Introduction to ROOT: final part

Mirco Dorigo
mirco.dorigo@ts.infn.it

LACD 2022-2023
March 30th, 2023

Let’s make the fit of the data

- We have seen the possible function to fit each component

- In data, we have 10% of the statistic of the simulation:
  we can afford also simpler model.
  We will use just one Gaussian function to model the signal and the mis-id component

- We will build a model that is the sum of the 3 components

- We will do it in a macro (although we could do it online too!)

Take `fitDeltaE.C`

First part pretty standard now…

```cpp
#include "TLegend.h"
#include "TStyle.h"

using namespace std;

void fitDeltaE(){

    const double min_de = -0.15;
    const double max_de =  0.15;
    //define an histogram to look at deltaE distribution
    TH1D* h_data = new TH1D("h_data",";#DeltaE [GeV]; Entries",40,min_de,max_de);

    //open file and take the tree
    TFile* file = TFile::Open("data.root");
    TTree* tree = (TTree*) file->Get("treeData");

    int tot_entries = tree->GetEntries();
    cout << "Total entries in the tree: " << tot_entries << endl;

    //link the variables with tree banches
    double B_de;
    double bkg_killer;
    tree->SetBranchAddress("B_de",&B_de);
    tree->SetBranchAddress("bkg_killer",&bkg_killer);

    //loop over the entries
    for(int iEntry; iEntry<tot_entries; ++iEntry){

        tree->GetEntry(iEntry);

        //skip all candidates below the optimal cut point
        if(bkg_killer<0.92) continue;

        //fill the histograms
        h_data->Fill(B_de);
    }
```

Take `fitDeltaE.C`

```
    //Let's define the PDF for the fit, using TF1
    //https://root.cern.ch/doc/master/classTF1.html

    //The total function that describes our observed distribution
    TF1* pdf = new TF1("pdf","gaus(0)+gaus(3)+pol1(6)",min_de,max_de);

    //signal gauss, normalisation constant
    pdf->SetParName  (0,  "N_{sig}");
    pdf->SetParameter(0,  100);
    //signal gauss, mean fixed
    pdf->SetParName  (1,  "#mu_{sig}");
    pdf->FixParameter(1,   0.);
    //signal gauss, std dev fixed
    pdf->SetParName  (2,  "#sigma_{sig}");
    pdf->FixParameter(2,0.015);
    //mis-id gauss, normalisation constant
    pdf->SetParName  (3,  "N_{misid}");
    pdf->SetParameter(3,   10);
    //mis-id gauss, mean fixed
    pdf->SetParName  (4,  "#mu_{misid}");
    pdf->FixParameter(4,0.042);
    //mis-id gauss, std dev fixed
    pdf->SetParName  (5,  "#sigma_{misid}");
    pdf->FixParameter(5,0.015);
    //background intercept and slope
    pdf->SetParName  (6,  "p_{0}^{bkg}");
    pdf->SetParName  (7,  "p_{1}^{bkg}");
```

TF1 function

All settings on parameters.
We fix parameters that
we know already
(from physics or simulation)
to ease the work of the fit.
The simplest the model,
the better.

Take `fitDeltaE.C`

- It’s all happening here with a very simple line!

```
//and now fit, in the range defined by the histogram (option R)
//option N = not draw (otherwise it draws a canvas with a plot by default)
cout << "\n First fit, fixing all possible parameters: \n\n";
h_data->Fit("pdf","RN");
```

- But plenty of options to do whatever we need…
- See the method `Fit()` in the reference guide.
- Note: `Fit()` works also for `TGraph(Errors)`.

Take `fitDeltaE.C`

Value of the fit function (χ^2 here)

Algorithm used to obtain the results

Important to check this!

```
First fit, fixing all possible parameters:
FCN=27.8948 FROM MIGRAD    STATUS=CONVERGED      79 CALLS           80 TOTAL
                     EDM=9.01287e-23    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  N_{sig}      1.69595e+02   7.30230e+00   1.84336e-02   9.63651e-13
  2  #mu_{sig}    0.00000e+00     fixed
  3  #sigma_{sig}   1.50000e-02     fixed
  4  N_{misid}    4.55419e+01   5.18582e+00   1.26809e-02  -7.00404e-13
  5  #mu_{misid}   4.20000e-02     fixed
  6  #sigma_{misid}   1.50000e-02     fixed
  7  p_{0}^{bkg}   4.17642e+01   1.24868e+00   2.98156e-03   8.93671e-12
  8  p_{1}^{bkg}  -7.71166e+01   1.19357e+01   3.07475e-02  -1.15545e-13
```

The fit results

Take `fitDeltaE.C`

- Can play with parameters, to obtain more information from data

```
    cout << "\n\n Let's try to release the signal std dev \n\n";
    pdf->ReleaseParameter(2); //signal gauss, std dev fixed
    h_data->Fit("pdf","RN");

    cout << "\n\n Update the mis-id std dev \n";
    cout << " And release also the mis-id mean \n";
    pdf->FixParameter(5, pdf->GetParameter(2)); //signal gauss, std dev fixed
    pdf->ReleaseParameter(4);
    //option L = binned likelihood fit
    cout << " and do a binned-likelihood fit, instead of a chi2 \n\n";
    h_data->Fit("pdf","LR");
```

- Can try also different fit methods, so in the last iteration we ask to fit with a binned-likelihood function, instead of the default χ^2

Take `fitDeltaE.C`

```
Let's try to release the signal std dev

FCN=27.5849 FROM MIGRAD    STATUS=CONVERGED     110 CALLS          111 TOTAL
                    EDM=4.36718e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME       VALUE            ERROR          SIZE      DERIVATIVE
  1  N_{sig}       1.66753e+02   8.83060e+00   1.79772e-02  -3.65160e-05
  2  #mu_{sig}     0.00000e+00      fixed
  3  #sigma_{sig}   1.54440e-02   8.09241e-04   1.50628e-06  -3.72362e-01
  4  N_{misid}     4.45076e+01   5.52977e+00   1.26128e-02  -3.30067e-06
  5  #mu_{misid}    4.20000e-02      fixed
  6  #sigma_{misid}   1.50000e-02      fixed
  7  p_{0}^{bkg}    4.16391e+01   1.26990e+00   2.96553e-03  -8.98781e-05
  8  p_{1}^{bkg}  -7.64094e+01   1.20082e+01   3.05822e-02  -9.12170e-06
```

2nd fit results, releasing the sigma for the signal

```
Update the mis-id std dev
And release also the mis-id mean
and do a binned-likelihood fit, instead of a chi2

Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
FCN=13.7672 FROM MIGRAD    STATUS=CONVERGED     148 CALLS          149 TOTAL
                    EDM=5.36238e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME       VALUE            ERROR          SIZE      DERIVATIVE
  1  N_{sig}       1.66476e+02   8.69389e+00   1.78671e-02  -3.46486e-06
  2  #mu_{sig}     0.00000e+00      fixed
  3  #sigma_{sig}   1.56044e-02   8.92083e-04   1.50313e-06   3.01092e-02
  4  N_{misid}     4.26560e+01   5.59930e+00   1.22986e-02   1.88504e-05
  5  #mu_{misid}    4.37828e-02   2.89346e-03   6.25797e-06   6.01764e-02
  6  #sigma_{misid}   1.54440e-02      fixed
  7  p_{0}^{bkg}    4.22155e+01   1.34014e+00   2.98857e-03   1.67546e-04
  8  p_{1}^{bkg}  -7.82222e+01   1.20763e+01   3.05956e-02   4.01627e-06
                              ERR DEF= 0.5
```

3rd fit results, releasing also mis-id mean. Use the binned likelihood here.

Take `fitDeltaE.C`

```
    //draw the result
    gStyle->SetOptStat(0);
    gStyle->SetOptFit(1111);
    TCanvas* c1 = new TCanvas("c1","c1",600,600);

    h_data->SetMinimum(0);
    h_data->SetMarkerColor(kBlack);
    h_data->SetMarkerStyle(8);
    h_data->SetMarkerSize(0.8);
    h_data->SetLineColor(kBlack);

    h_data->Draw("err");

    //just to draw each component separately...
    //the signal
    TF1* pdf_sig = new TF1("pdf_sig","gaus",min_de,max_de);
    pdf_sig->SetParameters(pdf->GetParameter(0),
                           pdf->GetParameter(1),
                           pdf->GetParameter(2));
    pdf_sig->SetLineColor(kRed);
    pdf_sig->SetLineWidth(2);
    pdf_sig->Draw("same");

    //the mis-id B->pipi
    TF1* pdf_misid = new TF1("pdf_misid","gaus",min_de,max_de);
    pdf_misid->SetParameters(pdf->GetParameter(3),
                             pdf->GetParameter(4),
                             pdf->GetParameter(5));
```

Just nice drawing of the results…

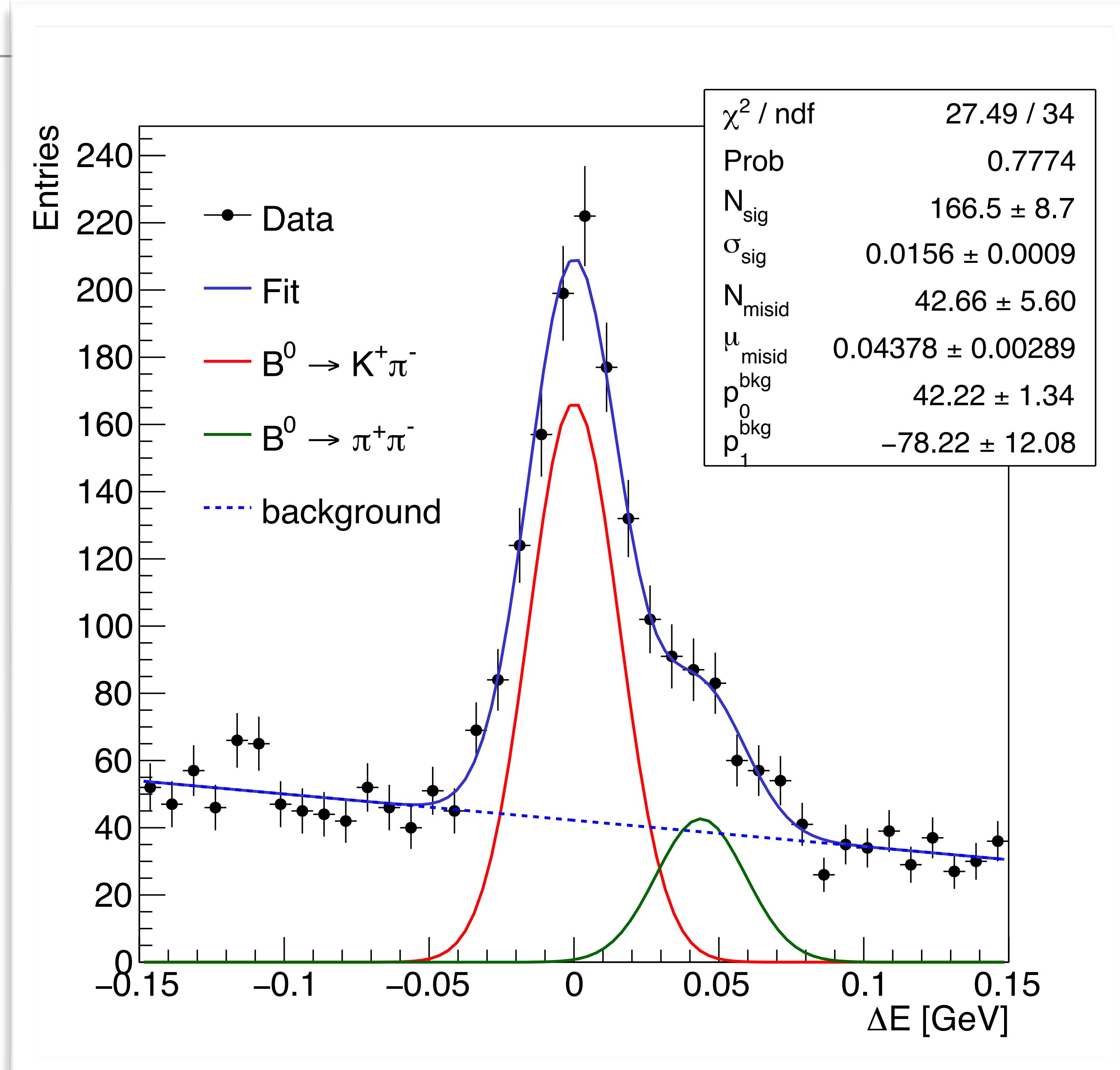
Take `fitDeltaE.C`

```
    c1->SaveAs("myFit.pdf");
    c1->SaveAs("myFit.C");

    //Get now what we wanted to know!
    double binW = h_data->GetXaxis()->GetBinWidth(1);
    cout << "\n\n From this fit model, \n";
    cout << "Candidate in data histogram: " << h_data->Integral() << endl;
    cout << "Total candidates from fit  : " << pdf->Integral(min_de,max_de)/binW << endl;
    cout << "Signal B->Kpi candidates   : " << pdf_sig->Integral(min_de,max_de)/binW << endl;
    cout << "Mis-id B->pipi candidates  : " << pdf_misid->Integral(min_de,max_de)/binW << endl;
    cout << "Background candidates      : " << pdf_bkg->Integral(min_de,max_de)/binW << endl;

    return;
}
```

Can also save the canvas to a pdf!

Retrieve the information we want: the yield of the components

We made it!




```
 From this fit model,
Candidate in data histogram: 2777
Total candidates from fit   : 2777.01
Signal B->Kpi candidates    : 868.211
Mis-id B->pipi candidates   : 220.176
Background candidates       : 1688.62
```

Calculate S and its uncertainty

- We didn't compute the uncertainty on the signal yield yet!

- We used a gauss pdf for the signal, its integral (divided by the bin width w) gives the signal yield:

$$\mathrm{pdf} = N e^{-\frac{(x-\mu)^2}{2\sigma^2}} \rightarrow S = N\sqrt{2\pi}\sigma/w$$

- To get the uncertainty on S, need to propagate the uncertainty from the fit on N and σ, considering their correlation.

Calculate S and its uncertainty

- Little addition in fitDeltaE.C

```
//option L = binned likelihood fit
//Use FitResultPtr to retreive all information about the fit
//Need to add the option S
TFitResultPtr fit = h_data->Fit("pdf","LRS");
//now we can get covariance matrix. We will store in a TMatrixDSym
TMatrixDSym cov = fit->GetCovarianceMatrix();
cov.Print();
```

- Now you have all information

```
cout << "======================================================" << endl;
cout << " Let's calculate the final result with its uncertainty " << endl;
//We can calculate the signal yield directly from the fit parameters
//of the signal pdf (gauss function)
double Nsig  = fit->Parameter(0);
double sigma = fit->Parameter(2);
double S = Nsig*sqrt(2.*TMath::Pi())*sigma/binW;

// and propagate the uncertainty
double errS = S * sqrt(cov(0,0)/Nsig/Nsig + cov(2,2)/sigma/sigma + 2*cov(0,2)/Nsig/sigma);

printf("The measurement of the signal yield is %.0f +- %.0f \n", S, errS);
//for curiosity let's print the correlation between Nsig and sigma,
//is it negligible?
printf("Corr(Nsig,sigma) = %.3f \n\n",cov(0,2)/sqrt(cov(0,0)*cov(2,2)));
```

Calculate S and its uncertainty

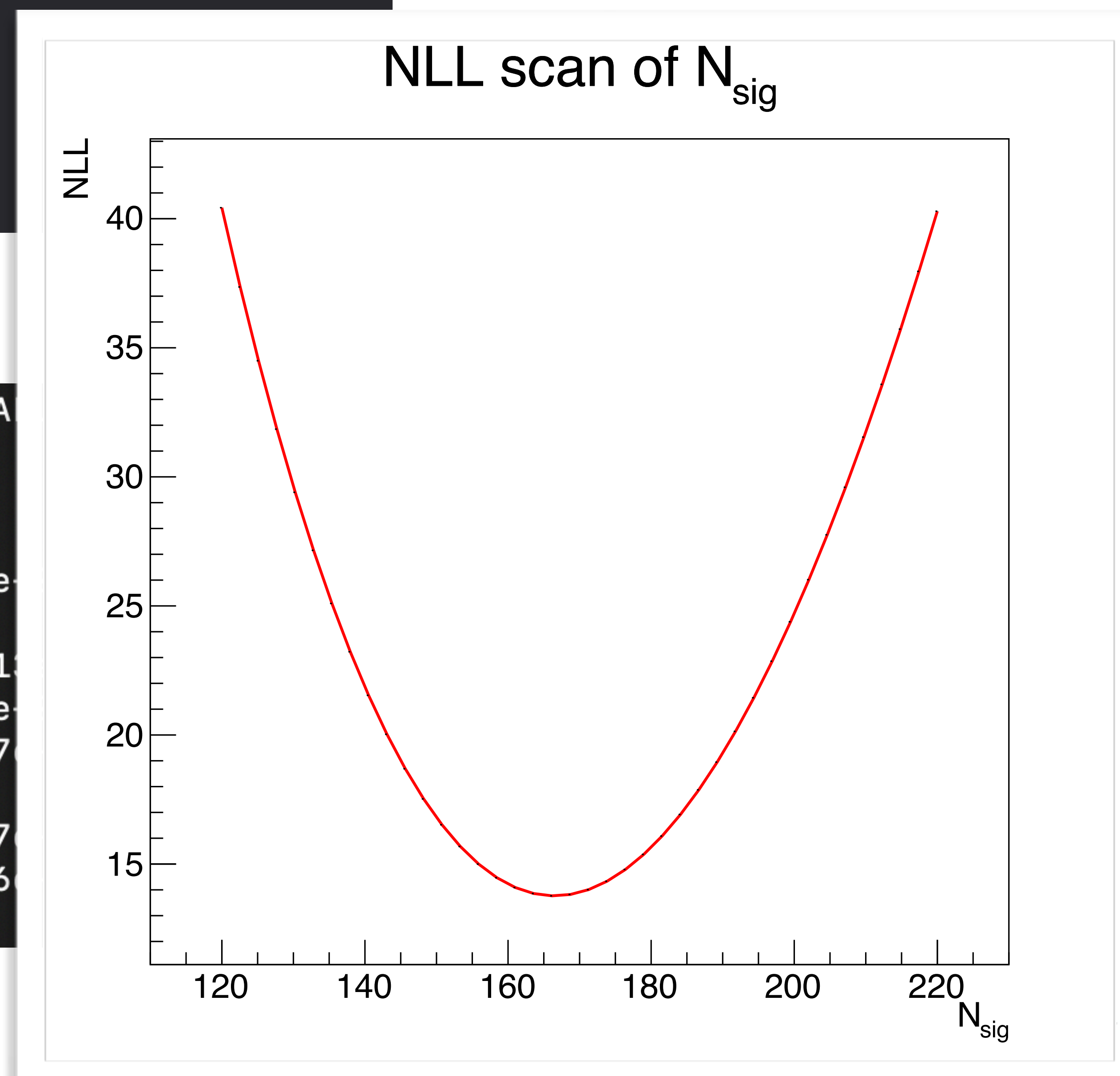
```
 From this fit model,
Candidate in data histogram: 2777
Total candidates from fit  : 2777.01
Signal B->Kpi candidates   : 868.211
Mis-id B->pipi candidates  : 220.176
Background candidates      : 1688.62
====================================================
 Let's calculate the final result with its uncertainty
The measurement of the signal yield is 868 +- 47
Corr(Nsig,sigma) = -0.511
```

FCN (χ^2 or likelihood) scan

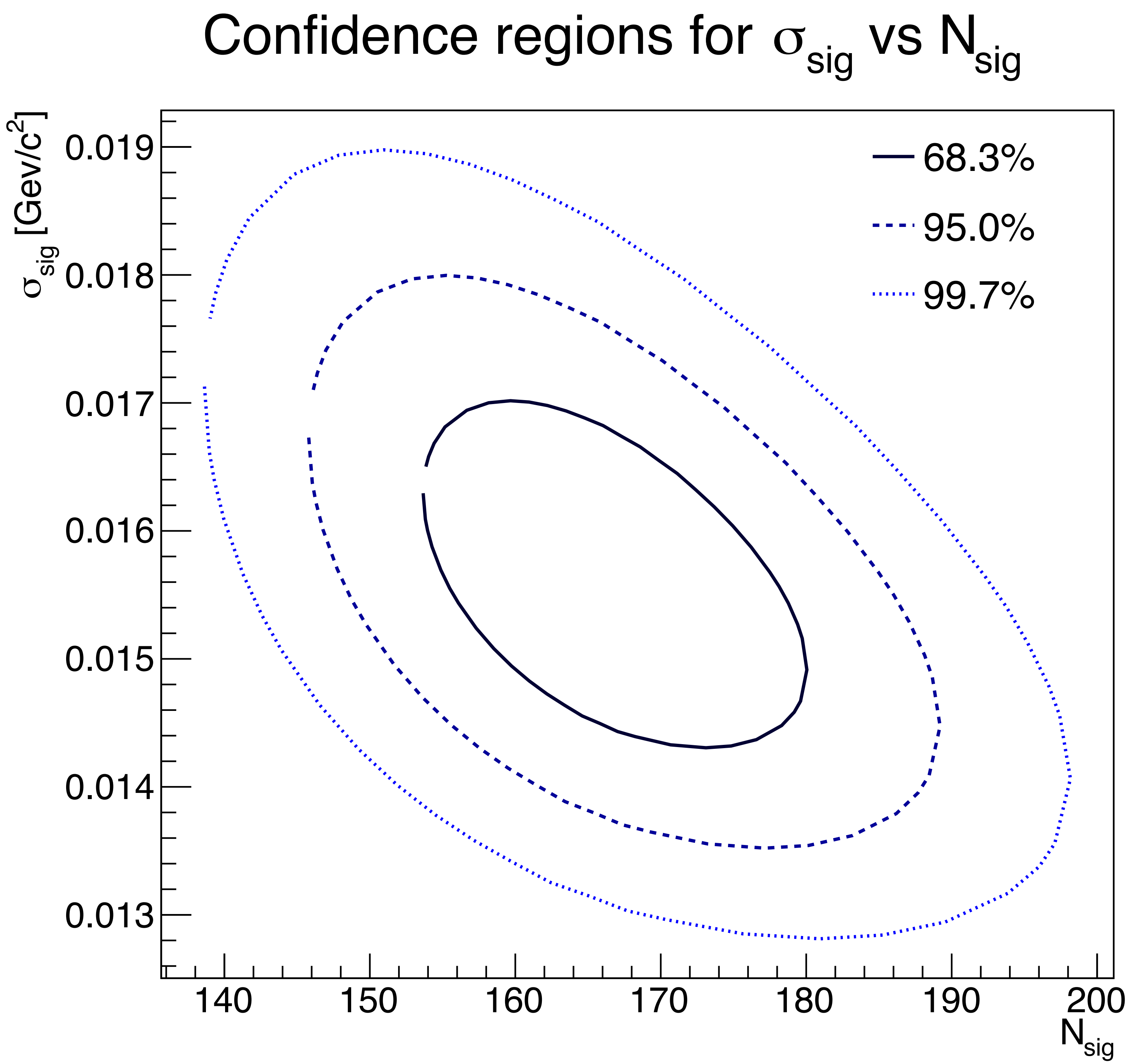
```
//Let's make a scan of the likelihood
//as a function of N_sig
TGraph* scan = new TGraph(40);
fit->Scan(0,scan,120,220);
TCanvas* c3 = new TCanvas("c3","c3",600,600);
scan->SetTitle("NLL scan of N_{sig}; N_{sig}; NLL");
scan->SetLineColor(kRed);
scan->SetLineWidth(2);
scan->Draw("APL");
c3->SaveAs("myScan.pdf");
```

NB: this is not a likelihood profile

```
FCN=13.7672 FROM MIGRAD    STATUS=CONVERGED     148 CA
                    EDM=5.36238e-08    STRATEGY= 1
 EXT PARAMETER                                STEP
 NO.   NAME       VALUE          ERROR        SIZE
  1  N_{sig}       1.66476e+02   8.69389e+00   1.78671e
  2  #mu_{sig}     0.00000e+00     fixed
  3  #sigma_{sig}    1.56044e-02   8.92083e-04   1.50313
  4  N_{misid}     4.26560e+01   5.59930e+00   1.22986e
  5  #mu_{misid}    4.37828e-02   2.89346e-03   6.25797
  6  #sigma_{misid}   1.54440e-02     fixed
  7  p_{0}^{bkg}    4.22155e+01   1.34014e+00   2.98857
  8  p_{1}^{bkg}   -7.82222e+01   1.20763e+01   3.05956
                              ERR DEF= 0.5
```



Confidence regions



Confidence regions

```
//Let's have a look: make 2D confidence regions
//1sigma contour: region enclosing 68.3% probability
TGraph* cont1sigma = new TGraph(50);
fit->Contour(0,2,cont1sigma,0.683);
cont1sigma->SetLineWidth(2);
cont1sigma->SetLineColor(kBlue+4);

//2sigma contour: region enclosing 95.5% probability
TGraph* cont2sigma = new TGraph(50);
fit->Contour(0,2,cont2sigma,0.955);
cont2sigma->SetLineStyle(2);
cont2sigma->SetLineWidth(2);
cont2sigma->SetLineColor(kBlue+2);

//3sigma contour: region enclosing 99.7% probability
TGraph* cont3sigma = new TGraph(50);
fit->Contour(0,2,cont3sigma,0.997);
cont3sigma->SetLineStyle(3);
cont3sigma->SetLineWidth(2);
cont3sigma->SetLineColor(kBlue);

//Draw all together, need to use TMultiGraph
TCanvas* c2 = new TCanvas("c2","c2",600,600);
TMultiGraph *mg = new TMultiGraph();
mg->SetTitle("Confidence regions for #sigma_{sig} vs N_{sig}; N_{sig}; #sigma_{sig} [Gev/c^{2}]");
mg->Add(cont1sigma,"l");
mg->Add(cont2sigma,"l");
mg->Add(cont3sigma,"l");
mg->Draw("a");
```

We made it!

- Congratulations for completing your (1st?) analysis with ROOT
- Hope this tour with a real-life example was useful (and also more interesting than a standard tutorial).
- Take your time to revisit all material and try it yourself.
For questions, doubts, curiosity don't hesitate to contact me.
We can organise a Q&A session too.
- If you are into data analysis at a new particle physics experiment, come to talk about opportunities in Belle II.

Exercises

- Write a macro that fit each single component of the sample (signal, mis-id, background).
- Try to fit also the M distribution of each component separately using simulation. Note: the background model is not easy, can discard it at the moment.
- If you were to fit the M distribution in data, do you think you can determine both signal and mis-id separately?

More exercises

- Make the likelihood scan for σ_{sig}
- Check “a posteriori” that the cut bkg_killer>0.92 is the optimal cut. What happen if you apply a tighter cut (>0.98) or a looser cut (>0.8)?
- Couldn’t we make the optimisation of the cut by fitting the data directly?

Exercises (3rd lesson)

1. Compute the signal efficiency, $\epsilon = S(\text{selected})/S(\text{total})$, for each cut `bkg_killer`. Draw a graph to show the efficiency as a function of the cut value, drawing also the error on the efficiency (that you need to calculate): use the class TGraphErrors.

2. What do you expect for the M distribution of the mis-id background? Draw it, by subtracting from the total distribution the signal and that of the non-B background (like we did for ΔE). Compare its distribution with that of the signal.

3. There is a variable `K_pid` in the tuples that gives the probability of a candidate kaon to be a real kaon. Draw its distribution: compare that of the signal (`isSig==1`) with that of the mis-id background (`isSig!=1 && isBkg!=1`).

4. Instead of using `DrawNormalized()`, scale to 1 the histogram integral using the `Scale()` method of `TH1` (check the integral value) and normal `Draw()` method.

Exercises (3rd lesson)

5. Find a cut value for `K_pid`, by maximising the $S/\sqrt{S+B}$, where S and B are the signal and mis-id background in the ΔE region $[-60,60]\,\mathrm{MeV}$.

6. Apply the full selection to the simulation and data samples (`data.root`), and draw the resulting distributions of M and ΔE.

NB: make sure all numbers and text in plots are well visible, by adjusting size of fonts, labels…

Exercise 1.

1. Compute the signal efficiency, $\epsilon = S(\text{selected})/S(\text{total})$, for each cut `bkg_killer`. Draw a graph to show the efficiency as a function of the cut value, drawing also the error on the efficiency (that you need to calculate): use the class TGraphErrors.

We will take `optimiseSelection.C` from the lesson and modify it.

Breaking down Exercise 1

```
#include "Riostream.h"
#include "TFile.h"
#include "TTree.h"
#include "TCanvas.h"
#include "TH1D.h"
#include "TGraph.h"

using namespace std;

void calculateEff(){

    //define the number of cuts to probe,
    //the range and the steps width
    const int ncuts = 15;
    double max_range = 1;
    double min_range = 0.7;
    double delta_cut = (max_range-min_range)/ncuts;

    //define the graph of the efficiency,
    //using TGraphErrors becasue I aslo want to
    //show the error on the efficiency
    TGraphErrors* g_eff = new TGraphErrors(ncuts);

    //Open file and take the tree
    TFile* file = TFile::Open("./simulation.root");
    TTree* tree = (TTree*) file->Get("simTree");
```

TGraphErrors class

Breaking down Exercise 1

```
    int tot_entries = tree->GetEntries("isBkg!=1");
    cout << "Total signal entries in the tree: " << tot_entries << endl;

    for(int icut=0; icut<ncuts; ++icut){

        //define the cut value to probe
        double cutval = min_range + icut*delta_cut;

        //put the cut in a string
        TString cutString = Form("bkg_killer > %.4f && isBkg!=1", cutval);

        //and retrieve the entries, directly from the tree, passing the selection
        double Nsig = tree->GetEntries(cutString);

        //calculate the efficiency and the error
        double eff = Nsig/tot_entries;
        double err_eff = sqrt(eff*(1.-eff)/tot_entries);

        //just a check
        printf("cut value = %.3f, eff = %.4f +- %.4f \n",cutval, eff, err_eff);

        //put the results in the graph
        g_eff->SetPoint(icut,cutval,eff);
        g_eff->SetPointError(icut,0,err_eff);

    }
```

Signal only,
denominator of the efficiency

efficiency calculation

Set the point and the error in the graph

Breaking down Exercise 1

```

    printf("\n The signal efficiency for bkg_killer>0.92 is %.3f  \n",
    g_eff->Eval(0.92));

    //and draw the graph
    TCanvas* c = new TCanvas("c","c",800,600);
    g_eff->SetMarkerStyle(8);
    g_eff->SetMarkerSize(0.2);
    g_eff->GetXaxis()->SetTitle("cut value");
    g_eff->GetYaxis()->SetTitle("signal efficiency");
    g_eff->Draw("APL");

    return;
}

```

Draw the result

Breaking down Exercise 1

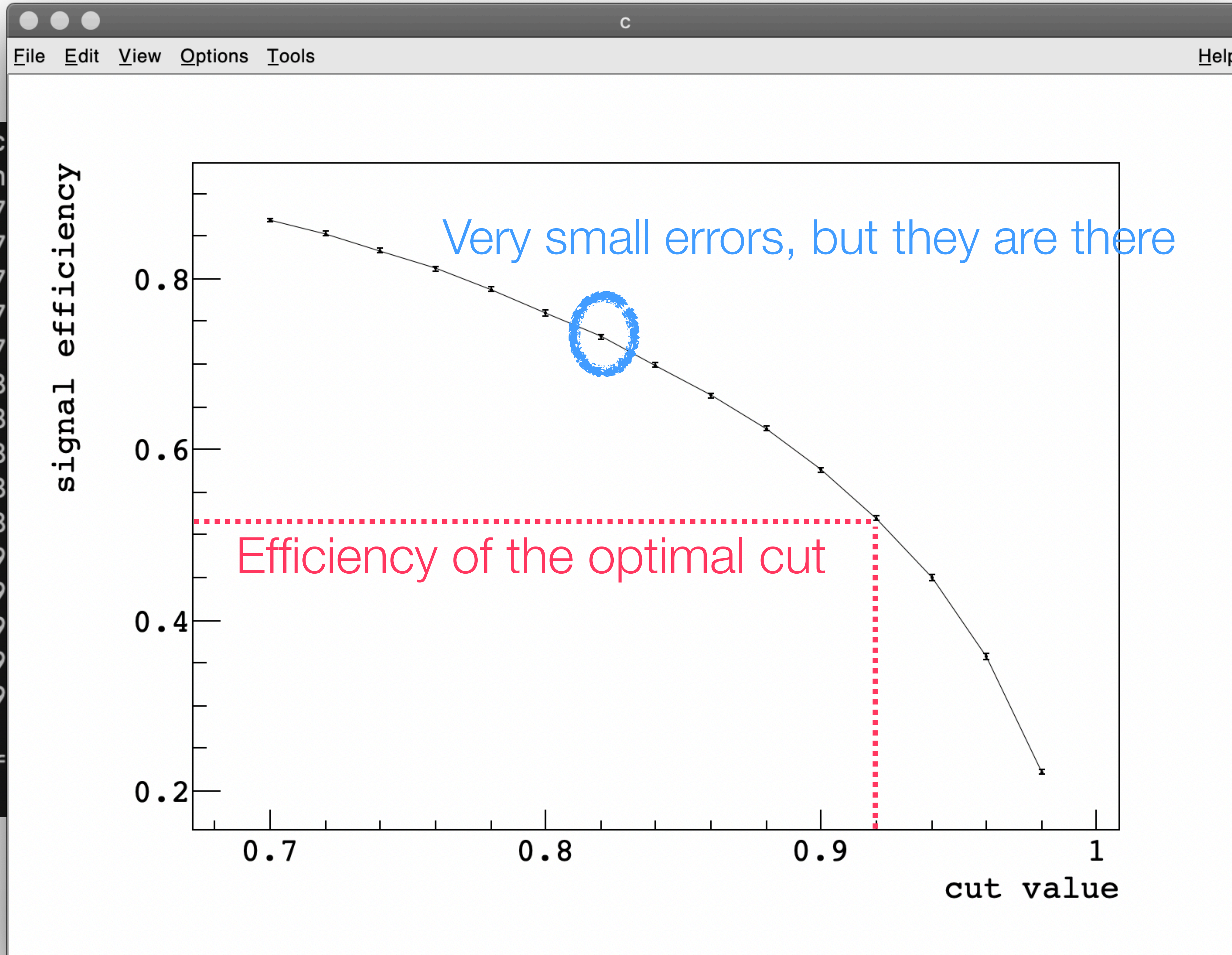
The output

```
Processing calculateEff.C...
Total signal entries in the tree: 21456
cut value = 0.700, eff = 0.8680 +- 0.0023
cut value = 0.720, eff = 0.8524 +- 0.0024
cut value = 0.740, eff = 0.8330 +- 0.0025
cut value = 0.760, eff = 0.8120 +- 0.0027
cut value = 0.780, eff = 0.7871 +- 0.0028
cut value = 0.800, eff = 0.7598 +- 0.0029
cut value = 0.820, eff = 0.7318 +- 0.0030
cut value = 0.840, eff = 0.6988 +- 0.0031
cut value = 0.860, eff = 0.6628 +- 0.0032
cut value = 0.880, eff = 0.6246 +- 0.0033
cut value = 0.900, eff = 0.5761 +- 0.0034
cut value = 0.920, eff = 0.5197 +- 0.0034
cut value = 0.940, eff = 0.4504 +- 0.0034
cut value = 0.960, eff = 0.3578 +- 0.0033
cut value = 0.980, eff = 0.2223 +- 0.0028

 The signal efficiency for bkg_killer>0.92 is 0.520
root [1]
```

Breaking down Exercise 1

The output

```
Processing calc
Total signal en
cut value = 0.7
cut value = 0.7
cut value = 0.7
cut value = 0.7
cut value = 0.7
cut value = 0.8
cut value = 0.8
cut value = 0.8
cut value = 0.8
cut value = 0.8
cut value = 0.9
cut value = 0.9
cut value = 0.9
cut value = 0.9
cut value = 0.9

 The signal eff
root [1]
```



Exercise 2

2. What do you expect for the M distribution of the mis-id background? Draw it, by subtracting from the total distribution the signal and that of the non-B background (like we did for ΔE). Compare its distribution with that of the signal.

We will take `inspectB.C` from the lesson and modify it.

We will take this occasion to revisit `Sumw2()`

Breaking down Exercise 2

```
#include "Riostream.h"
#include "TFile.h"
#include "TTree.h"
#include "TCanvas.h"
#include "TH1D.h"
#include "TLegend.h"

using namespace std;

void compareM(){

    //open file and take the tree
    TFile* file = TFile::Open("simulation.root");
    TTree* tree = (TTree*) file->Get("simTree");

    int tot_entries = tree->GetEntries();
    cout << "Total entries in the tree: " << tot_entries << endl;

    //link the variables with tree banches
    double mass, bkg_killer;
    int bkg, sig;
    tree->SetBranchAddress("B_m",&mass);
    tree->SetBranchAddress("isBkg",&bkg);
    tree->SetBranchAddress("isSig",&sig);
    tree->SetBranchAddress("bkg_killer",&bkg_killer);

```

Breaking down Exercise 2

```
//define an histogram to look at mass distribution
TH1D* h_m_tot = new TH1D("h_m_tot",";mass [GeV/c^{2}]; Entries",40,5.25,5.30);

//very very important to rember when manipulating histograms!!!
h_m_tot->Sumw2();

//clone the same histogram structure for signal, bkg, and unknown bkg
TH1D* h_m_sig = (TH1D*) h_m_tot->Clone("h_m_sig");

h_m_sig->Sumw2();//with Clone() we have already set it
                 //from the original TH1D, it will issue a "Warning"

TH1D* h_m_bkg = (TH1D*) h_m_tot->Clone("h_m_bkg");
TH1D* h_m_unknown = (TH1D*) h_m_tot->Clone("h_m_unknown");
```

IMPORTANT!!!

Not an issue…

```
Warning in <TH1D::Sumw2>: Sum of squares of weights structure already created
```

Breaking down Exercise 2

```
    //loop over the entries
    for(int iEntry; iEntry<tot_entries; ++iEntry){

        tree->GetEntry(iEntry);

        //skip all candidates below the optimal cut point
        if(bkg_killer<0.92) continue;

        //fill the histograms
        h_m_tot->Fill(mass);
        if(bkg) h_m_bkg->Fill(mass);
        else if(sig) h_m_sig->Fill(mass);

    }
```

Fill the histograms, just for the events that pass the cut

Breaking down Exercise 2

- Let's inspect just a bin (here 24): print out its content error for all histograms.
- Make then the operations to obtain the mass histogram for mis-id backgrd.

```
int abin = 24;
cout <<"\n From original histograms: " << endl;
printf("Total  histo, bin %i content: %.1f +- %.1f \n", abin, h_m_tot->GetBinContent(abin), h_m_tot->GetBinError(abin) );
printf("Signal histo, bin %i content: %.1f +- %.1f \n", abin, h_m_sig->GetBinContent(abin), h_m_sig->GetBinError(abin) );
printf("Backgr histo, bin %i content: %.1f +- %.1f \n", abin, h_m_bkg->GetBinContent(abin), h_m_bkg->GetBinError(abin) );

//subtract the signal
h_m_unknown->Add(h_m_tot, h_m_bkg, 1, -1);
h_m_unknown->Add(h_m_sig, -1);

cout <<"\n The derived histogram: " << endl;
printf("B->pipi histo, bin %i content: %.1f +- %.1f \n", abin, h_m_unknown->GetBinContent(abin), h_m_unknown->GetBinError(abin) );
printf("B->pipi histo, sqrt(bin %i content): %.1f \n", abin, sqrt(h_m_unknown->GetBinContent(abin)) );

double error_prop = sqrt(h_m_tot->GetBinError(abin)*h_m_tot->GetBinError(abin) +
                         h_m_sig->GetBinError(abin)*h_m_sig->GetBinError(abin) +
                         h_m_bkg->GetBinError(abin)*h_m_bkg->GetBinError(abin) );
printf("B->pipi histo, from error propagation: %.1f \n\n",error_prop );
```

- Check the result. We expect:

$$N_{\text{unkn.}} = N_{\text{tot}} - N_{\text{bkg}} - N_{\text{sig}}$$

$$\sigma_{\text{unkn.}} = \sqrt{\sigma_{\text{tot}}^2 + \sigma_{\text{bkg}}^2 + \sigma_{\text{sig}}^2} = \sqrt{N_{\text{tot}} + N_{\text{bkg}} + N_{\text{sig}}}$$

Breaking down Exercise 2

- Check the result. We expect:

$$N_{\text{unkn.}} = N_{\text{tot}} - N_{\text{bkg}} - N_{\text{sig}}$$
$$\sigma_{\text{unkn.}} = \sqrt{\sigma^2_{\text{tot}} + \sigma^2_{\text{bkg}} + \sigma^2_{\text{sig}}} = \sqrt{N_{\text{tot}} + N_{\text{bkg}} + N_{\text{sig}}}$$

```
 From original histograms:
Total  histo, bin 24 content: 2432.0 +- 49.3
Signal histo, bin 24 content: 1667.0 +- 40.8
Backgr histo, bin 24 content: 326.0 +- 18.1

 The derived histogram:
B->pipi histo, bin 24 content: 439.0 +- 66.5
B->pipi histo, sqrt(bin 24 content): 21.0
B->pipi histo, from error propagation: 66.5
```

Breaking down Exercise 2

- Let's take out the command `Sumw2()`

$$N_{\text{unkn.}} = N_{\text{tot}} - N_{\text{bkg}} - N_{\text{sig}}$$

$$\sigma_{\text{unkn.}} = \sqrt{\sigma^2_{\text{tot}} + \sigma^2_{\text{bkg}} + \sigma^2_{\text{sig}}} = \sqrt{N_{\text{tot}} + N_{\text{bkg}} + N_{\text{sig}}}$$

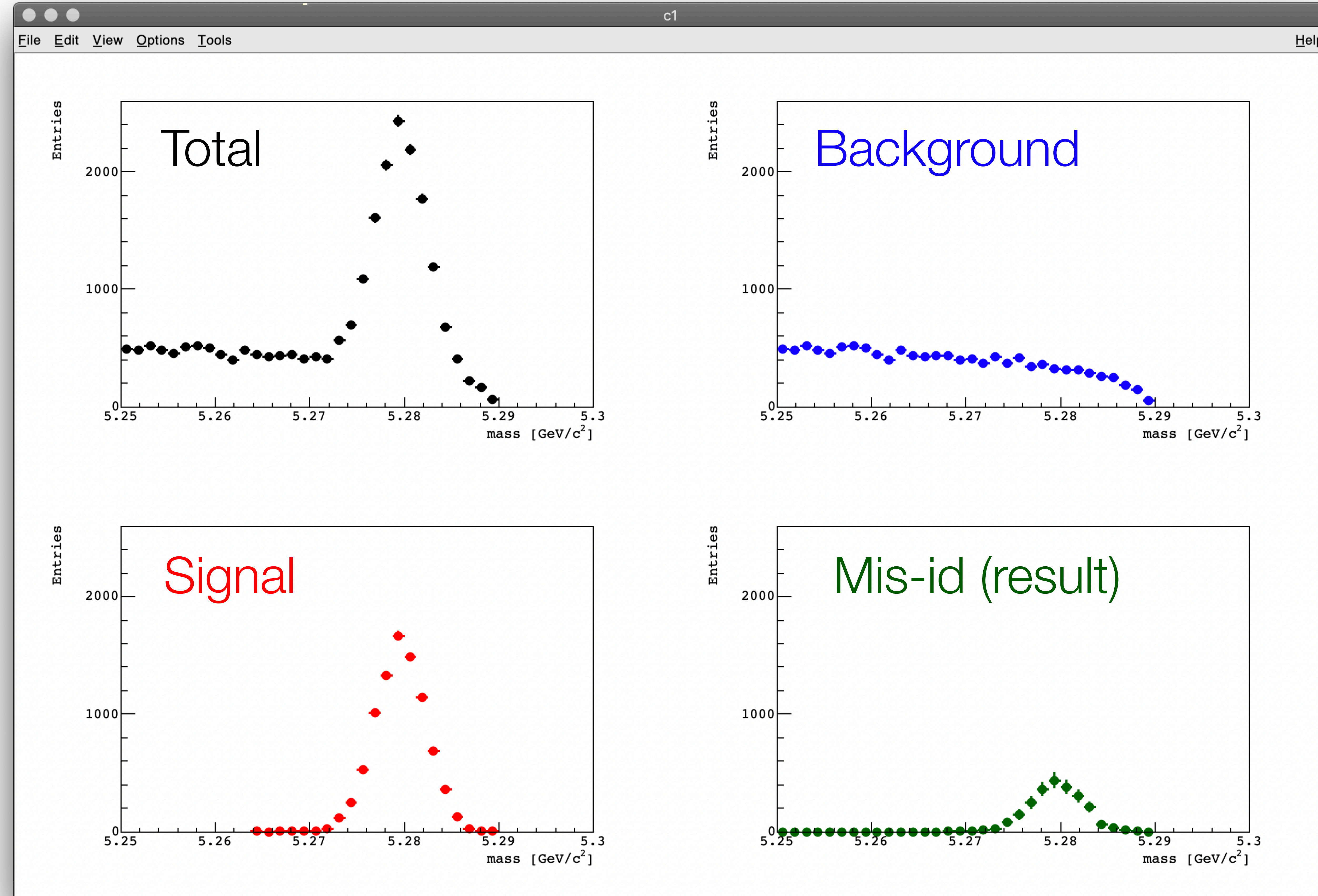
```
 From original histograms:
Total  histo, bin 24 content: 2432.0 +- 49.3
Signal histo, bin 24 content: 1667.0 +- 40.8
Backgr histo, bin 24 content: 326.0 +- 18.1

 The derived histogram:
B->pipi histo, bin 24 content: 439.0 +- 21.0
B->pipi histo, sqrt(bin 24 content): 21.0
B->pipi histo, from error propagation: 66.5
```

WRONG!!!

Breaking down Exercise 2

Just drawings…

```
//draw the histograms
TCanvas* c1 = new TCanvas("c1","c1",1200,800);
c1->Divide(2,2);
c1->cd(1);
h_m_tot->GetYaxis()->SetRangeUser(0,2600);
h_m_tot->Draw();
c1->cd(2);
h_m_bkg->GetYaxis()->SetRangeUser(0,2600);
h_m_bkg->SetMarkerColor(kBlue);
h_m_bkg->SetLineColor(kBlue);
h_m_bkg->Draw();
c1->cd(3);
h_m_sig->GetYaxis()->SetRangeUser(0,2600);
h_m_sig->SetMarkerColor(kRed);
h_m_sig->SetLineColor(kRed);
h_m_sig->Draw();
c1->cd(4);
h_m_unknown->GetYaxis()->SetRangeUser(0,2600);
h_m_unknown->SetMarkerColor(kMagenta);
h_m_unknown->SetLineColor(kMagenta);
h_m_unknown->Draw();
```


c1
File Edit View Options Tools Help
Total
Background
Signal
Mis-id (result)
Entries
0
1000
2000
5.25
5.26
5.27
5.28
5.29
5.3
mass [GeV/c^2]

Breaking down Exercise 2 and 4

4. Instead of using `DrawNormalized()`, scale to 1 the histogram integral using the `Scale()` method of `TH1` (check the integral value) and normal `Draw()` method.

```
    //compare signal and B->pipi shapes:

    //first scale the histograms to the same unit area
    cout << "Signal histo integral: " << h_m_sig->Integral() << endl;
    cout << "B->pipi histo integral: " << h_m_unknown->Integral() << endl;

    h_m_sig->Scale(1./h_m_sig->Integral());
    h_m_unknown->Scale(1./h_m_unknown->Integral());

    cout << "After normalisation to unit area: " << endl;
    cout << "Signal histo integral: "  << h_m_sig->Integral() << endl;
    cout << "B->pipi histo integral: " << h_m_unknown->Integral() << endl;

    //a useful way to spot differences in shape is making the ratio of the distirbutions
    TH1D* h_ratio = (TH1D*) h_m_unknown->Clone("ratio");
    h_ratio->Divide(h_m_sig);

```

Use of scale to normalise to unit area

Another common operation on histograms

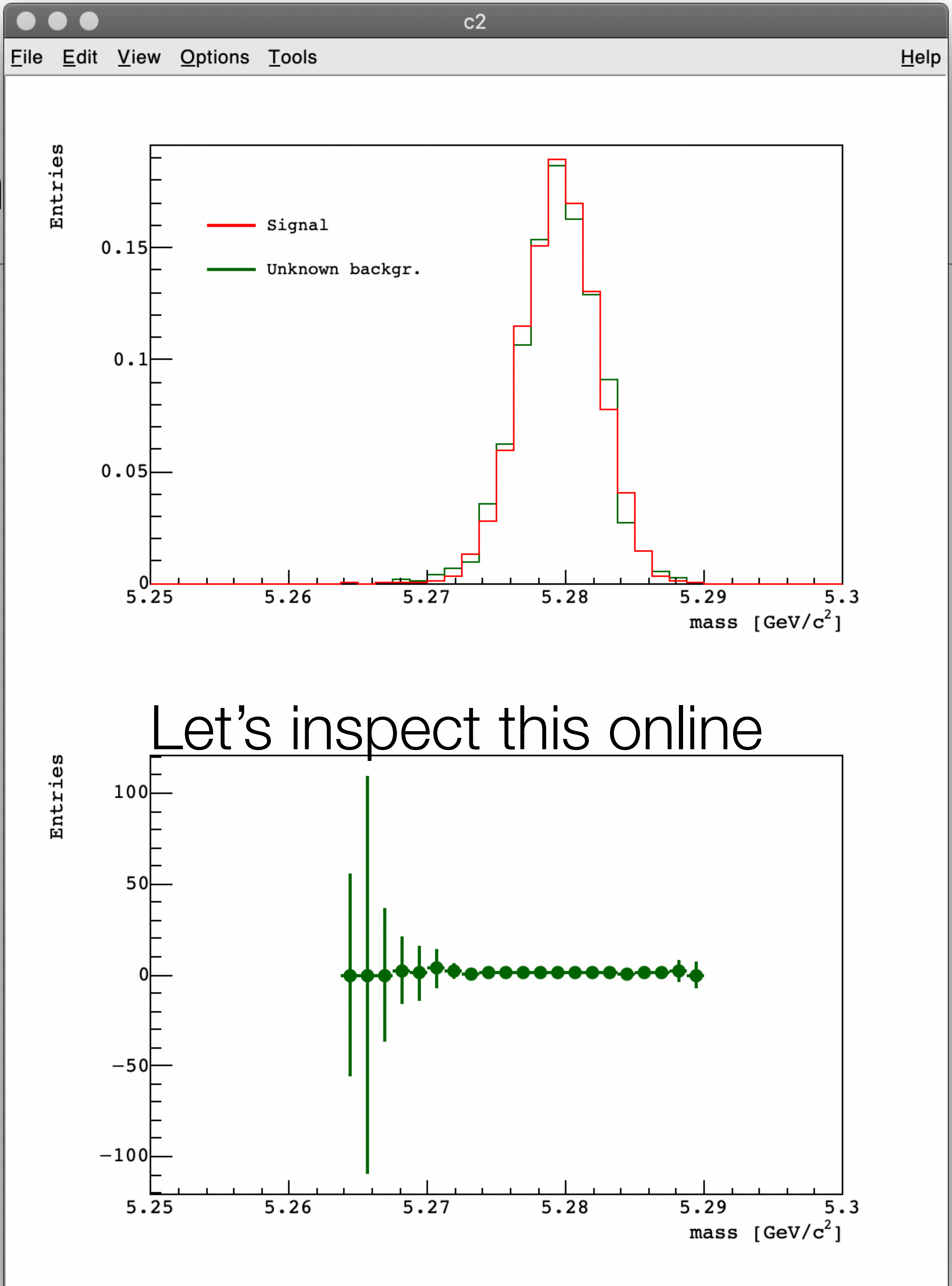
Breaking down Exercise 2 and 4

Draw the result

```
    TCanvas* c2 = new TCanvas("c2","c2",600,800);
    c2->Divide(1,2);
    c2->cd(1);
    h_m_unknown->Draw("histo");
    h_m_sig->Draw("histo same");

    //put a legend
    TLegend* leg = new TLegend(0.2,0.65,0.5,0.8);
    leg->AddEntry(h_m_sig,"Signal","L");
    leg->AddEntry(h_m_unknown,"Unknown backgr.","L");
    leg->Draw();

    c2->cd(2);
    h_ratio->Draw();
```

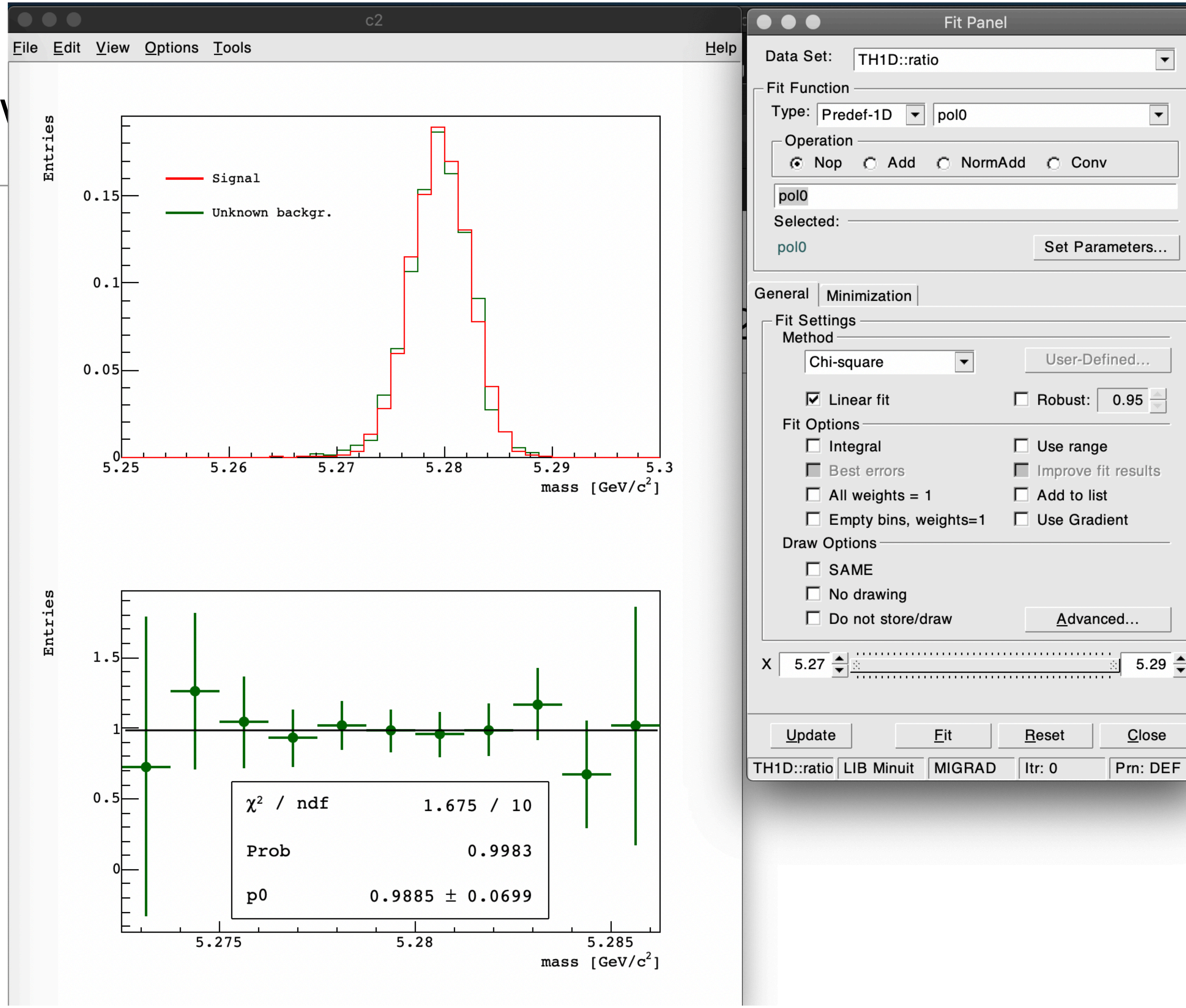
Breaking down

Draw the result



Let’s inspect this online

Breaking do



Zoom in the core of the distributions, to check the flatness of the ratio.

To quantify it, we can make a fit of the point with a constant and see the probability of the χ^2. Here it is (ridiculously) high…