Practical introduction to ROOT

Atommag- és Nehézionfizikai Téli Iskola 2016

Anna Julia Zsigmond (Wigner RCP)
zsigmond.anna@wigner.mta.hu

This tutorial

Introductory **presentation**

- ➔ What is ROOT? (ROOT 6 series)
- ➔ Basic functionalities

Tutorial

- ➔ Upsilon suppression in heavy-ion collisions
- ➔ Analysis of real CMS data

Goal for today to get familiar with basic functions

- ➔ Input, output
- ➔ Plotting the data
- ➔ Fitting the data

This tutorial

Not covered today

- → more complicated statistical questions
- → python
- → notebooks
- → ...

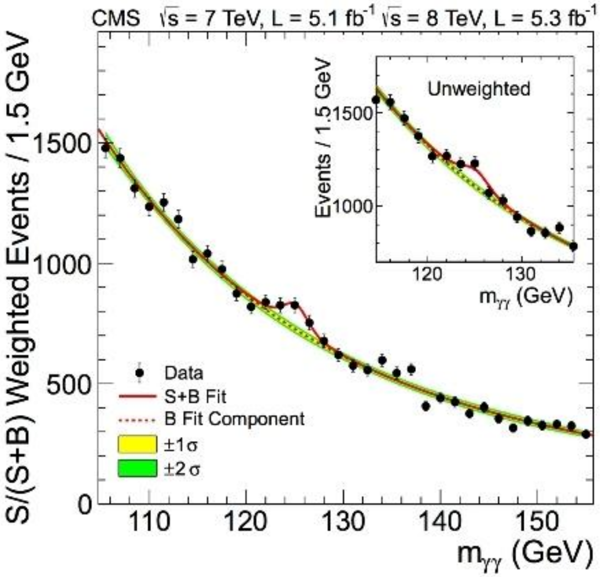
All information available on the ROOT website

https://root.cern.ch/

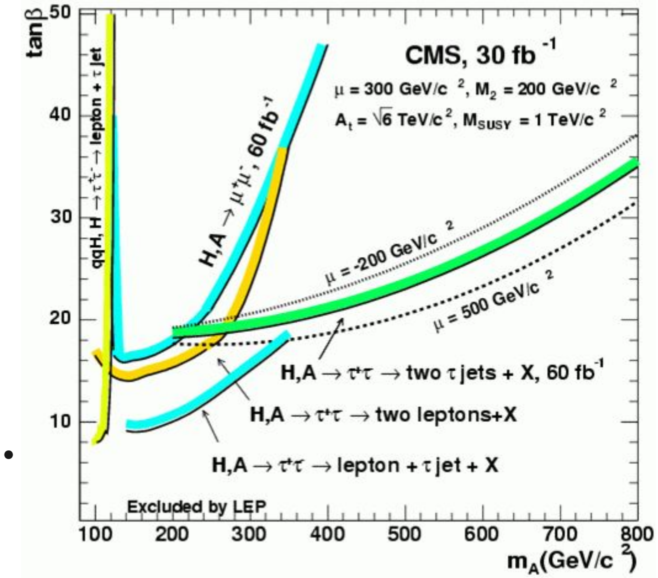
- → instructions
- → class reference
- → forum
- → ...

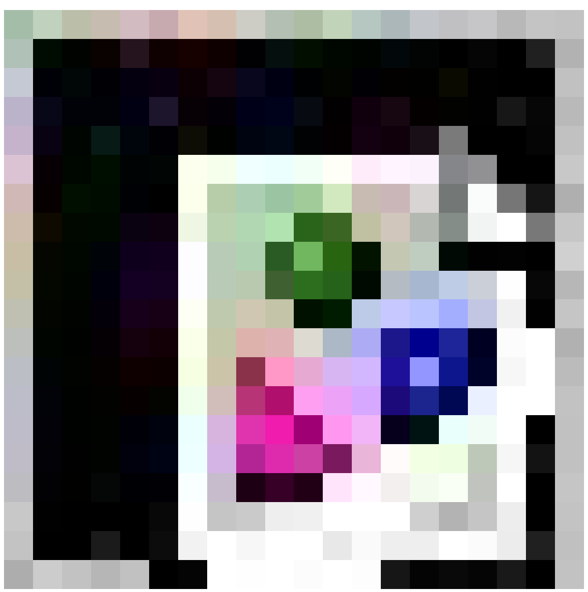
* Material shown today based on the Summer Student tutorial 2015

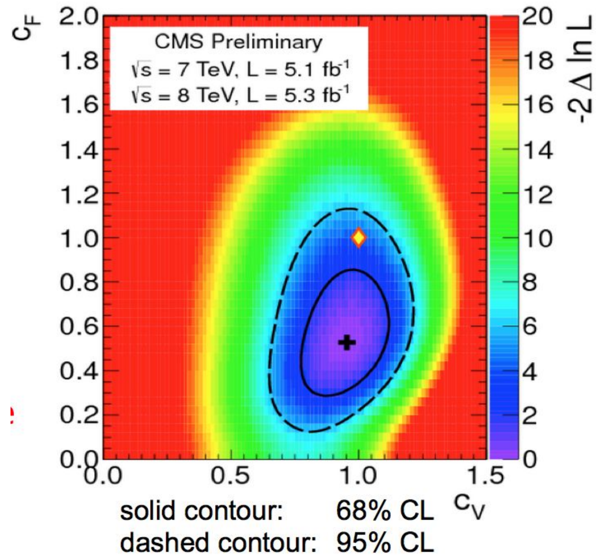
What can you do with ROOT?

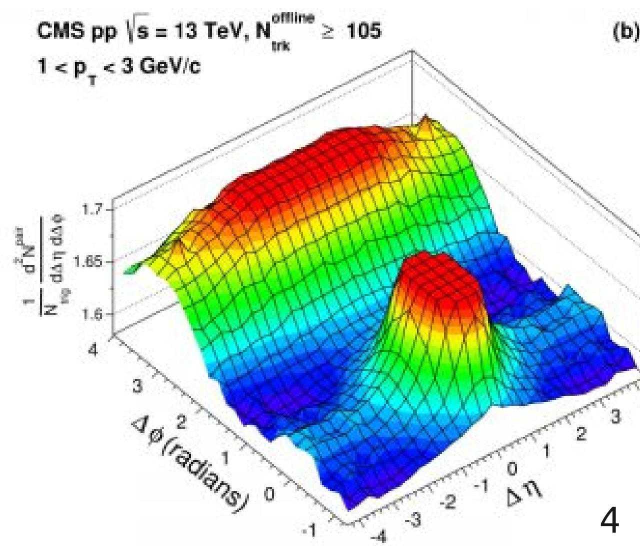


- ➔ all kinds of data visualization
- ➔ event display
- ➔ and much more…









ROOT in a nutshell

ROOT is a software toolkit providing building blocks for

- data processing
- data analysis
- data visualization
- data storage

Open Source Project

- you can also contribute

ROOT is mainly written in **C++**

Main tool in **high-energy physics** but appears also in other sciences and industry

- hundreds of PetaBytes of LHC data in ROOT format
- thousands of ROOT plots in scientific publications

ROOT in a nutshell

ROOT can be imagined as a family of building blocks for a variety of activities, for example:

- ➔ 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
- ➔ C++ interpretation: fully C++11 compliant
- ➔ Multivariate Analysis (TMVA): e.g. Boosted decision trees, neural networks
- ➔ HTTP servering, JavaScript visualisation, advanced graphics (2D, 3D, event display)
- ➔ PROOF: parallel analysis facility

Interpreter and I/O

ROOT is shipped with an interpreter, CLING

- ➔ C++ interpretation
- ➔ Just In Time (JIT) compilation
- ➔ C++ interactive shell
- ➔ Can interpret “macros” (non compiled programs)

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

- ➔ Impossible with C++ alone
- ➔ Petabytes/year of LHC data
- ➔ Method: `TObject::Write()`

You should get familiar with

- C++ syntax

```
#include <iostream>
using namespace std;
int main() {
 cout << "Hello world!" << endl;
 return 0;
}
```

Indicates content of .C or .cpp files

- basics of pointers
- basics of object oriented programming
  - class = data type and actions on it
  - data members
  - class methods
  - object = instance of a class
  - constructor

Let’s start ROOT

- If you have ROOT installed then type in the terminal
  `$ root --help`
- You will use the options frequently: `-b -n -q -l`
- Login script can be loaded at every start as defined in `.rootrc` file

Indicates terminal

```
$ root -b
   ------------------------------------------------------------
  | Welcome to ROOT 6.04/00                http://root.cern.ch |
  |                               (c) 1995-2014, The ROOT Team |
  | Built for linuxx8664gcc                                    |
  | From tag v6-04-00, 2 June 2015                             |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
   ------------------------------------------------------------

loaded
root [0] 3*3
(int) 9
```

ROOT as a calculator

```
$ root -l -n
root [0] 3+3
(int) 6
root [1] 2*(4+12)/5.
(double) 6.400000e+00
root [2] sqrt(3.)
(double) 1.732051e+00
root [3] 1 > 2
(bool) false
root [4] TMath::Pi()
(Double_t) 3.141593e+00
root [5] TMath::Sin(2.)
(Double_t) 9.092974e-01
root [6] TMath::Erf(2.)
(Double_t) 9.953223e-01
root [7] .q
```

→ ROOT interactive prompt can be used as an advanced calculator

→ C++ statements

→ Mathematical functions in the `TMath` namespace

→ Google “TMath”

https://root.cern.ch/root/html524/TMath.html

You can click on the link

ROOT interactively

- Variable declaration
- Command structures (e.g. for loop)
- Commands → type `.?`

```
$ root -l -n
root [0] const int N=5
(const int) 5
root [1] double values[N] = {2,3,5,8,13}
(double [5]) { 2.000000e+00, 3.000000e+00,
5.000000e+00, 8.000000e+00, 1.300000e+01 }
root [2] double sum = 0
(double) 0.000000e+00
root [3] for(int i=0; i<N; i++) sum += values[i]
root [4] sum
(double) 3.100000e+01
root [5] .q
```

Objects for today

- TF1 https://root.cern.ch/doc/master/classTF1.html
  - 1D function like f(x)
- TGraphErrors https://root.cern.ch/doc/master/classTGraphErrors.html
  - visualize the results of an analysis
  - points with errors in x and y direction
- TH1 https://root.cern.ch/doc/master/classTH1.html
  - base class of 1D histograms
  - fill with variable
  - fit with function
  - draw
- TTree https://root.cern.ch/doc/master/classTTree.html
  - basic structure to store your data
  - consists of TBranch objects
  - TBranch has name and class

Functions

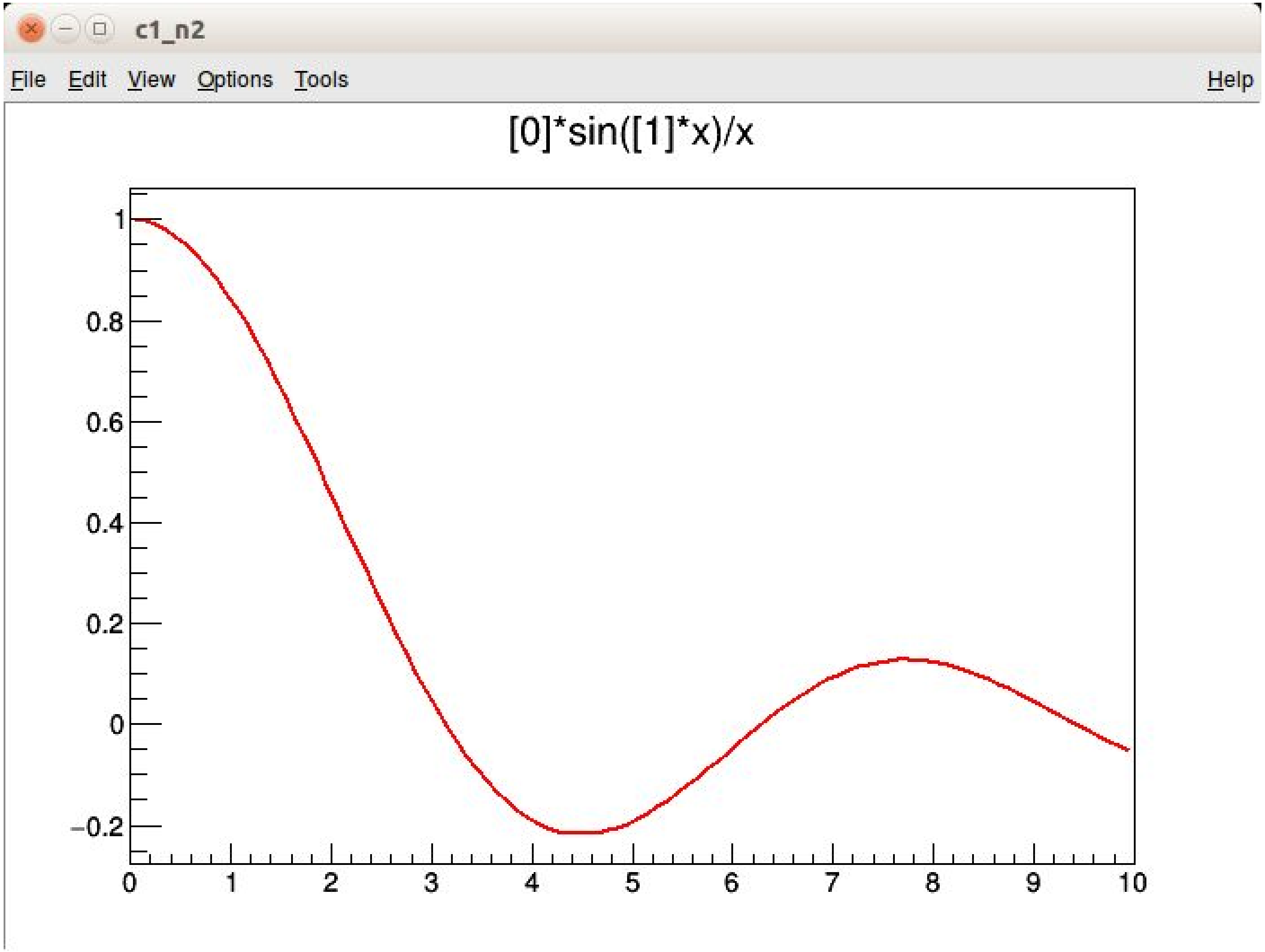
- One dimensional functions f(x) are represented by the `TF1` class
- Function has name, formula, parameters, range, …

```
$ root -l -n
root [0] TF1 f1("f1","sin(x)/x",0.,10.);
root [1] f1.Draw();
```

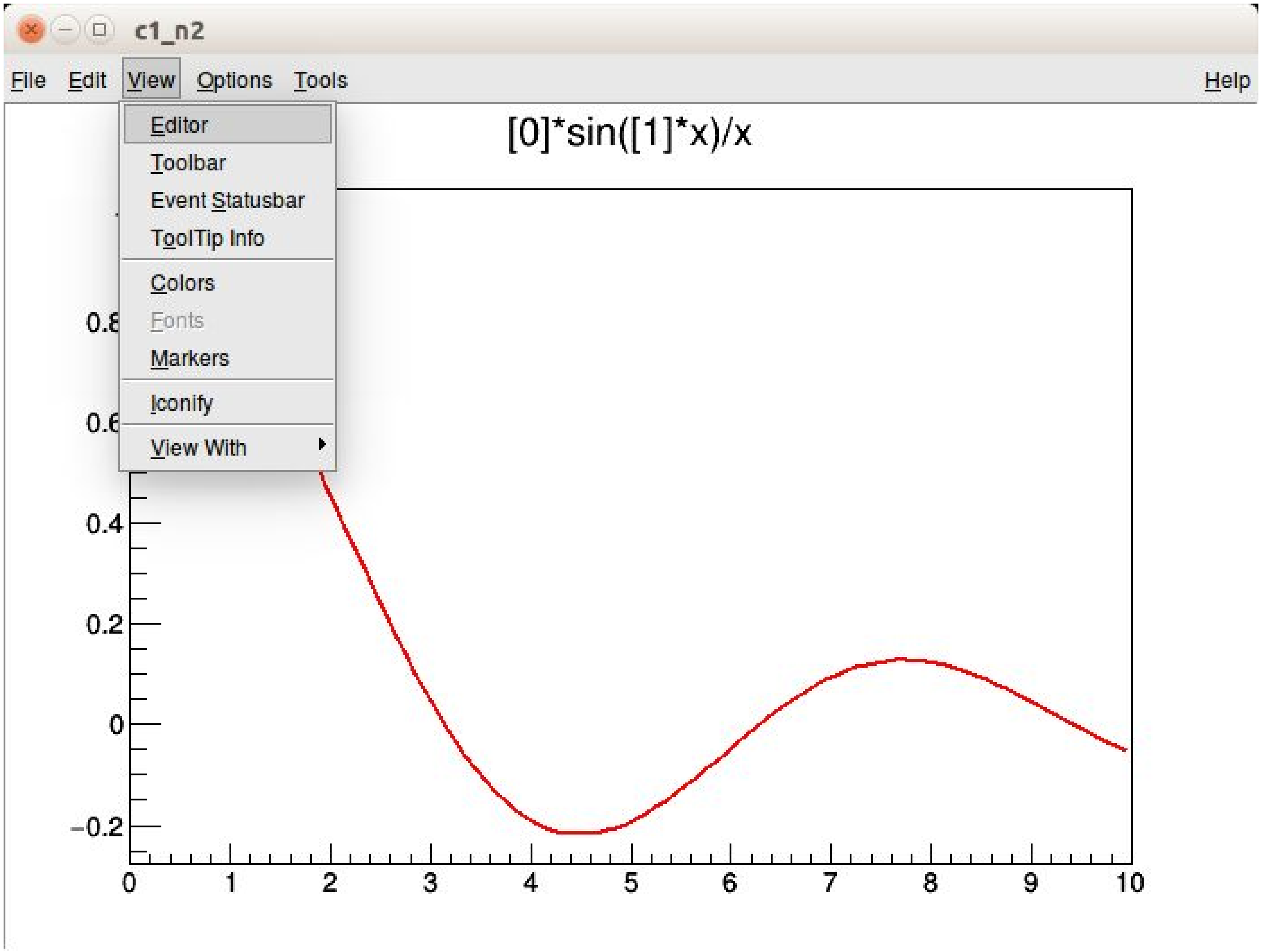
```
root [2] TCanvas *c2 = new TCanvas("c2","Title");
root [3] TF1 f2("f2","[0]*sin([1]*x)/x",0.,10.);
root [4] f2.SetParameters(1,1);
root [5] f2.Draw();
```

- You can create your own functions

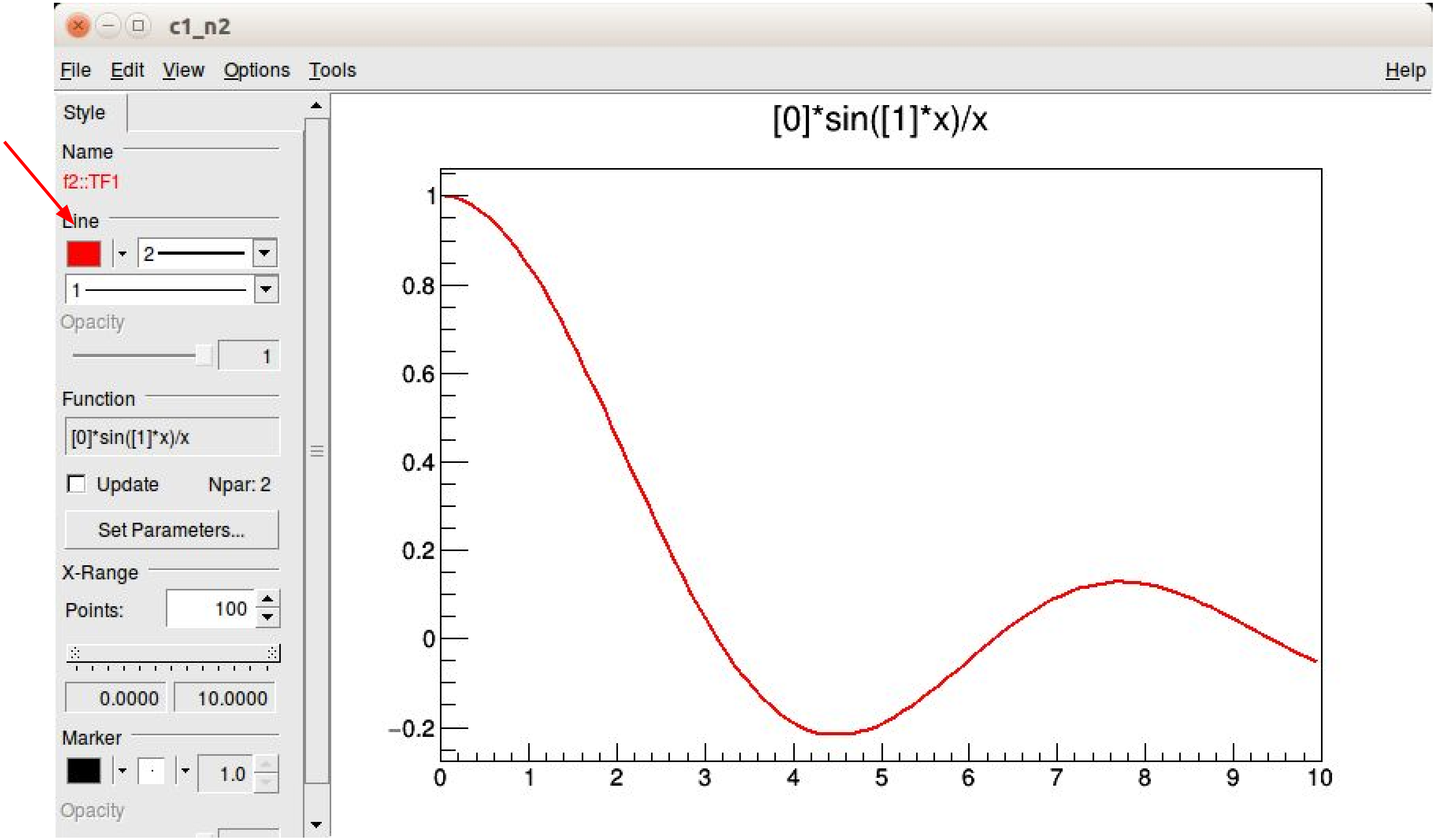
Interaction with the plot



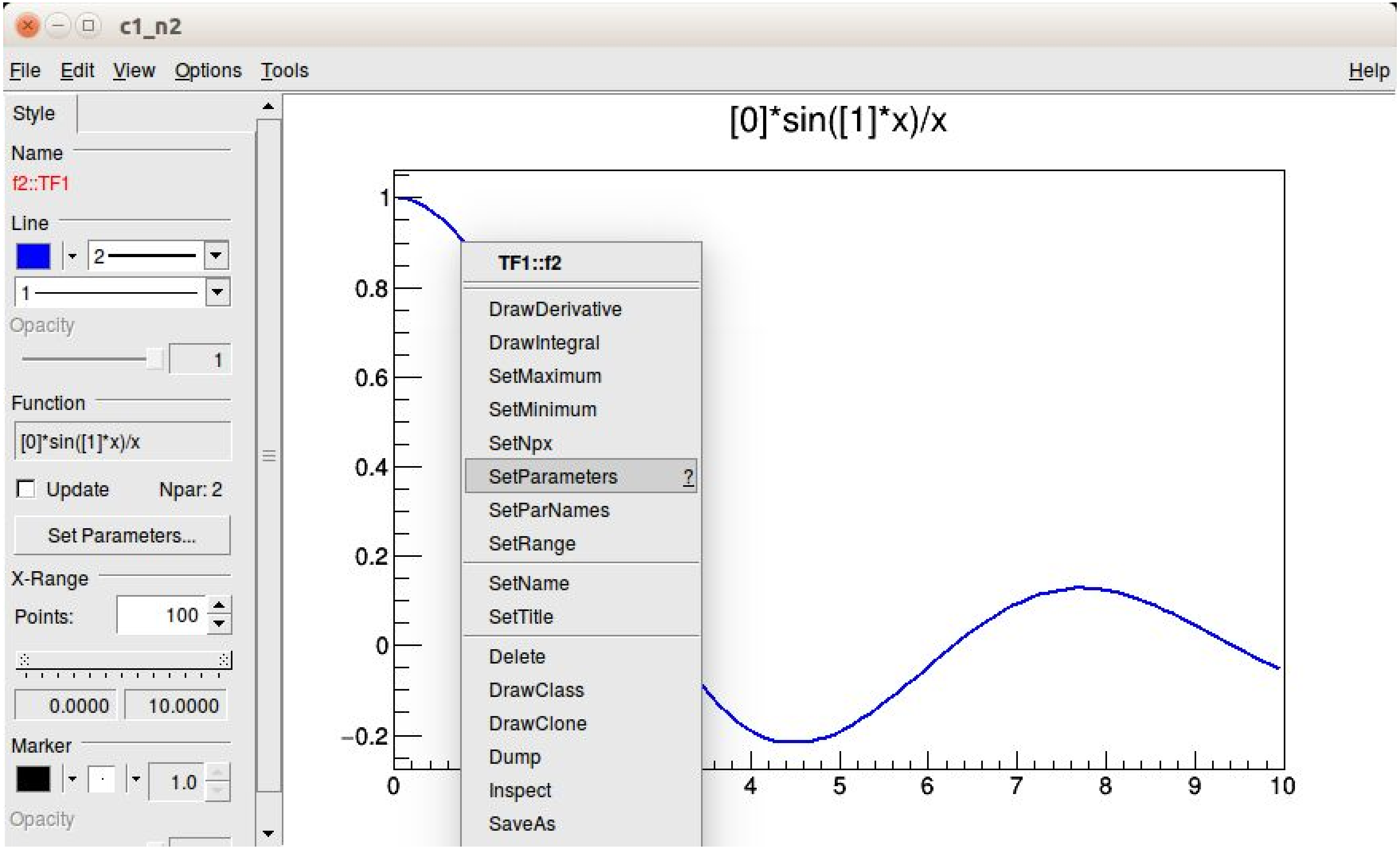
Interaction with the plot



Interaction with the plot



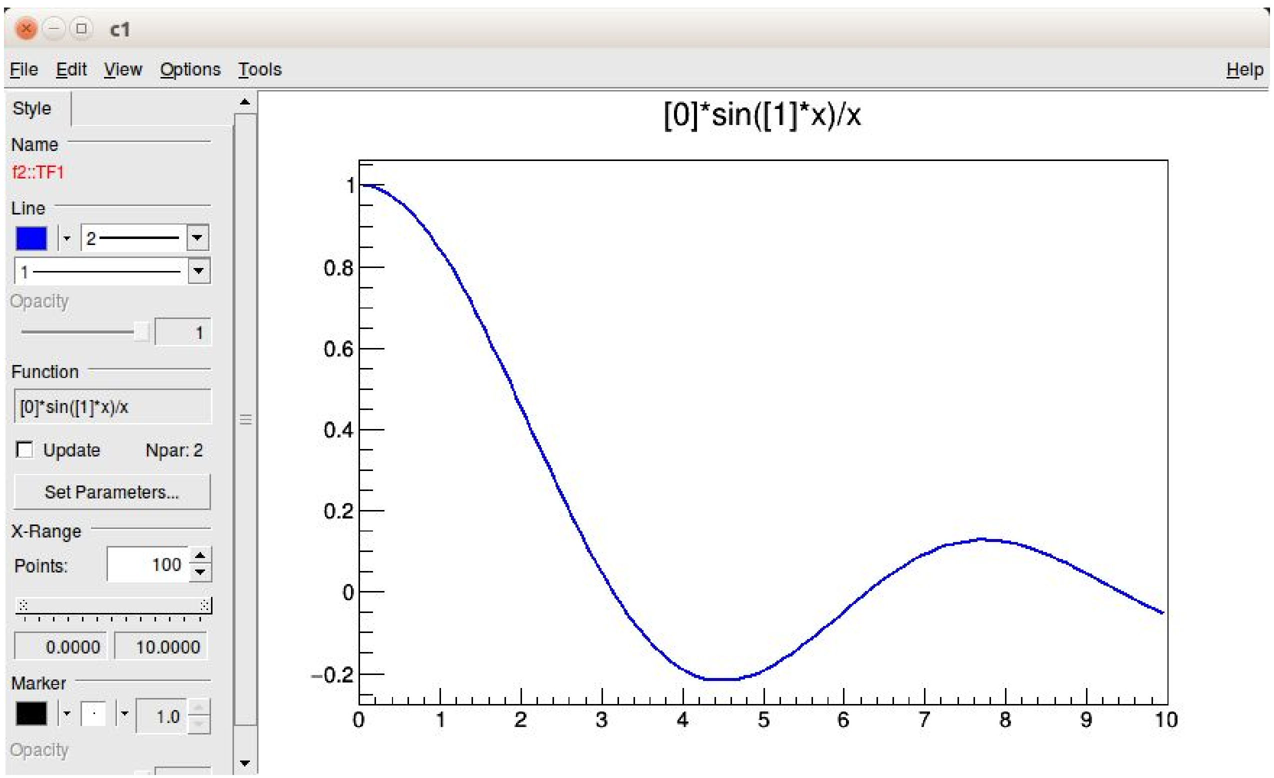
Interaction with the plot

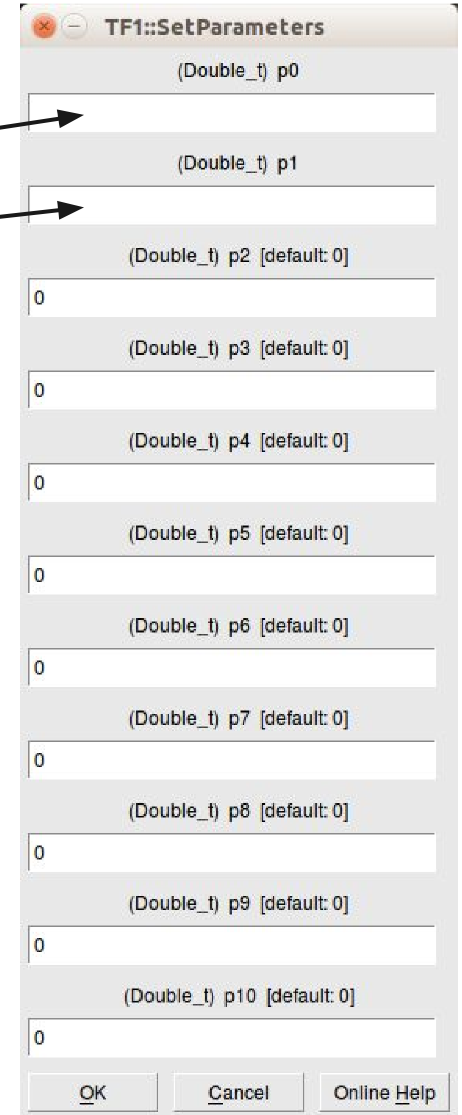


Interaction with the plot

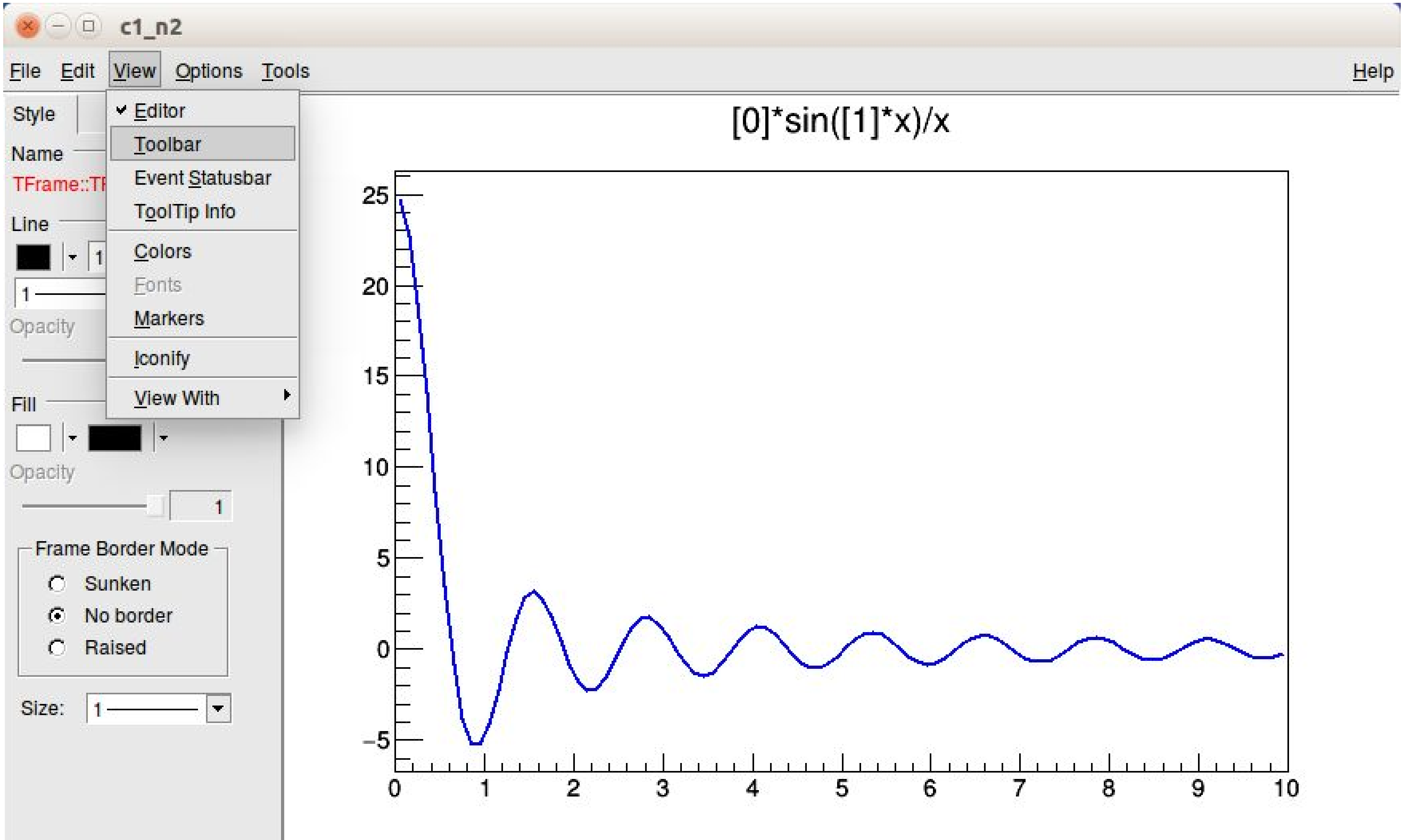
Two parameter function:

- ➔ [0] or p0 → normalization → set to 5
- ➔ [1] or p1 → frequency → set to 5

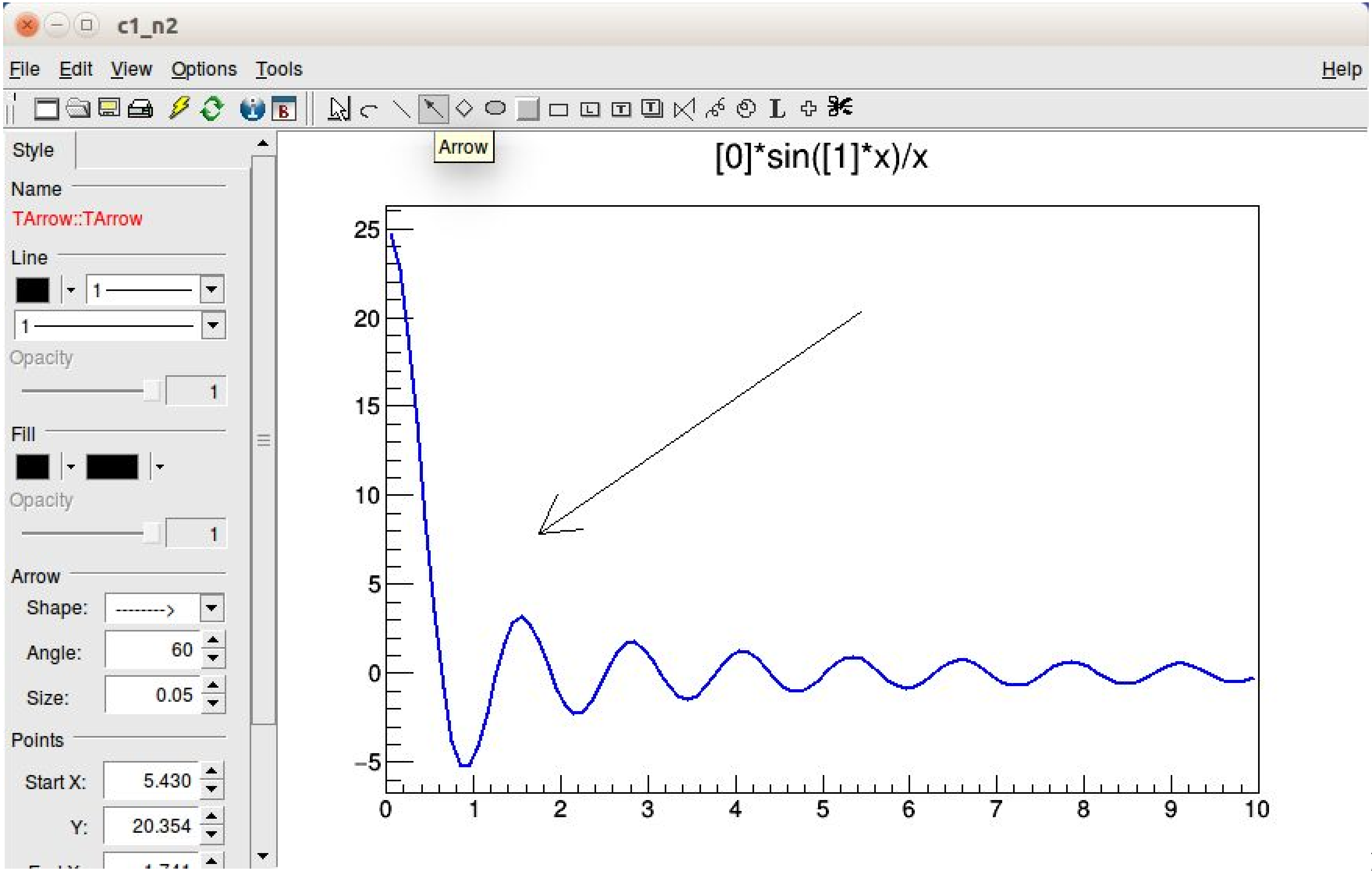




Interaction with the plot



Interaction with the plot



Graphs

→ Download ExampleData.txt

```
$ root -l -n
root [0] TGraphErrors gr("ExampleData.txt")
root [1] gr.Draw("AP")
```

axis points

→ Another example

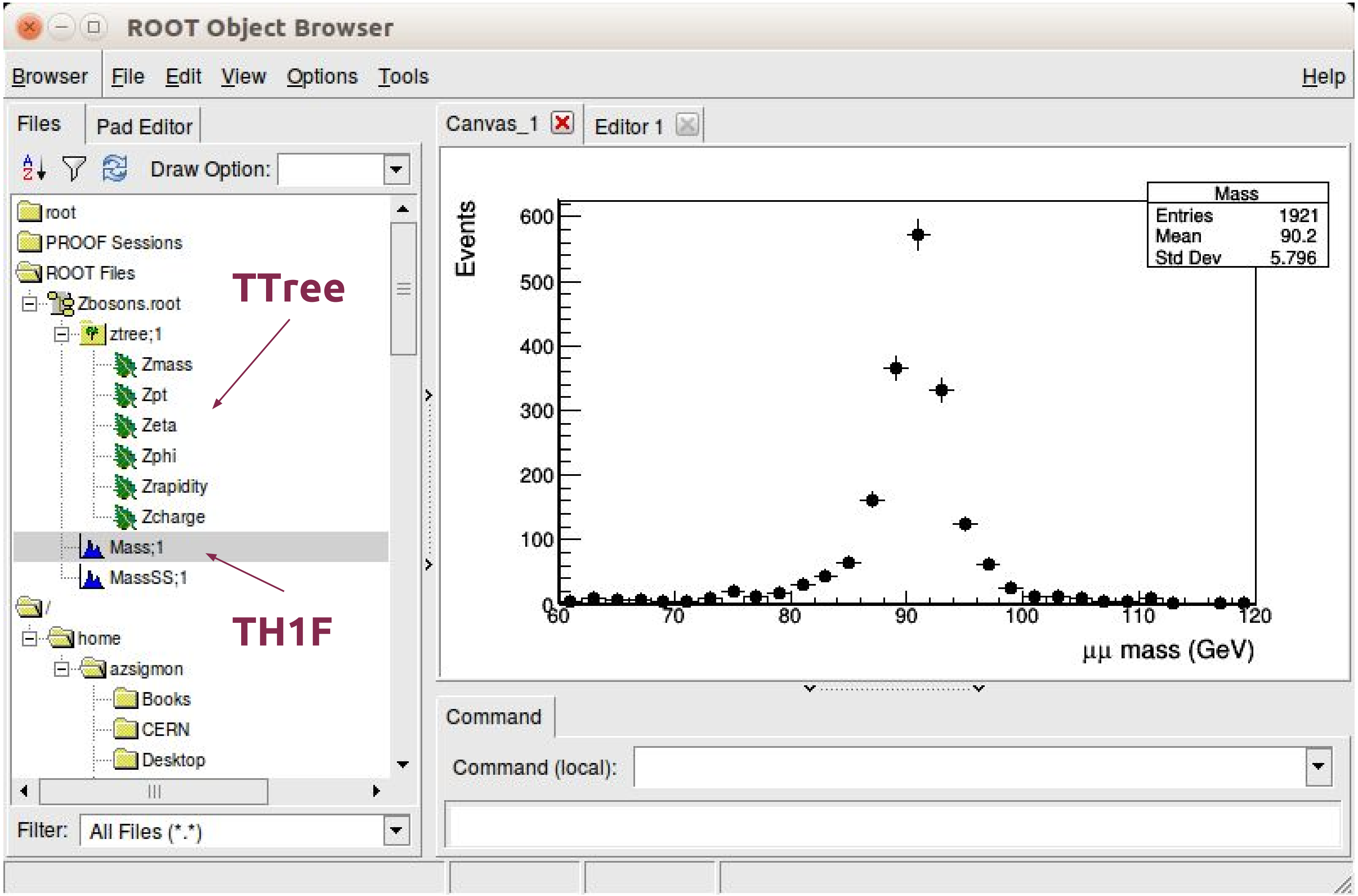
```
root [2] TGraph g
root [3] g.SetTitle("My graph;myX;myY")
root [4] g.SetPoint(0,1,0)
root [5] g.SetPoint(1,2,3)
root [7] g.SetMarkerStyle(kFullSquare)
root [8] g.SetMarkerColor(kRed)
root [9] g.SetLineColor(kOrange)
root [10] g.Draw("APL")
```

Discover ROOT file interactively

- Download `Zbosons.root`
  http://annazsigmond.web.elte.hu/root-tutorial/Zbosons.root
- Load file in root
- Inspect file with TBrowser

```
$ root -l -n Zbosons.root
root [0]
Attaching file Zbosons.root as _file0...
(class TFile *) 0x2da3ff0
root [1] new TBrowser
(class TBrowser *) 0x310ce40
```

TBrowser


ROOT Object Browser
Browser
File
Edit
View
Options
Tools
Help
Files
Pad Editor
Draw Option:
root
PROOF Sessions
ROOT Files
Zbosons.root
ztree;1
Zmass
Zpt
Zeta
Zphi
Zrapidity
Zcharge
Mass;1
MassSS;1
/
home
azsigmon
Books
CERN
Desktop
Filter:
All Files (*.*)
TTree
TH1F
Canvas_1
Editor 1
Mass
Entries 1921
Mean 90.2
Std Dev 5.796
Events
μμ mass (GeV)
Command
Command (local):

Discover ROOT file interactively

→ Inspect file in command line

```
$ root -l -n Zbosons.root
root [0]
Attaching file Zbosons.root as _file0...
(class TFile *) 0x2da3ff0
root [1] _file0->ls()
TFile**         Zbosons.root
 TFile*         Zbosons.root
  KEY: TTree    ztree;1  Z boson candidate events
  KEY: TH1F     Mass;1
  KEY: TH1F     MassSS;1
```

→ Access objects from file using pointers

```
root [2] TH1F *h1 = (TH1F*)_file0->Get("Mass");
root [3] h1->Draw()
```

Histograms

TH1 base class

- ➔ 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

same with 2D and 3D histograms e.g. TH2F

Example constructor

```
TH1F *h2 = new TH1F("h2","h2;mass (GeV);counts",60,60,120)
```

object

name

number of bins

max

title: histogram title; x axis title; y axis title

min

Read TTree interactively

→ Access tree and draw variable

```
root [3] TTree *t = (TTree*)_file0->Get("ztree")
root [4] t->Draw("Zmass")
```

→ Draw with specific conditions and drawing options

```
root [5] t->Draw("Zmass","Zcharge==0","ep")
```

→ Save output to histogram with >>

```
root [6] TH1F *h2 = new TH1F("h2","h2;mass
(GeV);counts",60,60,120)
root [7] t->Draw("Zmass>>h2")
root [8] h2->Draw("ep")
```

error bars **p**oints

Read TTree interactively

- 2D plots with different drawing options

```
root [10] t->Draw("Zphi:Zrapidity","","colz")
root [11] t->Draw("Zphi:Zrapidity","","lego")
```

- More complicated expressions

```
root [12] t->Draw("sin(Zphi)")
root [13] t->Draw("log(Zpt):Zrapidity:Zphi")
root [14] .q
```

- Many more possibilities with `TTree`
- For graphics options see these classes
  TAttFill, THistPainter, TGraphPainter, ...

Fit histogram interactively

- Fit histogram with predefined function

```
root [1] TTree *t = (TTree*)_file0->Get("ztree")
root [2] TH1F *h2 = new TH1F("h2","h2;mass
(GeV);counts",60,60,120)
root [3] t->Draw("Zmass>>h2")
root [4] h2->Fit("gaus")
```

- `gaus` predefined Gaussian function with 3 parameters
- Access fit results by index or name

```
root [5] h2->GetFunction("gaus")->GetParameter
(0)
root [6] h2->GetFunction("gaus")->GetParameter
("Mean")
root [7] h2->GetFunction("gaus")->GetParError(1)
```

ROOT macros

- ROOT macros are basically lightweight programs
- The general structure in file `MacroName.C`
- Function has same name as file

```
void MacroName() {

  //lines of C++ code

}
```

- Same lines as we typed in the ROOT prompt can be saved in a macro

ROOT macros

- The macro is executed in terminal

```
$ root MacroName.C
```

- or in ROOT prompt

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

- or loaded in ROOT and then executed as a function

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

Interpretation and compilation

- Before ROOT interprets the code (just in time compilation)
- ROOT can also compile the code in the prompt by adding a “+” as in

```
$ root Macro.C+
```

- Or generate shared library and execute function in two steps

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

- **Recommended to compile every code!**

Compilation ++

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

→ In the `myMacro.C` file

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

→ Compile and execute in terminal

```
$ g++ myMacro.C `root-config --cflags --libs` -o
myMacro
$ ./myMacro
```

Example macro

Goal is to fit the Z boson mass spectrum with a realistic function (convolution of a Gauss and a Breit-Wigner + exponential background) using the following steps

- ➔ define fit function
- ➔ read tree
- ➔ fill histogram
- ➔ fit histogram
- ➔ make a nice plot with axis labels, text and legend

Download `fitZbosons.C`

http://annazsigmond.web.elte.hu/root-tutorial/fitZbosons.C

Example macro

- ➔ Open file and tree

```
void fitZmacros() {
  TFile *inf = TFile::Open("Zbosons.root");
  TTree *ztree = (TTree*)inf->Get("ztree");
```

- ➔ Initialize tree

```
  float Zmass;
  int Zcharge;
  ztree->SetBranchAddress("Zmass",  &Zmass);
  ztree->SetBranchAddress("Zcharge",&Zcharge);
```

name of branch in tree

previously declared variable with the same type as branch

Example macro

- Define histogram

```
TH1F *hmass = new TH1F("hmass","",60,60,120);
```

- Loop over tree

```
for(int j=0; j<ztree->GetEntries(); j++) {
  ztree->GetEntry(j);
```

- Skip entry if non-zero charge
- Fill histogram otherwise

```
    if(Zcharge != 0) continue;
    hmass->Fill(Zmass);
}
```

Example macro

- Open a canvas

```
TCanvas *c1 = new TCanvas("c1","Dimuon mass",
600, 600);
```

width height name title

- Draw histogram with errorbars and points

```
hmass->Draw("ep");
```

- Save canvas in png (or pdf, eps, gif, …)

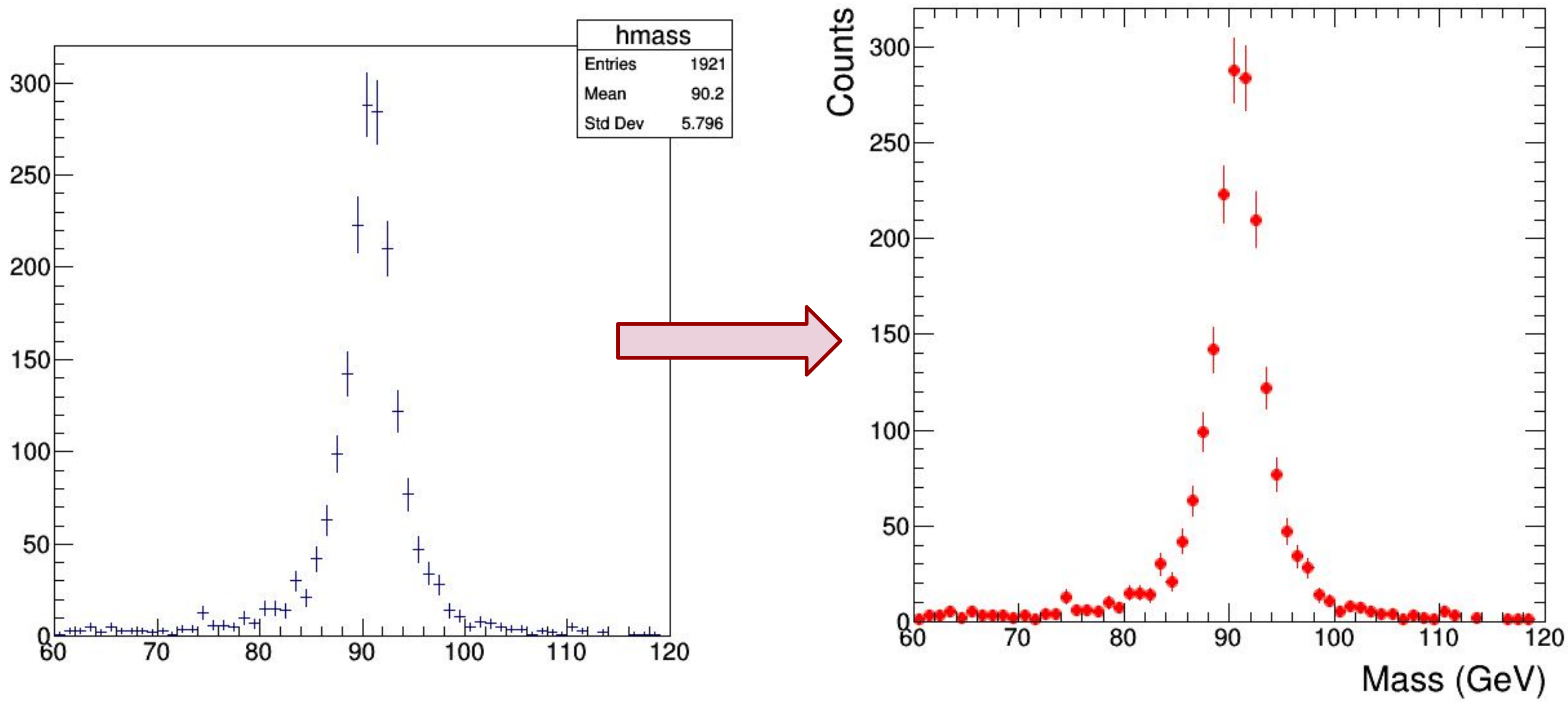
```
c1->SaveAs("./Zpeak.png");
```

- Run macro and check the resulting plot

```
$ root -l -n -b -q fitZbosons.C+
```

Example macro

➔ Let’s format the plot



Example macro

→ Set margins in relative coordinates

```
c1->SetTopMargin(0.05);
c1->SetRightMargin(0.05);
c1->SetBottomMargin(0.12);
c1->SetLeftMargin(0.13);
```

→ Set ticks on both sides

```
c1->SetTickx(1);
c1->SetTicky(1);
```

→ Set histogram style (kRed = 2 predefined constant)

```
hmass->SetMarkerStyle(20);
hmass->SetMarkerColor(kRed);
hmass->SetLineColor(kRed);
```

Example macro

- Set axis titles with size and offset

```
hmass->GetXaxis()->SetTitle("Mass (GeV)");
hmass->GetYaxis()->SetTitle("Counts");
hmass->GetXaxis()->SetTitleSize(0.05);
hmass->GetYaxis()->SetTitleSize(0.05);
hmass->GetYaxis()->SetTitleOffset(1.2);
```

- Remove stat box

```
gStyle->SetOptStat(0);
```

- gStyle is a preloaded TStyle object
- Usually we have a rootlogon.C script that sets the style of the plots and gets loaded at every ROOT start

Example macro

Includes for compilation

```
#include "TROOT.h"
#include "TMath.h"
#include "TFile.h"
#include "TTree.h"
#include "TH1.h"
#include "TF1.h"
#include "TCanvas.h"
#include "TStyle.h"
#include "TFitResult.h"
#include "TLegend.h"
#include "TLatex.h"
#include <iostream>
using namespace std;
```

- ➔ Each class that we use has to be included in the header
- ➔ `TFile`, `TTree`, `TH1`, `TCanvas`, `TStyle` are already used in our code → try running without them

Example macro

- Definition of fitting function is a convolution of Breit-Wigner and Gauss

```
Double_t RBWGaus(Double_t *x, Double_t *par) {
  //Fit parameters: ...
  //Setup for the integral ...
  for(Double_t i=1.0; i<=np/2; i++) {
    xx = xlow + (i-.5) * step;
    fbw = TMath::BreitWigner(xx,par[1],par[0]);
    sum += fbw * TMath::Gaus(x[0],xx,par[3]);
    //other side...
  }
  return (par[2] * step * sum * (1./sqrt
(2*TMath::Pi())) / par[3]);
}
```

User defined functions

- Outside the main code defined as regular C++ function
- Return value usually `Double_t`
- Arguments `Double_t *x` and `Double_t *par`
- The length of `x` is 1 for 1D → only `x[0]` appears
- `par` is an array of the parameters, their number appears in the definition of the `TF1` in the main code

```
TF1 *f = new TF1("f",RBWGaus,60,120,4);
```

name of TF1 object

range minimum

number of parameters

instead of formula the
name of the C++ function

range maximum

Example macro

➔ Set function parameters and their names

```
  f->SetParameters(2.495, 91.0, 2000.0, 2.0);
  f->SetParNames("BW width", "BW mean", "Area",
"Sigma");
```

➔ Setup drawing style of function

```
  f->SetLineColor(kBlue);
```

➔ Draw function on the histogram

```
  f->Draw("same");
```

➔ Check plot

Example macro

- Fix or limit parameters in the fit

```
f->FixParameter(0, 2.495); //PDG value
f->SetParLimits(1, 86, 96);
```

- Fit the histogram

```
TFitResultPtr fitr = hmass->Fit(f,"RNS","");
```

- `TFitResultPtr` returns pointer to fit results e.g. parameter values, errors, covariance matrix
- Second argument is the fitting options
  - `R`: use range from function
  - `N`: do not draw
  - `S`: return values to the `TFitResultPtr`
  - many more in TH1 class reference

Example macro

- Put a legend on the plot

```
TLegend *l = new TLegend(0.18,0.78,0.34,0.90);
l->SetTextSize(0.04);
l->AddEntry(hmass,"Z#rightarrow#mu#mu","lp");
l->AddEntry(f,"Fit","l");
l->Draw();
```

- Place in relative coordinates (xmin, ymin, xmax, ymax)
- Entry for each object with latex text
- Options for entries
  - `l`: line
  - `p`: point
  - `f`: fill (box)

Example macro

- ➔ Put text on the plot

```
TLatex *tx = new TLatex();
tx->SetTextSize(0.03);
tx->SetTextAlign(12);
tx->SetTextFont(42);
tx->SetNDC(kTRUE);
```

- ➔ Alignment = 10 * horizontal + vertical
  - horizontal: 1=left, 2=centered, 3=right
  - vertical: 1=bottom, 2=centered, 3=top
- ➔ Font = 10 * font number + precision
  - 4 = arial normal
  - 6 = arial bold
  - ...

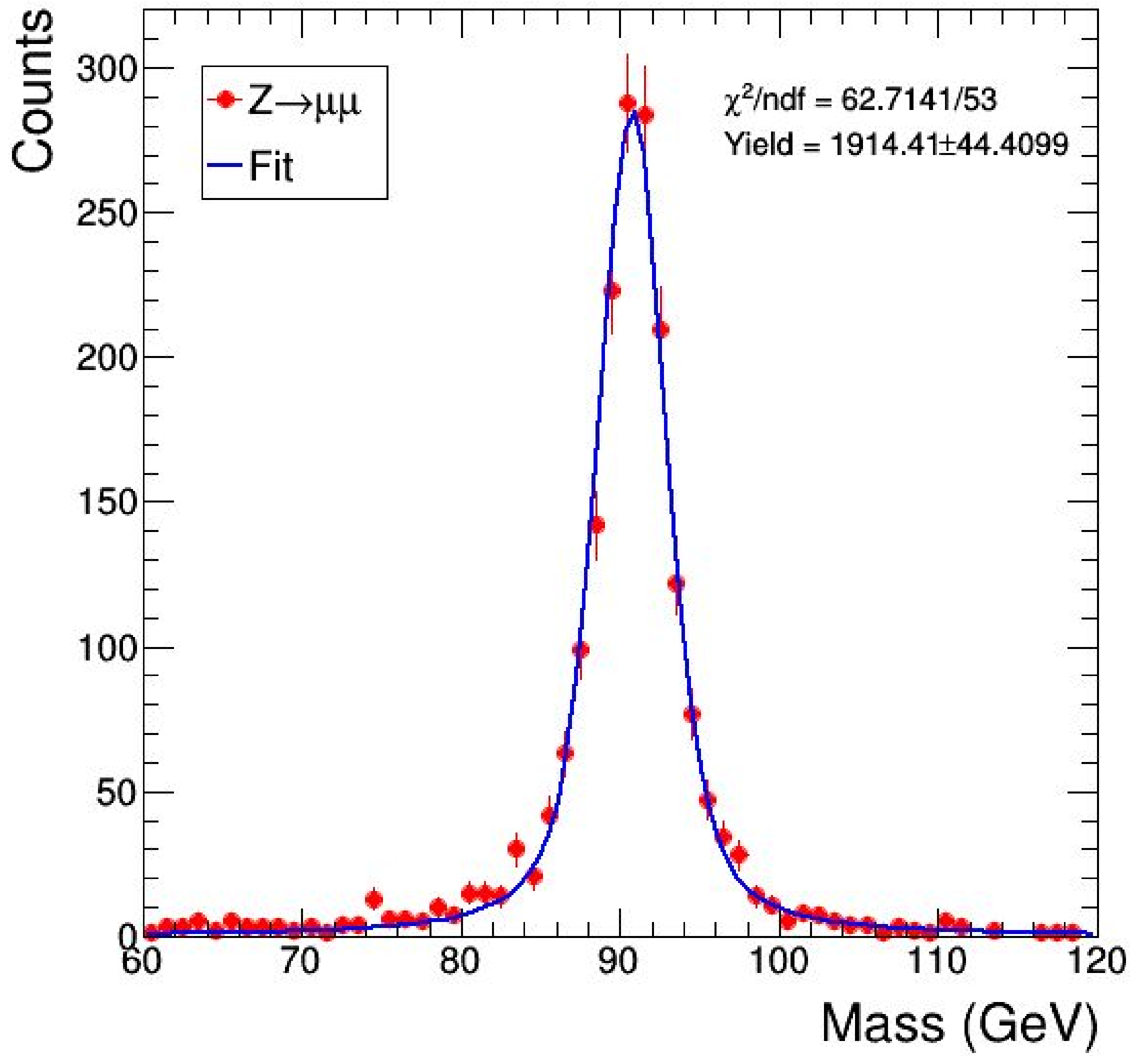
Example macro

- Put text on the plot

```
  tx->DrawLatex(0.63,0.87,Form("#chi^{2}/ndf = %g/%d",fitr->Chi2(),fitr->Ndf()));
```

- `DrawLatex` creates a copy with new parameters
  - x position
  - y position (in relative coordinates because `SetNDC(kTRUE)`)
  - string
- `Form()` works like `sprintf`
- `fitr->` is the way to access the results of the fit
  - `Chi2()`
  - `Parameter(1)`
  - `ParError(1)`
  - ...

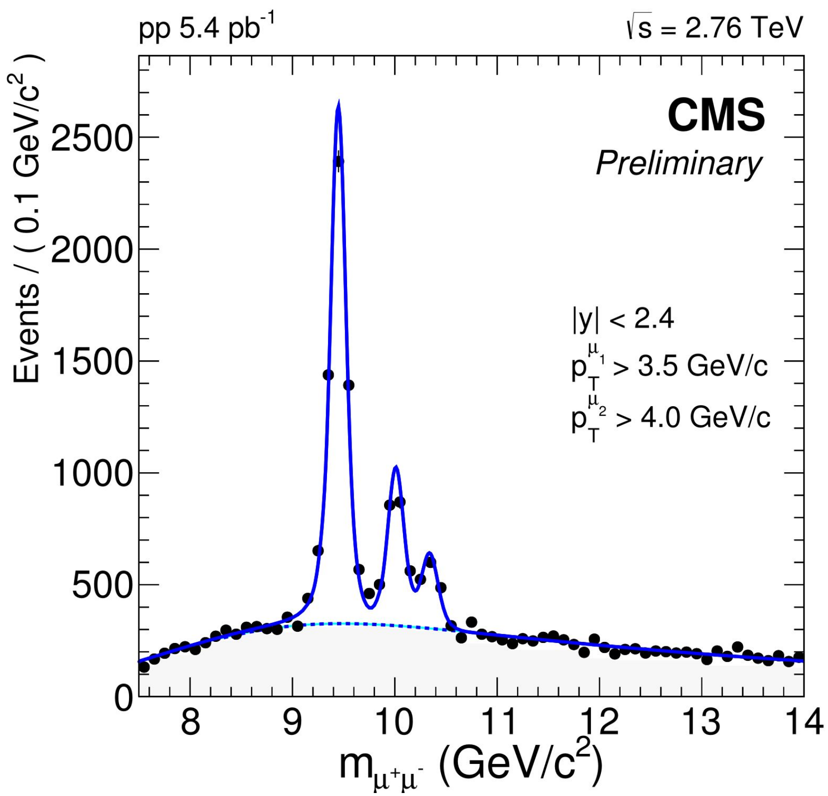
Example macro



Upsilon suppression in heavy-ion collisions

Introduction to long exercise

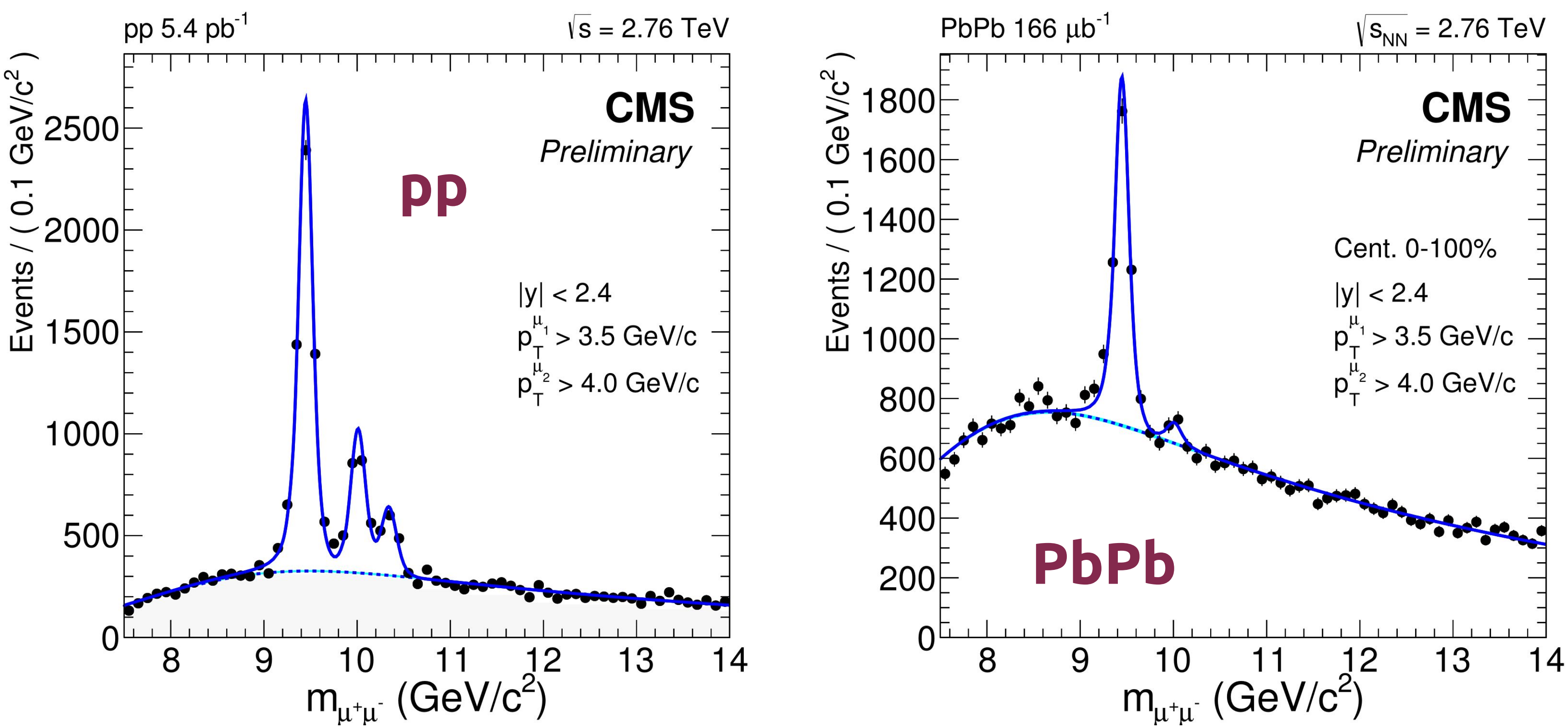
Upsilons in pp and PbPb collisions



- ➔ References:
  - ◆ http://arxiv.org/abs/1208.2826
  - ◆ CMS PAS HIN-15-001
- ➔ Bottom quark-antiquark bound states 1S, 2S, 3S
- ➔ Decaying to opposite charge muon pairs
- ➔ To determine the yield of each state, we fit the invariant mass distribution of muon pairs

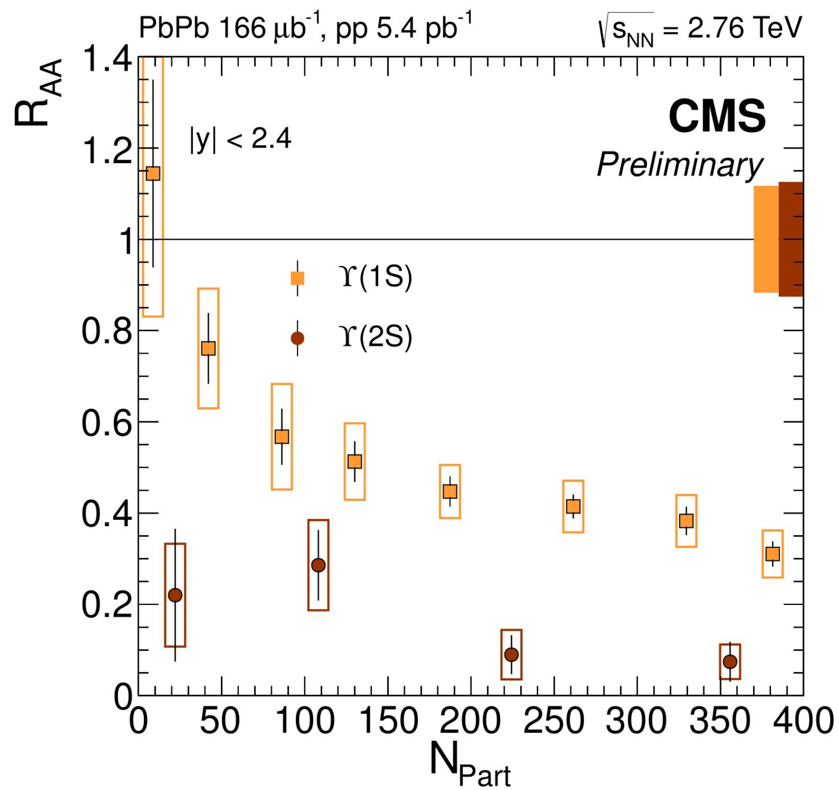
Upsilons in pp and PbPb collisions

→ In PbPb collisions a hot and dense matter (quark-gluon plasma) is produced where the upsilons are "melting"

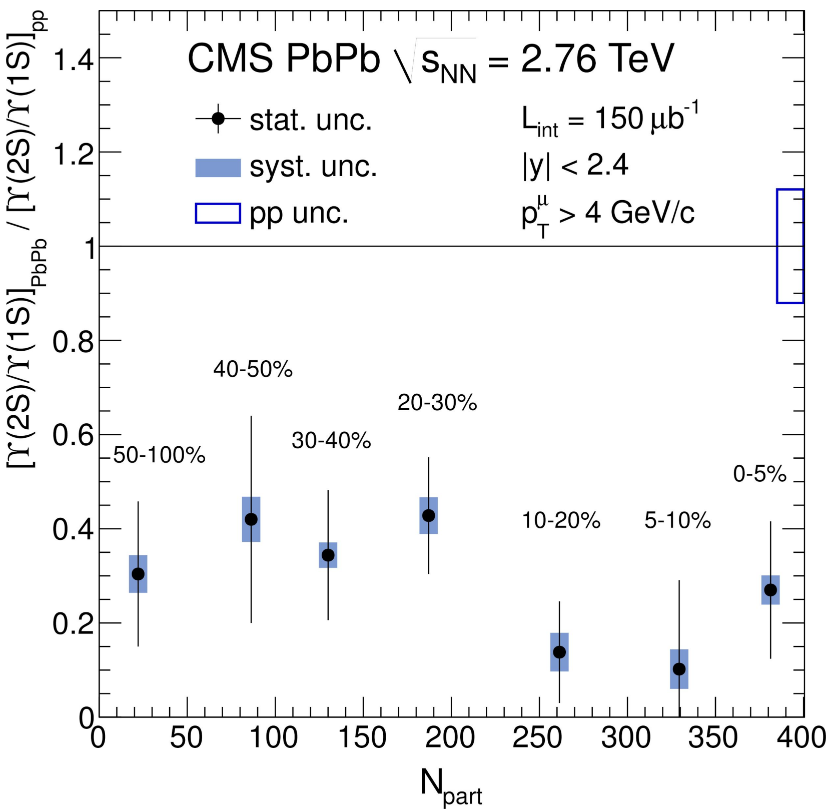


Upsilons in pp and PbPb collisions

- ➔ Ratio of yield in PbPb and pp as a function of collision centrality is decreasing
- ➔ Nuclear modification factor = PbPb yield / (pp yield * N_{coll})
- ➔ Centrality can be expressed in many ways
  - ◆ impact parameter
  - ◆ N_{part}
  - ◆ N_{coll}
  - ◆ % of total cross section → this is what we measure



Upsilons in pp and PbPb collisions



- ➔ Comparing 2S / 1S ratio in PbPb and pp collisions by double ratio
- ➔ Corrections and normalization cancels in the ratio
- ➔ Goal for today to produce such a measurement by fitting the invariant mass spectrum of muon pairs

Exercise details

- ➔ Download data files

  http://annazsigmond.web.elte.hu/root-tutorial/upsilonTree_2p76TeV_pp_data.root
  http://annazsigmond.web.elte.hu/root-tutorial/upsilonTree_2p76TeV_PbPb_data.root

- ➔ Download some fitting functions

  http://annazsigmond.web.elte.hu/root-tutorial/FitFunctions.h

- ➔ Open file and create skeleton code for looping over the tree

```
$ root -l -n upsilonTree_2p76TeV_PbPb_data.root
root [1] TTree *upsilonTree = (TTree*)_file0-
>Get("UpsilonTree")
root [2] upsilonTree->MakeClass()
Info in <TTreePlayer::MakeClass>: Files:
UpsilonTree.h and UpsilonTree.C generated from
TTree: UpsilonTree
```

Exercise details

- ➔ The muons have been already selected and paired
- ➔ Create invariant mass histograms in the 7 - 14 GeV mass region for both pp and PbPb `h->Fill (invariantMass)`
- ➔ Fill histogram only when both muons of the pair have p_T > 4 GeV (`muPlusPt > 4 && muMinusPt > 4`) and opposite charge (`QQsign == 0`)
- ➔ Save histograms

Exercise details

- ➔ Fit the invariant mass histograms to calculate the yield of Υ(1S), Υ(2S), Υ(3S) in both pp and PbPb
- ➔ Fit function is a sum of signal and background
- ➔ Try different functions
  - ◆ 3 × signal: Gaussian, Crystal-Ball
  - ◆ background: exponential, error function, polynomials
- ➔ Constrain fit parameters according to known physics
  - ◆ m(1S) = 9.4603 GeV, m(2S) = 10.023 GeV, m(3S) = 10.355 GeV
  - ◆ width scales with mass e.g. σ(2S) / σ(1S) = m(2S) / m(1S)
- ➔ Choose which fit looks best (e.g. lowest χ^2)

Exercise details

- Next step is to divide the PbPb data into centrality bins
- Additional 6 histograms in the following bins

Centrality	Bin	<Npart>	<Ncoll>
0 - 10%	[0, 3]	355 ± 3	1484 ± 120
10 - 20%	[4, 7]	261 ± 4	927 ± 81
20 - 30%	[8, 11]	187 ± 4	563 ± 53
30 - 40%	[12, 15]	130 ± 5	326 ± 34
40 - 50%	[16, 19]	86 ± 4	176 ± 21
50 - 100%	[20, 39]	22 ± 2	30 ± 5
0 - 100%	[0, 39]	114 ± 3	363 ± 32

More LHC physics

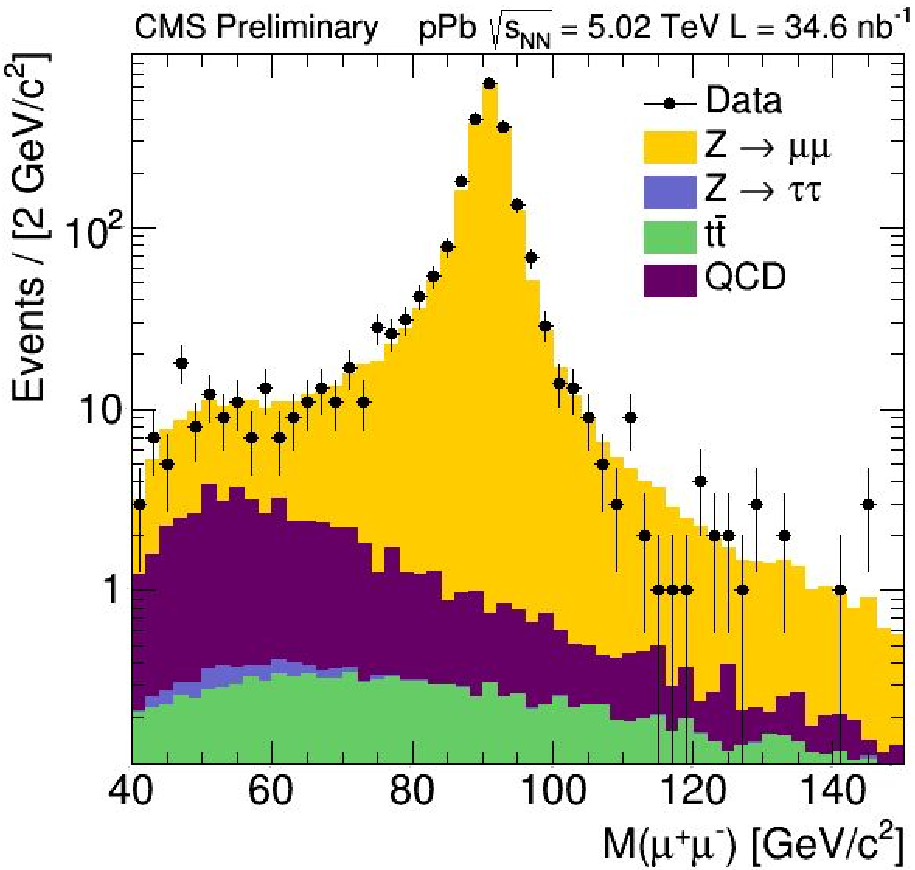
Válogatott fejezetek a nagyenergiás fizikából
előadás

ff1c9a107

https://sites.google.com/site/nagyenergiasfizika/

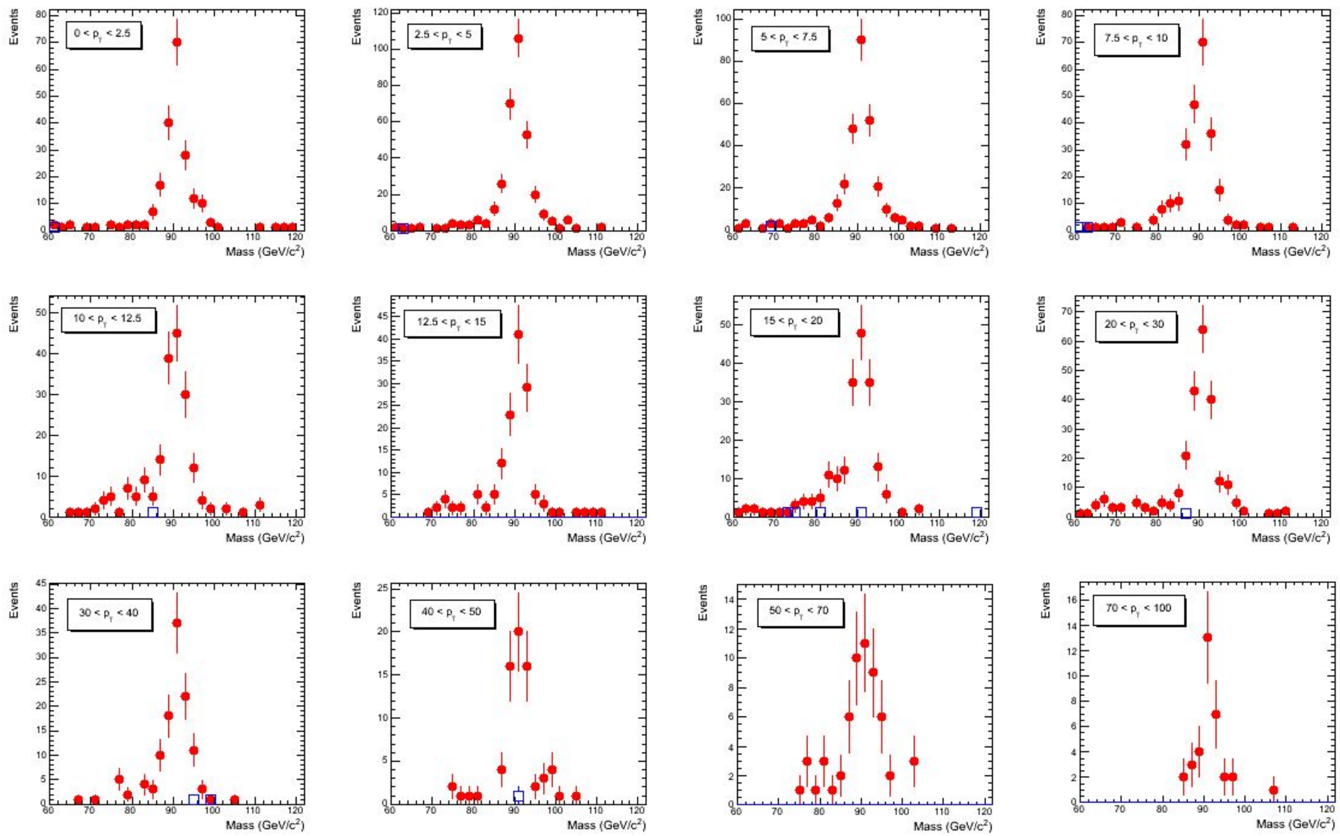
Additional useful graphics

THStack to put histrograms on top of each other



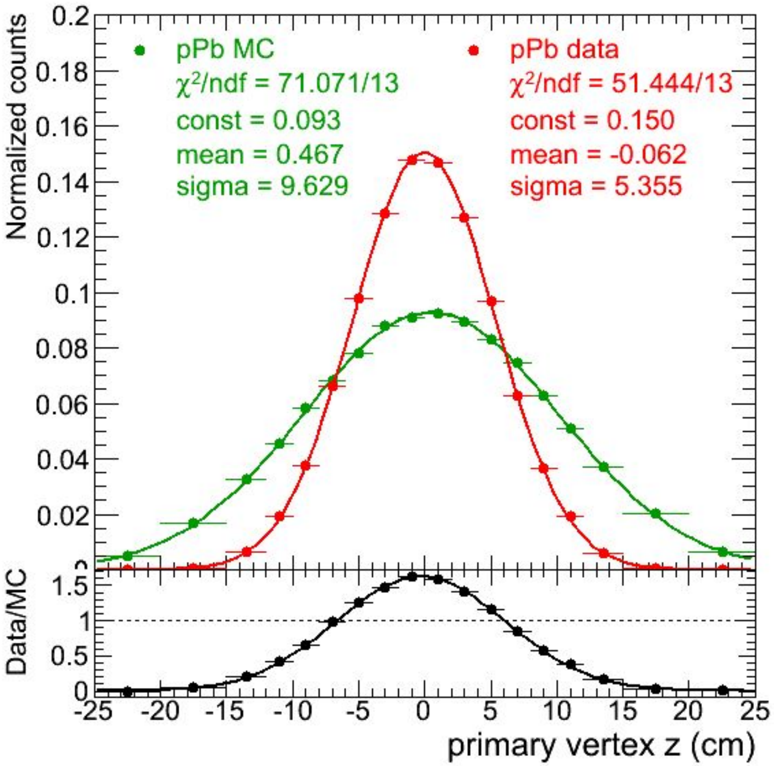
Additional useful graphics

TPad::Divide() (TCanvas::Divide()) to have more than one figure in one picture



Additional useful graphics

Multiple TPads in one TCanvas




```
void makeTwoPads(TCanvas *c1) {
 c1->cd();
 TPad *p1 = new TPad("p_1","",
0.0,0.25,1.0,1.0,0,0,0);
 p1->Draw();
 p1->SetNumber(1);
 p1->SetBottomMargin(0);
 p1->SetTopMargin(0.06);
 TPad *p2 = new TPad("p_1","",
0.0,0.0,1,0.25,0,0,0);
 p2->Draw();
 p2->SetNumber(2);
 p2->SetTopMargin(0);
 p2->SetBottomMargin(0.40);
 p2->SetLogy(0);
}
```