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

**Exercícios de Manipulação de dados com ROOT**
**(aula 6)**

**Professores:**
Eliza Melo, Dilson Damião e Mauricio Thiel

**Nome:**
Matheus da Costa Geraldes

# 1 Aviso

os códigos, as imagens e assim como os arquivos usados se encontram no meu Git: `https://github.com/Geraldes-Matheus/cruso-analise-de-dados-2024-2/tree/main/exercicio6/Data`

# 2 parte 1

Na primeira parte da análise, o objetivo foi aplicar cortes de seleção para escolher os leptons com boas características e calcular a massa invariante a partir dessas partículas. Os cortes aplicados foram os seguintes:

- O $pT$ (transverse momentum) dos leptons deve ser maior que 20 GeV.
- O *eta* (pseudorrápidez) dos leptons deve estar dentro do intervalo $|\eta| < 2.5$.

Esses cortes garantem que apenas leptons com boa qualidade de medida e suficientemente energéticos sejam considerados. Após a aplicação desses cortes, a massa invariante dos dois leptons com maior $pT$ foi calculada.

A massa invariante $M$ foi calculada pela seguinte fórmula:

$$M = \sqrt{2 p_{T1} p_{T2} \left(\cosh(\eta_1 - \eta_2) - \cos(\phi_1 - \phi_2)\right)}$$

Onde $p_{T1}$, $p_{T2}$ são os momentos transversos dos leptons, $\eta_1$, $\eta_2$ são as pseudorrápidezes, e $\phi_1$, $\phi_2$ são os ângulos azimutais dos leptons.

O número de eventos foi afetado pela aplicação dos cortes de seleção. Como esperado, a aplicação de cortes na energia ($p_T > 20\,\text{GeV}$) e no intervalo de pseudorrápidez ($|\eta| < 2.5$) resultou em uma redução no número total de leptons considerados, uma vez que eliminamos eventos com partículas de baixa energia ou com valores de $\eta$ fora da região de aceitação.

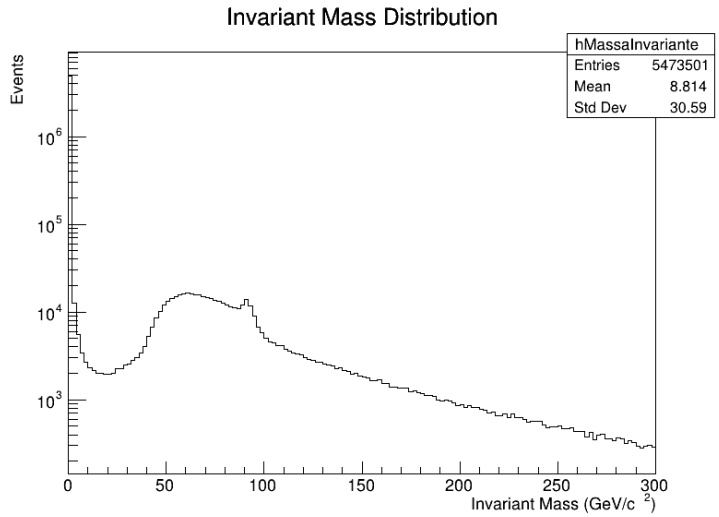


Figure 1: Distribuição da massa invariante dos leptons após a aplicação dos cortes de seleção ($p_T > 20$ GeV, $|\eta| < 2.5$).

Depois foi selecionado um sinal, que no caso foi escolhido o pico do Bóson Z.

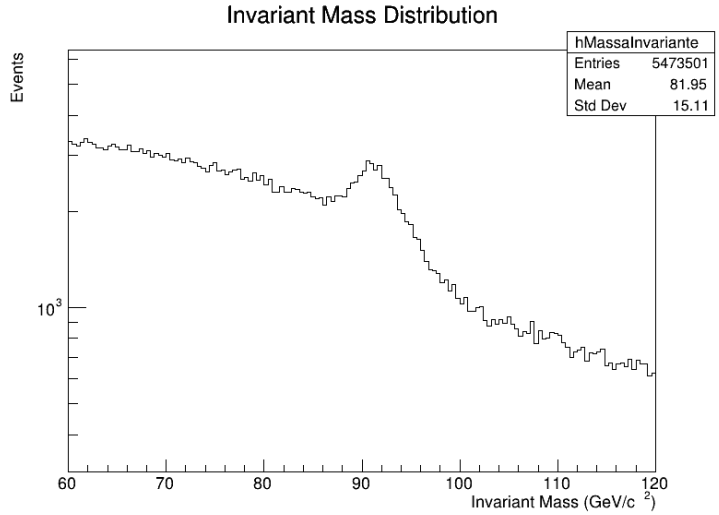


Figure 2: pico do Bóson Z

# 3 Parte 2: Ajuste da Distribuição da Massa Invariante

Na segunda parte do exercício, a tarefa foi realizar o ajuste dos dados utilizando duas funções: uma para o sinal (pico central) e uma para o fundo (comportamento de cauda). A ideia é ajustar uma função composta (sinal + fundo) aos dados observados, como uma forma de modelar a distribuição da massa invariante.

1. **Função de Sinal** (Crystal Ball): Representa o pico da distribuição, correspondente ao sinal desejado (por exemplo, o resíduo do Z-boson).

2. **Função de Fundo** (Exponencial): Representa o fundo da distribuição, descrevendo eventos que não pertencem ao sinal de interesse.

Deve-se ajustar essas duas funções aos dados observados (a distribuição de massa invariante), combinando-as em um modelo total. A fração do sinal (como uma parte do total) deve ser ajustada como um parâmetro.

O resultado do ajuste foi plotado em um gráfico que foi salvo no formato PNG com o nome "$\text{mass}_{Z}fit.png$".

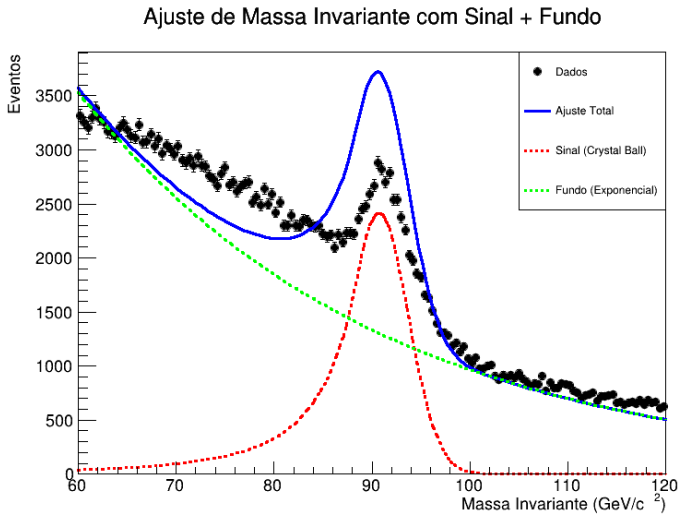


Figure 3: Distribuição da massa invariante dos leptons com ajuste de fundo exponencial e Gaussiano.

# 4 Código Implementado

## 4.1 Código da parte 1

```cpp
#include <TTree.h>
#include <TTreeReader.h>
#include <TTreeReaderArray.h>
#include <TCanvas.h>
#include <TH1F.h>
#include <TMath.h>
#include <iostream>
#include <vector>
#include <filesystem>
#include <algorithm>
#include <string>

double calcular_massa_invariante(const std::vector<float>& pt, const std::vector<float>& eta, const std::vector<float>& phi) {
    if (pt.size() >= 2) {
        return sqrt(2 * pt[0] * pt[1] * (TMath::CosH(eta[0] - eta[1]) - TMath::Cos(phi[0] - phi[1])));
    }
    return -1.0;
}

void analise() {
    // Diretórios para análise
    std::vector<std::string> diretorios = {
        "/opendata/eos/opendata/cms/mc/RunIISummer20UL16NanoAODv9/ZZTo4L_TuneCP5_13TeV_powheg_pythia8/NANOAODSIM/106X_mcRun2_asymptotic_v17-v1/2430000",
        "/opendata/eos/opendata/cms/mc/RunIISummer20UL16NanoAODv9/ZZTo4L_TuneCP5_13TeV_powheg_pythia8/NANOAODSIM/106X_mcRun2_asymptotic_v17-v1/2520000"
    };

    std::vector<double> e_massas_invariantes;

    // Histogramas
    TH1F* hMassaInvariante = new TH1F("hMassaInvariante", "Invariant Mass Distribution", 150, 0, 300);

    for (const auto& diretorio : diretorios) {
        for (const auto& entry : std::filesystem::directory_iterator(diretorio)) {
            std::string file_path = entry.path();
            TFile file(file_path.c_str(), "READ");
            if (!file.IsOpen()) continue;

            TTreeReader reader("Events", &file);
            TTreeReaderArray<float> Electron_pt(reader, "Electron_pt");
            TTreeReaderArray<float> Electron_eta(reader, "Electron_eta");
            TTreeReaderArray<float> Electron_phi(reader, "Electron_phi");
            TTreeReaderArray<float> Muon_pt(reader, "Muon_pt");
            TTreeReaderArray<float> Muon_eta(reader, "Muon_eta");
            TTreeReaderArray<float> Muon_phi(reader, "Muon_phi");
            TTreeReaderArray<float> Jet_pt(reader, "Jet_pt");
            TTreeReaderArray<float> Jet_eta(reader, "Jet_eta");
            TTreeReaderArray<float> Jet_phi(reader, "Jet_phi");
            TTreeReaderArray<float> Tau_pt(reader, "Tau_pt");
            TTreeReaderArray<float> Tau_eta(reader, "Tau_eta");
            TTreeReaderArray<float> Tau_phi(reader, "Tau_phi");

            while (reader.Next()) {
                // Vetor de leptons com pT e eta
                std::vector<std::pair<float, int>> leptons; // (pT, índice)
                for (size_t i = 0; i < Electron_pt.GetSize(); ++i) {
                    if (Electron_pt[i] > 20 && fabs(Electron_eta[i]) < 2.5) {
                        leptons.emplace_back(Electron_pt[i], i); // Elétrons
                    }
                }
                for (size_t i = 0; i < Muon_pt.GetSize(); ++i) {
```

```
61                      if (Muon_pt[i] > 20 && fabs(Muon_eta[i]) < 2.5) {
62                          leptons.emplace_back(Muon_pt[i], i + Electron_pt.
                                GetSize()); // Muons
63                      }
64                  }
65                  for (size_t i = 0; i < Tau_pt.GetSize(); ++i) {
66                      if (Tau_pt[i] > 20 && fabs(Tau_eta[i]) < 2.5) {
67                          leptons.emplace_back(Tau_pt[i], i + Electron_pt.
                                GetSize() + Muon_pt.GetSize()); // Taus
68                      }
69                  }
70
71                  // Ordenar pT de forma decrescente
72                  std::sort(leptons.rbegin(), leptons.rend());
73
74                  // Seleciona os dois leptons com maior pT
75                  if (leptons.size() >= 2) {
76                      int idx1 = leptons[0].second;
77                      int idx2 = leptons[1].second;
78
79                      float pt1, eta1, phi1, pt2, eta2, phi2;
80
81                      // Atribuir valores de pt, eta, phi de cada lepton
82                      if (idx1 < Electron_pt.GetSize()) {
83                          pt1 = Electron_pt[idx1];
84                          eta1 = Electron_eta[idx1];
85                          phi1 = Electron_phi[idx1];
86                      } else if (idx1 < Electron_pt.GetSize() + Muon_pt.GetSize
                            ()) {
87                          pt1 = Muon_pt[idx1 - Electron_pt.GetSize()];
88                          eta1 = Muon_eta[idx1 - Electron_pt.GetSize()];
89                          phi1 = Muon_phi[idx1 - Electron_pt.GetSize()];
90                      } else {
91                          pt1 = Tau_pt[idx1 - Electron_pt.GetSize() - Muon_pt.
                                GetSize()];
92                          eta1 = Tau_eta[idx1 - Electron_pt.GetSize() - Muon_pt
                                .GetSize()];
93                          phi1 = Tau_phi[idx1 - Electron_pt.GetSize() - Muon_pt
                                .GetSize()];
94                      }
95
96                      if (idx2 < Electron_pt.GetSize()) {
97                          pt2 = Electron_pt[idx2];
98                          eta2 = Electron_eta[idx2];
99                          phi2 = Electron_phi[idx2];
100                     } else if (idx2 < Electron_pt.GetSize() + Muon_pt.GetSize
                            ()) {
101                         pt2 = Muon_pt[idx2 - Electron_pt.GetSize()];
102                         eta2 = Muon_eta[idx2 - Electron_pt.GetSize()];
103                         phi2 = Muon_phi[idx2 - Electron_pt.GetSize()];
104                     } else {
105                         pt2 = Tau_pt[idx2 - Electron_pt.GetSize() - Muon_pt.
                                GetSize()];
106                         eta2 = Tau_eta[idx2 - Electron_pt.GetSize() - Muon_pt
                                .GetSize()];
107                         phi2 = Tau_phi[idx2 - Electron_pt.GetSize() - Muon_pt
                                .GetSize()];
108                     }
109
110                     // Calcular massa invariante
111                     std::vector<float> pt_values = {pt1, pt2};
112                     std::vector<float> eta_values = {eta1, eta2};
113                     std::vector<float> phi_values = {phi1, phi2};
114                     double massa_invariante = calcular_massa_invariante(
                            pt_values, eta_values, phi_values);
115
116                     if (massa_invariante >= 0) {
117                         e_massas_invariantes.push_back(massa_invariante);
```

```
                    }
                }
            }
        }
    }

    // Plot da Massa Invariante
    TCanvas* canvas = new TCanvas("canvasInvariantMass", "Invariant Mass
         Distribution", 800, 600);

    hMassaInvariante->SetLineColor(kBlack);
    canvas->SetLogy();
    for (const auto& massa : e_massas_invariantes) {
        if (massa >= 0) hMassaInvariante->Fill(massa);
    }
    hMassaInvariante->Draw();
    hMassaInvariante->GetXaxis()->SetTitle("Invariant Mass (GeV/c^{2})");
    hMassaInvariante->GetYaxis()->SetTitle("Events");
    canvas->SaveAs("massa_da_distribui  o.png");
    delete canvas;

    delete hMassaInvariante;
}
```

## 4.2 Código da parte 2

```
#include <TTree.h>
#include <TTreeReader.h>
#include <TTreeReaderArray.h>
#include <TCanvas.h>
#include <TH1F.h>
#include <TMath.h>
#include <RooFit.h>
#include <RooRealVar.h>
#include <RooDataHist.h>
#include <RooCBShape.h>
#include <RooExponential.h>
#include <RooAddPdf.h>
#include <RooPlot.h>
#include <iostream>
#include <vector>
#include <filesystem>
#include <algorithm>
#include <string>

double calcular_massa_invariante(const std::vector<float>& pt, const std::
    vector<float>& eta, const std::vector<float>& phi) {
    if (pt.size() >= 2) {
        return sqrt(2 * pt[0] * pt[1] * (TMath::CosH(eta[0] - eta[1]) - TMath
            ::Cos(phi[0] - phi[1])));
    }
    return -1.0;
}

void analisefit() {
    // Diret rios para an lise
    std::vector<std::string> diretorios = {
        "/opendata/eos/opendata/cms/mc/RunIISummer20UL16NanoAODv9/
            ZZTo4L_TuneCP5_13TeV_powheg_pythia8/NANOAODSIM/106
            X_mcRun2_asymptotic_v17-v1/2430000",
        "/opendata/eos/opendata/cms/mc/RunIISummer20UL16NanoAODv9/
            ZZTo4L_TuneCP5_13TeV_powheg_pythia8/NANOAODSIM/106
            X_mcRun2_asymptotic_v17-v1/2520000"
    };

    std::vector<double> e_massas_invariantes;

```

```
    for (const auto& diretorio : diretorios) {
        for (const auto& entry : std::filesystem::directory_iterator(
            diretorio)) {
            std::string file_path = entry.path();
            TFile file(file_path.c_str(), "READ");
            if (!file.IsOpen()) continue;

            TTreeReader reader("Events", &file);
            TTreeReaderArray<float> Electron_pt(reader, "Electron_pt");
            TTreeReaderArray<float> Electron_eta(reader, "Electron_eta");
            TTreeReaderArray<float> Electron_phi(reader, "Electron_phi");
            TTreeReaderArray<float> Muon_pt(reader, "Muon_pt");
            TTreeReaderArray<float> Muon_eta(reader, "Muon_eta");
            TTreeReaderArray<float> Muon_phi(reader, "Muon_phi");
            TTreeReaderArray<float> Tau_pt(reader, "Tau_pt");
            TTreeReaderArray<float> Tau_eta(reader, "Tau_eta");
            TTreeReaderArray<float> Tau_phi(reader, "Tau_phi");

            while (reader.Next()) {
                std::vector<std::pair<float, int>> leptons;
                for (size_t i = 0; i < Electron_pt.GetSize(); ++i) {
                    if (Electron_pt[i] > 20 && fabs(Electron_eta[i]) < 2.5) {
                        leptons.emplace_back(Electron_pt[i], i);
                    }
                }
                for (size_t i = 0; i < Muon_pt.GetSize(); ++i) {
                    if (Muon_pt[i] > 20 && fabs(Muon_eta[i]) < 2.5) {
                        leptons.emplace_back(Muon_pt[i], i + Electron_pt.
                            GetSize());
                    }
                }
                for (size_t i = 0; i < Tau_pt.GetSize(); ++i) {
                    if (Tau_pt[i] > 20 && fabs(Tau_eta[i]) < 2.5) {
                        leptons.emplace_back(Tau_pt[i], i + Electron_pt.
                            GetSize() + Muon_pt.GetSize());
                    }
                }

                std::sort(leptons.rbegin(), leptons.rend());

                if (leptons.size() >= 2) {
                    int idx1 = leptons[0].second;
                    int idx2 = leptons[1].second;

                    float pt1, eta1, phi1, pt2, eta2, phi2;

                    if (idx1 < Electron_pt.GetSize()) {
                        pt1 = Electron_pt[idx1];
                        eta1 = Electron_eta[idx1];
                        phi1 = Electron_phi[idx1];
                    } else if (idx1 < Electron_pt.GetSize() + Muon_pt.GetSize
                        ()) {
                        pt1 = Muon_pt[idx1 - Electron_pt.GetSize()];
                        eta1 = Muon_eta[idx1 - Electron_pt.GetSize()];
                        phi1 = Muon_phi[idx1 - Electron_pt.GetSize()];
                    } else {
                        pt1 = Tau_pt[idx1 - Electron_pt.GetSize() - Muon_pt.
                            GetSize()];
                        eta1 = Tau_eta[idx1 - Electron_pt.GetSize() - Muon_pt
                            .GetSize()];
                        phi1 = Tau_phi[idx1 - Electron_pt.GetSize() - Muon_pt
                            .GetSize()];
                    }

                    if (idx2 < Electron_pt.GetSize()) {
                        pt2 = Electron_pt[idx2];
                        eta2 = Electron_eta[idx2];
                        phi2 = Electron_phi[idx2];
```

```
                    } else if (idx2 < Electron_pt.GetSize() + Muon_pt.GetSize
                        ()) {
                        pt2 = Muon_pt[idx2 - Electron_pt.GetSize()];
                        eta2 = Muon_eta[idx2 - Electron_pt.GetSize()];
                        phi2 = Muon_phi[idx2 - Electron_pt.GetSize()];
                    } else {
                        pt2 = Tau_pt[idx2 - Electron_pt.GetSize() - Muon_pt.
                            GetSize()];
                        eta2 = Tau_eta[idx2 - Electron_pt.GetSize() - Muon_pt
                            .GetSize()];
                        phi2 = Tau_phi[idx2 - Electron_pt.GetSize() - Muon_pt
                            .GetSize()];
                    }

                    std::vector<float> pt_values = {pt1, pt2};
                    std::vector<float> eta_values = {eta1, eta2};
                    std::vector<float> phi_values = {phi1, phi2};
                    double massa_invariante = calcular_massa_invariante(
                        pt_values, eta_values, phi_values);

                    if (massa_invariante >= 0) {
                        e_massas_invariantes.push_back(massa_invariante);
                    }
                }
            }
        }
    }

    // 1. Criamos o histograma com os dados de massa invariante
    TH1F* hMassaInvariante = new TH1F("hMassaInvariante", "Distribui  o de
        Massa Invariante", 150, 60, 120);
    for (const auto& massa : e_massas_invariantes) {
        if (massa >= 0) hMassaInvariante->Fill(massa);
    }

    // 2. Definimos a vari vel para o RooFit
    RooRealVar x("x", "Massa Invariante (GeV/c^{2})", 60, 120);

    // 3. Criamos o RooDataHist a partir do histograma
    RooDataHist data("data", "Dados de Massa Invariante", RooArgList(x),
        hMassaInvariante);

    // 4. Definimos as fun  es para o sinal (Crystal Ball) e fundo (
        Exponencial)
    RooRealVar mean("mean", "M dia", 91.2, 90, 93); // M dia do Z boson
    RooRealVar sigma("sigma", "Desvio padr o", 2.3, 1.5, 3.0); // Largura do
         pico
    RooRealVar alpha("alpha", "Par metro alpha", 1.3, 0.8, 2.5); //
        Par metro alpha (cauda)
    RooRealVar n("n", "Par metro n", 7, 3, 15); // Par metro n (cauda)
    RooCBShape signal("signal", "Fun  o Crystal Ball", x, mean, sigma,
        alpha, n);

    // Fun  o de fundo exponencial
    RooRealVar tau("tau", "Par metro Tau (Fundo Exponencial)", -0.2, -1.0,
        0.0);
    RooExponential background("background", "Fundo Exponencial", x, tau);

    // 5. Criamos o modelo total como uma soma (signal + background)
    RooRealVar fracSignal("fracSignal", "Fra  o de Sinal", 0.6, 0.2, 0.4);
        // Fra  o do sinal
    RooAddPdf model("model", "Sinal + Fundo", RooArgList(signal, background),
         RooArgList(fracSignal));

    // 6. Ajuste do modelo aos dados
    model.fitTo(data, RooFit::PrintLevel(-1), RooFit::Range(60, 120));

    // 7. Criamos o gr fico para o ajuste
```

```
151     RooPlot* frame = x.frame();
152     data.plotOn(frame);
153     model.plotOn(frame, RooFit::LineColor(kBlue)); // Ajuste total
154     model.plotOn(frame, RooFit::Components("signal"), RooFit::LineColor(kRed)
            , RooFit::LineStyle(kDashed)); // Sinal (Crystal Ball)
155     model.plotOn(frame, RooFit::Components("background"), RooFit::LineColor(
            kGreen), RooFit::LineStyle(kDashed)); // Fundo (Exponencial)
156
157     // 8. Cria  o do canvas e exibi  o do gr fico
158     TCanvas* canvas = new TCanvas("canvasFit", "Ajuste␣do␣Sinal␣+␣Fundo",
            800, 600);
159     frame->SetTitle("Ajuste␣de␣Massa␣Invariante␣com␣Sinal␣+␣Fundo");
160     frame->GetXaxis()->SetTitle("Massa␣Invariante␣(GeV/c^{2})");
161     frame->GetYaxis()->SetTitle("Eventos");
162     frame->Draw();
163
164     // 9. Adiciona a legenda
165     TLegend* legend = new TLegend(0.7, 0.6, 0.9, 0.9);
166     legend->AddEntry(frame->getObject(0), "Dados", "P");
167     legend->AddEntry(frame->getObject(1), "Ajuste␣Total", "L");
168     legend->AddEntry(frame->getObject(2), "Sinal␣(Crystal␣Ball)", "L");
169     legend->AddEntry(frame->getObject(3), "Fundo␣(Exponencial)", "L");
170     legend->Draw();
171
172     // 10. Salva a imagem
173     canvas->SaveAs("mass_Z_fit.png");
174
175     // Libera a mem ria
176     delete canvas;
177     delete frame;
178     delete legend;
179     delete hMassaInvariante;
180 }
```