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

Lista 5

Professores: Dilson Damião, Eliza Melo e Maurício Thiel *Nome:* José Gonçalves Chaves Junior

EXERCICIO 1

Plotar os histogramas e ler os arquivos correspondentes ao grupo 3.

```
#include "TFile.h"
#include "TTreeReader.h"
#include "TMath.h"
#include "TH1F.h"
#include "TCanvas.h"
#include "TChain.h"
#include "TTreeReaderArray.h"
#include <vector>
#include <algorithm>
#include <numeric>
#include <iostream>

void read() {
    std::vector<std::string> diretorios = {
        "/opendata/eos/opendata/cms/Run2016G/DoubleEG/NANOAOD/
           UL2016_MiniAODv2_NanoAODv9-v1/100000/*.root",
        "/opendata/eos/opendata/cms/Run2016G/DoubleEG/NANOAOD/
           UL2016_MiniAODv2_NanoAODv9-v1/1010000/*.root",
        "/opendata/eos/opendata/cms/Run2016G/DoubleEG/NANOAOD/
           UL2016_MiniAODv2_NanoAODv9-v1/250000/*.root"
    };

    TChain chain("Events");
    for (const auto& path : diretorios) {
        chain.Add(path.c_str());
    }
    // Muons
    TTreeReaderArray<float> Muon_pt(reader, "Muon_pt");
    TTreeReaderArray<float> Muon_eta(reader, "Muon_eta");
    TTreeReaderArray<float> Muon_phi(reader, "Muon_phi");

    // Eletrons
    TTreeReader reader(&chain);
    TTreeReaderArray<float> Electron_pt(reader, "Electron_pt");
    TTreeReaderArray<float> Electron_eta(reader, "Electron_eta");
    TTreeReaderArray<float> Electron_phi(reader, "Electron_phi");

    // Taus
    TTreeReaderArray<float> Tau_pt(reader, "Tau_pt");
    TTreeReaderArray<float> Tau_eta(reader, "Tau_eta");
    TTreeReaderArray<float> Tau_phi(reader, "Tau_phi");

    // Jatos
    TTreeReaderArray<float> Jet_pt(reader, "Jet_pt");
    TTreeReaderArray<float> Jet_eta(reader, "Jet_eta");
    TTreeReaderArray<float> Jet_phi(reader, "Jet_phi");

    // Criando histogramas
    TH1F* hElectronPt = new TH1F("hElectronPt", "p_{T} eletron distribution", 50, 0,
       200);
```

```
47      TH1F* hElectronEta = new TH1F("hElectronEta", "#Eta eletron distribution", 50,
            -3, 3);
48      TH1F* hElectronPhi = new TH1F("hElectronPhi", "#Pi electron distribution", 50, -
            TMath::Pi(), TMath::Pi());
49
50      TH1F* hMuonPt = new TH1F("hMuonPt", "p_{T} muon distribution ", 50, 0, 200);
51      TH1F* hMuonEta = new TH1F("hMuonEta", "#Eta muon distribution", 50, -3.5, 3.5);
52      TH1F* hMuonPhi = new TH1F("hMuonPhi", "#Pi muon distribution", 50, -TMath::Pi(),
            TMath::Pi());
53
54      TH1F* hTauPt = new TH1F("hTauPt", "p_{T} tau distribution", 50, 0, 200);
55      TH1F* hTauEta = new TH1F("hTauEta", "#Eta  tau distribution", 50, -3.5, 3.5);
56      TH1F* hTauPhi = new TH1F("hTauPhi", "#Phi  tau distribution", 50, -TMath::Pi(),
            TMath::Pi());
57
58      TH1F* hJetPt = new TH1F("hJetPt", "Jets distribution", 50, 0, 200);
59      TH1F* hJetEta = new TH1F("hJetEta", "#Eta Jets distribution", 50, -3.5, 3.5);
60      TH1F* hJetPhi = new TH1F("hJetPhi", "#Phi  Jets distribution", 50, -TMath::Pi(),
            TMath::Pi());
61
62      TH1F* hInvariantMassMuon = new TH1F("hInvariantMassMuon", "", 50, 0, 200);
63      TH1F* hInvariantMassTau = new TH1F("hInvariantMassTau", "", 50, 0, 200);
64      TH1F* hInvariantMassElectron = new TH1F("hInvariantMassElectron", "", 50, 0, 200)
            ;
65
66          for (int i = 0; i < Electron_pt.GetSize(); ++i) {
67              hElectronPt->Fill(Electron_pt[i]);
68              hElectronEta->Fill(Electron_eta[i]);
69              hElectronPhi->Fill(Electron_phi[i]);
70          }
71
72          for (int i = 0; i < Muon_pt.GetSize(); ++i) {
73              hMuonPt->Fill(Muon_pt[i]);
74              hMuonEta->Fill(Muon_eta[i]);
75              hMuonPhi->Fill(Muon_phi[i]);
76          }
77
78          for (int i = 0; i < Tau_pt.GetSize(); ++i) {
79              hTauPt->Fill(Tau_pt[i]);
80              hTauEta->Fill(Tau_eta[i]);
81              hTauPhi->Fill(Tau_phi[i]);
82          }
83
84          for (int i = 0; i < Jet_pt.GetSize(); ++i) {
85              hJetPt->Fill(Jet_pt[i]);
86              hJetEta->Fill(Jet_eta[i]);
87              hJetPhi->Fill(Jet_phi[i]);
88          }
89
90          // massa invariante
91          if (Electron_pt.GetSize() >= 2) {
92              std::vector<TLorentzVector> electrons;
93              for (int i = 0; i < Electron_pt.GetSize(); ++i) {
94                  TLorentzVector electron;
95                  electron.SetPtEtaPhiM(Electron_pt[i], Electron_eta[i], Electron_phi[i
                        ], 0.000511); // Massa do el tron
96                  electrons.push_back(electron);
97              }
98              std::sort(electrons.begin(), electrons.end(), [](const TLorentzVector& a,
                     const TLorentzVector& b) {
99                  return a.Pt() > b.Pt();
100             });
101             if (electrons.size() >= 2) {
```

```
102                 TLorentzVector invMassElectron = electrons[0] + electrons[1];
103                 hInvariantMassElectron->Fill(invMassElectron.M());
104             }
105         }
106 
107         if (Muon_pt.GetSize() >= 2) {
108             std::vector<TLorentzVector> muons;
109             for (int i = 0; i < Muon_pt.GetSize(); ++i) {
110                 TLorentzVector muon;
111                 muon.SetPtEtaPhiM(Muon_pt[i], Muon_eta[i], Muon_phi[i], 0.105658); //
                        Massa do m on
112                 muons.push_back(muon);
113             }
114             std::sort(muons.begin(), muons.end(), [](const TLorentzVector& a, const
                    TLorentzVector& b) {
115                 return a.Pt() > b.Pt();
116             });
117             if (muons.size() >= 2) {
118                 TLorentzVector invMassMuon = muons[0] + muons[1];
119                 hInvariantMassMuon->Fill(invMassMuon.M());
120             }
121         }
122 
123         if (Tau_pt.GetSize() >= 2) {
124             std::vector<TLorentzVector> taus;
125             for (int i = 0; i < Tau_pt.GetSize(); ++i) {
126                 TLorentzVector tau;
127                 tau.SetPtEtaPhiM(Tau_pt[i], Tau_eta[i], Tau_phi[i], 1.77682); //
                        Massa do tau
128                 taus.push_back(tau);
129             }
130             std::sort(taus.begin(), taus.end(), [](const TLorentzVector& a, const
                    TLorentzVector& b) {
131                 return a.Pt() > b.Pt();
132             });
133             if (taus.size() >= 2) {
134                 TLorentzVector invMassTau = taus[0] + taus[1];
135                 hInvariantMassTau->Fill(invMassTau.M());
136             }
137         }
138     }
139 
140     // Criando canvas e plotando histogramas
141     //Plot dos momentos
142 TCanvas *c1 = new TCanvas("c1","Electrons distribution",1920,1080);
143     c1->Divide(2, 2);
144     c1->cd(1); hElectronPt->Draw();
145     c1->cd(2); hMuonPt->Draw();
146     c1->cd(3); hTauPt->Draw();
147     c1->cd(4); hJetPt->Draw();
148     c1->SaveAs("p_transversal.png");
149 
150     // Plot de eta
151     TCanvas *c2 = new TCanvas("c2", "Muons distribuction", 1920, 1080);
152     c2->Divide(2, 2);
153     c2->cd(1); hElectronEta->Draw();
154     c2->cd(2); hMuonEta->Draw();
155     c2->cd(3); hTauEta->Draw();
156     c2->cd(4); hJetEta->Draw();
157     c2->SaveAs("eta.png");
158 
159     // Plot Phi
160     TCanvas *c3 = new TCanvas("c3", "Taus distribuction", 1920, 1080);
```

```
161     c3->Divide(2, 2);
162     c3->cd(1); hElectronPhi->Draw();
163     c3->cd(2); hMuonPhi->Draw();
164     c3->cd(3); hTauPhi->Draw();
165     c3->cd(4); hJetPhi->Draw();
166     c3->SaveAs("phi.png");
167 
168     // Plot Massa invariante
169     TCanvas *c4 = new TCanvas("c4", "Jets distribution", 1920, 1080);
170     c4->Divide(2, 2);
171     c4->cd(1); hInvariantMassElectron->Draw();
172     c4->cd(2); hInvariantMassMuon->Draw();
173     c4->cd(3); hInvariantMassTau->Draw();
174     c4->SaveAs("massa_invariante.png");
175 
176 
177 }
```

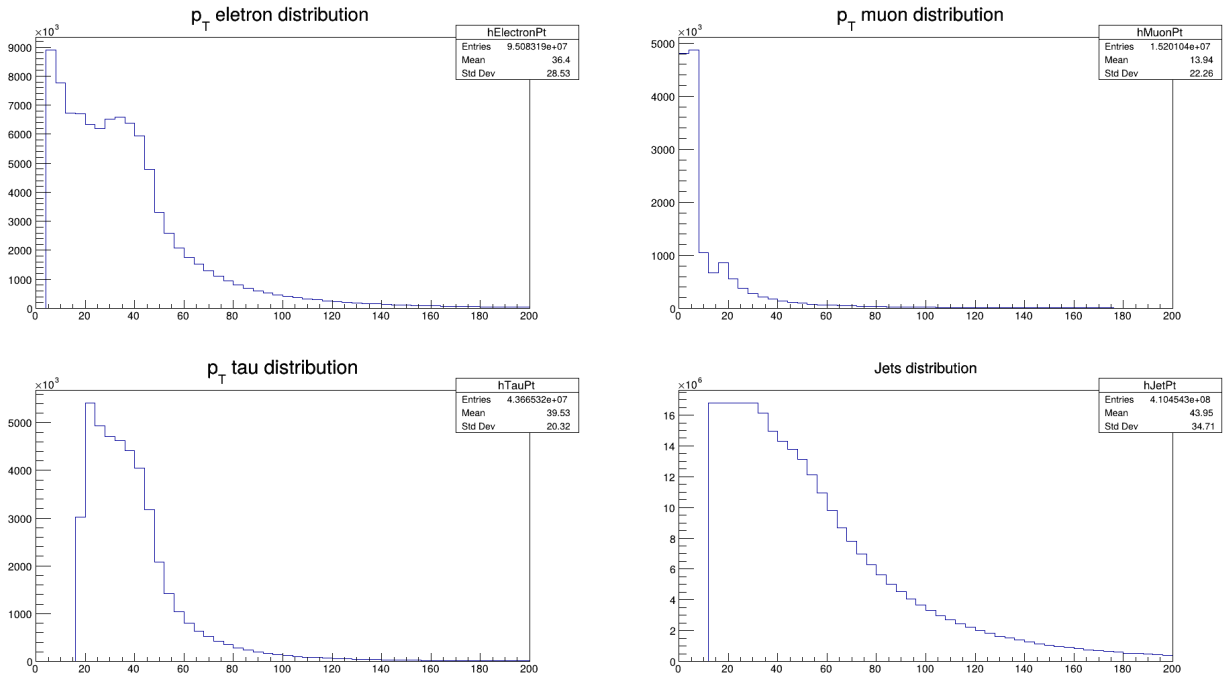

Figura 1: Momento transversal das partículas analisadas.

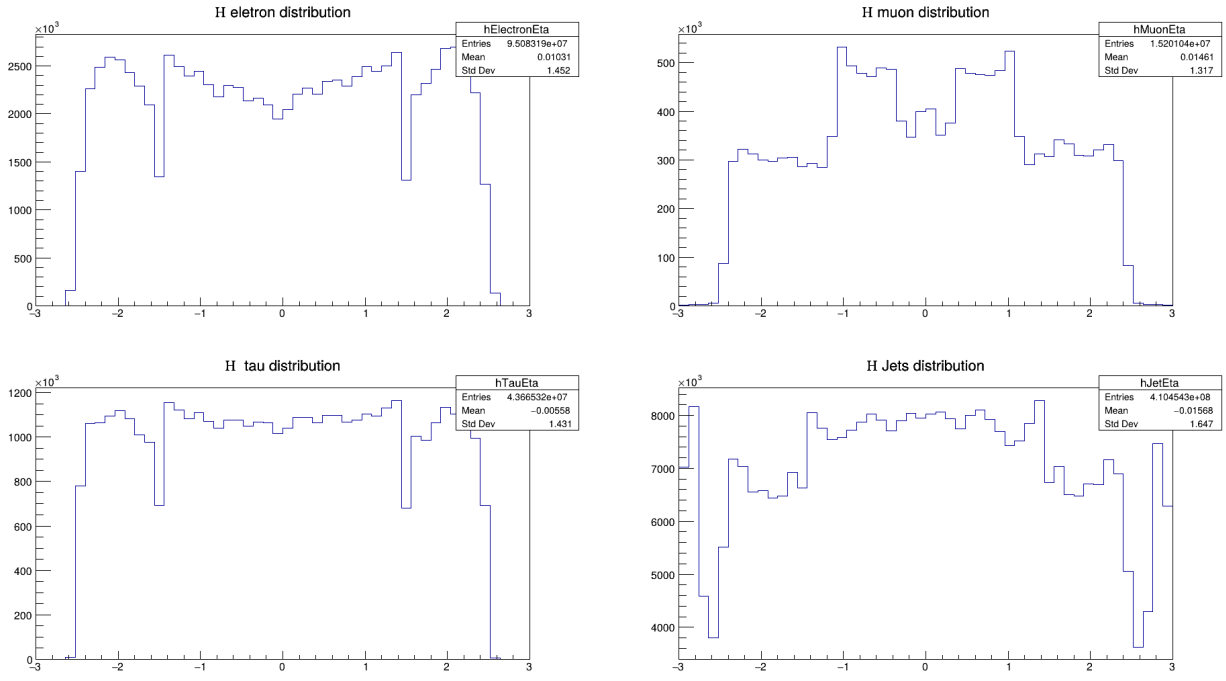

Figura 2: η das partículas analisadas.

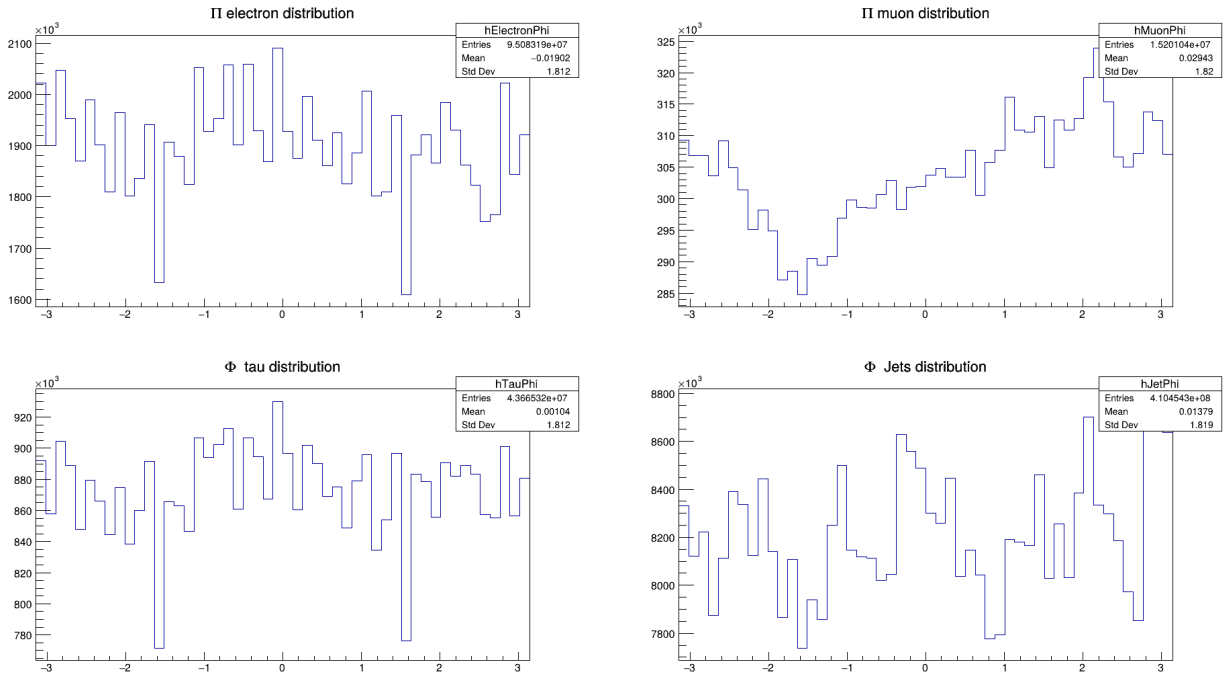

Figura 3: φ das partículas analisadas.

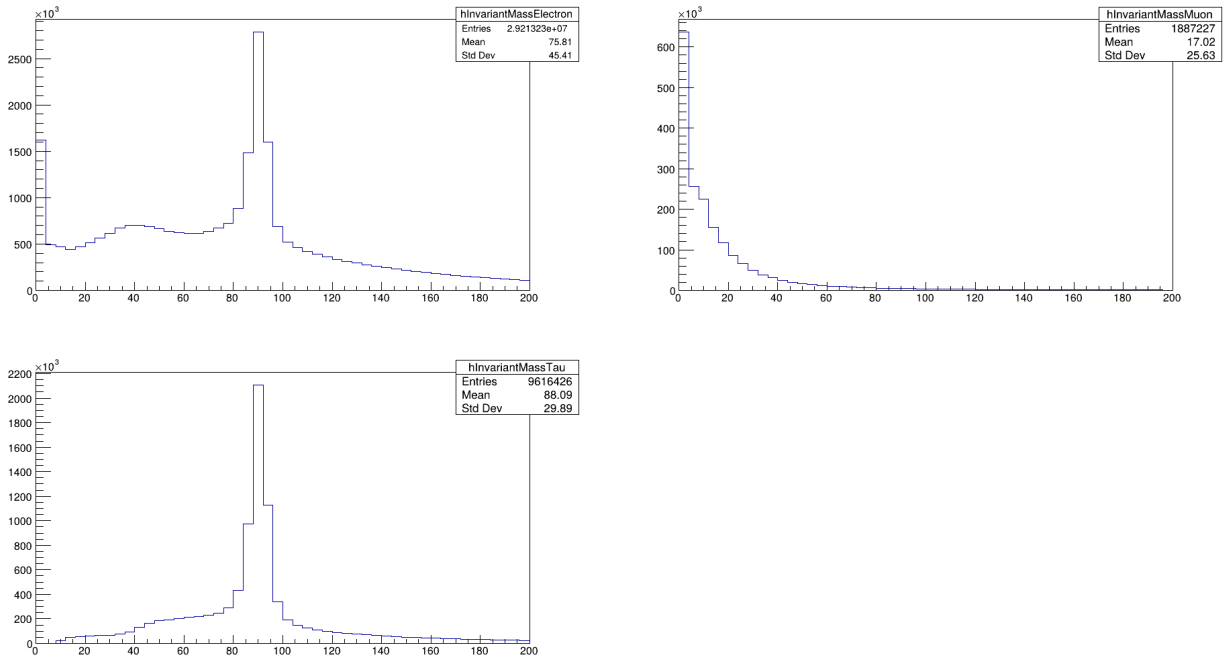

Figura 4: Massa invariante das partículas analisadas.