Introdução à Análise de dados em FAE **(05/09/2024)**

Exercicios de estatística para análise de dados em HEP

Professores: Eliza Melo, Dilson Damião e Mauricio Thiel *Name:* Jorge Júlio Barreiros Venuto de Siqueira

EXERCÍCIO 1

Para realizar o que é pedido no exercício 1, foi realizado o seguinte código em C:

Código em C

```
void exercicio_1() {

    RooRealVar x("x", "x", -10, 10);
    RooRealVar mean("mean", "mean", 0, -5, 5);
    RooRealVar sigma("sigma", "sigma", 1, 0.1, 5);
    RooRealVar alpha("alpha", "alpha", 1, 0.1, 5);
    RooRealVar n("n", "n", 1, 0, 10);

    RooCrystalBall cb("cb", "Crystal Ball PDF", x, mean, sigma, alpha, n);
    RooDataSet* data = cb.generate(x, 1000);
    RooFitResult* fitResult = cb.fitTo(*data, RooFit::Save());

    RooPlot* xframe = x.frame();
    data->plotOn(xframe);
    cb.plotOn(xframe);
   xframe->SetTitle("Ajuste da PDF Crystal Ball");
    xframe->GetXaxis()->SetTitle("x");
    xframe->GetYaxis()->SetTitle("Frequencia");

    TCanvas* c = new TCanvas("c", "Ajuste Crystal Ball", 800, 600);
    xframe->Draw();
    TLegend* legend = new TLegend(0.6, 0.7, 0.89, 0.89);
    legend->SetTextSize(0.07);
    legend->AddEntry(xframe->getObject(0), "Dados", "p");
    legend->AddEntry(xframe->getObject(1), "Ajuste", "l");
    legend->Draw();
    TLatex* latex = new TLatex();
    latex->SetNDC();
    latex->SetTextSize(0.05);

    double chi2 = xframe->chiSquare();
    int ndf = data->numEntries() - fitResult->floatParsFinal().getSize();
    latex->DrawLatex(0.14, 0.85, Form("Mean: %.3f #pm %.3f", mean.getVal(), mean.getError()));
    latex->DrawLatex(0.14, 0.8,
    Form("Sigma: %.3f #pm %.3f", sigma.getVal(), sigma.getError()));
    latex->DrawLatex(0.14, 0.75,
    Form("Alpha: %.3f #pm %.3f", alpha.getVal(), alpha.getError()));
    latex->DrawLatex(0.14, 0.7,
    Form("n: %.3f #pm %.3f", n.getVal(), n.getError()));
    latex->DrawLatex(0.14, 0.65,
    Form("#chi^{2}/ndf: %.2f/%d", chi2, ndf));
    c->SaveAs("fitCrystalBall.png"); // Salva o gráfico como PNG
}
```

Após isso, foi compilado utilizando o *ROOT*, com o comando

```
.x exercicio_1.C
```

resultando no que foi pedido:

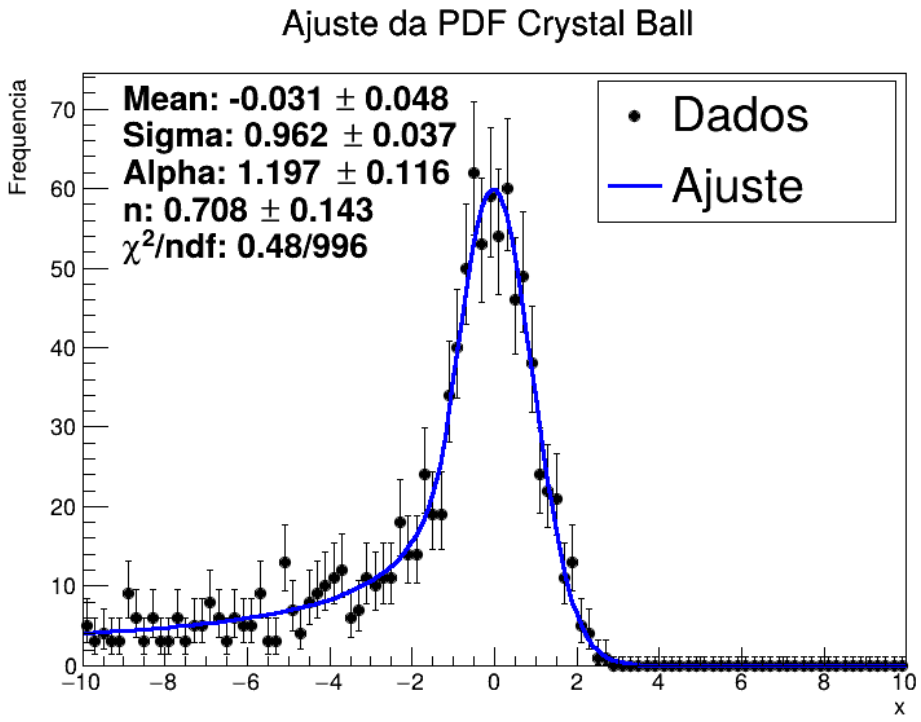

Figura 1: Resultado do exercício 1

Parâmetros da *Crystal Ball*:

- *mean*: Representa o valor médio (centro) da distribuição de massa para a ressonância, ou seja, a posição do pico. Para uma distribuição associada à ressonância J/, é esperado que o valor esteja próximo de 3.1 GeV/c^2.
- *sigma*: É o desvio padrão da parte gaussiana da distribuição, controlando a largura do pico em torno do valor médio. Um *sigma* maior significa uma largura maior do pico.
- *alfa*: Controla a assimetria da distribuição na *Crystal Ball*. Para valores positivos, a cauda estará à direita (lado alto), enquanto valores negativos fazem a cauda se estender para a esquerda (lado baixo).
- *n*: Define o decaimento da cauda da distribuição. Valores mais altos de *n* tornam a transição da parte gaussiana para a cauda mais suave, enquanto valores menores tornam a transição mais brusca.

Seus resultados estão expressos no gráfico da Figura 3.
Além disso, também foi calculado o valor de *chi²/ndf*, que quanto mais próximo de 1, indica o quão bem o ajuste descreve a distribuição dos dados.

EXERCÍCIO 2

Código em C

```
#include <iostream>
#include <RooRealVar.h>
#include <RooDataSet.h>
#include <RooExponential.h>
#include <RooFitResult.h>
#include <RooPlot.h>
#include <RooRandom.h>
#include <TCanvas.h>
#include <TLatex.h>

void exercicio_2() {
    RooRealVar x("x", "x", 0, 10);
    RooRealVar lambda("lambda", "decay constant", 1, 0.1, 2);
    RooExponential expDecay("expDecay", "Exponential Decay", x, lambda);
    RooDataSet* data = expDecay.generate(x, 1500);
    RooRealVar nEvents("nEvents", "Number of Events", 1500, 100, 5000);
    RooFitResult* fitResult = expDecay.fitTo(*data, RooFit::Save(), RooFit::Extended(kTRUE));
    RooPlot* xframe = x.frame();
    data->plotOn(xframe);
    expDecay.plotOn(xframe);
    xframe->GetXaxis()->SetTitle("x");
    xframe->GetYaxis()->SetTitle("Frequencia");

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

    xframe->Draw();
    TLatex latex;


    latex.SetNDC();
    latex.SetTextSize(0.03);
    latex.DrawLatex(0.2, 0.8, Form("Ajuste de \\lambda: %.3f", lambda.getVal()));

    latex.DrawLatex(0.2, 0.75, Form("Total de Eventos Ajustados: %.0f", nEvents.getVal()));
    c->SaveAs("fitExponential.png");
    std::cout << "Valor ajustado para lambda: " << lambda.getVal() << std::endl;
    std::cout << "Número total de eventos ajustados: " << nEvents.getVal() << std::endl;
    double trueLambda = 1;
    double trueEvents = 1500;

    std::cout << "Valor gerado para lambda: " << trueLambda << std::endl;

    std::cout << "Número total de eventos gerados: " << trueEvents << std::endl;

    std::cout
}
```

Após isso, foi compilado utilizando o Root, com o comando:

```
.x exercicio_2.C
```

resultando o que foi pedido.
Os resultados dos ajustes foram:

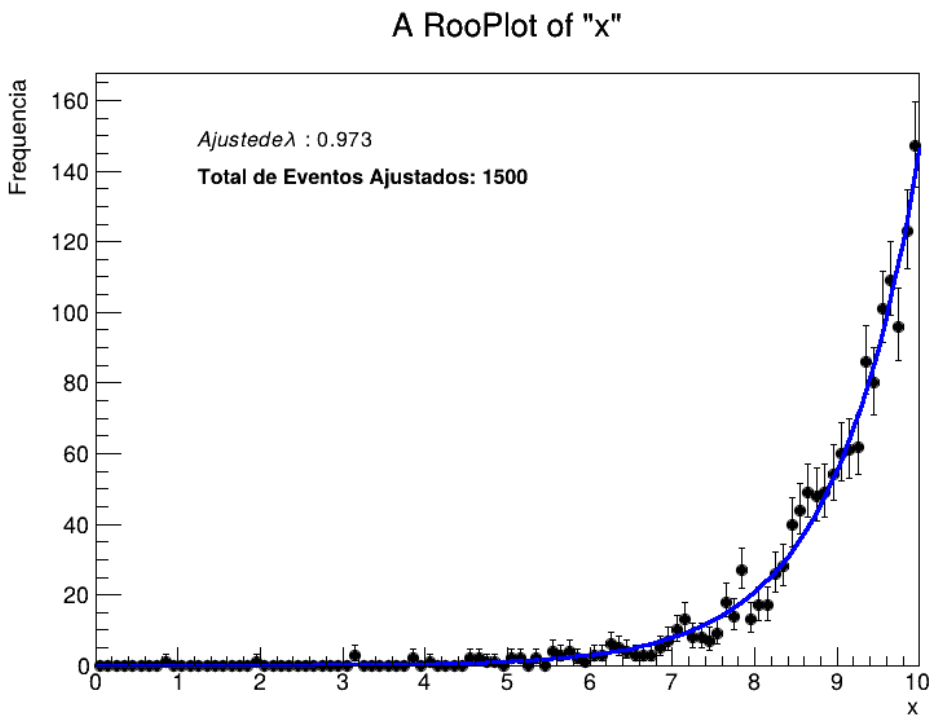

Figura 2: Resultado do exercício 2

- $\lambda = 0{,}97$
- Número total de eventos ajustados = 1500

Pode-se afirmar que os valores ajustados estão dentro da expectativa, visto que $\lambda < 1$.

EXERCICIO 3

Para realizar o exercício 3, primeiramente foi necessário baixar e entender como estavam dispostos os dados que seriam analizados.

Para isso, foi criado um código que iria identificar o conteúdo do arquivo.

Código em C para ler o arquivo

```
#include <iostream>
#include "TFile.h"
#include "TTree.h"

void listar_variaveis_v2() {
    TFile *file = TFile::Open("DataSet_lowstat.root");
    if (!file || file->IsZombie()) {
        std::cerr std::endl;
        return;
    }

    TTree *tree = (TTree*)file->Get("data");
    std::endl;
        return;
    }
    tree->Print();
    file->Close();
}
```

Feito isso, foi possível acessar o banco de dados e realizar o ajuste de uma crystalBall sobre os dados.

Código em C

```
void exercicio_3() {

    TFile *file = TFile::Open("DataSet_lowstat.root");
    RooDataSet *data = (RooDataSet*)file->Get("data");

    RooRealVar mass("mass", "Massa [GeV/c^{2}]", 2, 5.5);

    RooRealVar mean("mean", "mean", 3.1, 2.8, 3.2);
    RooRealVar sigma("sigma", "sigma", 0.3, 0.0001, 1.);
    RooRealVar alfa("alfa", "alfa", 1.5, -5., 5.);
    RooRealVar n("n", "n", 1.5, 0.5, 5.);
    RooCBShape CB("CB", "CB", mass, mean, sigma, alfa, n);

    RooRealVar a1("a1", "a1", -0.7, -2., 2.);
    RooRealVar a2("a2", "a2", 0.3, -2., 2.);
    RooRealVar a3("a3", "a3", -0.03, -2., 2.);
    RooPolynomial background("background",

    "The background PDF", mass, RooArgList(a1, a2, a3));

    RooRealVar frac("frac", "frac", 0.5, 0.0, 1.0);
    RooAddPdf model("model", "Modelo Sinal + Fundo",
    RooArgList(CB, background), RooArgList(frac));

    RooFitResult *fitResult = model.fitTo(*data);

    RooPlot *frame = mass.frame();    data->plotOn(frame);
    model.plotOn(frame);
    model.paramOn(frame);

    double chi2 = frame->chiSquare();
    std::cout << "chi2 / ndf = " << chi2 << std::endl;

    TCanvas *c = new TCanvas("c", "Ajuste JPsi", 800, 600);
    frame->Draw();

   TLatex chi2Text;    chi2Text.SetNDC();  // Define para
    chi2Text.SetTextSize(0.03);
    chi2Text.DrawLatex(0.15, 0.85, Form("#chi^{2}/ndf = %.2f", chi2));
    c->SaveAs("fit_result.png");
    file->Close();
}
```

Após isso, foi compilado utilizando o Root, com o comando

```
.x exercicio_3.C
```

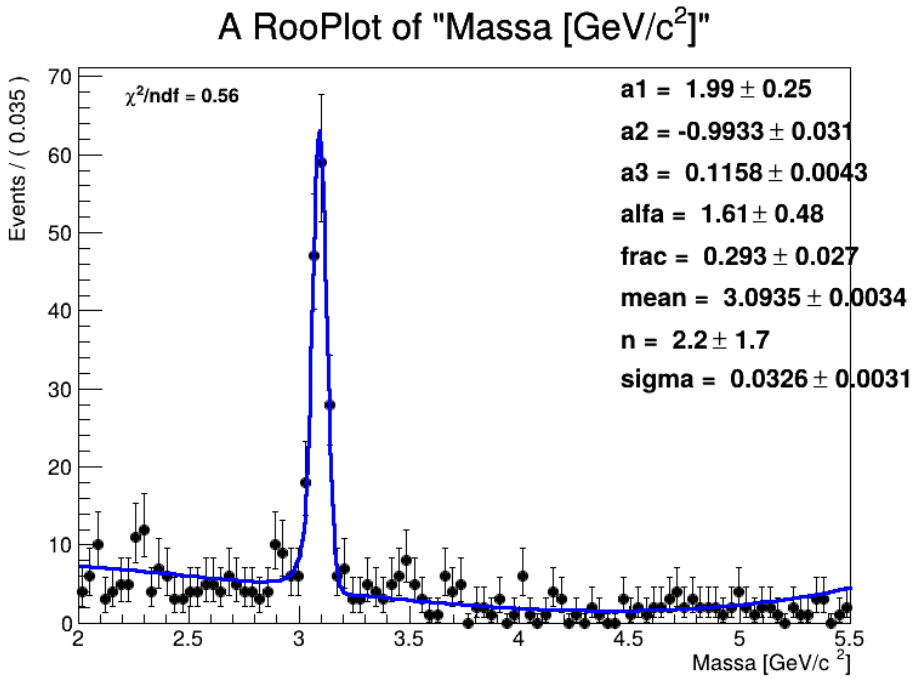

Figura 3: Resultado do exercício 3

Os parâmetros que foram ajustados:

- Parâmetros da *Crystal Ball*:
  - *mean*: Representa o valor médio (centro) da distribuição de massa para a ressonância, ou seja, a posição do pico. Para uma distribuição associada à ressonância J/ψ.
  - *sigma*: É o desvio padrão da parte gaussiana da distribuição, controlando a largura do pico em torno do valor médio. Um *sigma* maior significa uma largura maior do pico.
  - *alfa*: Controla a assimetria da distribuição na *Crystal Ball*. Para valores positivos, a cauda estará à direita (lado alto), enquanto valores negativos fazem a cauda se estender para a esquerda (lado baixo).
  - n: Define o decaimento da cauda da distribuição. Valores mais altos de n tornam a transição da parte gaussiana para a cauda mais suave, enquanto valores menores tornam a transição mais brusca.
- Parâmetros do Fundo (*a1*, *a2*, *a3*):
  - *a1*, *a2*, *a3*: Estes são coeficientes de um polinômio de ordem 2, representado pela função *RooPolynomial*, que modela o fundo (ou *background*) da distribuição de massa.
  - No código, a distribuição de fundo é uma função polinomial, o que significa que *a1*, *a2*, e *a3* são os coeficientes que determinam a forma do fundo, capturando variações na distribuição de massa que não correspondem ao pico do sinal do J/ψ, mas que aparecem como ruído ou fundo subjacente.
- *frac*:
  frac é o parâmetro de fração que define a combinação entre o sinal (*Crystal Ball*) e o fundo (polinômio). Valores de *frac* entre 0 e 1 controlam a proporção de sinal e fundo, onde *frac* próximo de 1 indica predominância de sinal e valores próximos de 0 indicam predominância de fundo.

Segundo o *Particle Data Group* (PDG), o valor da massa invariante do J/ψ é dado por:

$$M_{J/\psi}^{\mathrm{PDG}} = 3{,}0969 \pm 0{,}006\,\mathrm{GeV/c^2}.$$

No ajuste realizado, o valor da massa invariante do J/ψ obtido foi:

$$M_{J/\psi}^{\text{ajuste}} = 3{,}0935 \pm 0{,}0034\,\text{GeV/c}^2.$$

Para avaliar a compatibilidade entre esses valores, calcula-se o parâmetro de compatibilidade C da seguinte forma:

$$C = \frac{|3{,}0969 - 3{,}0935|}{\sqrt{(0{,}006)^2 + (0{,}0034)^2}} = 0{,}27\,\sigma.$$

Esse resultado indica que o valor da massa invariante determinado pelo ajuste é compatível com o valor do PDG, uma vez que o valor de compatibilidade, $0{,}27\,\sigma$, é menor que $2\,\sigma$. Assim, nosso ajuste para a massa do J/ψ está em excelente concordância com o valor de referência, reforçando a precisão do modelo utilizado.