Introdução à Análise de dados em FAE **(11/04/2024)**

Lista de exercícios 7

Professores: Sandro Fonseca, Eliza Melo, Maurício Thiel e Diego Torres *Name:* Miguel Lopes

EXERCÍCIO 1

Para esse exercício escolhi o pico do Z^0. Plotei o pico com e sem os cortes de $p_T \geq 8$ GeV/c e $\|\eta\| \leq 2{,}4$, não obtive diferença nos plotes, isso se dá por que esses cortes já são os limites do CMS para a detecção de μ. O ajuste no pico de ressonância foi realizado com a soma de uma CBS e uma Breit-Wigner. A baixo está o código que elaborei para essa questão e o plot do pico da massa do Z^0.

```
#define anadimuon_cxx
#include "anadimuon.h"

#include "RooRealVar.h"
#include "RooDataSet.h"
#include "RooDataHist.h"
#include "RooGaussian.h"
#include "RooExponential.h"
#include "RooCBShape.h"
#include "RooAddPdf.h"
#include "TCanvas.h"
#include "RooPlot.h"
#include "TTree.h"
#include "TH1D.h"
#include "TRandom.h"
#include "TMath.h"
#include <RooCBShape.h>
#include <RooBreitWigner.h>
using namespace RooFit ;

std::string extraString = "";

void anadimuons()
{

  // Load the dataset and prepare the
  TChain * chain = new TChain("oniaTree","");
  chain->Add("DataSkim4.root");
  anadimuon a(chain);

  // Select a subsample of the data with a number >= 0
  // Use the whole data with a negative number
  // For the exercise: fill with the day of the month of your birthday (1->31)
  a._sdig = -1;

  // Produce the full dimuon mass spectrum
  a.GetSpectrum();

  // Set limits for the histogram to use in the fit
  // a._mmin = 2.9;
  // a._mmax = 3.3;
  a._mmin = 60;
  a._mmax = 120;
  //extraString = "_Jpsi";       // NOTE: this is used in file name:
  extraString = "_Z0";  // NOTE: this is used in file name:
                               // do not use unsafe characters
```

```
  a.SelectPeak();

  // Fit with RooFit
  a.FitPeakRoofit();

  // Optional: you can try to fit with ROOT and compare the yield results
  // a.FitPeak();


}

// this is the main processing function
void anadimuon::GetSpectrum() {

  // check tree
  if (fChain == 0) return;

  // create and fill a simple mass histogram
  TH1F *hDimuonMass_normal = new TH1F("hDimuonMass_normal","hDimuonMass_normal"
      ,10000,0.2,200);
  FillHisto(hDimuonMass_normal);
  SaveHisto(hDimuonMass_normal);

  // now set log scales
  SaveHisto(hDimuonMass_normal,kTRUE);

  //define another (special) histogram: with variable (!) bin widths

  double xbins[100000];
  xbins[0] = .1;
  int nbins = 0;
  double binWidth=0.005;
  for (int i=1; xbins[i-1]<500; i++) {
    xbins[i] = xbins[i-1]*(1+binWidth);
    nbins++;
  }
  TH1F *hDimuonMass = new TH1F("hDimuonMass","hDimuonMass",nbins,xbins);
  FillHisto(hDimuonMass);
  // SaveHisto(hDimuonMass,kTRUE);

  // now: normalize yields (to adapt to variable binning!)
  for (int i=1; i<=hDimuonMass->GetNbinsX(); i++) {
    hDimuonMass->SetBinContent(i, hDimuonMass->GetBinContent(i)/hDimuonMass->
        GetBinWidth(i));
  }
  SaveHisto(hDimuonMass,kTRUE);

}

void anadimuon::SaveHisto(TH1F* hist, Int_t log) {

  gStyle->SetOptStat(0);
  gStyle->SetOptTitle(0);

  hist->GetXaxis()->SetTitle("#mu^{+}#mu^{-} invariant mass [GeV]");
  hist->GetYaxis()->SetTitle("Events / GeV");

  TCanvas *c = new TCanvas("c","c",800,600);

  if(log) {
    c->SetLogx();
    c->SetLogy();
  }
```

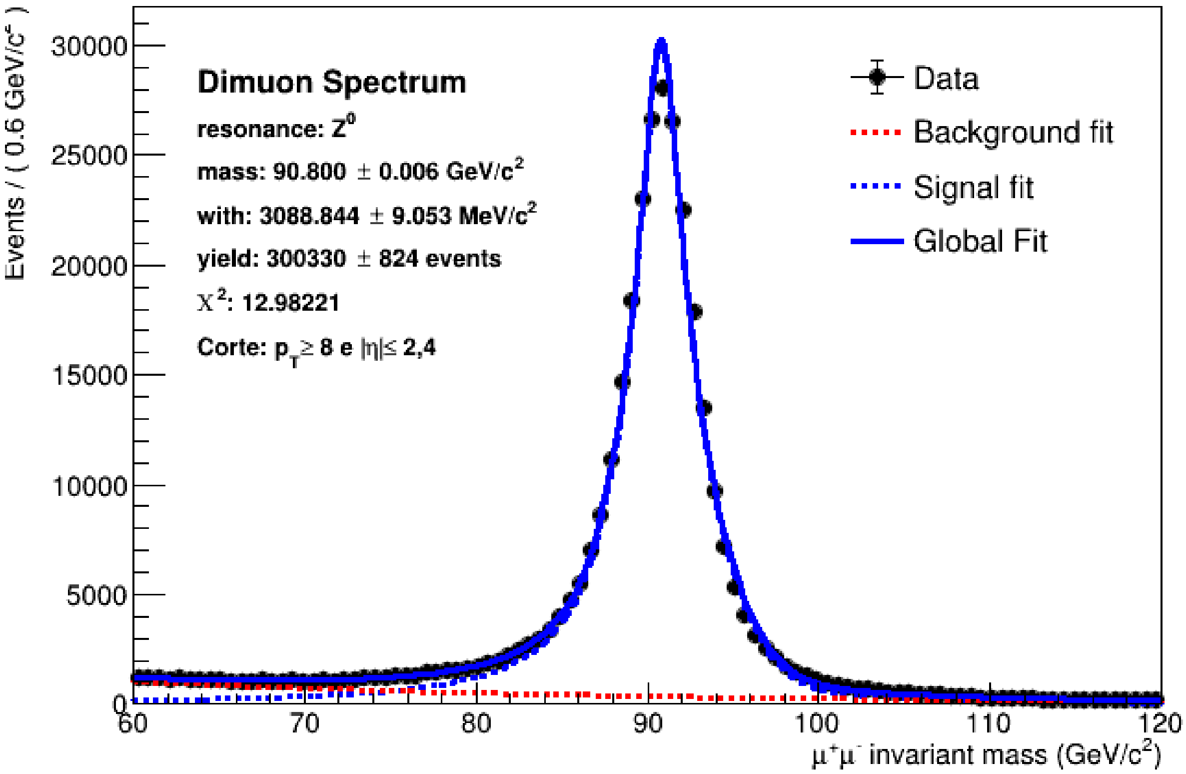
```

  hist->Draw("HIST");

  TString hn ("");
  hn += "plots/";
  hn += hist->GetName();
  if(log) hn += "_log";
  hn += ".png";
  c->SaveAs(hn);
  delete c;

  //TH1F* h2 = (TH1F*)hist->Clone();
  //h2->SetName(hn);
  _outFile->cd();
  hist->Draw("HIST");
  hist->Write();
  _outFile->Write();

}

void anadimuon::FillHisto(TH1F* hist) {

  // loop over the tree, and fill the histograms
  Long64_t maxEntries = fChain->GetEntries();

  Long64_t firstEntry = 0;
  Long64_t lastEntry = maxEntries;

  if (_sdig>=0) {
    firstEntry = (_sdig%10)*(maxEntries/10);
    lastEntry  = (_sdig%10+1)*(maxEntries/10)-1;
  }
  _nev = lastEntry-firstEntry+1;
  cout<<"Selecting "<<_nev<<" events ("<<firstEntry<<" -> "<<lastEntry<<")"<<endl;

  Long64_t nbytes = 0, nb = 0;
  for (Long64_t jentry=firstEntry; jentry<=lastEntry;jentry++) {

    Long64_t ientry = LoadTree(jentry);
    if (ientry < 0) break;
    nb = fChain->GetEntry(jentry);   nbytes += nb;

      //CORTES
        bool passCuts = true;
        double pTMin = 8.0;
        double etaMax = 2.4;

        // Cortes mu1
        if (muonP_p4->Pt() < pTMin || fabs(muonP_p4->Eta()) > etaMax) {
            passCuts = false;
        }

        // cortes mu2
        if (muonN_p4->Pt() < pTMin || fabs(muonN_p4->Eta()) > etaMax) {
            passCuts = false;
        }

        if (!passCuts) continue;

    //if ( Cut(ientry) < 0) continue;

    double mass = dimuon_p4->M();

```

```
    hist->Fill(mass);
  }

}


void anadimuon::SelectPeak()
{

  // create an histogram around a peak
  TH1F *hDimuonMass_peak = new TH1F("hDimuonMass_peak","hDimuonMass_peak",100,_mmin,
      _mmax);
  FillHisto(hDimuonMass_peak);
  SaveHisto(hDimuonMass_peak);

}


void anadimuon::FitPeakRoofit()
{

  // retrive histogram with selected peak
  TH1F* hpeak= 0;
  TString hname("hDimuonMass_peak");
  if(!_outFile) {cout << "Check input file." << endl; return;}
  _outFile->GetObject(hname,hpeak);
  if (!hpeak) {
    cout << "Check input histogram:" << hname <<  endl;
    return;
  }

  // Declare observable mass
  RooRealVar mass("mass","#mu^{+}#mu^{-} invariant mass",_mmin,_mmax,"GeV/c^{2}");

  // Create a binned dataset that imports contents of TH1 and associates its contents
      to observable 'mass'
  RooDataHist dh("dh","dh",mass,Import(*hpeak));

  // Define background model (exponential) and its parameter
  RooRealVar lambda("lambda","lambda",-0.3,-4.,0.);
  RooExponential background("background", "background", mass, lambda);

  // Define signal model (Gaussian) and its parameters
  RooRealVar mean("mean","mean",0.5*(_mmin+_mmax),_mmin,_mmax);
  RooRealVar sigma("sigma","sigma",0.1*(_mmax-_mmin),0.,0.5*(_mmax-_mmin));
  //RooGaussian signal("signal","signal",mass,mean,sigma);

  RooRealVar alpha("alpha", "alpha", 3.3, 0., 5.);
  RooRealVar n("n", "n",  5.1, 0., 20.);


  RooCBShape crystalball("CBS", "CBS", mass, mean, sigma, alpha, n); //
      -----------------------------------

  RooBreitWigner BW("BW","BW",mass,mean,sigma); //------------------------------------


  // Define variables for number of signal and background events
  double n_signal_initial = 0.8 * dh.sumEntries();
  double n_back_initial   = 0.2 * dh.sumEntries();
  RooRealVar n_signal("n_signal","n_signal",n_signal_initial,0.,dh.sumEntries());
  RooRealVar n_back("n_back","n_back",n_back_initial,0.,dh.sumEntries());

```

```

    RooRealVar frac1("frac1","frac1",0.46);
    RooAddPdf* signal;
    signal      = new RooAddPdf("signal", "signal", RooArgList(crystalball, BW),
        RooArgList(frac1));


    // Sum signal and background models
    RooAddPdf* model = new RooAddPdf("model","model", RooArgList(*signal, background),
        RooArgList(n_signal, n_back));

    // Perform the fit
    model->fitTo(dh) ;

    // Plot data, fitted function and its components in the same frame
    RooPlot* frame = mass.frame();
    frame->SetTitle("#mu^{+}#mu^{-} mass spectrum");

    dh.plotOn(frame,Name("dh"));

    model->plotOn(frame,Name("modelSig"),Components("signal"),LineStyle(kDashed)) ;
    model->plotOn(frame,Name("modelBkg"),Components("background"),LineStyle(kDashed),
        LineColor(kRed)) ;
    model->plotOn(frame,Name("model")) ;

    TCanvas roofit_canvas;
    frame->Draw();

    // draw the legend
    TLegend *legend=new TLegend(0.65,0.6,0.88,0.85);
    legend->SetBorderSize(0);
    legend->SetTextFont(40);
    legend->SetTextSize(0.04);
    legend->AddEntry(frame->findObject("dh"),"Data","lpe");
    legend->AddEntry(frame->findObject("modelBkg"),"Background fit","l");
    legend->AddEntry(frame->findObject("modelSig"),"Signal fit","l");
    legend->AddEntry(frame->findObject("model"),"Global Fit","l");
    legend->Draw();

    // display info + fit results
    TLatex L;
    L.SetNDC();
    L.SetTextSize(0.04);
    L.DrawLatex(0.15,0.8,"Dimuon Spectrum");
    L.SetTextSize(0.03);
    L.DrawLatex(0.15,0.75,"resonance: Z^{0}");
    L.DrawLatex(0.15,0.70,Form("mass: %5.3f #pm %5.3f GeV/c^{2}",
                               mean.getVal(), mean.getError()));
    L.DrawLatex(0.15,0.65,Form("with: %5.3f #pm %5.3f MeV/c^{2}",
                               sigma.getVal()*1000, sigma.getError()*1000));
    L.DrawLatex(0.15,0.60,Form("yield: %.0f #pm %.0f events",
                               n_signal.getVal(), n_signal.getError()));
      L.DrawLatex(0.15,0.55,Form("#Chi^{2}: %2.5f",
              frame->chiSquare()));
    L.DrawLatex(0.15,0.50,"Corte: p_{T}#geq 8 e |#eta|#leq 2,4");



    roofit_canvas.SaveAs(("plots/result_RooFit"+extraString+".png").c_str());
    roofit_canvas.SaveAs(("plots/result_RooFit"+extraString+".pdf").c_str());

}

```

```

void anadimuon::FitPeak()
{

  TCanvas *c0 = new TCanvas("peak","peak",800,600);

  // retrive histogram with selected peak
  TH1F* hpeak= 0;
  TString hname("hDimuonMass_peak");
  if(!_outFile) {cout << "Check input file." << endl; return;}
  _outFile->GetObject(hname,hpeak);
  if (!hpeak) {
    cout << "Check input histogram:" << hname <<  endl;
    return;
  }
  // make it pretty
  hpeak->GetXaxis()->SetTitle("#mu^{+}#mu^{-} invariant mass [GeV/c^{2}]");
  hpeak->GetYaxis()->SetTitle(Form("Events / %3.1f MeV/c^{2}",hpeak->GetBinWidth(1)
      *1000));
  hpeak->SetStats(0);
  hpeak->SetTitle("");
  hpeak->SetMarkerStyle(21);
  hpeak->SetMarkerSize(0.8);


  // define fit function (formula defined in a separate function, see at the end of
      the macro)
  const Int_t nfitpar(5);
  TF1* f = new TF1("f",fitfun,_mmin,_mmax,nfitpar);
  f->SetParameters(0.8*hpeak->GetEntries(),
                   0.5*(_mmin+_mmax),
                   0.1*(_mmax-_mmin),
                   0.2*hpeak->GetEntries(),
                   -0.3);
  f->SetParLimits(0,0,hpeak->GetEntries());
  f->SetParLimits(1,_mmin,_mmax);
  f->SetParLimits(2,0,0.5*(_mmax-_mmin));

  // perform the fit
  hpeak->Fit("f","","ep");

  // write fit results into array
  Double_t par[nfitpar];
  f->GetParameters(par);

  printf("\nFitResults:\n\tResonance mass: %5.3f +/- %5.3f GeV/c^2.\n",
         par[1],f->GetParErrors()[1]);

  // get the individual functions for separate representation
  TF1 *signalFcn = new TF1("signalFcn",signal,_mmin,_mmax,3);
  signalFcn->SetLineColor(kBlue);
  signalFcn->SetNpx(500);
  TF1 *backFcn = new TF1("backFcn",backgr,_mmin,_mmax,2);
  backFcn->SetLineColor(kGray);
  backFcn->SetLineStyle(2);

  signalFcn->SetParameters(par);
  signalFcn->Draw("same");

  backFcn->SetParameters(&par[3]);
  backFcn->Draw("same");

  // draw the legend
```

```
  TLegend *legend=new TLegend(0.7,0.65,0.88,0.85);
  legend->SetBorderSize(0);
  legend->SetTextFont(40);
  legend->SetTextSize(0.03);
  legend->AddEntry(hpeak,"Data","lpe");
  legend->AddEntry(backFcn,"Background fit","l");
  legend->AddEntry(signalFcn,"Signal fit","l");
  legend->AddEntry(f,"Global Fit","l");
  legend->Draw("same");

  // display info + fit results
  TLatex L;
  L.SetNDC();
  L.SetTextSize(0.04);
  L.DrawLatex(0.15,0.8,"Dimuon Spectrum");
  L.SetTextSize(0.03);
  L.DrawLatex(0.15,0.75,"resonance: J/#psi");
  L.DrawLatex(0.15,0.70,Form("mass: %5.3f #pm %5.3f GeV/c^{2}",
                             par[1], f->GetParErrors()[1]));
  L.DrawLatex(0.15,0.65,Form("with: %5.3f #pm %5.3f MeV/c^{2}",
                             par[2]*1000, f->GetParErrors()[2]*1000));
  //L.DrawLatex(0.15,0.60,Form("yield: %.0f #pm %.0f events",
              //             par[0]/hpeak->GetBinWidth(1), f->GetParErrors()[0]/hpeak
      ->GetBinWidth(1)));
  L.DrawLatex(0.15,0.60,"resonance: J/#psi");

  // save the fitted histogram
  c0->SaveAs(("plots/result_ROOT"+extraString+".png").c_str());
  c0->SaveAs(("plots/result_ROOT"+extraString+".pdf").c_str());

}

Double_t signal(Double_t *x, Double_t *par) {
  //Gaussian function
  return par[0]/par[2]/sqrt(2*TMath::Pi())*exp(-0.5*TMath::Power(((x[0]-par[1])/(par
      [2])),2));
}

Double_t backgr(Double_t *x, Double_t *par) {
  //exponential function
  return par[0]*exp(par[1]*x[0]);
}

Double_t fitfun(Double_t *x, Double_t *par) {
  //the total PDF function, sum of the above
  return signal(x,par) + backgr(x,&par[3]);
}
```



Histograma 1: Ressonância do bóson Z^0