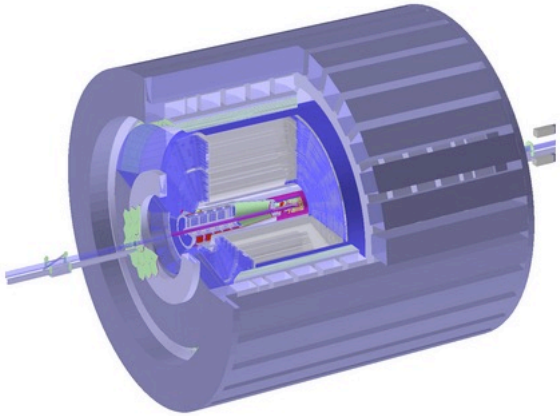
esipap

European School of Instrumentation
in Particle & Astroparticle Physics

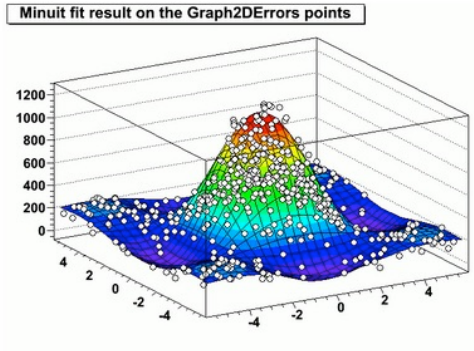
C++ programming

Computing session: ROOT

ROOT: overview

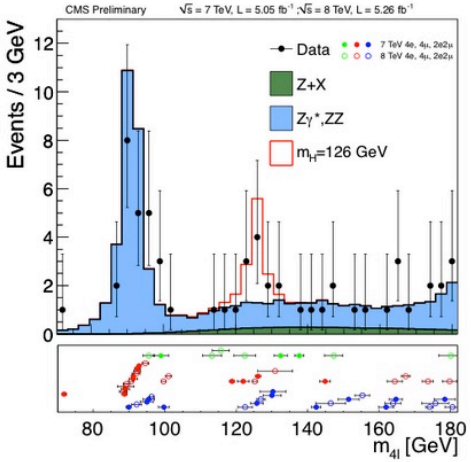


STAR detector

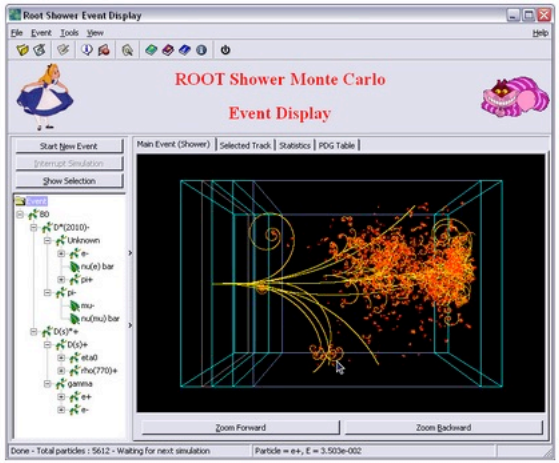


Minuit fit result on the Graph2DErrors points

- **Data analysis &Visualization**
- **Fitting & statistics**
- **Randomization** & MC
- Storing information
- Mathematical libraries
- Physics vectors
- GUI
- Detectors
- Event display



CMS Data MC Ratio Plot



An event display based on ROOT GUI

ROOT: useful documentation

- Website
- User's guide
- Reference's guide
- How to
- Tutorials

ROOT: several modes

ROOT can be used in several ways:

- Use the interpreter (**CINT**)
  - Extend with the use of macros
- Use **ACLIC** (compilation)
- Link ROOT libraries

1 - ROOT - CINT

```
bash$ root
```

A window showing the ROOT version will appear on the screen. Then a user console is openend with a prompt where instructions can be launched:

```
root [0]
```

You can now use ROOT interpreter, called CINT, to execute sequentially C++ instructions as shown below:

```
root [0] float a = 1.2345;
root [1] cout<<"a^0.5 = "<<sqrt(a)<<endl;
```

Finally to quit ROOT interpreter, you simply have to to do:

```
root [5] .q
```

The instructions that have been launched are stored in a file /.root_history. Moreover, in a similar way as you would do in a Terminal, you can use the top and bottom array to browse in the history from the ROOT interpreter.

1 - ROOT - CINT

It is interesting to notice that ROOT can be launch with several options. To know then, you can simply type:

```
root --help
```

As you can see from the output, it is also possible to use dedicated file to configure ROOT usage.

- **.rootrc** is a file containing some global default for your ROOT session. There are three locations where the system looks for this file: $ROOTSYS/system.rootrc, /.rootrc and ./.rootrc (the latter taking precedence over the former). If you type in a root session the command `gEnv->Print()` you see which defaults are active.
- **.rootalias.C**: It is loaded and executed at ROOT startup. If can define some often used functions.
- **.rootlogon.C**: It contains the code that will be executed at ROOT startup.
- **.rootlogoff.C**: it is called when the session is finishing.

These last 3 files are always taken from the current working directory. If you would like to have a global version of one or all of these files make the change in your /.rootrc.

1 – Macros

Start a simple example by using **TMath::Sqrt()**

- Check the documentation
- Use the auto-completion

- **Use it in CINT**

```
root [0] TMath::Sqrt(2)
(Double_t)1.41421356237309515e+00
```

- **Macro**
  - Simple case, no function, write the lines in a file test.C
  - Declare function(s) in a file test.C

```
1 {
2   TMath::Sqrt(2);
3 }
```

test.C

```
1 void f(double a){
2   cout<<"sqrt("<<a<<") = "<<TMath::Sqrt(a)<<endl;
3 }
```

In CINT:

```
root [0] .L test.C
root [1] f(5)
sqrt(5) = 2.23607
```

2– ACLIC compilation

It is required to properly:

- Include header files
- Define the namespace used
- …

```
#include <iostream>
#include <TMath.h>
using namespace std;

void psqrt(double a){
  cout<<"sqrt("<<a<<") = "<<TMath::Sqrt(a)<<endl;
}
```

In CINT:

.L test.cpp+
#→ compile and load only if the code is new or has been modified
psqrt(2.); #to call the function

.L test.cpp+
→ Force a new compilation
Psqrt(2.);

3 – Several modes

main.cc

```
#include <iostream>
#include "TMath.h"

using namespace std;

void f(double a){
  cout<<"sqrt("<<a<<") = "<<TMath::Sqrt(a)<<endl;
}


int main(){
  f(5);
  return 0;
}
```

Need to write a function main

In the terminal:

g++ `root-config --cflags --glibs` -o main main.cc

root-config –cflags: needed to access the headers
root-config –glibs: needed to access the libraries

Using math functions

Start a simple example by using **TMath::PoissonI(Double_t, Double_t)**

- Call the function
- Use the class **TF1**

```
root [0] TMath::PoissonI(

Double_t PoissonI(Double_t x, Double_t par)
root [0] TMath::PoissonI(2,2)
(Double_t)2.7067056647322540 5e-01
```

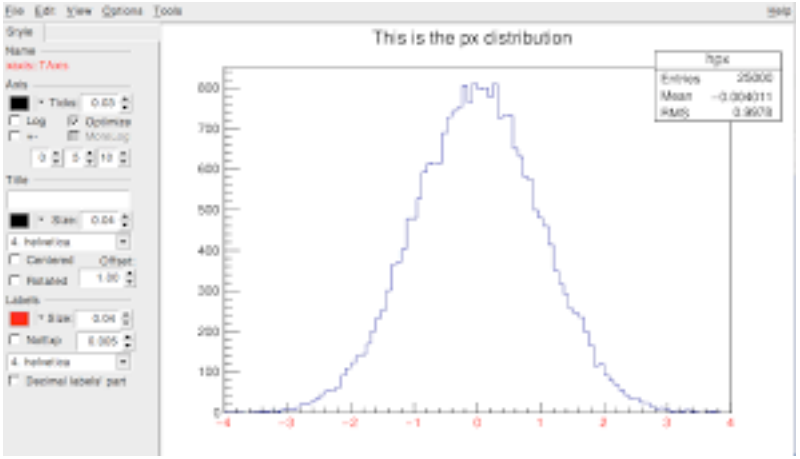
```
root [0] TF1 f("f","TMath::PoissonI(x,[0])",0,10);
root [1] f.SetParameter(0,2);
root [2] f.Eval(2)
(const Double_t)2.7067056647322540 5e-01
```

```
root [4] f.Integral(0,3)
(Double_t)6.7717316215148593 0e-01
root [5] f.Draw();
```

Histograms: TH1(F)

Use the class TH1F

- Check the documentation
- Use the auto-completion functionality
- Fill an histogram
- Access to statistics information
- Change the graphical parameters
  - Axis title
  - Line color and width
  - Y-range
- Save “it” in several format (.C, .root, .png)



TH1F Class Reference

Histogram Library

List of all members | Public Member Functions | Protected Member Functions | Friends | List of all members

1-D histogram with a float per channel (see TH1 documentation)}

Definition at line 567 of file TH1.h.

Public Member Functions

TH1F ()
Constructor. More...

TH1F (const char *name, const char *title, Int_t nbinsx, Double_t xlow, Double_t xup)
Create a 1-Dim histogram with fix bins of type float (see TH1::TH1 for explanation of parameters) More...

TH1F (const char *name, const char *title, Int_t nbinsx, const Float_t *xbins)
Create a 1-Dim histogram with variable bins of type float (see TH1::TH1 for explanation of parameters) More...

TH1F (const char *name, const char *title, Int_t nbinsx, const Double_t *xbins)
Create a 1-Dim histogram with variable bins of type float (see TH1::TH1 for explanation of parameters) More...

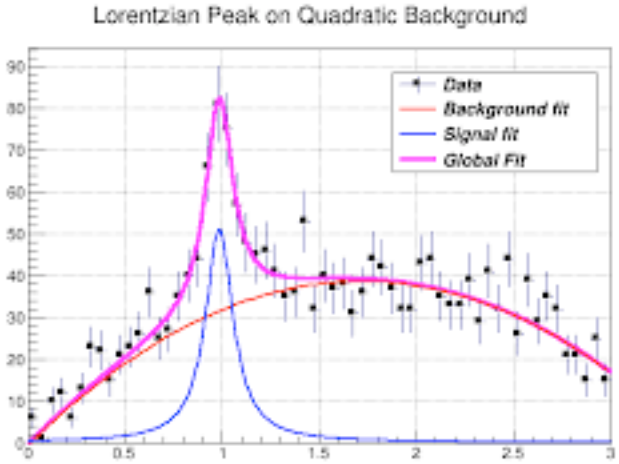
TH1F (const TVectorF &v)
Create a histogram from a TVectorF by default the histogram name is "TVectorF" and title = "". More...

TH1F (const TH1F &h1f)
Copy Constructor. More...

Canvas: TCanvas

Use the class TCanvas

- Check the documentation
- Change the scale (logScale)
- Create a new canvas divide into 2



Going a bit further:

- Drawing the error bars (see draw options)
- Overlaying 2 histograms
- Using a legend: **class TLegend**

Files: TFile

Use the class TFile

- Check the documentation
- Save the histo and/or the canvas
- Reopen the file
- Use a **TBrowser**

Now we can repeat this example but in a standalone compiled code

Fitting histograms

Interactive mode

- Exploring the possibilities
- Interpretation of the outputs
- Discussion about statistical results

Using methods in a macro/program

- Access the fitted values

Example to generate a random histo with a gaussian distribution:

```
TF1 *mygaus = new TF1("mygaus", "TMath::Gaus(x,3,5)",0,6");
TH1F h1("h1","test1",100,0,6");
h1.FillRandom("mygaus",10000);
h1.Draw();
```

TTree

Creating a TTree

- 2 branches (int/float)
- Filled with random values
- 10 entries
- Saved in a file: tree.root

```
#include "TTree.h"
#include "TFile.h"
#include "TRandom3.h"
int main(){
  //create variables
  int a;
  float b;
  //create a tree
  TTree* tree = new TTree("tree","");
  tree->Branch("br_a",&a);
  tree->Branch("br_b",&b);

  //will generate random numbers
  int nloop = 10;
  TRandom3 r;
  for(int i=0;i<nloop;i++){
    //set values for the 2 variables
    a = r.Poisson(5.1);
    b = r.Gaus(10,2);
    //cout<<a<<" "<<b<<endl;
    //fill the tree
    tree->Fill();
  }
  //Open a file
  TFile fout("tree.root","RECREATE");
  //Save the tree
  tree->Write();
  fout.Write();
  fout.Close(); // close the file
}
```

TTree

Creating a TTree

- Read the tree
- Print the values a/b

```
#include "TTree.h"
#include "TFile.h"
#include <iostream>

using namespace std;

int main(){
  //create variables
  int a;
  float b;
  //Open the file
  TFile fin("tree.root","READ");
  //Retrieve the tree
  TTree* tree = (TTree*) (fin.Get("tree"));
  //Create a "link" between the branches and the variables
  tree->SetBranchAddress("br_a",&a);
  tree->SetBranchAddress("br_b",&b);

  //Loop over the tree
  for(int i=0;i<tree->GetEntries();i++){
    //Read the entry i --> a & b will take the values of the current entry
    tree->GetEntry(i);
    cout<<a<<" "<<b<<endl;
  }
}
```

TTree

Going further

- In that example:
  - creating histograms
  - Performing fits …
- Draw function of TTree
- TTreeViewer