Introdução à Análise de dados em Física de Altas Energias

Exercícios de ROOT

Professores: Dilson de Jesus Damião, Mauricio Thiel e Eliza Melo

Aluno: Thiago Henrique de Sousa

(18/10/2024)

Aviso: Todos os códigos se encontram no github:
https://github.com/Sousa-Thiago/Analise__Fae_Tarefas/tree/Tarefas/Exercicios_ROOT

EXERCÍCIO 0

Esta tarefa é para analisar amostras de dodos de CMS/LHC do OpenData, obtidos através do caminho:

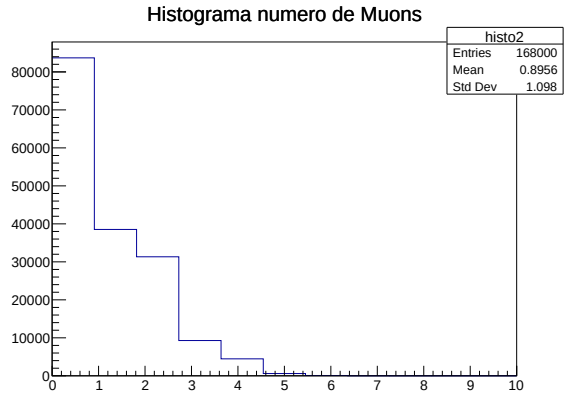
```
/opendata/eos/opendata/cms/mc/RunIISummer20UL16NanoAODv9/ZZTo4L_TuneCP5_13TeV_powheg_pytihia8
/NANOAODSIM/106X_mcRun2_asymptotic_v17-v1/2430000
```

Utilizamos a ferramenta *MakeClass* para gerar gráficos da distribuição de qualquer variável. Foi feito os gráficas das variáveis *nElectron* e *nMuon*. O código para gerar os gráficos foi:

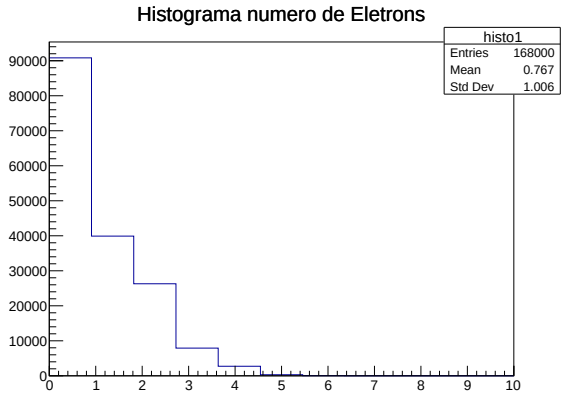
```
#define readerEvents_cxx
#include "readerEvents.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void readerEvents::Loop()
{
       if (fChain == 0) return;
    TH1F *histo1 = new TH1F("histo1", "Histograma numero de Eletrons", 11, 0, 10);
    TH1F *histo2 = new TH1F("histo2", "Histograma numero de Muons", 11, 0, 10);

   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
       // cout << "test" << endl;

       histo1->Fill(nElectron);
       histo2->Fill(nMuon);
```

```
}

TCanvas *canvas1 = new TCanvas("canvas1", "Histograma numero de Eletrons", 800, 600);
    histo1->Draw();
    canvas1->Print("histograma_muon.pdf");

    TCanvas *canvas2 = new TCanvas("canvas2", "Histograma numero de Muons", 800, 600);
    histo2->Draw();
    canvas2->Print("histograma_eletron.pdf");

}
```

E obtemos os seguintes gráficos:



(a) Gráfico do número de Elétrons.



(b) Gráfico do número de Múons.

Figure 1: Gráficos do número de Elétrons e Múons.

Fonte: O autor.

EXERCÍCIO 1 Código elaborado para esta tarefa:

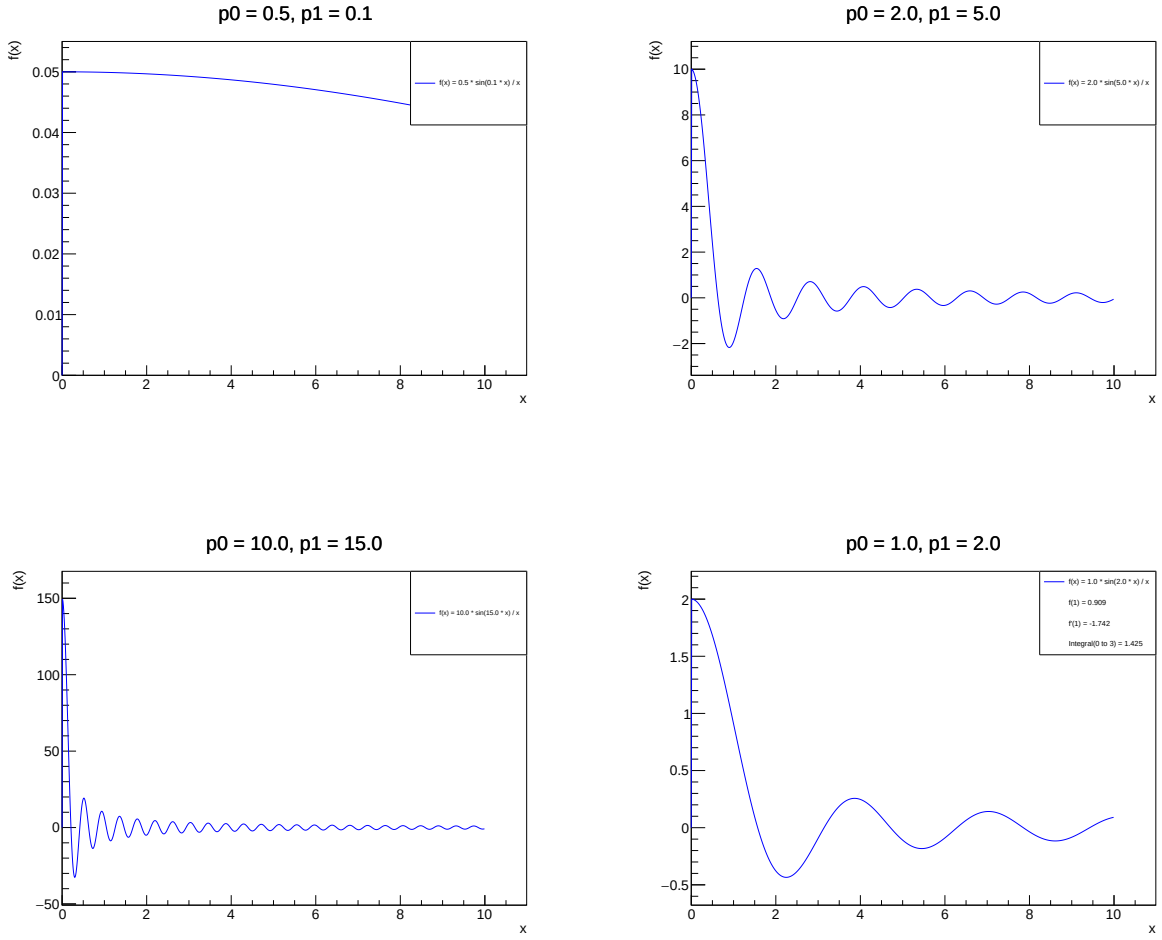
```
void exercicio3_1() {
    // Criando a função, parâmetros p0 e p1
    auto sine_function = [](double x, double p0, double p1) {
        return p0 * sin(p1 * x) / x;
    };

    // Parâmetros
    std::vector<std::pair<double, double>> parameters = {
        {0.5, 0.1},
        {2.0, 5.0},
        {10.0, 15.0},
        {1.0, 2.0}
    };

    // Intervalo de x
    const int nPoints = 1000;
```

```
    double x[nPoints];
    double y[nPoints];

    // Criar e salvar os gráficos
    for (size_t i = 0; i < parameters.size(); ++i) {
        double p0 = parameters[i].first;
        double p1 = parameters[i].second;

        // Preencher os valores de x e y
        for (int j = 0; j < nPoints; ++j) {
            x[j] = j / 100.0; // de 0 a 10
            if (x[j] != 0) {
                y[j] = sine_function(x[j], p0, p1);
            } else {
                y[j] = 0; // Evitar divisão por 0
            }
        }

        // Plot
        TGraph *graph = new TGraph(nPoints, x, y);
        graph->SetTitle(Form("p0 = %.1f, p1 = %.1f", p0, p1));
        graph->GetXaxis()->SetTitle("x");
        graph->GetYaxis()->SetTitle("f(x)");
        graph->SetLineColor(kBlue); // Plots azul

        // Legenda em todos os plots
        auto legend = new TLegend(0.7, 0.7, 0.9, 0.9); // Canto superior direito
        legend->AddEntry(graph, Form("f(x) = %.1f * sin(%.1f * x) / x", p0, p1), "l");

        // Verificando se p0=1 e p1=2 para cálculos adicionais
        if (p0 == 1.0 && p1 == 2.0) {
            double x_value = 1.0;
            double function_value = sine_function(x_value, p0, p1);

            // Definindo a função derivada
            TF1 *f = new TF1("f", [&](double *x, double *par) {
                return sine_function(x[0], par[0], par[1]);
            }, 0, 10, 2);
            f->SetParameters(p0, p1);
            double derivative_value = f->Derivative(x_value);

            // Definindo a função integral
            double integral_value = f->Integral(0, 3);

            // Adicionar resultados adicionais à legenda
            legend->AddEntry((TObject*)0, Form("f(1) = %.3f", function_value), "");
            legend->AddEntry((TObject*)0, Form("f'(1) = %.3f", derivative_value), "");
            legend->AddEntry((TObject*)0, Form("Integral(0 to 3) = %.3f", integral_value), "");

            delete f; // Limpar a função
        }

        // Salvar plot em PDF
        TCanvas *canvas = new TCanvas("canvas", "Canvas", 800, 600);
        graph->Draw("AL");
        legend->Draw(); // Legenda após o plot
        canvas->SaveAs(Form("plot_%lu.pdf", i + 1)); // Corrigido para %lu
        delete canvas; // Limpar o canvas depois de uso
```

```
75
76          // Limpar o plot para o próximo
77          delete graph;
78      }
79  }
80
```

Primeiramente definimos a função e obtemos os seguintes gráficos variando os parâmetros da mesma:

$$f(x) = \frac{p_0 sin(p_1 x)}{x} \tag{1}$$

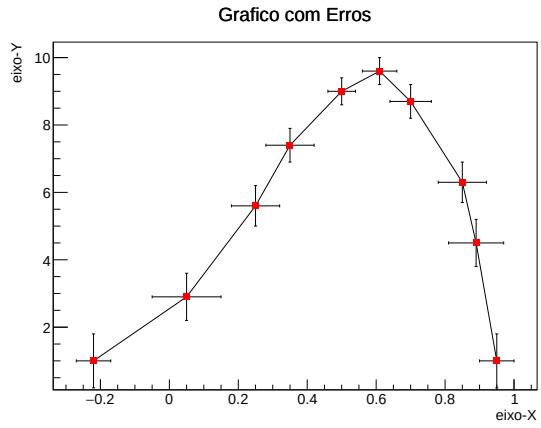


Fonte: O autor.

EXERCÍCIO 2 Neste exercício utilizamos um conjunto de dados chamado *graphdata.txt* e usamos a classe *TGraph* para fazer o gráfico. Foi criado um *TGraphError* utilizando o conjunto de dados com erros em x e y do arquivo $graphdata_error.txt$. Segue abaixo o código usado:

```
1
2  #include <TGraph.h>
3  #include <TGraphErrors.h>
4  #include <TCanvas.h>
5  #include <TStyle.h>
6  #include <iostream>
7
8  void plotGraphs() {
9      // Canvas
10     TCanvas *canvas = new TCanvas("canvas", "Graph with Errors", 800, 600);
11
12     // TGraph para os dados
13     TGraph *graph = new TGraph("graphdata.txt");
14     if (!graph) {
15         std::cerr << "Erro ao carregar graphdata.txt" << std::endl;
16         return;
17     }
18
19     // TGraphErrors para as barras de erros
20     TGraphErrors *graphErrors = new TGraphErrors("graphdata_error.txt");
21     if (!graphErrors) {
22         std::cerr << "Erro ao carregar graphdata_error.txt" << std::endl;
23         return;
24     }
25
26     // Configurações do plot
27     graph->SetMarkerStyle(22); // Estilo do marcador de caixa preta
28     graph->SetMarkerColor(kBlack);
29     graph->SetTitle("Grafico com Erros;eixo-X;eixo-Y");
30     graph->Draw("ALP"); // Desenha os pontos do TGraph com linhas
31
32     // Desenha o gráfico com erro
33     graphErrors->SetMarkerStyle(21); // Estilo do marcador de círculo
34     graphErrors->SetMarkerColor(kRed);
35     graphErrors->Draw("P"); // Desenha os pontos do TGraphErrors
36
37     // Salva o gráfico em PDF
38     canvas->Print("exercício_2.pdf");
39
40     // Limpa a memória
41     delete canvas;
42     delete graph;
43     delete graphErrors;
44 }
45
46 void exercicio3_2() {
47     plotGraphs();
48 }
```

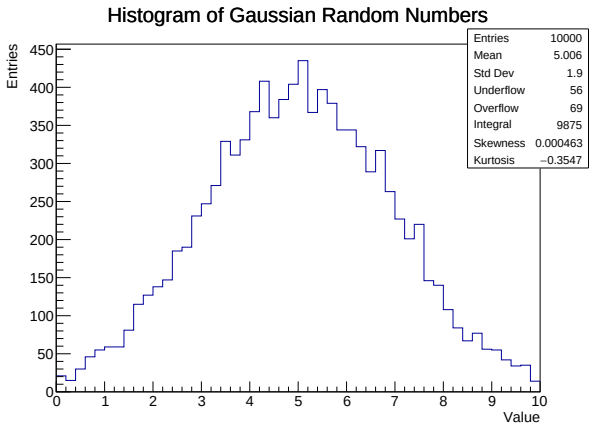
Obtendo o seguinte gráfico:



Fonte: O autor.

Exercício 3 Este exercício pede um histograma com 10000 valores aleatórios distribuídos 50 bins, média 5, desvio padrão 2 e num intervalo de 0 a 10. Segue o código:

```
1
2  #include <TH1F.h>
3  #include <TCanvas.h>
4  #include <TRandom3.h>
5  #include <TStyle.h>
6  #include <TApplication.h>
7
8  void exercicio3_3() {
9      // Gerando números aleatórios
10     TRandom3 random;
11
12     // Histograma com 50 bins de 0 e 10
13     TH1F *hist = new TH1F("hist", "Histogram of Gaussian Random Numbers;Value;Entries", 50, 0, 10);
14
15     // Histograma com 10000 valores aleatórios com distribuição gaussiana
16     for (int i = 0; i < 10000; i++) {
17         double value = random.Gaus(5, 2); // Média 5 e sigma 2
18         hist->Fill(value);
19     }
20
21     // Desenhar o histograma
22     TCanvas *canvas = new TCanvas("canvas", "Gaussian Histogram", 800, 600);
23
24     // Define a caixa de estatísticas
25     gStyle->SetOptStat("kseiorum"); // k: kurtosis, s: skewness, i: integral, o: overflows, u: under
26
27     // Desenha o histograma
28     hist->Draw();
29
30     // Salva o plot em PDF
31     canvas->Print("exercício_3.pdf");
32 }
33
```



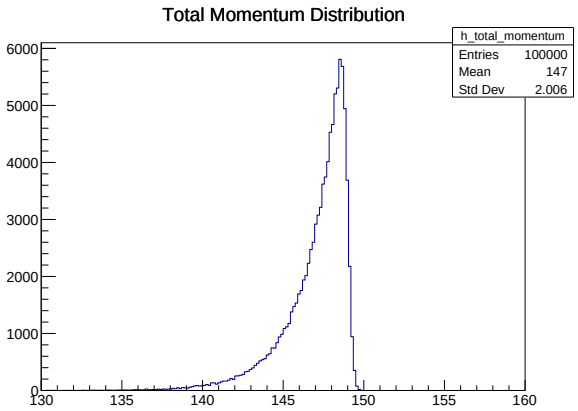
Fonte: O autor.

Exercício 4

Usamos a *tree* de um arquivo *tree.root* para fazer o gráfico da distribuição de momento total. Segue o código o gráfico desta tarefa:

```
void exercicio3_4() {
    // Abrindo o arquivo
    TFile *file = TFile::Open("tree.root");
    TTree *tree = (TTree*)file->Get("tree1");

    // Variáveis
    float ebeam, px, py, pz;
    tree->SetBranchAddress("ebeam", &ebeam);
    tree->SetBranchAddress("px", &px);
    tree->SetBranchAddress("py", &py);
    tree->SetBranchAddress("pz", &pz);

    // Cálcula da média da energia do feixe
    double totalEnergy = 0;
    Long64_t nEntries = tree->GetEntries();

    // Loop do cálculo da média
    for (Long64_t i = 0; i < nEntries; i++) {
        tree->GetEntry(i);
        totalEnergy += ebeam; // Soma a energia do feixe
    }

    // Vereficar se a média é > 0
    if (nEntries > 0) {
        double meanEnergy = totalEnergy / nEntries; // Média
        double lowerCut = meanEnergy - 0.2;
        double upperCut = meanEnergy + 0.2;

        // Criando os cortes
        TCut cut = Form("ebeam < %f || ebeam > %f", upperCut, lowerCut);

        // Montando o histograma
        TH1F *histogram = new TH1F("h_total_momentum", "Total Momentum Distribution", 200, 130, 160)

```

```
35          // Preenchendo o histograma TTree::Draw
36          tree->Draw("TMath::Sqrt(px*px + py*py + pz*pz)>>h_total_momentum", cut);
37
38          // Criar um canvas para desenhar o histograma
39          TCanvas *c2 = new TCanvas("c2", "Total Momentum Distribution", 800, 600);
40          histogram->Draw();
41
42          // Salvar em arquivo PDF
43          c2->SaveAs("exercício_4.pdf");
44      }
45      // Fechar o arquivo
46      file->Close();
47  }
```



Fonte: O autor.