Introduction to ROOT: part 3

Mirco Dorigo
mirco.dorigo@ts.infn.it

LACD 2023-2024
March 27th, 2024

Previous lesson

- Had a text file with momentum components of kaon and pion from Belle II data that should be candidates $B^0 \to K^+\pi^-$ decays.
- We have seen how to:
  - read the data from the text file;
  - compute a new variable (momentum, using e.g. `TVector3`);
  - make an histogram (`TH1D`) and draw it (`TCanvas`) and explore the histogram online;
  - store the data in a n-tuple (`TTree`) and save in a ROOT file (`TFile`).

Exercises

- We still have to see a signal peak…
- Let's build the variables. Calculate the mass $M = \sqrt{s/4 - |\vec{p}_B^*|^2}$, by using the class `TLorentzVector`.
- Another useful variable is the difference between the B-candidate energy in the CMS and half of the collision energy, $\Delta E = E^* - \sqrt{s}/2$. Calculate the variable.
- Plot the distribution of M and that of ΔE into two canvas.
  Is this what you expected? Describe the distributions (mean, standard dev…).
- Add the variable to your tree, and save the tree in a file, adding also the two canvas showing the distributions.

Breaking the exercise

```
#include "Riostream.h"
#include "TString.h"
#include "TH1D.h"
#include "TCanvas.h"
#include "TTree.h"
#include "TFile.h"
#include "TLorentzVector.h"
```

include the class

```
    double k_px,  k_py,  k_pz;
    double pi_px, pi_py, pi_pz;
    double B_m, B_de; // the variables that I want to calculate

    TTree* dataTree = new TTree("dataTree","B0toKpi data");
    dataTree->Branch("k_px",&k_px,"k_px/D");
    dataTree->Branch("k_py",&k_py,"k_py/D");
    dataTree->Branch("k_pz",&k_pz,"k_pz/D");
    dataTree->Branch("pi_px",&pi_px,"pi_px/D");
    dataTree->Branch("pi_py",&pi_py,"pi_py/D");
    dataTree->Branch("pi_pz",&pi_pz,"pi_pz/D");
    //add the two new variables to the tree
    dataTree->Branch("B_m",&B_m,"B_m/D");
    dataTree->Branch("B_de",&B_de,"B_de/D");

    //Let's define the histograms to look at the distributions
    TH1D* h_m = new TH1D("h_m"," ",40,5.25,5.30);
    TH1D* h_de = new TH1D("h_de"," ",40,-0.15,0.15);

    //usefull constants
    const double pi_m = 0.13957018; //pion mass in GeV/c2
    const double k_m = 0.493667;    //kaon mass in GeV/c2
    const double sqs = 10.5794;     //cms energy in GeV (Y(4S) mass...)
```

New variables in the tree

Histograms

Taken from PDG

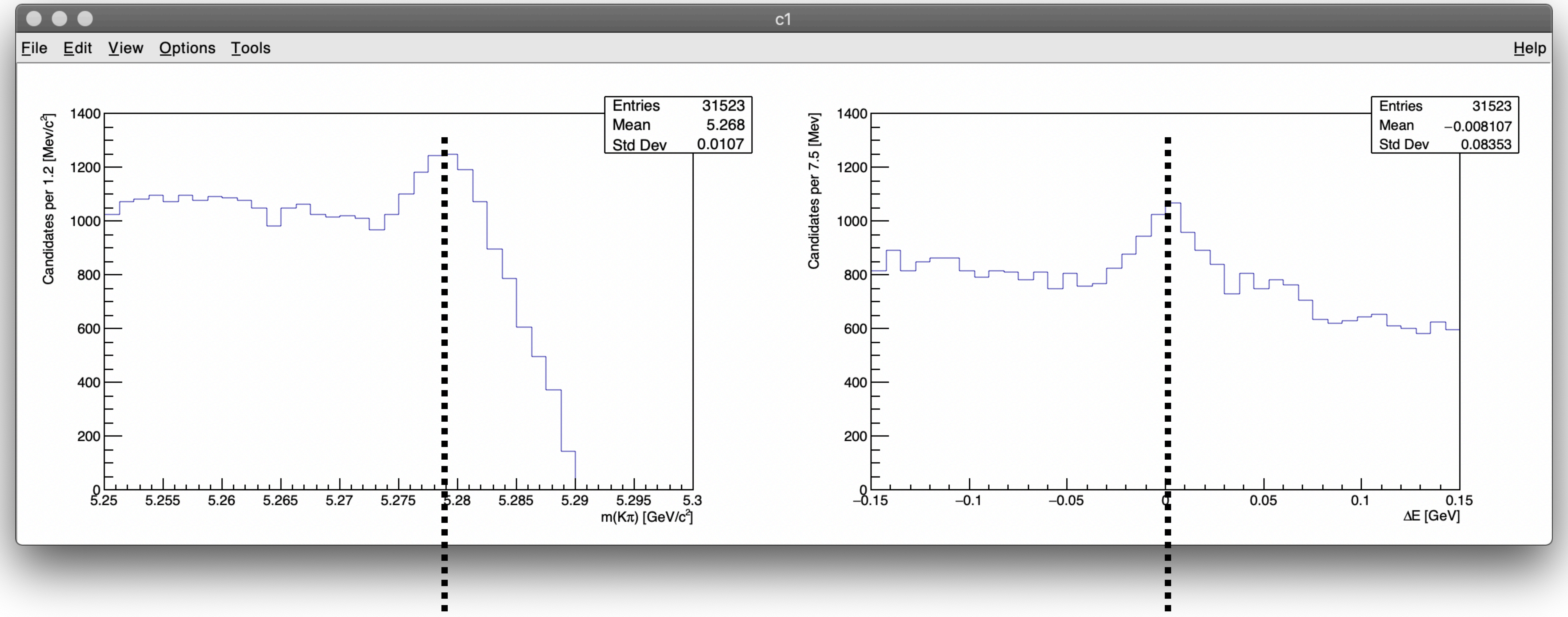
Breaking the exercise

```
while(file_in.is_open()){

    file_in >> k_px  >> k_py  >> k_pz
            >> pi_px >> pi_py >> pi_pz;

    if(file_in.eof()) break;

    //define the 4-momentum of the pion and the kaon
    TLorentzVector pi_p, k_p;
    pi_p.SetXYZM(pi_px,pi_py,pi_pz,pi_m);//set the components for the pion
    k_p.SetXYZM(k_px,k_py,k_pz,k_m); //and for the kaon

    TLorentzVector B_p = pi_p+k_p;//the B is the sum of the pion and kaon

    B_de = B_p.E() - sqs/2; //easy to get the energy
    B_m = sqrt( sqs*sqs/4 - B_p.Vect().Mag2() ); //and the mass

    //fill my histograms
    h_m->Fill(B_m);
    h_de->Fill(B_de);

    //fill the tree
    dataTree->Fill();

}
```

Breaking the exercise

```
    //save everything in a file
    TFile* dataFile = new TFile("data_B0toKpi.root","RECREATE");
    dataTree->Write();
    h_m->Write();
    h_de->Write();
    dataFile->Close();

    //let's make some plot
    gStyle->SetOptStat(1110);//this is a global style set
    TCanvas* c1 = new TCanvas("c1","c1",1200,400);
    c1->Divide(2,1);//I split my canvas into two part (called pad)
    c1->cd(1);//and go into the first pad
    h_m->GetXaxis()->SetTitle("m(K#pi) [GeV/c^{2}]"); //set title x
    h_m->GetYaxis()->SetTitle(Form("Candidates per %.1f [Mev/c^{2}]",
                               1.e3*h_m->GetXaxis()->GetBinWidth(1)));//title y
    h_m->GetYaxis()->SetRangeUser(0,1400); //set the interval to draw in y
    h_m->Draw(); // and draw

    c1->cd(2);//go to the second pad, and draw the other histogram
    h_de->GetXaxis()->SetTitle("#DeltaE [GeV]");
    h_de->GetYaxis()->SetTitle(Form("Candidates per %.1f [Mev]",
                               1.e3*h_de->GetXaxis()->GetBinWidth(1)));
    h_de->GetYaxis()->SetRangeUser(0,1400);
    h_de->Draw();

    return;
```

The peak



m(B^0) ~ 5.280 GeV/c^2

Expect ~0 for a B^0

Let's explore the data online

- You can draw your data in the tree from the prompt

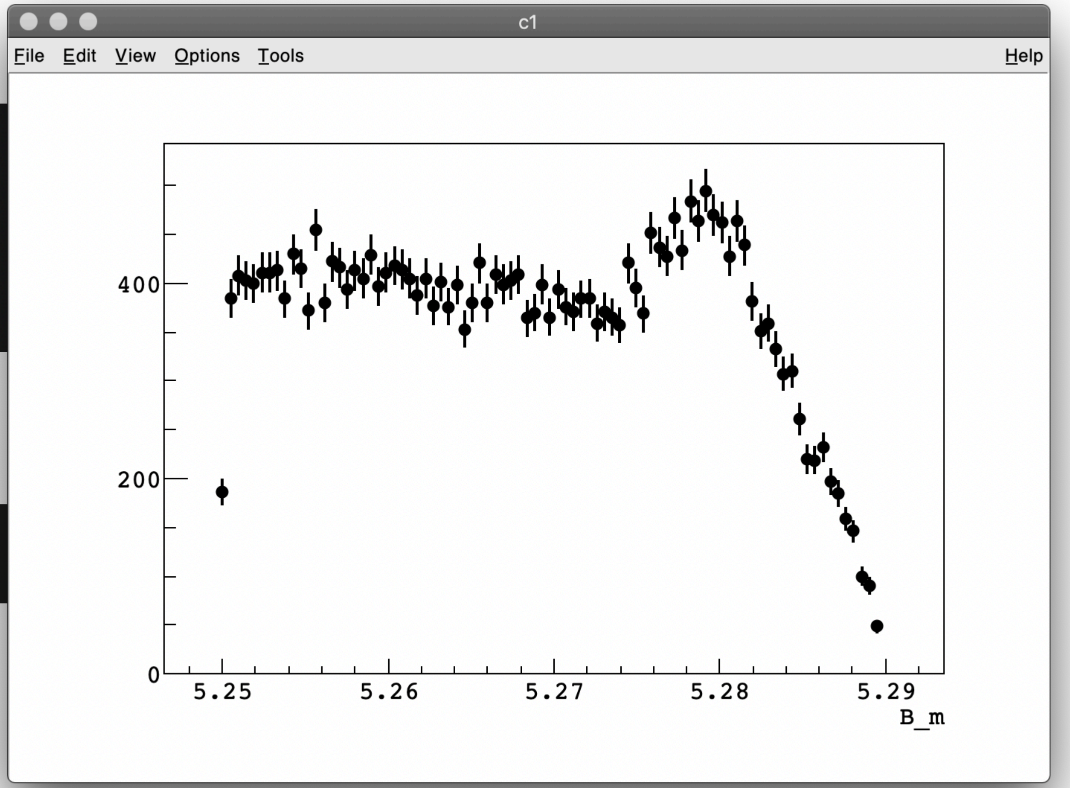
```
mb-md-01:thirdLesson dorigo$ rootl data_B0toKpi.root
root [0]
Attaching file data_B0toKpi.root as _file0...
(TFile *) 0x7fd8ce708370
root [1] .ls
TFile**         data_B0toKpi.root
 TFile*         data_B0toKpi.root
  KEY: TTree    dataTree;1      B0toKpi data
  KEY: TH1D     h_m;1
  KEY: TH1D     h_de;1
root [2] dataTree->Draw("B_m")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [3]
```

- Making also selections

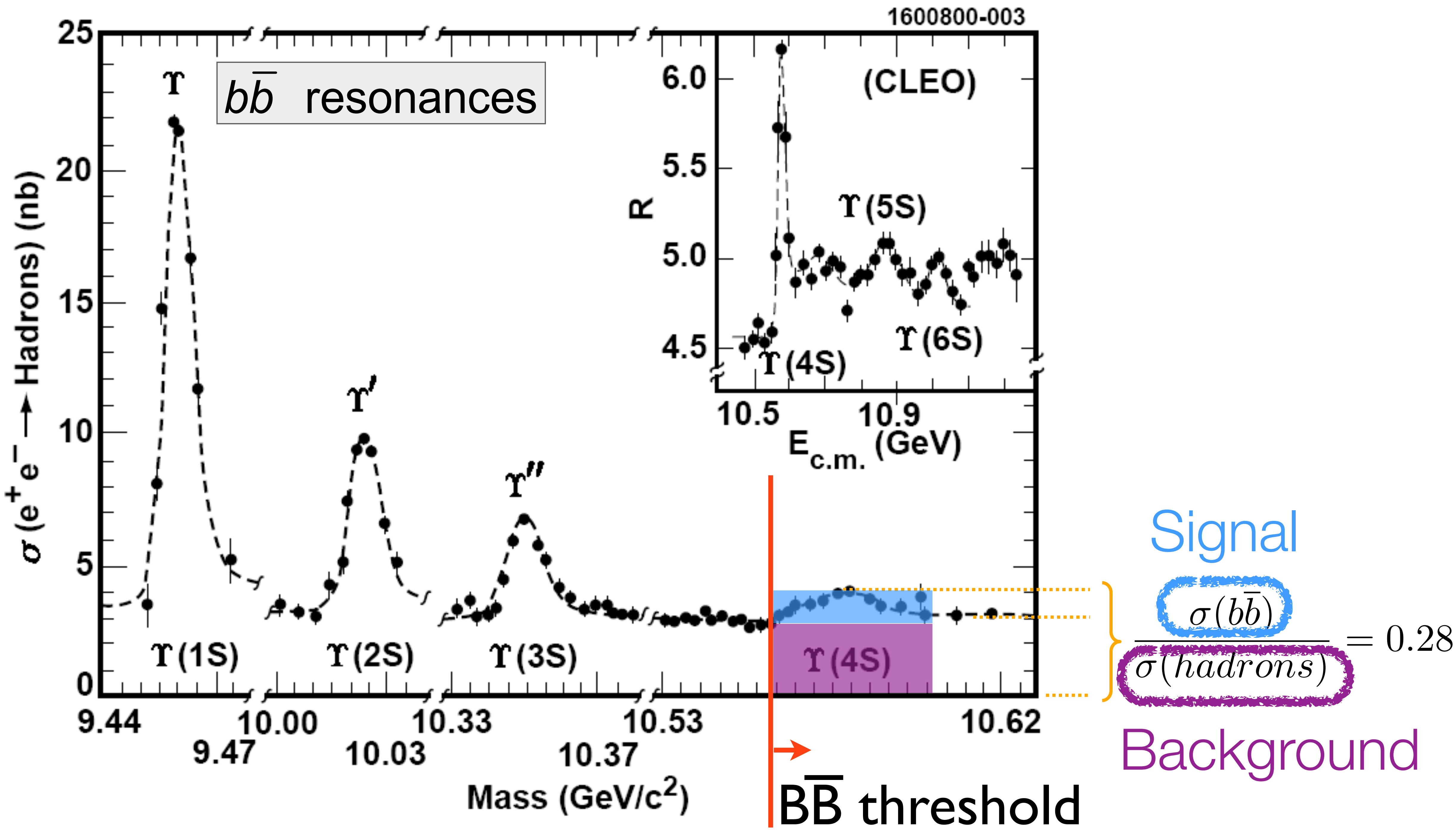
```
root [3] dataTree->Draw("B_m","B_m>5.27")
(long long) 14370
root [4] dataTree->Draw("B_m","B_m>5.25")
(long long) 31343
root [5]
```

- And adding draw options

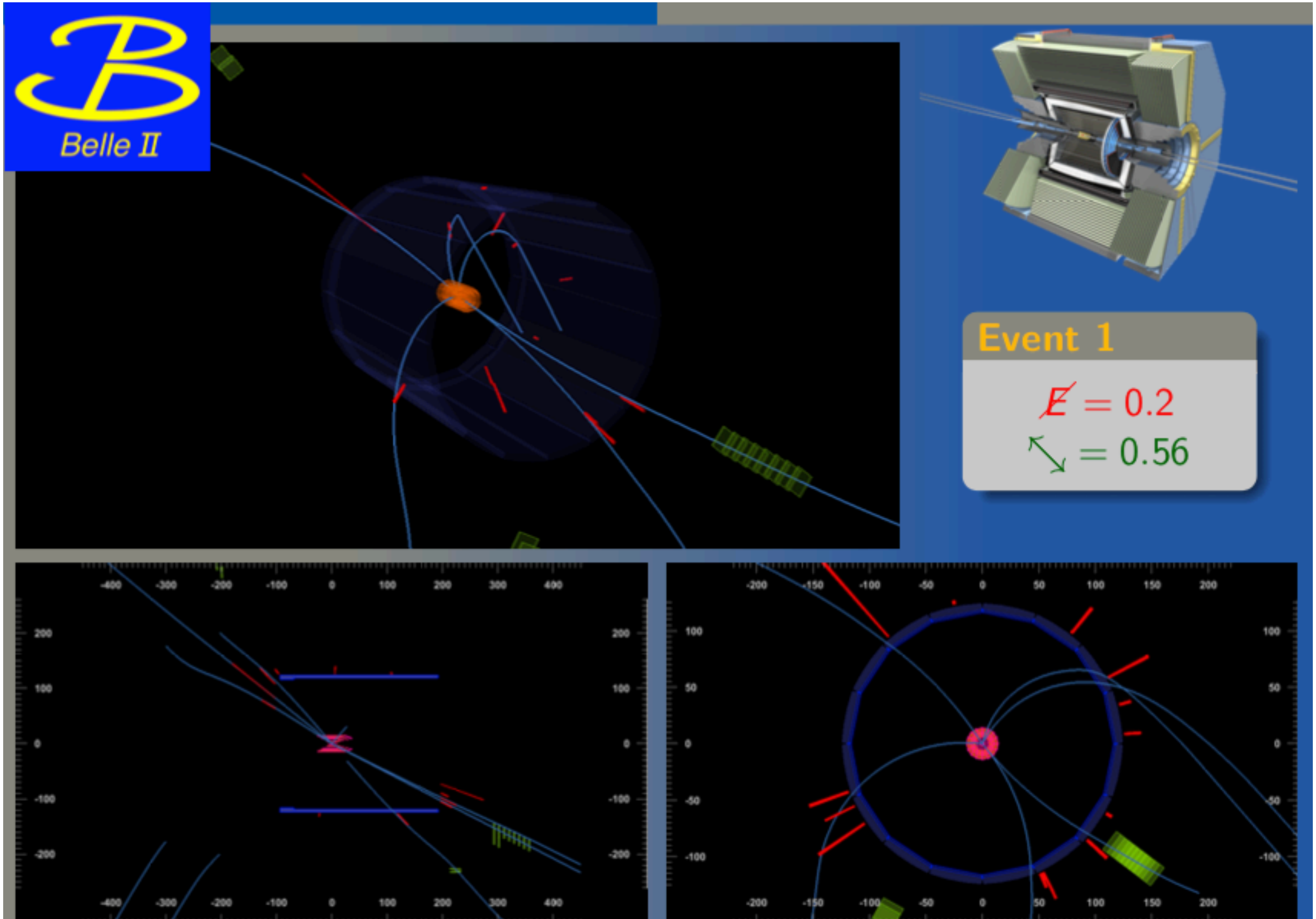
```
root [5] dataTree->Draw("B_m","B_m>5.25","err")
(long long) 31343
```



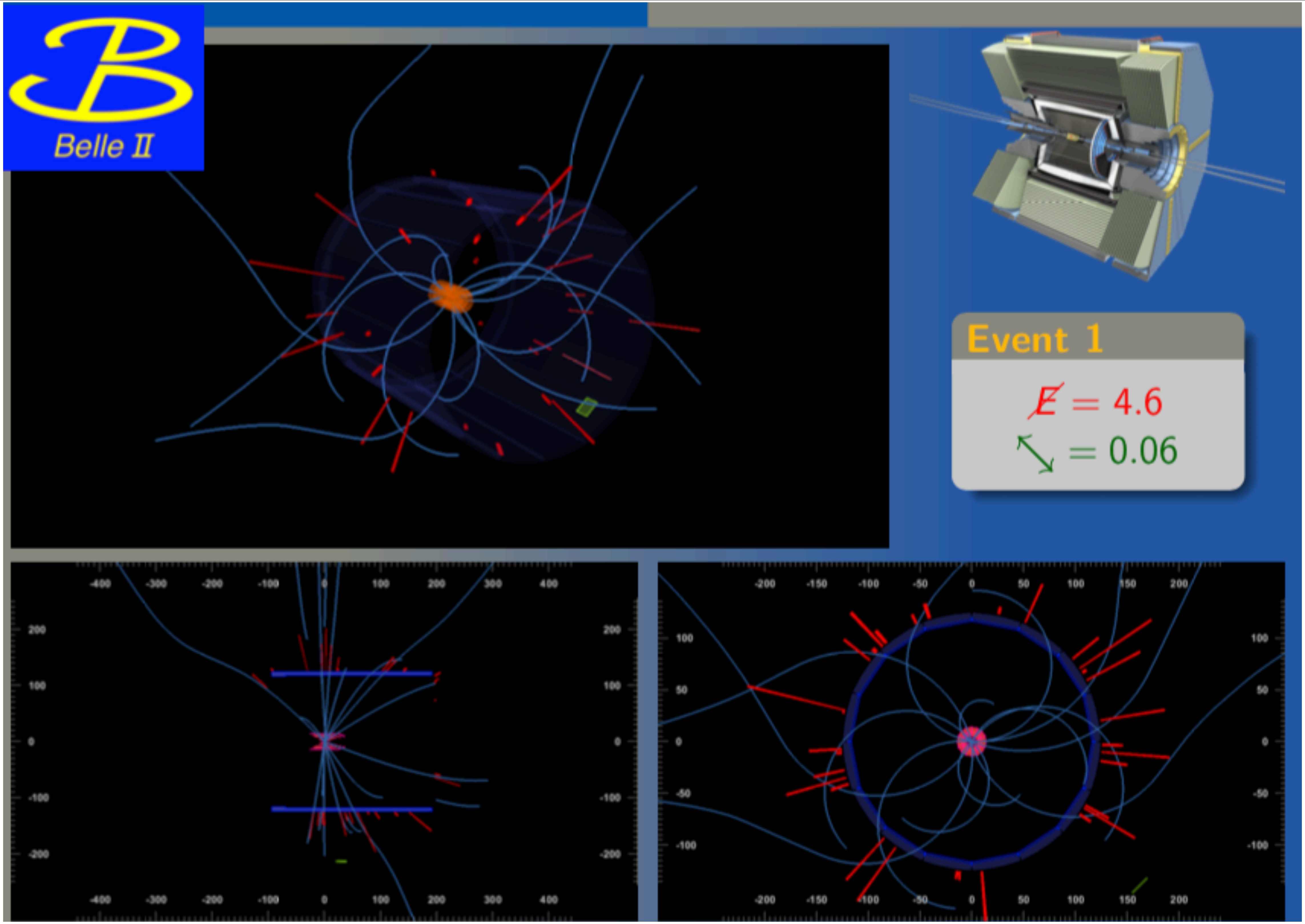
There is a lot of background



$q\overline{q}$ event



$B\bar{B}$ event



Let’s make a selection

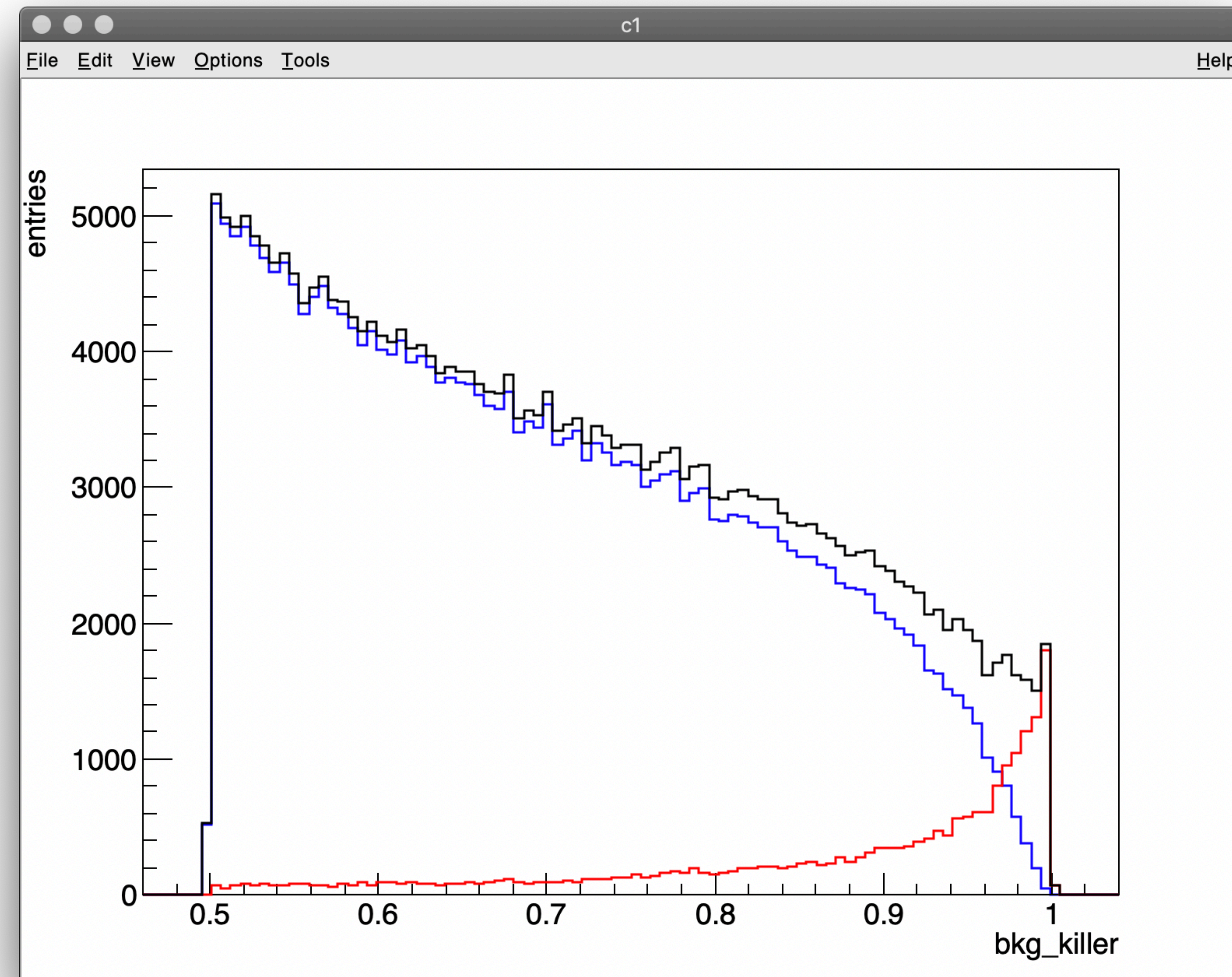
- The sample features background that dilutes our signal sensitivity.
- Our colleagues developed a smart way to distinguish signal from background, and gave us a n-tupla with a new variable.
- It is the output of a classifier that gives the probability of a candidate to be signal. The classifier, a “boosted decision tree” (BDT), is trained on signal and background simulated data, using 39 input variables. But we don’t care how it’s build, we just care about its capability to distinguish background from signal.
- Let’s use it to get rid of background and enhance signal sensitivity.

Let’s make a selection

- We will use simulated data: we generated a much larger sample than the data sample, simulating all physics processes and reconstructing all candidates as for the data.
- In simulation we know what is signal and what is background.
- So, let’s take the file `simulation.root` and explore it.
- Then, we will need to read this ROOT file in a macro.

The background killer

- This is the output of the classifier in our simulation, separated for signal, background, and their sum.



Reading a tree (`readTree.C`)

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

using namespace std;

void readTree(){

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

    //just a trivial check
    long tot_entries = tree->GetEntries();
    cout << "Total entries in the tree: " << tot_entries << endl;

    //define the variable we want to access to
    double B_m;
    int isBkg;
    //and link them to the branch address of the tree
    tree->SetBranchAddress("B_m",&B_m);
    tree->SetBranchAddress("isBkg",&isBkg);
```

Use directly the method while defining the (pointer to the) object

`Get()` is general from `TObject`, we need to “cast” the type

Very similar to the definition of the branches…

Reading a tree (`readTree.C`)

```
    //just two istogram to fill
    TH1D* h_m_sig = new TH1D("h_m_sig",";m(B) [GeV/c^{2}]; Entries",30,5.25,5.30);
    TH1D* h_m_bkg = (TH1D*) h_m_sig->Clone("h_m_bkg");

    //loop over the entries
    for(int iEntry; iEntry<tot_entries; ++iEntry){
        //take an entry
        tree->GetEntry(iEntry);
        //fill the histograms
        if(isBkg) h_m_bkg->Fill(B_m);
        else h_m_sig->Fill(B_m);
    }

```

Take the i-th entry, which means that all variables linked to the branch address take the values of the i-th candidate in the tree

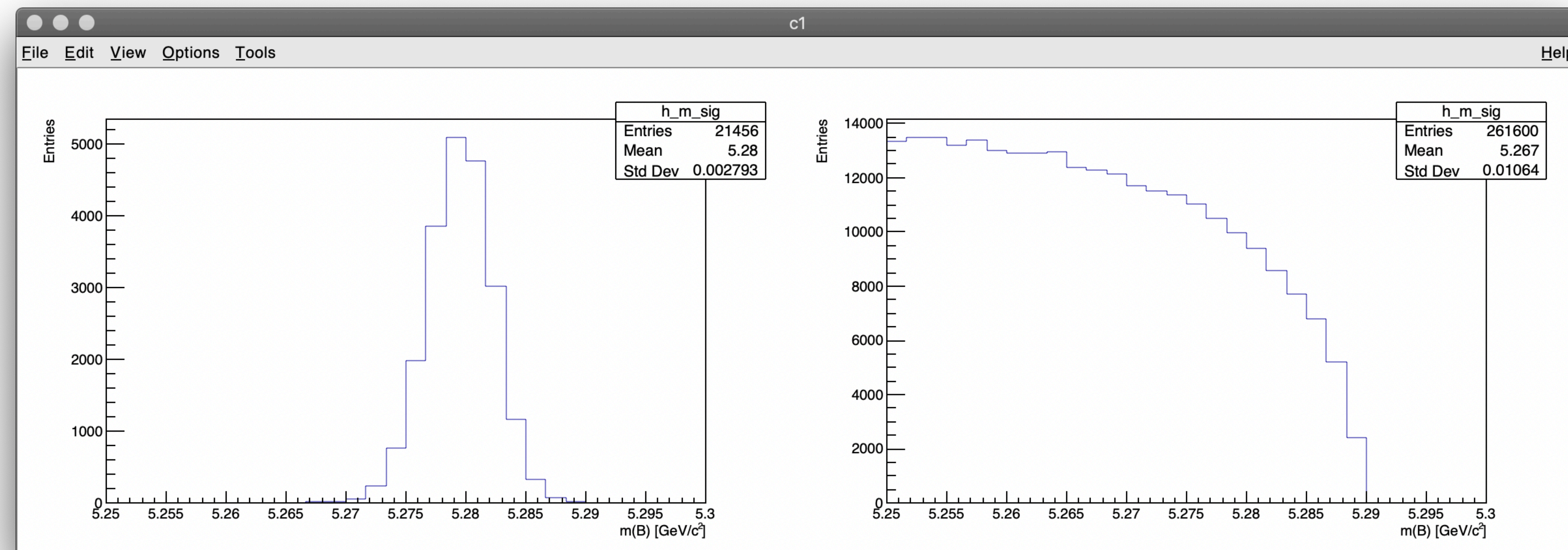
Reading a tree (readTree.C)

```
    //draw the histograms
    TCanvas* c1 = new TCanvas("c1","c1",1200,400);
    c1->Divide(2,1);
    c1->cd(1);
    h_m_sig->Draw();
    c1->cd(2);
    h_m_bkg->Draw();

    //generate some outputs
    int bin_min = h_m_sig->FindBin(5.27);
    int bin_max = h_m_sig->FindBin(5.29);
    double nSig = h_m_sig->Integral(bin_min, bin_max);
    double nBkg = h_m_bkg->Integral(bin_min, bin_max);

    cout << "Number of signal candidates: " << nSig << endl;
    cout << "Number of backgr candidates: " << nBkg << endl;
    cout << " S/N = " << nSig/(nSig+nBkg) << endl;
    cout << " S/sqrt(N) = " << nSig/sqrt(nSig+nBkg) << endl;

    return;
}
```

Reading a tree (`readTree.C`)

- The output

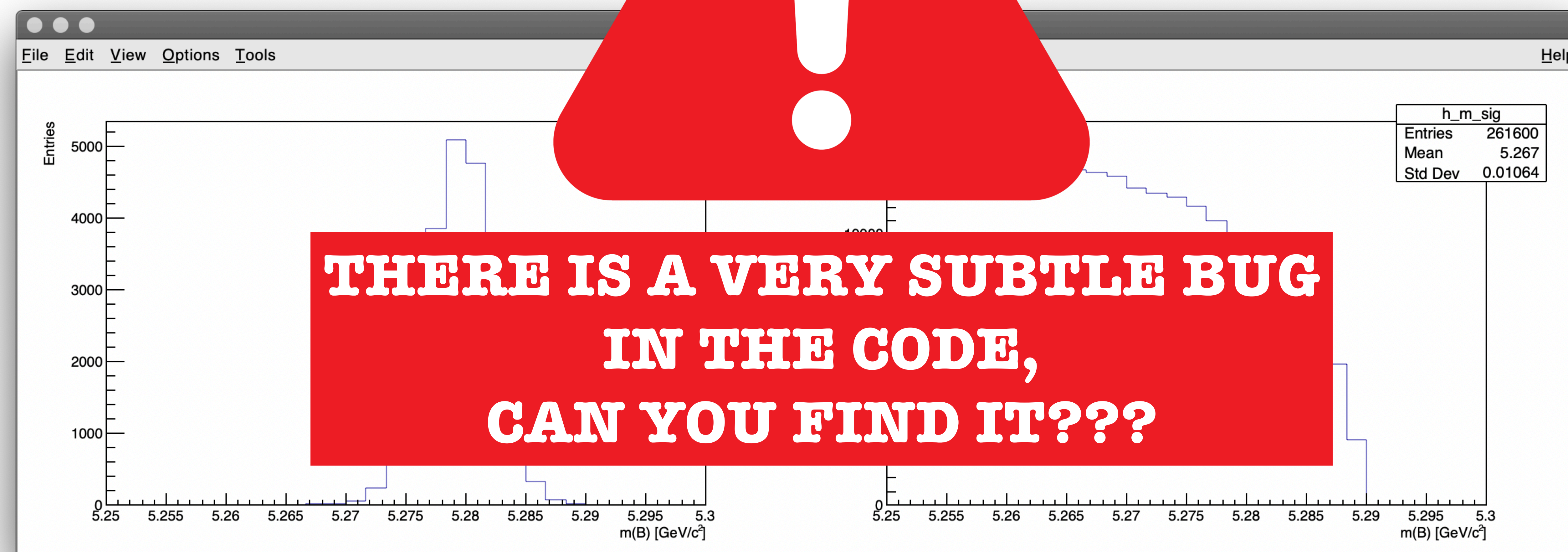
```
Processing readTree.C...
Total entries in the tree: 283056
Number of signal candidates: 21417
Number of backgr candidates: 118350
 S/N = 0.153234
 S/sqrt(N) = 57.287
root [1]
```



Reading a tree (`readTree.C`)

- The output

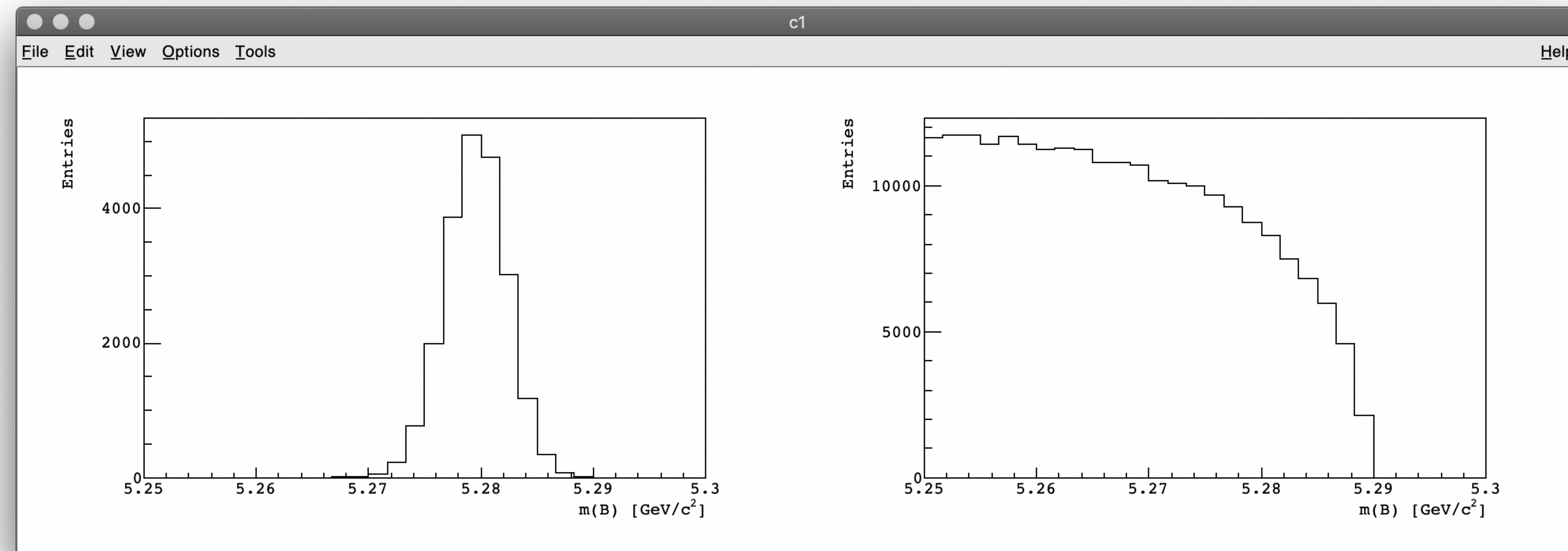
```
Processing readTree.C...
Total entries in the tree: 283056
Number of signal candidates: 21417
Number of backgr candidates: 118350
 S/N = 0.153234
 S/sqrt(N) = 57.287
root [1]
```



THERE IS A VERY SUBTLE BUG IN THE CODE, CAN YOU FIND IT???

Setting a default style

- We can put some default setting in a macro called `rootlong.C`
- No need to call the macro, it is loaded by default.



Optimise the selection

- Now we can work with the simulated data to optimise the cut on `bkg_killer`

- We maximise the signal significance, *i.e.* the function

$$\frac{S}{\sqrt{S+B}}$$

- We will count S and B in the mass distribution, just where the signal peak is ($M > 5.27\,\mathrm{GeV/c^2}$)

Optimise the selection (`optimiseSelection.C`)

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

using namespace std;

void optimiseSelection(){

    //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;

    //Two arrays to store the values of the cut
    double fom[ncuts];
    double cutval[ncuts];
```

We will make a graph of the FOM as a function of the cut, using the class TGraph

Optimise the selection (`optimiseSelection.C`)

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

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

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

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

        //put the cut in a string
        TString cutString = Form("bkg_killer > %.4f && B_m>5.27", cutval[icut]);

        //and retrieve the entries, directly from the tree, passing the selection
        double Nsig = tree->GetEntries(cutString+" && isBkg!=1");
        double Nbkg = tree->GetEntries(cutString+" && isBkg==1");

        //save the F.O.M.
        fom[icut] = Nsig/sqrt(Nsig+Nbkg);

        //just a check
        printf("cut value = %.3f, Nsig = %.0f, Nbkg = %.0f, FOM = %.3f\n",
               cutval[icut], Nsig, Nbkg, fom[icut]);

    }
```

Optimise the selection (`optimiseSelection.C`)

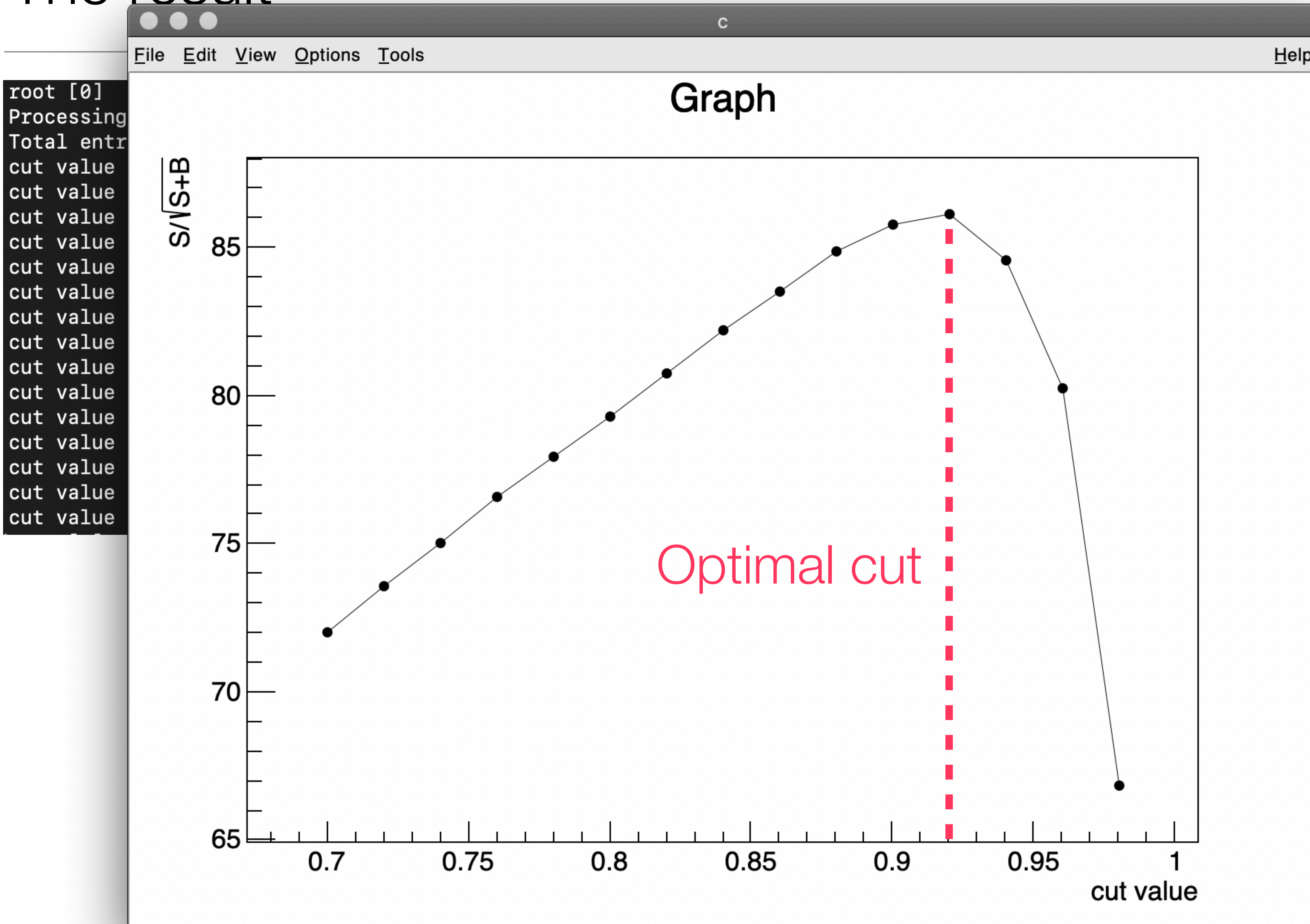
```
    //put all into a graph to siplay the FOM as a function ot the cut
    TGraph* g_fom = new TGraph(ncuts, cutval, fom);

    //and draw the graph
    TCanvas* c = new TCanvas("c","c",800,600);
    g_fom->SetMarkerStyle(8);
    g_fom->SetMarkerSize(0.8);
    g_fom->GetXaxis()->SetTitle("cut value");
    g_fom->GetYaxis()->SetTitle("S/#sqrt{S+B}");
    g_fom->Draw("APL");//A = axis, P = points, L = line

    return;
}
```

x values (`cutval`) y values (`fom`)

The result

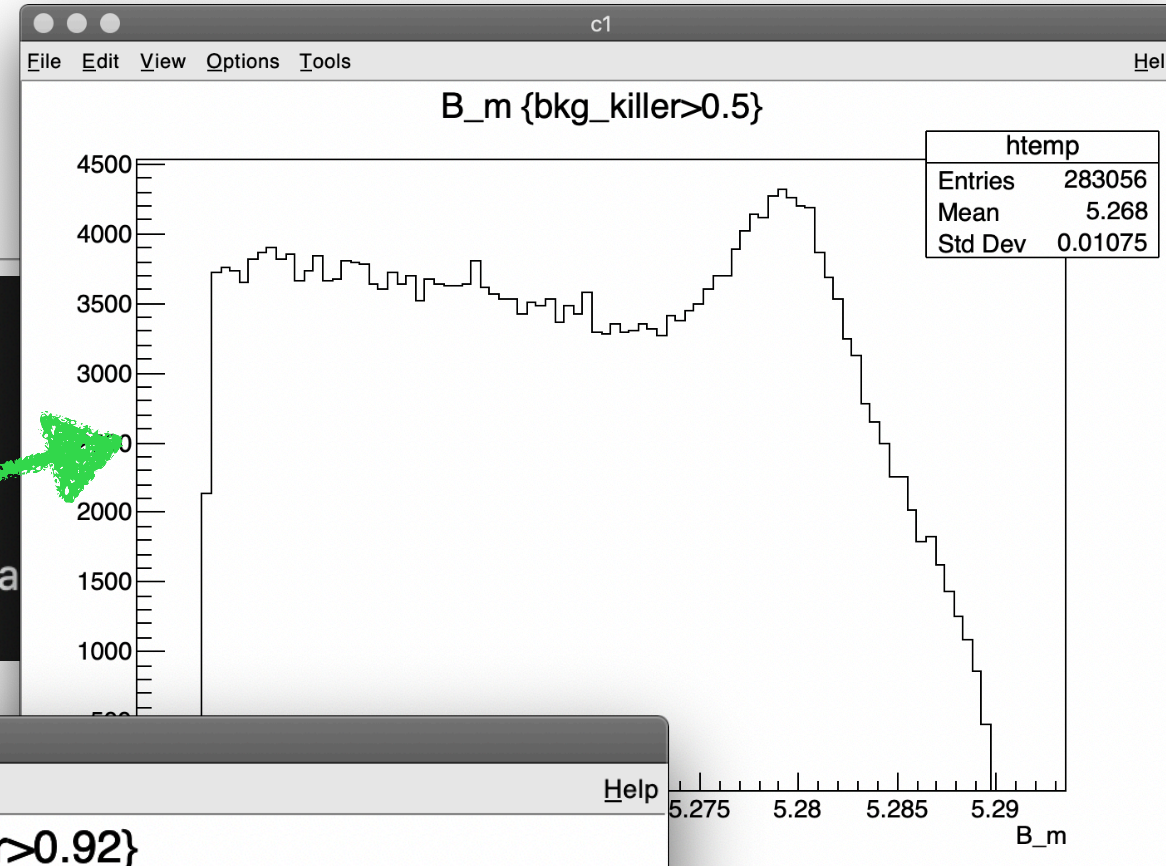
```
root [0]
Processing optimiseSelection.C...
Total entries in the tree: 283056
cut value = 0.700, Nsig = 18569, Nbkg = 47971, FOM = 71.986
cut value = 0.720, Nsig = 18236, Nbkg = 43221, FOM = 73.560
cut value = 0.740, Nsig = 17822, Nbkg = 38652, FOM = 74.995
cut value = 0.760, Nsig = 17372, Nbkg = 34122, FOM = 76.555
cut value = 0.780, Nsig = 16838, Nbkg = 29856, FOM = 77.922
cut value = 0.800, Nsig = 16256, Nbkg = 25758, FOM = 79.308
cut value = 0.820, Nsig = 15657, Nbkg = 21934, FOM = 80.754
cut value = 0.840, Nsig = 14951, Nbkg = 18134, FOM = 82.197
cut value = 0.860, Nsig = 14183, Nbkg = 14655, FOM = 83.519
cut value = 0.880, Nsig = 13364, Nbkg = 11449, FOM = 84.839
cut value = 0.900, Nsig = 12329, Nbkg = 8331, FOM = 85.775
cut value = 0.920, Nsig = 11125, Nbkg = 5570, FOM = 86.101
cut value = 0.940, Nsig = 9644, Nbkg = 3365, FOM = 84.554
cut value = 0.960, Nsig = 7658, Nbkg = 1453, FOM = 80.229
cut value = 0.980, Nsig = 4755, Nbkg = 305, FOM = 66.846
```

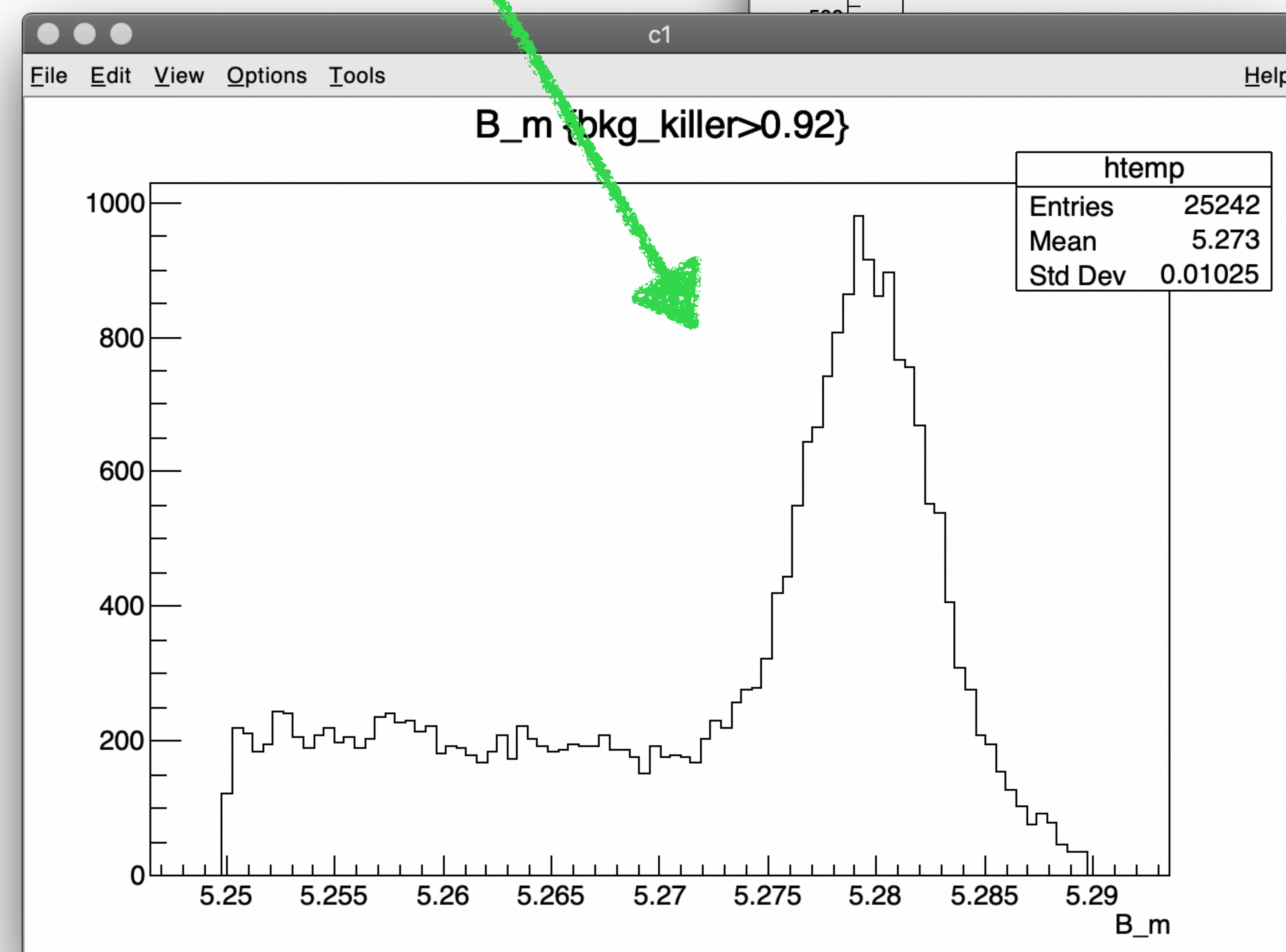
The result


root [0]
Processing
Total entr
cut value
File Edit View Options Tools Help
Graph
S/√S+B
85
80
75
70
65
0.7
0.75
0.8
0.85
0.9
0.95
1
cut value
Optimal cut

Let's see on simulated data

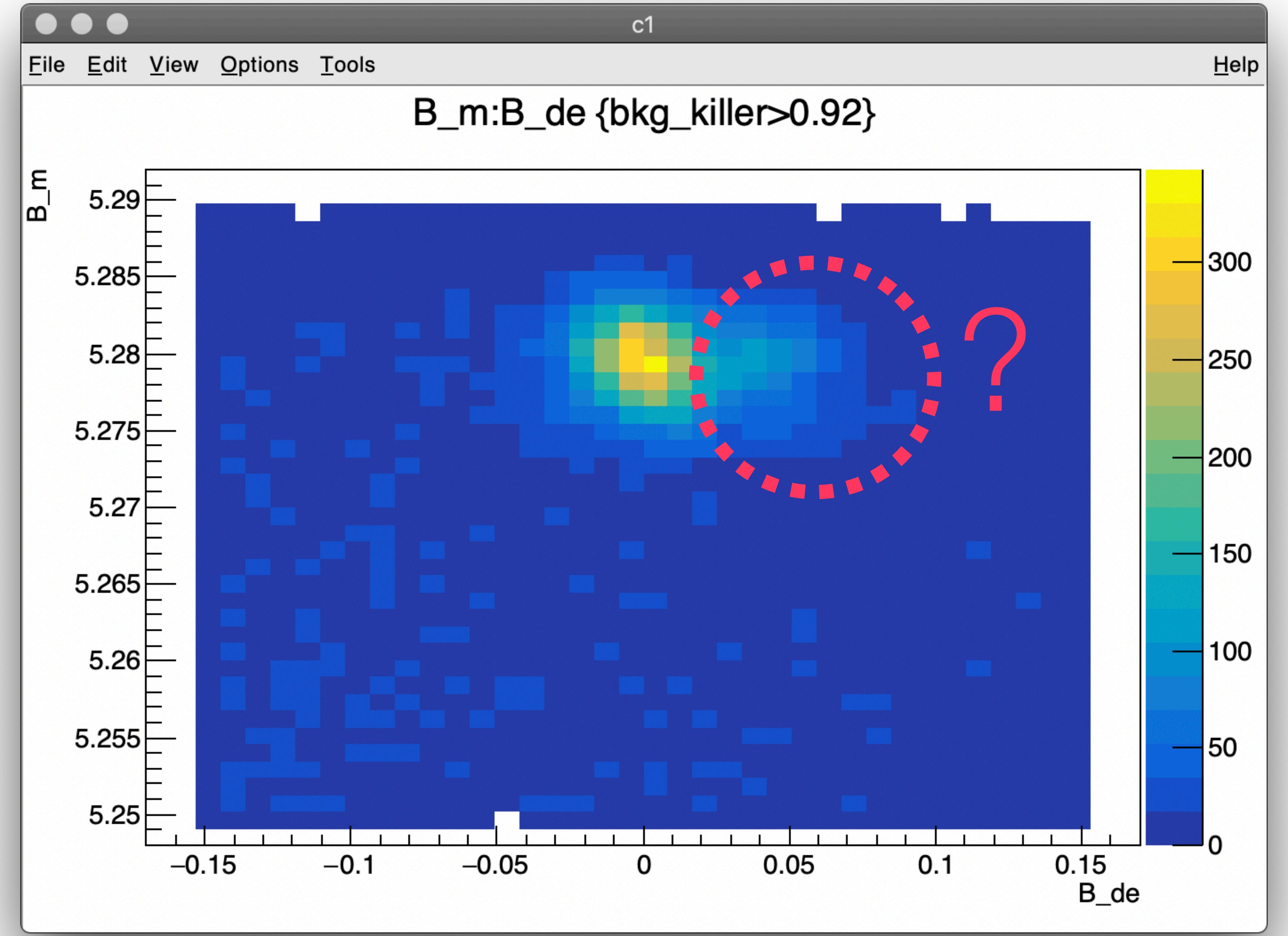
```
mb-md-01:thirdLesson dorigo$ rootl simulation.root
root [0]
Attaching file simulation.root as _file0...
(TFile *) 0x7f981b8de6f0
root [1] simTree->Draw("B_m","bkg_killer>0.5");
Info in <TCanvas::MakeDefCanvas>:  created default TCa
root [2] simTree->Draw("B_m","bkg_killer>0.92");
```





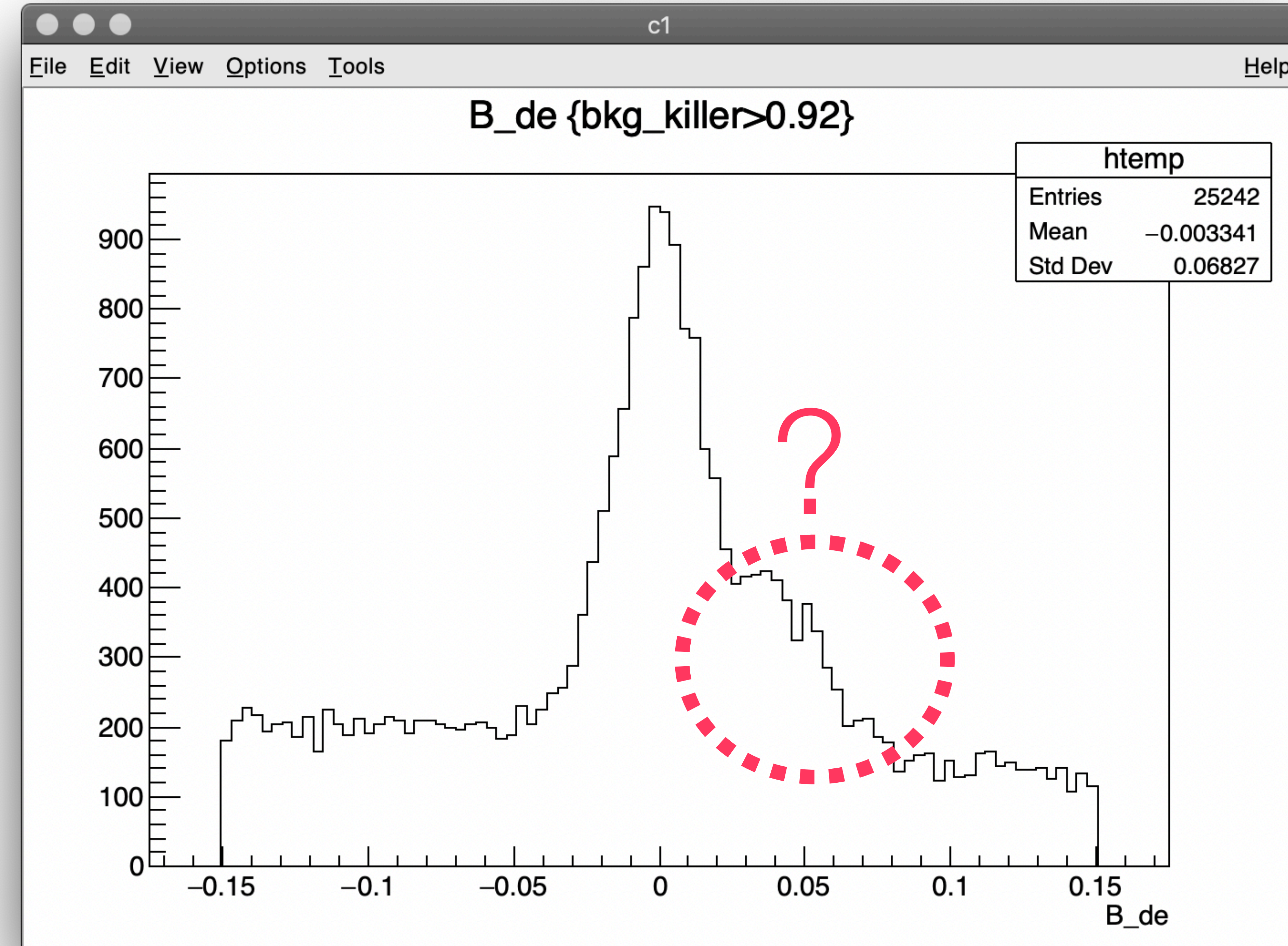
Let’s see on simulated data

```
root [3] simTree->Draw("B_m:B_de","bkg_killer>0.92","colz");
```



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*** $\Upsilon(4S) \rightarrow B\overline{B}$.

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

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

- Let's check in simulation. We have a variable that flag real $B^0 \rightarrow K^+\pi^-$ 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);

    }
```

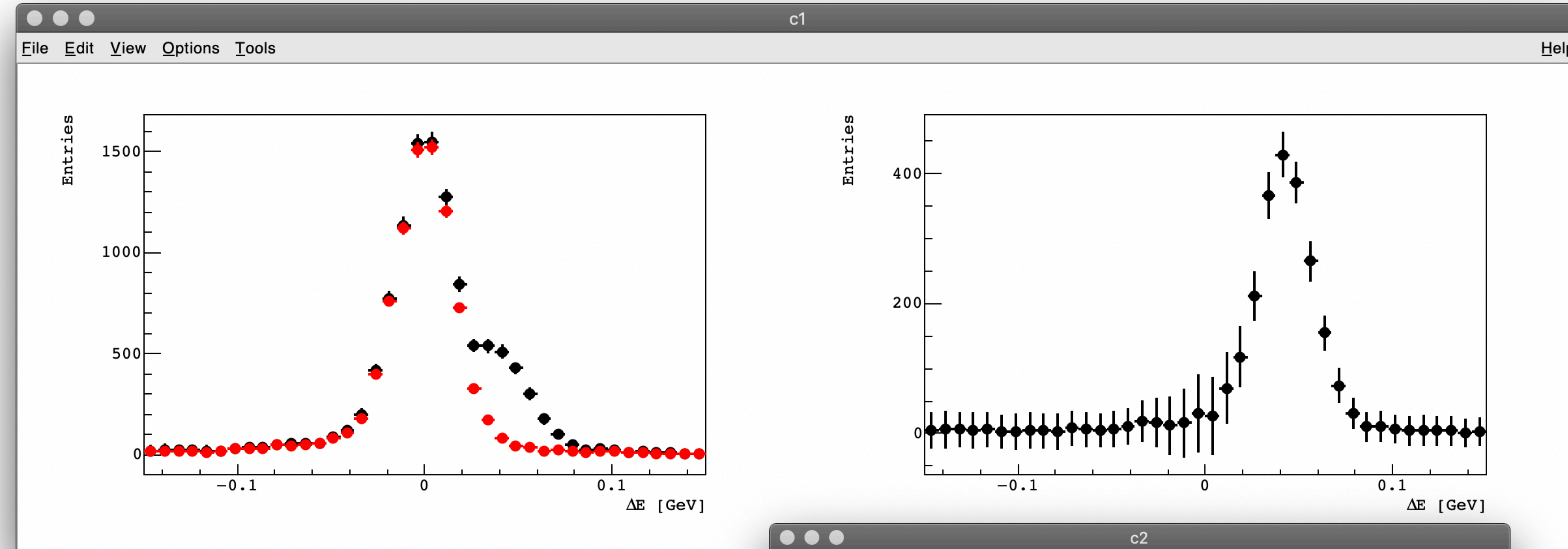
IMPORTANT!!!

Inspect B decays (`inspectB.C`)

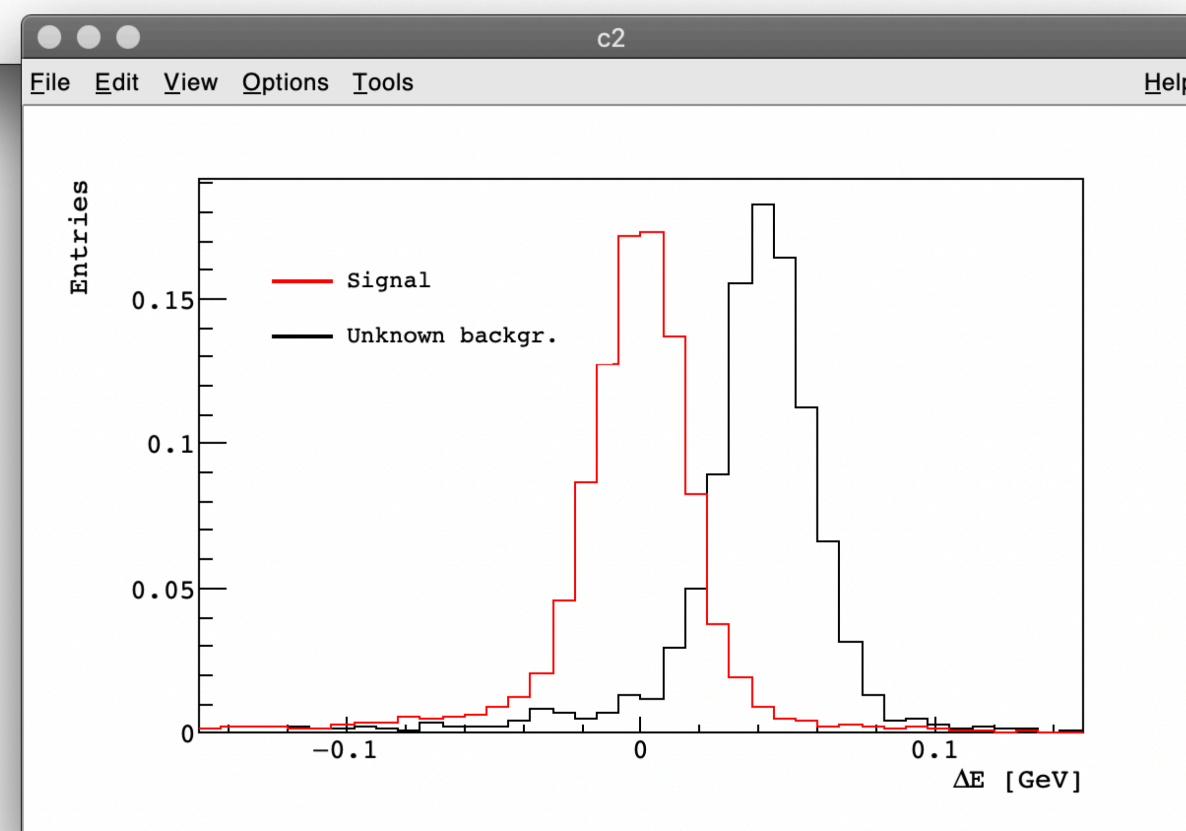
```
    //subtract the background from the total
    h_de_tot->Add(h_de_bkg,-1);

    //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

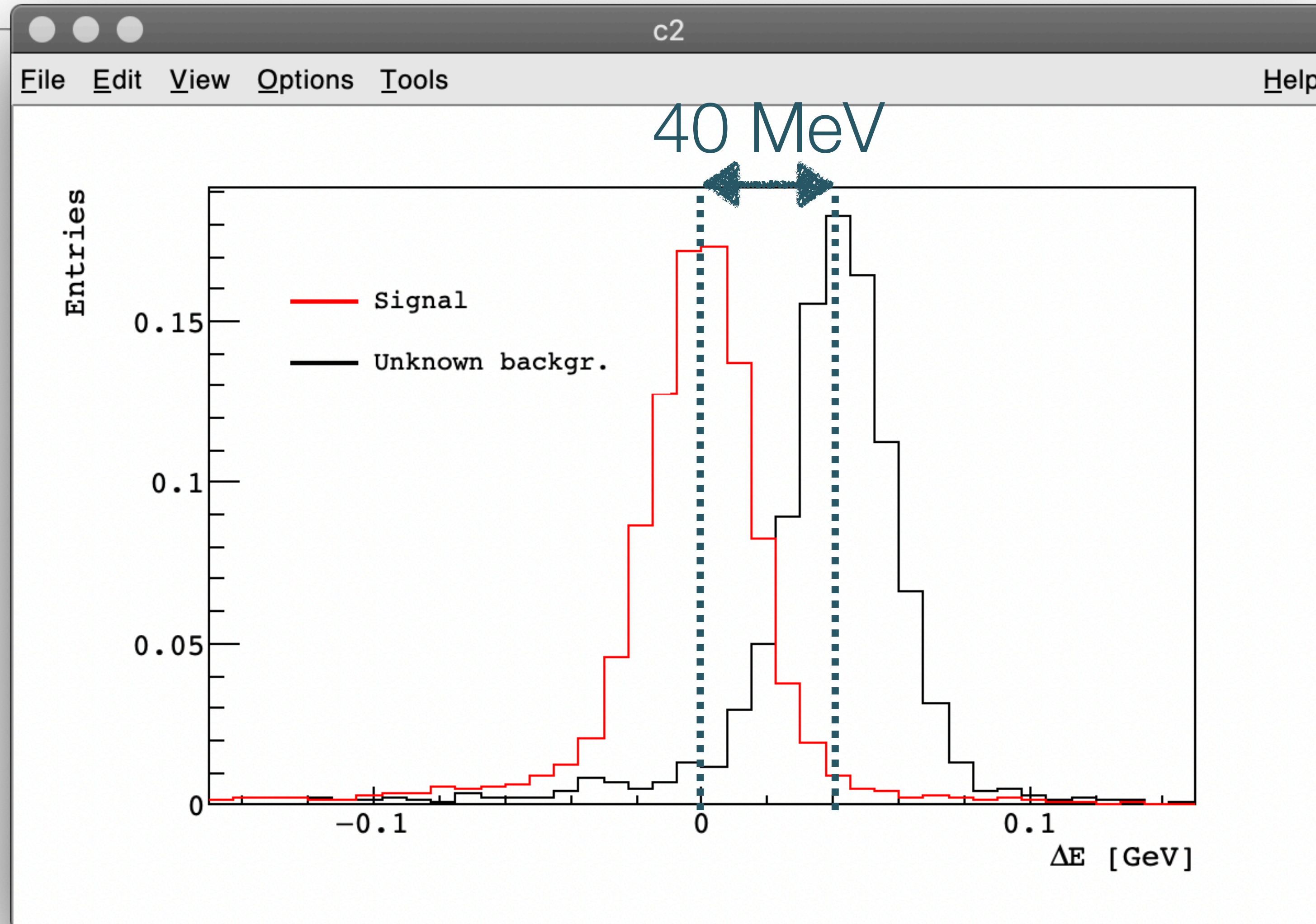
The output




```
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 masses, you will find about $+40\,\mathrm{MeV}$
- We can use a variable, built from PID detectors, to suppress this background.

Exercises (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.

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 after), and normal `Draw()` method.

Exercises (2)

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…