Universidade do Estado do Rio de Janeiro

Jully Moura

Atividade 4

Rio de Janeiro
2024

O que é o RooFit?

RooFit é uma biblioteca de software projetada para realizar análise estatística de dados experimentais, pricipalmente em física de partículas. O RooFit tem integração com outras bibliotecas e ferramentas para ajustar curvas, manejar dados complexos e visualizar resultados, sendo ideal para modelagem de distribuições de probabilidade e para extração de informações relevantes a partir de grandes volumes de dados. É uma biblioteca muito utilizada para colaborações do CERN.

Neste trabalho serão realizados 3 exercícios utilizando o RooFit e suas bibliotecas integradas para demonstrar o que aprendemos em sala de aula, sendo dois exercícios para fazer a análise de dados gerados aleatóriamente e um exercício para fazer a análise dos dados disponibilizados no site: `https://cernbox.cern.ch/index.php/s/DInqlmV9W52WPvY`

Exercício 01

Para realizar este exercício foi criado o código a seguir:

```
//Importando bibliotecas
#include <iostream>
#include <RooRealVar.h>
#include <RooPlot.h>
#include <RooFitResult.h>
#include <RooFit.h>
#include <RooRandom.h>
#include <RooDataSet.h>
#include <TCanvas.h>
#include <RooArgSet.h>
#include <RooCrystalBall.h>
#include <TPaveText.h>
#include <TLegend.h>

//Criando funcao
void atv1() {

    //Definindo x
    RooRealVar x("x", "x", -10, 10);

    //Definindo o modelo Crystal Ball
    RooRealVar media("media", "media", 0, -10, 10);
    RooRealVar sigma("sigma", "sigma", 1, 0.1, 10);
    RooRealVar alfa("alfa", "alfa", 1, 0.1, 10);
```

```
25      RooRealVar n("n", "n", 1, 0, 15);
26      RooCrystalBall crystalball("crystalball", "CrystalBall", x, media, sigma, alfa, n);
27
28      //Gerando dados
29      RooDataSet* dados = crystalball.generate(RooArgSet(x), 1000);
30
31      //Ajustando os dados com o modelo
32      RooFitResult* resultado = crystalball.fitTo(*dados, RooFit::Save());
33
34      //Criando um frame
35      RooPlot* frame = x.frame();
36
37      //Plotando os dados e o modelo no frame
38      dados->plotOn(frame);
39      crystalball.plotOn(frame);
40
41      //Criando um canvas
42      TCanvas* canvas = new TCanvas("canvas", "Eventos gerados com CrystalBall", 800, 600);
43
44      //Desenhando o frame
45      frame->Draw();
46
47      //Adicionando informacoes no grafico
48      TLatex *latex = new TLatex();
49
50      latex->SetNDC();
51      latex->SetTextSize(0.03);
52
53      latex->DrawLatex(0.175, 0.70, Form("Eventos: %d", (int)dados->numEntries()));
54      latex->DrawLatex(0.175, 0.66, Form("n: %.3f", n.getVal()));
55      latex->DrawLatex(0.175, 0.62, Form("Alfa: %.3f", alfa.getVal()));
56      latex->DrawLatex(0.175, 0.58, Form("Media: %.3f", media.getVal()));
57      latex->DrawLatex(0.175, 0.54, Form("Desvio Padrao: %.3f", sigma.getVal()));
58
59      //Salvando a imagem gerada
60      canvas->SaveAs("CrystalballEventos.png");
61  }
62
63  int main() {
64      //Executando a funcao
65      atv1();
66      return 0;
67  }
```

Para começar a confecção deste código, primeiro importou-se as bibliotecas necessárias para a execução do código utilizando **#Include** <>.

Então foi feita uma função **void{}** nomeada de **atv1()**, onde primeiro se definiu o x e os parâmetros para a criação do modelo Crystal Ball utilizando o comando **RooRealVar Variavel("Variável","Titulo da Variável",valor inicial,limite inferior,limite superior**).

Após a definição de váriáveis/parâmetro, foi criada a pdf Crystal Ball com o comando RooCrystal-Ball aplicando os parâmetros definidos anteriormente **RooCrystalBall crystalball("crystalball", "CrystalBall", x, media, sigma, alfa, n)**

Com o modelo definido, utilizou-se o **RooDataset*** para gerar mil eventos e o **RooFitResult*** para ajustar os dados gerados com o modelo Crystal Ball.

OBS: O **RooArgSet** permite agrupar diferentes variáveis, parâmetros e funções de forma organizada.

Para gerar o gráfico do resultado, primeiro foi criado um frame utilizando o **RooPlot*** e feito o plot dos dados e do modelo no frame com o comando **plot0n()**. É possível usar o plotOn() com o operador seta"->", que é usado para acessar membros ou métodos de um objeto através de um ponteiro, e com o operador ponto "**.**" que é usado para acessar membros ou métodos de um objeto diretamente.

Em seguida criou-se um canvas através do **TCanvas*** numa proporção de (800x600)px e foi desenhado o frame utilizando **Draw().**

A legenda foi adicionada criando um **TLatex**, depois que o TLatex foi chamado e nomeado de "latex", o método **SetNDC()** foi usado para definir o sistema de coordenadas do texto, o **SetTextSize** para definr o tamanho do texto e o DrawLatex para desenhar o texto em certa coordenada do gráfico.

OBS: No DrawLatex dos Eventos, foi utilizado **"%d"** e **(int)** pois o número de eventos era inteiro.

Para finalizar a **void atv1()**, a imagem gerada foi salva utilizado o **SaveAs()** e foi criada uma **int main()** para executar a função **atv1()**

Resultados

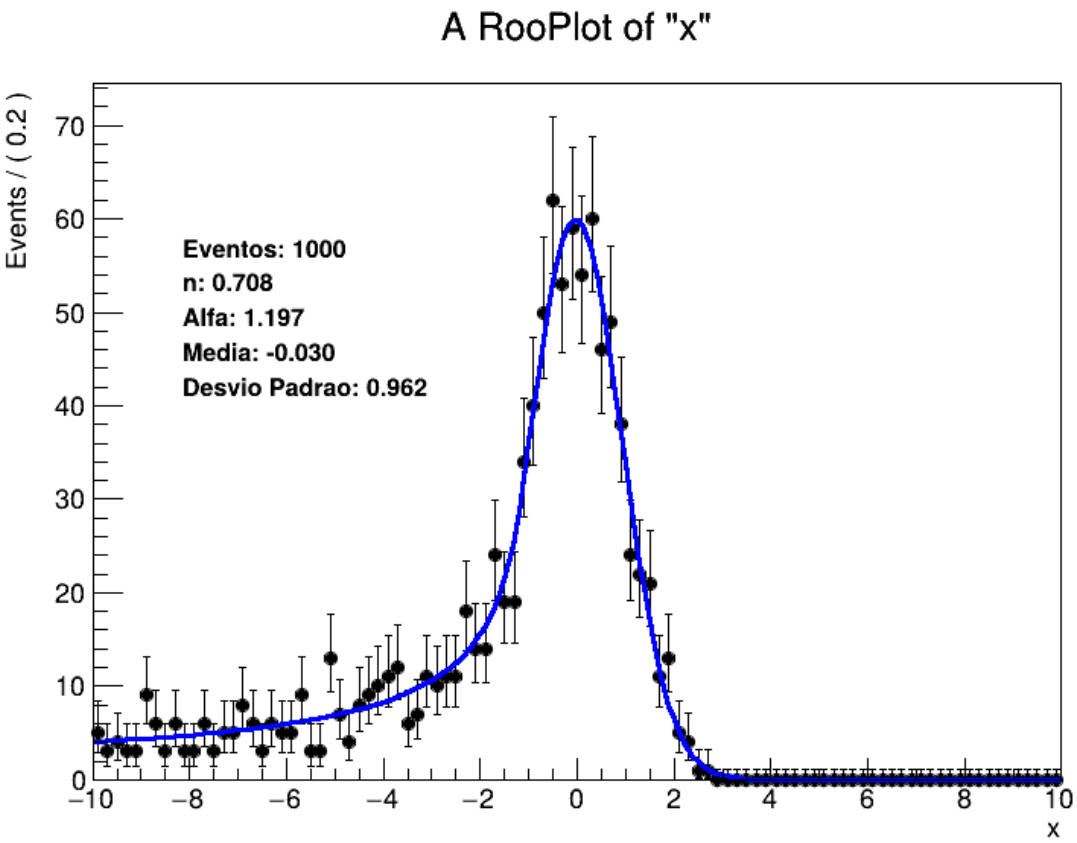

Figure 1: CrystalBallEventos

Exercício 02

Para realizar o exercício 2 o código criado é o que vemos abaixo:

```
//Importando bibliotecas
#include <RooRealVar.h>
#include <RooDataSet.h>
#include <RooExponential.h>
#include <RooFitResult.h>
#include <RooPlot.h>
#include <RooRandom.h>
#include <TCanvas.h>
#include <TLatex.h>
#include <cstdio>
#include <TPaveText.h>
#include <TLegend.h>

//Criando funcao
void atv2() {

    //Definindo x
    RooRealVar x("x", "x", 0, 10);

    //Definindo a taxa de decaimento
    RooRealVar lambda("lambda", "Taxa de decaimento", 1, 0.1, 2);

    //Definindo o numero de eventos
    RooRealVar Eventos("Eventos", "N mero de Eventos", 1500, 100, 5000);

    // Definindo modelo de funcao exponencial decrescente
    RooExponential expDecay("expDecay", "Decaimento Exponencial", x, lambda);

    // Gerando eventos
    RooDataSet* dados = expDecay.generate(RooArgSet(x), 1500);

    // Ajustando os dados com o modelo
    RooFitResult* resultado = expDecay.fitTo(*dados, RooFit::Save(), RooFit::Extended(
        kTRUE));

    // Criando um frame
    RooPlot* frame = x.frame();

    //Plorando os dados e o modelo no frame
    dados->plotOn(frame);
```

```
40      expDecay.plotOn(frame);
41
42      // Criando um canvas
43      TCanvas* canvas = new TCanvas("canvas", "Ajuste Exponencial", 800, 600);
44
45      //Dezenhando o frame no canvas
46      frame->Draw();
47
48      //Adicionando os valores de lambda e de eventos ao grafico
49      TLatex *latex = new TLatex();
50
51      latex->SetNDC();
52      latex->SetTextSize(0.03);
53      latex->DrawLatex(0.175, 0.70, Form("lambda: %.3f", lambda.getVal()));
54      latex->DrawLatex(0.175, 0.66, Form("Eventos: %.0f", Eventos.getVal()));
55
56      // Salvando a imagem gerada
57      canvas->SaveAs("Exponencial.png");
58  }
59
60
61  int main() {
62      //Executando a fun   o
63      atv2();
64      return 0;
65  }
```

Para começar o código, primeiro importou-se as bibliotecas necessárias para a execução do código utilizando **#Include** <>.

Então foi feita uma função **void{}** nomeada como **atv2()**, onde primeiro se definiu o x, o lambda, que é a taxa de decaímento, e o numero de eventos com o comando **RooRealVar**.

A partir do **RooExponential** foi criado o modelo de função exponencial utilizando as variáveis definidas anteriormente.

Com o modelo definido, foram gerados 1500 eventos utilizando o **RooDataSet*** e feito o ajuste com o modelo exponencial utilizando o **RooFitResult***.

Para gerar o gráfico do resultado foi criado um frame utilizando o **RooPlot*** e feito o plot dos dados e do modelo no frame com o comando **plot0n()**.

Com isso, um canvas foi gerado através do **TCanvas*** numa proporção de (800x600)px e foi desenhado o frame utilizando **Draw().**

A legenda foi adicionada criando um **TLatex**, depois que o TLatex foi chamado e nomeado de "latex", o método **SetNDC()** foi usado para definir o sistema de coordenadas do texto que foi desenhado, o **SetTextSize** para definr o tamanho do texto e o DrawLatex para desenhar o texto em certa coordenada do gráfico.

Para finalizar a **void atv2()**, a imagem gerada foi salva utilizado o **SaveAs()** e foi criada uma **int main()** para executar a função **atv2()**

Resultados

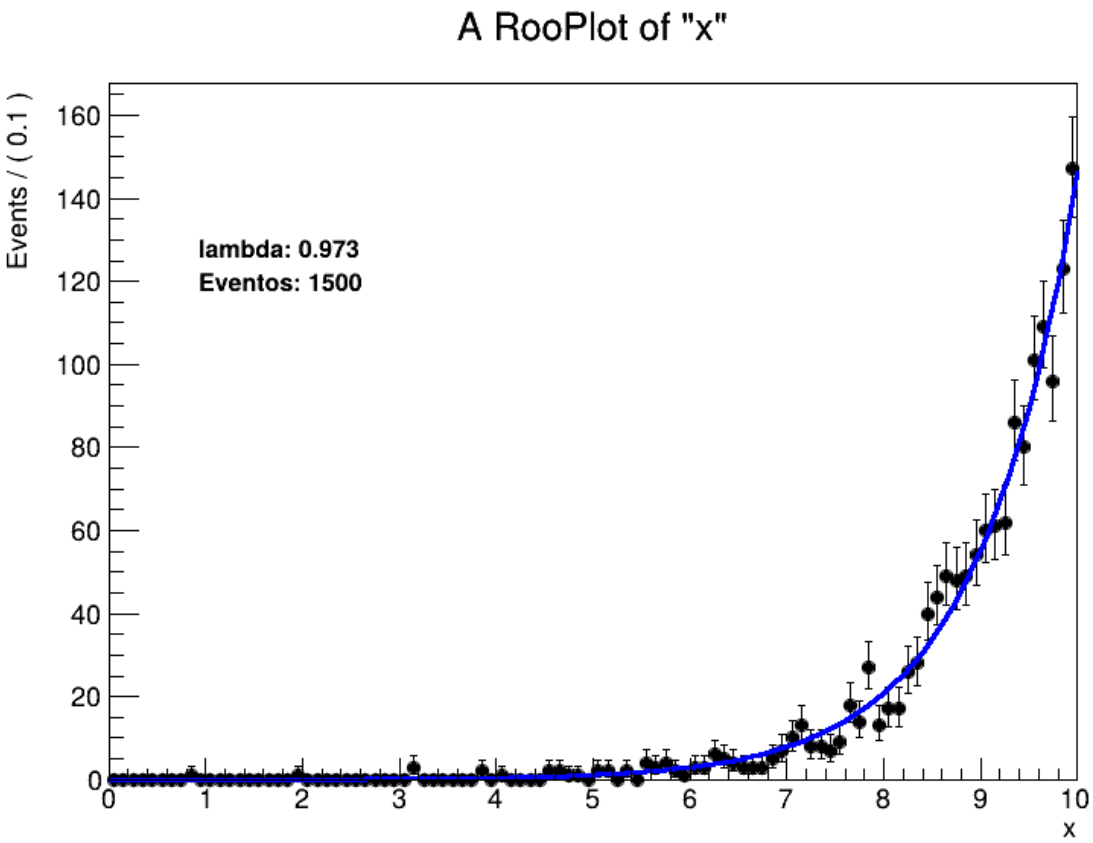

Figure 2: Exponencial

$\lambda = 0.973$

Eventos ajustados = 1500

Como $0.973 < 1$, está dentro da expectativa.

Exercício 03

Para realizar o exercício 3, foi baixado o arquivo "DataSet_lowstat.root" disponibilizado no site `https://cernbox.cern.ch/index.php/s/DInqlmV9W52WPvY`

```
//Importando bibliotecas
#include <iostream>
#include <RooRealVar.h>
#include <RooPlot.h>
#include <RooFitResult.h>
#include <RooFit.h>
#include <RooRandom.h>
#include <RooDataSet.h>
#include <TCanvas.h>
#include <RooArgSet.h>
#include <RooCrystalBall.h>
#include <TPaveText.h>
#include <TLegend.h>
#include <stdio.h>
#include <stdlib.h>
#include <TCanvas.h>
#include <TH1F.h>
#include <TF1.h>
#include <TFile.h>
#include<RooAddPdf.h>
#include <TLatex.h>

//Criando funcao
void atv3(){

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

    //Definindo a variavel de massa
    RooRealVar mass("mass", "Massa [GeV/c^2]", 2, 6);

     //Numero de eventos de sinal e de fundo esperados
    RooRealVar nsig("nsig","Numero de eventos de sinal", 500,0,1000);
    RooRealVar nbkg("nbkg","Numero de eventos de fundo", 500,0,1000);

     //Definindo Pdf de sinal (CrystallBall)
    RooRealVar media("media", "media", 3.1, 2.9, 3.3);
    RooRealVar sigma("sigma", "sigma", 0.4, 0.0001, 1.5);
```

```
40      RooRealVar alfa("alfa", "alfa",1.5, -5., 6);
41      RooRealVar n("n", "n", 1.2, 0.1, 4);
42      RooCrystalBall crystallball("crystallball", "Sinal", mass, media, sigma, alfa, n);
43
44      //Definindo pdf de fundo (Polynomial)
45      RooRealVar a1("a1", "a1", -0.5, -4., 4.);
46      RooRealVar a2("a2", "a2", 0.5, -4., 4.);
47      RooRealVar a3("a3", "a3", -0.5, -4., 4.);
48      RooPolynomial pol("polynomial", "Fundo", mass, RooArgList(a1, a2, a3));
49
50      //Definindo modelo estendido combinando sinal e fundo
51      RooAddPdf model("modelo", "Sinal + Fundo",RooArgList(crystallball, pol), RooArgList(
            nsig,nbkg));
52
53      //Ajustando os dados com o modelo
54      RooFitResult* result = model.fitTo(*data, RooFit::Save(), RooFit::Extended());
55
56      //Criando um Canvas
57      TCanvas *canvas = new TCanvas("canvas","canvas",1200,600);
58
59      //Criando um frame
60      RooPlot* frame = mass.frame();
61
62      //Plotando os dados e o modelo no frame
63      data->plotOn(frame);
64      model.plotOn(frame);
65
66      //Adicionando os par metros do modelo ajustado ao grafico
67      model.paramOn(frame);
68
69      //Desenhando o frame
70      frame->Draw();
71
72      //Criando X^2/ndf
73      double chi2 = frame->chiSquare();
74      int ndf = data->numEntries() - result->floatParsFinal().getSize();
75      double chi2_ndf = chi2 / ndf;
76
77      //Adicionando X^2/ndf ao gr fico
78      TLatex *latex = new TLatex();
79
80      latex->SetNDC();
81      latex->SetTextSize(0.03);
82      latex->DrawLatex(0.175, 0.70, Form("X^2/ndf: %.7f", chi2_ndf));
```

```
83
84      //Salvando a imagem gerada
85      canvas->SaveAs("jpsi.png");
86
87      }
88  int main() {
89      // Executando a funcao
90      atv3();
91      return 0;
92  }
```

O código foi iniciado importando bibliotecas necessárias para a execução dele utilizando o comando **#Include <>**.

Como neste exercício vamos analisar dados do arquivo "DataSet_lowstat.root", antes de começar a função **void()**, utilizou-se o comando **Tfile** para abrir o arquivo e o **RooDataSet*** para pegar os dados "data" do aqruivo aberto. **OBS:**Usei os nomes em inglês para bater com o que estava no documento.

Então se iniciou a função **void{}** nomeada como **atv3()**, onde primeiro se definiu a variável "mass" e o número de eventos de sinal e de fundo esperados utilizando o **RooRealVar**. Neste código a organização ficou um pouco diferente, para que não ficasse mais organizado, os parâmetros/variáveis não foram definidos todos juntos em um único passo. Então, após definir mass, a pdf de sinal (Crystal Ball) foi definida utilizando **RooRealVar** para os parâmetros e **RooCrystalBall** para criar o modelo.

Para definir a pdf de fundo(Polynomial) foi utilizado o comando **RooRealVar** para os parâmetros e o **RooPolynomial** para criar o modelo.

Com a pdf de sinal(CrystalBall) e a pdf de fundo(Polynomial) definidas, criou-se o modelo estendido combinando as duas pdfs utilizando o comando **RooAddPdf**.

Com o modelo pronto, foi feito o ajuste dos dados com o modelo estendido(CrystalBall + Polynomial) através do **RooFitResult***.

Para gerar o gráfico, um canvas foi criado através do **TCanvas*** numa proporção de (1200x600)px, para que o texto com os dados ficasse dentro dos limites do gráfico, e um frame relacionado a massa foi gerado através do **RooPlot***.

Tendo criado o frame, os dados e o modelo foram plotados no frame com o comando **plotOn()**.

Para adicionar os parâmentros do modelo ajustado ao gráfico como pedido no exercício, foi utilizado o comando **paramOn()** e, logo em seguida, o frame foi "desenhado" com o comando **Draw()**.

Este exercício solicita que se faça o teste estatístico no ajuste calculando o X^2/ndf e que o exiba no gráfico, para calcular foram utilizados:

- double chi2: Para achar valor de X^2

- int ndf: Para achar valor de ndf

- double chi2/ndf : Para acahar valor de X^2/ndf

Já para exibir o valor no gráfico, foi criado um **TLatex** e nomeado de "latex", o método **SetNDC()** foi usado para definir o sistema de coordenadas do texto que foi desenhado, o **SetTextSize** para definir o tamanho do texto e o DrawLatex para desenhar o texto em certa coordenada do gráfico.

Para finalizar a **void atv3()**, a imagem gerada foi salva utilizado o **SaveAs()** e foi criada uma **int main()** para executar a função **atv3()**.

Resultados

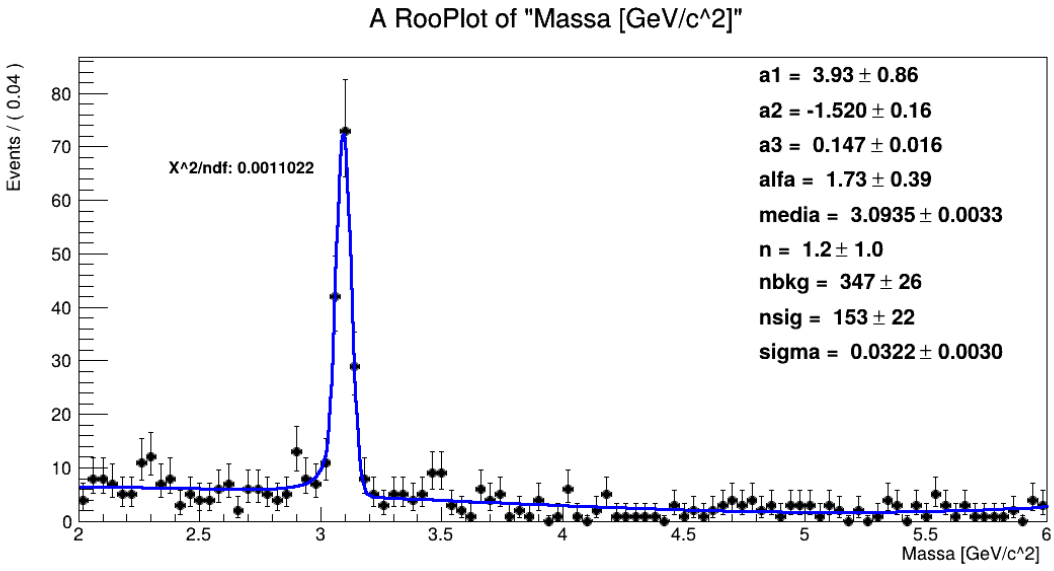

Figure 3: jpsi

Como foi ecnontrado um X^2/ndf < 1, significa que os dados se ajustaram bem ao modelo.