Introdução à Análise de dados em FAE **(Data: 28/09/20)**

Relatório RooFit.

Professores: Sandro Fonseca, Sheila Mara, Eliza Melo. *Name:* João Pedro Gomes Pinheiro (`jgomespi`).

Exercício baseado no Tutorial sobre o RooFit.

Todos os exercícios descritos neste relatório estão disponíveis no repositório GitHub em: *https://github.com/jgomespi/RooFit*.

De tal forma que, para executar na sua máquina, basta executar:

```
git clone git@github.com:jgomespi/RooFit.git
cd RooFit
```

Exercício 1:

Vamos setar os parâmetros da gaussiana (x, μ, σ) e, posteriormente, dada por:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

No RooFit, fazemos:

```
// Declaring the variables
RooRealVar x("x", "x", -10, 10);
RooRealVar mean("mean", "mean of gaussian", 0, -10, 10);
RooRealVar sigma("sigma", "width of gaussian", 1, 0.1, 10);
// Declaring the gaussian distribuition
RooGaussian gauss("gauss", "gaussian PDF", x, mean, sigma);
```

Vamos gerar 1.000 pontos aleatórios que obedecem a distribuição de Gauss, fazendo:

```
// Creating 1000 random gauss points
RooDataSet *data1 = gauss.generate(x, 10000);
```

Agora vamos plotar os dados num *frame* e fitar a distribuição gaussiana aos dados:

```
// Declaring a frame
RooPlot *xframe1 = x.frame(Title("Gaussian pdf with data"));
// Plot the data and the gauss function on the frame
data1->plotOn(xframe1);
gauss.plotOn(xframe1);
// Fit the gauss function to the data
gauss.fitTo(*data1);
```

O mesmo procedimento pode ser feito com uma distribuição exponencial, dada por:

$$f(x) = Ae^{\lambda x}$$

Lembrando que o RooFit ajusta a constante de normalização aos dados automaticamente, basta definirmos, além do x já definido, a variável λ, ligada à inclinação da exponencial:

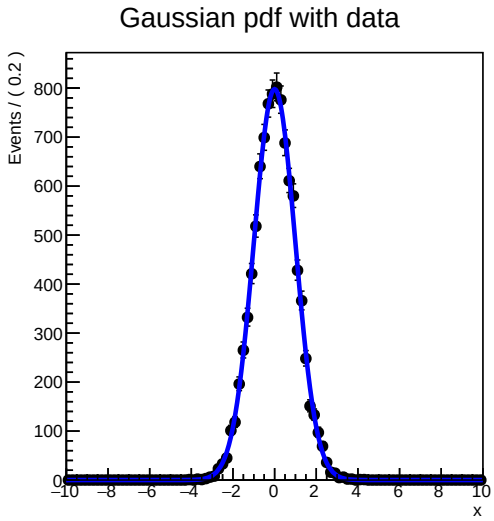
```
// Declaring lambda varible to the expo dist
RooRealVar lambda("lambda", "slope", -0.1, -5., 0.);
```

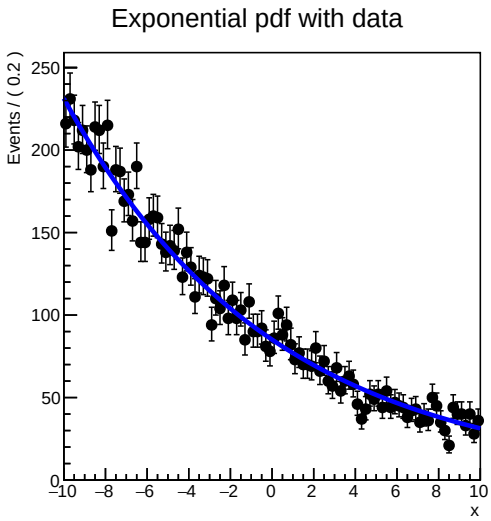
O processo seguinte é muito semelhate ao feito para a gaussiana, ou seja:

```
// Declaring the exponential distribuition
RooExponential expo("expo","exponential PDF", x, lambda);

// Creatring 1000 random exponencial points
RooDataSet *data2 = expo.generate(x, 10000);

```






```
// Declaring another frame
RooPlot *xframe2 = x.frame(Title("Exponential pdf with data"));
// Plot the data and the expo function on the frame
data2->plotOn(xframe2);
expo.plotOn(xframe2);
// Fit the expo dist to the data
expo.fitTo(*data2);
```

É interessante notar que, como setamos um valor inicial negativo para λ, a distribuição de pontos aleatória (e consequentemente a curva ajustada) será uma exponencial decrescente.

Vamos desenhar os plots num Canvas com os comandos a seguir:

```
TCanvas *c = new TCanvas("Exercise_1", "Exercise_1", 800, 400);
c->Divide(2);
c->cd(1);
gPad->SetLeftMargin(0.15);
xframe1->GetYaxis()->SetTitleOffset(1.6);
xframe1->Draw();
c->cd(2);
gPad->SetLeftMargin(0.15);
xframe2->GetYaxis()->SetTitleOffset(1.6);
xframe2->Draw();
```

Os resultados estão na Figura ??:

Este script está no Repositório GitHub e pode ser executado através de:

```
root -l exercise1.C
```

Exercício 2:

Com base no `RooDataSet` chamado `data` presente no arquivo `DataSet_lowstat.root`, vamos ajustar a soma de uma distribuição Crystal Ball, representando a ressonância do J/ψ, com uma distribuição gaussiana, representando a ressonância do $\psi(2S)$ e um polinômio para o fundo. Sabemos que a função Crystal Ball é dada por:

$$f(x) = N \begin{cases} e^{-\dfrac{x-\mu}{2\sigma^2}} & \text{for } \dfrac{x-\mu}{\sigma} > -\alpha \\ A\left(B - \dfrac{x-\mu}{\sigma}\right)^{-n} & \text{for } \dfrac{x-\mu}{\sigma} \leq -\alpha \end{cases}$$

sendo N uma constante de normalização e A e B dados por:

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

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

A distribuição do $\psi(2S)$ deveria, a princípio, obedecer uma Crystal Ball. Mas, como podemos ver pela definição acima, a Crystal Ball é idêntica à distribuição gaussiana do seu lado direito. Como logo ao lado esquerdo do $\psi(2S)$ temos a ressonância do J/ψ, podemos considerar, para o $\psi(2S)$, uma distribuição gaussiana.

Inicialmente, declaramos as PDFs, os parâmetros das PDFs e os atribuímos a elas:

```
// Declaring the mass variable
RooRealVar mass("mass", "mass", 2., 6.);

// Parameters of Crystall-Ball of J/Psi
RooRealVar meanJpsi("meanJpsi", "Mean of J/#psi", 3.1, 2.8, 3.2);
RooRealVar sigmaJpsi("sigmaJpsi", "Sigma of J/#psi", 0.3, 0.0001, 1.);
RooRealVar alphaJpsi("alphaJpsi", "Alpha of J/#psi", 1.5,-5.,5.);
RooRealVar nJpsi("nJpsi","n of J/#psi", 1.5,0.5,5.);

// Declaring the Crystal Ball PDF for JPsi
RooCBShape CB_Jpsi( "CB_Jpsi","The J/#psi Crystall Ball", mass , meanJpsi , sigmaJpsi
    , alphaJpsi , nJpsi );

// For Upsilon(2S), we will user the same sigma value of JPsi
RooRealVar meanPsi("meanPsi", "Mean of #psi(2S)", 3.7, 3.6, 3.8);
RooRealVar sigmaPsi("sigmaPsi", "Sigma of #psi(2S)", 0.3,0.0001, 1.);

// Declaring the Gauss PDF for Psi
RooGaussian gaussPsi("gaussPsi", "The #psi(2S) gaussian PDF", mass, meanPsi, sigmaPsi
    );

// Declaring the polinomial parameters and PDF for background
RooRealVar a1("a1", "a1", -0.7, -2.,2.);
RooRealVar a2("a2", "a2", 0.3, -2.,2.);
RooRealVar a3("a3", "a3", -0.03, -2.,2.);

// Declaring the polinomial PDF for background
RooPolynomial PolBG("PolBG", "Polinomial PDF for BG", mass, RooArgList(a1,a2,a3));
```

Posteriormente, declaramos os números esperados de eventos (J/ψ, $\psi(2S)$ e background), que serão importantes no ponderamento do ajuste:

```
// Declaring the widths, which is the number of each event (JPsi, Psi2S and BG):
RooRealVar NJpsi("NJpsi", "Number of J/#psi events", 1500., 0.1, 10000.);
RooRealVar NPsi("NPsi", "Number of #psi (2S) events", 100., 0.1, 5000.);
RooRealVar NBG("NBG", "Number of background events",  5000., 0.1, 50000.);
```

Então, fazemos a soma dos PDFs e o ajustamos ao DataSet presente no arquivo:

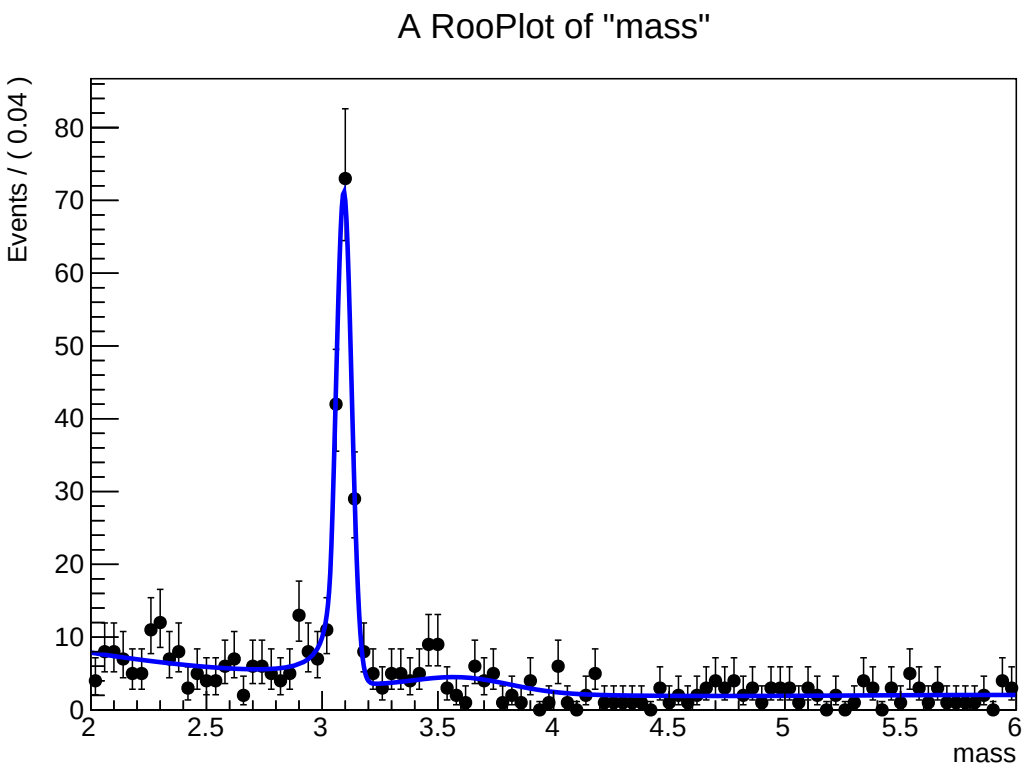
```
RooAddPdf sumPDF("sumPDF", "Sum of PDFs", RooArgList(CB_Jpsi,gaussPsi,PolBG),
    RooArgList(NJpsi,NPsi,NBG));
sumPDF.fitTo(*data);
```

Desenhamos a distribuição ajustada e normalizada sobreposta aos dados e salvamos num .pdf:

```
// Draw a frame with the data and the sum of PDF normalized
RooPlot *xframe = mass.frame();
data->plotOn(xframe);
sumPDF.plotOn(xframe,Normalization(1.0,RooAbsReal::RelativeExpected));
xframe->Draw();
xframe->SaveAs("results.pdf","pdf");
```

O resultado esperado está na Figura ??.

Finalmente, escrevemos num Workspace e salvamos o Workspace num arquivo .root:




```
RooWorkspace w("w");
w.import(*data);
w.import(sumPDF);
w.writeToFile("Results.root");
```

As instruções acima estão numa macro chamada `exercise2.C` que pode ser executada abrindo uma sessão root e inserindo os seguintes comandos:

```
TFile *f = new TFile("DataSet_lowstat.root");
.x exercise2.C
```