Introdução à Análise de dados em FAE **(Data: 24/11/2020)**

Relatório -Monte Carlo/pythia8

Professores: Sandro Fonseca, Sheila Amaral, Eliza Melo *Nome:* Silas Santos de Jesus

Este relatório contém as reespotas dos exerícios numerados de 1 até 6 do trabalho de Monte Carlo/pythia8. Os exercícios 1, 2a, 2b e 3 foram executados utilizando o método de Monte Carlo e os demais pythia8/Monte Carlo.

Aqui se encontram as explicações detalhadas dos códigos criados para cada exercício. Esses códigos estão presentes neste repositório: https://github.com/Silas-SJ/Monte-Carlos-Pythia8

Exercício 1

Write a code that estimate the area of a unit disk using the hit-or-miss Monte Carlo method. We know the radius of the unit disk is 1 thus the unit circle is inscribed within a square of length 2. Tip: generate samples within this square and count the number of points falling within the disk. To test whether the point is inside (hit) or outside (miss) the disk, we simply need to measure the distance of the sample from the origin (the center of the unit disk) and check whether this distance is smaller (or equal) than the disk radius (which is equal to 1 for a unit disk).

O nome do arquivo com o código deste exercício é: exer1.cc

Podemos começar parametrizando o exercício: temos duas figuras geométricas que podemos colocar com mesmo centro, ou seja, o circulo estará inscrito no quadrado. Vamos considerar a razão entre as áreas do circulo e do quadrado:

$$\frac{A_c}{A_q} = \frac{\pi r^2}{4} = \frac{\pi.1^2}{4} = \frac{\pi}{4}$$

A área limite a ser considerada é a área do quadrado nos intervalos: $(-1 \leq x \leq 1)$ e $(-1 \leq y \leq 1)$.

A área do circulo é limitada por : $r = \sqrt{x^2 + y^2}$

A condição é que um ponto pode estar dentro ou fora do circulo, se $r \leq 1$ o ponto está dentro, do contrário, fora.

Vamos começar escolhendo as bibliotecas e declarando as variáveis que iremos utilizar:

```
//Bibliotecas utilizadas
#include "TRandom3.h"
#include <iostream>
#include <cmath>
TRandom3 rd3; //para criar os numeros randomicos
int nev = 10000; // numero de eventos, no caso pontos
//iniciando a construcaoo da logica
int main ()
{
//Declarando as variaveis
double a;
double x;
double y;
double r;
double j=0.0; //casos favoraveis
```

Agora vamos criar os números randômicos que cada parte xy irá corresponder a um ponto dentro do quadrado:

```
//iniciando a contagem
for (int i=0;i<nev;i++) {
// numeros randomicos x e y
x = rd3.Uniform(-1.,1.);
y = rd3.Uniform(-1.,1.);
```

Vamos criar a função para o raio do circulo:

```
r = sqrt(pow(x,2) + pow(y,2));
```

Agora vem a principal condição, se o ponto tiver a coordenada menor ou igual a r ele está dentro do circulo, do contrário, fora. O j = j+1 irá contar esses casos:

```
if (r<=1.){
j = j +1;
}
```

Para mostrar o valor na tela, vamos usar o comando:

```
a = 4*j/nev; // valor de pi, mas como o circulo eh unitario, fica sendo valor da area
std::cout << "Area do circulo unitario= " << a << " u.a   (unidades de area)" << std
    ::endl;
```

Como o raio do circulo é 1, ele não aparece no cálculo. A área do disco (circulo) em questão é: 3,1424

Exercício 2

Evaluate the integral using both methods.

$$\int_0^3 (1-x^2)^2 dx$$

Os arquivos com os códigos desse problema são: exer2a.cc e exer2b.cc

Os métodos utilizados foram: rejeição e direto. No arquivo exer2a.cc está o médoto da rejeição e no exer2b.cc está o método direto.

Vamos começar:

a) Método da rejeição:

Sabemos que a integral em x vai de 0 até 3, ou seja, esses são os valores máximo e mínimo de x ($0 \leq x \leq 3$). O valor mínimo de y é 0 e o máximo pode ser obtido substituindo o valor máximo de x na função dentro da integral, mas sem integrar:

$$y_{max} = f(3) = (1-3^2)^2 = 64$$

Logo o intervalo de y é: $0 \leq y \leq 64$

De maneira similar ao exercício 1, podemos pensar numa figura delimitada por $(0 \leq x \leq 3)$ e $0 \leq y \leq 64$ e considerar que dentre os pontos gerados dentro desta figura (no caso, um retângulo), quais estão dentro da área da integral.

Vamos começar com as bibliotecas e variáveis:

```
//Bibliotecas utilizadas
#include "TRandom3.h"
#include <iostream>
#include <cmath>
TRandom3 rd3; //para criar os numeros randomicos
int nev = 100000; // numero de eventos, no caso pontos
//iniciando a construcao da logica
int main ()
{
//Declarando as variaveis
double a;
double x;
double y;
double funcao;
double j=0.0; //casos favoraveis
```

Agora vamos iniciar a contagem dos números randômicos xy:

```
//iniciando a contagem
for (int i=0;i<nev;i++) {
// numeros randomicos x e y
x = rd3.Uniform(0.,3.);
```

```

y = rd3.Uniform(0.,64.);
```

Vamos criar a função f(X) que está dentro da integral:

```
// funcao para ser calculada
funcao = pow((1-x*x),2);
```

Por fim, vem a condição principal do problema:

```
// Condicao principal do problema
if (y<=funcao){
j = j +1;
}
```

Para mostrar o valor da integral na tela vamos utilizar o comando:

```
a = 3*64*j/nev; // valor da integral
std::cout << "integral de (1-x^{2})^{2} dx = " << a << std::endl;
```

O valor da integral calculada é 33,8419.

b) Método direto
O método direto consiste em aplicar o somatório na função e calcular a integral diretamente por:

$$I = \frac{(b-a}{N} \sum_{i=0}^{N} f(x_i(b-a)+a)$$

Para fazer isso vamos substituir $[x_i(b-a)+a]$ no lugar de x daquela função da integral $[(1-x^2)^2]$, assim fica:

$$(1-(x_i(b-a)+a)^2)^2$$

Na integral do problema, os limites são 0 e 3. Ou seja, a = 0 e b = 3.
Devemos considerar necessariamente que os valores dos números randômicos estão no intervalo de 0 e 1.
Então vamos começar com as bibliotecas e as variáveis:

```
//Bibliotecas utilizadas
#include "TRandom3.h"
#include <iostream>
#include <cmath>
TRandom3 rd3; //para criar os numeros randomicos
int nev = 100000; // numero de eventos, no caso pontos
//iniciando a construcao da logica
int main ()
{
//Declarando as variaveis
int a=0;
int b=3;
double sum=0; // eh para calcular a soma
double I;
double x;
```

Agora vamos declarar o número randômico e iniciar a contagem

```
//iniciando a contagem
for (int i=0;i<nev;i++) {
// numero randomico x
x = rd3.Uniform(0.,1.); // lembrando que x esta no intervalo [0,1]
```

Vamos calcular o somatório:

```
// funcao para ser calculada, lembrando que a integral eh (1-x^{2})^{2}.
// Ou seja, substituir x(b-a)+a em (1-x^{2})^{2}
sum = sum+pow((1-pow((x*(b-a)+a),2)),2);
```

E por último o valor da integral:

```
I = sum*(b-a)/nev; // valor da integral
std::cout << "integral de (1-x^{2})^{2} dx = " << I << std::endl;
```

O valor da integral é 33,4842.

Exercício 3

Write a code to calculate the differential cross section of the Rutherford scattering.

A seção de choque diferencial de Rutherford é dada por:

$$\frac{d\sigma}{d\Omega} = (\frac{e^2}{8\pi.m.\epsilon_0.v_0^2})^2 \frac{1}{sin^4(\frac{\theta}{2})}$$

os valores das constantes foram tirados da internet.

Nesse problema a variável randômica é θ. Uma forma interessante é representar a seção de choque em um histograma. No caso de um espalhamento alfa, a partícula pode sair com ângulo θ variando de 0 até π ou entre $\frac{-\pi}{2}$ até $\frac{\pi}{2}$.

Se olharmos para a função seno que está no denominador podemos ver que quanto menor for o valor de θ maior será o valor da seção de choque. No entanto, não podemos escolher θ igual a 0, pois o resultado tende ao infinito. Mas podemos escolher arbitrariamente um valor de θ igual a 0,1 que é um pouco próximo de 0.

vamos começar com as bibliotecas e variáveis que iremos utilizar.

```
//Bibliotecas utilizadas
#include "TRandom3.h"
#include "TH1F.h"
#include "TROOT.h"
#include "TFile.h"
#include "TMath.h"
#include <iostream>
#include <cmath>
TRandom3 rd3; //para criar os numeros randomicos
int nev = 100000; // numero de eventos, no caso pontos
//iniciando a construcao da logica
int main ()
{
//Declarando as variaveis
double e=1,6*10e-19; // carga eletrica do eletron em Coulumb
double m=3,73*10e9; // massa da particula alfa em eV
double vi=1,5*10e7; //velocidade inicial media da particula alfa
double ep= 8,854*10e-12; // valor da permissividade eletrica no vacuo em C^2/N.m^2
double teta;
double x;
double k;
double y;
double j;
double sigmachoque;// secao de choque diferencial
```

Agora vamos iniciar a contagem dos eventos e escrever as condições levando em consideração os valores máximo e mínimo do número randômico x:

```
//iniciando a contagem
for (int i=0;i<nev;i++) {
teta= rd3.Uniform(-TMath::Pi()/2,TMath::Pi()/2); // teta randomico no intervalo de [-
    pi/2, pi/2]
k = pow((e*e/(8*m*ep*vi*vi*TMath::Pi())),2); //constante k, so para simplificar os
    calculos
y = 1/pow(sin(teta/2),4);
j = k*(1/pow(sin(0.1/2),4)); // valor maximo
x = rd3.Uniform(0,j); // lembrando que x esta no intervalo [0,j]
sigmachoque = k*y; // valor da secao de choque diferencial
```

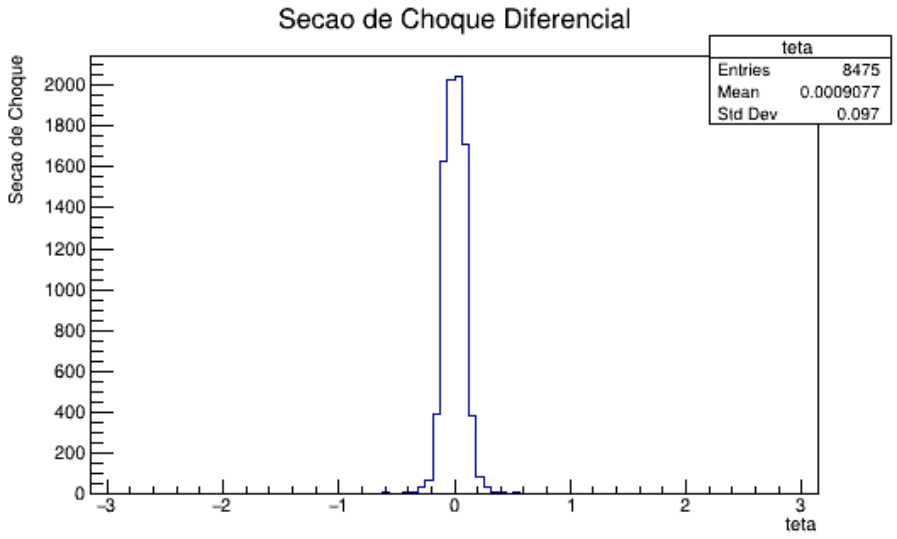
Agora vem a condição principal do problema:

```
if (x <= sigmachoque){
hist->Fill(teta);
```

Lembrnado que hist é o histograma que iremos criar agora para representar o gráfico da seção de choque:

```
//Seria interessante representar esse resultado na forma de um histograma pois se
    trata de uma secao de choque diferencial

TH1F *   hist = new TH1F("teta","Secao de Choque Diferencial",100,-TMath::Pi(),TMath
    ::Pi());
hist->GetXaxis()->SetTitle("teta");
hist->GetYaxis()->SetTitle("Secao de Choque");
TFile * fp = new TFile("hist_exer3.root","recreate");

hist->Write();
fp->Close();
```

Figura 1: Seção de Choque diferencial em função de θ



Lembrando que o gráfico está armazenado em um arquivo .root e pode ser acessado pelo root com o comando:

```
root -i hist_exer3.root
TBrowser t
```

Exercício 4

Para resolver este exercício e os outros que virão utilizamos o pythia8. Como estamos utilizando o miniconda3 fizemos a instalação do pythia8 pelo conda com os comandos:

```
conda config --add channels conda-forge
conda install pythia8
```

Agora vamos ao exercício:

Using the simplifies skeleton, modify the script to generate some events in LHC environment (at $\sqrt{s} = 14$ TeV and $\sqrt{s} = 7$ TeV).

a) The process to be considered are HardQCD:qqbar2bbbar , HardQCD:gg2bbbar, HardQCD:gg2qqbarg, HardQCD:qqbar2qqbargDiff and HardQCD: qqbar2qqbargSame. You should also turno on all quark flavours by HardQCD:nQuark New = 5

b) How can we select only events containing b quarks? How can we select only B-mesons?

c) Save particle information into a tree.

Para ajudar na compreensão dos códigos utilizados leia o artigo: https://pdg.lbl.gov/2014/reviews/rpp2014-rev-monte-carlo-numbering.pdf especialmente as páginas 3 até 5, pois iremos retirar alguns dados delas.

O código fonte ex1.C, que é o esqueleto que iremos utilizar e os principais arquivos podem ser adquiridos no repositório do github: https://github.com/Analise-Dados-FAE/Aula-MonteCarlo-Pythia8/tree/main/ex1

primeiramente incluímos algumas bibliotecas que iremos utilizar:

```
#include "Pythia8/Pythia.h"
#include "TH1.h"
#include "TTree.h"
#include "TFile.h"
#include <math.h>
#include "TSystem.h"
#include "TStyle.h"
using namespace Pythia8;
```

Na letra a) do exercício pede para incluir alguns processos, então podemos simplesmente utilizar o comando: pythia.readString("Processo a ser considera = on"); o comando 'on' liga o processo.

```
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");
```

Na letra b) para fazer a contagem apenas de quarks b e mésons B. Sabendo que pelo artigo o quark b é identificado por 5 e o méson B por 511 e 521 pois são três mésons B.

Vamos utilizar o code range presente no slide na página de status code. Para as partículas de subprocessos o code range é (21-29).

Os principais comando que iremos utilizar são:

```
pythia.init(); // Para iniciarlizar todos os processos subsequentes gerados.
pythia.next(); // para gerar o proximos eventos
pythia.stat // Para rodar a estatisica
```

Vamos começar gerando as colisões pp com energia de 7 TeV = 7000 GeV e depois com energia de 14 TeV = 14000 GeV separadamente para rodar uma de cada vez.

```
// Gerando as colisoes e os niveis de exergia: colisoes proton proton que irao
    ocorrer
pythia.readString("Beams:idA = 2212"); // Proton
pythia.readString("Beams:idB = 2212");  // Proton
pythia.readString("Beams:eCM = 7000."); // Energia do processo em GeV
```

Agora vamos para os processos subsequentes que são o quark b e o méson B, aqui serão gerados 100mil eventos:

```
// Iniciando os processos subsequentes
  pythia.init();

int bquark = 0; // variable to count b quarks in hadron process
int Bmeson = 0; // variable to count B mesons in final state
int ev= 100000; // numero de eventos que estamos gerando
```

Para selecionar apenas os eventos com quark b e apenas com méson B devemos criar dois loppings separados:

```
  // iniciando a contagem dos eventos e selecionando os quarks b
  for (int iEvent = 0; iEvent < ev; ++iEvent) {
    if (!pythia.next()) continue;
    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;
}
}
}

```

```
  // iniciando a contagem de eventos e selecionado os mesons B
  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;
}
}
}
```

Na letra c) para salvar os dados numa Tree vamos criar um histograma e preencher com os dados. Algo interessante é printar os resultados na tela para saber quantos quarks b e mésons B foram gerados:

```

// Iniciando os processos subsequentes
  pythia.init();
  TFile *file = TFile::Open("exer4.root","recreate"); // criando um arquivo root para
      armazenar os histogramas
  Event *event = &pythia.event;
  TTree *T = new TTree("T","exer4 Tree");
  T->Branch("event",&event);
```

E

```
  // iniciando a contagem de eventos e selecionado os mesons B
  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();
 }

  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;
}
```

Ao rodar 100mil eventos para 7 TeV obtivemos: Quantidade de eventos com quark b: 90180 Quantidade de eventos com Méson B: 88658

Ao rodar 100mil eventos para 14 TeV obtivemos: Quantidade de eventos com quark b: 87266 Quantidade de eventos com Méson B: 85977

Caso queira fazer alguns teste para outros tipos de colisões o arquivo onde se encontra esse código é exer4.C, os arquivos exer4.root e exer4b.root são correspondentes as energias de 7 TeV e 14 TeV, respectivamente.

Exercício 5

Use e+e- annihilation as an environment for the clean study of final-state QCD radiation. Specifically study

how the average number of final-state partons increases with ECM. Also, how well/badly the number of partons is described by Poisson distributions. Hint: some useful settings are: WeakSingleBoson:ffbar2gmZ = on PDF:lepton = off HadronLevel:all = off 23:onMode = off 23:onIfAny = 1 2 3 4 5

Desta vez iremos rodar os códigos para uma quantidade menor de eventos, nesse caso 10mil, pois para 100mil leva um certo tempo para terminar de rodar.

Vamos aproveitar algumas coisas do exercício anterior, como por exemplo algumas bibliotecas:

```
#include "Pythia8/Pythia.h"
#include "TH1.h"
#include "TTree.h"
#include "TFile.h"
#include <math.h>
#include "TSystem.h"
#include "TStyle.h"
using namespace Pythia8;
```

Vamos começar com os tipos de colisões, no caso elétron-pósitron, para isso podemos procurar o id dessas partículas no artigo mencionado no exercício 4. Nesse caso para o elétron o id é 11 e para o pósitron é -11, pois é a antipartícula do elétron.

```
int main() {
// particulas que irao colidir, no caso colisao eletron e antieletron(positron)
Pythia pythia;
pythia.readString("Beams:idA = 11"); // eletron
pythia.readString("Beams:idB = -11"); // antieletron (positron)
pythia.readString("Beams:eCM = 100."); // energia em GeV
```

Agora vamos acrescentar as dicas dada pelo exercício:

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

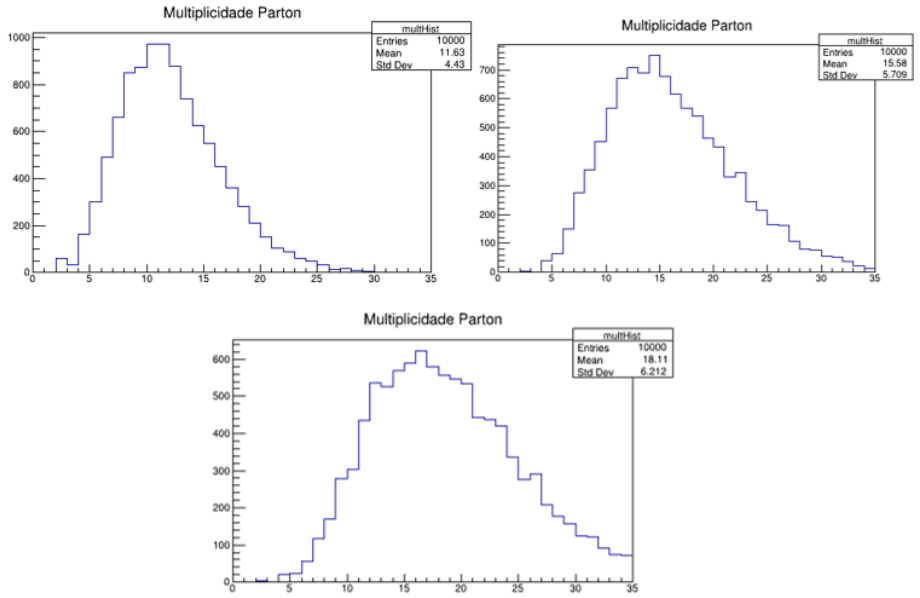
Vamos começar iniciando o pythia e já criar o histograma para estudar a relação dos parton com a energia do centro de massa, vamos rodar o programa três vezes para diferentes energias de centro de massa: 100, 200 e 300 GeVs.

```
// Iniciando o pythia
  pythia.init();
//Arquivo .root onde serao salvos os histogramas
  TFile *file = TFile::Open("exer5.root","recreate");
  // Histograma da multiplicidade parton
  TH1F *multHist = new TH1F("multHist","Multiplicidade Parton", 50, 0, 50);
  Event *event = &pythia.event;
  int ev = 10000; //numero de eventos
  int multp; // multiplicidade parton
  // Loop dos eventos
  for (int iEvent = 0; iEvent < ev; ++iEvent) {
    if (!pythia.next()) continue;
    multp = 0;
    // condicao para selecionar os eventos desejados
    for (int i = 0; i < pythia.event.size(); ++i){
      if (pythia.event[i].isFinal() && pythia.event[i].isParton()) ++multp; //
          seleciona apenas a parte final dos partons
}
    multHist->Fill(multp); // preenche o histograma com os dados da multiplicidade
}

  // Print stat info
  pythia.stat();
  multHist->Write();

```

```
25    delete file;
26    return 0;
27 }
```

A Figura a seguir mostra a relação entre os partons e a energia do centro de massa:

Figura 2: Relação do Parton com a energia do centro de massa (300, 200 e 100) GeV



Podemos ver que quanto maior é a energia menor é a multiplicidade parton para o par elétron-pósitron.