

Monte Carlo - Pythia8

Professores: Sandro Fonseca, Sheila Mello, 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: MC-Pythia`. Os demais exercícios são desenvolvidos no presente relatório. Todos os programas aqui utilizados se encontram no mesmo repositório `GitHub`, no 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:

```
1 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:

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

Substitua `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:

```
1 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:qqbar2bbbbbar`, `HardQCD:gg2bbbbbar`, `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 comandos descritos aqui se encontram no arquivo `ex4.C`.

Os processos que serão considerados neste caso são setados a partir de:

```
1 // Process to be considered
2 pythia.readString("HardQCD:qqbar2bbbbbar = on");
3 pythia.readString("HardQCD:gg2bbbbbar = on");
4 pythia.readString("HardQCD:gg2qqbarg = on");
5 pythia.readString("HardQCD:qqbar2qqbargDiff = on");
6 pythia.readString("HardQCD:qqbar2qqbargSame = on");
7 pythia.readString("HardQCD:nQuarkNew = 5");
```

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

```
1 // Collision features
2 pythia.readString("Beams:eCM = 7000."); // try to 14000.
3 pythia.readString("Beams:idA = 2212");
4 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`).

```

1  pythia.init();
2  // Declaring files and trees
3  TFile *file = TFile::Open("ex3.root","recreate");
4  Event *event = &pythia.event;
5  TTree *T = new TTree("RunI","RunI Tree");
6  T->Branch("event",&event);
7
8  int bquark = 0; // variable to count how many events has a b quarks in hard process
9  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.

```

1  // Loop over events:
2  for (int iEvent = 0; iEvent < 10000; ++iEvent) {
3      if (!pythia.next()) continue;
4      // Loop over particles
5      for (int i = 0; i < pythia.event.size(); ++i) {
6          if (pythia.event[i].id() == 5 &&
7              std::abs(pythia.event[i].status()) >= 21 &&
8              std::abs(pythia.event[i].status()) <= 29){
9              ++bquark;
10             break;
11         }
12     }
13 }

```

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.

```

1  // Loop over events
2  for (int iEvent = 0; iEvent < 10000; ++iEvent) {
3      if (!pythia.next()) continue;
4      // Loop over particles
5      for (int i = 0; i < pythia.event.size(); ++i) {
6          if (std::abs(pythia.event[i].id()) == 511 || std::abs(pythia.event[i]
7              .id()) == 521){
8              if (pythia.event[i].isFinal() == 0){
9              ++Bmeson;
10             break;
11         }
12     }
13     T->Fill();
14 }

```

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.

```

1  pythia.stat();
2  cout << "Quantidade de eventos com quark b: " << bquark << endl;
3  cout << "Quantidade de eventos com Meson B: " << Bmeson << endl;
4  T->Print();
5  T->Write();
6  delete file;
7  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:

e^+e^- parton multiplicity in final state

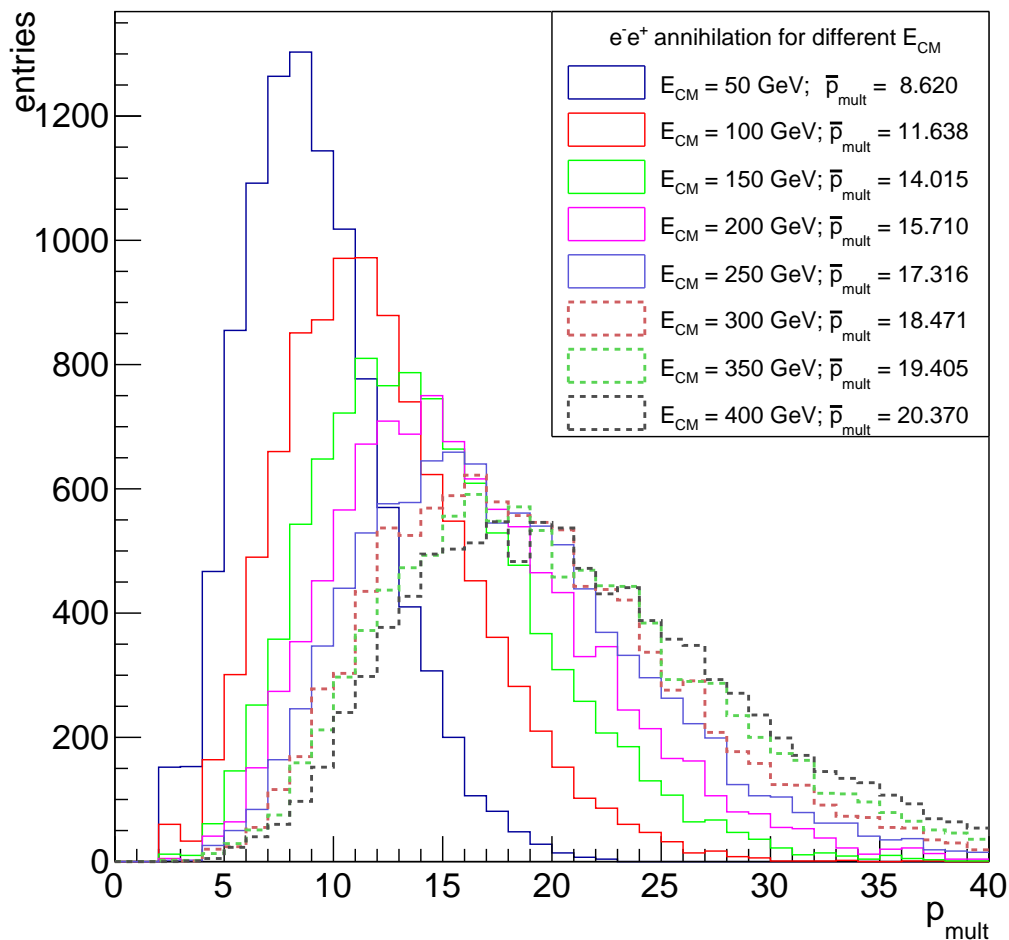


Figura 1: Multiplicidade de pártons no estado final da aniquilação e^+e^- para diferentes valores de E_{CM} . O número médio de partons de estado final cresce à medida que a energia do centro de massa cresce.

```

1 // Set the beam particles id
2 Pythia pythia;
3 pythia.readString("Beams:eCM = 50.");
4 pythia.readString("Beams:idA = 11");
5 pythia.readString("Beams:idB = -11");

```

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

```

1 // Options
2 pythia.readString("WeakSingleBoson:ffbar2gmZ = on");
3 pythia.readString("PDF:lepton = off");
4 pythia.readString("HadronLevel:all = off");
5 pythia.readString("23:onMode = off");

```

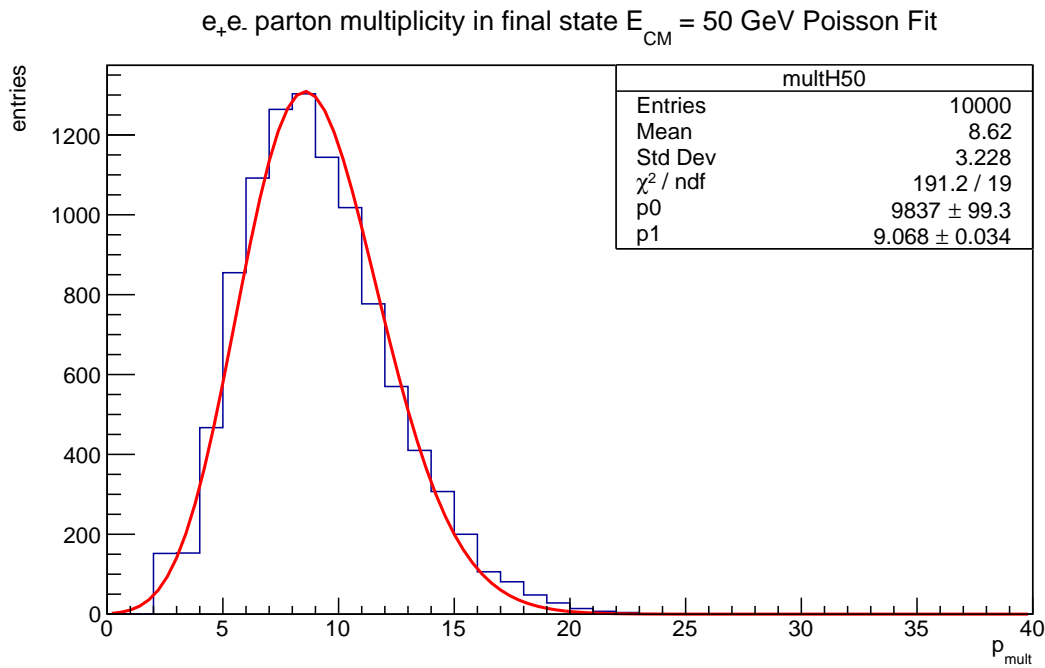


Figura 2: Multiplicidade de partons no estado final da aniquilação e^+e^- para $E_{CM} = 50$ GeV. Ajustado a uma distribuição poissoniana.

```

6  pythia.readString("23:onIfAny = 1 2 3 4 5");
7
8  // Initialize
9  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.

```

1  // Declaring the file where the hist will be written
2  // Update option will write the hists to the same file
3  TFile *file = TFile::Open("ex5.root","update");
4  // Declaring the histogram where the parton multiplicity will be filled. Its name has
   to change every time the program is run.
5  TH1F *multH = new TH1F("multH50","Parton Multiplicity", 40, 0, 40);
6  Event *event = &pythia.event;
7  // The variable that will count the parton multiplicity
8  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()`.

```

1  // Begin event loop. Generate event. Skip if error. List first one.
2  for (int iEvent = 0; iEvent < 10000; ++iEvent) {
3      if (!pythia.next()) continue;
4      // Set the multiplicity to 0 at the begin of the event
5      mult = 0;
6      // Find number of all final charged particles and fill histogram.
7      for (int i = 0; i < pythia.event.size(); ++i){
8          // Select only final state partons
9          if (pythia.event[i].isFinal() && pythia.event[i].isParton()) ++mult;
10     }
11     multH->Fill( mult );

```

```

12      // End of event loop. Statistics. Histogram. Done.
13  }

```

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`.

```

1  // Print stat info
2  pythia.stat();
3  // Creat a canvas to draw the mult hist
4  TCanvas* c1 = new TCanvas("c1","Parton Multiplicity",800,800);
5  multH->Draw();
6  // Save in pdf format
7  c1->SaveAs("multH_50.pdf","pdf");
8  // Write in the file
9  multH->Write();
10
11 delete file;
12 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:

```

1  // pp colision at ecm = 14 TeV
2  pythia.readString("Beams:eCM = 14000.");
3  pythia.readString("Beams:idA = 2212");
4  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 .

```

1  // Options
2  pythia.readString("SoftQCD:nonDiffraction = on");
3  pythia.readString("PartonLevel:MPI = on");

```

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

```

1  // Declaring the file where the hist will be written
2  // Update option will write the hists to the same file
3  TFile *file = TFile::Open("ex6.root","update");
4  // Declaring the histogram of pT and eta. Its name has to change every time the
   program run.
5  TH1F *etaHist = new TH1F("etaMPIHist", "dN/d#eta x #eta", 100, -4, 4);
6  TH1F *pTHist = new TH1F("pTMPIHist", "dN/dp_{T} x p_{T}", 100, 0, 100);
7  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 .

```

1  int nev = 1000000; // number of events
2  // Begin event loop. Generate event. Skip if error. List first one.
3  for (int iEvent = 0; iEvent < nev; ++iEvent) {
4      if (!pythia.next()) continue;
5      // Begin particle loop
6      for (int i = 0; i < pythia.event.size(); ++i){
7          // Get px, py e pz
8          px_part = pythia.event[i].px();
9          py_part = pythia.event[i].py();
10         pz_part = pythia.event[i].pz();
11         // Calculating pT
12         pT_part = pow(px_part*px_part+py_part*py_part,0.5);
13         // Verify if pT in nonzero
14         if (pT_part == 0) continue;
15         // Calculating theta
16         theta_part = atan(pT_part/pz_part);
17         // Calculating eta
18         eta_part;
19         if (theta_part >=0) eta_part = -log(tan(theta_part/2));
20         if (theta_part < 0) eta_part = log(tan(-1*theta_part/2));
21         // Select final and charged particles
22         if (pythia.event[i].isFinal() && pythia.event[i].isCharged()) {
23             // Fill pt and eta histograms
24             if (pT_part != 0) pTHist->Fill(pT_part);
25             etaHist->Fill(eta_part);
26         }
27     }
28 // End of event loop. Statistics. Histogram. Done.
29 }

```

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

```

1  // Print stat info
2  pythia.stat();
3  // Creat a canvas to draw the mult hist
4  TCanvas* c1 = new TCanvas("c1","",800,800);
5  c1->Divide(1, 2);
6  c1->cd(1);
7  etaHist->SetXTitle("#eta");
8  etaHist->SetYTitle("dN/d#eta");
9  etaHist->Draw();
10
11 c1->cd(2);
12 gPad->SetLogy();
13 pTHist->SetXTitle("p_{t} (GeV)");
14 pTHist->SetYTitle("dN/dp_{T} (GeV^{-1})");
15 pTHist->Draw();
16 // Save in pdf format (this change every time the program run)
17 c1->SaveAs("minimumbias_withMPI.pdf","pdf");
18 // 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 $|\eta|$).

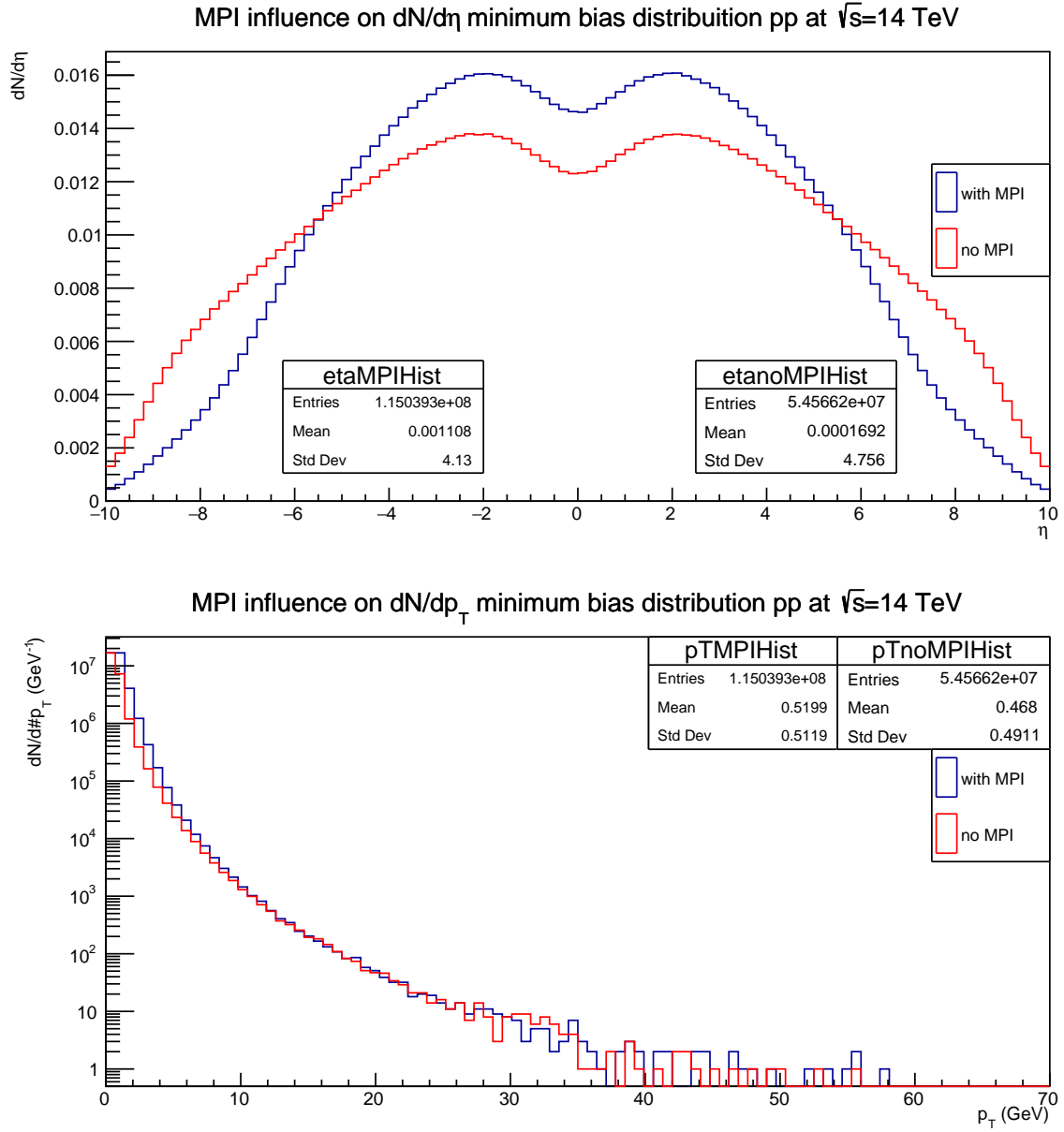


Figura 3: Distribuições normalizadas de $\frac{dN}{d\eta}$ e $\frac{dN}{dp_T}$ para 1 milhão de eventos gerados com (em azul) e sem (em vermelho) interação múltipla de pártons, com $\sqrt{s} = 14$ TeV.