analysis.cpp

```cpp
#include <TApplication.h>
#include <TCanvas.h>
#include <TF1.h>
#include <TFile.h>
#include <TH1F.h>
#include <TROOT.h>
#include <TStyle.h>
#include <TSystem.h>

#include <iomanip>
#include <iostream>

int main() {
  TApplication theApp("App", 0, 0);
  // Apri il file ROOT
  TCanvas*     canvas  = new TCanvas("c1", "Analysis", 800, 600);
  TCanvas*     canvas2 = new TCanvas("c2", "Analysis", 800, 600);
  canvas->Divide(3, 2);
  canvas->cd(1);

  TFile* file = TFile::Open("IstogrammiParticelle.root");

  if (!file || file->IsZombie()) {
    std::cerr << "Error during ROOT file opening." << std::endl;
    return 1;
  }

  // Caricamento degli istogrammi
  TH1D* hInvariantMass        = (TH1D*)file->Get("hInvariantMass");
  TH1D* hMassSameSign         = (TH1D*)file->Get("hMassSameSign");
  TH1D* hMassOppositeSign     = (TH1D*)file->Get("hMassOppositeSign");
  TH1D* hMassPionKaonOpposite = (TH1D*)file->Get("hMassPionKaonOpposite");
  TH1D* hMassPionKaonSame     = (TH1D*)file->Get("hMassPionKaonSame");
  TH1D* hMassKStarDecay       = (TH1D*)file->Get("hMassKStarDecay");
  TH1D* hType                 = (TH1D*)file->Get("hType");
  TH1D* hEnergy               = (TH1D*)file->Get("hEnergy");
  TH1D* hTheta                = (TH1D*)file->Get("hTheta");
  TH1D* hPhi                  = (TH1D*)file->Get("hPhi");
  TH1D* hPout                 = (TH1D*)file->Get("hPout");
  TH1D* hPtrasv               = (TH1D*)file->Get("hPtrasv");

  if (!hInvariantMass || !hMassOppositeSign || !hMassSameSign
      || !hMassPionKaonOpposite || !hMassPionKaonSame || !hMassKStarDecay
      || !hType || !hEnergy || !hTheta || !hPhi || !hPout || !hPtrasv) {
    std::cerr << "Error during histograms loading." << std::endl;
    file->Close();
    return 2;
  }
```

```
49
50    hType->GetXaxis()->SetBinLabel(1, "\u03C0+");
51    hType->GetXaxis()->SetBinLabel(2, "\u03C0-");
52
53    // costruisco un vettore di istogrammi
54    std::array<TH1D*, 12> h_array = {hType,
55                                     hInvariantMass,
56                                     hMassOppositeSign,
57                                     hMassSameSign,
58                                     hMassPionKaonOpposite,
59                                     hMassPionKaonSame,
60                                     hMassKStarDecay,
61                                     hEnergy,
62                                     hTheta,
63                                     hPhi,
64                                     hPout,
65                                     hPtrasv};
66
67    std::array<double, 12> expected_array = {
68      1.01e+07, 5.05e+08, 2.55e+08, 2.50e+08, 4.40e+07, 4.40e+07,
69      1.00e+05, 1.01e+07, 1.01e+07, 1.01e+07, 1.01e+07, 1.01e+07,
70    };
71
72    // Numero di ingressi
73    for (int i = 0; i < 12; ++i) {
74      std::cout << "Entries in " << std::setw(21) << h_array[i]->GetName() << ": "
75                << std::setw(11) << h_array[i]->GetEntries()
76                << ", expected: " << expected_array[i] << std::endl;
77    }
78    std::cout << '\n';
79
80    // verifico distibuzione particelle
81    for (int i = 1; i <= hType->GetNbinsX(); i++) {
82      std::cout << "Number of " << hType->GetXaxis()->GetBinLabel(i) << ": "
83                << std::setw(11) << hType->GetBinContent(i)
84                << "  Percentage: " << std::setw(7)
85                << hType->GetBinContent(i) / hType->GetEntries() * 100 << "%"
86                << std::endl;
87    }
88    std::cout << '\n';
89
90    // Fit della distribuzione angolare con una funzione uniforme
91    TF1* f_uniform = new TF1("f_uniform", "[0] * 100000");
92    f_uniform->Print();
93    std::cout << '\n';
94
95    std::cout << "\u03C6 uniform fit results:" << std::endl;
96    hPhi->Fit("f_uniform", "Q");
97    std::cout << "  Constant:\t" << f_uniform->GetParameter(0) << " ± "
98              << f_uniform->GetParError(0) << std::endl;
```

```
99    std::cout << "  \u03C7²:\t\t" << f_uniform->GetChisquare() << std::endl;
100   std::cout << "  NDF:\t\t" << f_uniform->GetNDF() << std::endl;
101   std::cout << "  \u03C7²/NDF:\t"
102             << f_uniform->GetChisquare() / f_uniform->GetNDF() << std::endl;
103   std::cout << "  Probability:\t" << f_uniform->GetProb() << std::endl;
104   std::cout << '\n';
105   hPhi->Draw("P SAME");
106
107   canvas->cd(2);
108   std::cout << "\u03B8 uniform fit results:" << std::endl;
109   hTheta->Fit("f_uniform", "Q");
110   std::cout << "  Constant:\t" << f_uniform->GetParameter(0) << " ± "
111             << f_uniform->GetParError(0) << std::endl;
112   std::cout << "  \u03C7²:\t\t" << f_uniform->GetChisquare() << std::endl;
113   std::cout << "  NDF:\t\t" << f_uniform->GetNDF() << std::endl;
114   std::cout << "  \u03C7²/NDF:\t"
115             << f_uniform->GetChisquare() / f_uniform->GetNDF() << std::endl;
116   std::cout << "  Probability:\t" << f_uniform->GetProb() << std::endl;
117   std::cout << '\n';
118   hTheta->Draw("P SAME");
119
120   canvas->cd(3);
121   // Fit della distribuzione del modulo dell'impulso con una funzione
122   // esponenziale
123   TF1* f_exponential = new TF1("f_exponential", "[0]*exp(-[1]*x)");
124
125   f_exponential->Print();
126   std::cout << "Momentum exponential fit results:" << std::endl;
127   hPout->Fit("f_exponential", "Q");
128   std::cout << "  Intercept:\t" << f_exponential->GetParameter(0) << " ± "
129             << f_exponential->GetParError(0) << std::endl;
130   std::cout << "  Slope:\t" << f_exponential->GetParameter(1) << " ± "
131             << f_exponential->GetParError(1) << std::endl;
132   std::cout << "  Average:\t" << f_exponential->Mean(0, 6) << " ± "
133             << f_exponential->GetParError(1) * f_exponential->Mean(0, 6)
134                    * f_exponential->Mean(0, 6)
135             << std::endl;
136   std::cout << "  \u03C7²:\t\t" << f_exponential->GetChisquare() << std::endl;
137   std::cout << "  NDF:\t\t" << f_exponential->GetNDF() << std::endl;
138   std::cout << "  \u03C7²/NDF:\t"
139             << f_exponential->GetChisquare() / f_exponential->GetNDF()
140             << std::endl;
141   std::cout << "  Probability:\t" << f_exponential->GetProb() << std::endl;
142   std::cout << '\n';
143   hPout->Draw("P SAME");
144
145   canvas->cd(4);
146   TF1* f_gaussian = new TF1("f_gaussian", "gaus(0)", 0.7, 1.1);
147   f_gaussian->SetLineColor(kBlack);
148
```

```
149   // sottraggo gli istogrammi delle masse invarianti con carica opposta e stessa
150   // carica
151   TH1F* hInvariantMassSub1 = new TH1F(
152       "hInvariantMassSub1",
153       "Difference in invariant mass between concordant and discordant charge particles",
154       hMassSameSign->GetNbinsX(), 0, 6);
155   hInvariantMassSub1->Add(hMassOppositeSign, 1);
156   hInvariantMassSub1->Add(hMassSameSign, -1);
157   hInvariantMassSub1->GetXaxis()->SetTitle("Invariant mass [GeV/c^{2}]");
158   hInvariantMassSub1->GetYaxis()->SetTitle("Events");
159   hInvariantMassSub1->SetLineColor(kGreen);
160
161   // Eseguiamo il fit sull'istogramma risultato dalla sottrazione
162   hInvariantMassSub1->Fit("f_gaussian", "Q", nullptr, 0.7, 1.1);
163
164   hInvariantMassSub1->Draw("HIST SAME");
165
166   // Stampiamo i parametri del fit
167   std::cout << "Concordant - discordant particles gaussian fit results:"
168             << std::endl;
169   std::cout << "  Amplitude:\t" << f_gaussian->GetParameter(0) << " ± "
170             << f_gaussian->GetParError(0) << std::endl;
171   std::cout << "  K* mass:\t" << f_gaussian->GetParameter(1) << " ± "
172             << f_gaussian->GetParError(1) << std::endl;
173   std::cout << "  K* width:\t" << f_gaussian->GetParameter(2) << " ± "
174             << f_gaussian->GetParError(2) << std::endl;
175   std::cout << "  \u03C7²:\t\t" << f_gaussian->GetChisquare() << std::endl;
176   std::cout << "  NDF:\t\t" << f_gaussian->GetNDF() << std::endl;
177   std::cout << "  \u03C7²/NDF:\t"
178             << f_gaussian->GetChisquare() / f_gaussian->GetNDF() << std::endl;
179   std::cout << "  Probability:\t" << f_gaussian->GetProb() << std::endl;
180   std::cout << '\n';
181
182   canvas->cd(5);
183
184   TH1F* hInvariantMassSub2 = new TH1F(
185       "hInvariantMassSub2",
186       "Difference in invariant mass between concordant and discordant #pi/K couples",
187       hMassPionKaonSame->GetNbinsX(), 0, 6);
188   hInvariantMassSub2->Add(hMassPionKaonOpposite, 1);
189   hInvariantMassSub2->Add(hMassPionKaonSame, -1);
190   hInvariantMassSub2->GetXaxis()->SetTitle("Invariant mass [GeV/c^{2}]");
191   hInvariantMassSub2->GetYaxis()->SetTitle("Events");
192   hInvariantMassSub2->SetLineColor(kRed);
193
194   // Eseguiamo il fit sull'istogramma risultato dalla sottrazione
195   hInvariantMassSub2->Fit("f_gaussian", "Q", nullptr, 0.7, 1.1);
196
197   hInvariantMassSub2->Draw("HIST SAME");
198
```

```
199   // Stampiamo i parametri del fit
200   std::cout << "Concordant - discordant \u03C0/K couples gaussian fit results:"
201             << std::endl;
202   std::cout << "  Amplitude:\t" << f_gaussian->GetParameter(0) << " ± "
203             << f_gaussian->GetParError(0) << std::endl;
204   std::cout << "  K* mass:\t" << f_gaussian->GetParameter(1) << " ± "
205             << f_gaussian->GetParError(1) << std::endl;
206   std::cout << "  K* width:\t" << f_gaussian->GetParameter(2) << " ± "
207             << f_gaussian->GetParError(2) << std::endl;
208   std::cout << "  \u03C7²:\t\t" << f_gaussian->GetChisquare() << std::endl;
209   std::cout << "  NDF:\t\t" << f_gaussian->GetNDF() << std::endl;
210   std::cout << "  \u03C7²/NDF:\t"
211             << f_gaussian->GetChisquare() / f_gaussian->GetNDF() << std::endl;
212   std::cout << "  Probability:\t" << f_gaussian->GetProb() << std::endl;
213   std::cout << '\n';
214
215   canvas->cd(6);
216
217   // Eseguiamo il fit sull'istogramma risultato dalla sottrazione
218   hMassKStarDecay->Fit("f_gaussian", "Q", nullptr, 0.7, 1.1);
219
220   hMassKStarDecay->Draw("HIST SAME");
221
222   // Stampiamo i parametri del fit
223   std::cout << "True K* decays gaussian fit results:" << std::endl;
224   std::cout << "  Amplitude:\t" << f_gaussian->GetParameter(0) << " ± "
225             << f_gaussian->GetParError(0) << std::endl;
226   std::cout << "  K* mass:\t" << f_gaussian->GetParameter(1) << " ± "
227             << f_gaussian->GetParError(1) << std::endl;
228   std::cout << "  K* width:\t" << f_gaussian->GetParameter(2) << " ± "
229             << f_gaussian->GetParError(2) << std::endl;
230   std::cout << "  \u03C7²:\t\t" << f_gaussian->GetChisquare() << std::endl;
231   std::cout << "  NDF:\t\t" << f_gaussian->GetNDF() << std::endl;
232   std::cout << "  \u03C7²/NDF:\t"
233             << f_gaussian->GetChisquare() / f_gaussian->GetNDF() << std::endl;
234   std::cout << "  Probability:\t" << f_gaussian->GetProb() << std::endl;
235   std::cout << '\n';
236
237   canvas2->cd();
238   hInvariantMassSub1->Draw("HIST");
239   hInvariantMassSub2->Draw("HIST SAME");
240   hMassKStarDecay->Draw("HIST SAME");
241
242   while (gROOT->GetListOfCanvases()->FindObject("c1")
243          && gROOT->GetListOfCanvases()->FindObject("c2")) {
244     gSystem->ProcessEvents(); // Process any events (including canvas events)
245     gSystem->Sleep(100);      // Add a small delay to prevent 100% CPU usage
246   }
247
248   std::cout << "Canvas closed. Exiting program." << std::endl;
```

```
249
250   return 0;
251 }
```