Introduction to ROOT: part 4

Mirco Dorigo
mirco.dorigo@ts.infn.it

LACD 2024-2025
March 28th, 2025

INFN
TRIESTE

Take home messages from last class

1. We learnt how to inspect data through distributions (histograms), 1D and 2D. Can do it “interactively” or in a script.

2. We know how to make fancy plots. **Make sure that your plot clearly shows the message you want to convey:** the content must be right and the format is important (visible data/titles/numbers/labels/legend…)

3. **Root by default sets bin errors as sqrt of the entries.** For proper error propagations, use `Sumw2()`.

4. To compare distributions, normalised them to the same (unit) area and make ratios.

Today: fitting

- Fitting is a very broad topic: it would require several lessons.
- I assume you have some background on theory of parameter estimation: χ^2, likelihood, pdf, …
- We will see very simple fits to data points and histograms. With those we can already solve many problems.
- Bear in mind that's not the full story at all!

First case: fit to check a calibration

- A colleague gives us a txt file `px_calibration.txt`
- It contains pairs of measurements `(x,y)`, corresponding to the calibration of the x-component of the particle momentum:
  - `x` is the (absolute value of) measured momentum
  - `y` is the (absolute valued of) calibrated momentum.
    It also has an uncertainty.
- Let's store the data with a TGraphError and check the linear relation of the measurements.

The data

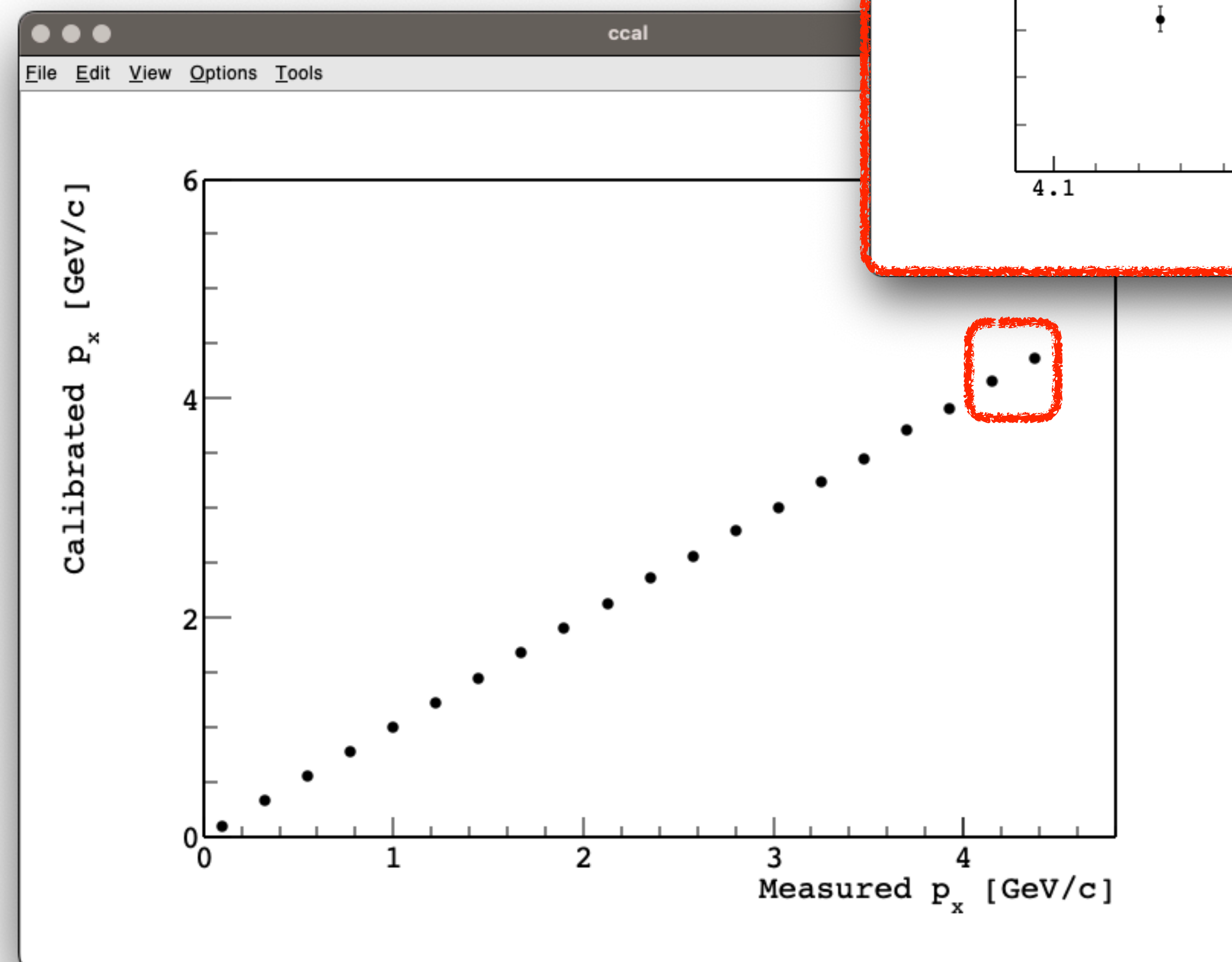
x	y	err_y
0.1	0.100499	0.000333333
0.325	0.324294	0.00108333
0.55	0.55215	0.00183333
0.775	0.774884	0.00258333
1	1.00412	0.00333333
1.225	1.22465	0.00408333
1.45	1.44347	0.00483333
1.675	1.67437	0.00558333
1.9	1.90008	0.00633333
2.125	2.12064	0.00708333
2.35	2.36635	0.00783333
2.575	2.56232	0.00858333
2.8	2.79931	0.00933333
3.025	3.00317	0.0100833
3.25	3.23276	0.0108333
3.475	3.45088	0.0115833
3.7	3.7142	0.0123333
3.925	3.91056	0.0130833
4.15	4.16203	0.0138333
4.375	4.36664	0.0145833

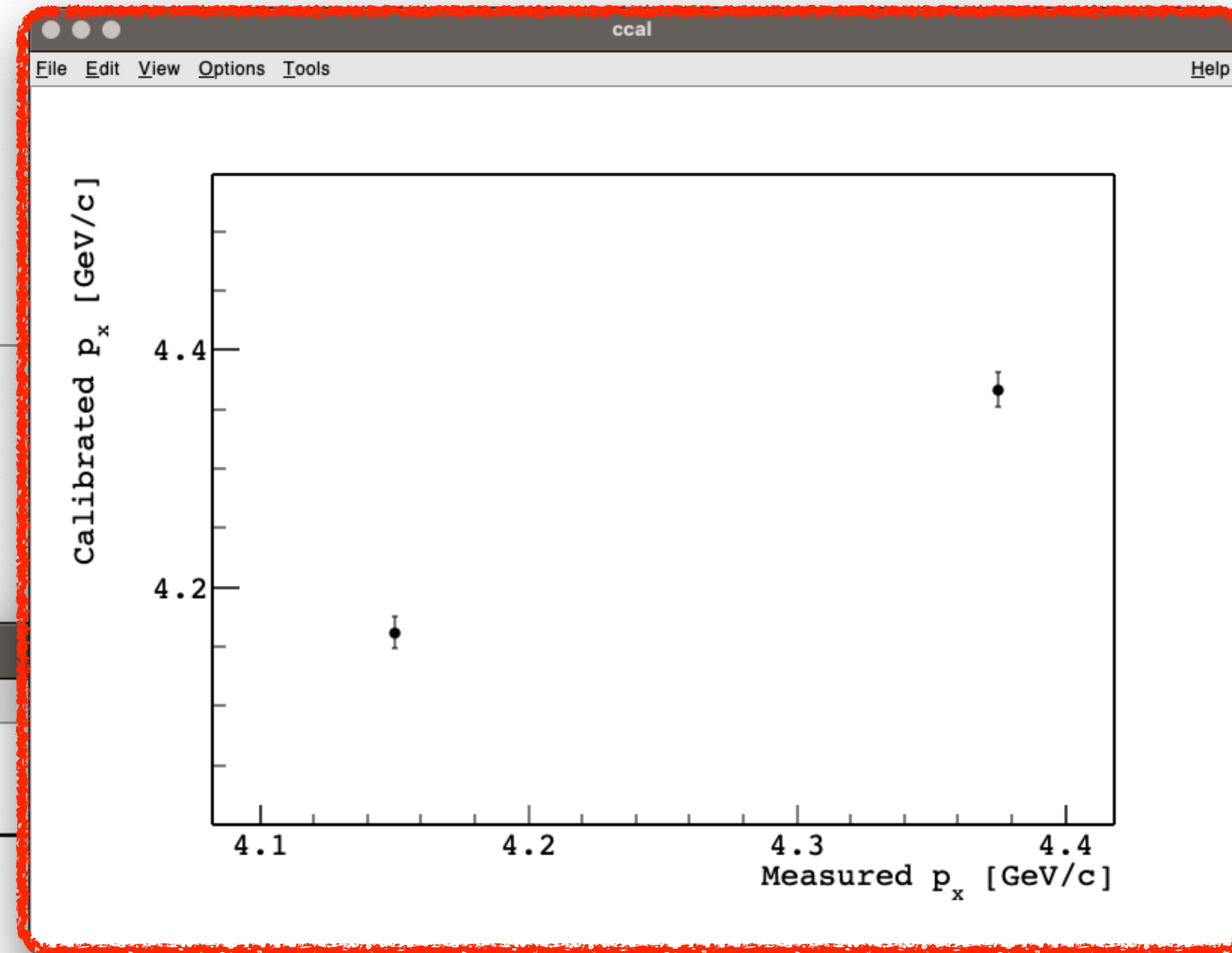
Check the calibration

Check the class

```cpp
#include "TCanvas.h"
#include "TGraphErrors.h"
#include "TF1.h"

using namespace std;

void checkCalib(){

    //We construct a graph to store the calibration.
    //With this constructors we directly read the txt file.
    //Check the reference guide for the different options.
    TGraphErrors* gCal = new TGraphErrors("px_calibration.txt","%lg %lg %lg");

    //Comment the line above, and uncomment that below. Check the difference.
    //TGraphErrors* gCal = new TGraphErrors("px_calibration_errX.txt");

    //Some style choices
    gCal->GetXaxis()->SetTitle("Measured p_{x} [GeV/c]");
    gCal->GetYaxis()->SetTitle("Calibrated p_{x} [GeV/c]");
    gCal->GetYaxis()->SetRangeUser(0,6);
    gCal->SetTitle(0);
    gCal->SetMarkerStyle(8);
    gCal->SetMarkerSize(0.8);
```

The data


ccal
File Edit View Options Tools
Calibrated p x [GeV/c]
Measured p x [GeV/c]
0
1
2
3
4
6


ccal
File Edit View Options Tools Help
Calibrated p x [GeV/c]
4.4
4.2
4.1
4.2
4.3
4.4
Measured p x [GeV/c]

Fit the data

- We use a linear function ($p_0 + p_1 x$, in root is `pol1`) and do a χ^2 fit:

```
    //We define a function to fit the calibration data
    TF1* f_calib = new TF1("f_calib","pol1",0,5);
    gCal->Fit("f_calib");

    //Let's draw the calibration graph and save it in a pdf.
    TCanvas* ccal = new TCanvas("ccal","ccal",800,600);
    gCal->Draw("AP");
    ccal->SaveAs("calibration.pdf");
```

`Fit()` method

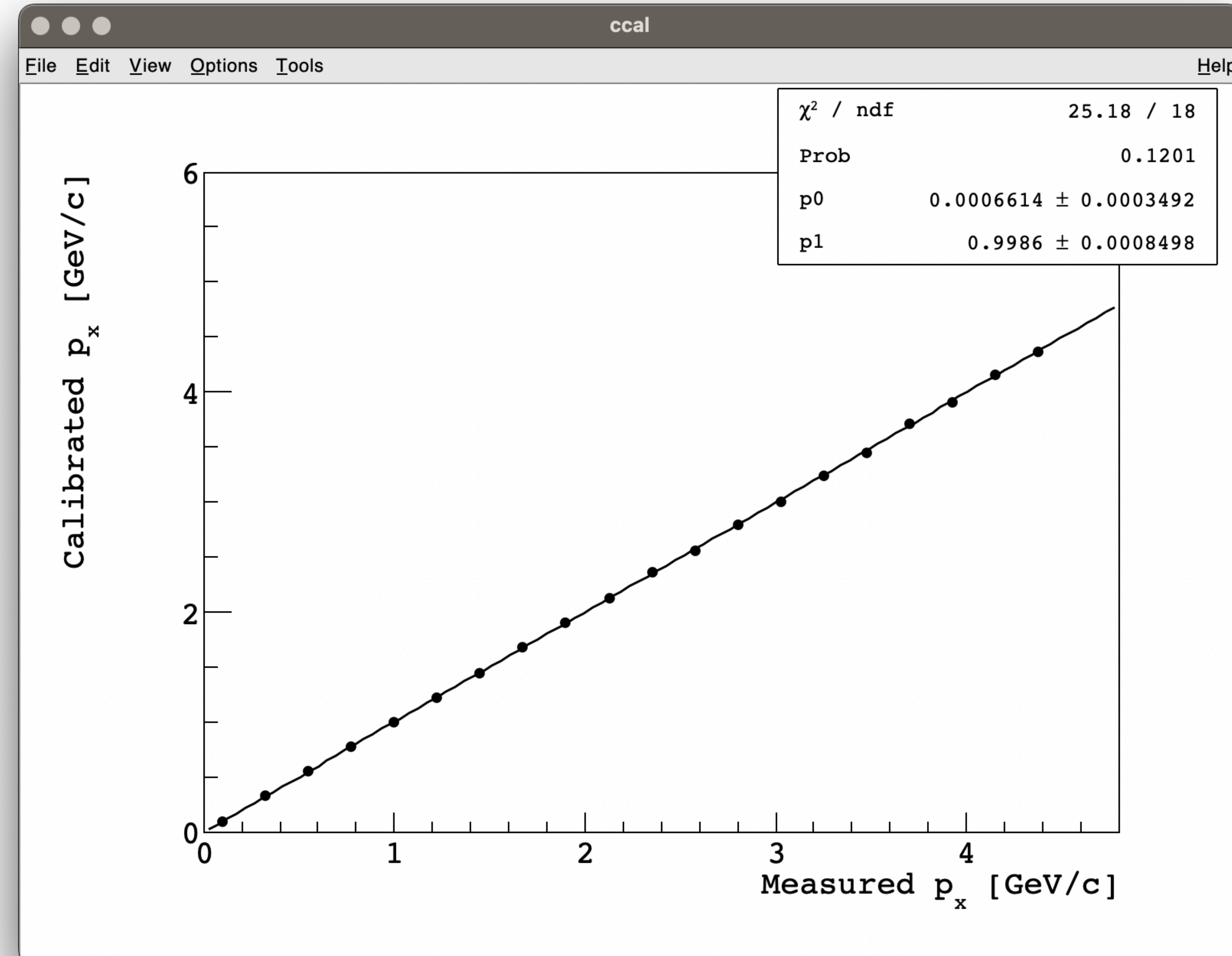
Value of the χ^2

Degrees of freedom (data — parameters)

```
mbp-md-01:lesson4_material dorigo$ root -l checkCalib.C
root [0]
Processing checkCalib.C...
****************************************
Minimizer is Linear / Migrad
Chi2                      =      25.1762
NDf                       =           18
p0                        =   0.00066135   +/-   0.000349226
p1                        =     0.998577   +/-   0.000849808
Info in <TCanvas::Print>: pdf file calibration.pdf has been created
```

Parameter results

Fit the data



ccal
File Edit View Options Tools
Help
χ² / ndf 25.18 / 18
Prob 0.1201
p0 0.0006614 ± 0.0003492
p1 0.9986 ± 0.0008498
Calibrated p_x [GeV/c]
Measured p_x [GeV/c]
0
1
2
3
4
0
2
4
6

Fit quality indicators

- The χ^2 value, compared to the degrees of freedom (dof), enables to calculate the *p* value of the fit (`TMath::Prob(chi2,dof)`).
  In our example, the χ^2 value is 25.2, dof is 18, and the *p* value is 12%.

- In addition, pulls are very useful:

$$\frac{y_i - f(x_i)}{\sigma_{y_i}}$$

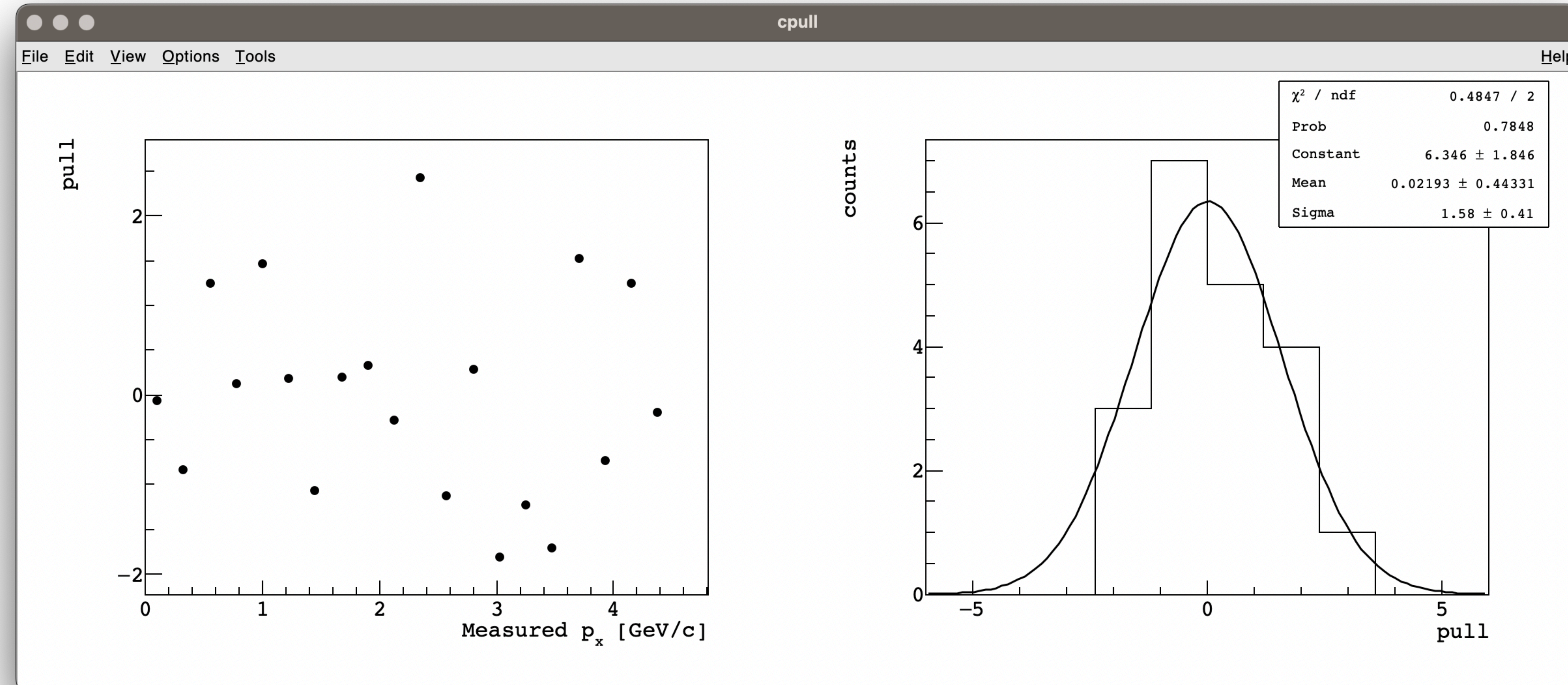
- For a good fit, their distribution should be a normal gaussian.

- Let's calculate and draw the pulls and their distribution.

Pulls check

```
    //it's very useful in a fit to draw the pulls:
    //the residual (fit - data) divided by the uncertainty
    //The pulls should be distributed as a normal gaussian.
    TGraph* gPulls = new TGraph(gCal->GetN());
    TH1D* hPulls = new TH1D("hPulls",";pull;counts", 10, -6, 6);

    for(int i=0; i<gCal->GetN(); ++i){

      double ydata = gCal->GetPointY(i);
      double xdata = gCal->GetPointX(i);
      double error = gCal->GetErrorY(i);

      //if you have error on x, you need to consider it.
      //Calculate it and add in quadrature to "error".

      double pull = (ydata - f_calib->Eval(xdata))/error;

      gPulls->SetPoint(i, xdata, pull);
      hPulls->Fill(pull);
    }
```

Pulls check


cpull
File Edit View Options Tools
Help
pull
Measured p_x [GeV/c]
counts
pull
χ² / ndf 0.4847 / 2
Prob 0.7848
Constant 6.346 ± 1.846
Mean 0.02193 ± 0.44331
Sigma 1.58 ± 0.41

Fitting with uncertainties on `x`

Often, our data have uncertainty also on the `x` values.
Take a look at the txt file `px_calibration_errX.txt`

x	y	err_x	err_y
0.1	0.100499	0.000166667	0.000333333
0.325	0.324294	0.000541667	0.00108333
0.55	0.55215	0.000916667	0.00183333
0.775	0.774884	0.00129167	0.00258333
1	1.00412	0.00166667	0.00333333
1.225	1.22465	0.00204167	0.00408333
1.45	1.44347	0.00241667	0.00483333
1.675	1.67437	0.00279167	0.00558333
1.9	1.90008	0.00316667	0.00633333
2.125	2.12064	0.00354167	0.00708333
2.35	2.36635	0.00391667	0.00783333
2.575	2.56232	0.00429167	0.00858333
2.8	2.79931	0.00466667	0.00933333
3.025	3.00317	0.00504167	0.0100833
3.25	3.23276	0.00541667	0.0108333
3.475	3.45088	0.00579167	0.0115833
3.7	3.7142	0.00616667	0.0123333
3.925	3.91056	0.00654167	0.0130833
4.15	4.16203	0.00691667	0.0138333
4.375	4.36664	0.00729167	0.0145833

Does a χ^2 fit consider the uncertainties on `x`? How can we do?

Fitting with uncertainties on x

Always have a look at the Reference Guide!

TGraphErrors fit:

In case of a TGraphErrors or TGraphAsymmErrors object, when x errors are present, the error along x, is projected along the y-direction by calculating the function at the points `x−ex_low` and `x+ex_high`, where `ex_low` and `ex_high` are the corresponding lower and upper error in x. The chi-square is then computed as the sum of the quantity below at each data point:

$$\frac{(y - f(x))^2}{ey^2 + (\frac{1}{2}(exl + exh)f'(x))^2}$$

where x and y are the point coordinates, and 'f'(x)` is the derivative of the function` f(x)`.

In case of asymmetric errors, if the function lies below (above) the data point, `ey` is `ey_low` (`ey_high`).

The approach used to approximate the uncertainty in y because of the errors in x is to make it equal the error in x times the slope of the line. This approach is called "effective variance method" and the implementation is provided in the function FitUtil::EvaluateChi2Effective

Notes on TGraph/TGraphErrors Fitting:

1. By using the "effective variance" method a simple linear regression becomes a non-linear case, which takes several iterations instead of 0 as in the linear case.
2. The effective variance technique assumes that there is no correlation between the x and y coordinate.
3. The standard chi2 (least square) method without error in the coordinates (x) can be forced by using option "EX0"
4. The linear fitter doesn't take into account the errors in x. When fitting a TGraphErrors with a linear functions the errors in x will not be considered. If errors in x are important, use option "F" for linear function fitting.

   When the fitting function is linear (contains the ++ sign) or the fitting function is a polynomial, a linear fitter is initialised.

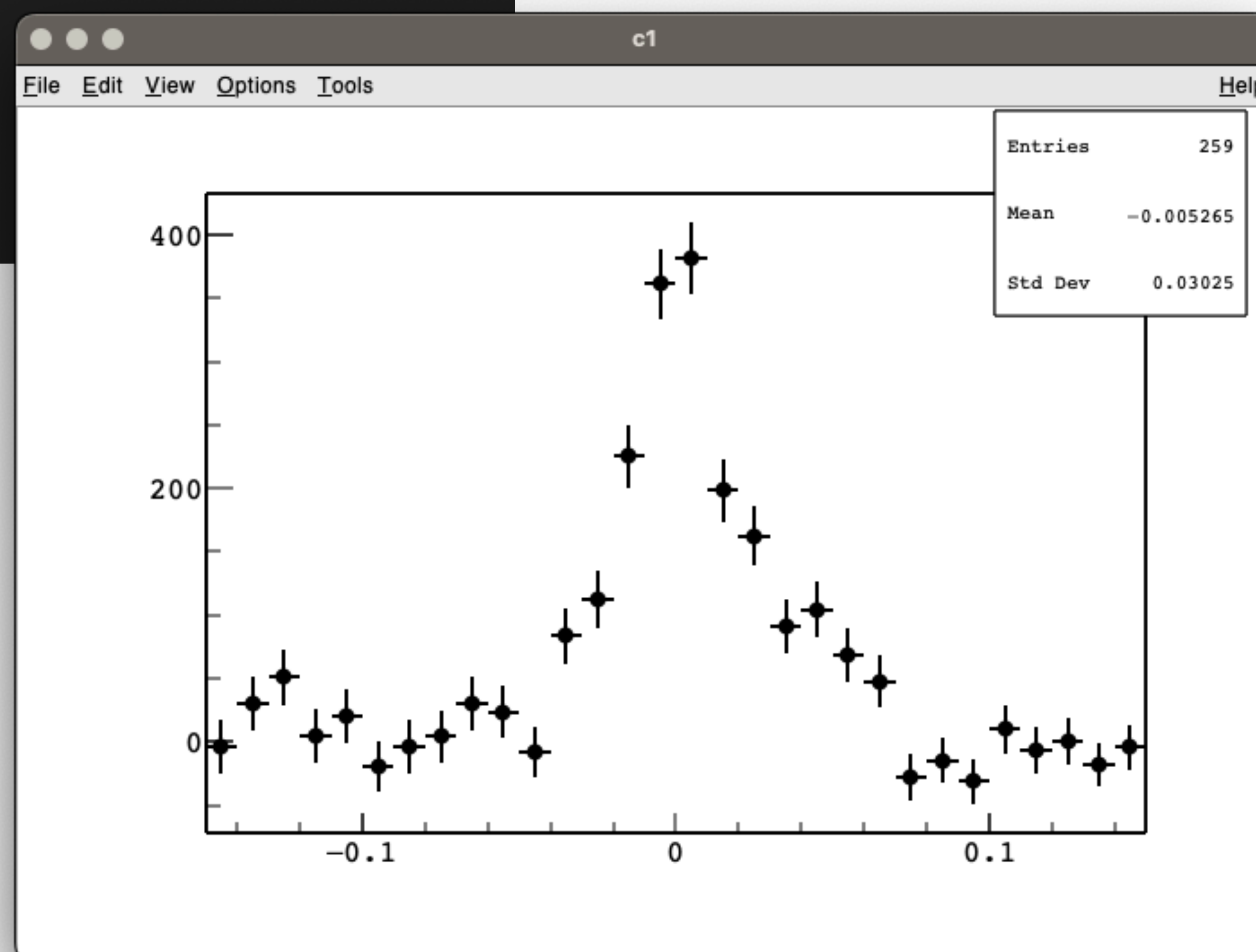
Exercises from last class

1. Modify histoPeak.C to plot the M distributions of the left and right ΔE sidebands. Compare the two distributions: plot them normalised in the same canvas and plot their ratios. Fit the ratio with a pol0 and a pol1 using the DrawPanel and comment the results

2. Obtain the ΔE signal distribution. To do that, proceed similarly to what we did in class: subtract the background from a signal-region histogram. To define the signal and background events, use:
signal for $M > 5.275\,\mathrm{GeV/c^2}$; background for $M < 5.275\,\mathrm{GeV/c^2}$.
When subtracting the background histogram, scale its integral by 0.4.

Making exercise 2 (from the prompt)

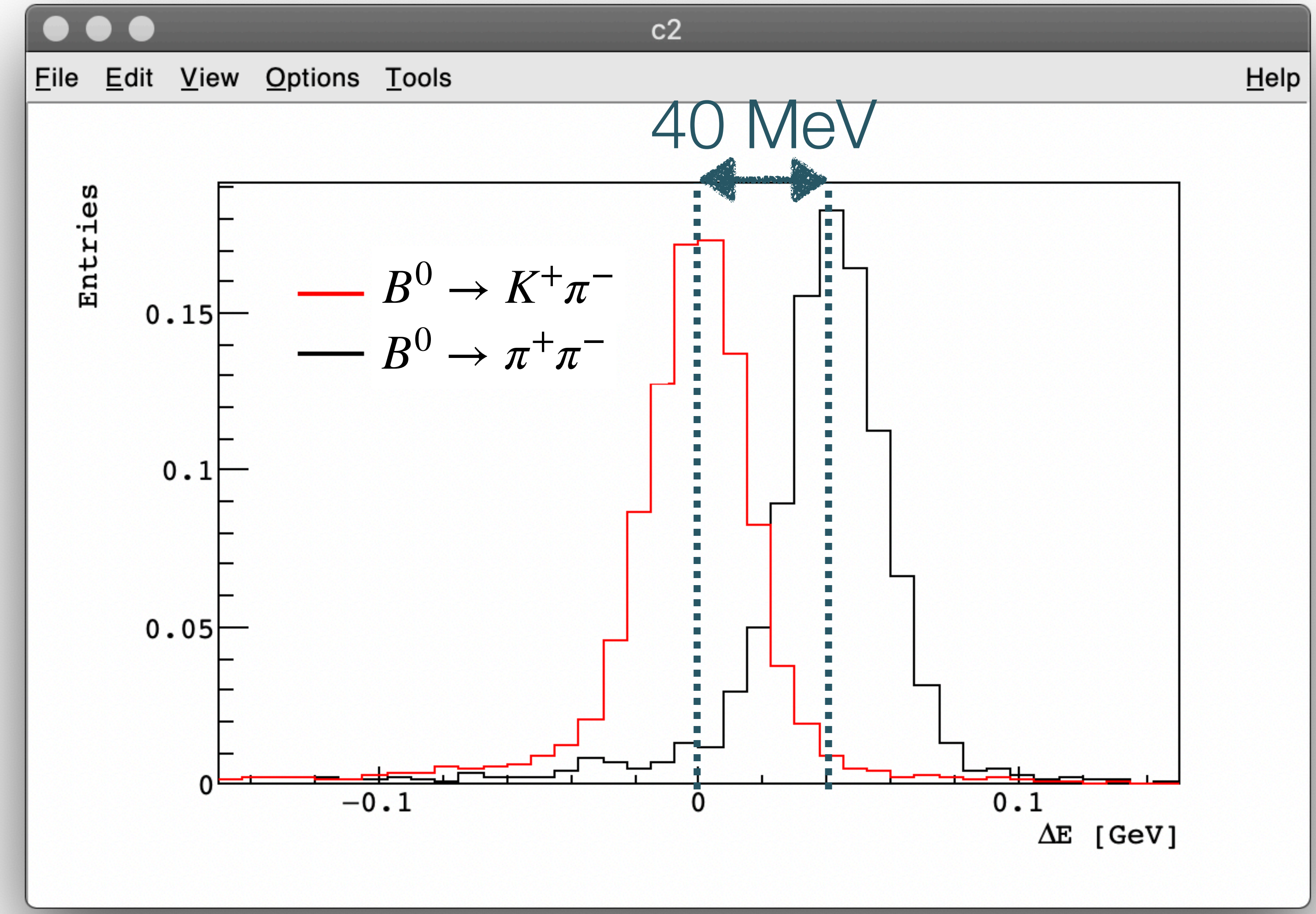
```
mb-md-01:thirdLesson dorigo$ root -l data_B0toKpi.root
root [0]
Attaching file data_B0toKpi.root as _file0...
(TFile *) 0x7fcc6cbb5d90
root [1] TH1D* hs = new TH1D("hs","hs",30,-0.15,0.15);
root [2] TH1D* hb = new TH1D("hb","hb",30,-0.15,0.15);
root [3] hs->Sumw2()
root [4] hb->Sumw2()
root [5] dataTree->Draw("B_de>>hs","B_m>5.275")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(long long) 10345
root [6] dataTree->Draw("B_de>>hb","B_m<5.275")
(long long) 21178
root [7] hs->Add(hb,-0.4)
(bool) true
root [8] hs->Draw()
```



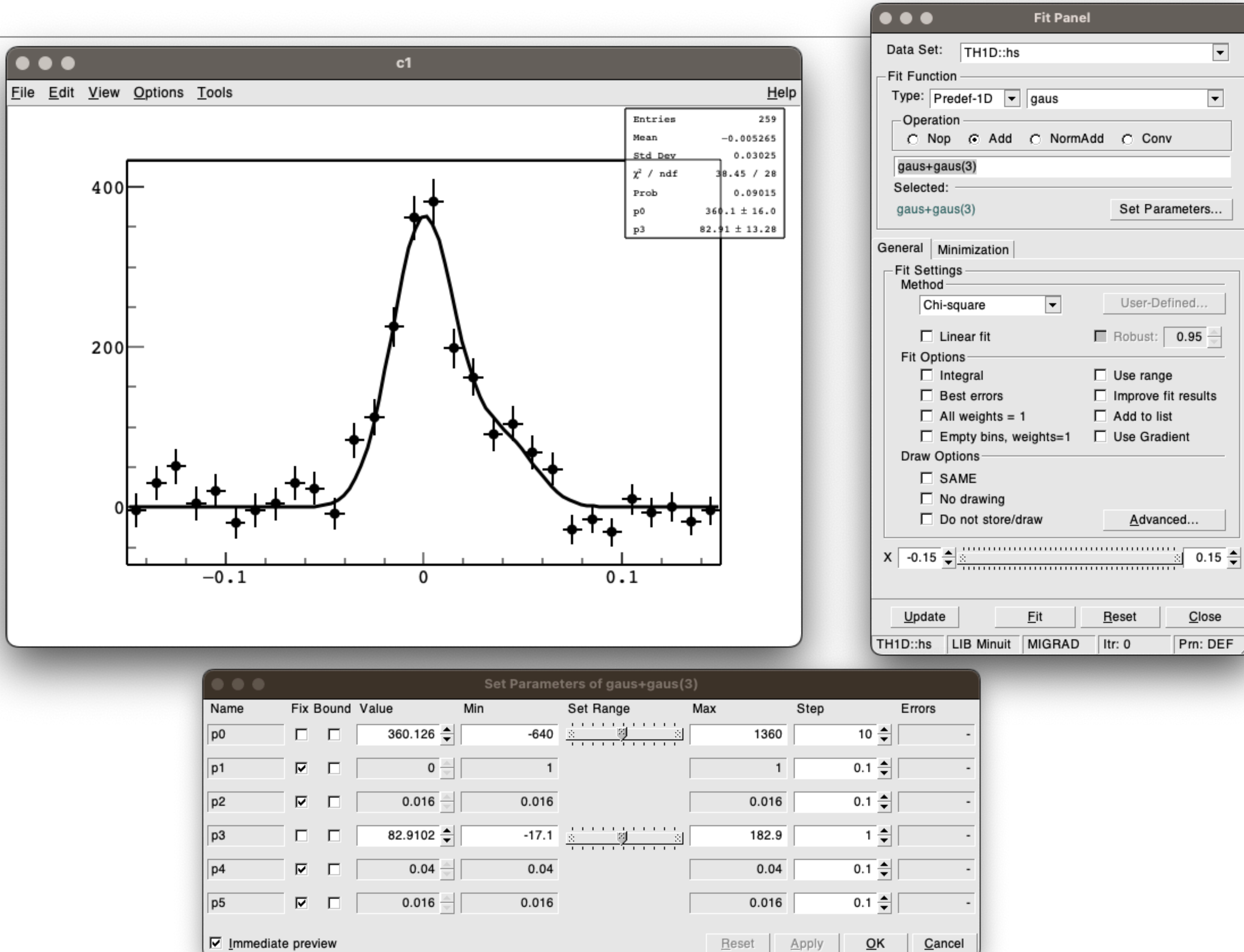
Background from other B decays

- Among $\Upsilon(4S) \to B\bar{B}$ 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 $B^0 \to K^+\pi^-$. These events events have the same M distribution as for the signal, but different ΔE.

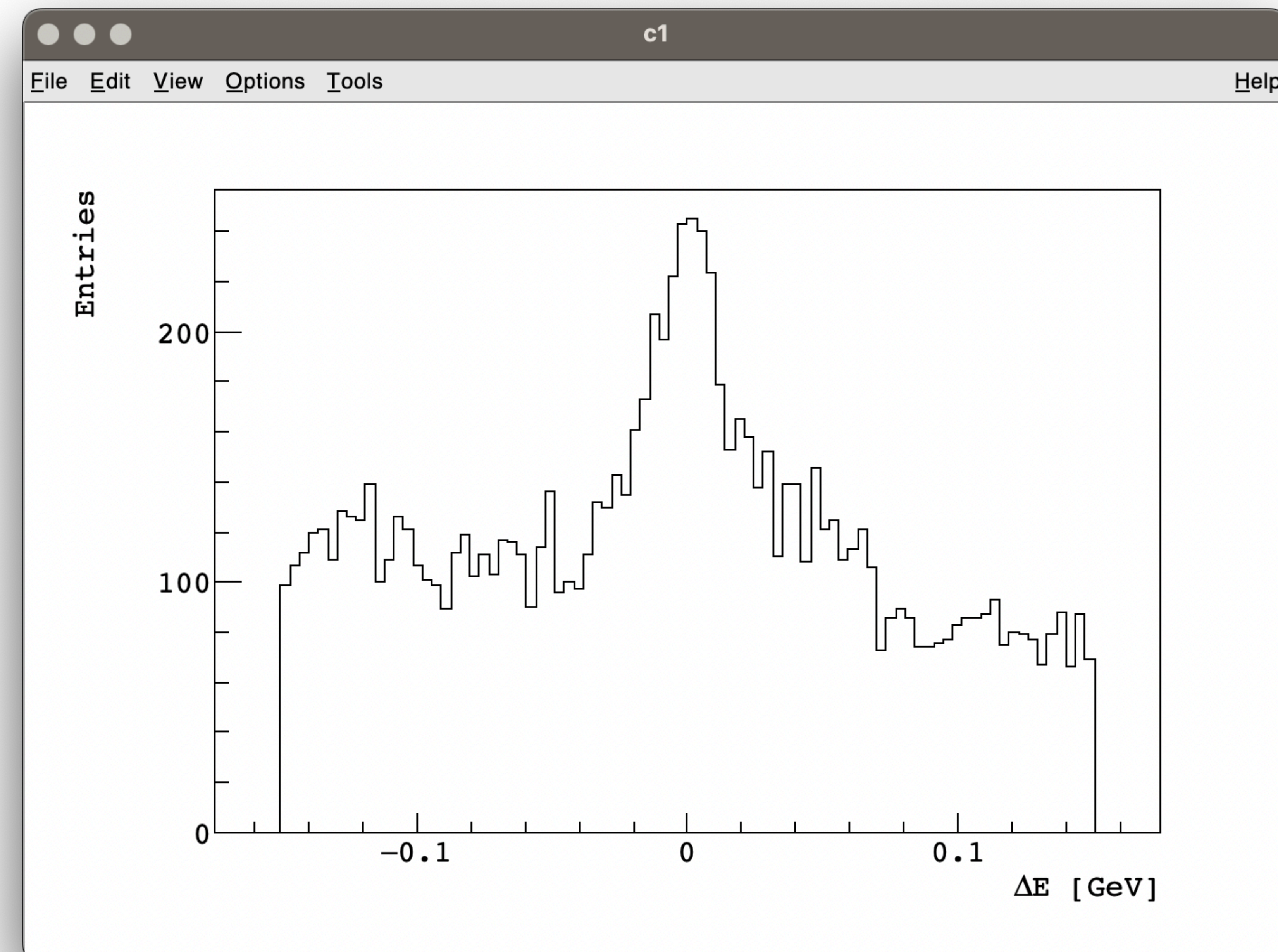


Let’s try to fit the two contributions



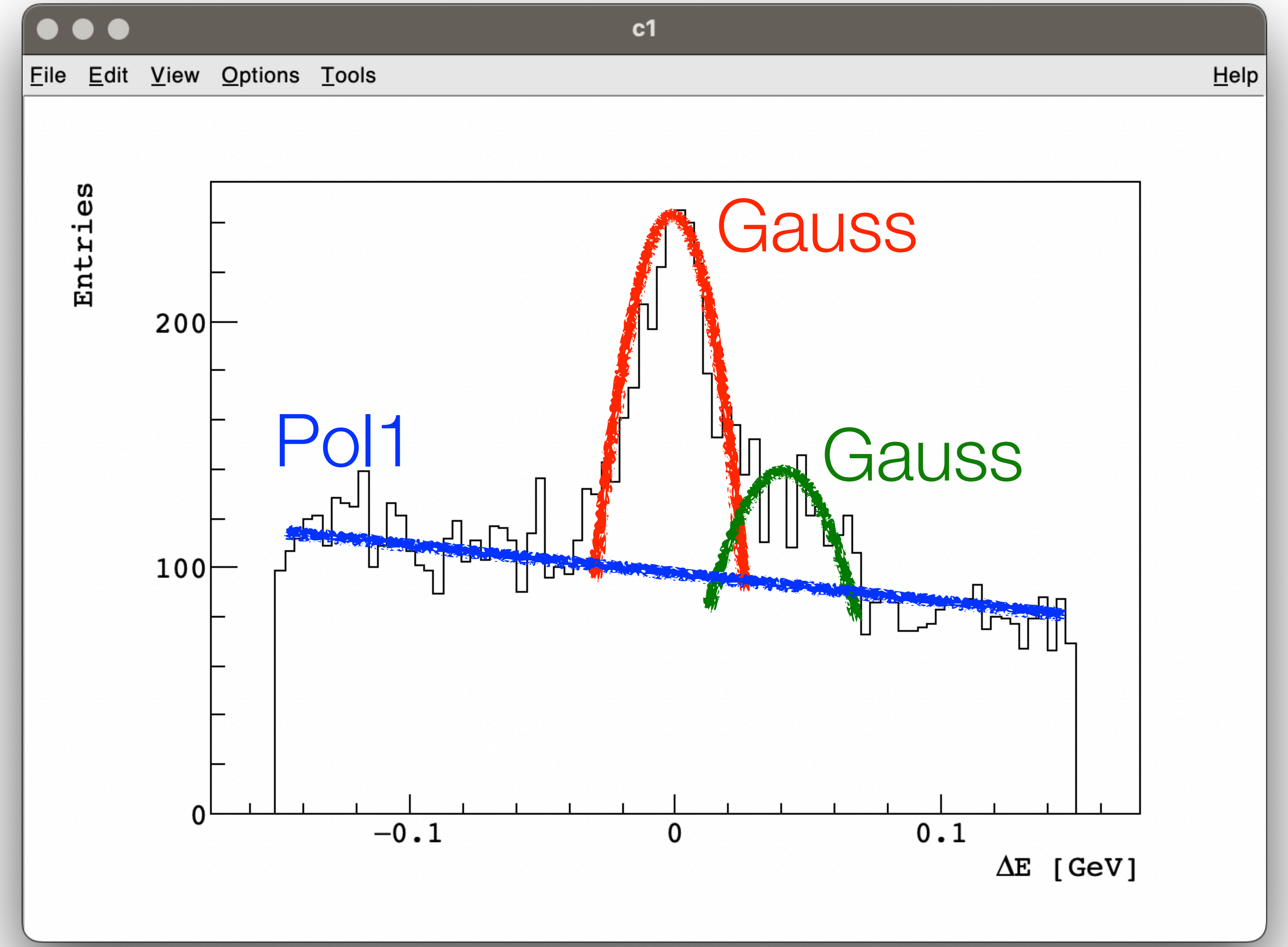
Second case: fit to an histogram

- Let's try now to fit the ΔE distribution to obtain the number of $B^0 \to K^+\pi^-$ candidates (our original goal).



Second case: fit to an histogram

- We need to model 3 components: the signal, the background and the misreconstructed decay.



Take `fitDeltaE.C`

First part pretty standard now…

```
void fitDeltaE(){

    double min_de=-0.15;
    double max_de= 0.15;
    //define an histogram to look at deltaE distribution
    TH1D* h_data = new TH1D("h_data",";#DeltaE [GeV]; Entries",30,min_de,max_de);

    //open file and take the tree
    TFile* file = TFile::Open("data_B0toKpi.root");
    TTree* tree = (TTree*) file->Get("dataTree");

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

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

    //loop over the entries and fill the histogram
    for(int iEntry=0; iEntry<tot_entries; ++iEntry){

        tree->GetEntry(iEntry);

        //can remove some trivial background
        if(B_m<5.275) continue;

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

Define the pdf (function for the fit)

```
//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,  "Norm_{sig}");
pdf->SetParameter(0,  400);//some starting value
//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(2,0.016);
//mis-id gauss, normalisation constant
pdf->SetParName  (3,  "Norm_{misid}");
pdf->SetParameter(3,   40);//some starting value
//mis-id gauss, mean fixed
pdf->SetParName  (4,  "#mu_{misid}");
pdf->FixParameter(4,0.040);
//mis-id gauss, std dev fixed
pdf->SetParName  (5,  "#sigma_{misid}");
pdf->FixParameter(5,0.016);
//background intercept and slope
pdf->SetParName  (6,  "p_{0}^{bkg}");
pdf->SetParName  (7,  "p_{1}^{bkg}");
```

TF1 function

Settings of parameters.
We fix parameters that we know already (from physics) 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 definined 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()` (for TH1) in the reference guide.

Take `fitDeltaE.C`

Value of the fit function (χ^2 here)

Degrees of freedom (number of bins — parameters)

```
 First fit, fixing all possible parameters:

****************************************
Minimizer is Minuit2 / Migrad
Chi2                      =      39.5581
NDf                       =           26
Edm                       =  3.11309e-22
NCalls                    =           78
Norm_{sig}                =      365.121   +/-   15.3378
#mu_{sig}                 =            0                    (fixed)
#sigma_{sig}              =        0.016                    (fixed)
Norm_{misid}              =       95.717   +/-   12.6453
#mu_{misid}               =         0.04                    (fixed)
#sigma_{misid}            =        0.016                    (fixed)
p_{0}^{bkg}               =       278.54   +/-   3.69758
p_{1}^{bkg}               =     -422.157   +/-   35.6279
```

The parameter 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";
pdf->FixParameter(5, pdf->GetParameter(2)); //signal gauss, std dev fixed
//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 (option L), instead of the default χ^2

Take `fitDeltaE.C`

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

****************************************
Minimizer is Minuit2 / Migrad
Chi2                      =       37.743
NDf                       =           25
Edm                       =  1.01343e-06
NCalls                    =          111
Norm_{sig}                =      380.533   +/-    19.5773
#mu_{sig}                 =            0                      (fixed)
#sigma_{sig}              =    0.0147713   +/-    0.00087106
Norm_{misid}              =      103.281   +/-    13.6478
#mu_{misid}               =         0.04                      (fixed)
#sigma_{misid}            =        0.016                      (fixed)
p_{0}^{bkg}               =      279.435   +/-    3.74749
p_{1}^{bkg}               =     -427.845   +/-    35.8358


 Update the mis-id std dev
 and do a binned-likelihood fit, instead of a chi2

Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
****************************************
Minimizer is Minuit2 / Migrad
MinFCN                    =      19.3246
Chi2                      =      38.6491
NDf                       =           25
Edm                       =  4.56442e-09
NCalls                    =          112
Norm_{sig}                =      382.229   +/-    19.1769
#mu_{sig}                 =            0                      (fixed)
#sigma_{sig}              =    0.0148023   +/-    0.000834354
Norm_{misid}              =      106.108   +/-    14.0808
#mu_{misid}               =         0.04                      (fixed)
#sigma_{misid}            =    0.0147713                      (fixed)
p_{0}^{bkg}               =      281.097   +/-    3.76375
p_{1}^{bkg}               =     -424.788   +/-    36.039
```

2nd fit results, releasing the sigma for the signal

3rd fit results. Use the binned likelihood here.

Take `fitDeltaE.C`

```
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));
pdf_misid->SetLineColor(kGreen+3);
pdf_misid->SetLineWidth(2);
pdf_misid->Draw("same");

//the background
TF1* pdf_bkg = new TF1("pdf_bkg","pol1",min_de,max_de);
pdf_bkg->SetParameters(pdf->GetParameter(6),
                       pdf->GetParameter(7));
pdf_bkg->SetLineColor(kBlue);
pdf_bkg->SetLineWidth(2);
pdf_bkg->SetLineStyle(2);
pdf_bkg->Draw("same");

TLegend* leg = new TLegend(0.18,0.55,0.45,0.85);
leg->SetBorderSize(0);
leg->AddEntry(h_data,"Data","PL");
leg->AddEntry(pdf,"Fit","L");
leg->AddEntry(pdf_sig,"B^{0} #rightarrow K^{+}#pi^{-}","L");
leg->AddEntry(pdf_misid,"B^{0} #rightarrow #pi^{+}#pi^{-}","L");
leg->AddEntry(pdf_bkg,"background","L");
leg->Draw();
```

Just nice drawing of the results… let's draw each component separately: we need to define its function and set the parameters from the fit results

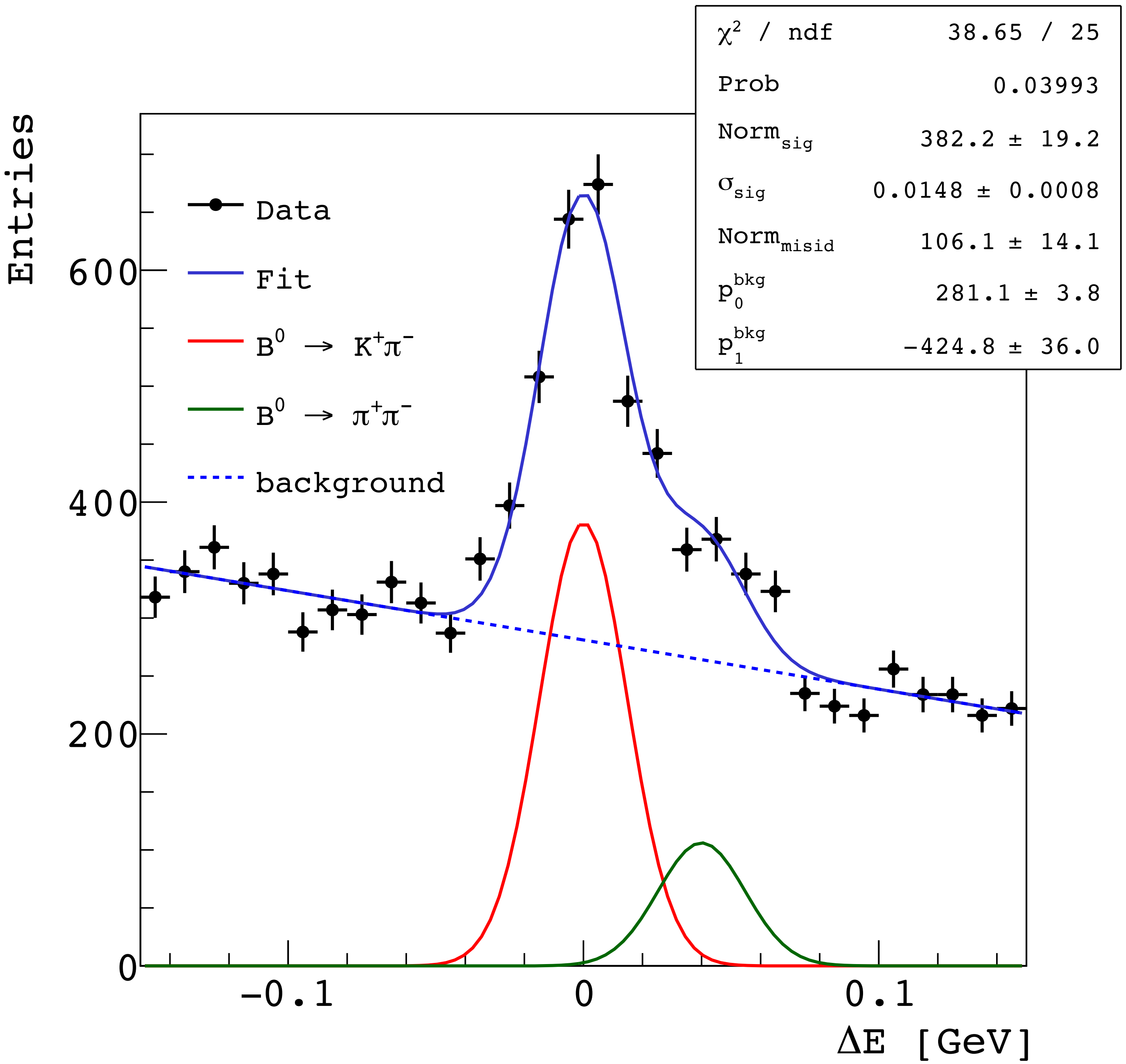
Set a legend in the plot: TLegend class

Take `fitDeltaE.C`

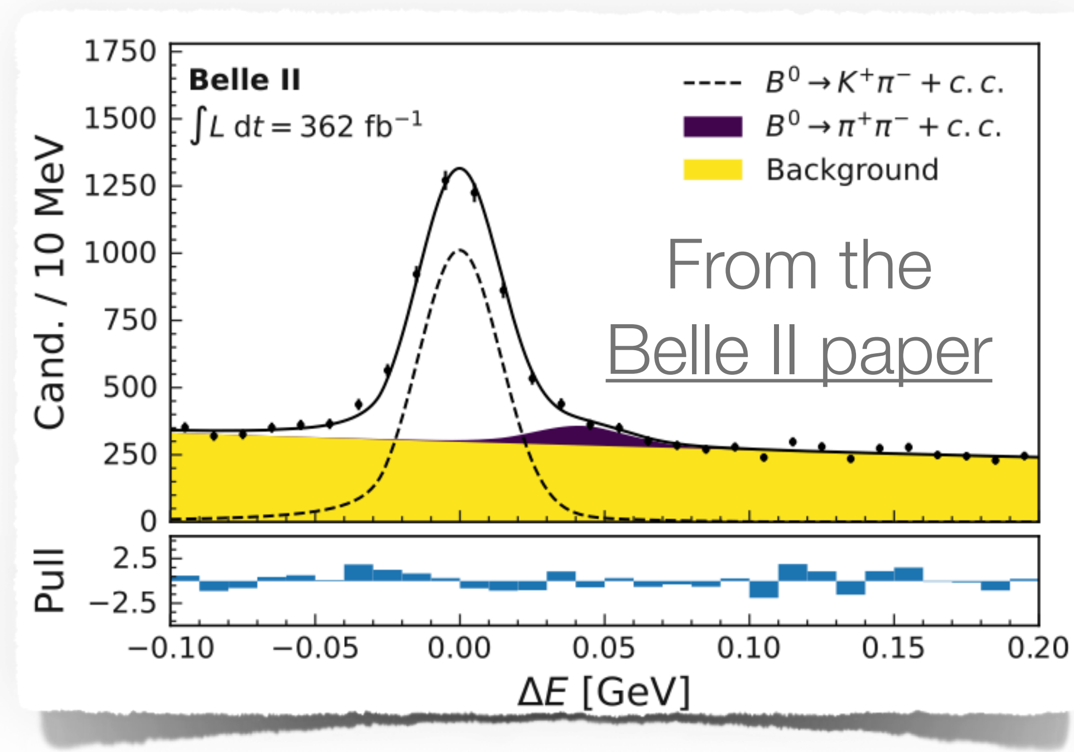
Retrieve the information we want: the yield of the components

```
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;

```

We made it (?)




```
 From this fit model,
Candidate in data histogram: 10244
Total candidates from fit  : 10244
Signal B->Kpi candidates   : 1418.22
Mis-id B->pipi candidates  : 392.878
Background candidates      : 8432.91
```



The uncertainty is missing!

- We didn't compute the uncertainty on the signal yield!
- We used a gauss pdf for the signal, its integral (divided by the bin width w) gives the signal yield:

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

- 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

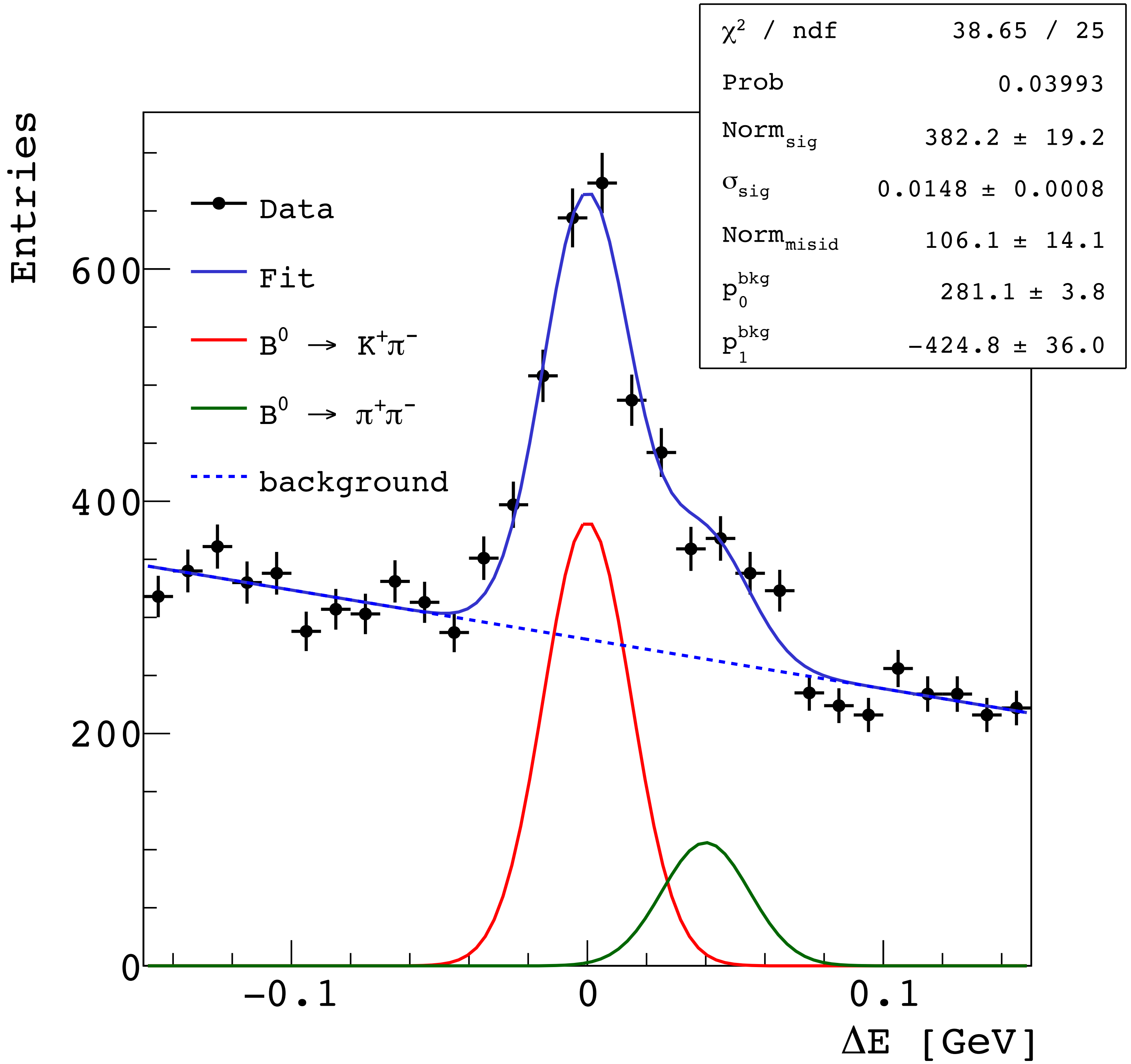
- Small addition in fitDeltaE.C

```
cout << "==================================================" << endl;
cout << " Let's calculate the final result with its uncertainty \n " << endl;

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

- Now we have the full information

```
//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);
```

Calculate S and its uncertainty




```
The measurement of the signal yield is 1418 +- 72
```

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).
  If so, please share your feedback in the course evaluation form.
- Take your time to revisit all material and try it yourself.
  For questions, doubts, curiosity don't hesitate to contact me.
  We can organise Q&A sessions.
- If you are into data analysis at a particle physics experiment,
  come to talk about opportunities in Belle II.

Exercises

1. Run `checkCalib.C` taking in input `px_calibration_errX.txt` and make the necessary modification to the macro (see the comments therein).
   Study the differences with respect to case seen in class.

2. Add the pulls (using a TH1D instead of a TGraph) to the fit of the ΔE distribution and check their distribution.

3. Compute the correlation between the parameters in the calibration fit.

4. Not a root exercise, but useful for the final exam. Consider the efficiency for a requirement, defined as the ratio $\varepsilon = P/N$, where P is the number of events that pass the requirement out of N total events. Calculate the uncertainty on ε.