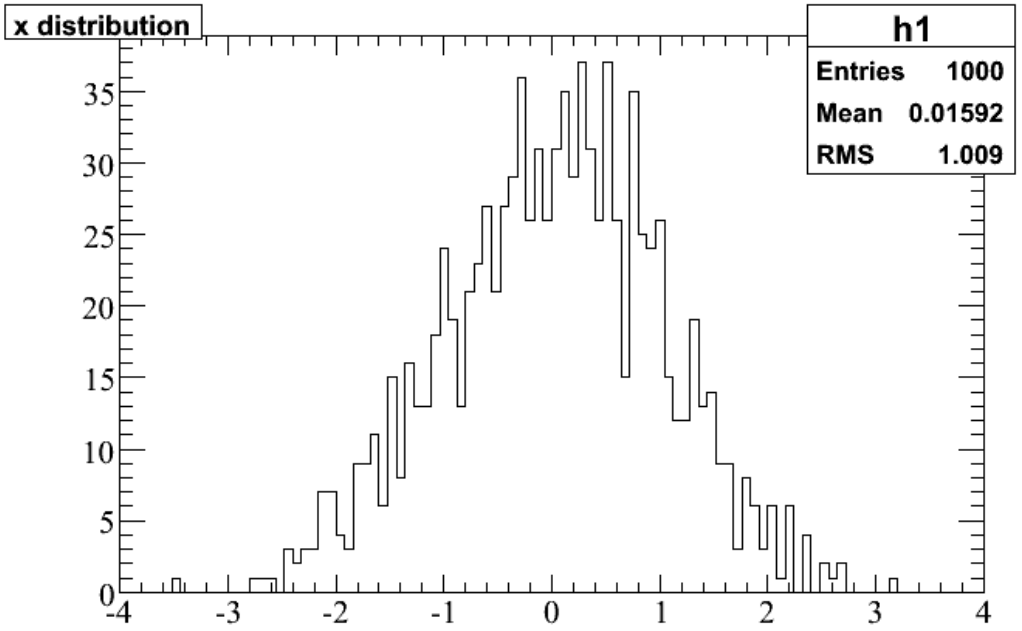
ROOT : Quick introduction- 2

1-Dimensional Histograms : TH1I,TH1F,TH1D

```
TH1F* name = new TH1F("name","Title", Bins, lowest bin, highest bin);
```

- **Example**:

```
TH1F* h1 = new TH1F("h1","x distribution",100,-4,4);
Float_t x = 0;
for(Int_t i=0; i<1000; i++){
    x=gRandom->Gaus(0,1);
    h1->Fill(x);
}
h1->Draw();
```



The complete list of Methods can be found in:
http://root.cern.ch/root/html/TH1F.html

Histograms

```
// using different constructors


TH1I* h1 = new TH1I("h1", "h1 title", 100, 0.0, 4.0);
TH2F* h2 = new TH2F("h2", "h2 title", 40, 0.0, 2.0,30, -1.5, 3.5);
TH3D* h3 = new TH3D("h3", "h3 title", 80, 0.0, 1.0,100, -2.0, 2.0,50, 0.0, 3.0);


// cloning a histogram
TH1F* hc = (TH1F*)h1->Clone("title of the cloned histogram");


// projecting histograms
// the projections always contain double values !


TH1D* hx = h2->ProjectionX(); // ! TH1D, not TH1F
TH1D* hy = h2->ProjectionY(); // ! TH1D, not TH1F
```

Histograms : A complex example

```
// histograms filled and drawn in a loop
void hsum() {

TCanvas *c1 = new TCanvas("c1","The HSUM example",200,10,600,400);
c1->SetGrid();


// Create some histograms.
TH1F *total  = new TH1F("total","This is the total distribution",100,4,4);
TH1F *main   = new TH1F("main","Main contributor",100,-4,4);
TH1F *s1     = new TH1F("s1","This is the first signal",100,-4,4);
TH1F *s2     = new TH1F("s2","This is the second signal",100,-4,4);

total->Sumw2();  // store the sum of squares of weights
total->SetMarkerStyle(21);
total->SetMarkerSize(0.7);
main->SetFillColor(16);
s1->SetFillColor(42);
s2->SetFillColor(46);
```

Macro: hsum.C

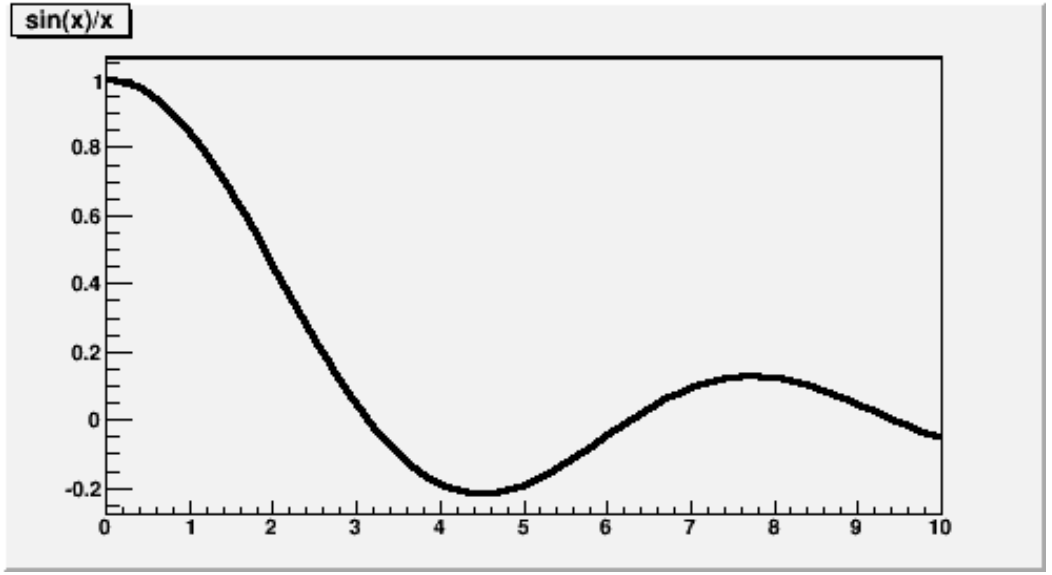
Histograms : A complex example

```
// Fill histograms randomly

    gRandom->SetSeed();

    Float_t xs1, xs2, xmain;
    for (Int_t i=0; i<10000; i++) {
        xmain = gRandom->Gaus(-1,1.5);
        xs1   = gRandom->Gaus(-0.5,0.5);
        xs2   = gRandom->Landau(1,0.15);
        main->Fill(xmain);
        s1->Fill(xs1,0.3);
        s2->Fill(xs2,0.2);
        total->Fill(xmain);
        total->Fill(xs1,0.3);
        total->Fill(xs2,0.2);
     }
  }
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
```

Macro: hSum.C

Formulas and Functions

```
void formula2(){
    TCanvas *c1 = new TCanvas("c1","Example with Formula",500,500);
    // Create a one dimensional function and draw it //
    TF1 * fun1 = new TF1("fun1","sin(x)/x",0,10);
    c1->cd(1);
    fun1->Draw();
    cout << " Deriv " << fun1->Derivative(2.) << endl;
    cout << " Integral " << fun1->Integral(0.,3.) << endl;
    cout << " Func val " << fun1->Eval(1.2456789) << endl;
  }
```



Macro: formula2.C

6

Creating a TF1 with Parameters

The second way to construct a **TF1** is to add parameters to the expression. Here we use two parameters:

```
root[] TF1 *f1 = new TF1("f1","[0]*x*sin([1]*x)",-3,3);
```

The parameter index is enclosed in square brackets. To set the initial parameters explicitly you can use:

```
root[] f1->SetParameter(0,10);
```

This sets parameter 0 to 10.
You can also use SetParameters to set multiple parameters at once.

```
root[] f1->SetParameters(10,5);
```

This sets parameter 0 to 10 and parameter 1 to 5.
We can now draw the **TF1**:

```
root[] f1->Draw();
```

Creating a TF1 with a User Function

```
// define a function with 3 parameters
Double_t fitf(Double_t *x,Double_t *par){
   Double_t arg = 0;
   if (par[2] != 0) arg = (x[0] - par[1])/par[2];
   Double_t fitval = par[0]*TMath::Exp(-0.5*arg*arg);
   return fitval;
}


void userfunctexample() {
   // Create a TF1 object using the function defined above. The last
   //  parameter specify the number of parameters for the function.
   TF1 * func = new TF1("fit",fitf,-3,3,3);
   // set the parameters to the mean and RMS of the histogram
   func->SetParameters(500,0.,0.5);
   // give the parameters meaningful names
   func->SetParNames ("Constant","Mean_value","Sigma");
   // call TH1::Fit with the name of the TF1 object
   func->Draw();
}
```


Macro: userfunctexample.C

Write and Read a ROOT file

- **Write (in a macro):**

```
TFile *myfile = new TFile("fillrandom.root","RECREATE");
myfile->cd();
form1->Write();
sqroot->Write();
h1f->Write();
myfile->Close();
```

- **Read (in a macro):**

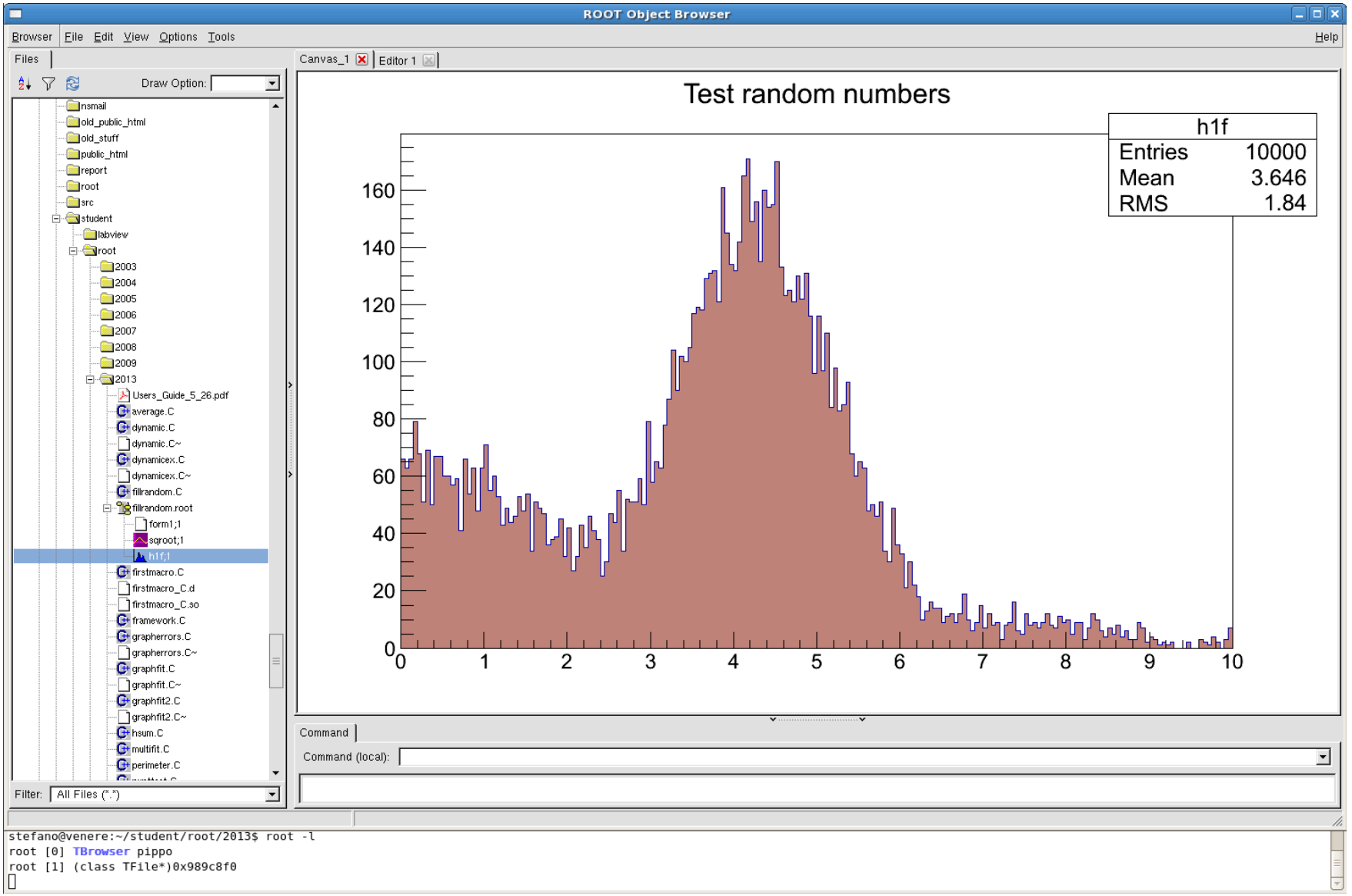
```
TFile *pfile = new TFile("fillrandom.root");
TH1F * ph1 = (TH1F *) pfile->Get("h1f"); // take the object from the TFile
```

- **Read (in the terminal):**

```
root [0] TFile *pfile = new TFile("fillrandom.root");
root [1] pfile->ls(); // it is useful only to know what is stored inside your
TFile if you have no idea of how it has been created.
root [2] TH1F * ph1 = (TH1F *) pfile->Get("h1f"); // take the object from the
TFile
```

TBrowser : A GUI to analyze the TFile

```
root [0] new TBrowser()
```



Passing arguments to a macro

```
// define a function with 3 parameters
Double_t fitf(Double_t *x,Double_t *par){
   Double_t arg = 0;
   if (par[2] != 0) arg = (x[0] - par[1])/par[2];
   Double_t fitval = par[0]*TMath::Exp(-0.5*arg*arg);
   return fitval;
}

void userfunctexample(Int_t min = -3, Int_t max = 3) {
   // Create a TF1 object using the function defined above. The last
   //  parameter specify the number of parameters for the function.
   TF1 * func = new TF1("fit",fitf,min,max,3);
   // set the parameters to the mean and RMS of the histogram
   func->SetParameters(500,0.,0.5);
   // give the parameters meaningful names
   func->SetParNames ("Constant","Mean_value","Sigma");
   // call TH1::Fit with the name of the TF1 object
   func->Draw();
   TString nameout = "Funz.";
   nameout+=min;
   nameout+=".";
   nameout+=".root";
   TFile *fo = new Tfile(nameout,"RECREATE");
   fo->cd();
   func->Write();
   fo->Close();
}
```

Read a TTree from a TFile

Example:

```
root[] TFile *f = new TFile("file.root");
root[] TNtuple *n = (TNtuple*)f -> Get("ntuple");
root[] n->MakeClass("MyClass");
```

MyClass.h and MyClass.C are created in your local directory

MyClass::Loop consists of a for-loop calling GetEntry for each entry.

Inside the Loop method you can add the histogram/function/fit you want add.

Once you implement your analysis objects, you have to run the code.

How to run the analysis:

```
root[] .L MyClass.C+
root[] MyClass m
root[] m.Loop()
```

Esercitazione 10

Esercizio

- The exercise is divided into 3 part:

  1)Processing data from a TTree, filling a histogram, and writing the results to an output file

  2)Reading a file that contains multiple histograms and interpreting the results, writing the final plots to a pdf file

  3)Reading a file that contains a histogram and fitting the histogram in different ways, writing the results to a pdf file

- Exercise 1:
  - Input files: Data.root, MC.root
    - Thanks to ATLAS and CERN Open Data for the data and simulation and Steven Schramm (Universite' de Geneve) for filter the data and prepare the exercise
  - Output ROOT file: selection.root
  - Solution code (ROOT): selection.cpp
- Exercise 2:
  - Input file: selection.root
  - Output plot file: analysis.pdf
  - Solution code (ROOT): analysis.cpp
- Exercise 3:
  - Input file: selection.root
  - Output plot file: fit.pdf
  - Solution code (ROOT): fit.cpp

Exercise 1: Processing data

- Create a macro the read the ".root" files (you case use the MakeClass function of the TTree object)
- Inside the macro create the histogram that we want to fill:
  - We want to look at the di-lepton invariant mass to reconstruct the Z boson invariant mass (M=90 MeV/c^2)
- Loop over the TTree entries
- Calculate the pairs of lepton four-vectors for each event and the di-lepton invariant mass:
  - Each time GetEntry is called, all of the variables we defined are overwritten with new values for the new event. We can now check that there are two leptons in the event and make their four-vectors, then use that to calculate the di-lepton invariant mass.
  - We have to check that there are actually two leptons, as we can't calculate a di-lepton mass if there is only one lepton in the event.
  - If there are not two leptons, then we should skip this event (continue to the next event).
  - After we have confirmed that there are two leptons, we can build their four-vectors and store them as TLorentzVector objects, which contain lots of useful functions.
  - TLorentzVector lepton0 , lepton1 ;

```
lepton0.SetPtEtaPhiE ( pT [0] , eta [0] , phi [0] , nrg [0]);
lepton1.SetPtEtaPhiE ( pT [1] , eta [1] , phi [1] , nrg [1]);
```

  - We now have four-vector representations of the two leptons. However, we want the di-lepton system, which is the four-vector sum of the two leptons. Then, we want to store the mass of that system.

```
// Previous code in the loop is here
TLorentzVector dilepton = lepton0 + lepton1 ;
double dileptonMass = dilepton . M ();
```

  - Fill the histogram

Exercise 1: Processing data

- Write the histogram to the output file and then close the output file, which is what triggers it to be written to disk rather than living in memory:

```
outHistFile -> Close();
```

- Check the output file using the `TBrowser`
- NOTE- this is the end of the main part of this exercise, while the below is adding more complex functionality. If you are short on time, you can download the file named “selection.root” and use that for the next two exercises.
  - Extend the program to handle both data and MC
  - Normally in physics, we want to compare our data with our simulation, representing the expectation
  - For example, is the peak we saw in the last part representative of the Z, or is it too small or too large?
  - To do this, we need our program to be able to run over both data and MC
- When you run over MC, some special scaling factors are needed:
  - the MC TTree has some more extra information that can be used:

```
tree -> SetBranchAddress ( "mcWeight "           ,& mcWeight );
tree -> SetBranchAddress ( "scaleFactor_PILEUP"  ,& sf_PILEUP );
tree -> SetBranchAddress ( "scaleFactor_MUON"    ,& sf_MUON );
tree -> SetBranchAddress ( "scaleFactor_TRIGGER" ,& sf_TRIGGER );
```

- We then have to calculate the total weight to apply, which is just the product of these four values.

```
weight = mcWeight * sf_PILEUP * sf_MUON * sf_TRIGGER ;
```

- Then, when we are filling the histogram, apply this weight:

```
mll->Fill( dileptonMass , weight );
```

- Now your MC has been corrected to more correctly model the expected data distribution.

Exercise 2-Data analysis – plotting

- Open the input histogram file(s) for reading
- Retrieve the two histograms from the file

```
TH1D * dataHisto = ( TH1D *) histFile - > Get ( " data " );
TH1D * mcHisto = ( TH1D *) histFile - > Get ( " MC " );
```

- Plot the two histograms in two pads of a canvas
- Normalize the MC histogram
  - If you look at the plots, you will see that the y-axis has a very different range. The y-axis of the MC plot seems to be roughly 7 orders of magnitude larger than the data y-axis!
  - There are several reasons for this, but for now, the main point is that this is an artificial difference. As such, We want to normalize the MC histogram to data to remove this offset.
  - We don't want to normalize it in a way that makes the distributions agree bin-by-bin, as they we could never search for new physics or measure the standard model properties. Instead, we want to just get the overall scale correct.
  - One way to do this is to scale by the total number of events in data vs the number of events in MC. This is done using the integral of the histograms (which is the total number of events, as the y-axis is just the number of events in a given bin).

```
mcHisto -> Scale ( dataHisto -> Integral()/ mcHisto -> Integral());
```

  - Alternatively, you can decide to normalize the two histograms is a specific mass region
- After the normalization, draw the data and MC on the same plot in a new canvas

Exercise 2-Data analysis – plotting

- Make a split-panel plot:
- First, let's prepare by creating a histogram which is the ratio of the data divided by the MC. Do this by creating a copy of the data histogram (to avoid modifying the original one) and dividing that histogram by the MC.

```
TH1D * ratio = ( TH1D *) dataHisto - > Clone ();
ratio ->Divide ( mcHisto );
ratio -> SetLineColor ( kRed );
```

- On a canvas create a TPad (a sub-canvas) with a horizontal extent of the full range (0, y, 1, y) and a vertical extend of the upper 70% of the range (x, 0.3, x, 1). That is, this sub-canvas should be the upper 70% of the full canvas.
- Set the sub-canvas to have a logarithmic y-axis, draw the sub-canvas on the canvas, set the sub-canvas to be the owner of whatever is drawn next, and then draw the MC and data histograms (not their ratios) on this upper sub-canvas.

```
TPad pad1 ( " pad1 " ," pad1 " ,0 ,0.3 ,1 ,1);
pad1.SetLogy( true );
pad1.Draw();
pad1.cd();
mcHisto -> Draw( " h " );
dataHisto -> Draw( " pe , same " );
```

- Now, go back to the full canvas and make a second sub-canvas covering the bottom 30% of the plot, and then draw the ratio on that sub-canvas. Note that this sub-canvas is a linear scale, not logarithmic, as this is a ratio.

```
canvas.cd ();
TPad pad2( "pad2" ,"pad2" ,0 ,0.05 ,1 ,0.3);
pad2.Draw();
pad2.cd();
ratio->Draw( "pe" );
canvas.SaveAs( "plot.pdf");
```

Exercise 2-Data analysis – plotting

- Change the pad spacing to better use canvas space
  - First, remove the border spacing on the bottom of the top pad

```
pad1.SetBottomMargin (0);
```

- Now, remove the border spacing on the top of the bottom pad.
- Also add a bit of extra space on the bottom of the top pad so that we have some space for later.

```
pad2.SetTopMargin(0);
pad2.SetBottomMargin(0.25);
```

- Once this is done, the upper and lower plots touch, which makes sense as you only need to have one x-axis label for the two plots (they are the same axis).
- Adjust the size and contents of axis labels and markers: First, get rid of the x-axis labels on the top plot, this removing the random ×10 3 on the far right of the plot

```
mcHisto -> SetTitle( " " );
mcHisto -> GetXaxis() -> SetLabelSize(0);
mcHisto -> GetXaxis() -> SetTitleSize(0);
```

- Then increase the size of the y-axis label, which is necessary because the pad is only 70% of the full vertical range, so we need to increase the text size to get back to a reasonable value.

```
mcHisto - > GetYaxis () - > SetTitleSize (0.05);
```

- Now do the similar things to the bottom panel, also increasing the text sizes everywhere as this is now only 30% of the full vertical range. Note that the title offset is changed at one point to bring the label closer to the axis (so it doesn't go off the canvas).

```
ratio -> SetTitle( "" );
ratio -> GetXaxis() -> SetLabelSize(0.12);
ratio -> GetXaxis() -> SetTitleSize(0.12);
ratio -> GetYaxis() -> SetLabelSize(0.1);
ratio -> GetYaxis() -> SetTitleSize(0.15);
ratio -> GetYaxis() -> SetTitle( " Data / MC " );
ratio -> GetYaxis() -> SetTitleOffset(0.3);
```

Exercise 2-Data analysis – plotting

- To guide the eye, it is useful to add a flat line at 1 on ratio plots. This is done using the TLine object, where you specify the (x start , y start , x end , and y end) coordinates of the line.
  - Note that this should be created after you draw the ratio histogram. That is, we are adding a line on top of the ratio plot.

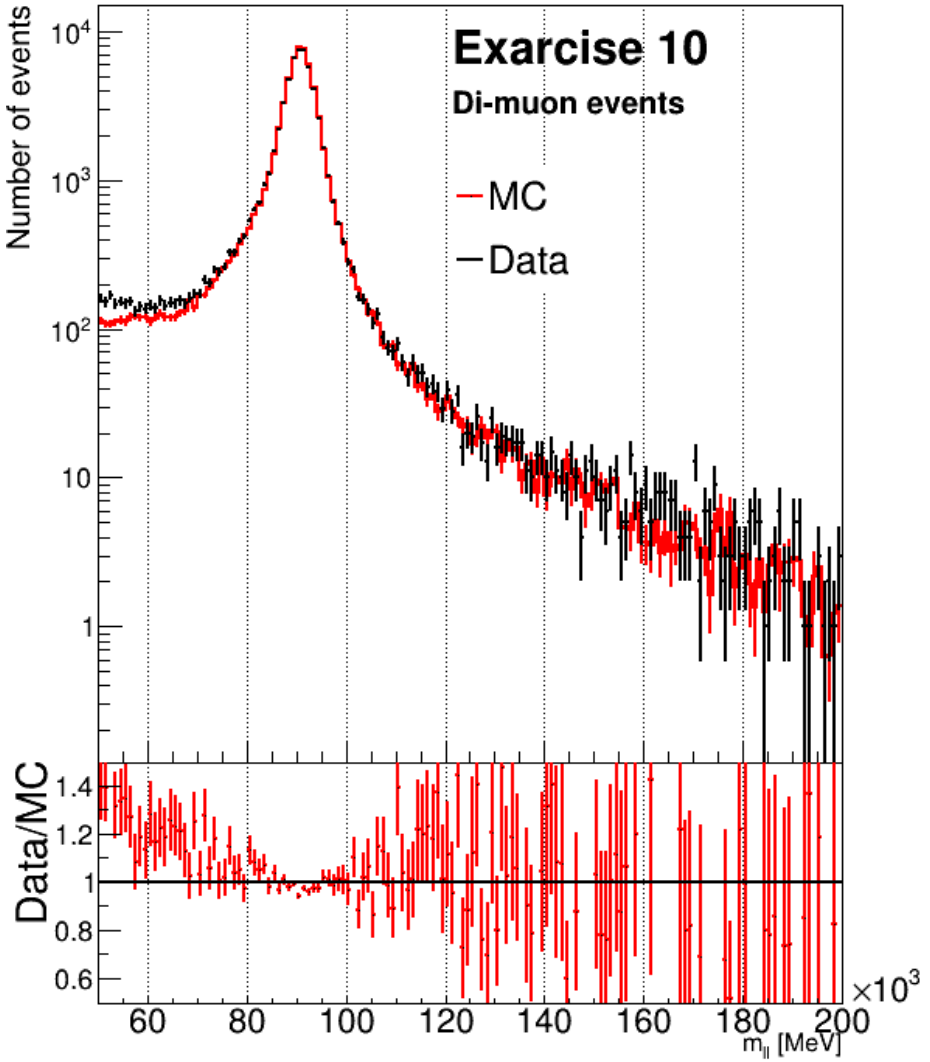
```
TLine line (50. e3 ,1 ,200. e3 ,1);
line.SetLineColor ( kBlack );
line.SetLineWidth (2);
line.Draw ( " same " );
```

- Now when you look at the data, you can see that there is generally agreement for di-lepton masses above roughly 70 GeV (70×10^3 MeV) within the statistical precision. However, there is a significant difference at lower mass values. This is because the MC sample we have used is actually only for one process, $Z \rightarrow ll$, and is thus ignoring the Standard Model $\gamma \rightarrow ll$ process. That is why data is above MC, namely we are neglecting a real process present in data when calculating our MC expectation.

Exercise 2-Data analysis – plotting

- Add a legend to the plot:
  - Right now, we know that the data is black and the MC is red. However, if we show this plot to someone else, they won't know this. It is thus very important to add legends to all of your plots.
  - Legends should be added after drawing all of the relevant histograms, adding a legend on top of the plot.
  - We also set the width of the line around the box to 0 to remove it.
  - Note that the coordinates that you use to specify the legend location are pad-global coordinates. That means that 0 is the bottom/left of the pad, 1 is the top/right of the pad, and any point in between is a fractional value. These numbers are relevant to the pad it is being drawn on, not the full canvas.
  - Add other labels to the plot:
    - It is often useful to add additional labels to plots. This can be done using TLatex, which allows you to write text at arbitrary locations on the plot.
    - Note that we use the SetNDC() function to specify that we want to use global coordinates, not specific points of a given plot.
    - In this case, you only need to specify the starting x and y coordinates, not also the ending coordinates.
  - Labels should be added after everything else, before writing the final plot to the output pdf file.

Example


Exarcise 10
Di-muon events
MC
Data
Number of events
10^4
10^3
10^2
10
1
Data/MC
1.4
1.2
1
0.8
0.6
60
80
100
120
140
160
180
200
$\times 10^3$
m_{ll} [MeV]

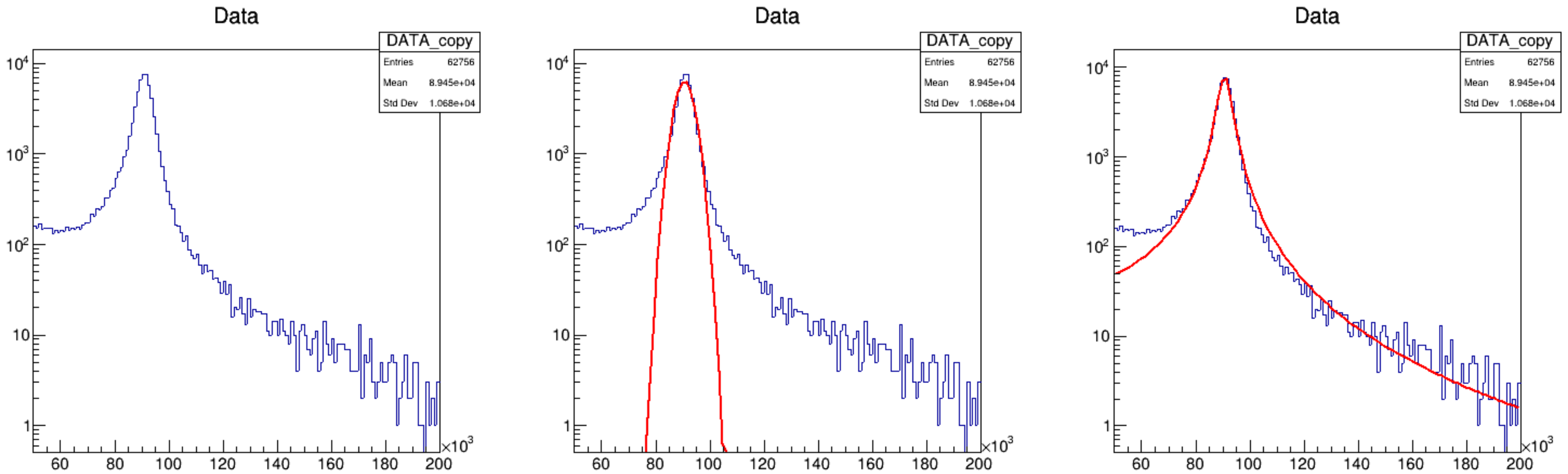
Exercise 3-Data analysis – fitting

- Create the basic structure of a ROOT macro, open the file, read the data histogram, and close the file
- Create the TCanvas and prepare the output file
- Plot the basic histogram as our starting point
- Fit a Gaussian to the peak: fitting Gaussians in ROOT is remarkably easy, and this is a very common task.
  - The Gaussian function is pre-defined in ROOT. Create a TF1 (T for ROOT, F for function, 1 for 1-dimensional) as below.

```
TF1 gaussFit ( " gaussfit " ," gaus " ,81. e3 ,101. e3 );
```

- Fit a Breit-Wigner to the peak
- If you have managed to complete everything so far, use what you learned from exercise 2 to make a split-panel plot of the data and its fit in the top panel, and the ratio of the data to its fit in the bottom panel.

Example


Data
DATA_copy
Entries 62756
Mean 8.945e+04
Std Dev 1.068e+04
10^4
10^3
10^2
10
1
60 80 100 120 140 160 180 200
$\times 10^3$
Data
DATA_copy
Entries 62756
Mean 8.945e+04
Std Dev 1.068e+04
10^4
10^3
10^2
10
1
60 80 100 120 140 160 180 200
$\times 10^3$
Data
DATA_copy
Entries 62756
Mean 8.945e+04
Std Dev 1.068e+04
10^4
10^3
10^2
10
1
60 80 100 120 140 160 180 200
$\times 10^3$

Breit-Wigner

```
//Breit-Wigner function
Double_t mybw(Double_t* x, Double_t* par)
{
  Double_t arg1 = 14.0/22.0; // 2 over pi
  Double_t arg2 = par[1]*par[1]*par[2]*par[2]; //Gamma=par[1]  M=par[2]
  Double_t arg3 = ((x[0]*x[0]) - (par[2]*par[2]))*((x[0]*x[0]) - (par[2]*par[2]));
  Double_t arg4 = x[0]*x[0]*x[0]*x[0]*((par[1]*par[1])/(par[2]*par[2]));
  return par[0]*arg1*arg2/(arg3 + arg4);
}
```