

## Relatório ROOT e PYROOT.

*Professores:* Sandro Fonseca, Sheila Amaral, Eliza Melo. *Name:* João Pedro Gomes Pinheiro (jgomespi).

**1ª Parte: Exercício baseado no Tutorial sobre o ROOT.**

Foi produzido um "esqueleto" chamado `Analyze` designado a fazer um *loop* sobre uma *tree* chamada `tree1` presente no arquivo `experiment.root`. A este "esqueleto" foram acrescentados alguns comandos pelos professores. Abaixo podemos ver uma cópia do arquivo `Analyze.cc` da forma como foi entregue aos alunos do curso. Ao decorrer do relatório, vamos mostrando quais comandos foram acrescentados, bem como seus objetivos. Ao fim, será mostrado o resultado final do arquivo `Analyze.cc`.

```

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     // Don'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 }

```

O código acima cria um histograma chamado `chi2Hist` no método `Analyze::Begin`, e acrescenta o nome dos eixos  $x$  e  $y$ . No método `Analyze::Process`, o histograma `chi2Hist` é preenchido com a variável `chi2`, presente em `tree1`. No método `Analyze::Terminate` utiliza-se o comando `Draw` para desenhar o histograma na tela.

Podemos acrescentar barras de erro no histograma `chi2Hist`, basta acrescentar a opção "E1" dentro do comando `Draw`, ou seja, a linha 38 do programa acima fica:

```

1  // Draw chi2Hist with error bars

```

```
2 chi2Hist->Draw("E1");
```

Além disso, o *range* do histograma foi alterado para de 0 a 2 de forma a facilitar a visualização das barras de erro.

Vamos criar um histograma para plotar a variável `ebeam` e um *scatterplot* com as variáveis `ebeam` e `chi2`. No método `Analyze::Begin`, acrescentamos os comandos que criam um histograma TH1 chamado `ebeamHist`, com 100 bins, que vai de 149,0 a 151,0. Além disso, criamos um histograma TH2 chamado `chi2ebeamHist`, que receberá as variáveis `ebeam` e `chi2`, este será nosso *scatterplot*. Observe que tanto o *range* quando a *binagem* são iguais às dos respectivos histogramas TH1 de cada variável. Também inserimos títulos nos eixos  $x$  e  $y$  de ambos os histogramas, como pode ser visto abaixo:

```
1 // Create a ebeamHist histogram, to Fill ebeam var, and set its axis titles.
2 ebeamHist = new TH1D("ebeam", "Histogram of ebeam", 100, 149., 151.);
3 ebeamHist->GetXaxis()->SetTitle("ebeam (GeV)");
4 ebeamHist->GetYaxis()->SetTitle("number of events");
5
6 // Create a scatterplot for ebeam and chi2 vars and set its axis titles.
7 chi2ebeamHist = new TH2D("chi2ebeam", "ScatterPlot of chi2 and ebeam", 100, 0, 2.,
8   100, 149., 151.);
9 chi2ebeamHist->GetXaxis()->SetTitle("chi2");
10 chi2ebeamHist->GetYaxis()->SetTitle("ebeam (GeV)");
```

No método `Analyze::Process` preenchemos os histogramas `ebeamHist` e `chi2ebeamHist`, como mostrado abaixo:

```
1 // Fill ebeamHist with ebeam var
2 ebeamHist->Fill(*ebeam);
3 // Fill chi2ebeamHist with ebeam and chi2 vars
4 chi2ebeamHist->Fill(*chi2,*ebeam);
```

No método `Analyze::Terminate`, desenhamos os histogramas da tela. Vamos ajustar uma gaussiana na distribuição de `ebeam` e desenhá-la com barras de erro com o seguinte comando:

```
1 //Fit a gaussian to ebeam distribution and draw ebeamHist with error bars
2 ebeamHist->Fit("gaus","V","E1",149.,151.);
```

Vamos desenhar o *scatterplot* com a paleta de cores, e para isso acrescentamos a opção `COLZ` no comando `Draw`. Além disso, ajustamos a posição no título do eixo  $y$ , que acaba ficando muito próximo do eixo. Também utilizamos comandos para ajustar a posição da caixa de estatística, que estava sobrepondo o paleta de cores.

```
1 // Set the position of Stat Box
2 gStyle->SetStatX(0.9);
3 gStyle->SetStatY(0.9);
4
5 // Set the position of chi2ebeamHist Y axis title
6 chi2ebeamHist->GetYaxis()->SetTitleOffset(1.4);
7 // Draw chi2ebeamHist scatterplot with "COLZ" option, which include the palette
8   colors
9 chi2ebeamHist->Draw("COLZ");
```

Agora, vamos calcular o  $p_T$  e o valor de  $\theta$ , a partir dos valores de  $p_x$  e  $p_y$  presentes em `tree1`. As equações 0.1 e 0.2 definem estas variáveis.

$$p_T = \sqrt{p_x^2 + p_y^2} \quad (0.1)$$

$$\theta = \arctan\left(\frac{y}{x}\right) \quad (0.2)$$

Primeiramente, no método `Analyze::Begin` definimos os respectivos histogramas TH1 que serão preenchidos com estas variáveis (`ptHist` e `thetaHist`) e escolhemos seu *range* e sua *binagem*.

```
1 // Create a ptHist histogram to Fill pT var and set its axis titles.
2 ptHist = new TH1D("pt", "Histogram of pT", 100, 0, 35);
3 ptHist->GetXaxis()->SetTitle("pT (GeV)");
4 ptHist->GetYaxis()->SetTitle("number of events");
5
```

```

6 // Create a thetaHist histogram to Fill theta var and set its axis titles.
7 thetaHist = new TH1D("theta", "Histogram of theta", 100, -3.15, 3.15);
8 thetaHist->GetXaxis()->SetTitle("theta");
9 thetaHist->GetYaxis()->SetTitle("number of events");

```

No arquivo `Analyze.h` acrescentamos as linhas abaixo para declarar as variáveis do tipo `Float_t` (`pT` e `theta`). Estas linhas devem ser acrescentadas dentro da classe `Analyze`, na parte `public`.

```

1 // Declaring pT and theta vars.
2 Float_t pT;
3 Float_t theta;

```

No método `Analyze::Process`, utilizando a biblioteca `TMath`, calculamos  $p_T$  e  $\theta$  com base nas Equações 0.1 e 0.2. Por fim, preenchemos os respectivos histogramas.

```

1 // Calc pT:
2 pT = TMath::Sqrt((*px)*(*px)+(*py)*(*py));
3 ptHist->Fill(pT);
4
5 // Calc theta:
6 theta = TMath::ATan2((*py),(*px));
7 thetaHist->Fill(theta);

```

No método `Analyze::Terminate`, restauramos a posição da caixa de estatística com o comando `Reset` e ajustamos a posição do título do eixo  $y$  do histograma `ptHist`. Por fim, desenhmos os histogramas `thetaHist` e `ptHist`.

```

1 // Reset the stat box
2 gStyle->Reset();
3 // Set the position of ptHist Y axis title
4 ptHist->GetYaxis()->SetTitleOffset(1.4);
5 // Draw thetaHist and ptHist
6 thetaHist->Draw();
7 ptHist->Draw();

```

Para contar a quantidade de eventos com  $p_z < 145,0$  GeV, definimos dois contadores ( $i$  e  $j$ ) no arquivo `Analyze.h`, logo abaixo da definição de `pT` e `theta`, e atribuímos valor zero a eles:

```

1 // Declaring i and j to count the number of eventos above some cut defined on .C file
2 Int_t i=0;
3 Int_t j=0;

```

Agora sim vamos realizar a contagem dos eventos que possuem  $p_z < 145,0$  GeV. No arquivo `Analyze.C`, no método `Analyze::Process`, acrescentamos os seguintes comandos:

```

1 // j will count all events
2 j++;
3 // i will count events with pz < 145.0 GeV
4 if (TMath::Abs(*pz)<145.) {
5     // Here we print the value of pz (when pz<145 GeV) on screen
6     std::cout << *pz << i << std::endl;
7     i++;
8 }

```

Para descobrirmos quantos eventos possuem  $p_z < 145,0$  GeV, inserimos o comando abaixo no método `Analyze::Terminate`:

```

1 // Print on screen how many events have pz<145 GeV
2 std::cout << "The number of events with pz<145.0 GeV is " << i << std::endl;

```

Para criar o arquivo `.root` (`experiment-output.root`) e escrever os histogramas nele, devemos inserir os comandos abaixo no método `Analyze::Terminate`:

```

1 // Recreate a file named "experiment-output.root" and write the histograms
2 TFile f("experiment-output.root","recreate");
3 chi2Hist->Write();
4 ebeamHist->Write();
5 chi2ebeamHist->Write();
6 ptHist->Write();

```

```

7  thetaHist->Write();
8  f.Write();
9  f.Close();

```

Os programas e arquivos necessários para rodar esta análise estão num repositório GitHub que pode ser executado desta forma:

```

1  git clone git@github.com:jgomespi/ROOT-AnaliseDados.git
2  cd ROOT-AnaliseDados/

```

Entre no root e compile o programa:

```

1  root -l
2  [0] .L Analyze.C

```

Para rodar o programa, basta executar o comando abaixo numa sessão root:

```

1  [1] TFile *f = new TFile("experiment.root"); tree1->Process("Analyze.C")

```

Os outputs serão os valores de  $p_z$  (para eventos onde este é menor que 145,0 GeV), os parâmetros do ajuste do histograma *ebeamHist* e, por fim, o número de eventos com  $p_z < 145,0$  GeV. Além disso, serão desenhados os histogramas na tela e o arquivo *experiment-output.root* é criado com todos os histogramas citados aqui.

Abaixo deixamos uma cópia da versão final dos arquivos *Analyze.C* e *Analyze.h*, que também podem ser encontrados no GitHub.

```

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  TH1* ebeamHist = NULL;
9  TH2* chi2ebeamHist = NULL;
10 TH1* thetaHist = NULL;
11 TH1* ptHist = NULL;
12
13 void Analyze::Begin(TTree * /*tree*/)
14 {
15     TString option = GetOption();
16
17     //*****Initialization section*****
18     chi2Hist = new TH1D("chi2", "Histogram of Chi2", 100, 0, 2.);
19     chi2Hist->GetXaxis()->SetTitle("chi2");
20     chi2Hist->GetYaxis()->SetTitle("number of events");
21
22     // Create a ebeamHist histogram, to Fill ebeam var, and set its axis titles.
23     ebeamHist = new TH1D("ebeam", "Histogram of ebeam", 100, 149., 151.);
24     ebeamHist->GetXaxis()->SetTitle("ebeam (GeV)");
25     ebeamHist->GetYaxis()->SetTitle("number of events");
26
27     // Create a scatterplot for ebeam and chi2 vars and set its axis titles.
28     chi2ebeamHist = new TH2D("chi2ebeam", "ScatterPlot of chi2 and ebeam", 100, 0, 2.,
29                               100, 149., 151.);
30     chi2ebeamHist->GetXaxis()->SetTitle("chi2");
31     chi2ebeamHist->GetYaxis()->SetTitle("ebeam (GeV)");
32
33     // Create a ptHist histogram to Fill pT var (Float_t type, also created here) and set
34     // its axis titles.
35     ptHist = new TH1D("pt", "Histogram of pT", 100, 0, 35);
36     ptHist->GetXaxis()->SetTitle("pT (GeV)");
37     ptHist->GetYaxis()->SetTitle("number of events");
38
39     // Create a thetaHist histogram to Fill theta var (Float_t type, also created here)
40     // and set its axis titles.
41     thetaHist = new TH1D("theta", "Histogram of theta", 100, -3.15, 3.15);
42     thetaHist->GetXaxis()->SetTitle("theta");

```

```

40  thetaHist->GetYaxis()->SetTitle("number of events");
41
42
43  }
44
45  void Analyze::SlaveBegin(TTree * /*tree*/){}
46
47  Bool_t Analyze::Process(Long64_t entry)
48  {
49  // Dont delete this line! Without it the program will crash
50  fReader.SetLocalEntry(entry);
51
52  //*****Loop section*****
53  GetEntry(entry);
54  chi2Hist->Fill(*chi2);
55  // Fill ebeamHist with ebeam var
56  ebeamHist->Fill(*ebeam);
57  // Fill chi2ebeamHist with ebeam and chi2 vars
58  chi2ebeamHist->Fill(*chi2,*ebeam);
59
60  // Calc pT:
61  pT = TMath::Sqrt((*px)*(*px)+(*py)*(*py));
62  ptHist->Fill(pT);
63
64  // Calc theta:
65  theta = TMath::ATan2((*py),(*px));
66  thetaHist->Fill(theta);
67
68  // j will count all events
69  j++;
70  // i will count events with pz < 145.0 GeV
71  if (TMath::Abs(*pz)<145.) {
72  // Here we print the value of pz (when pz<145 GeV) on screen
73  std::cout << *pz << i << std::endl;
74  i++;
75  }
76
77  return kTRUE;
78  }
79
80  void Analyze::SlaveTerminate(){}
81
82  void Analyze::Terminate()
83  {
84  //*****Wrap-up section*****
85  // Draw chi2Hist with error bars
86  chi2Hist->Draw("E1");
87
88  //Fit a gaussian to ebeam distribution and draw ebeamHist with error bars
89  ebeamHist->Fit("gaus","V","E1",149.,151.);
90
91  // Set the position of Stat Box
92  gStyle->SetStatX(0.9);
93  gStyle->SetStatY(0.9);
94
95  // Set the position of chi2ebeamHist Y axis title
96  chi2ebeamHist->GetYaxis()->SetTitleOffset(1.4);
97  // Draw chi2ebeamHist scatterplot with "COLZ" option, which include the palette
   colors
98  chi2ebeamHist->Draw("COLZ");
99
100 // Reset the stat box
101 gStyle->Reset();

```

```

102 // Set the position of ptHist Y axis title
103 ptHist->GetYaxis()->SetTitleOffset(1.4);
104 // Draw thetaHist and ptHist
105 thetaHist->Draw();
106 ptHist->Draw();
107
108 // Print on screen how many events have pz<145 GeV
109 std::cout << "The number of events with pz<145.0 GeV is " << i << std::endl;
110
111 // Recreate a file named "experiment-output.root" and write the histograms
112 TFile f("experiment-output.root","recreate");
113 chi2Hist->Write();
114 ebeamHist->Write();
115 chi2ebeamHist->Write();
116 ptHist->Write();
117 thetaHist->Write();
118 f.Write();
119 f.Close();
120 }

```

```

1  #ifndef Analyze_h
2  #define Analyze_h
3
4  #include <TRoot.h>
5  #include <TChain.h>
6  #include <TFile.h>
7  #include <TSelector.h>
8  #include <TTreeReader.h>
9  #include <TTreeReaderValue.h>
10 #include <TTreeReaderArray.h>
11
12 // Headers needed by this particular selector
13
14
15 class Analyze : public TSelector {
16 public :
17     TTreeReader      fReader;  //!

```

```
44 virtual void    Terminate();
45 // Declaring pT and theta vars.
46 Float_t pT;
47 Float_t theta;
48 // Declaring i and j to count the number of eventos above some cut defined on .C file
49 Int_t i=0;
50 Int_t j=0;
51 ClassDef(Analyze,0);
52
53 };
54
55 #endif
56
57 #ifdef Analyze_cxx
58 void Analyze::Init(TTree *tree)
59 {
60 fReader.SetTree(tree);
61 }
62
63 Bool_t Analyze::Notify()
64 {
65 return kTRUE;
66 }
67
68
69 #endif // #ifdef Analyze_cxx
```