Introduction to ROOT: part 4

Mirco Dorigo
mirco.dorigo@ts.infn.it

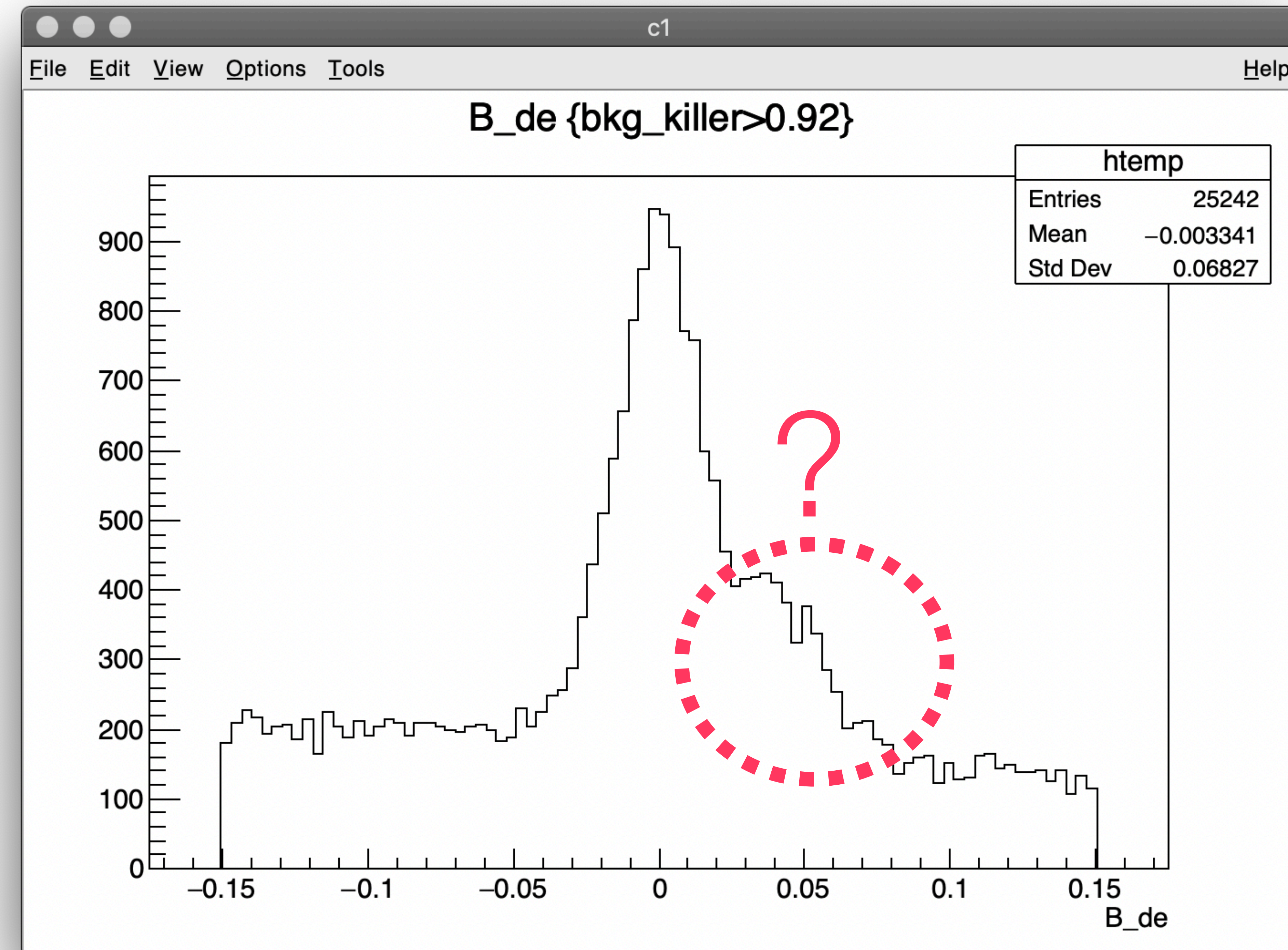
LACD 2022-2023
March 29th, 2023

Previous lesson

- Accessing the data from a `TTree` in a ROOT file
- Inspect the data looking at distributions, make a selection, in macros and live (also in 2 dimensions)
- Making a graph to display data pairs.
- Manipulating histograms (use of `Sum2w()`, and treatment of bin-content errors).

What’s this shoulder?

```
root [7] simTree->Draw("B_de","bkg_killer>0.92");
```



Background from other B decays

- `bkg_killer` is built to suppress events that are ***not*** $\Upsilon(4S) \to B\overline{B}$.

- Among $\Upsilon(4S) \to B\overline{B}$ events, there are *B* decays that are not signal, but that can be mis-reconstructed as our signal.

- For instance a pion in $B^0 \to \pi^+\pi^-$ decays can be mis-identified as kaon and be reconstructed as $B^0 \to K^+\pi^-$

- Let's check in simulation. We have a variable that flag real $B^0 \to K^+\pi^-$ signal candidates only.

Inspect B decays (`inspectB.C`)

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

using namespace std;

void inspectB(){

    //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 B_de, bkg_killer;
    int isBkg, isSig;
    tree->SetBranchAddress("B_de",&B_de);
    tree->SetBranchAddress("isBkg",&isBkg);
    tree->SetBranchAddress("isSig",&isSig);
    tree->SetBranchAddress("bkg_killer",&bkg_killer);
```

A class to add legends in plot (search the reference class)

All quite standard now

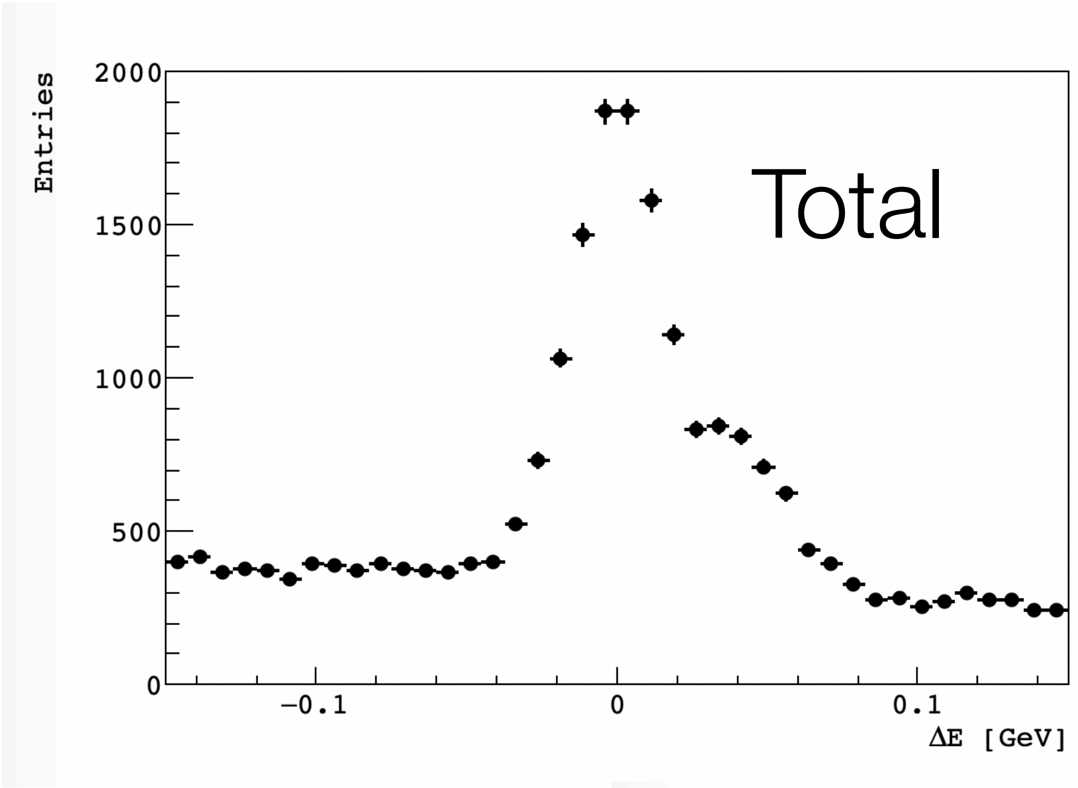
To select only signal

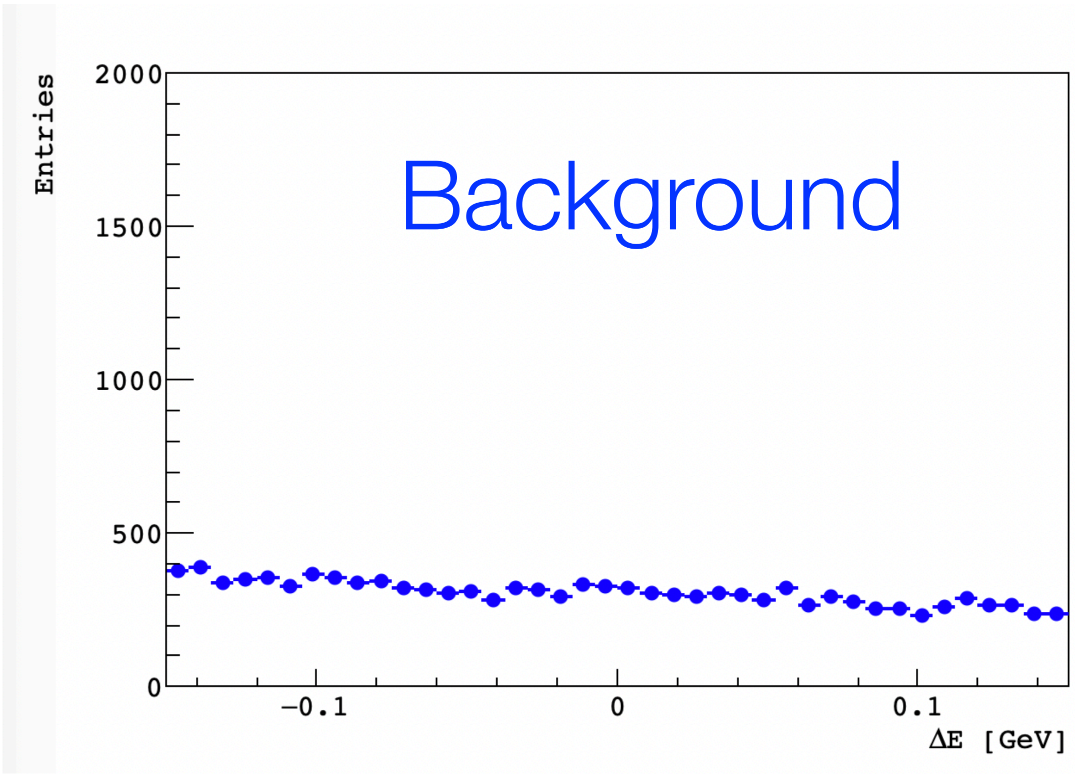
Inspect B decays (`inspectB.C`)

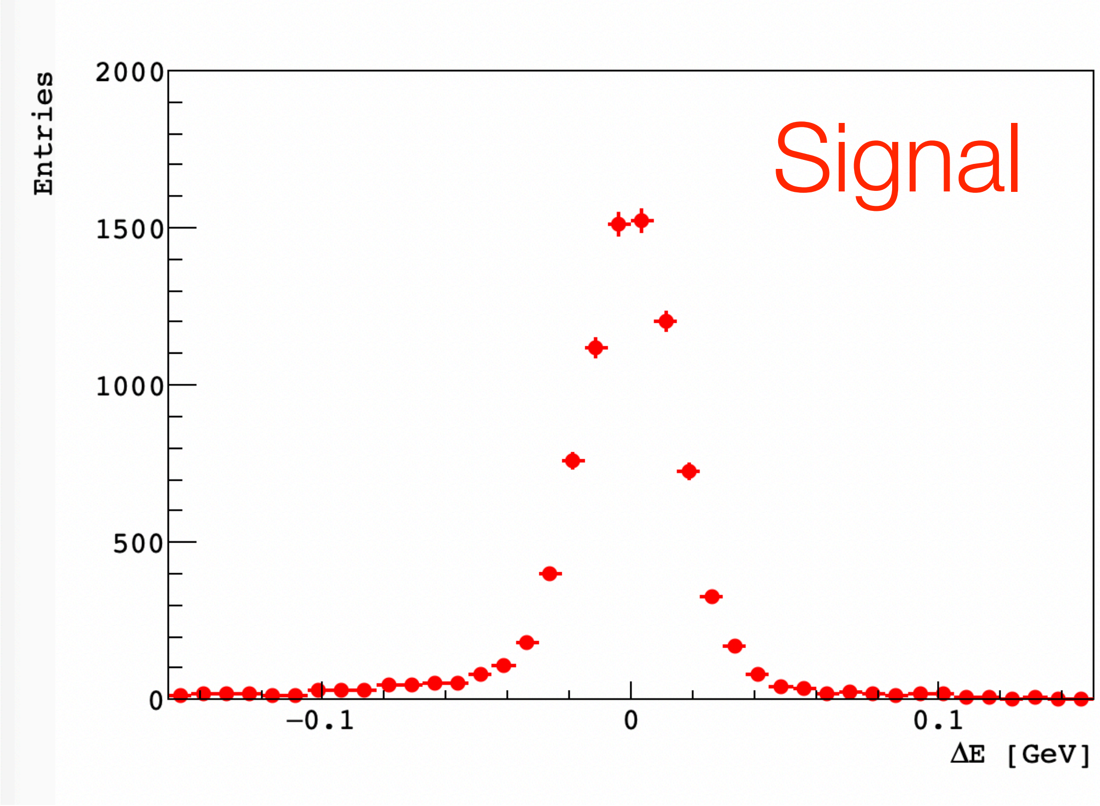
```
    //define an histogram to look at deltaE distribution
    TH1D* h_de_tot = new TH1D("h_de_tot",";m(B) [GeV]; Entries",40,-0.15,0.15);

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

    //clone the same histogram structure for signal, bkg, and unknown bkg
    TH1D* h_de_sig = (TH1D*) h_de_tot->Clone("h_de_sig");
    TH1D* h_de_bkg = (TH1D*) h_de_tot->Clone("h_de_bkg");
    TH1D* h_de_unknown = (TH1D*) h_de_tot->Clone("h_de_unknown");

    //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_de_tot->Fill(B_de);
        if(isBkg) h_de_bkg->Fill(B_de);
        else if(isSig) h_de_sig->Fill(B_de);

    }
```

IMPORTANT!!!

Inspect B decays (`inspectB.C`)





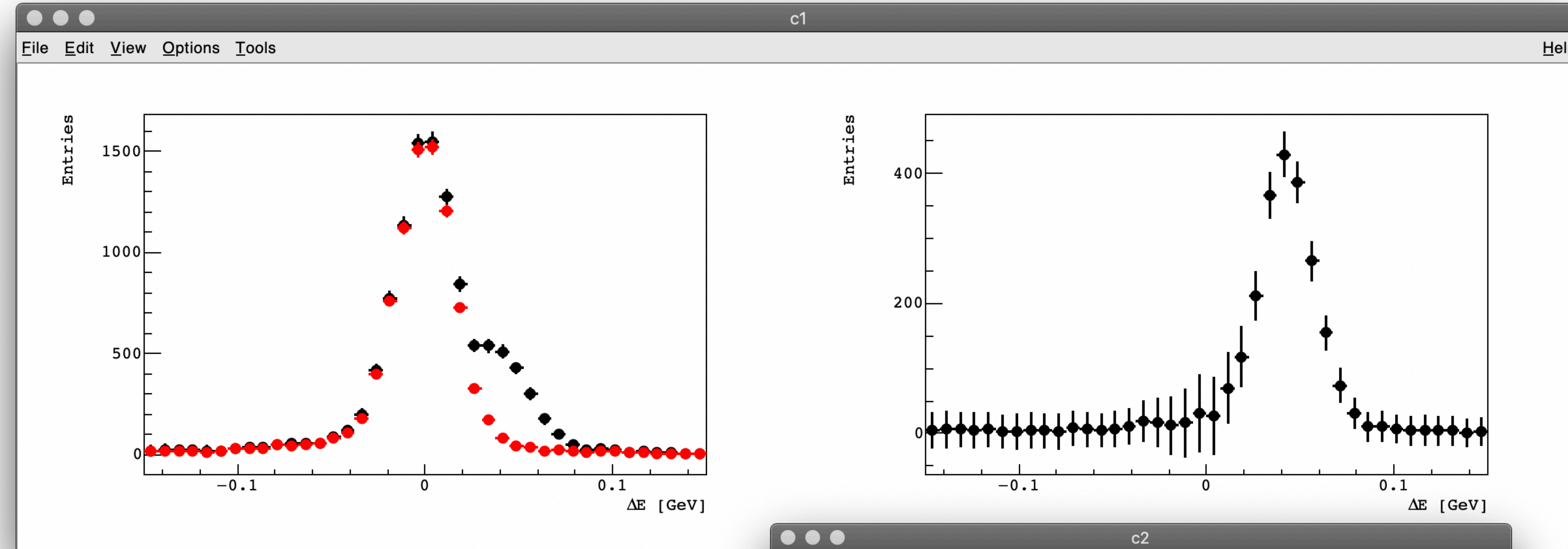


Inspect B decays (`inspectB.C`)

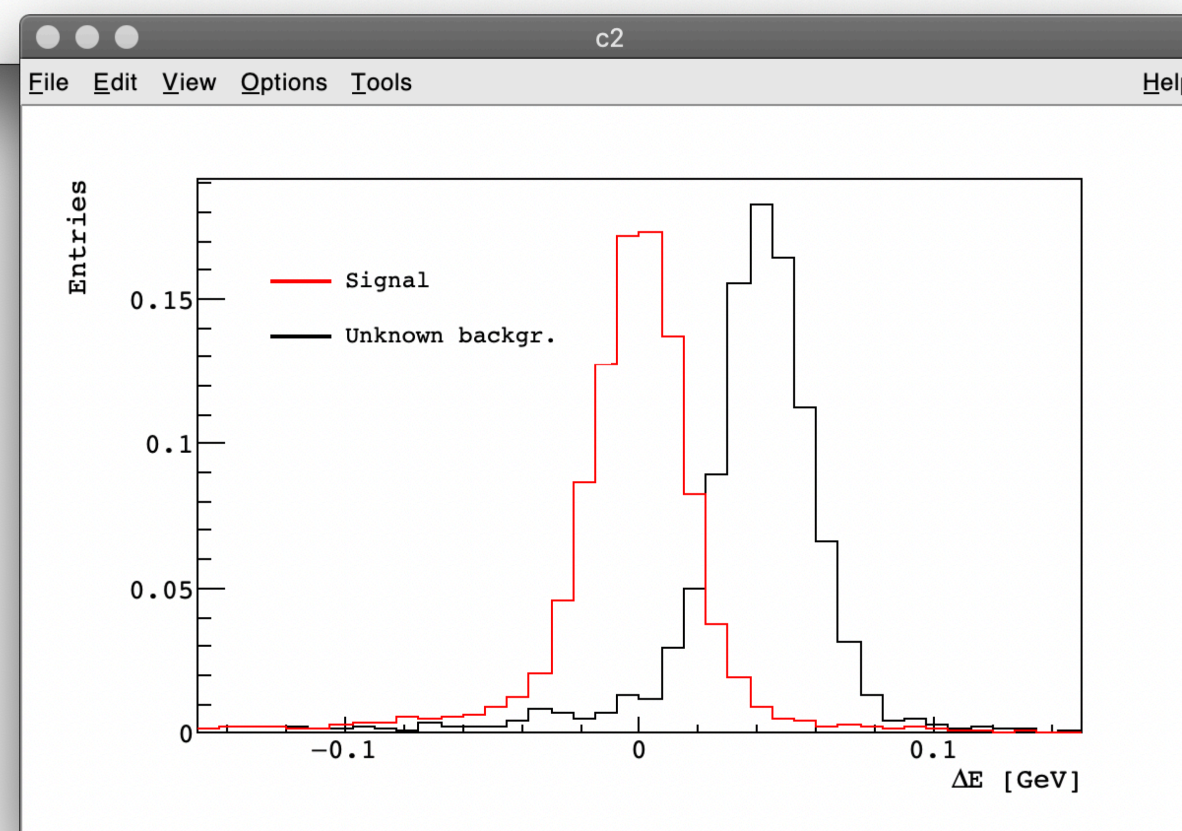
```
    //subtract the background from the total
    h_de_tot->Add(h_de_bkg,-1);

    //subtract the signal
    h_de_unknown->Add(h_de_tot, h_de_sig, 1, -1);


    //draw the histograms
    TCanvas* c1 = new TCanvas("c1","c1",1200,400);
    c1->Divide(2,1);
    c1->cd(1);
    h_de_tot->Draw();
    h_de_sig->SetLineColor(kRed);
    h_de_sig->SetMarkerColor(kRed);
    h_de_sig->Draw("same");
    c1->cd(2);
    h_de_unknown->Draw();

    //compare signal and unkown background shapes
    TCanvas* c2 = new TCanvas("c2","c2",600,400);
    h_de_unknown->DrawNormalized("histo");
    h_de_sig->DrawNormalized("histo same");

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

    cout << "Integral from signal: " << h_de_sig->Integral() << endl;
    cout << "Integral from unkn. back.: " << h_de_unknown->Integral() << endl;
```

We are manipulating the bin contents of the histograms here.
Only with `Sumw2()` the uncertainty on the bin content is properly calculated

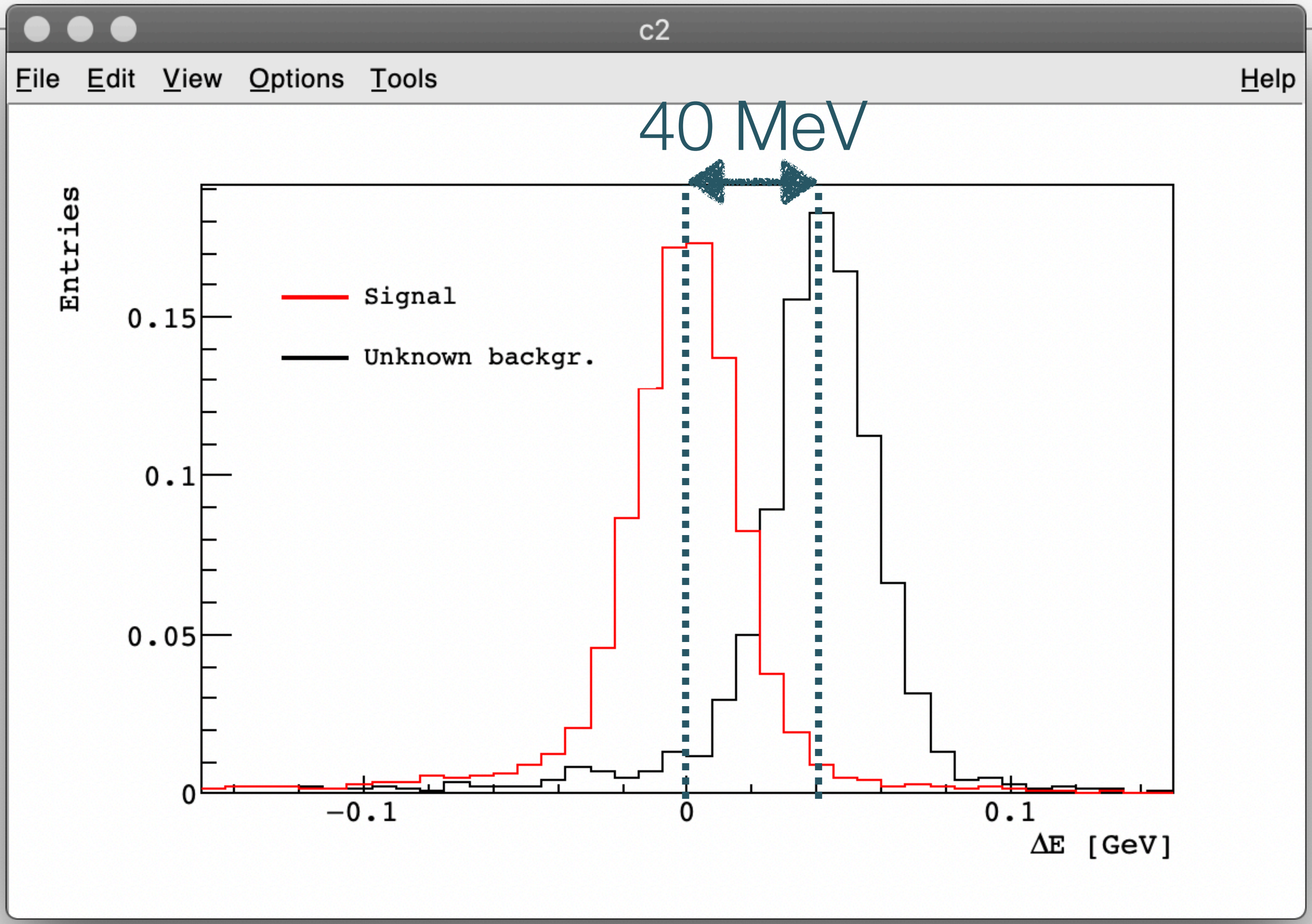
The output




```
root [0]
Processing inspectB.C...
Total entries in the tree: 283056
Integral from signal: 8798
Integral from unkn. back.: 2352
```



Misidentified background



- Indeed, this is given by pion-to-kaon misidentification. If you calculate the shift in ΔE due to the different pion-kaon mass, you will find about $+40\,\mathrm{MeV}$.
- We can use a variable, built from PID detectors, to suppress this background.

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.

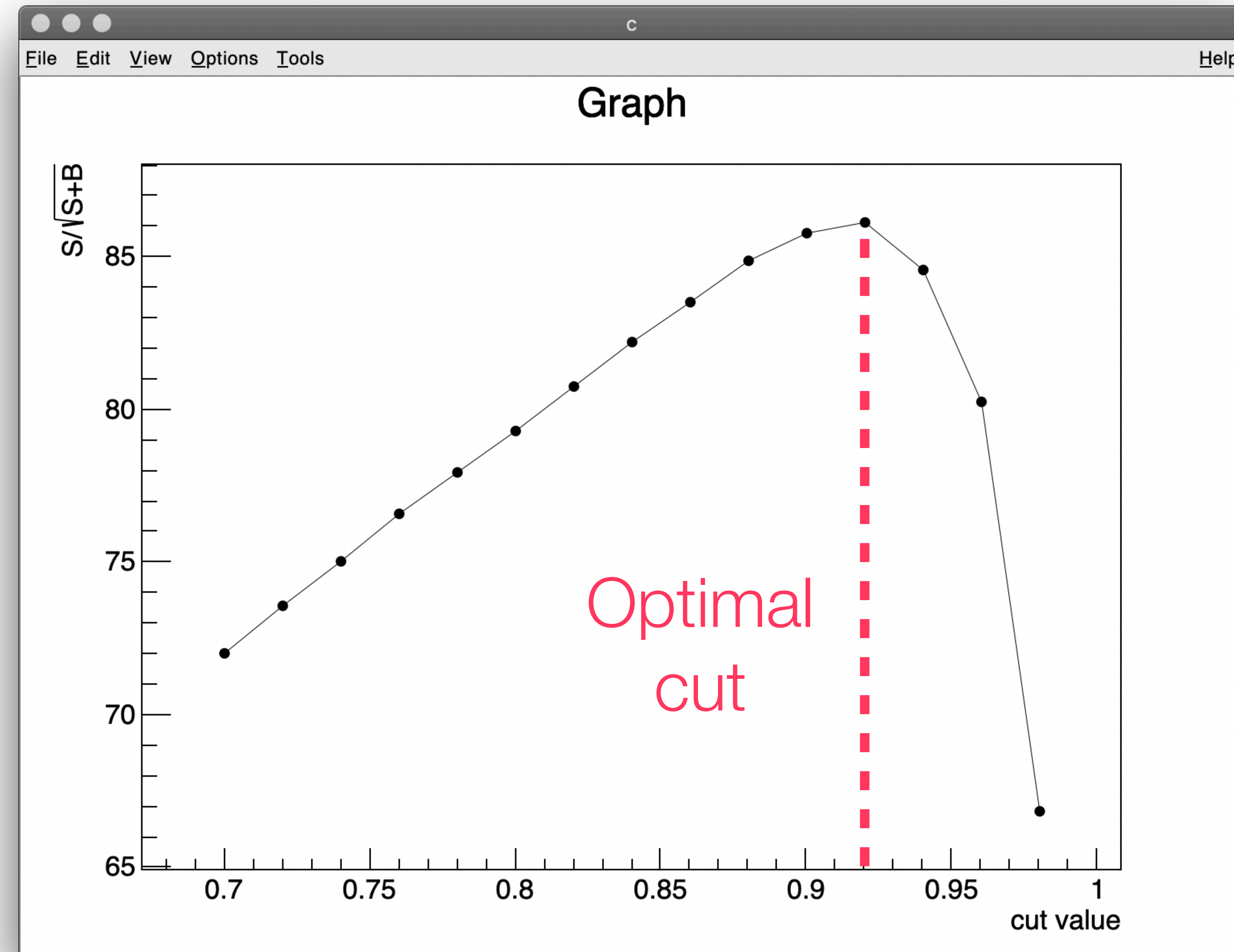
Exercises (3rd lesson)

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.

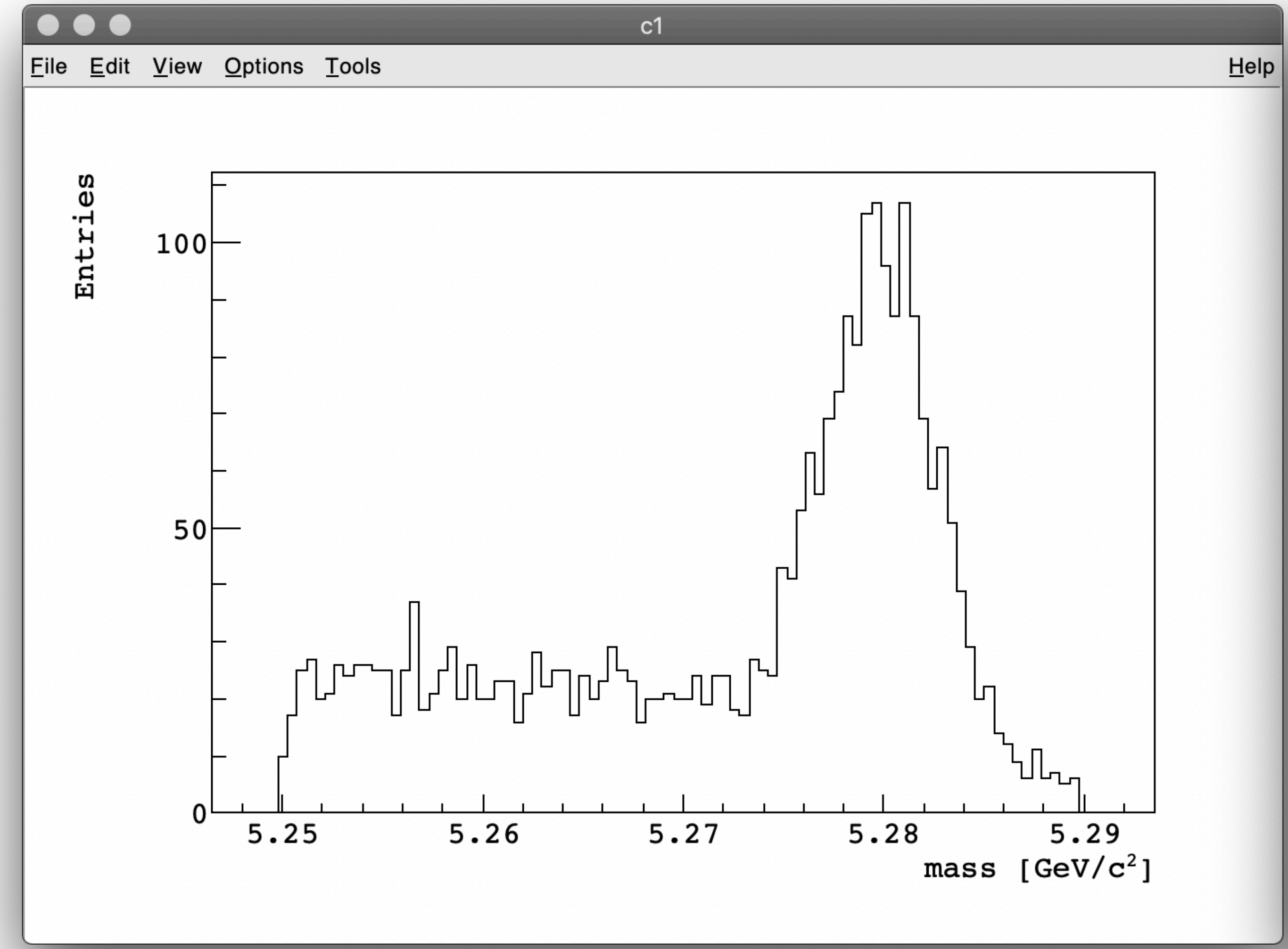
NB: make sure all numbers and text in plots is well visible, by adjusting size of fonts, labels…

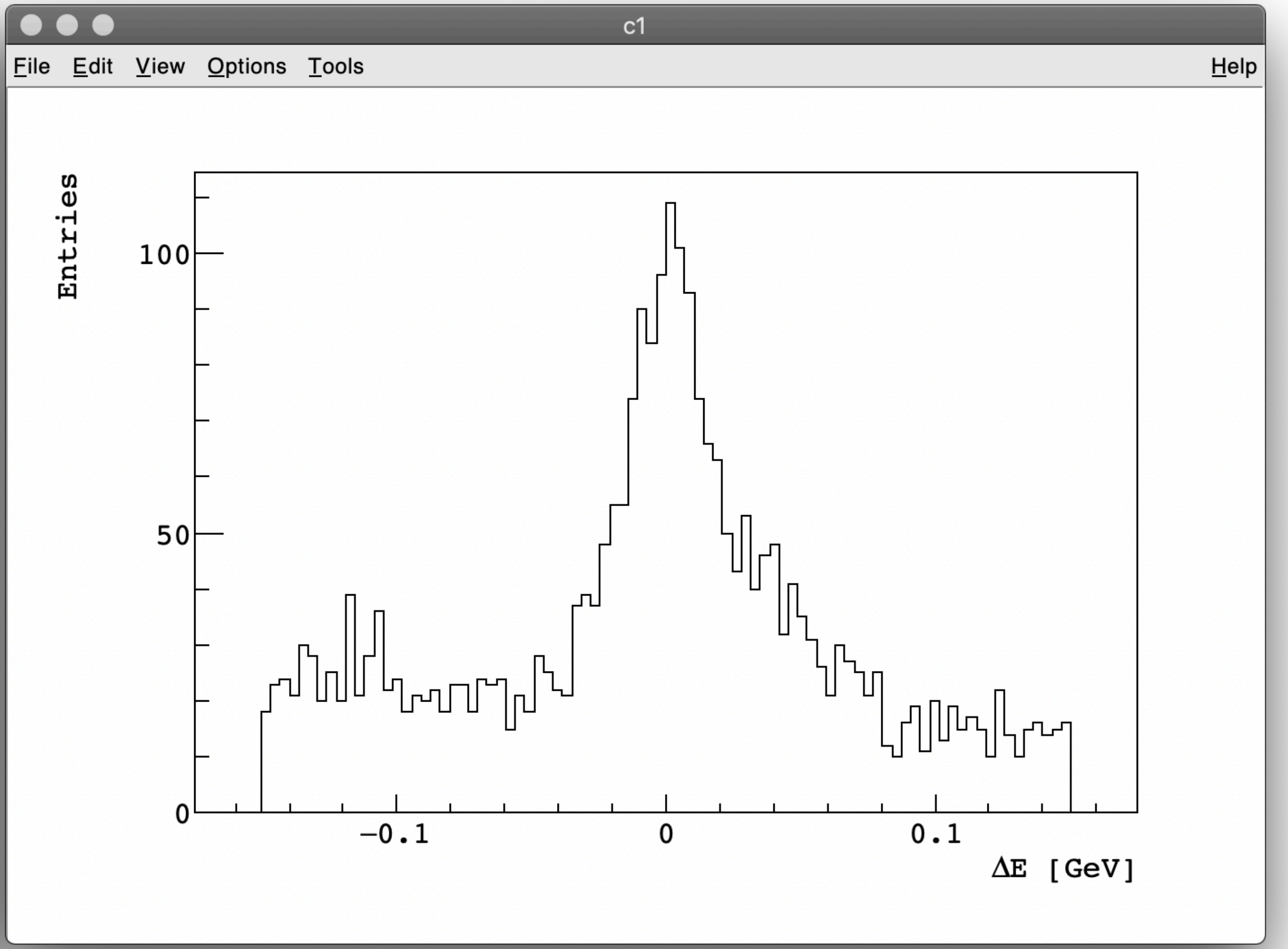
Optimised selection



Look at data

- Our original goal was to measure the signal yield in data.
- Let's take a look at data: check out M and ΔE distributions in `data.root`, after the cut `bkg_killer>0.92`.
- Put them in a ROOT file.





Let’s make a fit of our data sample

- We will fit the ΔE distribution (why?).
- Of course, we cannot just select only signal as in simulation. We can “statistically disentangle” the components that make up the observed distribution. We will do a fit to get the sample composition.
- We can model the fit function (probability density function, pdf) from simulation, studying each component.
- Then we will apply our model to fit the data.

A note…

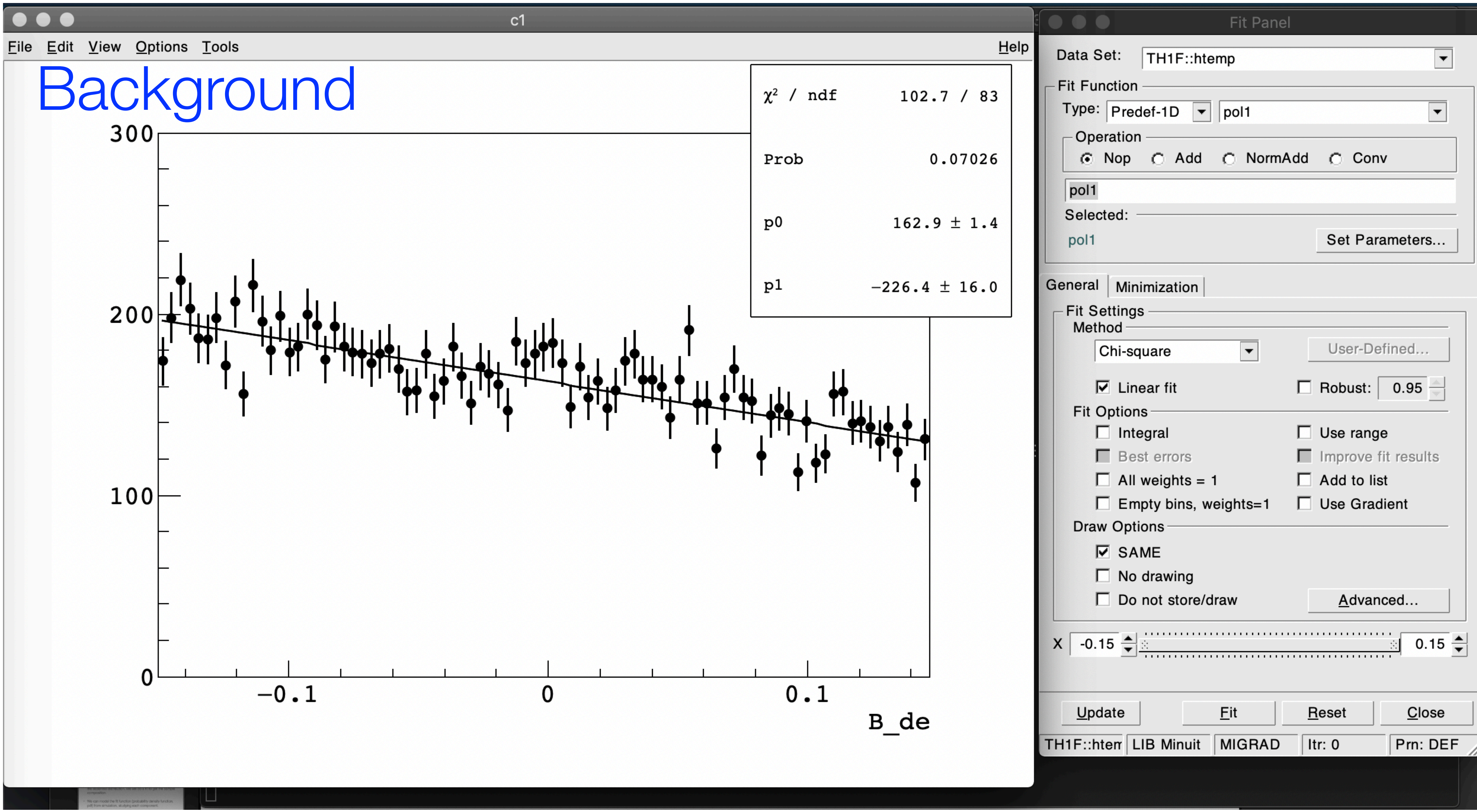
- Fitting methods are a very large topic that would require several lessons.

- I'm sure you have some background from the statistic lessons, and that you have seen already topics like parameter estimation, χ^2, likelihood, pdf, and so on…

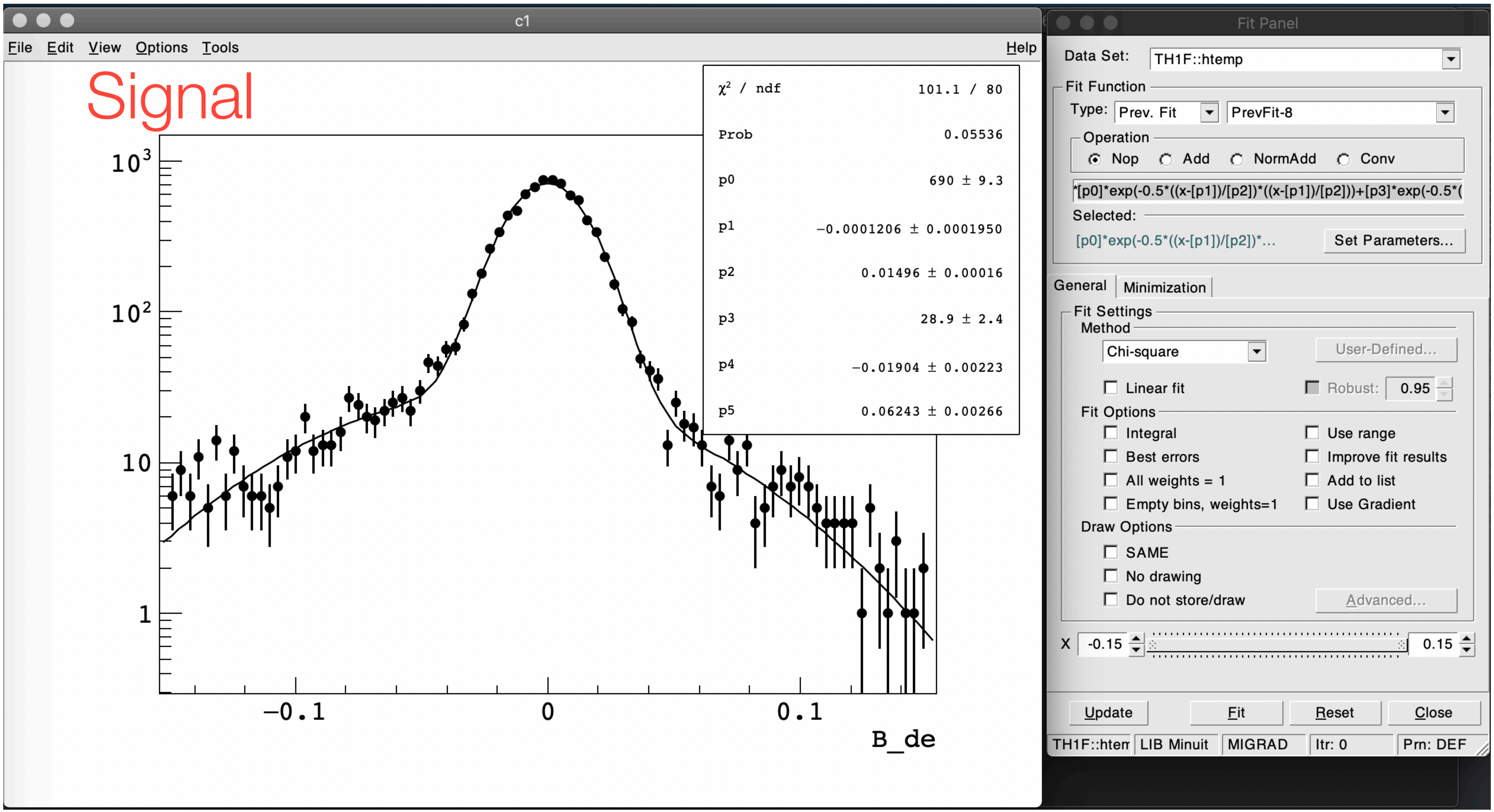
- Here we will see very simple fits to histograms (i.e. binned data), but that enables us to achieve already some good results.

- Bear in mind that's not the full story, at all!
  It's just the tip of the iceberg…

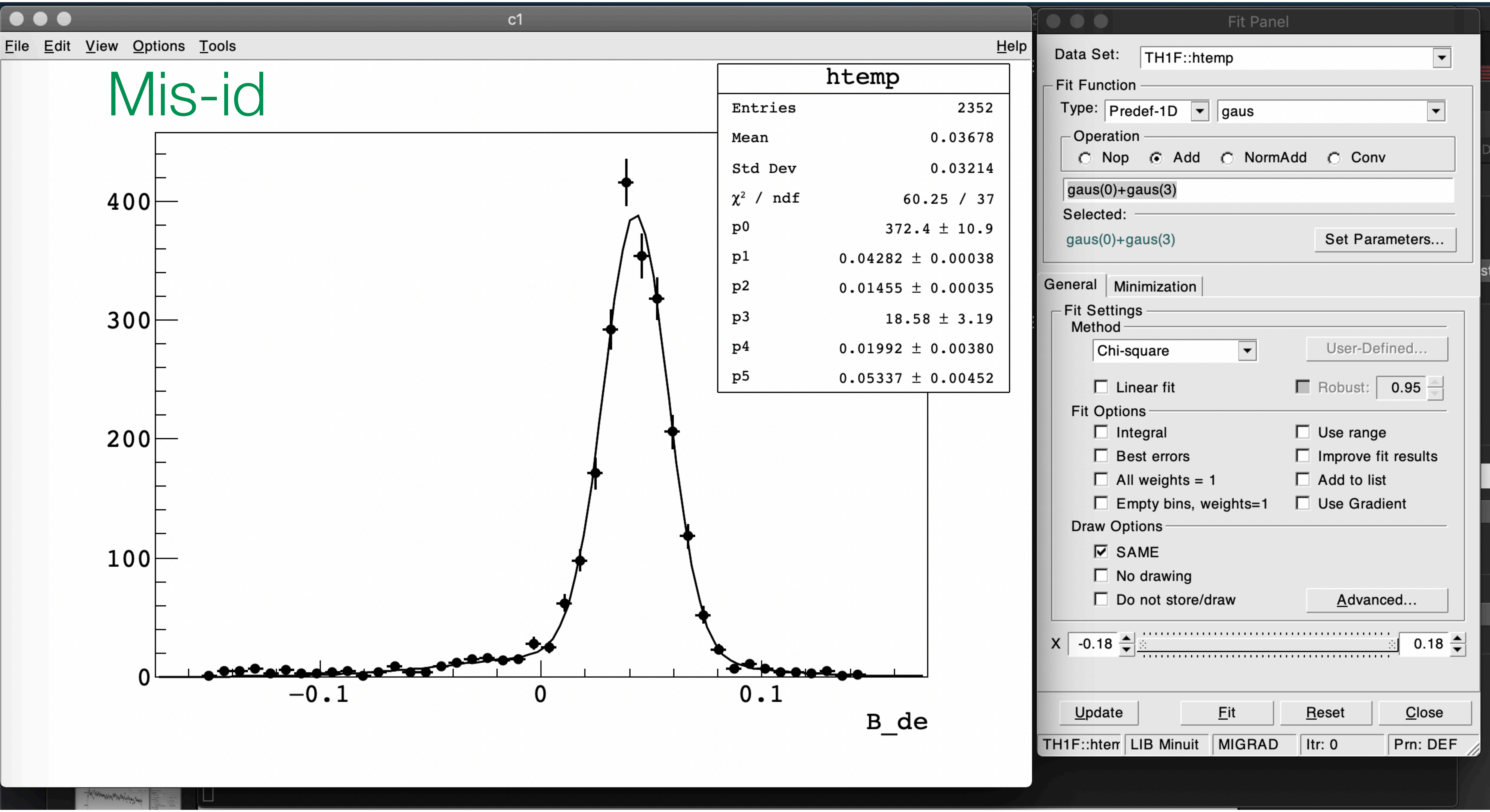
Model the components using the FitPanel



Model the components using the FitPanel



Model the components using the FitPanel



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…

```
#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…

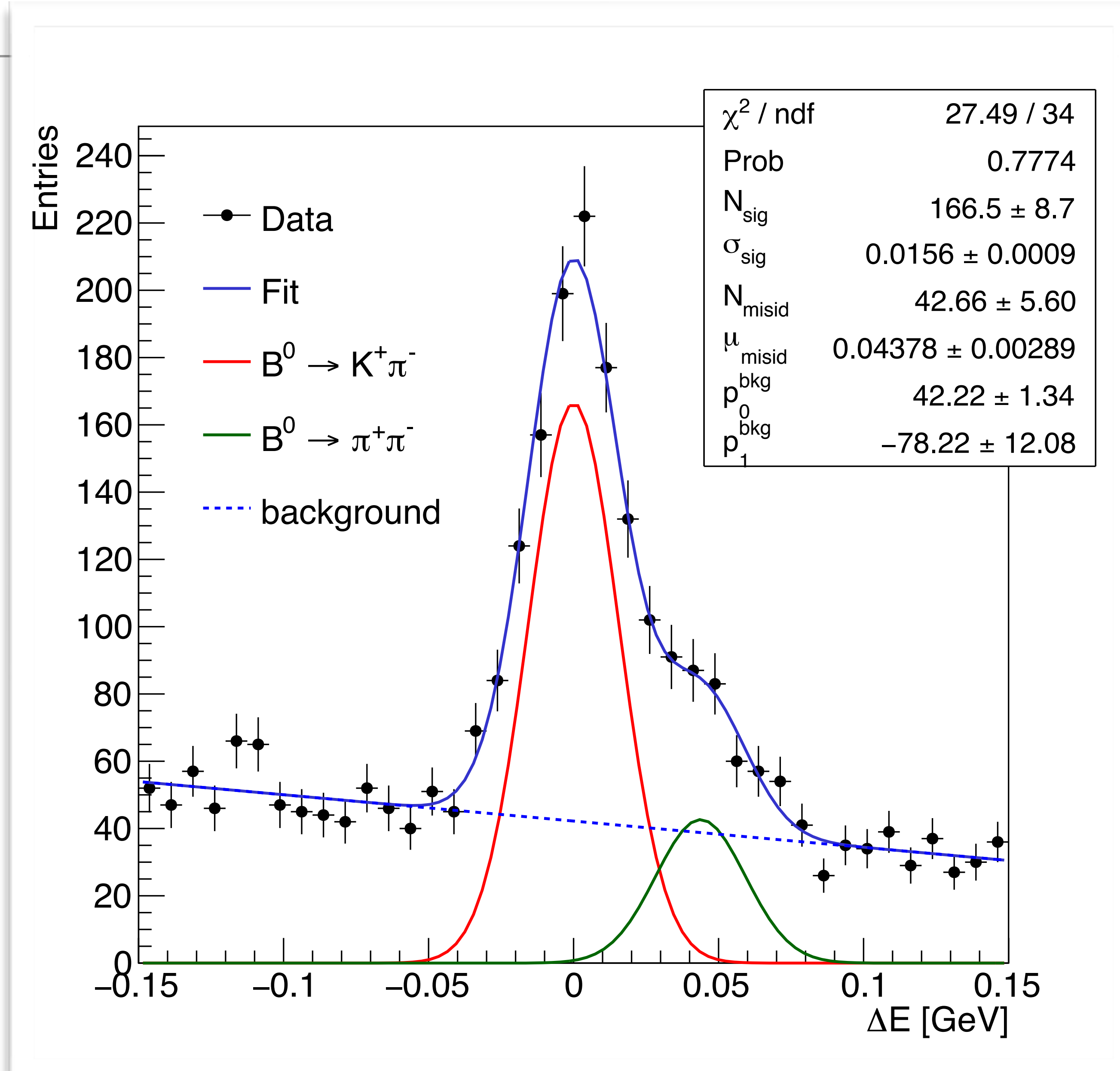
Take `fitDeltaE.C`

```
    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
```

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?