**Introdução à Análise de dados em FAE** **(Data: 03/11/2020)**

# Relatório 2 - Root

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

Este relatório tem como ponto de partida um arquivo previamente pronto chamado de Analyze.c, onde serão feitas algumas modificações na forma de respostas aos exercícios. Cada exercício propõe uma modificação que será a seguir.

Para facilitar a compreensão dos comados utilizados, será colocado aqui o arquivo fonte (Analyze.c) e cada modificação será mostrada para facilitar o acompanhamento.

```
1  #define Analyze_cxx
2  #include "Analyze.h"
3  #include <TH2.h>
4  #include <TStyle.h>
5
6  //**********Definition section**********
7  TH1* chi2Hist = NULL;
8
9  void Analyze::Begin(TTree * /*tree*/)
10 {
11    TString option = GetOption();
12
13    //**********Initialization section**********
14    chi2Hist = new TH1D("chi2", "Histogram of Chi2", 100, 0, 20);
15    chi2Hist->GetXaxis()->SetTitle("chi2");
16    chi2Hist->GetYaxis()->SetTitle("number of events");
17 }
18
19 void Analyze::SlaveBegin(TTree * /*tree*/){}
20
21 Bool_t Analyze::Process(Long64_t entry)
22 {
23    //  D o n t  delete this line! Without it the program will crash
24    fReader.SetLocalEntry(entry);
25
26    //**********Loop section**********
27    GetEntry(entry);
28    chi2Hist->Fill(*chi2);
29
30    return kTRUE;
31 }
32
33 void Analyze::SlaveTerminate(){}
34
35 void Analyze::Terminate()
36 {
37     //**********Wrap-up section**********
38     chi2Hist->Draw();
39 }
```

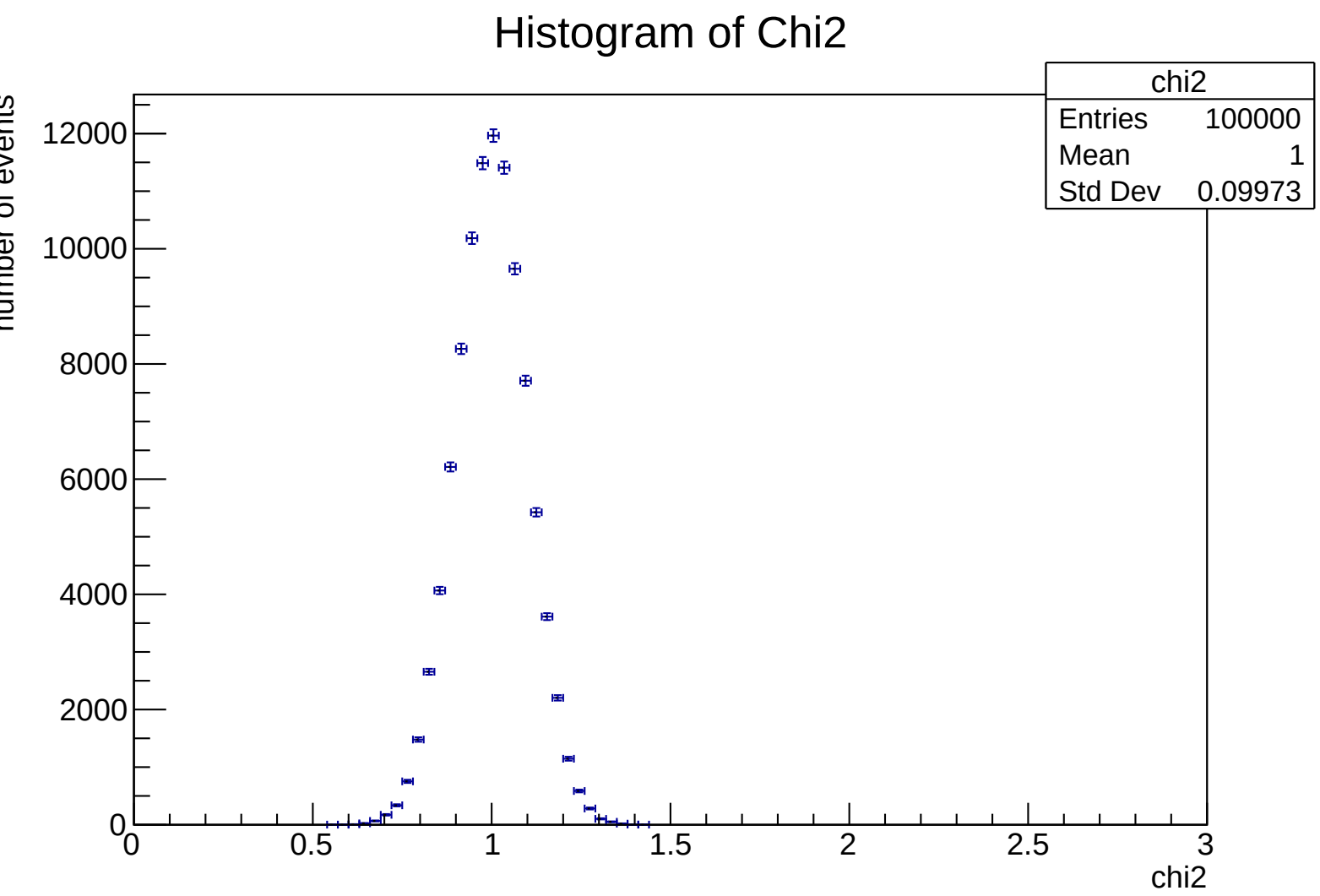
**Exercício 1:**
Revise the Analyze::Terminate method in Analyze.C to draw the histograms with error bars

Para adicionar barras de erros no histograma do chi2, podemos simplesmente incluir o comando BE1 (Barra de Erros) no método terminate, dentro dos parenteses do comando "Draw"na linha 38:

```
1  Void Analyze::Terminate ()
2  {
3    // **************Wrap-up section***********
```

```
4     chi2Hist->Draw("BE1");
5  }
```

Para melhorar a grade de visualização do histograma chi2, o tamanho do eixo x foi mudado de (0, 20) para (0, 3). Segue abaixo a imagem do histrograma chi2:


Histogram of Chi2
chi2
Entries 100000
Mean 1
Std Dev 0.09973
number of events
12000
10000
8000
6000
4000
2000
0
0
0.5
1
1.5
2
2.5
3
chi2

**Exercício 2:**
Revise Analyze.C to create, fill and display an additional histogram of the variable ebeam (with error bars and axis label)

Vamos adicionar o histograma da variável ebeam. Para fazer isso, vamos no método Analyze Begin e criar um histograma chamado "ebeamHist"e colocar os nomes nos eixos do histograma para melhor representá-lo:

```
void Analyze::Begin(TTree * /*tree*/)
{
   TString option = GetOption();

   //**********Initialization section**********
   chi2Hist = new TH1D("chi2", "Histogram of Chi2", 100, 0, 3);
   chi2Hist->GetXaxis()->SetTitle("chi2");
   chi2Hist->GetYaxis()->SetTitle("number of events");

   // Cria um histograma chamado de ebeamHist e coloca os nomes nos eixos
   ebeamHist = new TH1D("ebeam", "Histogram of ebeam", 100, 148., 152.);
   ebeamHist->GetXaxis()->SetTitle("ebeam (GeV)");
   ebeamHist->GetYaxis()->SetTitle("number of events");

}
```

A janela do eixo x vai de 148 até 152 para valorizar a visualização dos dados.
Para adicionar a barra de erros nesse histograma, vamos utilizar o método Analyze Terminate e usar o comando "Draw ("EB1")"para desenhar o histograma com barras de erros?

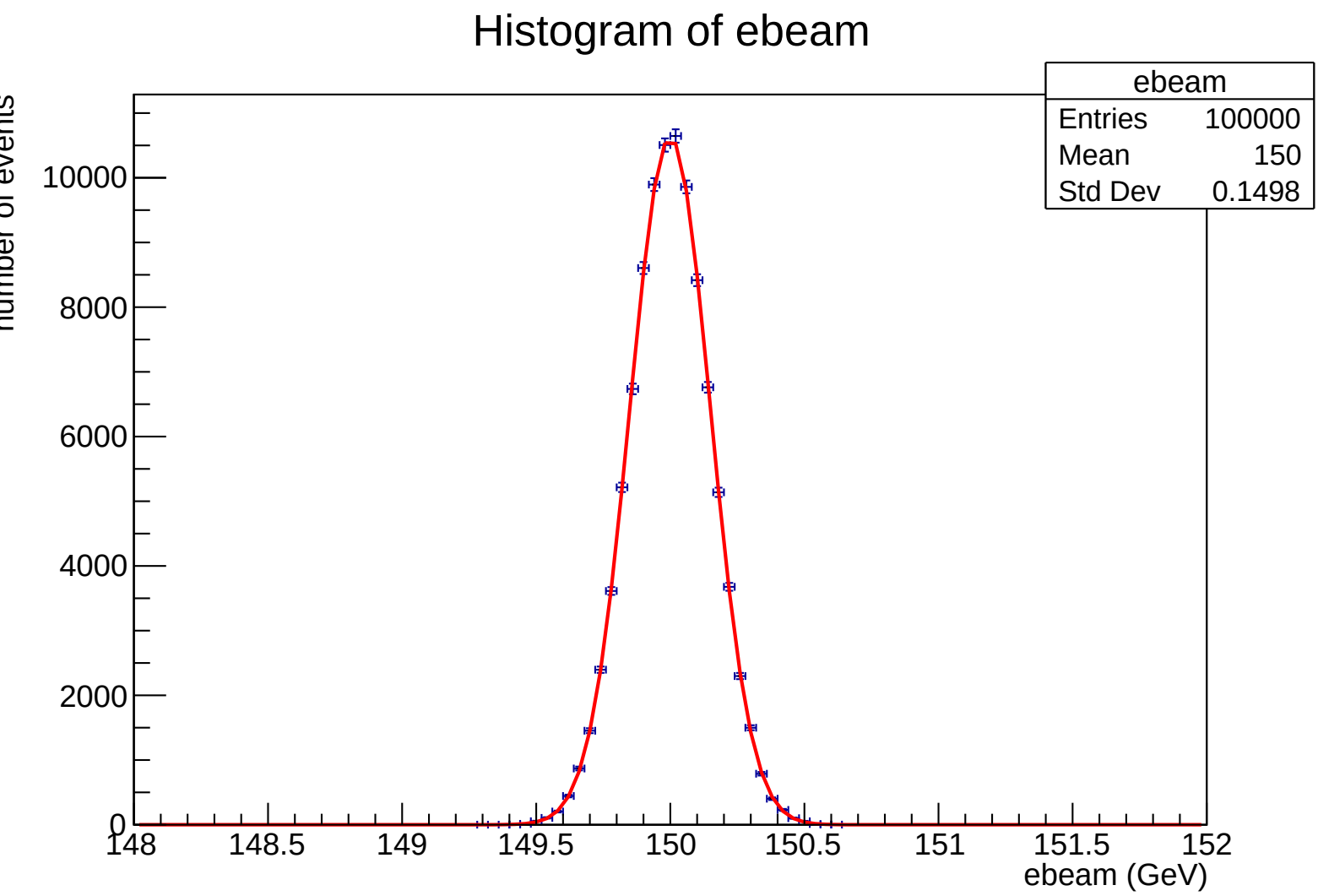
```
void Analyze::Terminate()
{
    //**********Wrap-up section**********
    // Desenha chi2Hist com barras de erros
    chi2Hist->Draw("BE1");

    //Desenha o histograma ebeam com barras de erros
    ebeamHist->Draw("BE1");
```

**Exercício 3:**
Fit the ebeam histogram to a gaussian distribution.

Vamos fazer um fit do histograma ebeamHist usando uma gaussiana, vamos lembrar de desenhar com barras de erros.
Para fazer isso vamos utilizar o método Analyze Terminate e usar o comando "Fit":

```
void Analyze::Terminate()
{
    //**********Wrap-up section**********
    // Desenha chi2Hist com barras de erros
    chi2Hist->Draw("BE1");

    //Desenha o histograma ebeam com barras de erros
    ebeamHist->Draw("BE1");
    // Faz o Fit do histrograma ebeamHist usando uma gaussiana e com barras de erros
    ebeamHist->Fit("gaus","V","BE1",148.,152.);
```

Segue a imagem do histograma ebeam com fit da gaussiana:


Histogram of ebeam
ebeam
Entries 100000
Mean 150
Std Dev 0.1498
number of events
10000
8000
6000
4000
2000
0
148
148.5
149
149.5
150
150.5
151
151.5
152
ebeam (GeV)

**Exercício 4:**
Add another plot: a scatterplot of chi2 versus ebeam. Don't forget to label the axes.

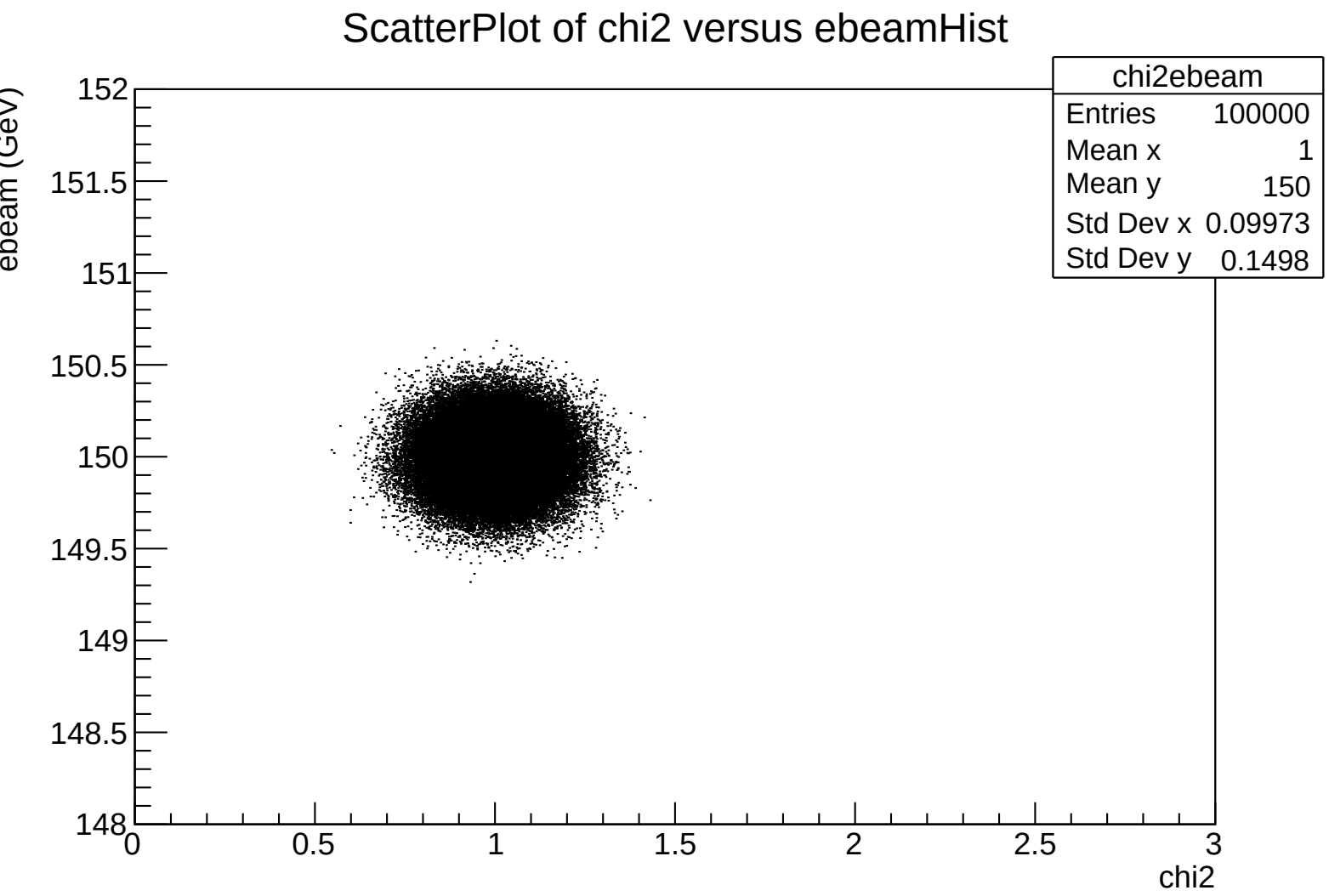
Para fazer um plot do chi2 versus ebeam, vamos criar uma nova TH2D no método Analyze Begin:

```
void Analyze::Begin(TTree * /*tree*/)
{
   TString option = GetOption();

   //**********Initialization section**********
   chi2Hist = new TH1D("chi2", "Histogram of Chi2", 100, 0, 3);
   chi2Hist->GetXaxis()->SetTitle("chi2");
   chi2Hist->GetYaxis()->SetTitle("number of events");

   // Cria um histograma chamado de ebeamHist e coloca os nomes nos eixos
   ebeamHist = new TH1D("ebeam", "Histogram of ebeam", 100, 148., 152.);
   ebeamHist->GetXaxis()->SetTitle("ebeam (GeV)");
   ebeamHist->GetYaxis()->SetTitle("number of events");

   // Cria um scatterplot para chi2 versus ebeamHist e coloca os nomes nos eixos
   chi2ebeamHist = new TH2D("chi2ebeam", "ScatterPlot of chi2 versus ebeamHist", 100,
        0, 3, 100, 148., 152.);
   chi2ebeamHist->GetXaxis()->SetTitle("chi2");
   chi2ebeamHist->GetYaxis()->SetTitle("ebeam (GeV)");
```

Não podemos esquecer de preencher os scatterplot, ou seja, os histogramas ebeamHist e o chi2ebeamHist. Para fazer isso vamos usar o método Analyze Process:

```
void Analyze::SlaveBegin(TTree * /*tree*/){}

Bool_t Analyze::Process(Long64_t entry)
{
   //  D o n t  delete this line! Without it the program will crash
   fReader.SetLocalEntry(entry);

   //**********Loop section**********
   GetEntry(entry);
   chi2Hist->Fill(*chi2);
   // Preenche o ebeamHist com a vari vel ebeam
   ebeamHist->Fill(*ebeam);
   // Preenche o chi2ebeamHist com as vari veis ebeam e chi2
   chi2ebeamHist->Fill(*chi2,*ebeam);
```

Segue a imagem do histograma chi2 versus ebeam:


ScatterPlot of chi2 versus ebeamHist
chi2ebeam
Entries 100000
Mean x 1
Mean y 150
Std Dev x 0.09973
Std Dev y 0.1498
ebeam (GeV)
152
151.5
151
150.5
150
149.5
149
148.5
148
0
0.5
1
1.5
2
2.5
3
chi2

**Exercício 5:**

Calculate pt in an analysis macro and make a histogram of the variable. (Remember that all n-tuple variables are pointers: pt = TMath::sqrt((*px)*(*px) + (*py)*(py));)

Nesse exercícios vamos calcular as variáveis pt e produzi-las em um histograma.
Para fazer isso podemos usar o comando Analyze Begin para criar um histograma chamado ptHist:

```
void Analyze::Begin(TTree * /*tree*/)
{
   TString option = GetOption();

   //**********Initialization section**********
   chi2Hist = new TH1D("chi2", "Histogram of Chi2", 100, 0, 3);
   chi2Hist->GetXaxis()->SetTitle("chi2");
   chi2Hist->GetYaxis()->SetTitle("number of events");

   // Cria um histograma chamado de ebeamHist e coloca os nomes nos eixos
   ebeamHist = new TH1D("ebeam", "Histogram of ebeam", 100, 148., 152.);
   ebeamHist->GetXaxis()->SetTitle("ebeam (GeV)");
   ebeamHist->GetYaxis()->SetTitle("number of events");

   // Cria um scatterplot para chi2 versus ebeamHist e coloca os nomes nos eixos
   chi2ebeamHist = new TH2D("chi2ebeam", "ScatterPlot of chi2 versus ebeamHist", 100,
       0, 3, 100, 148., 152.);
   chi2ebeamHist->GetXaxis()->SetTitle("chi2");
   chi2ebeamHist->GetYaxis()->SetTitle("ebeam (GeV)");

   // Cria um histograma chamado ptHist com seus eixos nomeados
   ptHist = new TH1D("pt", "Histogram of pt", 100, 0, 28);
   ptHist->GetXaxis()->SetTitle("pt (GeV)");
   ptHist->GetYaxis()->SetTitle("number of events");
```

A janela do eixo x do histograma pt vai de (0 até 28) para bem representar os dados.
Para preencher este histogrma, vamos usar o método Analyze Process para cálcular o pt, para fazer isso devemos usar uma biblioteca chamada de Tmath:

```
void Analyze::SlaveBegin(TTree * /*tree*/){}

Bool_t Analyze::Process(Long64_t entry)
{
   //  D o n t  delete this line! Without it the program will crash
   fReader.SetLocalEntry(entry);

   //**********Loop section**********
   GetEntry(entry);
   chi2Hist->Fill(*chi2);
   // Preenche o ebeamHist com a vari vel ebeam
   ebeamHist->Fill(*ebeam);
   // Preenche o chi2ebeamHist com as vari veis ebeam e chi2
   chi2ebeamHist->Fill(*chi2,*ebeam);

   // Calcula o pt:
   pt = TMath::Sqrt((*px)*(*px)+(*py)*(*py));
   ptHist->Fill(pt);
```

**Exercício 6:**

Include a histogram of theta, using the math function TMath::ATan2(pt,pz).

Agora vamos adicionar um histograma chamado de thetaHist e preencher, utilizando os mesmos métodos que usamos para criar e preencher o histograma ptHist

Criar o histograma thetaHist:

```
void Analyze::Begin(TTree * /*tree*/)
{
   TString option = GetOption();

   //**********Initialization section**********
   chi2Hist = new TH1D("chi2", "Histogram of Chi2", 100, 0, 3);
   chi2Hist->GetXaxis()->SetTitle("chi2");
   chi2Hist->GetYaxis()->SetTitle("number of events");

   // Cria um histograma chamado de ebeamHist e coloca os nomes nos eixos
   ebeamHist = new TH1D("ebeam", "Histogram of ebeam", 100, 148., 152.);
   ebeamHist->GetXaxis()->SetTitle("ebeam (GeV)");
   ebeamHist->GetYaxis()->SetTitle("number of events");

   // Cria um scatterplot para chi2 versus ebeamHist e coloca os nomes nos eixos
   chi2ebeamHist = new TH2D("chi2ebeam", "ScatterPlot of chi2 versus ebeamHist", 100,
        0, 3, 100, 148., 152.);
   chi2ebeamHist->GetXaxis()->SetTitle("chi2");
   chi2ebeamHist->GetYaxis()->SetTitle("ebeam (GeV)");

   // Cria um histograma chamado ptHist com seus eixos nomeados
   ptHist = new TH1D("pt", "Histogram of pt", 100, 0, 28);
   ptHist->GetXaxis()->SetTitle("pt (GeV)");
   ptHist->GetYaxis()->SetTitle("number of events");

   // Cria um histograma chamado de thetaHist com seus respectivos eixos
   thetaHist = new TH1D("theta", "Histogram of theta", 100, -0.6, 0.6);
   thetaHist->GetXaxis()->SetTitle("theta");
   thetaHist->GetYaxis()->SetTitle("number of events");
```

Os eixos do histograma theta vai de (-0,6 até 0,6) para que o histograma fique bem enquadrado.
Preencher o histograma thetaHist

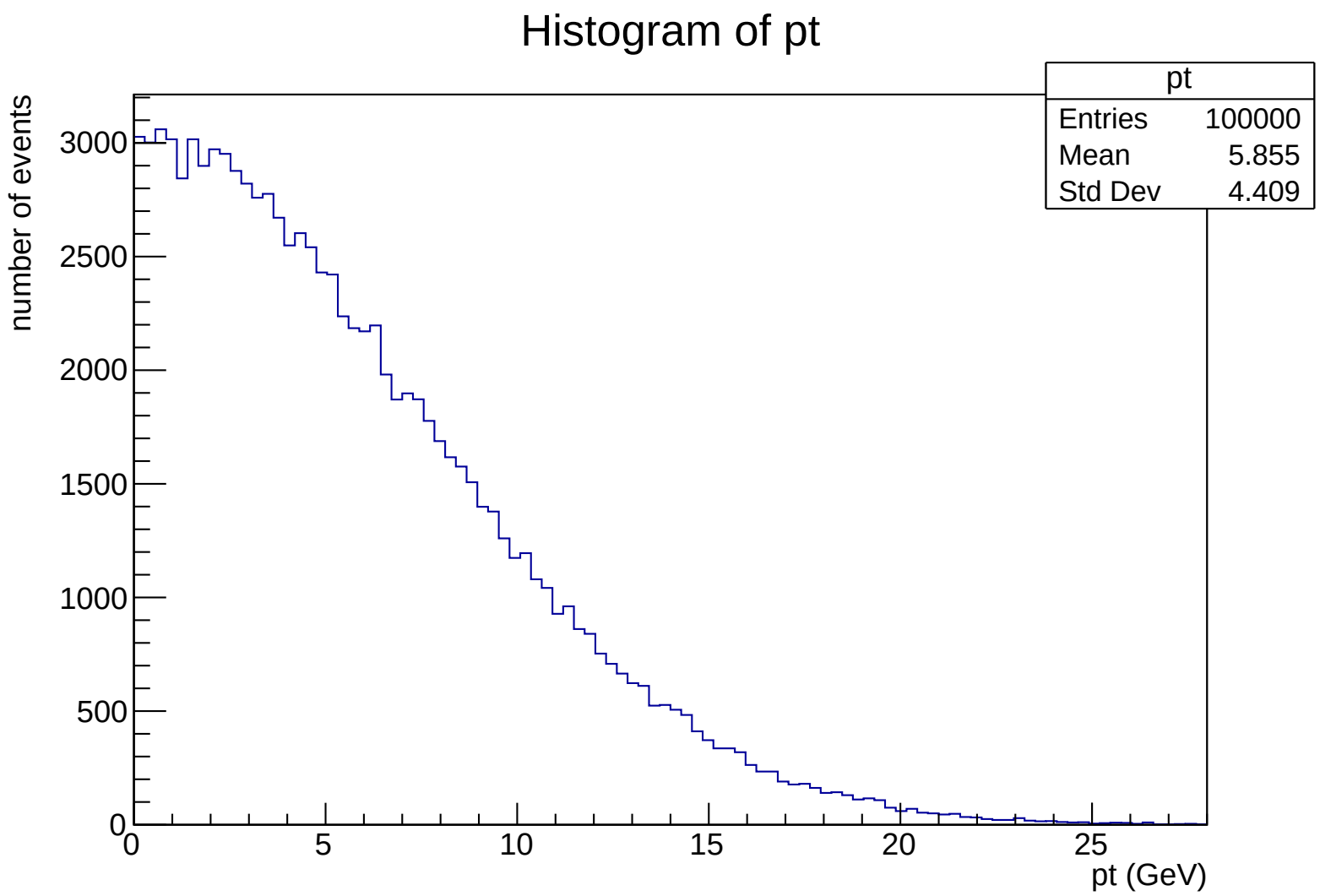
```
void Analyze::SlaveBegin(TTree * /*tree*/){}

Bool_t Analyze::Process(Long64_t entry)
{
   //  D o n t  delete this line! Without it the program will crash
   fReader.SetLocalEntry(entry);

   //**********Loop section**********
   GetEntry(entry);
   chi2Hist->Fill(*chi2);
   // Preenche o ebeamHist com a vari vel ebeam
   ebeamHist->Fill(*ebeam);
   // Preenche o chi2ebeamHist com as vari veis ebeam e chi2
   chi2ebeamHist->Fill(*chi2,*ebeam);

   // Calcula o pt:
   pt = TMath::Sqrt((*px)*(*px)+(*py)*(*py));
   ptHist->Fill(pt);

   // Calcula o valor de theta:
   theta = TMath::ATan2((pt),(*pz));
   thetaHist->Fill(theta);
```

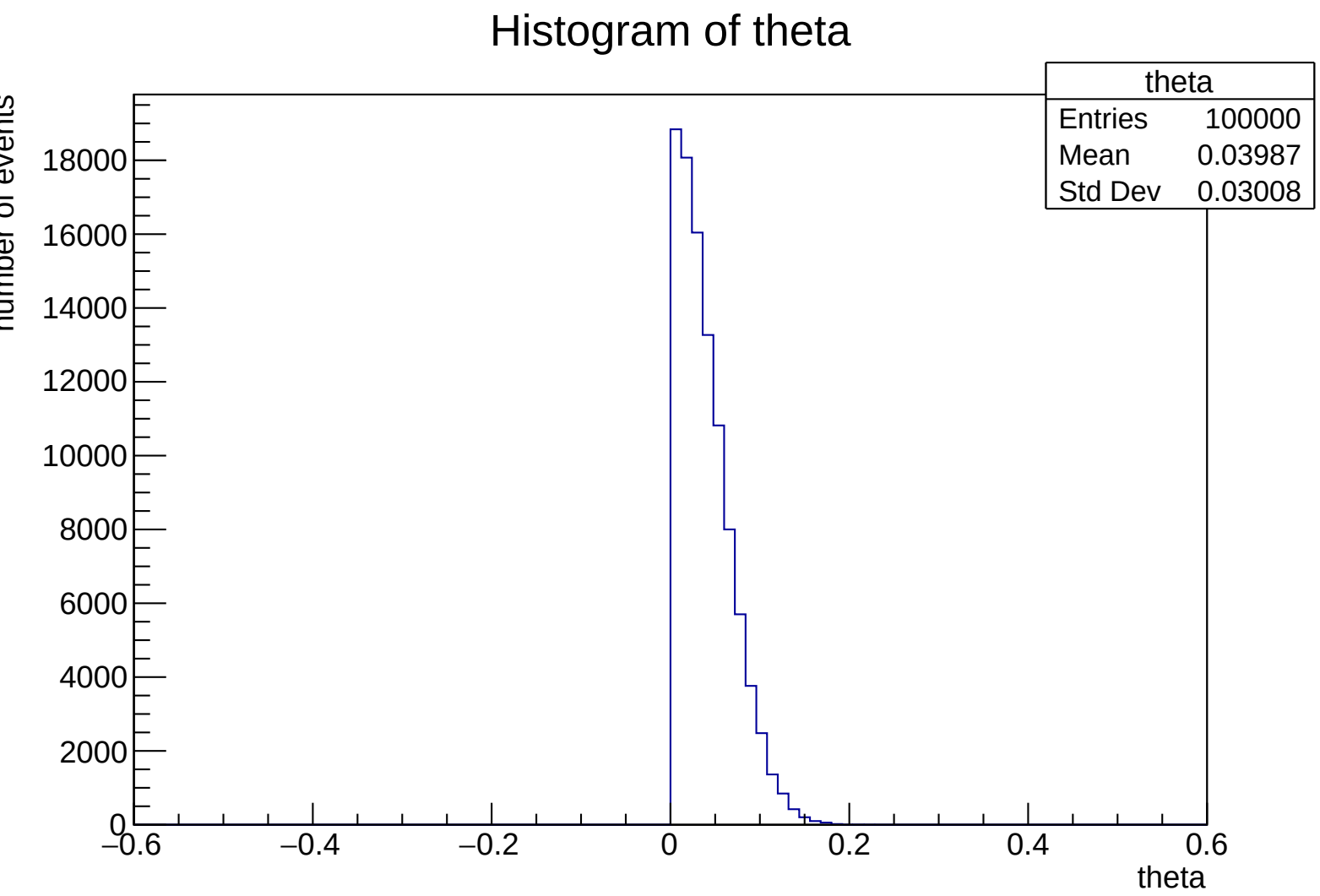
Como criamos, calculamos e preenchemos os histogramas ptHist e thetaHist, devemos fazer a declaração das variáveis do tipo Flot, para isso vamos usar um arquivo que está incluido no arquivo Analyze.C que é o arquivo Analyze.h, na parte Analyze Public. Vamos no final dela declarar as variáveis pt e theta:

```
class Analyze : public TSelector {
public :
   TTreeReader     fReader;  //!the tree reader
   TTree          *fChain = 0;   //!pointer to the analyzed TTree or TChain

```

```
   // Readers to access the data (delete the ones you do not need).
   TTreeReaderValue<Int_t> event = {fReader, "event"};
   TTreeReaderValue<Float_t> ebeam = {fReader, "ebeam"};
   TTreeReaderValue<Float_t> px = {fReader, "px"};
   TTreeReaderValue<Float_t> py = {fReader, "py"};
   TTreeReaderValue<Float_t> pz = {fReader, "pz"};
   TTreeReaderValue<Float_t> zv = {fReader, "zv"};
   TTreeReaderValue<Float_t> chi2 = {fReader, "chi2"};


   Analyze(TTree * /*tree*/ =0) { }
   virtual ~Analyze() { }
   virtual Int_t   Version() const { return 2; }
   virtual void    Begin(TTree *tree);
   virtual void    SlaveBegin(TTree *tree);
   virtual void    Init(TTree *tree);
   virtual Bool_t  Notify();
   virtual Bool_t  Process(Long64_t entry);
   virtual Int_t   GetEntry(Long64_t entry, Int_t getall = 0) { return fChain ?
       fChain->GetTree()->GetEntry(entry, getall) : 0; }
   virtual void    SetOption(const char *option) { fOption = option; }
   virtual void    SetObject(TObject *obj) { fObject = obj; }
   virtual void    SetInputList(TList *input) { fInput = input; }
   virtual TList  *GetOutputList() const { return fOutput; }
   virtual void    SlaveTerminate();
   virtual void    Terminate();
   // Declara as variaveis pt e theta
   Float_t pt;
   Float_t theta;
```

Segue as imagens dos histogramas pt e theta:


Histogram of pt
pt
Entries 100000
Mean 5.855
Std Dev 4.409
number of events
3000
2500
2000
1500
1000
500
0
0
5
10
15
20
25
pt (GeV)


Histogram of theta
theta
Entries 100000
Mean 0.03987
Std Dev 0.03008
number of events
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
–0.6
–0.4
–0.2
0
0.2
0.4
0.6
theta

**Exercício 7:**

Apply a cut in your analysis macro. Your goal is to count the number of events for which pz is less than 145 GeV, and then display the value.

Agora vamos aplicar o corte na variável pz e contar o número de eventos com $pz < 145\ GeV$

Para fazer a contagem do número de eventos, vamos ao arquivo Analyze.h e declarar dois contatores i e j começando com valor inicial zero:

```
class Analyze : public TSelector {
public :
   TTreeReader     fReader;  //!the tree reader
   TTree          *fChain = 0;   //!pointer to the analyzed TTree or TChain

   // Readers to access the data (delete the ones you do not need).
   TTreeReaderValue<Int_t> event = {fReader, "event"};
   TTreeReaderValue<Float_t> ebeam = {fReader, "ebeam"};
   TTreeReaderValue<Float_t> px = {fReader, "px"};
   TTreeReaderValue<Float_t> py = {fReader, "py"};
   TTreeReaderValue<Float_t> pz = {fReader, "pz"};
   TTreeReaderValue<Float_t> zv = {fReader, "zv"};
   TTreeReaderValue<Float_t> chi2 = {fReader, "chi2"};


   Analyze(TTree * /*tree*/ =0) { }
   virtual ~Analyze() { }
   virtual Int_t   Version() const { return 2; }
   virtual void    Begin(TTree *tree);
   virtual void    SlaveBegin(TTree *tree);
   virtual void    Init(TTree *tree);
   virtual Bool_t  Notify();
   virtual Bool_t  Process(Long64_t entry);
   virtual Int_t   GetEntry(Long64_t entry, Int_t getall = 0) { return fChain ?
       fChain->GetTree()->GetEntry(entry, getall) : 0; }
   virtual void    SetOption(const char *option) { fOption = option; }
   virtual void    SetObject(TObject *obj) { fObject = obj; }
   virtual void    SetInputList(TList *input) { fInput = input; }
   virtual TList  *GetOutputList() const { return fOutput; }
   virtual void    SlaveTerminate();
   virtual void    Terminate();
   // Declara as variaveis pt e theta.
   Float_t pt;
   Float_t theta;
   // Declara os contadores i e j, lembrando que eles serao usando no arquivo Analyze
       .c
   Int_t i=0;
   Int_t j=0;
   ClassDef(Analyze,0);
```

Agora vamos fazer os cortes para $*pz < 145$. Para fazer isso, vamos no arquivo Analyze.c no método Analyze Process:

```
void Analyze::SlaveBegin(TTree * /*tree*/){}

Bool_t Analyze::Process(Long64_t entry)
{
   //  D o n t  delete this line! Without it the program will crash
   fReader.SetLocalEntry(entry);

   //**********Loop section**********
   GetEntry(entry);
   chi2Hist->Fill(*chi2);
   // Preenche o ebeamHist com a vari vel ebeam
   ebeamHist->Fill(*ebeam);
```

```
   // Preenche o chi2ebeamHist com as vari veis ebeam e chi2
   chi2ebeamHist->Fill(*chi2,*ebeam);

   // Calcula o pt:
   pt = TMath::Sqrt((*px)*(*px)+(*py)*(*py));
   ptHist->Fill(pt);

   // Calcula o valor de theta:
   theta = TMath::ATan2((pt),(*pz));
   thetaHist->Fill(theta);

   // i conta o numero de eventos
   i++;
   // j conta o numero de eventos com *pz < 145 GeV
   if (TMath::Abs(*pz)<145.) {
      std::cout << *pz << j << std::endl;
      j++;
   }
```

Agora, vamos no método Analyze Terminate para colocar o número de eventos *pz ¡ 145 GeV.

Vamos aproveitar e usar o comando Draw para mostrar os histogramas ptHist e thetaHist criados anteriomente:

```
void Analyze::Terminate()
{
    //**********Wrap-up section**********
    // Desenha chi2Hist com barras de erros
    chi2Hist->Draw("BE1");

    //Desenha o histograma ebeam com barras de erros
    ebeamHist->Draw("BE1");
    // Faz o Fit do histrograma ebeamHist usando uma gaussiana e com barras de erros
    ebeamHist->Fit("gaus","V","BE1",148.,152.);

    // Desenha os histogramas ptHist e thetaHist
    thetaHist->Draw();
    ptHist->Draw();
    // Mostra o numero de eventos com *pz < 145 GeV
    std::cout << "The number of events with pz<145.0 GeV: " << j << std::endl;
```

**Exercício 7:**

Revise your code to write the histograms to a file.

Agora vamos salvar os histogramas em um arquivo.root, para fazer isso, vamos usar o método Analyze Terminate e recriar um arquivo chamado "histograma.root"onde serão escritos os histogramas desenvolvidos nos exercícios anteriores:

```
void Analyze::Terminate()
{
    //**********Wrap-up section**********
    // Desenha chi2Hist com barras de erros
    chi2Hist->Draw("BE1");

    //Desenha o histograma ebeam com barras de erros
    ebeamHist->Draw("BE1");
    // Faz o Fit do histrograma ebeamHist usando uma gaussiana e com barras de erros
    ebeamHist->Fit("gaus","V","BE1",148.,152.);

    // Desenha os histogramas ptHist e thetaHist
    thetaHist->Draw();
    ptHist->Draw();
    // Mostra o numero de eventos com *pz < 145 GeV
    std::cout << "The number of events with pz<145.0 GeV: " << j << std::endl;

```

```
    // Recria um arquivo chamado "histograma.root" com os histogramas criando ao
        longo do desenvolvimento
    TFile f("histograma.root","recreate");
    chi2Hist->Write();
    ebeamHist->Write();
    chi2ebeamHist->Write();
    ptHist->Write();
    thetaHist->Write();
    f.Write();
    f.Close();
}
    thetaHist->Write();
    f.Write();
    f.Close();
```

Os arquivos Analize.C, Analize.h, experiment.root e histograma.root e até mesmo este relatório podemos ser acessados na página do github: https://github.com/Silas-SJ/root-py-FAE

Lembrando que os histogramas mostrados foram gerados executando o arquivo Analyze.C no terminal do root:

Como eu estou usando o miniconda, eu usei os comandos:

```
\\ voce pode copiar o meu diretorio para o seu computador com o comando:

git clone https://github.com/Silas-SJ/root-py-FAE

\\ Em seguida acesse a pasta com o comando:
cd root-py-FAE

\\ my_root_env foi o nome que eu dei para o meu root, voce deve colocar o que voc
    escolheu quando instalou o root

conda activate my_root_env
root

\\ Voce pode compilar o arquivo Analyze.C com o comando:

.L Analyze.C

\\ Em seguida:

TFile * f = new TFile ("experiment.root") ; tree1 - > Process ("Analyze.C")

\\caso apareca algum erro, voc  pode fechar sair e entrar no root e seguir sem
    compilar indo direto para o comando

TFile * f = new TFile ("experiment.root") ; tree1 - > Process ("Analyze.C")
```