Exercise/Hands-on #4

Interpolation of $\psi(2S) \rightarrow \mu^+\mu^-$ with an extended binned ML fit (with RooFit)

Statistical Data Analysis for HEP

Prof. Alexis Pompili (University of Bari Aldo Moro)*

Erasmus+ Teaching Mobility Program / 16-20 October 2023 @ Sofia Physics Faculty

* alexis.pompili@ba.infn.it (or alexis.pompili@cern.ch)

Part-1 / First fit : Gaussian for the signal + Chebyshev polynomial for the background

First fitting approach to the reconstructed signal $\psi(2S) \rightarrow \mu^+\mu^-$ in CMS

We start with the small input ROOT file: **psiprime_bin9_histo.root**

Later we will use the larger ROOT file **hlt_5_newSoftMuon_alsoInPsiPrimeWind.root**

First version of the code to run: **psiPrime_fit.C**
It implements two subsequent fits with two different models.
Let's now focus on the first one.
It uses : - a Gaussian function for the signal
- a Chebyshev polynomial for the background

Let's introduce the Chebyshev polynomials in the next slide.

When using in the fit model a standard polynomial parametrization (`RooPolynomial`, https://root.cern.ch/doc/master/classRooPolynomial.html)

RooPolynomial implements a polynomial p.d.f of the form.

$$f(x) = \mathcal{N} \cdot \sum_i a_i * x^i$$

By default, the coefficient a_0 is chosen to be 1, as polynomial probability density functions have one degree of freedom less than polynomial functions due to the normalisation condition. $\mathcal{N}$ is a normalisation constant that is automatically calculated when the polynomial is used in computations.

… very often it results in strong correlations between coefficients that introduce - **issues in the fit stability**
- inability to find the right solution at high order

This can be solved (i.e. mitigating fit instability) using better polynomial parametrization, such as Chebyshev or Bernstein polynomials.

Let's discuss the first ones now.

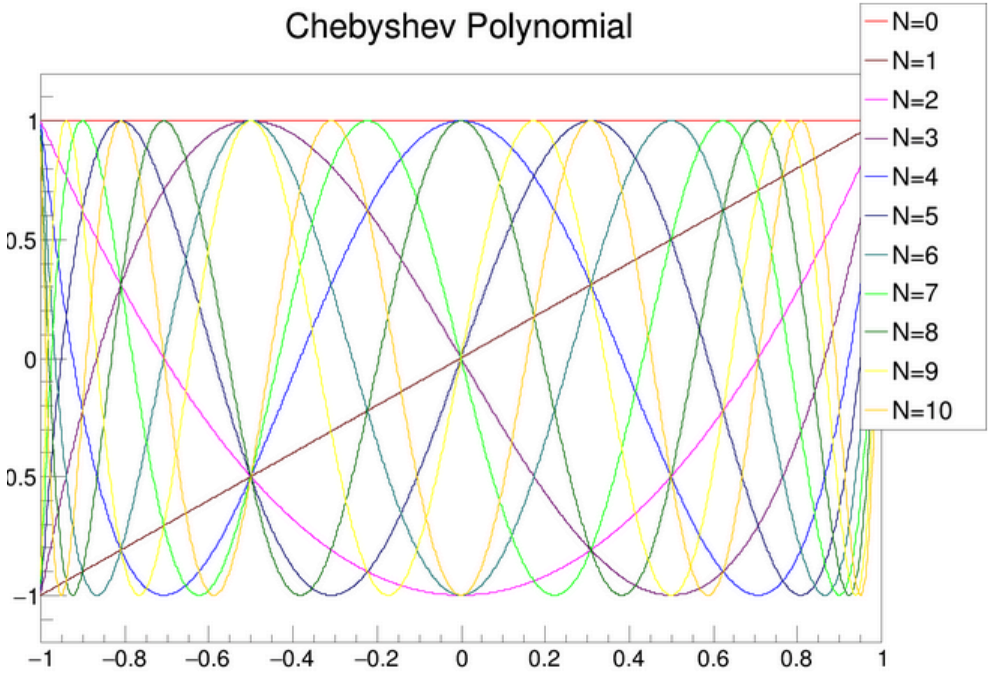
Chebyshev polynomials - II

When using the RooChebyshev class (https://root.cern.ch/doc/master/classRooChebychev.html) consider that ...

... **the number of parameters corresponds to the degree of the considered polynomial !**

The coefficient that goes with $T_0(x) = 1$ (i.e. the constant polynomial) is implicitly assumed to be 1, and the list of coefficients supplied by callers starts with the coefficient that goes with $T_1(x) = x$ (i.e. the linear term). (*)

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_2(x) = 2x^2 - 1$$
$$T_3(x) = 4x^3 - 3x$$
$$T_4(x) = 8x^4 - 8x^2 + 1$$
$$T_5(x) = 16x^5 - 20x^3 + 5x$$
$$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$$
$$T_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x$$
$$T_8(x) = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1$$
$$T_9(x) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$$



These polynomials are typically extensively exploited in numerical approximation tasks.

(*) If we have c_0 and c_1 the polynomium will have degree 2 and will be: $C_2(x) = 1 + c_0 x + c_1(2x^2 - 1) = (1 - c_1) + c_0 x + 2c_1 x^2$

For comparison the standard polynomial would be: $P_2(x) = p_0 + p_1 x + p_2 x^2$

Let’s examine the macro code - `psiPrime_fit.C` - used to produce the fitting tasks:

```
/////////////////////////////////////
// run with root: .x psiPrime_fit.C
/////////////////////////////////////
#include <vector>
using namespace RooFit; // to use RooFit package (in ROOT)
//
void psiPrime_fit() {
  gROOT->ForceStyle();
  gStyle->SetTitleOffset(1.4, "Y");
  gStyle->SetOptFit(1);
  //
  // -- select the input file:
  //
  //TFile* f1 = TFile::Open("./hlt_5_newSoftMuon_alsoInPsiPrimeWind.root","read"); // note: "read" mode
  //
  // smaller file extracted from the previous one:
  TFile* f1 = TFile::Open("./psiprime_bin9_histo.root","read");
  //
  // -- select and get the histogram to fit (bin-9):
  //
  TH1F* hPsiPrime;
  hPsiPrime = (TH1F*) f1->Get("PsiPrimeMass_bin9");
  //
  // -- create the "Canvas" (graphic space to temporarily store the output):
  //
  TCanvas *myC = new TCanvas("myC","PsiPrimeMassPlot", 700, 700);
  //
  // -- get histogram' features and define the random variable
  Double_t xMin = hPsiPrime->GetXaxis()->GetXmin();
  Double_t xMax = hPsiPrime->GetXaxis()->GetXmax();
  Int_t nBins = hPsiPrime->GetNbinsX();
  Float_t bin_width = hPsiPrime->GetBinWidth(1.);  // needed later
  //
  RooRealVar xVar("xVar", "m(#mu^{+}#mu^{-}) [GeV/c^{2}]", xMin, xMax);
  xVar.setBins(nBins);
  //
  // -- create the histogram as (a pointer to) an object that can be interpolated by RooFit:
  //
  RooDataHist* MuMuHist = new RooDataHist("#mu#mu_hist", hPsiPrime->GetTitle(), RooArgSet(xVar), Import(*hPsiPrime,kFALSE));
  //
  // -- note: this object, as defined, will undergo a BINNED Maximum Likelihood Fit (BML-fit)
  //
  /////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////////////////
//
// -- Create the fitting model
//
// -- signal model
RooRealVar mG("mean", "mean", 3.7, 3.67, 3.73);
RooRealVar sigma1("#sigma_{1}", "sigma1", 0.02, 0.001, 0.1);
RooGaussian sigPDF("sigPDF", "Signal", xVar, mG, sigma1);
//
// -- bkg model
RooRealVar c1("c_{1}", "c1", -0.1 ,-10, 10);
RooRealVar c2("c_{2}", "c2", -0.1 ,-10, 10);
RooChebychev bkgPDF("bkgPDF", "bkgPDF", xVar, RooArgSet(c1,c2));
//
RooRealVar nSig("nSig", "Number of signal candidates ", 2e+5, 1., 1e+6);
RooRealVar nBkg("nBkg", "Bkg component", 120e+3, 1., 1e+6);
//
RooAddPdf* totalPDF = new RooAddPdf("totalPDF", "totalPDF", RooArgList(sigPDF, bkgPDF), RooArgList(nSig, nBkg));
// the PDF defined in this way is implicitely configured to be an EXTENDED BML-fit
//
//////////////////////////////////////////////////////////////////////////////////////////
//
// -- Execute the Binned ML fit (HESSE is the default; add MINOS if you want)
//
//totalPDF->fitTo(*MuMuHist, Extended(kTRUE), Minos(kTRUE));
// implicitely:  totalPDF->fitTo(*MuMuHist, Extended(kTRUE), Minos(kFALSE));
totalPDF->fitTo(*MuMuHist, Extended(kTRUE), Minos(kFALSE));  // we write it explicitely
//
//////////////////////////////////////////////////////////////////////////////////////////
```

Gaussian resolution function

Chebyshev (2nd ord.)

fitting model

fit execution (with options)

```
// -- Prepare graphical representation (usual one plus the pulls)
//
RooPlot* xframe = xVar.frame();
xframe->SetTitle( hPsiPrime->GetTitle() );
//xframe->SetYTitle("Candidates / 5 MeV/c^{2}"); // 120 bins x 600MeV, but better to get the bin-width automatically (next line)
char newlabel[255];
sprintf(newlabel, "Candidates/(%.3f GeV)", bin_width);
xframe->SetYTitle(newlabel);
//xframe->SetTitleOffset(1.45,"Y");
//
MuMuHist->plotOn(xframe);
totalPDF->plotOn(xframe);
//
totalPDF->plotOn(xframe, Components(RooArgSet(sigPDF)), LineColor(kRed));
totalPDF->plotOn(xframe, Components(RooArgSet(bkgPDF)), LineColor(kGreen), LineStyle(kDashed) );
//
//totalPDF->plotOn(xframe); // non needed unless I add pulls; in the latter case show why i need to decomment the line
//
totalPDF->paramOn(xframe, Parameters(RooArgSet(mG,sigma1,nSig)), Layout(0.55,0.9,0.9)); // box with parameters' best estimates
//
//xframe->getAttText()->SetTextSize(0.03); // not-needed
//
/////////////////////////-- goodness-of-fit with bin-by-bin pulls --////////
//
RooPlot* framePull = xVar.frame(); // frame for the pulls
framePull->SetTitle("Pulls bin-by-bin");
framePull->addObject( (TObject*)xframe->pullHist(), "p" ); // RooPlot has a list of objects (TObjects) that can be drawn
framePull->SetMinimum(-6);
framePull->SetMaximum(6);
// --
RooHist* hPulls = xframe->pullHist(); // useful later
// --
//
myC->Divide(0,2); // split the Canvas in 2 Pads
//
myC->cd(2); // go to bottom pad
gPad->SetPad(0.,0.,1.,0.3);
framePull->Draw();
//
TLine* line = new TLine(3.4,0.,4.,0.);
line->SetLineColor(2);
line->Draw("same");
TLine* lineUp = new TLine(3.4,3.,4.,3.);
lineUp->SetLineColor(4);
lineUp->SetLineStyle(kDashed);
lineUp->Draw("same");
TLine* lineDown = new TLine(3.4,-3.,4.,-3.);
lineDown->SetLineColor(4);
lineDown->SetLineStyle(kDashed);
lineDown->Draw("same");
//
myC->cd(1); // go to top pad
gPad->SetPad(0.,0.3,1.,1.);
xframe->Draw();
//
myC->SaveAs("./Plots/PsiPrimeMassFit_gauss_poly2ord.png");
// myC->Update();
myC->Clear();
//
TCanvas *myCP = new TCanvas("myCP","Plot of bin-by-bin pulls", 700, 700);
myCP->cd();
hPulls->Draw("ALP");
myCP->SaveAs("./Plots/Pulls.png");
gSystem->Sleep(15000);
myCP->Clear();
delete myCP;
//
```

setting up the RooPlot object
(with data and fit model -
 - total model and components)

bin-by-bin pulls are detailed in next slides

this is just to show that … **the pull histogram is a `RooHist` object**
(it can be drawn as usual …)

… the 2nd part of the macro (with a different fit model) follows…..

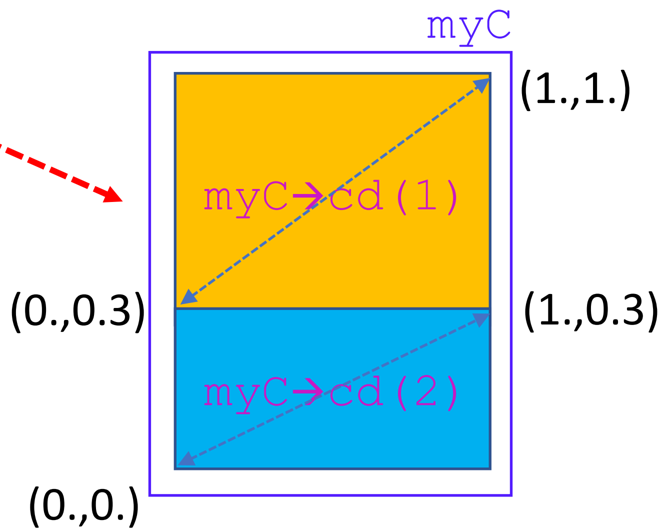
Goodness-of-fit : bin-by-bin pulls

To better investigate the quality of the fit we can add this simple piece of code that allows to introduce the **bin-by-bin pulls**:

```
///////////////// goodness of fits with pulls for each bin :
//
RooPlot *framePull = xVar.frame();
framePull->SetTitle("Pulls bin-by-bin");
framePull->addObject( (TObject*)xframe->pullHist(), "p" ) ;
framePull->SetMinimum(-6);
framePull->SetMaximum(6);
//
myC->Divide(0,2);
myC->cd(2);
gPad->SetPad(0.,0.,1.,0.3);
framePull->Draw();
TLine *line = new TLine(3.4,0.,4.,0.);
line->SetLineColor(2);
line->Draw("same");
myC->cd(1);
gPad->SetPad(0.,0.3,1.,1.);
xframe->Draw();
//
```

Appending the pull histogram in the ***list of objects*** of the `RooPlot` frame

divided in 2 `pads`



The bin-by-bin pulls are characterized by the following properties:

1) the uncertainty on each pull is unitary (this is shown in the next slide) [check also on the plots]

2) the projection on the y coordinate of the pulls should provide a distribution very close to a standard Gaussian $(\mu = 0, \sigma = 1)$

The ***normalized residual*** (deviation divided by its uncertainty) is similar to the square root of a chi-square ($\sqrt{\chi^2}$) supplied with its sign, reason for which it is offten called *pseudo chi-square*. We can denote it as $\pm\sqrt{\chi^2}$.

Of course, the histogram of the normalized residuals must have the same # of bins of the fitted histogram.

$$\pm\sqrt{\chi^2_{(i)}} = \frac{x^i_{Exp} - x^i_{Th}}{\sigma^i_{Exp}} \equiv \frac{N_i - F_i}{\sigma_i} = \frac{N_i - F_i}{\sqrt{N_i}}$$

where

- $N_i =$ experimental value
- $F_i =$ expected value (from fit model)
- $\sigma_i =$ uncertainty associated to the experimental value

Note: we assume negligible the uncertainty on the expected value (which is typically reasonable, i.e.: $\sigma^i_{Exp} \gg \sigma^i_{Th}$) so that :

$$\left(\sigma^i_{Tot}\right)^2 = \left(\sigma^i_{Exp}\right)^2 + \left(\sigma^i_{Th}\right)^2 \cong \left(\sigma^i_{Exp}\right)^2$$

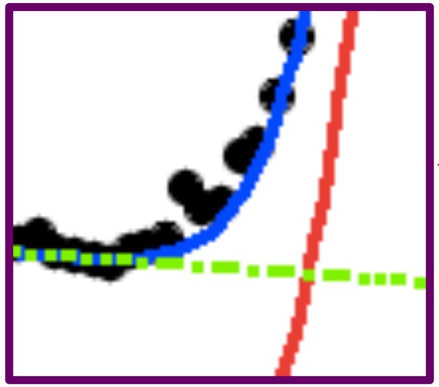
The uncertainty on the normalized residuals (for each bin) is calculated by applying the usual variance propagation law:

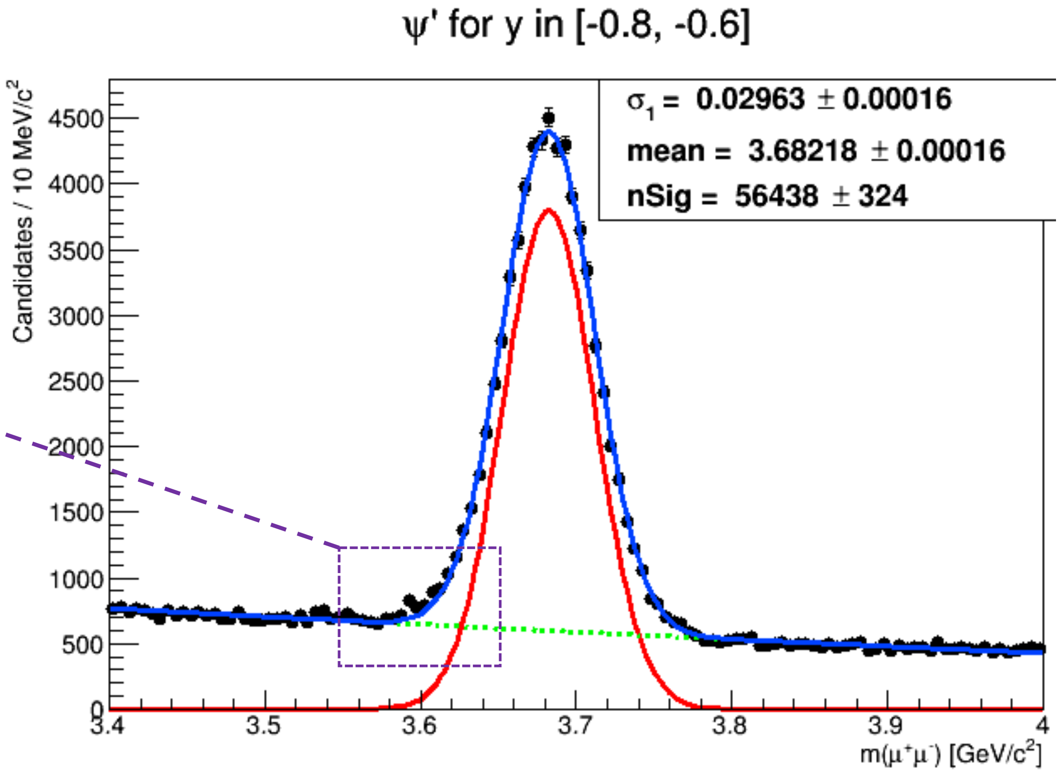
$$\sigma^2_{\pm\sqrt{\chi^2}} = \left(\frac{d}{dN}\left(\frac{N-F}{\sqrt{N}}\right)\right)^2 \cdot \left(\sqrt{N}\right)^2 = \left(\frac{\sqrt{N} - \frac{(N-F)}{2\sqrt{N}}}{N}\right)^2 \cdot N = \left(\frac{N - \frac{(N-F)}{2}}{N\sqrt{N}}\right)^2 \cdot N = \left(\frac{2N - N + F}{2N}\right)^2 = \left(\frac{1}{2}\frac{N+F}{N}\right)^2$$

Thus: $\sigma_{\pm\sqrt{\chi^2}} = \frac{1}{2}\frac{N+F}{N}$. **At high statistics** $N \approx F \Rightarrow \sigma_{\pm\sqrt{\chi^2}} \approx 1$

Part-1 / Try a new better fit : Crystal-Ball vs Gaussian to model the tail

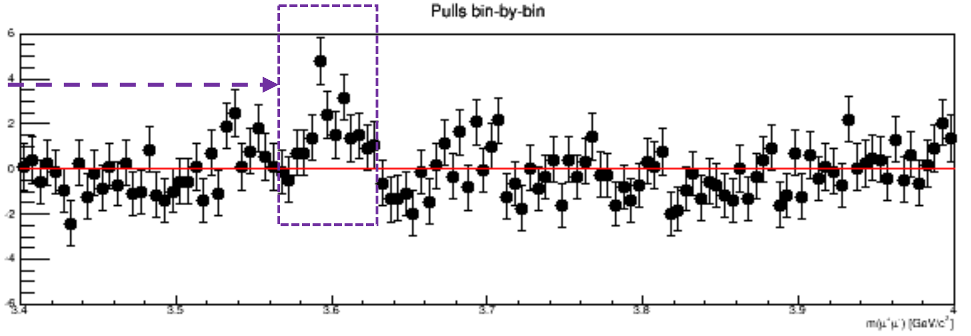
In the previous fit we have interpolated the distribution of $m(\mu^+\mu^-)$ in rapidity bin-9:





Note that the tail at low values of the invariant mass i.e. at the left shoulder of the signal peak is not well described by the gaussian.

This has been better appreciated by inspecting the bin-by-bin pulls:



As we will soon discuss we are going to substitute a plain Gaussian with a **Crystal Ball function** which integrates the Gaussian with a power function representing the tail (details at next slide)!
This tail is called **radiative tail** because it is due to a QED process (called **internal bremsstrahlung**) for which **a muon emits final state radiation: the energy carried away by the photon represents a type of radiative loss (hence the lower invariant mass values at the signal tail).**

Crystal ball function

The Crystal Ball function, named after the Crystal Ball Collaboration (hence the capitalized initial letters), is a probability density function commonly used to model various lossy processes in high-energy physics. It consists of a Gaussian core portion and a power-law low-end tail, below a certain threshold. The function itself and its first derivative are both continuous.

The Crystal Ball function is given by:

$$f(x;\alpha,n,\bar{x},\sigma) = N \cdot \begin{cases} \exp(-\frac{(x-\bar{x})^2}{2\sigma^2}), & \text{for } \frac{x-\bar{x}}{\sigma} > -\alpha \\ A \cdot (B - \frac{x-\bar{x}}{\sigma})^{-n}, & \text{for } \frac{x-\bar{x}}{\sigma} \leqslant -\alpha \end{cases}$$

Gaussian core (above a certain threshold α)

Power-law low-end tail (below a certain threshold α)

where

$$A = \left(\frac{n}{|\alpha|}\right)^n \cdot \exp\left(-\frac{|\alpha|^2}{2}\right)$$

$$B = \frac{n}{|\alpha|} - |\alpha|$$

2 more fit parameters

N is a normalization factor and α, n, x and σ are parameters which are fitted with the data.

To get an idea of the effect of the two parameters describing the tail let's discuss, in next slide, a pair of helpful figures [borrowed form an internal CMS analysis note (AN-14-003)] -->

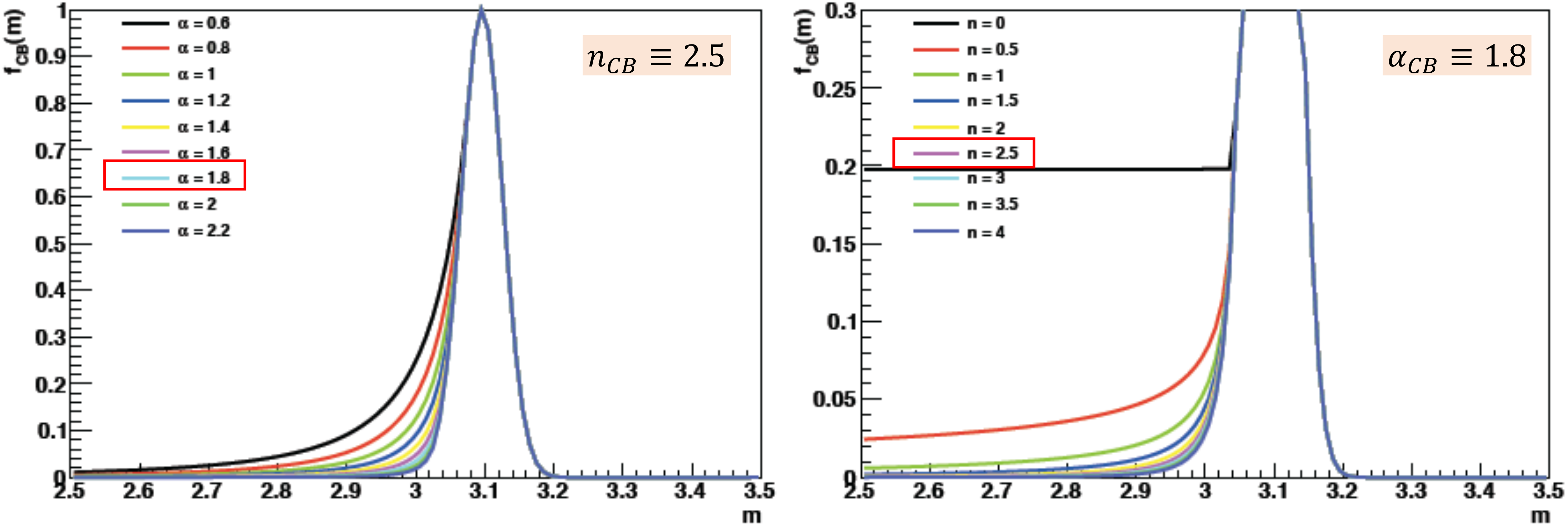


Figure 16: Shapes of the CB function for several different (n_{CB}, α_{CB}) values, fixing $n_{CB} = 2.5$ (left) or $\alpha_{CB} = 1.8$ (right).

(from CMS-AN-14-003)

New fit model

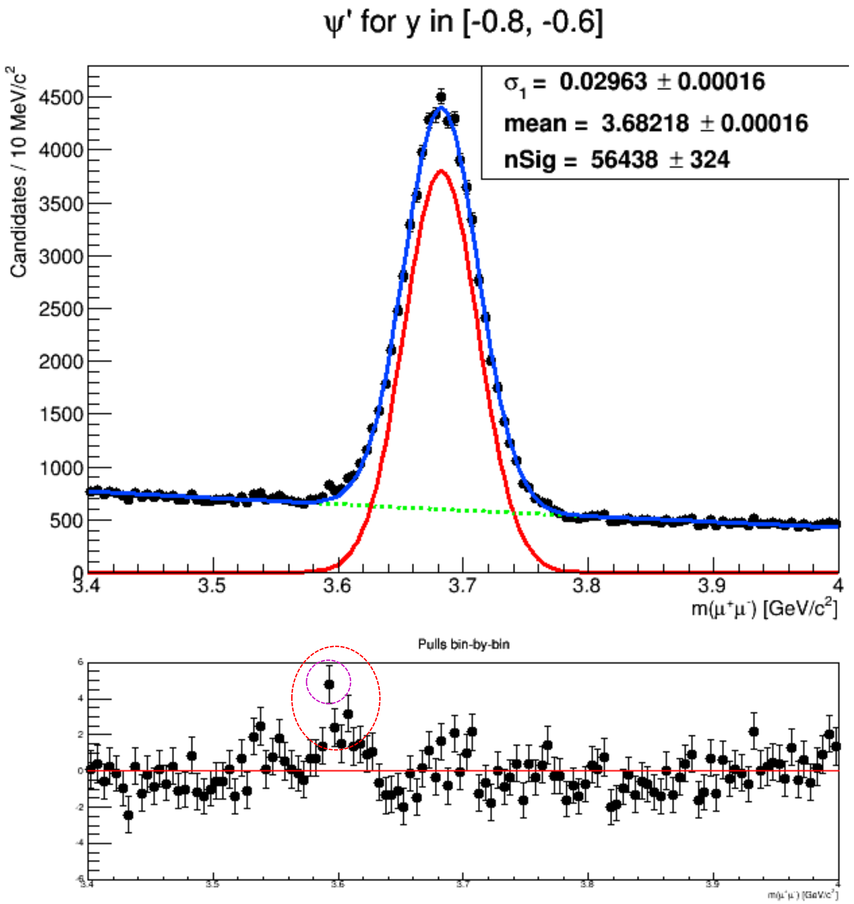
In the new fit model, we will:

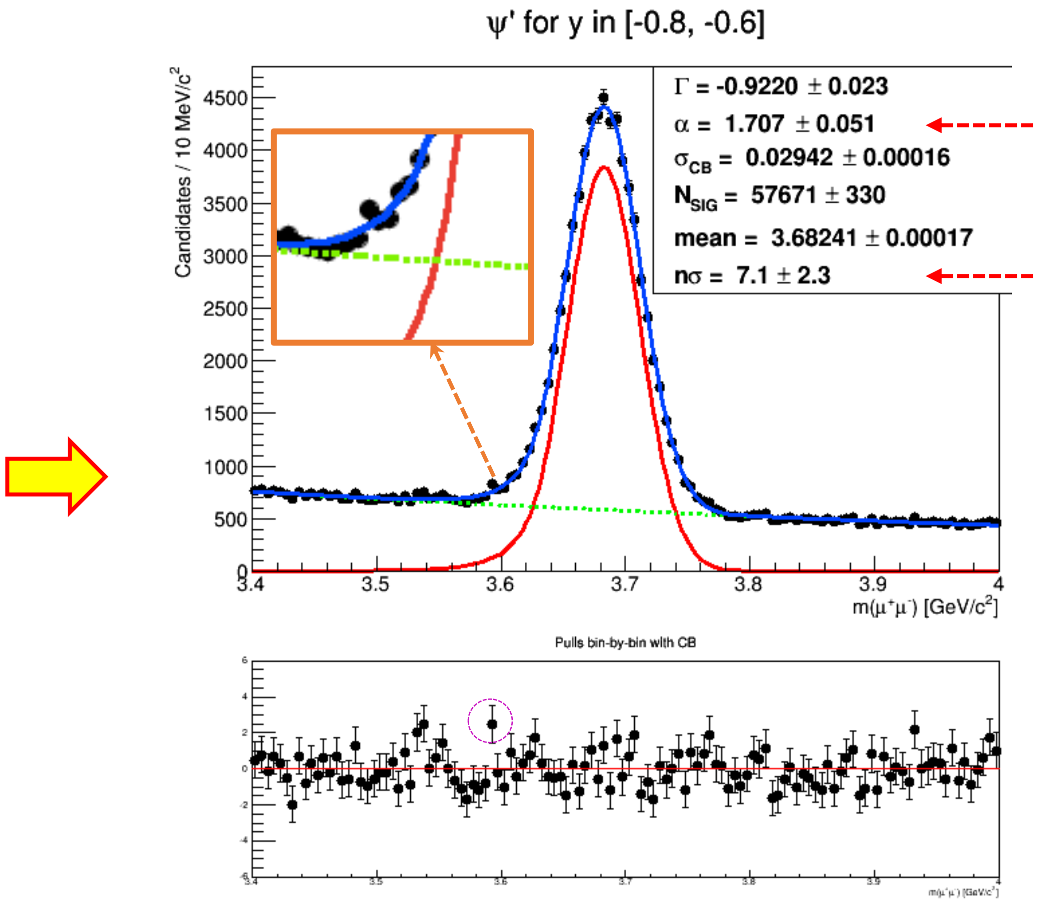
1) **substitute the Gaussian function with the Crystal Ball one**. In RooFit : `RooGaussian` ⇨ `RooCBShape`

2) change the bkg model using an exponential instead of the polynomial. In RooFit : `RooChebyshev` ⇨ `RooExponential`

Note: the two things are not directly related (you can try as exercise/homework to apply only (1)).

The bin-by-bin pulls' method allows to monitor the goodness-of-fit in the two cases.





```
//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////// 2nd part of the exercise: try CB+exp ///////////
//////////////////////////////////////////////////////////////////////////////////
RooAbsPdf *sigCBPdf ;
RooAbsPdf *bkgExpPdf ;
//
RooRealVar mGCB("mean_{CB}", "meanCB", 3.7, 3.67, 3.73);
RooRealVar sigma1CB("#sigma_{CB}", "sigma1CB", 0.02, 0.001, 0.1);
RooRealVar alpha("#alpha","alpha",2., 0.5, 10.);
RooRealVar nSigma("n#sigma","nSigma", 2., 0.1, 50.);
sigCBPdf = new RooCBShape("sigCBPdf","sigCBPdf",xVar,mGCB,sigma1CB,alpha,nSigma);
//
RooRealVar gamma("#Gamma","Gamma",-1e-1, -2., -1e-2) ;
bkgExpPdf = new RooExponential("bkgExpPdf","bkgExpPdf",xVar, gamma);
//
RooRealVar nBkgExp("N_{EXPBKG}","nBkgExp",120e+3,1.,1e+6);
RooRealVar nSigCB("N_{SIG}","nSigCB",2e+5,1.,1e+6);
//
RooAddPdf* totalCBExpPDF = new RooAddPdf("totalCBExpPDF","totalCBExpPDF", RooArgList(*sigCBPdf, *bkgExpPdf), RooArgList(nSigCB, nBkgExp));
//
totalCBExpPDF->fitTo(*MuMuHist, Extended(kTRUE), Minos(kFALSE));
//
RooPlot* xframe1 = xVar.frame();
xframe1->SetTitle( hPsiPrime->GetTitle() );
xframe1->SetYTitle("Candidates / 10 MeV/c^{2}");
MuMuHist->plotOn(xframe1);
totalCBExpPDF->plotOn(xframe1);
//
totalCBExpPDF->plotOn(xframe1, Components(RooArgSet(*sigCBPdf)), LineColor(kRed));
totalCBExpPDF->plotOn(xframe1, Components(RooArgSet(*bkgExpPdf)), LineColor(kGreen), LineStyle(kDashed) );
//
totalCBExpPDF->paramOn(xframe1, Parameters(RooArgSet(mGCB,sigma1CB,alpha,nSigma,nSigCB,gamma)), Layout(0.529,0.99,0.9));
//
myC->cd();
xframe1->Draw();
myC->SaveAs("./Plots/PsiPrimeMassFit_CB_Exp.png");
//
/////////////////////////////////////////////////////////////////
//
delete myC;
f1->Delete();
//
}
```

Classroom exercise: implement in this piece of code the goodness-of-fit information (bin-by-bin) pulls

Exercise : Discuss how the estimate of the number of ψ'candidates changes in the 3 fits with these different models:

- Gaussian + Chebyshev
- Crystal Ball + Chebyshev (this is not implemented in the current code) [but it's used in the 2nd part]
- Crystal Ball + Exponential

Homework : Put together the two subsamples in bin-13 and bin-20 (i.e. add the two corresponding histograms)

and try to fit the dimuon mass distribution of the obtained mixture (see theory of mixtures in the theroy part):

- try to use one Gaussian and check if its estimated $\boldsymbol{\sigma}$ is close to what expected

$$\boldsymbol{\sigma}_{\boldsymbol{eff}\,(\mathbf{13+20})} = \sqrt{\boldsymbol{\varphi}_{\mathbf{13}}\,\boldsymbol{\sigma}_{\mathbf{13}}^{\mathbf{2}} + \boldsymbol{\varphi}_{\mathbf{20}}\,\boldsymbol{\sigma}_{\mathbf{20}}^{\mathbf{2}}}$$

- try to use two Gaussians (with same mean) and see if their estimated $\boldsymbol{\sigma}$s are close to what expected ($\boldsymbol{\sigma}_{\mathbf{13}}$, $\boldsymbol{\sigma}_{\mathbf{20}}$)

(in this case help the fit assuming the fractions of signal ($\boldsymbol{\varphi}_{\mathbf{13}}, \boldsymbol{\varphi}_{\mathbf{20}}$) from the # of candidates in the single bins

Homework : show that the projection on the y coordinate of the pulls should provide a distribution very close to a standard Gaussian (try to use the `RooHist` object)

Part-2 / study of the rapidity dependence of the mass resolution

We have already well discussed that CMS dimuon mass resolution changes as a function of the rapidity:

$$\sigma_{m\,(\mu^+\mu^-)} = f(y) \qquad \text{where} \qquad y = \frac{1}{2}\ln\frac{E+p_z}{E-p_z} = \frac{1}{2}\ln\frac{1+\beta\cos\theta}{1-\beta\cos\theta}$$

We now want to study this dependence by performing the fit of all the rapidity bins.

With this aim we will use the larger ROOT file **hlt_5_newSoftMuon_alsoInPsiPrimeWind.root**

The code to run is now: **myFinalRapidity.C**

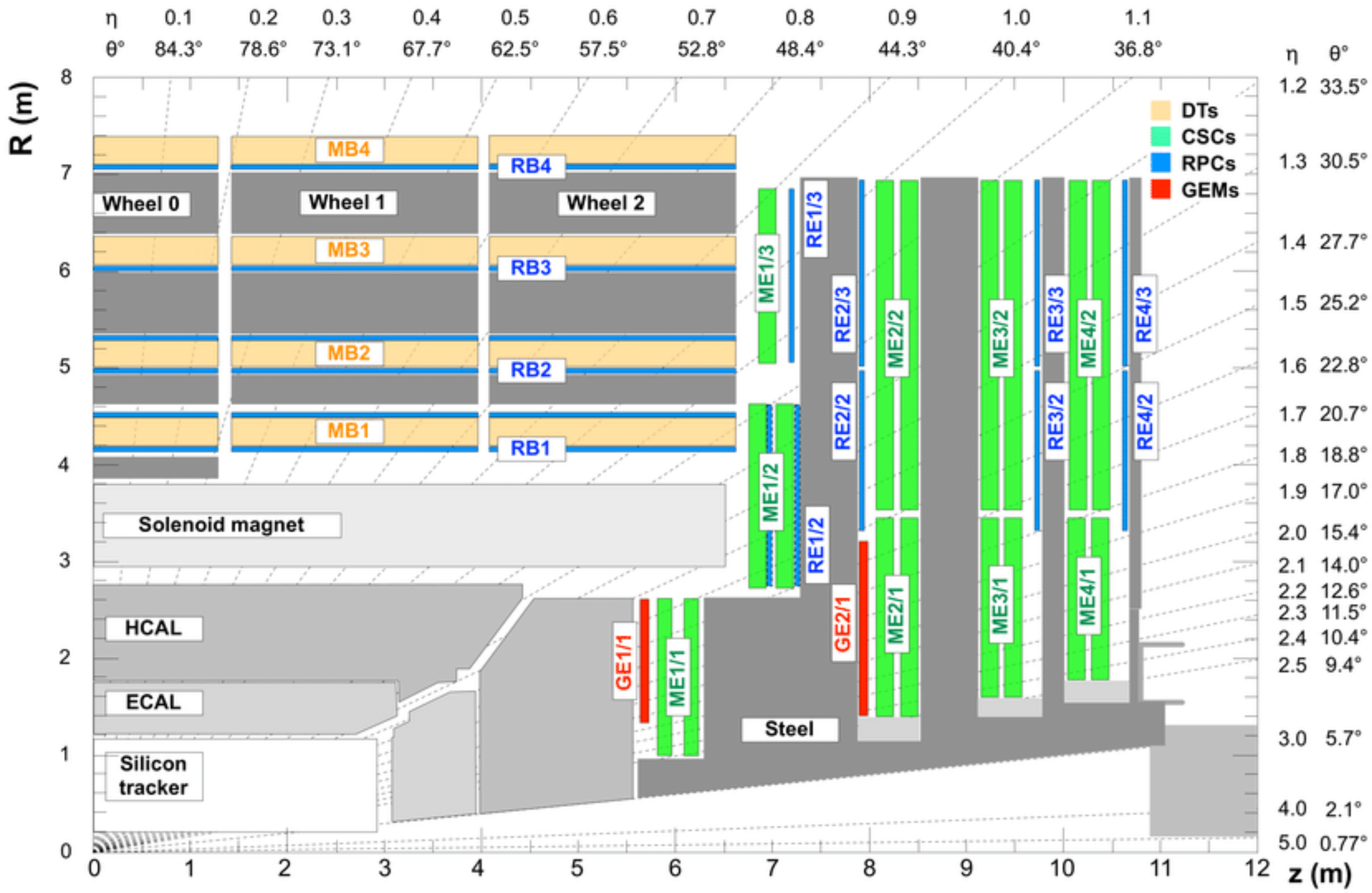
Considering that the background can vary a lot from bin to bin it is crucial to get successfull fits in all the bins and this requires to use some flexible model so we can try a Chebyshev polynomial of 2nd order for the background (as we did initially).

For the signal we use, for the reason already explained, a *one-sided* ("low-tail") Crystal Ball.

This is a quadrant of the CMS detector showing the η –regions in subdetector of the muon system (included the proposed GEM detectors for the upgrade) :

[note: pseudorapidity is the rapidity for relativistic particles: $y = \frac{1}{2}\ln\frac{1+\beta\cos\theta}{1-\beta\cos\theta} \rightarrow \frac{1}{2}\ln\frac{1+\cos\theta}{1-\cos\theta} = -\ln\tan\frac{\theta}{2} = \eta$ **]**

$\beta \rightarrow 1$



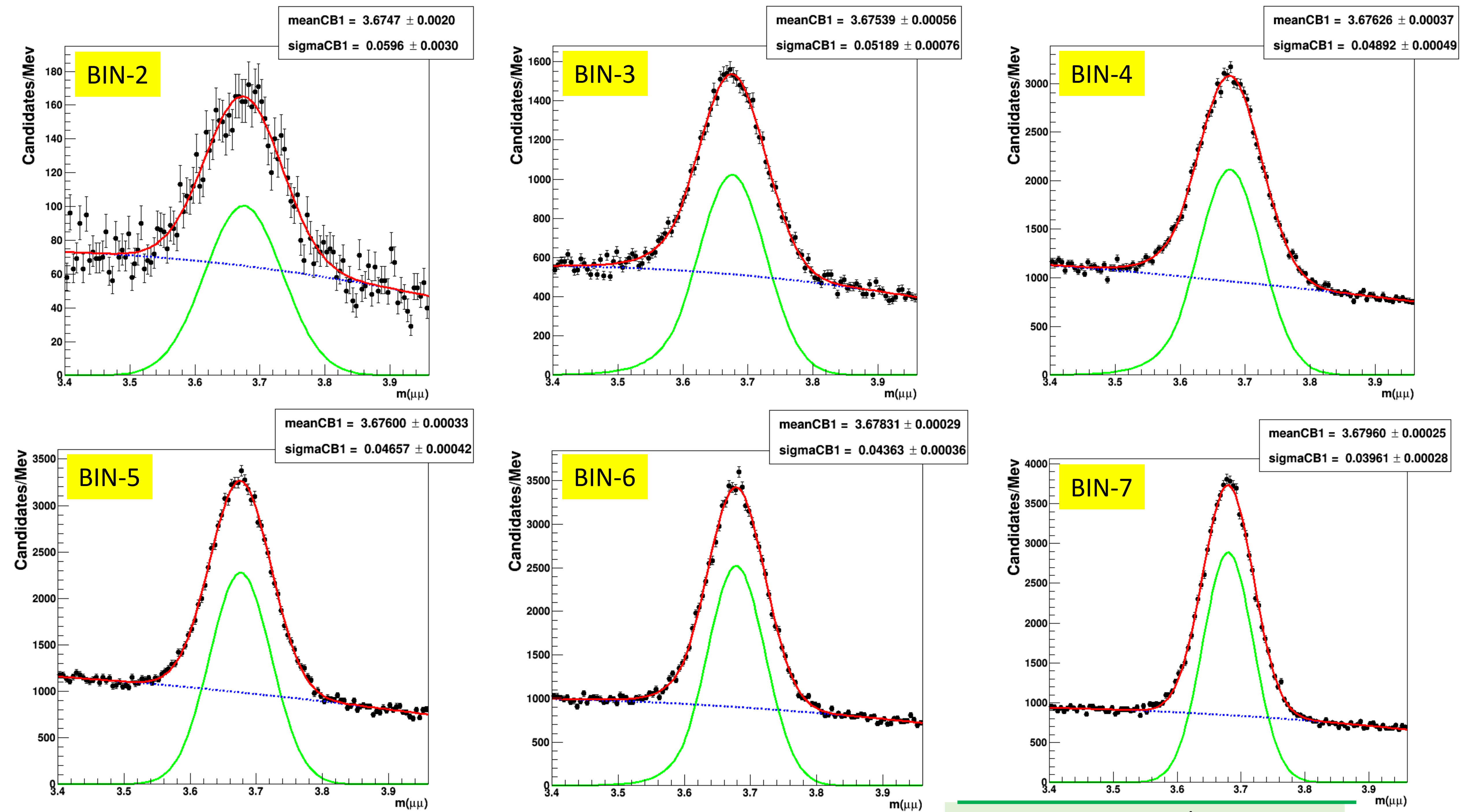
We will consider the following **rapidity bins**:

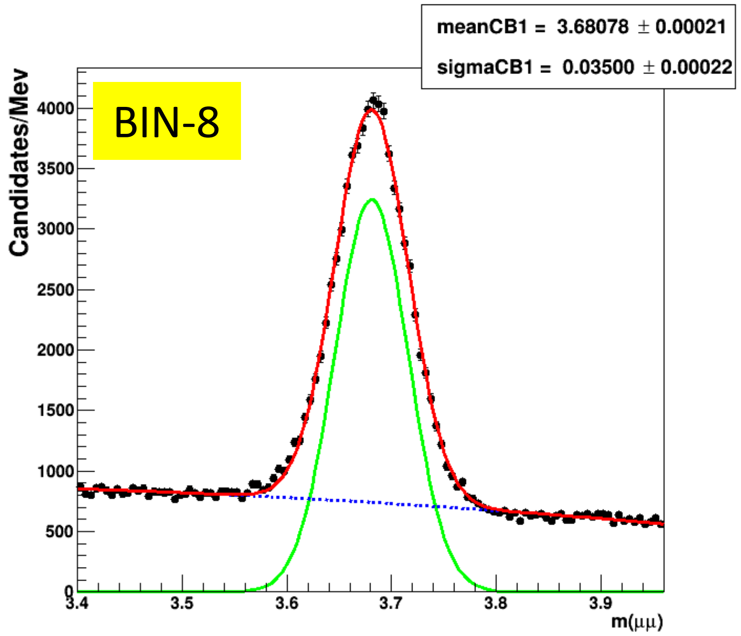
Bin	Rapidity
~~1~~	~~-2.4 / -2.2~~
2	-2.2 / -2.0
3	-2.0 / -1.8
4	-1.8 / -1.6
5	-1.6 / -1.4
6	-1.4 / -1.2
7	-1.2 / -1.0
8	-1.0 / -0.8
9	-0.8 / -0.6
10	-0.6 / -0.4
11	-0.4 / -0.2
12	-0.2 / 0.0

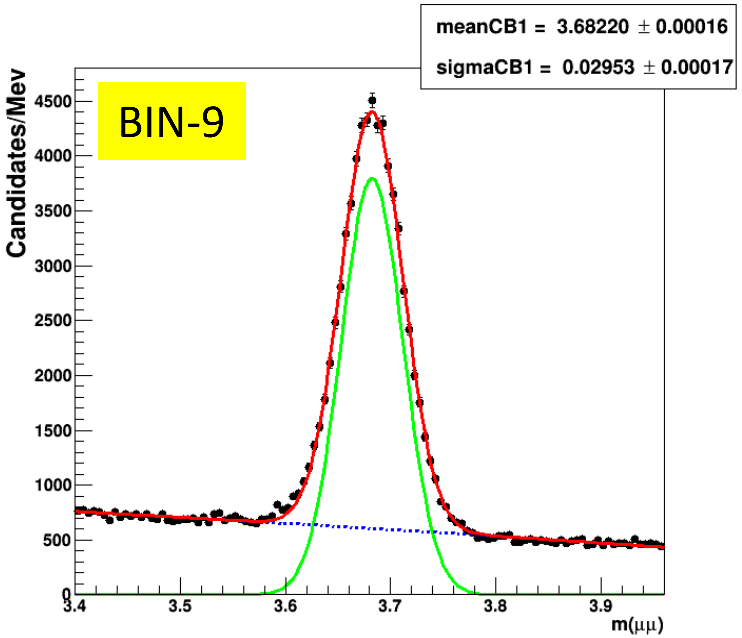
Bin	Rapidity
13	0.0 / +0.2
14	+0.2 / +0.4
15	+0.4 / +0.6
16	+0.6 / +0.8
17	+0.8 / +1.0
18	+1.0 / +1.2
19	+1.2 / +1.4
20	+1.4 / +1.6
21	+1.6 / +1.8
22	+1.8 / +2.0
23	+2.0 / +2.2
~~24~~	~~+2.2 / +2.4~~

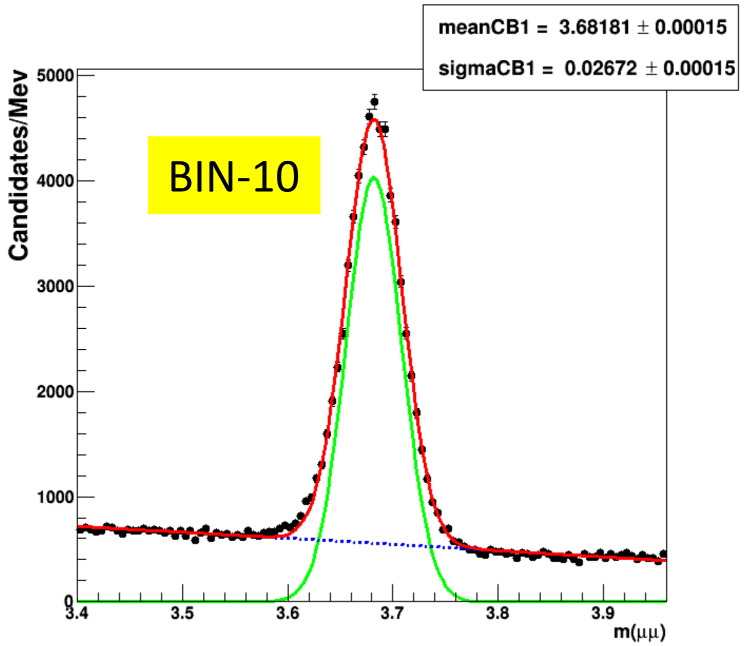
I will skip the two extreme bins due to low statistics (*overlined in red*)

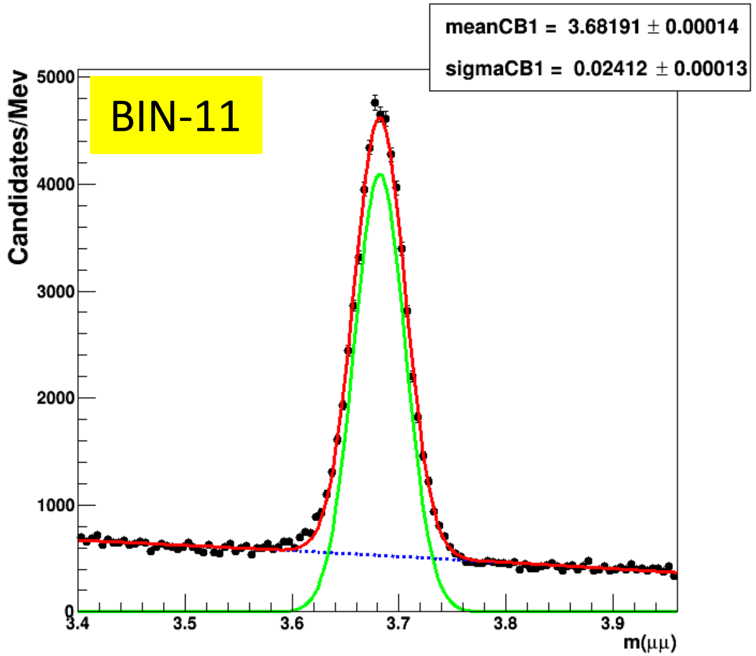
In the code we will loop over the *loop index* from 0 to 21
… and correspondingly bin-index will change from 2 to 23.

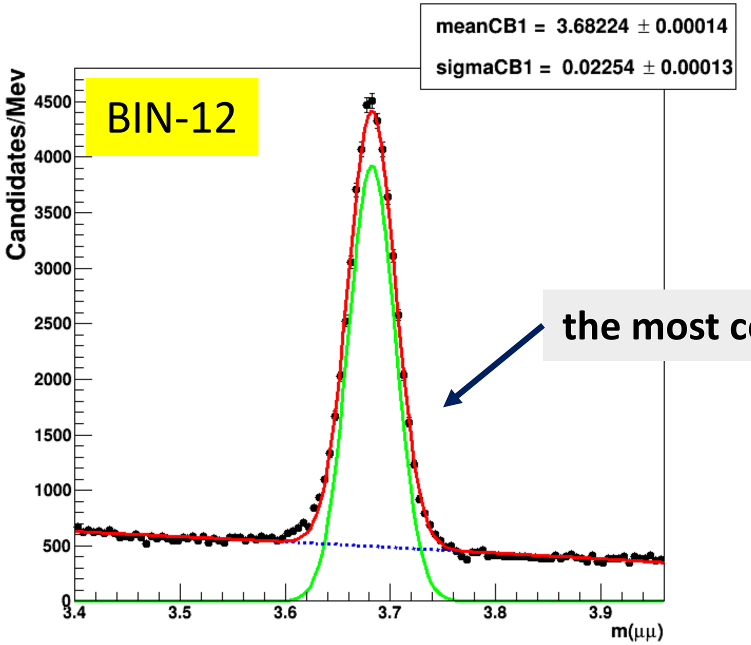

BIN-2
meanCB1 = 3.6747 ± 0.0020
sigmaCB1 = 0.0596 ± 0.0030
BIN-3
meanCB1 = 3.67539 ± 0.00056
sigmaCB1 = 0.05189 ± 0.00076
BIN-4
meanCB1 = 3.67626 ± 0.00037
sigmaCB1 = 0.04892 ± 0.00049
BIN-5
meanCB1 = 3.67600 ± 0.00033
sigmaCB1 = 0.04657 ± 0.00042
BIN-6
meanCB1 = 3.67831 ± 0.00029
sigmaCB1 = 0.04363 ± 0.00036
BIN-7
meanCB1 = 3.67960 ± 0.00025
sigmaCB1 = 0.03961 ± 0.00028
Candidates/Mev
m(μμ)



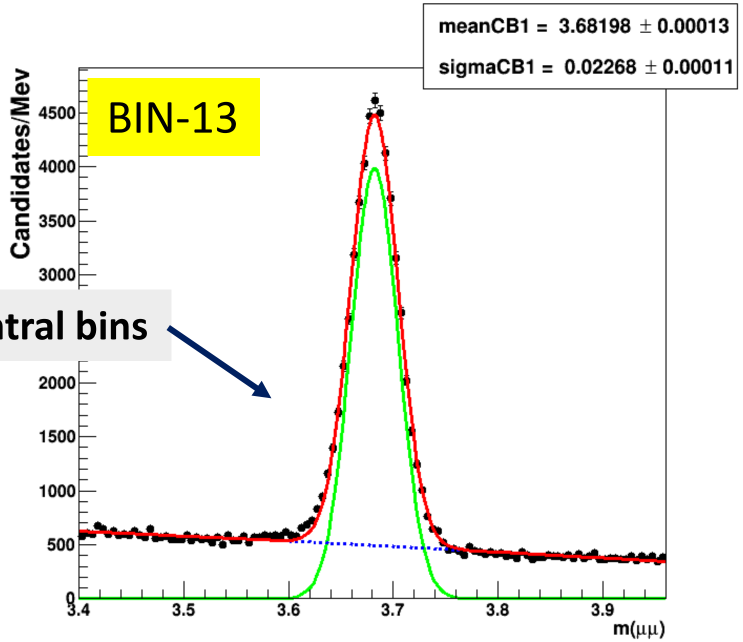


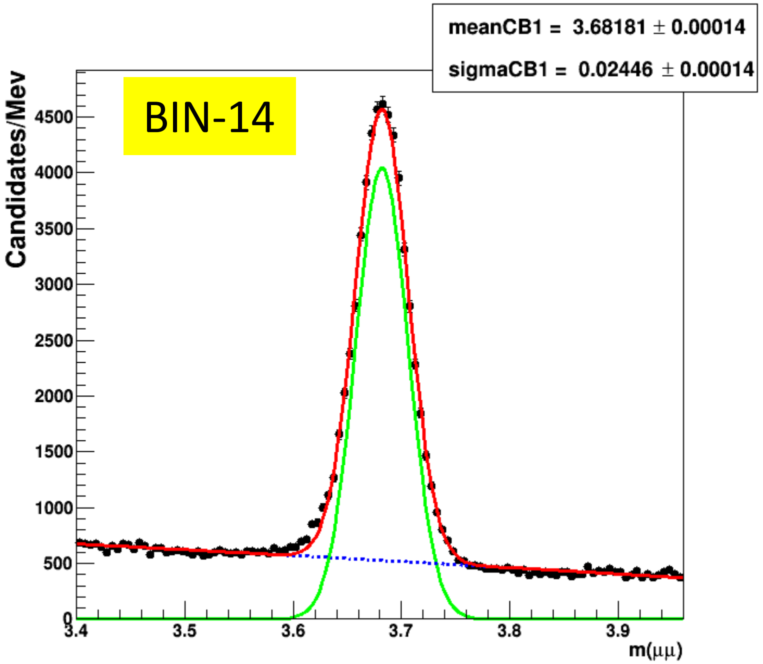


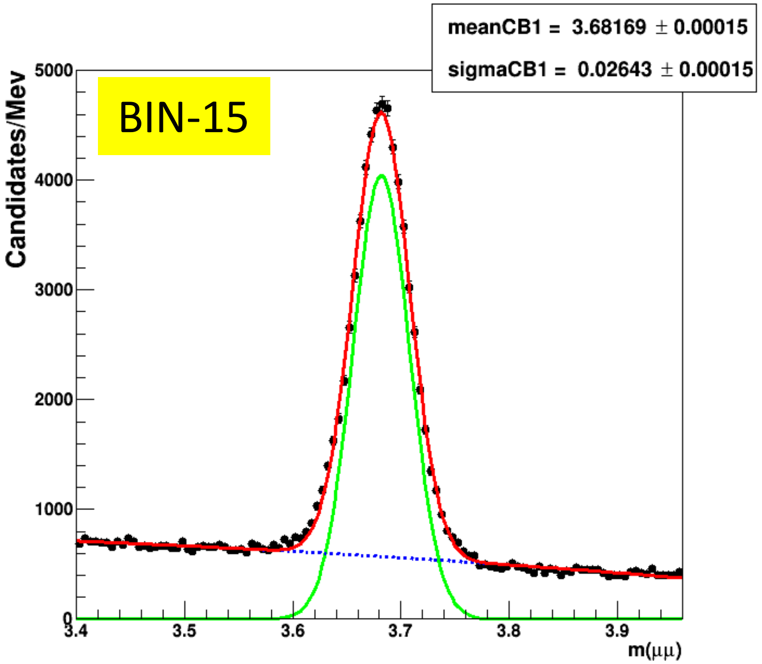


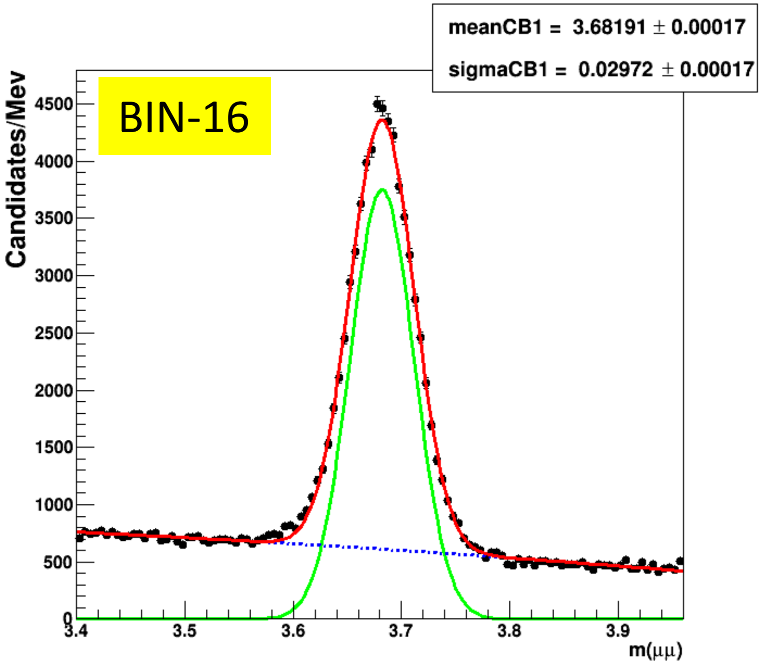


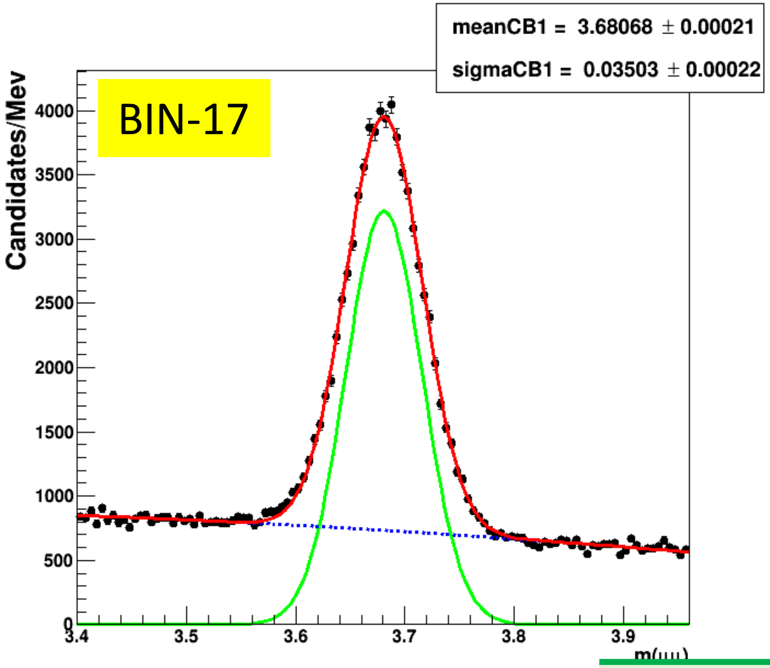
the most central bins



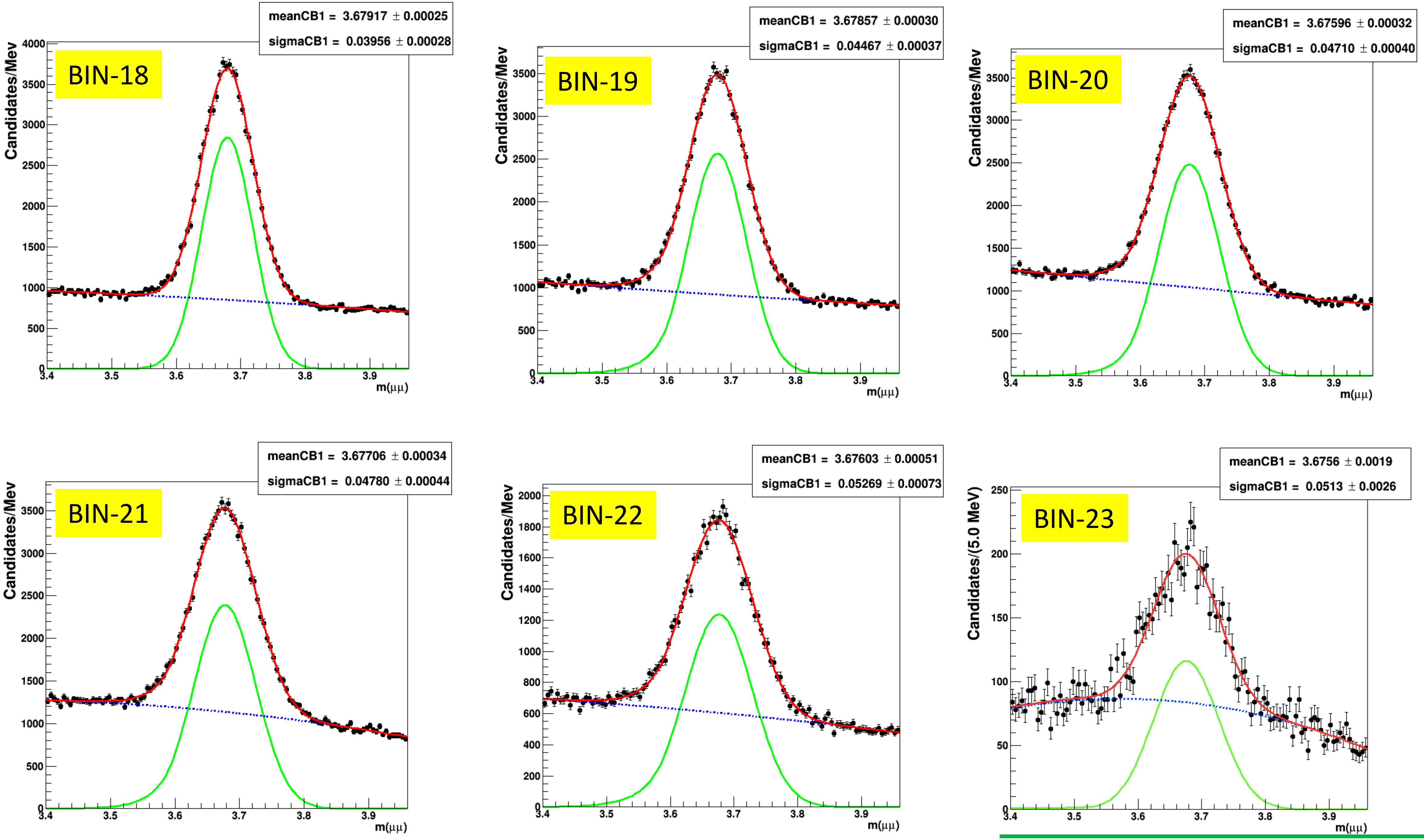








All these plots are produced in the subdir `/Plots` in files `bin#.png` (with #=2,…,23)


meanCB1 = 3.67917 ± 0.00025
sigmaCB1 = 0.03956 ± 0.00028
BIN-18
Candidates/Mev
m(μμ)
meanCB1 = 3.67857 ± 0.00030
sigmaCB1 = 0.04467 ± 0.00037
BIN-19
Candidates/Mev
m(μμ)
meanCB1 = 3.67596 ± 0.00032
sigmaCB1 = 0.04710 ± 0.00040
BIN-20
Candidates/Mev
m(μμ)
meanCB1 = 3.67706 ± 0.00034
sigmaCB1 = 0.04780 ± 0.00044
BIN-21
Candidates/Mev
m(μμ)
meanCB1 = 3.67603 ± 0.00051
sigmaCB1 = 0.05269 ± 0.00073
BIN-22
Candidates/Mev
m(μμ)
meanCB1 = 3.6756 ± 0.0019
sigmaCB1 = 0.0513 ± 0.0026
BIN-23
Candidates/(5.0 MeV)
m(μμ)

Let’s see the macro code used to produce the previous result:

```
#include <TLegend.h>
#include <iostream>
#include <TColor.h>
#include <TAxis.h>
#include <RooRealVar.h>
#include <RooGlobalFunc.h> // needed to resize the text in the statistics box of a frame
//
using namespace std;
using namespace RooFit;
//
void myFinalRapidity(){
  //
  gROOT->Reset();
  gROOT->Clear();
  //
  gROOT->SetStyle("Plain");
  gStyle->SetOptStat(10);
  //
  // prepare file in read mode
  TFile f1("./hlt_5_newSoftMuon_alsoInPsiPrimeWind.root","READ");
  //
  TString numeri[22] =  {"2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23"};
  //
  //-logic : bin-2=-2.2/-2.0 ; bin-3=-2.0/-1.8 ; bin-4=-1.8/-1.6 ; bin-5=-1.6/-1.4 ; bin-6=-1.4/-1.2 ; bin-7=-1.2/-1.0 ; bin-8=-1.0/-0.8 ; bin-9=-0.8/-0.6
  //          bin-10=-0.6/-0.4 ; bin-11=-0.4/-0.2 ; bin-12=-0.2,0.0 ; bin-13=0.0/0.2 ; bin-14=0.2/0.4 ; bin-15=0.4,0.6 ; bin-16=0.6/0.8
  //          bin-17=0.8/1.0 ; bin-18=1.0/1.2 ; bin-19:1.2/1.4 ; bin-20=1.4/1.6; bin-21=1.6/1.8 ; bin-22=1.8/2.0 ; bin-23=2.0/2.2
  //
  //          (skipping two extreme bins: bin-1=-2.4/-2.2 & bin-24=2.2/2.4 because if the uncertainty due to the low statistics)
  //
  //Canvas for my plots:
  //
  TCanvas *myCanvas = new TCanvas("myCanvas", "myCanvas", 700, 700);
  gPad->SetBottomMargin(0.15);
  gPad->SetLeftMargin(0.1);
  gPad->SetRightMargin(0.1);
  gPad->SetTopMargin(0.1);

  //-- vectors for the final graph
  double vec[22];
  double vec_err[22];
  // double vec_err_h[22]; // Se eventualmente usassi Minos
  // double vec_err_l[22]; // Se eventualmente usassi Minos
  //
  //--this is if we want to do give a vector of initial values to the fit:
  //double vec_init_sig_param[22] = {3000,5000,7500,10000,15000,20000,30000,40000,50000,65000,80000,80000,65000,50000,40000,30000,20000,15000,10000,7500,500
0,40000};
  //
```

Starting LOOP on the 22 BINS

```
for (int i = 0; i<22; i++)
  {
   // Open file and read histogram
   TH1D *histo = (TH1D*)f1.Get("PsiPrimeMass_bin"+numeri[i]);
   //
   Int_t nBins = histo->GetNbinsX();
   Float_t bin_width = histo->GetBinWidth(1.);  // needed later
   //
   // Creation of the object RooDataHist from the histogram
   RooRealVar x("x", "x", 3.4, 3.96);
   RooDataHist *psiprime = new RooDataHist(histo->GetName(), histo->GetTitle(), RooArgSet(x), RooFit::Import(*histo, kFALSE));
   //
   //Frame for plotting and style options
   RooPlot *xframe = x.frame(Title(""));
   //xframe->SetTitle("#mu#mu invariant mass spectrum");
   xframe->SetTitle("");
   xframe->SetTitleOffset(1.32,"y");
   xframe->SetLabelSize(0.025, "y");
   xframe->SetTitleSize(0.038, "y");
   //
   char newlabel[255];
   sprintf(newlabel, "Candidates/(%.1f MeV)", 1000*bin_width);
   xframe->SetYTitle(newlabel);
   //
   xframe->SetLabelSize(0.025, "x");
   xframe->SetTitleSize(0.029, "x");
   xframe->SetTitleOffset(0.93,"x");
   xframe->SetXTitle("m(#mu#mu)");
   //
   // "putting" the RooDataHist object on the RooPlot
   psiprime->plotOn(xframe);
   //
   //
   //char title[128]=""; // taking off the title:
   psiprime->SetTitle("");
   //
```

```
//---------------START FIT
//
// Signal peak
RooRealVar meanCB1("meanCB1", "meanCB1", 3.675, 3.64, 3.72);
RooRealVar sigmaCB1("sigmaCB1", "sigmaCB1", 0.040, 0.001, 0.700);
RooRealVar alpha1("alpha1", "alpha1", 1.5, 0.01, 15.);
RooRealVar nCB1("nCB1", "nCB1", 2.0, 0.001, 20.);
//
//cout << "\n alpha1 is = " << alpha1.getVal() <<  ", when i = " << i << endl; // just a test
//
RooCBShape myCB1("myCB1", "myCB1", x, meanCB1, sigmaCB1, alpha1, nCB1);
//
// Background shape
//
// Try Chebychev Poly of order2
//
RooRealVar c1("c1", "1st coeff", -0.30, -1000, 100); //originale -0.3
RooRealVar c2("c2", "2nd coeff", 0.1, -1000, 100); // originale 0.01
RooChebychev cheby("cheby", "Chebichev 2", x, RooArgList(c1,c2));
//
// Signal & Background yields
//-- it is possible to feed a vector of starting values if needed
//RooRealVar nSig("nSig", "Number of signal candidates", vec_init_sig_param[i], 1e+2, 1e+8);
RooRealVar nSig("nSig", "Number of signal candidates", 4e+5, 1e+2, 1e+8);
RooRealVar nBkg("nBkg", "Number of bkgrd candidates", 60e+4, 1e+2, 1e+8);
//
RooAddPdf *totalPdf = new RooAddPdf("totalPdf", "totalPdf", RooArgList(myCB1, cheby), RooArgList(nSig, nBkg));
//
// Fit command
totalPdf->fitTo(*psiprime, Extended(kTRUE)); // by default only HESSE is executed (MINOS is not)
//
// Filling vectors
vec[i] = 1000.0*sigmaCB1.getValV(); // multiply by 1000 to get MeV
//vec_err_h[i]= sigmaCB1.getAsymErrorHi(); // If I use MINOS in the fit
//vec_err_l[i]= sigmaCB1.getAsymErrorLo(); // If I use MINOS in the fit
vec_err[i] = 1000*sigmaCB1.getError();
//
totalPdf->plotOn(xframe, RooFit::LineColor(kRed));
totalPdf->plotOn(xframe, RooFit::Components(RooArgSet(myCB1)),LineColor(kGreen));
totalPdf->plotOn(xframe, RooFit::Components(RooArgSet(cheby)), RooFit::LineStyle(kDashed));
//
// Redraw the full fit curve to allow for correct pulls (if will be done)
totalPdf->plotOn(xframe, RooFit::LineColor(kRed));
//
// Box of parameters
totalPdf->paramOn(xframe, Parameters(RooArgList(meanCB1, sigmaCB1)),Layout(0.57, 0.998, 0.99));
//
// Do the draw of the frame
myCanvas->cd();
//
xframe->getAttText()->SetTextSize(0.03) ;
xframe->Draw();
//
myCanvas->SaveAs("./Plots/bin"+numeri[i]+".png");
//
} // for-loop closed
//
gSystem->Sleep(15000);
myCanvas->Clear();
delete myCanvas;
//
```

Crystal Ball

Chebyshev (2nd ord.)

Crystal Ball + Chebyshev (2nd ord.)

Extended Binned ML fits (one per bin)

Closing LOOP on the 22 BINS

(the filled vectors will be used in the 2nd part)

I finished the set of 22 fits in rapidity bins and with the filled vectors **vec[i]** & **vec_err[i]**;
Now I can start the 2nd part where I plot and fit the mass resolution as a function of the rapidity.

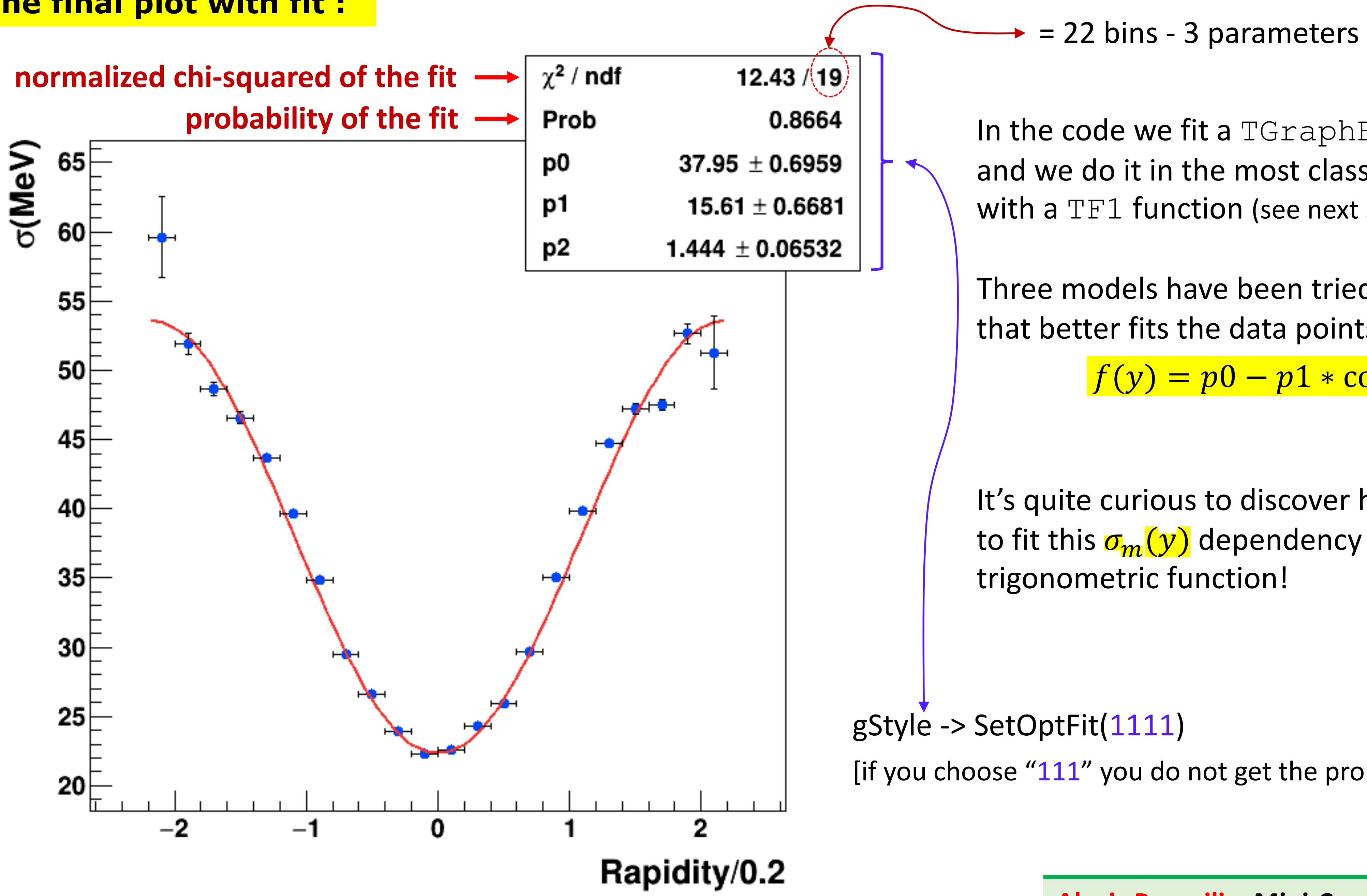
```
////////////////////////////////////////////- 2nd PART - ////////////////////////////
//
// As central values we assign the center of each rapidity bin
double rapidity[22] = {-2.1, -1.9, -1.7, -1.5, -1.3, -1.1, -0.9, -0.7, -0.5, -0.3, -0.1, 0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5, 1.7, 1.9, 2.1};
// As uncertainty we assign the bin's half-width (0.1 in this case)
double rapidity_err[22];
for (int i=0; i<22; i++)
   {
    rapidity_err[i]=0.1;
   }
//
//-- if the fit would have been providing asymmetric errors on the parameter.
//TGraphAsymmErrors *grafico_errori = new TGraphAsymmErrors(22, rapidity, vec, rapidity_err, rapidity_err, vec_err_l, vec_err_h);
//
TCanvas *final = new TCanvas("final", "final", 700, 700);
TGraphErrors *grafico_errori = new TGraphErrors(22, rapidity, vec, rapidity_err, vec_err);
gPad->SetBottomMargin(0.15);
gPad->SetLeftMargin(0.15);
gPad->SetRightMargin(0.1);
gPad->SetTopMargin(0.1);
grafico_errori->SetMarkerStyle(20);
grafico_errori->SetMarkerColor(kBlue);
grafico_errori->SetTitle("");
grafico_errori->GetXaxis()->SetTitle("Rapidity/0.2");
grafico_errori->GetXaxis()->SetLabelSize(0.03);
grafico_errori->GetXaxis()->SetTitleOffset(1.1);
grafico_errori->GetYaxis()->SetTitleOffset(1.1);
grafico_errori->GetYaxis()->SetLabelSize(0.03);
grafico_errori->GetYaxis()->SetTitle("#sigma(MeV)");
grafico_errori->GetYaxis()->SetDecimals(1);
grafico_errori->Draw("AP");
//
// Try a Parabola
//TF1 *myFunc = new TF1("myFunc", "[0]+ [1]*x + [2]*x*x", -2.2, 2.2);
//myFunc->SetParameters(0.02, 1, 1);

// Try an hyperbolic Cosine
//
//TF1 *myFunc = new TF1("myFunc", "[0]+ 0.5*[1]*TMath::Exp([2]*x)+0.5*[1]*TMath::Exp(-[2]*x)", -2.2, 2.2);
// myFunc->SetNpx(100);
//myFunc->SetParameters(30., 1, 0.01);
//myFunc->SetParLimits(0,-10, 22);
//myFunc->SetParLimits(1, 0., 30.);
//myFunc->SetParLimits(2,-100, 100);
//
// Try a simple Cosine:
TF1 *myFunc = new TF1("myFunc", "[0]-[1]*TMath::Cos([2]*x)", -2.2, 2.2);
myFunc->SetParameters(35., 15., 1.);
myFunc->SetParLimits(0,10, 60);
myFunc->SetParLimits(1, 5., 30.);
myFunc->SetParLimits(2,-10, 10);
//
myFunc->SetLineColor(kRed);
myFunc->SetLineWidth(2);
//
gStyle->SetOptFit(1111);
grafico_errori->Fit(myFunc,"R");
//
final->SaveAs("./Plots/grafico_finale.png");
//
delete final;
}
```

setup graph

fitting the `TGraphErrors` object

The final plot with fit :



In the code we fit a `TGraphError` object and we do it in the most classic `ROOT` way, i.e. with a `TF1` function (see next slides).

Three models have been tried but this is the one that better fits the data points:

$$f(y) = p0 - p1 * \cos(p2 * y)$$

It's quite curious to discover how it is possible to fit this $\sigma_m(y)$ dependency with such a simple trigonometric function!

gStyle -> SetOptFit(1111)

[if you choose "111" you do not get the probability of the fit]

Homework : …. a “variation on the theme”:

show the distribution of the signal yield (number of candidates) as a function of rapidity bin

(use again a `TGraphErrors`)