Estimation of the Figures Of Merit (*significance*, *purity* & *signal-to-bkg ratio*) of a physics signal

A.Pompili - SDAL course - Exercise 11a

The code is mutuated from the previous macro, modified and now called **myGenExpGausFOMSignal.C**

For presentation purposes we consider a stronger - than earlier - generated signal.

```
void myGenExpGausFOMSignal(TString argv, int bins){
  //
  gROOT->SetStyle("Plain");
  gStyle->SetOptStat(10);
  gStyle->SetOptFit(111);
  //
  int events = atoi(argv.Data());  // converte string "numero" in numero intero
  TString name = argv;
  //
  RooRealVar xvar("xvar", "", 18., 34.);
  xvar.setBins(bins);
  //
  //-- BKG MODEL
  //
  RooRealVar m0("m0","m0",-0.1, -2., 2.);
  RooExponential myExp("myExp","Exponential", xvar, m0);
  //
  //-- SIGNAL MODEL
  //
  RooRealVar meanG("meanG","Gaussian mean",26., 25., 27.);
  RooRealVar sigmaG("sigmaG","Gaussian sigma/resolution",0.4,0.36,0.44);
  RooGaussian myGauss("myGauss","Gaussian",xvar,meanG,sigmaG);
  //
  //-- TOTAL MODEL
  //
  cout << "Events = " << events << endl;
  //
  //---suppose a signal rapresented by the 2% of the whole distribution:
  double sigFrac = 0.02;
  //
  int sigCand = sigFrac * events;
  //int sigCandM = 0.1 * sigFrac * events;
  //int sigCandM =0;
  int sigCandP = 5 * sigFrac * events;
  cout << "sigCand =" << sigCand << endl;
  //
  int bkgCand = (1 - sigFrac) * events;
  int bkgCandM = 0.1 * (1 - sigFrac) * events;
  int bkgCandP = 5 * (1 - sigFrac) * events;
  cout << "bkgCand =" << bkgCand << endl;
  //
  // note that signal yield is positive by definition; generated value is given by sigCand:
  RooRealVar yield_sig("yield_sig","yield of Gaussan signal component", sigCand, 0, sigCandP);
  RooRealVar yield_bkg("yield_bkg","yield of Exponential bkg component", bkgCand, bkgCandM, bkgCandP);
  RooAddPdf total("totalPDF", "totalPDF", RooArgList(myGauss,myExp), RooArgList(yield_sig,yield_bkg));
  //
  //--> Generating pseudo-data
  ////////////////////////////
  //
  timeval trand;
  gettimeofday(&trand,NULL);
  long int msRand = trand.tv_sec * 1000 + trand.tv_usec / 1000;
  cout << "\n------" << endl;
  cout << "msRand = " << msRand;
  cout << "\n------" << endl;
  RooRandom::randomGenerator()->SetSeed(msRand);
  //
  RooDataSet* data = total.generate(xvar,events);
  TH1D* histo_data = (TH1D*)data->createHistogram("histo_data",xvar,Binning(bins,xvar.getMin(),xvar.getMax()));
  //
```

```
/////////////////////////////////////////////////////
////
//// root [0] .L myGenExpGaussFOMSignal.C++
//// root [1] myGenExpGaussFOMSignal("100000",100)
////
/////////////////////////////////////////////////////
```

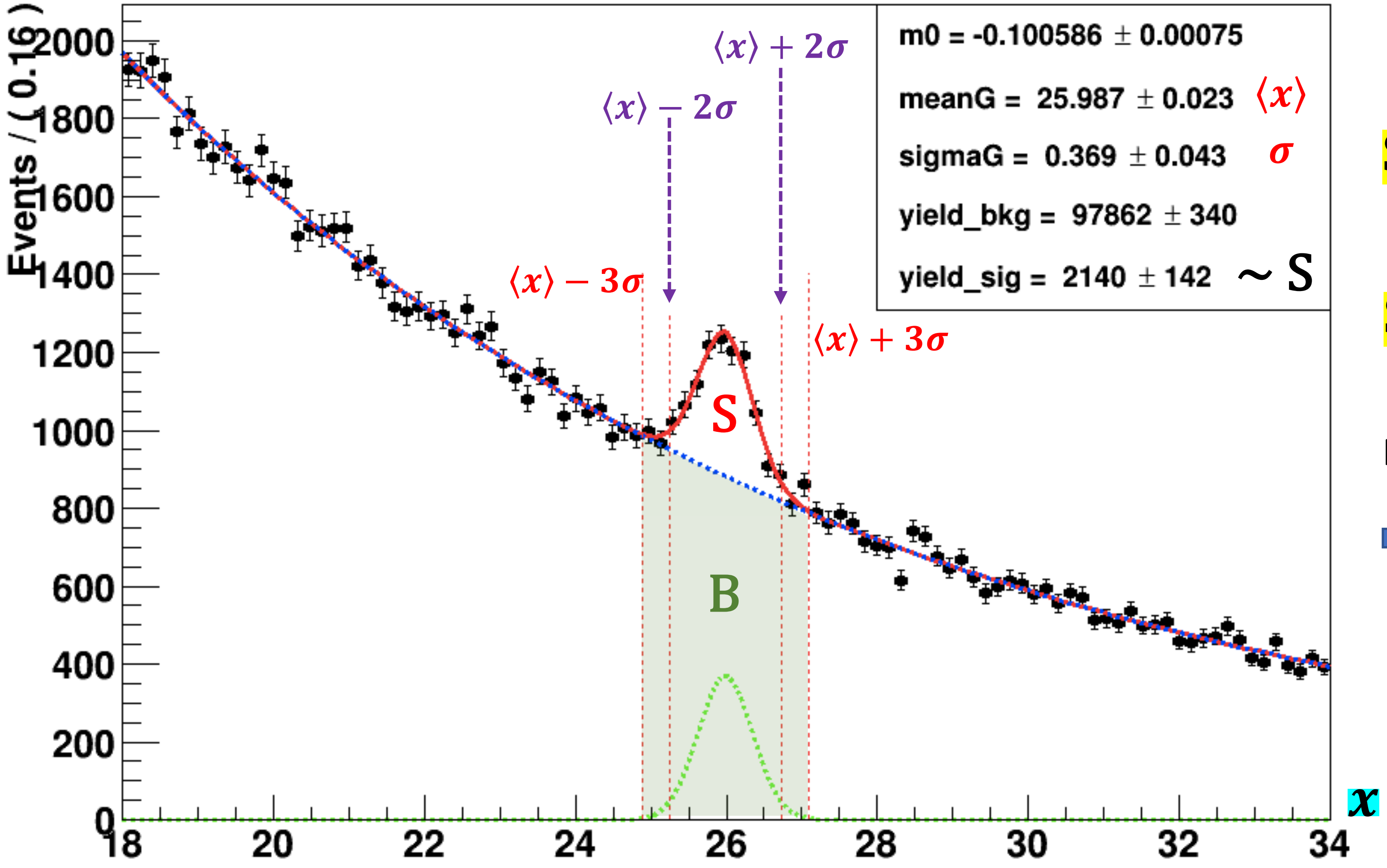
I need also the histogram …
not only the unbinned data

The fits' sequence is similar to the one in the previous exercise.
The first is the full free fit:

```
cout << "==============================================================" << endl;
cout << "FULLY FREE FIT : Gaussian model for the fit without constraints " << endl;
cout << "--------------------------------------------------------------" << endl;
//
//--> Fitting pseudo-data with total model
//
// Note that fit is EXTENDED without having to say it explicitely
//
RooAbsReal* nll_free = total.createNLL(*data,NumCPU(4));
//RooMinuit min_free(*nll_free); // changing a bit with previous exercises: using Minuit2 now.
RooMinimizer min_free(*nll_free);
min_free.setMinimizerType("Minuit2");
min_free.migrad();
min_free.hesse();
RooFitResult* fitres_free = min_free.save();
//
```

We will estimate the FOMs after the full free fit, like we would do in a real situation with real data.

This is the full free fit result:



The most relevant FOMs:

SS Signal Significance $= \dfrac{S^2}{(S+B)}$ or $\dfrac{S}{\sqrt{S+B}}$

SP Signal Purity $= \dfrac{S}{(S+B)}$

Note that: $SS = \dfrac{S^2}{(S+B)} = S \cdot \dfrac{S}{(S+B)} = S \cdot SP$

Maximizing both are incompatible operations

S/B Signal-to-Bkg ratio $= \dfrac{S}{B}$

Typically…

A candidates' selection aims to maximize the SS to better extract a rare signal & estimate - for instance - its Branching Fraction. It's usually calculated in a 3σ -window, namely: $[\langle x \rangle - 3\sigma, \langle x \rangle + 3\sigma]$. In this case the selection is *weaker* than in the next case.

A candidates' selection aims to maximize the SP to extract an enough pure set of signal candidates to be used for the measurement - for instance - of a property of the physical signal. It's usually calculated in a 2σ -window i.e.: $[\langle x \rangle - 2\sigma, \langle x \rangle + 2\sigma]$

```
//
//=========== Figures Of Merit study (significance, purity, ...)
//
//--> take the full free fit and immagine we do not have generated the data
//
Double_t mean_free_fit = meanG.getVal();
Double_t sigma_free_fit = sigmaG.getVal();
//
cout << "Best value for parameter meanG by the full free fit = " << mean_free_fit << endl;
cout << "Best value for parameter sigmaG by the full free fit = " << sigma_free_fit << endl;
//
//--> define a signal window:
Double_t minus_3s = mean_free_fit - 3.0*sigma_free_fit;
Double_t plus_3s = mean_free_fit + 3.0*sigma_free_fit;
cout << minus_3s << " -- 3sigma interval -- " << plus_3s << endl;
//
Double_t minus_2s = mean_free_fit - 2.0*sigma_free_fit;
Double_t plus_2s = mean_free_fit + 2.0*sigma_free_fit;
cout << minus_2s << " -- 2sigma interval -- " << plus_2s << endl;
//
//--> associate border lines to this window:
TLine *line_minus = new TLine(minus_3s,0.,minus_3s,1400.);
line_minus->SetLineColor(2);
line_minus->SetLineWidth(1);
line_minus->SetLineStyle(2);
TLine *line_plus = new TLine(plus_3s,0.,plus_3s,1400.);
line_plus->SetLineColor(2);
line_plus->SetLineWidth(1);
line_plus->SetLineStyle(2);
//
TLine *line_minus_2s = new TLine(minus_2s,0.,minus_2s,1300.);
line_minus_2s->SetLineColor(2);
line_minus_2s->SetLineWidth(1);
line_minus_2s->SetLineStyle(2);
TLine *line_plus_2s = new TLine(plus_2s,0.,plus_2s,1300.);
line_plus_2s->SetLineColor(2);
line_plus_2s->SetLineWidth(1);
line_plus_2s->SetLineStyle(2);
//
xvar.setRange("peakRange",minus_3s,plus_3s);
RooArgSet nSetPeak(xvar);
xvar.setRange("peakRange2s",minus_2s,plus_2s);
RooArgSet nSetPeak2s(xvar);
//
```

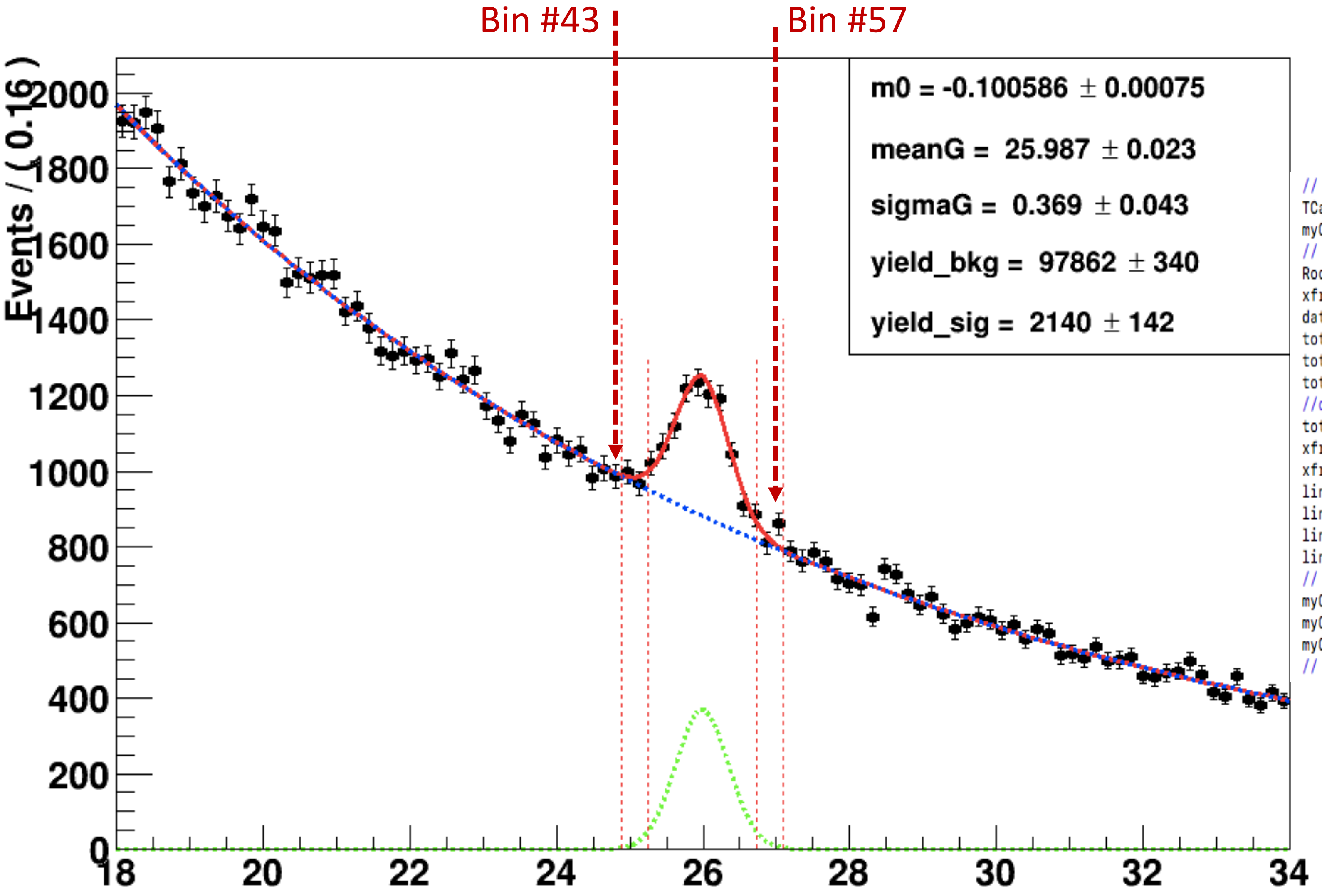
```
xvar.setRange("peakRange",minus_3s,plus_3s);
RooArgSet nSetPeak(xvar);
xvar.setRange("peakRange2s",minus_2s,plus_2s);
RooArgSet nSetPeak2s(xvar);
//
//-- I need the post-fit PDF of the signal S
RooAbsReal* fracSigRange = myGauss.createIntegral(nSetPeak,nSetPeak,"peakRange"); // fraction of the fitted signal in the window
Double_t nSig_window_3s = yield_sig.getVal() * fracSigRange->getVal();
cout << "Signal candidates in +/-3s window: " << nSig_window_3s << endl;
//
RooAbsReal* fracSigRange2s = myGauss.createIntegral(nSetPeak2s,nSetPeak2s,"peakRange2s");
Double_t nSig_window_2s = yield_sig.getVal() * fracSigRange2s->getVal();
//
//-- I need the post-fit PDF of the background B
RooAbsReal* fracBkgRange = myExp.createIntegral(nSetPeak,nSetPeak,"peakRange"); // fraction of the fitted bkg in the window
RooFormulaVar nBkg_window_3s("nBkg_window_3s","(@0*@1)",RooArgList(*fracBkgRange,yield_bkg));
cout << "Bkg candidates in +/-3s window: " << nBkg_window_3s.getVal() << endl;
//
RooAbsReal* fracBkgRange2s = myExp.createIntegral(nSetPeak2s,nSetPeak2s,"peakRange2s");
RooFormulaVar nBkg_window_2s("nBkg_window_2s","(@0*@1)",RooArgList(*fracBkgRange2s,yield_bkg));
//
//-- I need the total post-fit PDF of S+B
RooAbsReal* fracTotRange = total.createIntegral(nSetPeak,nSetPeak,"peakRange"); // fraction of the fitted total in the window
cout << "Total fraction of candidates effectively +/-3s window: fracTotRange = " << fracTotRange->getVal() << endl;
Double_t nTot_window_3s = (yield_sig.getVal() + yield_bkg.getVal()) * fracTotRange->getVal();
cout << "Total candidates in +/-3s window: " << nTot_window_3s << endl;
//
RooAbsReal* fracTotRange2s = total.createIntegral(nSetPeak2s,nSetPeak2s,"peakRange2s");
Double_t nTot_window_2s = (yield_sig.getVal() + yield_bkg.getVal()) * fracTotRange2s->getVal();
//
//
//-- signal candidates can be estimated in another way to avoid relying on the fit model
//   namely by subtraction of the bkg candidates from the entries of the histogram in the signal window / under the peak
//Double_t integral_below_peak = histo_data->Integral(minus_3s,plus_3s,"");
//-- For this purpose is better to identify first the border bins:
cout << "Border Bin-minus = " << histo_data->FindFixBin(minus_3s) << endl;
cout << "Border Bin-plus = " << histo_data->FindFixBin(plus_3s) << endl;
//-- let's integrate between the border bins:
Double_t histo_integral_below_peak = histo_data->Integral(histo_data->FindFixBin(minus_3s),histo_data->FindFixBin(plus_3s),"");
Double_t histo_integral_below_peak_2s = histo_data->Integral(histo_data->FindFixBin(minus_2s),histo_data->FindFixBin(plus_2s),"");
//
cout << "Integral of the histogram below the peak (3s window) = " << histo_integral_below_peak << endl;
Double_t nSig_window_3s_alter = histo_integral_below_peak - nBkg_window_3s.getVal();
cout << "Signal candidates in +/-3s window as histo-bkg (alternative): " << nSig_window_3s_alter << endl;
Double_t nSig_window_2s_alter = histo_integral_below_peak_2s - nBkg_window_2s.getVal();
//
cout << "Significance in the +/-3s signal window = " << (nSig_window_3s / sqrt(nSig_window_3s + nBkg_window_3s.getVal())) << endl;
cout << "Significance in the +/-3s signal window (alternative) = " << (nSig_window_3s_alter / sqrt(histo_integral_below_peak)) << endl;
//
// cout << "Purity in the +/-2s signal window = " << (nSig_window_2s / (nSig_window_2s + nBkg_window_2s.getVal())) << endl;
//-- we can also use more elegantly:
cout << "Purity in the +/-2s signal window = " << (nSig_window_2s / nTot_window_2s) << endl;
cout << "Purity in the +/-2s signal window (alternative) = " << (nSig_window_2s_alter / histo_integral_below_peak_2s) << endl;
//
cout << "Signal-to-noise in the +/-3s signal window = " << (nSig_window_3s / nBkg_window_3s.getVal()) << endl;
cout << "Signal-to-noise in the +/-3s signal window (alternative) = " << (nSig_window_3s_alter / nBkg_window_3s.getVal()) << endl;
//
//========================================
```

The previous piece of code provides the following output with the relevant results **(*)** :

```
24.8796 -- 3sigma interval -- 27.0938
25.2487 -- 2sigma interval -- 26.7248
[#1] INFO:Eval -- RooRealVar::setRange(xvar) new range named 'peakRange' created with bounds [24.8796,27.0938]
[#1] INFO:Eval -- RooRealVar::setRange(xvar) new range named 'peakRange2s' created with bounds [25.2487,26.7248]
Signal candidates in +/-3s window: 2133.82
Bkg candidates in +/-3s window: 12226.3
Total fraction of candidates effectively +/-3s window: fracTotRange = 0.143599
Total candidates in +/-3s window: 14360.2
Border Bin-minus = 43
Border Bin-plus = 57
Integral of the histogram below the peak (3s window) = 15519
Signal candidates in +/-3s window as histo-bkg (alternative): 3292.65
Significance in the +/-3s signal window = 17.8064
Significance in the +/-3s signal window (alternative) = 26.431
Purity in the +/-2s signal window = 0.200538
Purity in the +/-2s signal window (alternative) = 0.252795
Signal-to-noise in the +/-3s signal window = 0.174526
Signal-to-noise in the +/-3s signal window (alternative) = 0.269308
```

Note that the difference between the standard and the alternative **(*)** evaluation happens to be because the values (lines) delimiting the windows (i.e. $[\langle x\rangle - n\sigma, \langle x\rangle + n\sigma]$ with n=2,3) cannot cover exactly the border bins! (see next slide)
The difference can change itself from one generation to the other, becoming sometimes small.
Increasing the binning (i.e. reducing the bin-width) would help to reduce the difference.

(*) "alternative" means I use the histogram (integrated on the window) instead of the total fit function (integrated on the range)




```
//
TCanvas *myC = new TCanvas("RooCanvas","Roofit Canvas", 1000, 700);
myC->cd();
//
RooPlot* xframe_free = xvar.frame("");
xframe_free->SetTitle("");
data->plotOn(xframe_free);
total.plotOn(xframe_free,LineColor(kRed));
total.plotOn(xframe_free,Components(RooArgSet(myGauss)),LineColor(kGreen),LineStyle(kDashed));
total.plotOn(xframe_free,Components(RooArgSet(myExp)),LineColor(kBlue),LineStyle(kDashed));
//data->plotOn(xframe_free);
total.paramOn(xframe_free, Layout(0.6,0.9,0.9));
xframe_free->getAttText()->SetTextSize(0.03);
xframe_free->Draw();
line_minus->Draw("same");
line_plus->Draw("same");
line_minus_2s->Draw("same");
line_plus_2s->Draw("same");
//
myC->SaveAs("myFullFreeFit_StrongSignal.png");
myC->Update();
myC->Clear();
//
```

Bin #43 is included in the Integral of the histogram in the signal region thus overestimating the bkg & total;
the bkg can be instead taken by the bkg-fit model and that would not include the bin #43 contribution:
this explains why the bkg from the fit is underestimated or the bkg/total from the histogram is overestimated.
In general the alternative way of calculation overestimates the FOMs.

The rest of the fits is performed as usual to extract the **Statistical Significance of the Signal** (as seen in the earlier exercise):

```
============================= MINIMUM NLL for each FIT =======================================
lambda_0 : -783737
-----------------------------
lambda_1 : -783995
-----------------------------
q0 = 2*(lambda_0 - lambda_1) : 515.052
----------------------------------------------------------------------------------------------
============================= STAT. SIGNIF. Method-1 (STA-SIGNIF-1) =====================================
STAT. SIGNIF. 1st method = Z0 = sqrt(q0) i.e. eq.(52) by Cowan et al. (EPJC,2011): 22.6948
-----------------------------------------------------------------------------------------------------------
```