Introdução à Análise de dados em FAE **(Data: 24/11/20)**

Relatório Pythia8.

Professores: Sandro Fonseca, Sheila Mara, Eliza Melo. *Name:* João Pedro Gomes Pinheiro (`jgomespi`).

Exercícios baseados no Tutorial sobre o Pythia8.

Os exercícios 1, 2 e 3 estão disponíveis no `notebook jupyter` *Lista-MonteCarlo.ipynb*, no repositório `GitHub`: jgomespi/MC-Pythia8. Os demais exercícios são descritos aqui, com todos os programas utilizados no mesmo repositório `GitHub` do diretório `ex1`.

Para executar os programas descritos abaixo, você deve ter os arquivos `pythiaLinkdef.h` e `pythiaROOT.h` dentro do diretório `ex1`. Entre neste diretório e execute:

```
rootcint -f pythiaDict.cxx -c -I$PYTHIA8/include pythiaROOT.h   pythiaLinkdef.h
```

Em que `$PYTHIA8` é o *path* para o diretório de instalação do Pythia. Posteriormente, execute:

```
g++ -o exN exN.C pythiaDict.cxx -I$PYTHIA8/include `root-config --cflags  --glibs` -
    lEG -lEGPythia8 -L$PYTHIA8/lib -lpythia8
```

Substitue `exN.C` pelo arquivo desejado.

Tive um problema ao encontrar a biblioteca `libpythia8.so`.

O problema foi solucionado copiando forçosamente esta biblioteca para o diretório `ex1` através de:

```
cp -f $PYTHIA8/lib/libpythia8.so .
```

Feito isso, todos os comandos executaram corretamente.

Exercício 4:

Utilizando o esqueleto simplificado, modifique o *script* para gerar alguns eventos no ambiente do LHC ($\sqrt{s} = 14$ TeV e $\sqrt{s} = 7$ TeV).

a) Os processos a serem considerados são: `HardQCD:qqbar2bbbar`, `HardQCD:gg2bbbar`, `HardQCD:gg2qqbarg`, `HardQCD:qqbar2qqbargDiff` **e** `HardQCD:qqbar2qqbargSame`**. Também permita todos os sabores de *quark* utilizando** `HardQCD:nQuark New = 5`**.**

b) Como podemos selecionar apenas eventos com quarks b? E como podemos selecionar apenas mésons B?

c) Salve as informações das partículas numa `tree`.

Vamos aproveitar o esqueleto `ex1.C`, fornecido pela professora Sheila, e adaptá-lo para nosso caso. Os processos que serão considerados neste caso são setados a partir de:

```
// Process to be considered
pythia.readString("HardQCD:qqbar2bbbar = on");
pythia.readString("HardQCD:gg2bbbar = on");
pythia.readString("HardQCD:gg2qqbarg = on");
pythia.readString("HardQCD:qqbar2qqbargDiff = on");
pythia.readString("HardQCD:qqbar2qqbargSame = on");
pythia.readString("HardQCD:nQuarkNew = 5");
```

Estamos considerando uma colisão *pp* a uma energia de 7 TeV e 14 TeV, portanto, os comandos utilizados serão:

```
// Colision features
pythia.readString("Beams:eCM = 7000."); // try to 1400.
pythia.readString("Beams:idA = 2212");
pythia.readString("Beams:idB = 2212");
```

Posteriormente, inicializamos o Pythia e declaramos os arquivos e `trees` utilizadas. Também são declaradas as variáveis que vão contar quantos eventos possuem um quark b no processo duro (`bquark`) e quantos eventos possuem um méson B no estado final (`Bmeson`).

```
pythia.init();
// Declaring files and trees
TFile *file = TFile::Open("ex3.root","recreate");
Event *event = &pythia.event;
TTree *T = new TTree("RunI","RunI Tree");
T->Branch("event",&event);

int bquark = 0; // variable to count how many events has a b quarks in hard process
int Bmeson = 0; // variable to count B mesons in final state
```

Faremos as contagens separadamente. Em ambos os casos faremos um loop sob 10.000 eventos. Utilizamos os métodos `id()` (deve ser igual a 5) e `status()` (deve estar entre 21 e 29) para garantir que selecionamos um quark b produzidos durante o processo duro. Ao encontrar uma partícula com estas especificações, paramos o loop das partículas daquele evento, pois basta um quark b para selecionarmos o evento.

```
// Loop over events:
for (int iEvent = 0; iEvent < 10000; ++iEvent) {
        if (!pythia.next()) continue;
        // Loop over particles
        for (int i = 0; i < pythia.event.size(); ++i) {
                if (pythia.event[i].id() == 5 &&
                   std::abs(pythia.event[i].status()) >= 21 &&
                   std::abs(pythia.event[i].status()) <=29){
                        ++bquark;
                        break;
                }
        }
}
```

Em seguida, fazemos outro loop procurando por mésons B no estado final. Vamos selecionar eventos com B^0, B^+ e B^-, ou seja, cujo módulo do `id()` é igual a 511 ou 521. Utilizamos o método `isFinal()` para selecionar aqueles de estado final.

Ao fim, preenchemos a tree com as informações das partículas.

```
// Loop over events
for (int iEvent = 0; iEvent < 10000; ++iEvent) {
        if (!pythia.next()) continue;
        // Loop over particles
        for (int i = 0; i < pythia.event.size(); ++i) {
                if (std::abs(pythia.event[i].id()) == 511 || std::abs(pythia.event[i
                    ].id()) == 521){
                        if (pythia.event[i].isFinal() == 0){
                                ++Bmeson;
                                break;
                        }
                }
        }
        T->Fill();
}
```

Ao final, escrevemos informações sobre a estatística da geração e escrevemos a tree e o arquivo. Além disso, será escrito na tela o número de eventos com quark b no processo duro e o número de eventos com méson B no estado final.

```
pythia.stat();
cout << "Quantidade de eventos com quark b: "<< bquark << endl;
cout << "Quantidade de eventos com Meson B: "<< Bmeson << endl;
T->Print();
T->Write();
delete file;
return 0;
```

Rodamos 10.000 eventos na energia $\sqrt{s} = 7$ TeV, encontrando 8.992 eventos com quark b no processo duro e 8.901 eventos com méson B (B^0, B^+ ou B^-) no estado final. Para a energia $\sqrt{s} = 7$ TeV, encontramos 8.727 eventos com quark b no processo duro e 8.606 eventos com méson B (B^0, B^+ ou B^-) no estado final.

Exercício 5:

Utilize a aniquilação e_+e_- como um ambiente limpo para o estudo da radiação QCD de estado final. Especificamente, estude como o número médio de partons cresce com a energia do centro de massa. Além disso, também verifique o quão bem (ou mal) o número de partons é descrito pela distribuição de Poisson.

Inicialmente, foi declarada a variável `pythia` e foram setadas as partículas do feixe para elétron (`id`=11) e pósitron (`id`=-11). A energia de centro de massa será, a princípio, escolhida como 50 GeV:

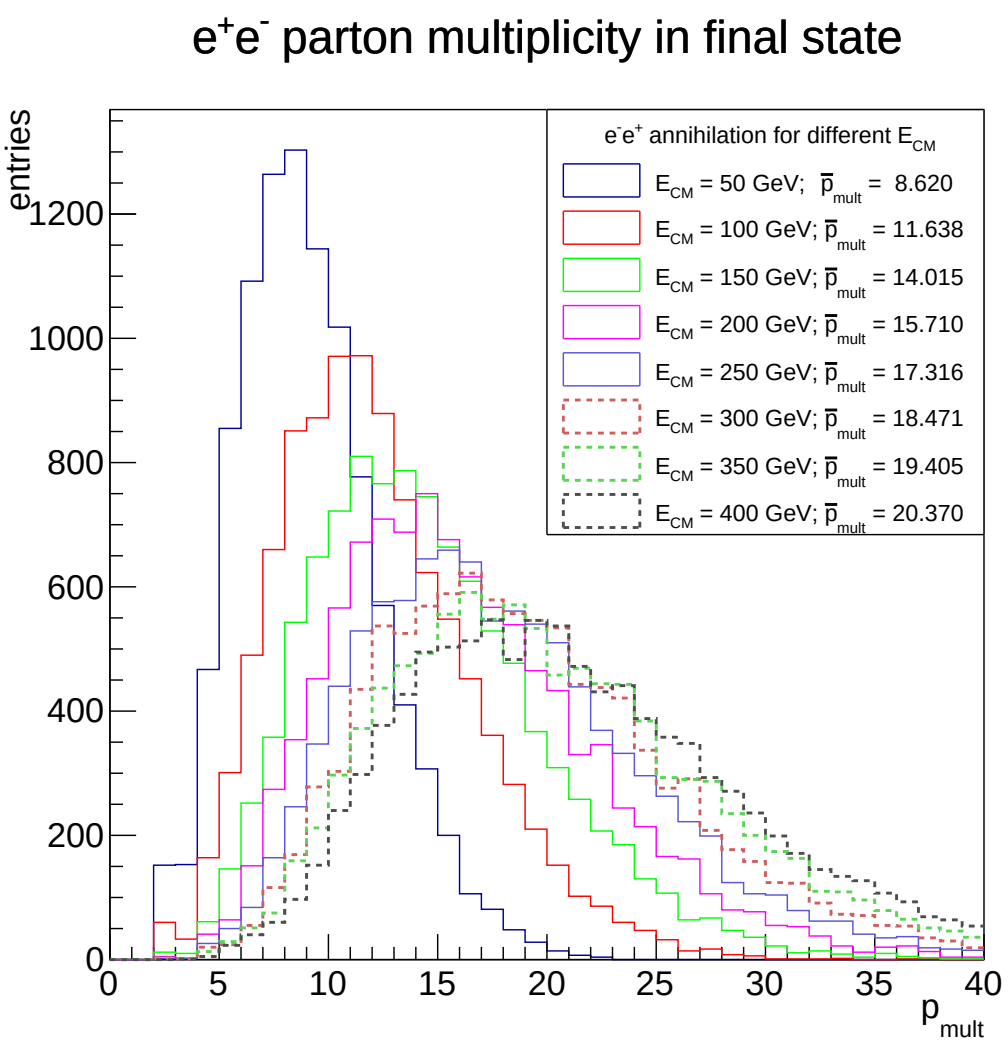

Figura 1: Multiplicidade de pártons no estado final da aniquilação e_+e_- para diferentes valores de E_{cm}. O número médio de partons de estado final cresce à medida que a energia do centro de massa cresce.

```
// Set the beam particles id
Pythia pythia;
pythia.readString("Beams:eCM = 50.");
pythia.readString("Beams:idA = 11");
pythia.readString("Beams:idB = -11");
```

Posteriormente, foram setadas as opções sugeridas no enunciado do exercício e inicializamos o `pythia`:

```
// Options
pythia.readString("WeakSingleBoson:ffbar2gmZ = on");
pythia.readString("PDF:lepton = off");
pythia.readString("HadronLevel:all = off");
pythia.readString("23:onMode = off");
```

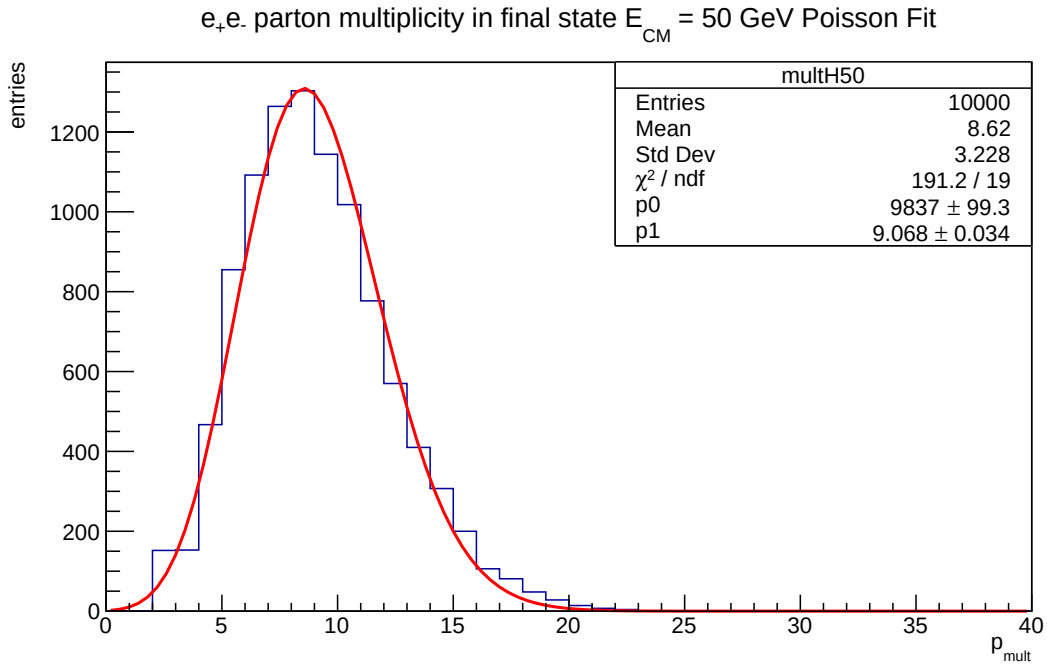

Figura 2: Multiplicidade de partons no estado final da aniquilação e_+e_- para $E_{cm} = 50$ GeV. Ajustado a uma distribuição poissoniana.

```
pythia.readString("23:onIfAny = 1 2 3 4 5");

// Initialize
pythia.init();
```

A seguir, declaramos o arquivo onde serão armazenados os histogramas para diferentes valores de energia de centro de massa. A opção `update` vai garantir que o arquivo não seja sobrescrito ao rodarmos o programa mais de uma vez. Declaramos também o histograma que vai armazenar a quantidade de pártons de estado final dos eventos. Posteriormente, declaramos a variável `event` para o loop de eventos. A variável `mult` vai armazenar a multiplicidade de partons de estado final nos eventos, ou seja, quantos partons de estado final existem em cada evento.

```
// Declaring the file where the hist will be written
// Update option will write the hists to the same file
TFile *file = TFile::Open("ex5.root","update");
// Declaring the histogram where the parton multiplicity will be filled. Its name has
    to change every time the program is run.
TH1F *multH = new TH1F("multH50","Parton Multiplicity", 40, 0, 40);
Event *event = &pythia.event;
// The variable that will count the parton multiplicity
int mult;
```

Fazemos o loop sob os eventos e sob as partículas de cada evento. Vamos gerar 10.000 eventos para termos uma boa estatística. Contamos quantos pártons de estado final temos utilizando os métodos `isFinal()` e `isParton()`.

```
// Begin event loop. Generate event. Skip if error. List first one.
for (int iEvent = 0; iEvent < 10000; ++iEvent) {
        if (!pythia.next()) continue;
        // Set the multiplicity to 0 at the begin of the event
        mult = 0;
        // Find number of all final charged particles and fill histogram.
        for (int i = 0; i < pythia.event.size(); ++i){
                // Select only final state partons
                if (pythia.event[i].isFinal() && pythia.event[i].isParton()) ++mult;
        }
        multH->Fill( mult );
```

```
        // End of event loop. Statistics. Histogram. Done.
}
```

Porfim, desenhamos informações sobre a geração na tela, desenhamos e salvamos o histograma com o número de pártons em cada um dos 10.000 eventos gerados, tanto no arquivo `ex5.root` quanto em formato `.pdf`.

```
// Print stat info
pythia.stat();
// Creat a canvas to draw the mult hist
TCanvas* c1 = new TCanvas("c1","Parton Multiplicity",800,800);
multH->Draw();
// Save in pdf format
c1->SaveAs("multH_50.pdf","pdf");
// Write in the file
multH->Write();

delete file;
return 0;
```

Estes comandos estão escritos no arquivo `ex5.C`, presente no repositório `GitHub`. Rodamos este programa para os valores de energia do centro de massa: $E_{cm} = 50$, 100, 150, 200, 250, 300, 350 e 400 GeV. Escrevemos o programa `read_hist.C`, que lê os histogramas armazenados em `ex5.root` e os desenha superpostos e com legenda. Podemos observar os plots da multiplicidade de pártons no estado final da aniquilação e_+e_- pode ser visto na Figura 1. Observamos que $\bar{p}_{mult}$ cresce à medida que E_{cm} cresce.

Além disso, foi escolhido o histograma para $E_{cm} = 50$ GeV e ajustamos uma distribuição de Poisson, como pode ser visto na Figura 2. O χ^2 é da mesma ordem de grandeza que o número de graus de liberdade da distribuição, portanto podemos dizer que há um bom ajuste.

Exercício 6:

Estude as propriedades de eventos *minimum-bias*, por exemplo, plote as distribuições $\frac{dn}{d\eta}$, $\frac{dn}{dp_T}$ e $\langle p_T \rangle$ das partículas carregadas de estado final no LHC. Estude como essas distribuições mudam se as MPIs são desligadas.
Alguns comandos úteis são: `SoftQCD:minBias = on` **e** `PartonLevel:MI = off`**.**

Neste caso, vamos mostrar apenas os principais comandos, o restante do programa pode ser visto no arquivo `ex6.C` no repositório `GitHub`.

Estamos considerando colisões pp a uma energia de centro de massa $\sqrt{s} = 14$ TeV (cenário do Run II do LHC), portanto:

```
// pp colision at ecm = 14 TeV
pythia.readString("Beams:eCM = 14000.");
pythia.readString("Beams:idA = 2212");
pythia.readString("Beams:idB = 2212");
```

Posteriormente, setamos as opções sugeridas. A primeira liga os eventos *minimum bias*, a segunda liga (ou desliga) as interações múltiplas de pártons. Esta última será alterada para estudar qual a diferença observada nos plots de $\frac{dN}{dp_T}$ em função de p_T e $\frac{dN}{d\eta}$ em função de η.

```
// Options
pythia.readString("SoftQCD:nonDiffractive = on");
pythia.readString("PartonLevel:MPI = on");
```

A seguir, abrimos o arquivo de saída e declaramos os histogramas e variáveis utilizadas:

```
// Declaring the file where the hist will be written
// Update option will write the hists to the same file
TFile *file = TFile::Open("ex6.root","update");
// Declaring the histogram of pT and eta. Its name has to change every time the
    program run.
TH1F *etaHist = new TH1F("etaMPIHist",  "dN/d#eta x #eta", 100, -4, 4);
TH1F *pTHist = new TH1F("pTMPIHist",    "dN/dp_{T} x p_{T}", 100, 0, 100);
Double_t px_part, py_part, pz_part, pT_part, theta_part, eta_part;
```

Foram gerados 1 milhão de eventos e calculadas as variáveis p_T e η, a partir de p_x, p_y e p_z, segundo as fórmulas abaixo:

$$p_T = \sqrt{p_x^2 + p_y^2}$$

$$\theta = \arctan\left(\frac{p_T}{p_z}\right)$$

$$\eta = -\ln\left(\tan\left(\frac{\theta}{2}\right)\right)$$

Selecionamos apenas partículas carregadas e de estado final, utilizando os métodos `isFinal()` e `isCharged()`. Preenchemos os histogramas com as variáveis p_T e η.

```
int nev = 1000000; // number of events
// Begin event loop. Generate event. Skip if error. List first one.
for (int iEvent = 0; iEvent < nev; ++iEvent) {
        if (!pythia.next()) continue;
        // Begin particle loop
        for (int i = 0; i < pythia.event.size(); ++i){
                // Get px, py e pz
                px_part = pythia.event[i].px();
                py_part = pythia.event[i].py();
                pz_part = pythia.event[i].pz();
                // Calculating pT
                pT_part = pow(px_part*px_part+py_part*py_part,0.5);
                // Verify if pT in nonzero
                if (pT_part == 0) continue;
                // Calculating theta
                theta_part = atan(pT_part/pz_part);
                // Calculating eta
                eta_part;
                if (theta_part >=0) eta_part = -log(tan(theta_part/2));
                if (theta_part < 0) eta_part = log(tan(-1*theta_part/2));
                // Select final and charged particles
                if (pythia.event[i].isFinal() && pythia.event[i].isCharged()) {
                        // Fill pt and eta histograms
                        if (pT_part != 0) pTHist->Fill(pT_part);
                        etaHist->Fill(eta_part);
                }
        }
// End of event loop. Statistics. Histogram. Done.
}
```

Porfim, simplesmente escrevemos os histogramas no arquivo de saída e fechamos o arquivo:

```
// Print stat info
pythia.stat();
// Creat a canvas to draw the mult hist
TCanvas* c1 = new TCanvas("c1","",800,800);
c1->Divide(1, 2);
c1->cd(1);
etaHist->SetXTitle("#eta");
etaHist->SetYTitle("dN/d#eta");
etaHist->Draw();

c1->cd(2);
gPad->SetLogy();
pTHist->SetXTitle("p_{t} (GeV)");
pTHist->SetYTitle("dN/dp_{T} (GeV^{-1})");
pTHist->Draw();
// Save in pdf format (this change every time the program run)
c1->SaveAs("minimumbias_withMPI.pdf","pdf");
// Write in the file
```

```
19 pTHist->Write();
20 etaHist->Write();
21
22 delete file;
23 return 0;
```

Rodamos o arquivo `ex6.C`, descrito acima, para a opção `PartonLevel:MPI = on`, gerando os histogramas `etaMPIHist` e `pTMPIHist`. Posteriormente, rodamos para `PartonLevel:MPI = off`, gerando os histogramas `etanoMPIHist` e `pTnoMPIHist`. Estes histogramas estão escritos no arquivo e escrevendo no arquivo `ex6.root`. Utilizamos o programa read_hist2.C para ler este arquivo, sobrepor e normalizar as distribuições de p_T e η. O resultado está na Figura 3. Conforme o esperado, existem muito mais partículas produzidas ao permitirmos interações múltiplas de pártons, por isso apresentamos as distribuições normalizadas. Enquanto $\frac{dN}{dp_T}$ não apresenta diferença apreciativa na sua forma, a distribuição de $\frac{dN}{d\eta}$ nos mostra que, ao permitirmos interações múltiplas de pártons, observamos mais partículas produzidas na região frontal (valores maiores de $|\eta|$).

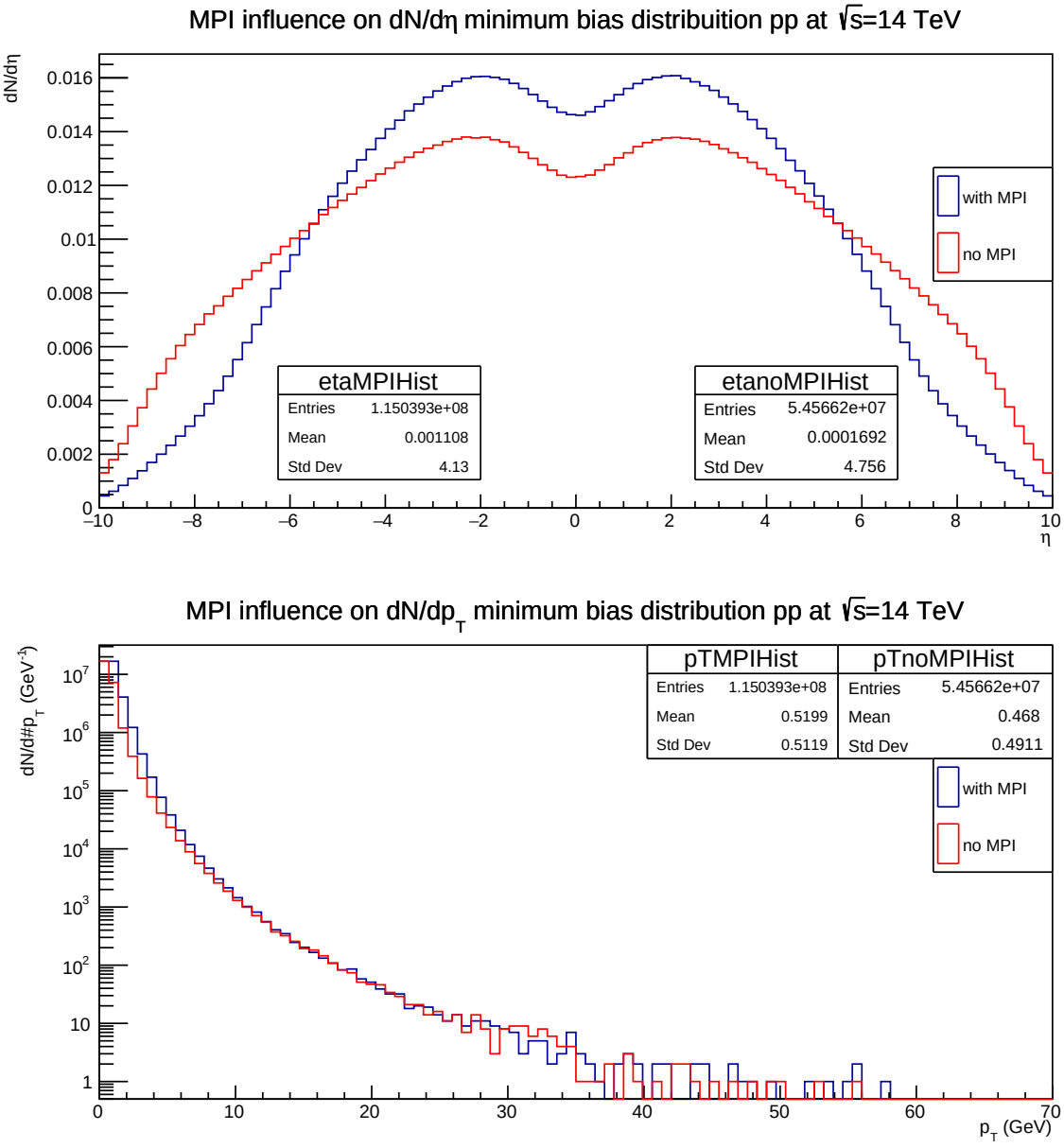

Figura 3: Distribuições normalizadas de $\frac{dN}{d\eta}$ e $\frac{dN}{dp_T}$ para 1 milhão de eventos gerados com (em azul) e sem (em vermelho) interação múltipla de pártons, com $\sqrt{s} = 14$ TeV.