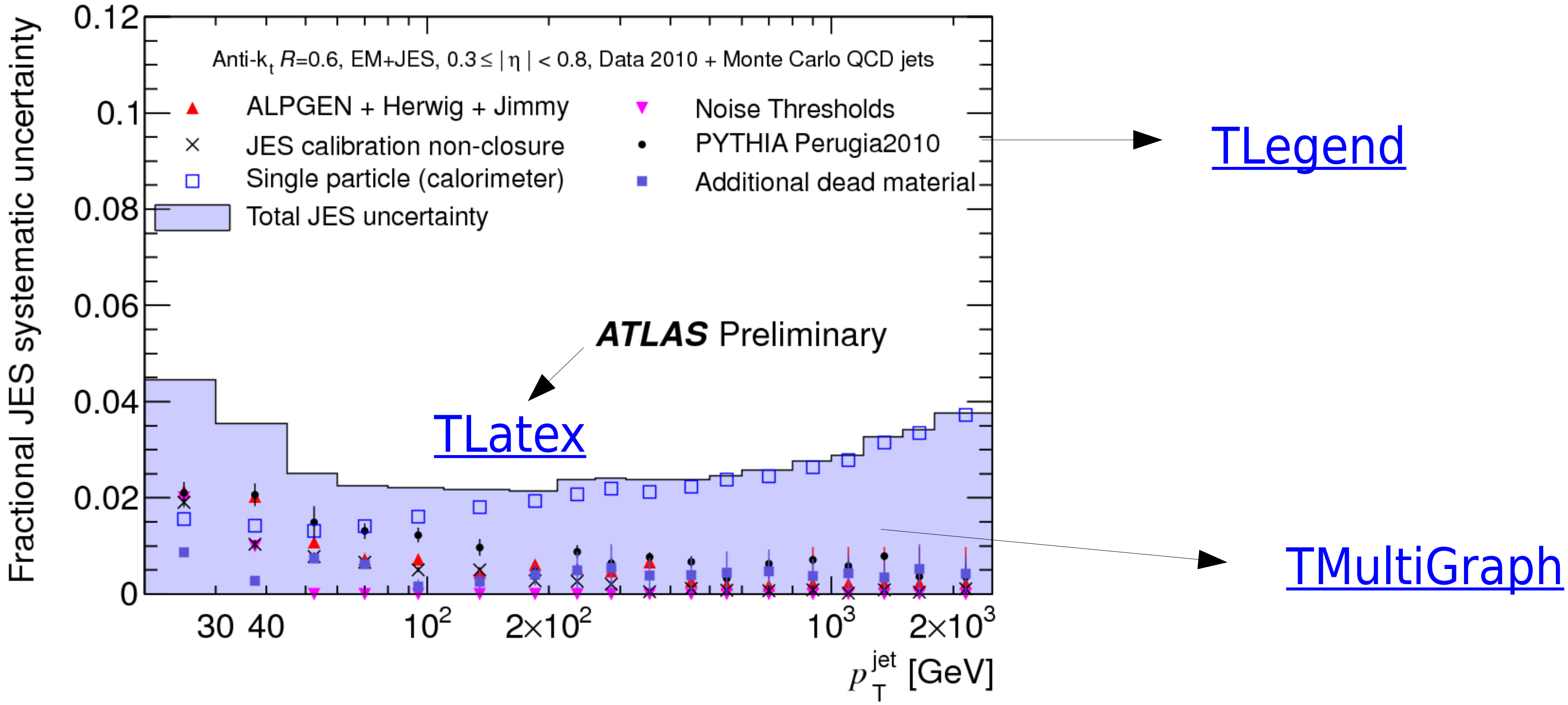
```
{
   TGraphAsymmErrors * g = new TGraphAsymmErrors();
   //set a couple of points - index starts from 0
   //parameters: point index, x coordinate, y coordinate
   g->SetPoint(0, 1.0, 2.0);
   g->SetPoint(1, 2.0, 5.0);
   //set the errors
   //parameters: point index,
   //x err down, x err up, y err down, y err up
   g->SetPointError(0, 0.25, 0.35, 1.0, 1.1);
   g->SetPointError(1, 0.65, 0.5, 2.5, 2.0);

   g->SetMarkerStyle(kFullSquare);
   g->SetMarkerSize(1.0);
   g->SetMarkerColor(kRed);
   g->SetLineColor(kRed);
   //in TGraph, need to draw Axis (A)
   //want to draw markers (P) and line (L)
   g->Draw("AP");
}
```



TGraphAsymmErrors.C

Many TGraphs

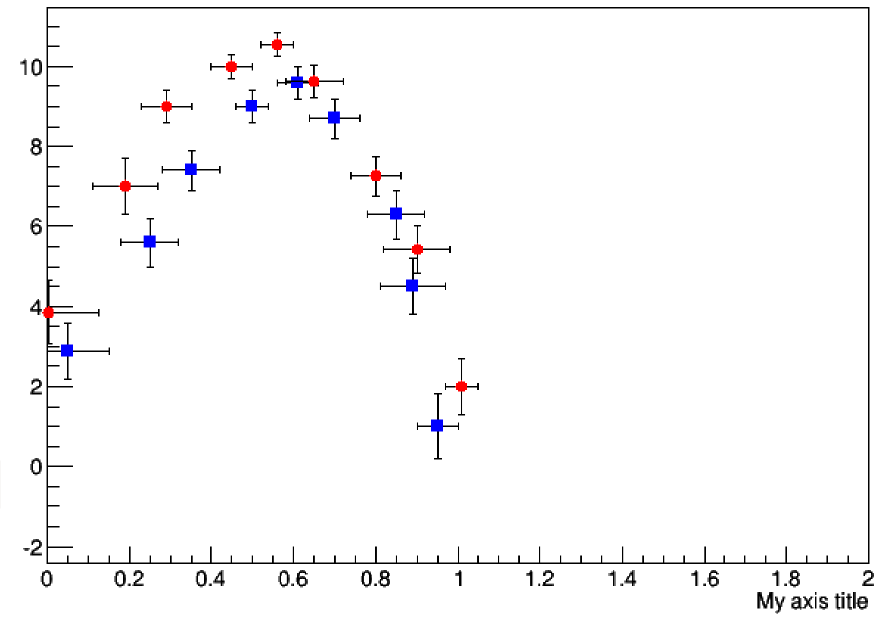
How to plot **many graphs** at once?



TMultigraph

How to plot **many graphs** at once?

```
//(snip) instantiate two graphs

//add pointers to graphs
mg->Add(gr1);
mg->Add(gr2);

//set title and range for both graphs at once
mg->Draw("A");
mg->GetXaxis()->SetLimits(0,2);
mg->GetXaxis()->SetTitle("My axis title");
mg->Draw("AP");
```



TMultiGraph.C

Data storage and more: TTrees

ROOT Tutorial
HASCO school – 17/07/2012

What is a TTree?

TTree: made for **saving (and processing) data**

Simple idea:
it's like a table with
rows = events
columns = data fields

...more complex
(more functionalities)
than this: e.g.

- TTree can contain entire objects (branches → leaves)
- TTree can perform operations on itself (scanning, dumping to histogram, cuts)

branches

entries

Run number	Event number	m_jj (GeV)
203353	87535595	2669.6731
203353	74292059	2617.4563
203353	84096111	2452.7685
203353	74541499	2450.3027
203353	87499206	2399.0742

UNIVERSITÉ DE GENÈVE

ATLAS

Preparing a TTree

Branching a TTree → creating **data fields** to save entries

```
//construct the TTree
TTree * t = new TTree("myFirstTree", "myFirstTree");

//have some variables that will be read from the TTree
int runNumber = 0, eventNumber = 0;
double mjj = 0;

//let's branch the TTree
//arguments: branch name, address of variable, variable name and type
//see http://root.cern.ch/root/html/TTree.html#TTree:Branch
t->Branch("runNumber",&runNumber,"runNumber/I");
t->Branch("eventNumber",&eventNumber,"eventNumber/I");
t->Branch("mjj",&mjj,"mjj/D");

//see what the TTree contains
t->Print();
```

This will associate the variables to the tree so it will read from the right locations in memory

TTreeBasic.C

```
root [23] .x TTreeBasic.C
******************************************************************************
*Tree    :myFirstTree: myFirstTree                                            *
*Entries :        0 : Total =            1777 bytes  File  Size =          0 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :runNumber : runNumber/I                                            *
*Entries :        0 : Total  Size=        490 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    1 :eventNumber : eventNumber/I                                        *
*Entries :        0 : Total  Size=        498 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    2 :mjj       : mjj/D                                                  *
*Entries :        0 : Total  Size=        474 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
```

Filling a TTree

Filling a TTree → **inserting entries** in data fields

```
//now let's loop on some toy events
for (unsigned int iEvent = 0; iEvent<10; iEvent++) {
  runNumber = 150000;
  eventNumber = iEvent;
  //fictitious dijet mass...
  mjj = double(runNumber*iEvent)/1000. ;

  //let the TTree pick up the variables for each event
  t->Fill();
}
```

TTreeBasic.C

```
//see what the TTree contains
t->Print();
```

```
******************************************************************************
*Tree    :myFirstTree: myFirstTree                                            *
*Entries :       10 : Total =            2459 bytes  File  Size =          0 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :runNumber : runNumber/I                                            *
*Entries :       10 : Total  Size=        712 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    1 :eventNumber : eventNumber/I                                        *
*Entries :       10 : Total  Size=        724 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    2 :mjj       : mjj/D                                                  *
*Entries :       10 : Total  Size=        724 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
```

See the TTree class doc for more ways to fill a TTree...

Reading a TTree: Scan

Simple by-eye **inspection** of TTree entries

Ttree::Scan()

Without any arguments, Scan() will display all entries and all branches sequentially

```
root [3] myFirstTree->Scan()
************************************************
*    Row   * runNumber * eventNumb *       mjj *
************************************************
*        0 *    150000 *         0 *         0 *
*        1 *    150000 *         1 *       150 *
*        2 *    150000 *         2 *       300 *
*        3 *    150000 *         3 *       450 *
*        4 *    150000 *         4 *       600 *
*        5 *    150000 *         5 *       750 *
*        6 *    150000 *         6 *       900 *
*        7 *    150000 *         7 *      1050 *
*        8 *    150000 *         8 *      1200 *
*        9 *    150000 *         9 *      1350 *
************************************************
```

Reading a TTree: Scan

Simple by-eye **inspection** of TTree entries

Ttree::Scan("branchName")

You can Scan() single / multiple branches (first argument of the function needs to be the branch name)

```
root [4] myFirstTree->Scan("mjj")
************************
*    Row   *       mjj *
************************
*        0 *         0 *
*        1 *       150 *
*        2 *       300 *
*        3 *       450 *
*        4 *       600 *
*        5 *       750 *
*        6 *       900 *
*        7 *      1050 *
*        8 *      1200 *
*        9 *      1350 *
************************
```

Cuts on a TTree with TTree::Scan

Simple by-eye inspection of TTree entries + apply **cuts**

Ttree::Scan(“”,”branchName>cut”)

You can apply cuts using Scan() and the syntax of TFormulas

e.g.

```
[0]*sin(x) +
[1]*exp(-[2]*x)

2*pi*sqrt(x/y)
```

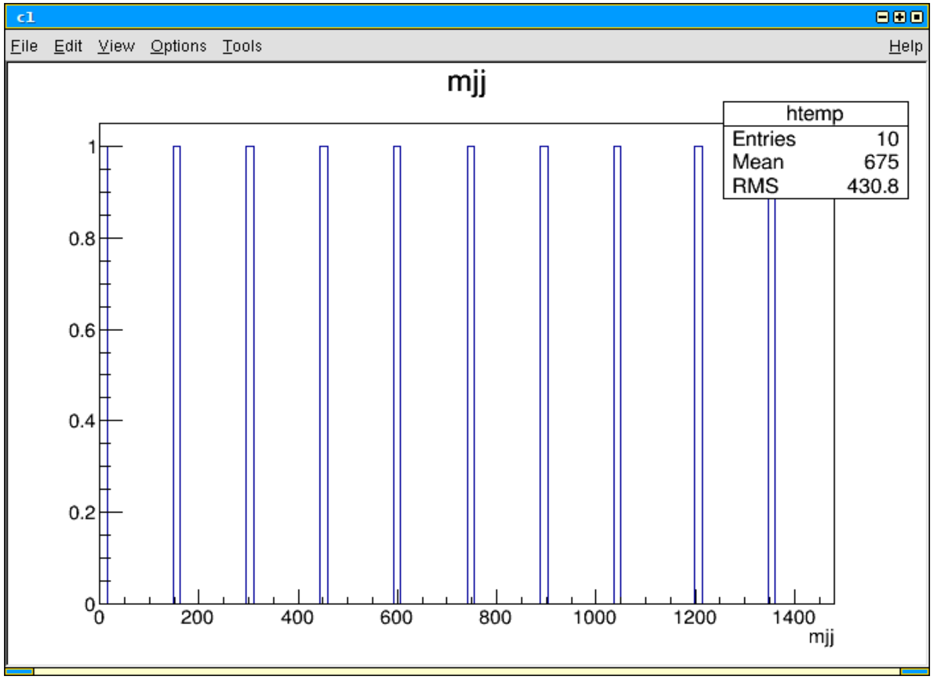
```
root [5] myFirstTree->Scan("","mjj>1000")
************************************************
*    Row   * runNumber * eventNumb *       mjj *
************************************************
*        7 *    150000 *         7 *      1050 *
*        8 *    150000 *         8 *      1200 *
*        9 *    150000 *         9 *      1350 *
************************************************
```

Drawing a TTree

TTree branches can easily be **drawn** on 1D histograms

TTree::Draw(“branchName”, “cuts”,””, “histogram painting options”)

```
root [16] myFirstTree->Draw("mjj")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```



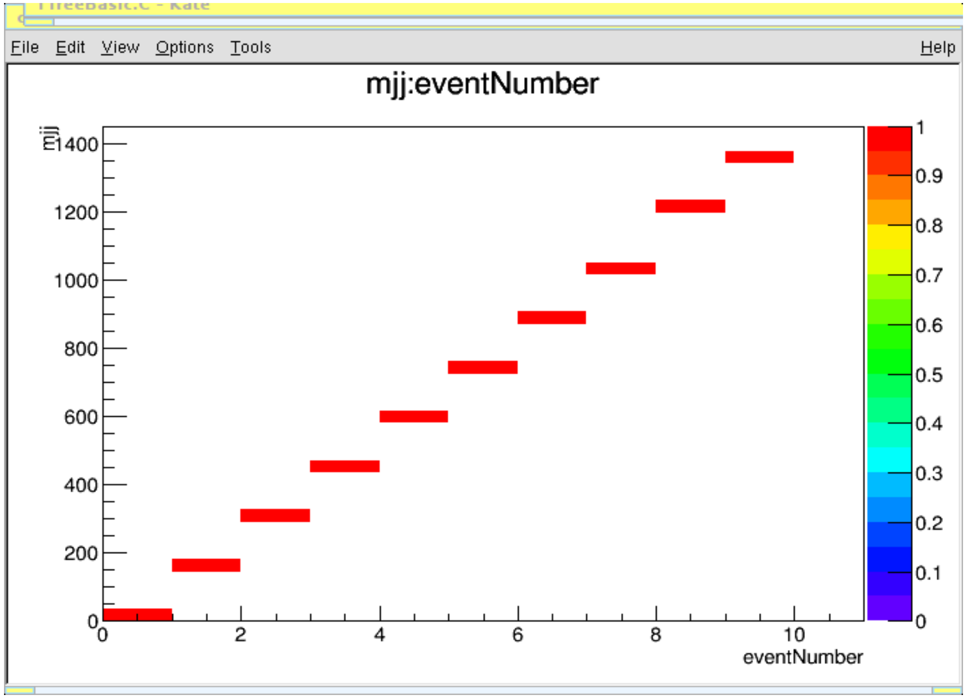
...not very physical...

Drawing a TTree

TTree branches can easily be **drawn** on 2D (or 3D) histograms

TTree::Draw(“branchName1:branchName2”, “...”)

```
root [18] myFirstTree->Draw("mjj:eventNumber","", "COLZ")
```



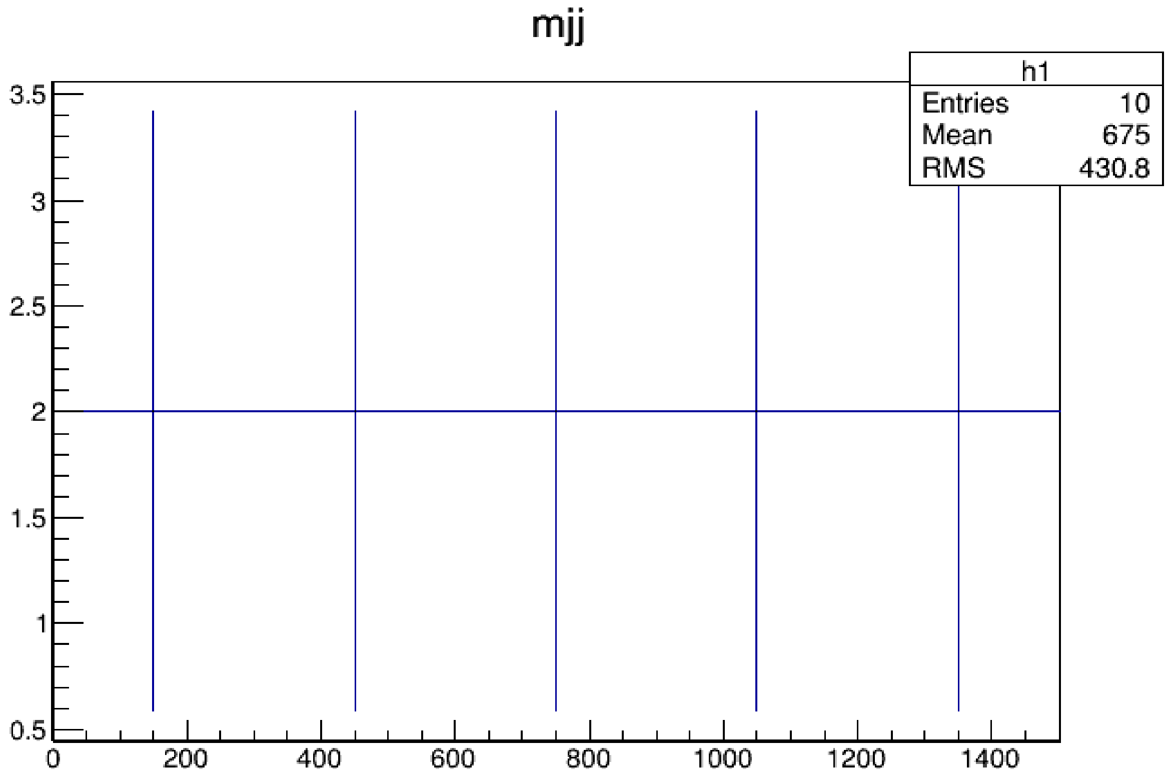
...still not very physical...

Drawing a TTree

The result of Draw() can be **saved** on a custom histogram

TTree::Draw(“branchName”, “branchName>h1(TH1 nBinsX, xLow, xHigh)”)

```
root [29] myFirstTree->Draw("mjj>>h1(5,0,1500)", "", "COLZ")
(Long64_t)10
root [30] h1->Draw("E")
```

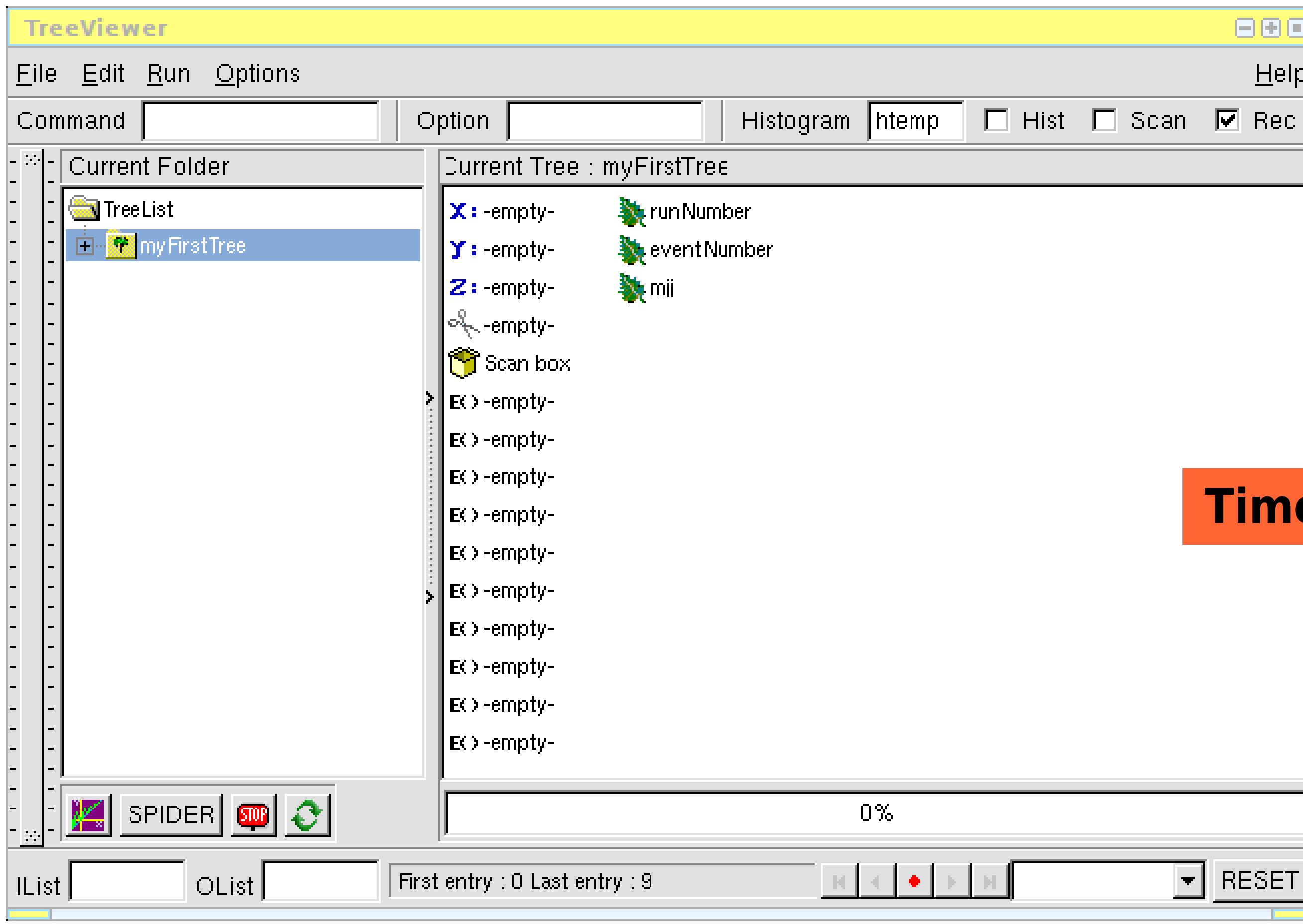


UNIVERSITÉ DE GENÈVE

ATLAS

Inspecting TTree with TTreeViewer

Scan(), Draw() and more by **clicking** on **branches**



Time for a demo

TChains

A TChain is a TTree (inheritance...) - advantage: split over many files

...after having generated two separate large TTrees...

```
root [0] TChain * c = new TChain("myTree")
root [1] c->Print()
root [2] c->Add("ChainExample_*.root")
```

Chained TTrees must have the same branches and the same name, given to the TChain

Wildcards work to give files containing Ttrees to TChain

```
root [3] c->Print()
******************************************************************************
*Chain   :myTree    : /home/cate/Work/HASCO/ChainExample_1.root              *
******************************************************************************
******************************************************************************
*Tree    :myTree    : myTree                                                 *
*Entries :   100000 : Total =         2408222 bytes  File  Size =    2199125 *
*        :          : Tree compression factor =   1.09                       *
******************************************************************************
*Br    0 :x         : x/D                                                    *
*Entries :   100000 : Total  Size=     802626 bytes  File Size  =     733443 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   1.09     *
*............................................................................*
*Br    1 :y         : y/D                                                    *
*Entries :   100000 : Total  Size=     802626 bytes  File Size  =     732473 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   1.09     *
*............................................................................*
*Br    2 :z         : z/D                                                    *
*Entries :   100000 : Total  Size=     802626 bytes  File Size  =     732150 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   1.10     *
*............................................................................*
******************************************************************************
*Chain   :myTree    : /home/cate/Work/HASCO/ChainExample_2.root              *
******************************************************************************
******************************************************************************
*Tree    :myTree    : myTree                                                 *
*Entries :   100000 : Total =         2408222 bytes  File  Size =    2198380 *
*        :          : Tree compression factor =   1.09                       *
******************************************************************************
```

Tomorrow…

- Reading TTrees efficiently: TSelector
- Random number generation
- Fitting in ROOT and more
- pyROOT

(things will get more interesting for the experienced ones among you!)