Introdução à Análise de dados em FAE e tecnologias associadas **(DATA)**

Manipulando dados com o ROOT - Parte I

Professores: Dilson Damião,Eliza Melo e Mauricio Thiel *Name:* João Marcos Mosdesto Ribeiro

TEXTO

parte A - Cortes de seleção antes do plot da massa M

```
#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>

// Fun  o para calcular a massa invariante com dois leptons
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;  // Retorna -1 caso n o tenha pelo menos dois leptons
}

void Zmasscut() {
    // Diret rios de entrada
    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"
    };

    // Vetor para armazenar as massas invariantes
    std::vector<double> e_massas_invariantes;

    // Inicializando o histograma de massa invariante
    TH1F* hMassaInvariante = new TH1F("hMassaInvariante", "Invariant Mass
        Distribution", 200, 70, 110);

    // Processar os arquivos
    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");
```

```
43              TTreeReaderArray<float> Electron_eta(reader, "Electron_eta");
44              TTreeReaderArray<float> Electron_phi(reader, "Electron_phi");
45              TTreeReaderArray<float> Muon_pt(reader, "Muon_pt");
46              TTreeReaderArray<float> Muon_eta(reader, "Muon_eta");
47              TTreeReaderArray<float> Muon_phi(reader, "Muon_phi");
48              TTreeReaderArray<float> Jet_pt(reader, "Jet_pt");
49              TTreeReaderArray<float> Jet_eta(reader, "Jet_eta");
50              TTreeReaderArray<float> Jet_phi(reader, "Jet_phi");
51              TTreeReaderArray<float> Tau_pt(reader, "Tau_pt");
52              TTreeReaderArray<float> Tau_eta(reader, "Tau_eta");
53              TTreeReaderArray<float> Tau_phi(reader, "Tau_phi");
54
55              while (reader.Next()) {
56                  // Vetor de leptons com pT e eta
57                  std::vector<std::pair<float, int>> leptons; // (pT,  ndice )
58                  for (size_t i = 0; i < Electron_pt.GetSize(); ++i) {
59                      if (Electron_pt[i] > 20 && fabs(Electron_eta[i]) < 2.5) {
60                          leptons.emplace_back(Electron_pt[i], i); // El trons
61                      }
62                  }
63                  for (size_t i = 0; i < Muon_pt.GetSize(); ++i) {
64                      if (Muon_pt[i] > 20 && fabs(Muon_eta[i]) < 2.5) {
65                          leptons.emplace_back(Muon_pt[i], i + Electron_pt.GetSize());
                                // M ons
66                      }
67                  }
68                  for (size_t i = 0; i < Tau_pt.GetSize(); ++i) {
69                      if (Tau_pt[i] > 20 && fabs(Tau_eta[i]) < 2.5) {
70                          leptons.emplace_back(Tau_pt[i], i + Electron_pt.GetSize() +
                                Muon_pt.GetSize()); // Taus
71                      }
72                  }
73
74                  // Ordenar leptons por pT em ordem decrescente
75                  std::sort(leptons.rbegin(), leptons.rend());
76
77                  // Selecionar os dois leptons com maior pT
78                  if (leptons.size() >= 2) {
79                      int idx1 = leptons[0].second;
80                      int idx2 = leptons[1].second;
81
82                      float pt1, eta1, phi1, pt2, eta2, phi2;
83
84                      // Atribuir valores de pt, eta, phi para cada lepton
85                      if (idx1 < Electron_pt.GetSize()) {
86                          pt1 = Electron_pt[idx1];
87                          eta1 = Electron_eta[idx1];
88                          phi1 = Electron_phi[idx1];
89                      } else if (idx1 < Electron_pt.GetSize() + Muon_pt.GetSize()) {
90                          pt1 = Muon_pt[idx1 - Electron_pt.GetSize()];
91                          eta1 = Muon_eta[idx1 - Electron_pt.GetSize()];
92                          phi1 = Muon_phi[idx1 - Electron_pt.GetSize()];
93                      } else {
94                          pt1 = Tau_pt[idx1 - Electron_pt.GetSize() - Muon_pt.GetSize()
                                ];
95                          eta1 = Tau_eta[idx1 - Electron_pt.GetSize() - Muon_pt.GetSize
                                ()];
96                          phi1 = Tau_phi[idx1 - Electron_pt.GetSize() - Muon_pt.GetSize
                                ()];
97                      }
98
99                      if (idx2 < Electron_pt.GetSize()) {
100                         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()];
                    }

                    // Calcular a massa invariante
                    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);
                    }
                }
            }
        }
    }

    // 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("invariant_mass_Z_v2.png");

    // Limpeza
    delete canvas;
    delete hMassaInvariante;
}
```

Análise do Impacto dos Limiar para p_T e η no Número de Eventos

O número de eventos pode ser afetado pelos limiares impostos para o momento transversal (p_T) e a pseudorapidez (η) dos leptons utilizados no cálculo da massa invariante. A seguir, analisamos como esses limiares influenciam a seleção de eventos.

Limiar para p_T

O limiar para p_T foi fixado em 20 GeV, o que significa que apenas leptons com $p_T \geq 20\,\text{GeV}$ são considerados na análise. O parâmetro p_T está relacionado à energia transversal da partícula, e a exigência de um valor mínimo de p_T visa:

- **Reduzir o fundo de partículas de baixa energia**, que são mais difíceis de detectar e têm uma maior probabilidade de serem confundidas com o ruído do detector.

- **Focar em partículas com maior energia**, que são mais relevantes em eventos de alta energia, como aqueles envolvendo o Z boson ou outros processos de interesse físico.
- **Reduzir o número de eventos com leptons de baixa energia**, já que partículas com p_T baixo são menos significativas para a análise e têm maior chance de não serem registradas corretamente.

Portanto, a escolha de $p_T \geq 20\,\text{GeV}$ ajuda a aumentar a qualidade da amostra de dados, ao mesmo tempo em que pode diminuir o número de eventos selecionados.

Limiar para η

O limiar para η, a pseudorapidez, foi definido como $|\eta| < 2.5$, restringindo os leptons àqueles que estão dentro de uma região central do detector, onde a eficiência de detecção é melhor. Essa condição afeta o número de eventos da seguinte forma:

- **Eliminação de leptons fora da região de alta eficiência do detector**, ou seja, leptons com $|\eta| > 2.5$ são descartados, pois estariam fora da área onde o detector é mais sensível e precisa.
- **Foco em partículas bem registradas**, pois os leptons dentro da faixa de $|\eta| < 2.5$ têm maior chance de serem corretamente detectados e identificados.

A escolha de $|\eta| < 2.5$ é comum em experimentos de física de partículas, como os realizados nos experimentos CMS e ATLAS, onde essa região abrange as áreas de boa cobertura do detector.

Impacto no Número de Eventos

A aplicação desses limiares ($p_T \geq 20\,\text{GeV}$ e $|\eta| < 2.5$) resulta em uma redução no número de eventos, pois partículas com baixa energia (p_T) ou fora da região de alta eficiência do detector ($|\eta| > 2.5$) são excluídas. No entanto, essa redução é compensada pela melhoria na qualidade dos eventos selecionados, uma vez que apenas leptons de alta energia e bem registrados são considerados.

Ao escolher limiares mais baixos para p_T ou mais amplos para η, o número de eventos pode aumentar, mas isso pode comprometer a precisão e a relevância dos dados, uma vez que partículas de menor energia ou fora da região de boa cobertura podem gerar resultados menos confiáveis.

Parte 2 - Ajustes para os cortes aplicados com mais dados

```
#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>

// Defini  o da fun  o de massa invariante
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 Zmasscutfit() {
    // 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",
        "/opendata/eos/opendata/cms/mc/RunIISummer20UL16NanoAODv9/ZZTo2Q2L_mllmin4p0_TuneCP5_13TeV-amcatnloFXFX-pythia8/NANOAODSIM/106X_mcRun2_asymptotic_v17-v1/80000"
    };

    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;
```

```
80
81                      if (idx1 < Electron_pt.GetSize()) {
82                          pt1 = Electron_pt[idx1];
83                          eta1 = Electron_eta[idx1];
84                          phi1 = Electron_phi[idx1];
85                      } else if (idx1 < Electron_pt.GetSize() + Muon_pt.GetSize()) {
86                          pt1 = Muon_pt[idx1 - Electron_pt.GetSize()];
87                          eta1 = Muon_eta[idx1 - Electron_pt.GetSize()];
88                          phi1 = Muon_phi[idx1 - Electron_pt.GetSize()];
89                      } else {
90                          pt1 = Tau_pt[idx1 - Electron_pt.GetSize() - Muon_pt.GetSize()
                                ];
91                          eta1 = Tau_eta[idx1 - Electron_pt.GetSize() - Muon_pt.GetSize
                                ()];
92                          phi1 = Tau_phi[idx1 - Electron_pt.GetSize() - Muon_pt.GetSize
                                ()];
93                      }
94
95                      if (idx2 < Electron_pt.GetSize()) {
96                          pt2 = Electron_pt[idx2];
97                          eta2 = Electron_eta[idx2];
98                          phi2 = Electron_phi[idx2];
99                      } else if (idx2 < Electron_pt.GetSize() + Muon_pt.GetSize()) {
100                         pt2 = Muon_pt[idx2 - Electron_pt.GetSize()];
101                         eta2 = Muon_eta[idx2 - Electron_pt.GetSize()];
102                         phi2 = Muon_phi[idx2 - Electron_pt.GetSize()];
103                     } else {
104                         pt2 = Tau_pt[idx2 - Electron_pt.GetSize() - Muon_pt.GetSize()
                                ];
105                         eta2 = Tau_eta[idx2 - Electron_pt.GetSize() - Muon_pt.GetSize
                                ()];
106                         phi2 = Tau_phi[idx2 - Electron_pt.GetSize() - Muon_pt.GetSize
                                ()];
107                     }
108
109                     std::vector<float> pt_values = {pt1, pt2};
110                     std::vector<float> eta_values = {eta1, eta2};
111                     std::vector<float> phi_values = {phi1, phi2};
112                     double massa_invariante = calcular_massa_invariante(pt_values,
                            eta_values, phi_values);
113
114                     if (massa_invariante >= 0) {
115                         e_massas_invariantes.push_back(massa_invariante);
116                     }
117                 }
118             }
119         }
120     }
121
122     // 1. Criamos o histograma com os dados de massa invariante
123     TH1F* hMassaInvariante = new TH1F("hMassaInvariante", "Distribui  o de Massa
            Invariante", 150, 60, 120);
124     for (const auto& massa : e_massas_invariantes) {
125         if (massa >= 0) hMassaInvariante->Fill(massa);
126     }
127
128     // 2. Definimos a vari vel para o RooFit
129     RooRealVar x("x", "Massa Invariante (GeV/c^{2})", 60, 120);
130
131     // 3. Criamos o RooDataHist a partir do histograma
132     RooDataHist data("data", "Dados de Massa Invariante", RooArgList(x),
            hMassaInvariante);
133
```

```
    // 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
    RooPlot* frame = x.frame();
    data.plotOn(frame);
    model.plotOn(frame, RooFit::LineColor(kBlue)); // Ajuste total
    model.plotOn(frame, RooFit::Components("signal"), RooFit::LineColor(kRed), RooFit
        ::LineStyle(kDashed)); // Sinal (Crystal Ball)
    model.plotOn(frame, RooFit::Components("background"), RooFit::LineColor(kGreen),
        RooFit::LineStyle(kDashed)); // Fundo (Exponencial)

    // 8. Cria  o do canvas e exibi  o do gr fico
    TCanvas* canvas = new TCanvas("canvasFit", "Ajuste do Sinal + Fundo", 800, 600);
    frame->SetTitle("Ajuste de Massa Invariante com Sinal + Fundo");
    frame->GetXaxis()->SetTitle("Massa Invariante (GeV/c^{2})");
    frame->GetYaxis()->SetTitle("Eventos");
    frame->Draw();

    // 9. Adiciona a legenda
    TLegend* legend = new TLegend(0.7, 0.6, 0.9, 0.9);
    legend->AddEntry(frame->getObject(0), "Dados", "P");
    legend->AddEntry(frame->getObject(1), "Ajuste Total", "L");
    legend->AddEntry(frame->getObject(2), "Sinal (Crystal Ball)", "L");
    legend->AddEntry(frame->getObject(3), "Fundo (Exponencial)", "L");
    legend->Draw();

    // 10. Salva a imagem
    canvas->SaveAs("mass_Z_fit.png");

    // Libera a mem ria
    delete canvas;
    delete frame;
    delete legend;
    delete hMassaInvariante;
}
```

Análise dos Limiares de Seleção e Ajuste de Massa Invariante

O código apresentado tem como objetivo calcular a massa invariante de pares de leptons (eletrons, múons e taus) e realizar um ajuste da distribuição dessa massa utilizando o modelo de uma função de sinal (Crystal Ball) combinada com um fundo exponencial. A seguir, são apresentadas as conclusões e considerações sobre a aplicação dos limiares de p_T e η, e seu impacto na qualidade do ajuste de massa invariante.

Filtragem dos Leptons

A seleção dos leptons para o cálculo da massa invariante é feita com base nos seguintes limiares:

- $p_T > 20\,\text{GeV}$: Apenas partículas com p_T superior a 20 GeV são consideradas. Isso ajuda a filtrar leptons de baixa energia, que são menos relevantes para a análise de eventos de alta energia, como aqueles associados ao Z boson.
- $|\eta| < 2.5$: O filtro de pseudorapidez $|\eta|$ garante que apenas leptons dentro da região de boa eficiência de detecção do detector sejam considerados.

Esses dois limiares são aplicados para melhorar a qualidade da amostra de dados, reduzindo a quantidade de eventos de baixa energia ou com leptons mal medidos, mas também resultando na redução do número de eventos analisados.

Impacto no Número de Eventos

A aplicação dos limiares de p_T e η reduz significativamente o número de eventos, uma vez que apenas leptons de alta energia e bem detectados são considerados. Isso pode resultar em:

- **Redução do número de eventos**: Apenas os leptons mais significativos são selecionados, o que pode levar a uma amostra de dados menor.
- **Melhoria na qualidade dos eventos**: A filtragem elimina leptons de baixa qualidade, resultando em uma amostra mais limpa e relevante para o ajuste da massa invariante.

Impacto no Ajuste de Massa Invariante

O código realiza o ajuste da distribuição da massa invariante utilizando um modelo composto por uma função de sinal (Crystal Ball) e uma função de fundo exponencial. A seleção dos eventos com base em p_T e η impacta diretamente o ajuste de massa invariante:

- **Ajuste com Menos Dados**: A redução no número de eventos pode comprometer a estatística do ajuste, resultando em um modelo menos preciso caso a amostra de dados seja pequena.
- **Melhora na Qualidade do Ajuste**: A amostra reduzida de leptons mais significativos contribui para um ajuste mais preciso, já que a função de fundo exponencial e a função de sinal (Crystal Ball) são ajustadas a dados mais confiáveis.

Conclusões Objetivas

- A aplicação de limiares de $p_T > 20\,\text{GeV}$ e $|\eta| < 2.5$ melhora a qualidade da amostra de dados ao eliminar leptons de baixa energia ou mal medidos.
- O número de eventos disponíveis para o ajuste da massa invariante é reduzido, o que pode diminuir a precisão do ajuste devido à menor estatística, mas também melhora a confiabilidade dos resultados.
- A combinação de uma função de sinal (Crystal Ball) e fundo exponencial, ajustada a dados selecionados, resulta em uma distribuição de massa invariante mais precisa e com menor incerteza.
- A redução de eventos pode afetar a estatística do ajuste, mas ao mesmo tempo assegura que os dados utilizados sejam mais representativos e consistentes com o modelo físico.

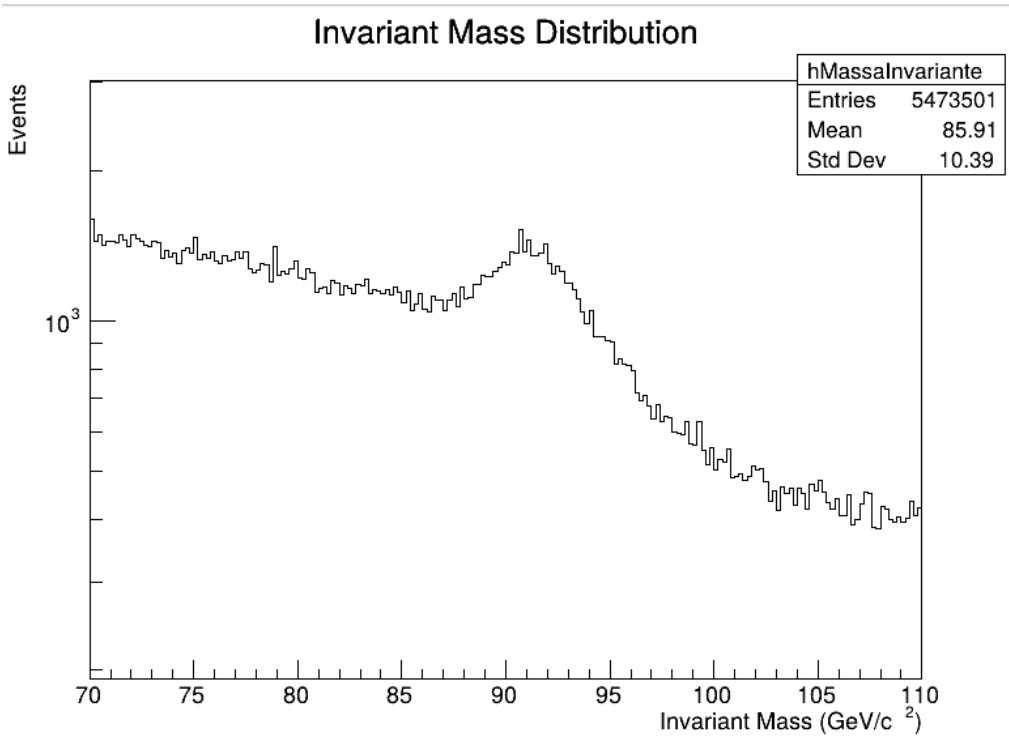

Figura 1: Distribuição de Massa Invariante com cortes.

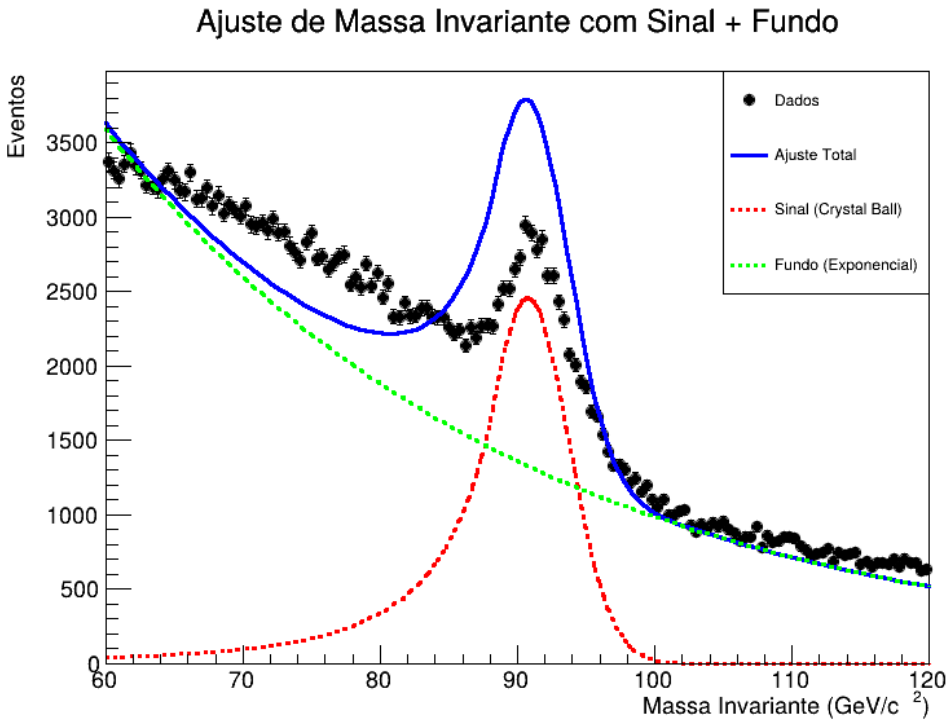

Figura 2: Distribuição de Massa Invariante com o ajuste do sinal (Crystal Ball) e fundo exponencial.