Introdução à Análise de dados em FAE **(03/05/2024)**

Exercícios - Geradores de Monte Carlo

Professores: Sandro Fonseca, Maurício Thiel, Eliza Melo *Name:* Isis Prazeres Mota

EXERCICIO 1

Este código calcula a estimativa de π para os valores de N definidos no array Ns, para cada teste realizando N lançamentos da agulha.

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

void estimatePi() {
    int Ns[] = {10, 50, 100, 1000};  // Valores de N para testar
    double l = 1.0;  // comprimento da agulha
    double d = 1.2;  // distancia entre as linhas

    for (int N : Ns) {
        int m = 0;  // contador de intersecoes
        TRandom3 rand(0);  // gerador de numeros aleatorios

        for (int i = 0; i < N; ++i) {
            double x = rand.Uniform(0, d / 2.0);  // distancia ate a linha mais proxima
            double theta = rand.Uniform(0, M_PI);  // angulo da agulha

            if (x <= (l / 2.0) * sin(theta)) {
                m++;
            }
        }

        if (m == 0) {
            std::cout << "Nenhum cruzamento detectado para N=" << N << ", impossivel estimar Pi." << std::endl;
            continue;
        }

        double pi_estimate = (2.0 * N / m) * (l / d);  // estimativa corrigida de Pi
        std::cout << "Estimativa de Pi para N=" << N << ": " << pi_estimate << std::endl;
    }
}
```

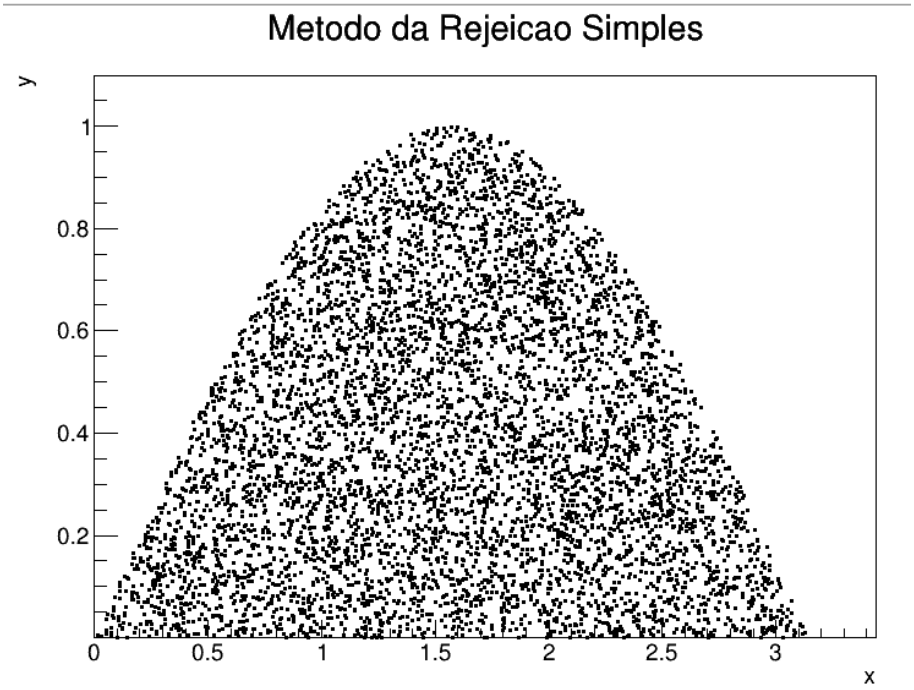
```
root [5] .x estimatePi.cpp
Estimativa de Pi para N=10: 4.16667
Estimativa de Pi para N=50: 2.77778
Estimativa de Pi para N=100: 3.26797
Estimativa de Pi para N=1000: 3.09789
```

EXERCICIO 2

TEXTO

```cpp
#include <TRandom3.h>
#include <TCanvas.h>
#include <TGraph.h>
#include <cmath>
#include <iostream>

double f(double x) {
    return sin(x);  // Funcao para calcular a integral
}

void integralEstimation() {
    int N = 10000;     // Numero de pontos para simulacao
    double a = 0;      // Limite inferior de x
    double b = M_PI;   // Limite superior de x
    double f_max = 1.0;  // Valor maximo de f(x) no intervalo [0, pi]

    TRandom3 rand(0); // Gerador de numeros aleatorios
    int m = 0;        // Contador de pontos abaixo da curva
    double x, y;

    // Preparar para plotar
    TGraph *gr = new TGraph();
    gr->SetTitle("Metodo da Rejeicao Simples; x; y");
    gr->SetMarkerStyle(7);

    // Simulacao do metodo da rejeicao
    for (int i = 0; i < N; ++i) {
        x = rand.Uniform(a, b);
        y = rand.Uniform(0, f_max);
        if (y <= f(x)) {
            m++;
            gr->SetPoint(gr->GetN(), x, y);  // Adiciona apenas pontos abaixo da
                curva
        }
    }

    // Calculo da integral
    double area_retangulo = (b - a) * f_max;
    double integral = area_retangulo * ((double)m / N);
    std::cout << "Estimativa da integral: " << integral << std::endl;

    // Plot
    TCanvas *c = new TCanvas("c", "Metodo da Rejeicao", 800, 600);
    gr->Draw("AP");
    c->SaveAs("IntegralEstimation.png");  // Salva o plot como PNG
}

int main() {
    integralEstimation();
    return 0;
}
```

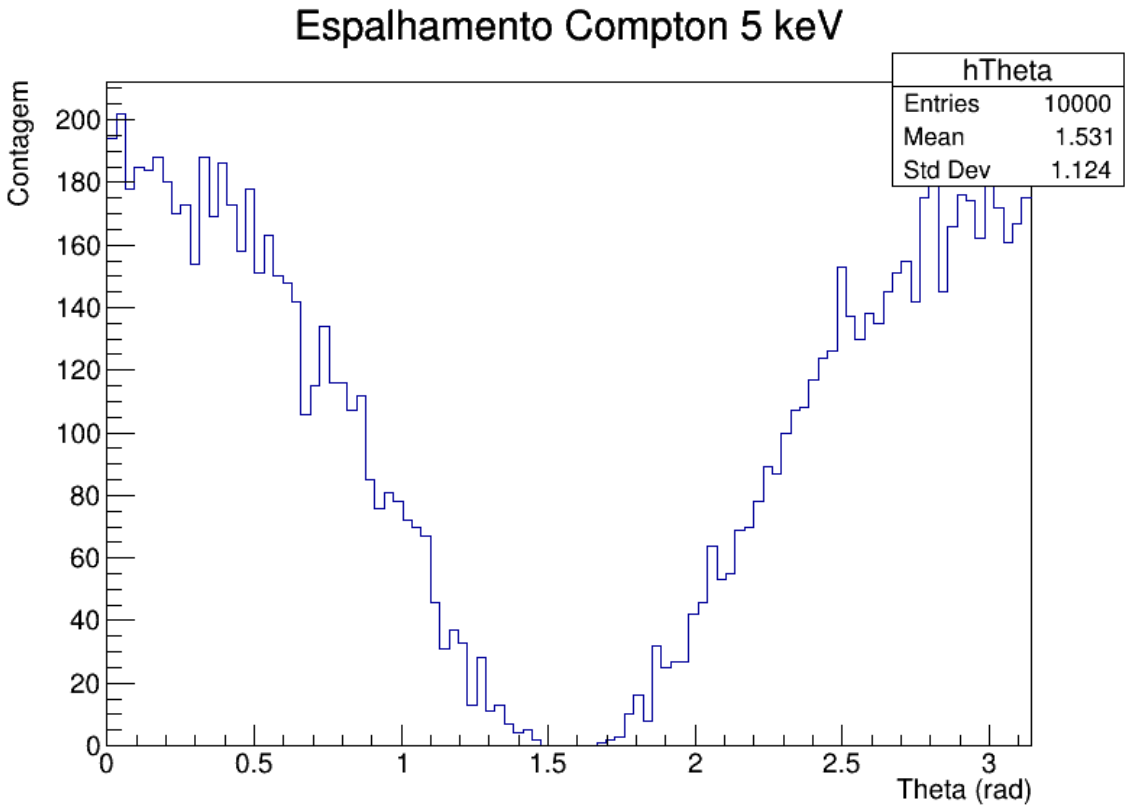
```
root [1] .x integralEstimation.cpp
Estimativa da integral: 2.00308
Info in <TCanvas::Print>: file IntegralEstimation.png has been created
```

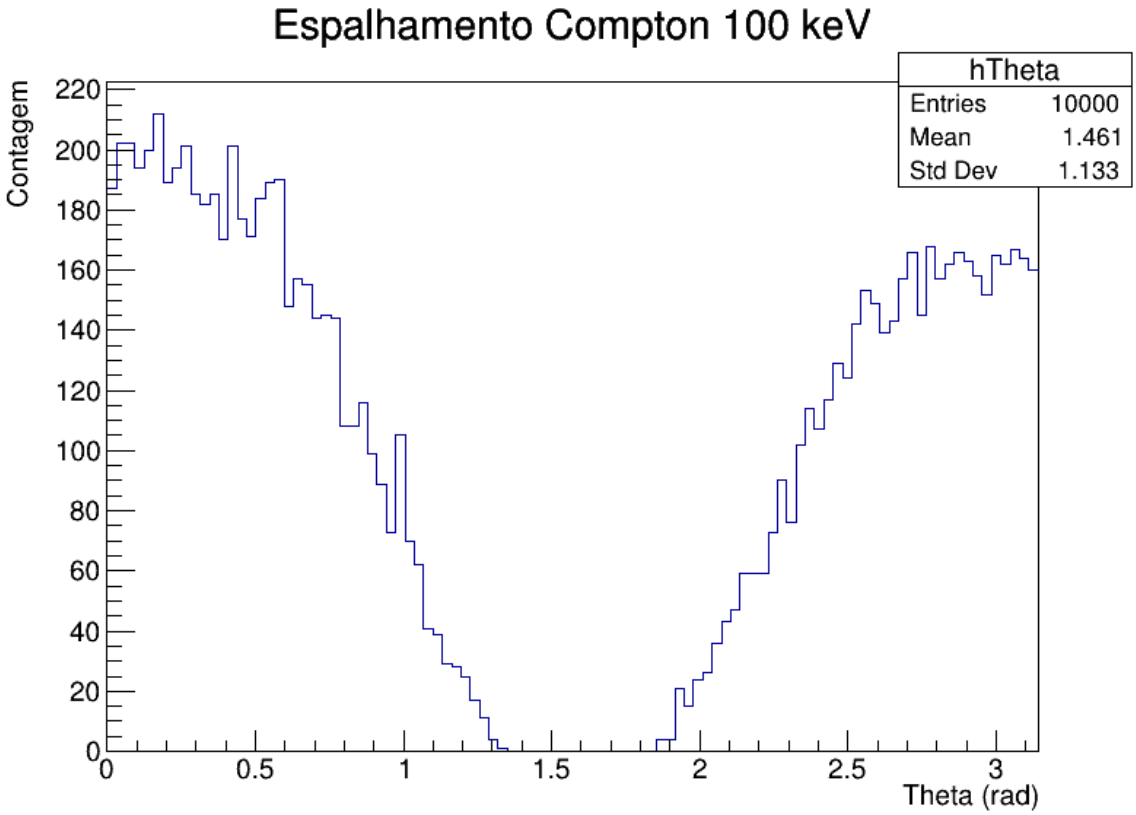


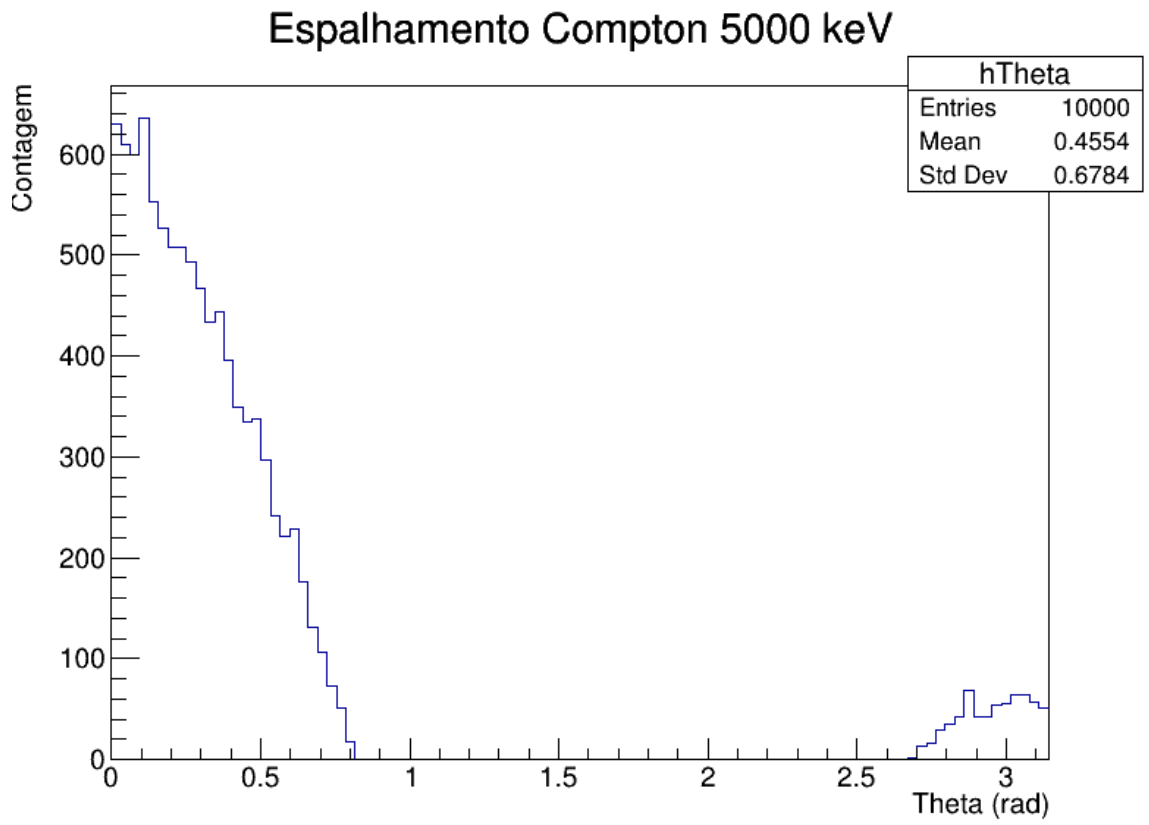
EXERCICIO 3

```
#include <TCanvas.h>
#include <TFile.h>
#include <TH1D.h>
#include <TRandom3.h>
#include <TMath.h>
#include <iostream>
#include <memory>
#include <TApplication.h>

// Definicao da funcao simulateTheta que recebe energia em keV e numero de experimentos como parametros
void simulateTheta(Double_t energy_keV, Int_t nexp) {
    // Convertendo a energia de keV para MeV para uso em calculos fisicos
    Double_t alpha = energy_keV * 0.001;

    // Inicializando o gerador de numeros aleatorios
    TRandom3 rand(0);

    // Criando um arquivo ROOT para salvar o histograma
    auto file = std::make_unique<TFile>(Form("histo_%dkeV.root", int(energy_keV)), "RECREATE");

    // Criando um histograma para registrar os valores de angulo Theta
    auto hTheta = std::make_unique<TH1D>("hTheta", Form("Espalhamento Compton %d keV; Theta (rad); Contagem", int(energy_keV)), 100, 0, TMath::Pi());

    // Variavel para contar quantos eventos foram aceitos
    Double_t totalAccepted = 0;
    std::cout << "Simulaca o Iniciada para " << energy_keV << " keV" << std::endl;

    // Loop sobre o numero de experimentos
    for (Int_t j = 0; j < nexp; ++j) {
        bool accepted = false;
        Int_t attempts = 0;
```

```
32          while (!accepted && attempts < 1000) { // Loop de tentativas ate aceitar o
                evento ou alcancar o limite de tentativas
33              Double_t zz = 1.076 * rand.Rndm(); // Gerar um numero aleatorio ponderado
34              Double_t theta = TMath::Pi() * rand.Rndm(); // Gerar um valor aleatorio
                    de theta entre 0 e pi
35              // Calculando a funcao de densidade de probabilidade modificada
36              Double_t fx = (pow((1 / (1 + alpha * (1 - TMath::Cos(theta)))), 2) * (1 +
                     alpha * (1 - TMath::Cos(theta)))) - TMath::Sin(theta) * TMath::Sin(
                    theta) * TMath::Sin(theta);
37
38              // Verificacao se o evento e aceito
39              if (zz <= fx) {
40                  hTheta->Fill(theta);
41                  accepted = true;
42                  totalAccepted++;
43              }
44              attempts++;
45          }
46      }
47
48      // Salvando resultados e informando o usuario
49      std::cout << "Total Aceito: " << totalAccepted << " de " << nexp << std::endl;
50      hTheta->Write();
51
52      // Preparando e mostrando o histograma em um canvas
53      auto c1 = std::make_unique<TCanvas>("c1", "Espalhamento Compton", 800, 600);
54      hTheta->Draw();
55      c1->Modified();
56      c1->Update();
57      std::cout << "Desenho Completo. Feche o canvas para terminar." << std::endl;
58
59      // Mantem a aplicacao rodando ate que o usuario feche o canvas
60      while (!gSystem->ProcessEvents()) {
61          gSystem->Sleep(50);
62      }
63  }
64
65  // Ponto de entrada principal do programa que verifica os argumentos da linha de
        comando
66  int main(int argc, char **argv) {
67      if (argc != 3) {
68          std::cout << "Uso: " << argv[0] << " <energia_keV> <nexp>" << std::endl;
69          return 1;
70      }
71      Double_t energy_keV = atof(argv[1]);
72      Int_t nexp = atoi(argv[2]);
73
74      TApplication theApp("App", &argc, argv);
75      simulateTheta(energy_keV, nexp);
76      return 0;
77  }
```


Espalhamento Compton 5 keV
hTheta
Entries 10000
Mean 1.531
Std Dev 1.124
Contagem
Theta (rad)

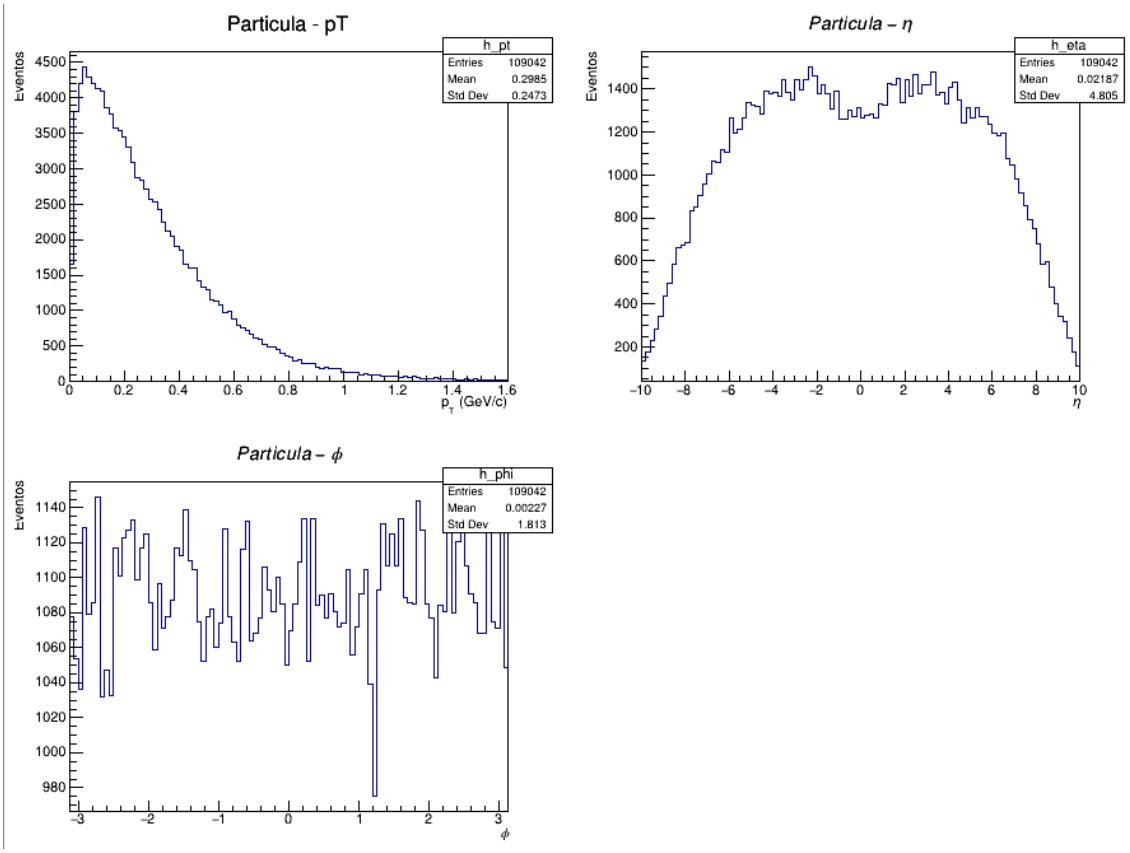

Espalhamento Compton 100 keV
hTheta
Entries 10000
Mean 1.461
Std Dev 1.133
Contagem
Theta (rad)



EXERCICIO 4

```
import pythia8
from ROOT import TFile, TTree, TH1F, vector, TCanvas

# Inicializa o Pythia
pythia = pythia8.Pythia()

# Configura os parametros do Pythia
pythia.readString("Beams:eCM = 13000.")  # Energia do centro de massa em GeV
pythia.readString("HardQCD:all = on")    # Processos QCD rigidos

# Inicializa o gerador Pythia
pythia.init()

# Inicializa o arquivo ROOT e os histogramas
f = TFile("output.root", "RECREATE")
h_pt = TH1F("h_pt", "Particula - pT; p_{T} (GeV/c); Eventos", 100, 4, 4)
h_eta = TH1F("h_eta", "Particula - \eta; \eta; Eventos", 100, -10, 10)
h_phi = TH1F("h_phi", "Particula - \phi; \phi; Eventos", 100, -3.14, 3.14)

# Inicializa a arvore
tree = TTree("tree", "Eventos Pythia")
n_particles = vector('float')()
n_eta = vector('float')()
n_phi = vector('float')()

# Definicao dos ramos
tree.Branch("Particle_pt", n_particles)
tree.Branch("Particle_eta", n_eta)
tree.Branch("Particle_phi", n_phi)

```

```
# Loop de eventos
for iEvent in range(1000):
    if not pythia.next():
        continue
    n_particles.clear()
    n_eta.clear()
    n_phi.clear()
    for particle in pythia.event:
        if particle.isFinal() and particle.isVisible():
            pt = particle.pT()
            eta = particle.eta()
            phi = particle.phi()
            if pt > -5:  # Filtro de pT
                n_particles.push_back(pt)
                n_eta.push_back(eta)
                n_phi.push_back(phi)
                h_pt.Fill(pt)
                h_eta.Fill(eta)
                h_phi.Fill(phi)
    tree.Fill()

# Criar canvas para desenhar os histogramas apos preenchimento
c = TCanvas("c", "Canvas para Histogramas", 900, 700)
c.Divide(2, 2)

c.cd(1)
h_pt.Draw()

c.cd(2)
h_eta.Draw()

c.cd(3)
h_phi.Draw()

c.Update()

# Escreve e fecha o arquivo ROOT
f.Write()
f.Close()

# Espera uma acao do usuario para fechar os programas
input("Pressione Enter para sair...")
```


Particula - pT
h_pt
Entries 109042
Mean 0.2985
Std Dev 0.2473
Eventos
p_T (GeV/c)
Particula – η
h_eta
Entries 109042
Mean 0.02187
Std Dev 4.805
Eventos
η
Particula – ϕ
h_phi
Entries 109042
Mean 0.00227
Std Dev 1.813
Eventos
ϕ