Introdução à Análise de dados em FAE **(09/05/2024)**

Lista de exercícios 5

Professores: Sandro Fonseca, Eliza Melo, Maurício Thiel e Diego Torres *Name:* Miguel Lopes

EXERCÍCIO 1

Tendo como base o experimento de Bufoon, calculei o valor de π para os casos N = 10, 50, 100, 1000. Neste exercício optei por não criar uma função onde eu pudesse só alterar o número de N, ao invés disso, como o código é pequeno, só repeti os comandos para os 4 casos.

Para a primeira questão elaborei esse código:

```
#include <iostream>
#include <cmath>
#include "TRandom3.h"
#include "TMath.h"

int MC_Ex1() {
    double d = 1.0;  // Dist ncia entre linhas paralelas
    double L = 0.5;  // comprimento da agulha
    int joga10 = 0;  int joga50 = 0; int joga100 = 0; int joga1000 = 0;// Contador de
        cruzamentoss
    int N = 10;
    double Estimativa_pi;


    for (int i = 0; i < N; ++i) {
        // ngulo  entre 0 e pi
        double theta = gRandom->Uniform(0, TMath::Pi());

        // posi  o no centro da agulha
        double g = gRandom->Uniform(0, d / 2);

        if (L / 2 * sin(theta) > g) {
            joga10++;
        }
    }

// Equa  o para pi
     Estimativa_pi = (2.0 * L *N) / (d * joga10);


    //resultados
    std::cout << "N = " << N << ", Cruzamentos = " << joga10
              << ", Estimativa de    = " << Estimativa_pi << std::endl;

//=============================================================================

    for (int i = 0; i < 50; ++i) {
        // ngulo  entre 0 e pi
        double theta = gRandom->Uniform(0, TMath::Pi());

        // posi  o no centro da agulha
        double g = gRandom->Uniform(0, d / 2);

        if (L / 2 * sin(theta) > g) {
            joga50++;
        }
```

```
    }

// Equação para
     Estimativa_pi = (2.0 * L *50) / (d * joga50);


    //resultados
    std::cout << "N = " << 50 << ", Cruzamentos = " << joga50
              << ", Estimativa de    = " << Estimativa_pi << std::endl;

//=============================================================================


    for (int i = 0; i < N*10; ++i) {
        // ngulo  entre 0 e pi
        double theta = gRandom->Uniform(0, TMath::Pi());

        // posição no centro da agulha
        double g = gRandom->Uniform(0, d / 2);

        if (L / 2 * sin(theta) > g) {
            joga100++;
        }
    }


// Equação para
     Estimativa_pi = (2.0 * L *N*10) / (d * joga100);

    // resultados
    std::cout << "N = " << N*10 << ", Cruzamentos = " << joga100
              << ", Estimativa de    = " << Estimativa_pi << std::endl;

//=============================================================================

    for (int i = 0; i < N*100; ++i) {
        // ngulo  entre 0 e pi
        double theta = gRandom->Uniform(0, TMath::Pi());

        // posição no centro da agulha
        double g = gRandom->Uniform(0, d / 2);

        if (L / 2 * sin(theta) > g) {
            joga1000++;
        }
    }

        // Equação para
     Estimativa_pi = (2.0 * L *N*100) / (d * joga1000);

    //resultados
    std::cout << "N = " << N*100 << ", Cruzamentos = " << joga1000
              << ", Estimativa de    = " << Estimativa_pi << std::endl;


    return 0;
}
```

A saída no terminal para esse código foi:

```
root [0]
Processing MC_Ex1.C...
N = 10, Cruzamentos = 3, Estimativa de    = 3.33333
N = 50, Cruzamentos = 17, Estimativa de    = 2.94118
N = 100, Cruzamentos = 31, Estimativa de    = 3.22581
```

```
N = 1000, Cruzamentos = 306, Estimativa de    = 3.26797
```

EXERCÍCIO 2

Usando o método de rejeição simples, fiz um código para resolver integrais. No meu exemplo, utilizei a integral x^2.

Meu código foi:

```
#include <iostream>
#include <cmath>
#include "TRandom3.h"

int MC_Ex2() {
    double a = 0;  // Limite de baixo
    double b = 1;  // Limite de cima
    int N = 10000;  // N mero de eventos

    double ymax = 1.0;  // limite superior


    TRandom3 random;

    int P_na_Curva = 0;  // Conta quantos pontos tem em baixo da curva

    for (int i = 0; i < N; i++) {

        double x = random.Uniform(a, b);
        double y = random.Uniform(0, ymax);

        //fun   o x^2
        if (y <= x * x) {
            P_na_Curva++;
        }
    }

    // A  rea  do ret ngulo
    double a_retangulo = (b - a)*ymax;

    // Calculo da integral
    double estimated_integral = (P_na_Curva / (double)N)*a_retangulo;

    std::cout << "A integral calculada de 0 a 1 para x^2   : "
              << estimated_integral << std::endl;

    return 0;
}
```

A saída no terminal para esse código foi:

```
root [0]
Processing MC_Ex2.C...
A integral calculada de 0 a 1 para x^2   : 0.3324
```

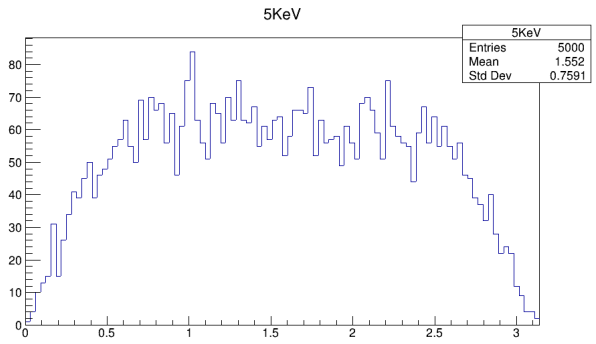
EXERCÍCIO 3

Neste exercício, não notei diferença ao mudar as energias, só vi diferença na distribuição ao mudar o número de eventos.

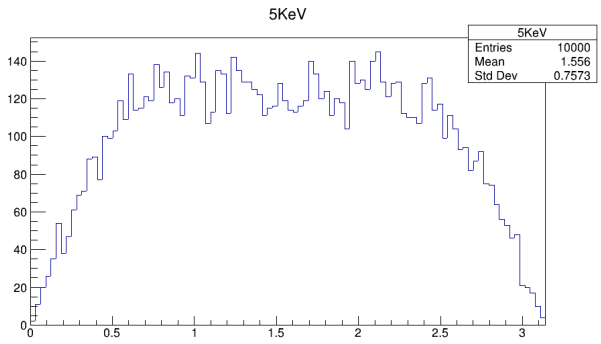
Meu código para esse exercício é:

```
#include "TMath.h"
#include <iostream>
#include "TRandom.h"
#include "TH1.h"
```

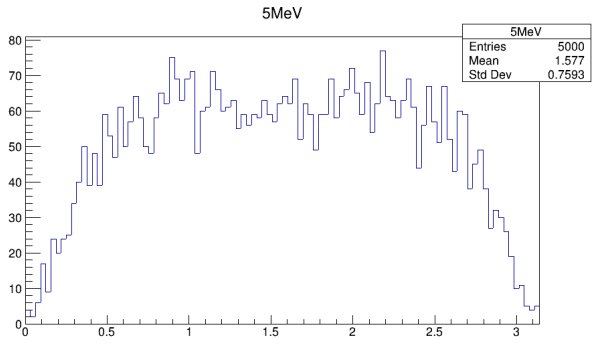
```
#include <math.h>

using namespace std;
double m = 0.511;
double k;
int nexp = 10000;
int cont = 0;
double x;
void MC_Ex3() {

TFile f("histo2.root","UPDATE");
TH1D * h1 = new TH1D("5KeV", "5KeV", 100,0, 3.141);

for(double j = 0; j< nexp; j++) {
//
        for(double i =0; i <1200; i++) {
                double zz = 1.076*(gRandom->Rndm());

                x =3.14*(gRandom->Rndm());

        if (zz <= ((pow((1/(1+0.01*(1-cos(x)))),2))*(((1/(1+0.01*(1-cos(x))))
            +(1+0.01*(1-cos(x))))-sin(x)*sin(x)))*sin(x)){i=1200;}
        }
        h1->Fill(x);
        }

h1->Write();
        cout << cont << endl;
        cont = cont + 1;
}
```



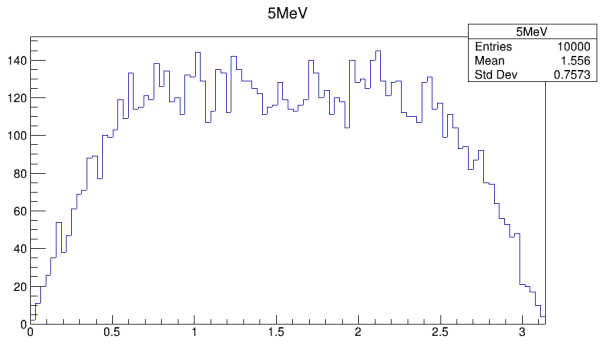
(a)



(b)



(c)



(d)

EXERCÍCIO 4

Toda essa questão foi feita no google colab, tive problemas para baixar o conda no meu computador, o que resultou na utilização do condar. Pelo jeito que o código está, acredito que ele funcione caso o leitor utilize as instruções dos slides. Mas adicionei comentários no código caso queiram reproduzir esse resultado no colab.

Meu código em Python é:

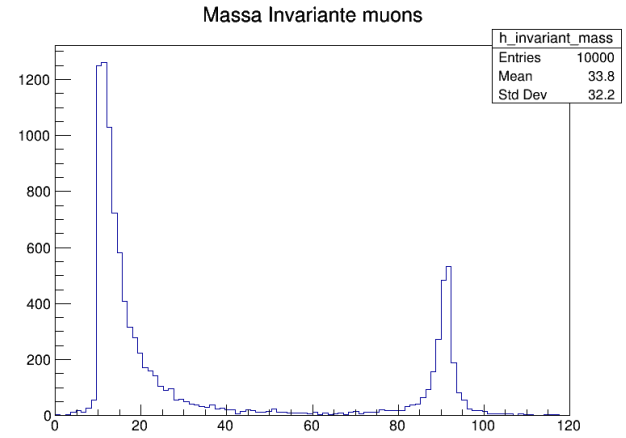
```
# Essa quest o foi feita no google colab pois tive dificuldade de colocar o conda no meu computador
#
# Vou colocar aqui, separado por tra os "-----------------" as c lulas que botei no colab para fazer esse c digo rodar.event
# No colab coloquei o nome do c digo de "run_pythia.py", mas para enviar, preferi seguir o padr o dos meus exerc cios


# ---------------------------------------------------------------
# !wget -q -c https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
# !bash Miniconda3-latest-Linux-x86_64.sh -b -f -p /usr/local
# import os
# os.environ["PATH"] += ":/usr/local/bin"
# !conda --version
#----------------------------------------------------------------

#----------------------------------------------------------------
#!conda create -c conda-forge --name pythia_env python=3.8 pythia8 root -y
#----------------------------------------------------------------



#----------------------------------------------------------------
#code = """ #DESCOMENTAR PARA RODAR NO COLAB
import pythia8
import ROOT
from ROOT import TFile, TTree, vector, TH1F, TLorentzVector, TCanvas


f = TFile("output.root", "RECREATE")
tree = TTree("Pythia Events", "Eventos do Pythia")

# Vetores para os dados
n_pt = vector('float')()
n_eta = vector('float')()
n_phi = vector('float')()

tree.Branch("Particle_pt", n_pt)
tree.Branch("Particle_eta", n_eta)
tree.Branch("Particle_phi", n_phi)

# Inicialize o Pythia para simular produ  o de b son Z com decaimento em m ons
pythia = pythia8.Pythia()
pythia.readString("Beams:eCM = 13000.")
pythia.readString("WeakBosonAndParton:qg2gmZq = on")
pythia.readString("23:onMode = off")  # Desativar todos decaimentos do Z
pythia.readString("23:onIfAny = 13")  # Ativar decaimento do Z em m ons
pythia.init()


muons = []

h_invariant_mass = TH1F("h_invariant_mass", "Massa Invariante muons", 100, 0, 120)
h_pt = TH1F("h_pt", "Particle_pt", 100, 0, 20)
h_eta = TH1F("h_eta", "Particle_eta", 100, -5, 5)
h_phi = TH1F("h_phi", "Particle_phi", 100, -3.14, 3.14)

```

```
# Loop para gerar eventos
for iEvent in range(10000):
    muons.clear()  # Limpar a lista de m ons
    if not pythia.next():
        continue

    n_pt.clear()  # Limpar os vetores para part culas finais
    n_eta.clear()
    n_phi.clear()

    # Preencher os vetores
    for particle in pythia.event:
        if particle.isFinal() and particle.isVisible():
            n_pt.push_back(particle.pT())
            n_eta.push_back(particle.eta())
            n_phi.push_back(particle.phi())
            h_pt.Fill(particle.pT())
            h_eta.Fill(particle.eta())
            h_phi.Fill(particle.phi())

            if abs(particle.id()) == 13:  # Identificar m ons
                muons.append(TLorentzVector(particle.px(), particle.py(), particle.pz
                    (), particle.e()))

            # Calcular a massa invariante para dois m ons
                if len(muons) == 2:
                    mass = (muons[0] + muons[1]).M()
                    h_invariant_mass.Fill(mass)

    tree.Fill()  # Preencher a tree


f.Write()




# Canvas para Particle_pt
c1 = TCanvas("c1", "Canvas 1", 800, 600)
h_pt.Draw()
c1.SaveAs("particle_pt.png")  # Salvar como PNG


c2 = TCanvas("c2", "Canvas 2", 800, 600)
h_eta.Draw()
c2.SaveAs("particle_eta.png")


c3 = TCanvas("c3", "Canvas 3", 800, 600)
h_phi.Draw()
c3.SaveAs("particle_phi.png")


c4 = TCanvas("c4", "Canvas 4", 800, 600)
h_invariant_mass.Draw()
c4.SaveAs("invariant_mass.png")
f.Close()

#""" #DESCOMENTAR PARA RODAR NO COLAB

#////////////////////////////////////////////////////////
#Esse peda o est va na mesma c lula que o c digo no colab, aqui est  comentado
```

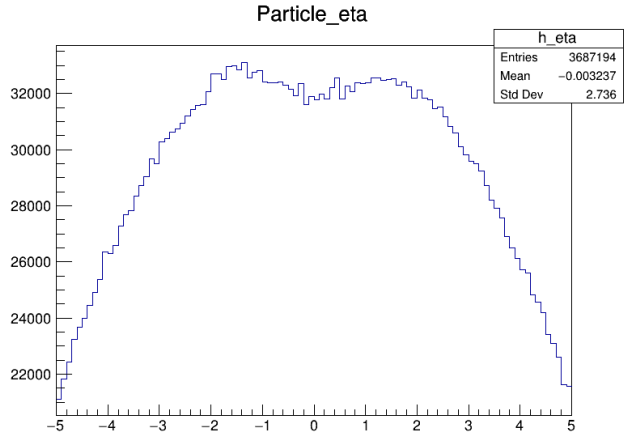
```
    para caso voc  queira rodar direto do seu coputador
#with open("run_pythia.py", "w") as file:
#    file.write(code)
#////////////////////////////////////////////////////////////
#------------------------------------------------------------


#------------------------------------------------------------
#!conda run -n pythia_env python run_pythia.py
#------------------------------------------------------------
```

Resolvi plotar a massa invariante de dois μ, que estão reconstruindo a massa do bóson Z^0 e o fenômeno de Drell–Yan. Note que as variáveis cinemática estão incluindo todas as partículas geradas pelo Pythia. Como o pythia gerou um bóson Z^0 por evento, temos apenas 10000 eventos de Z^0.

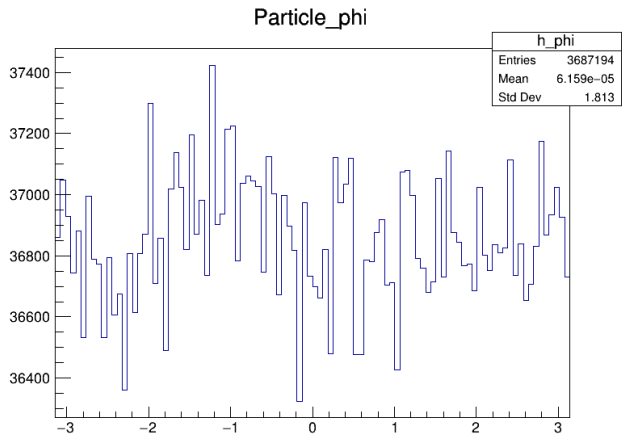
Meus plots ficaram assim:



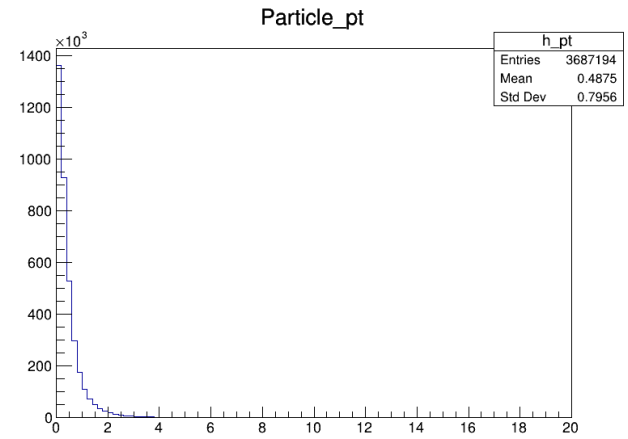
(a)



(b)



(c)



(d)