Relazione ROOT

Nicolò Montalti

6 gennaio 2021

1 Introduzione

Il programma si divide in due parti: una di generazione e una di analisi. Nella prima vengono generati 10^5 gruppi di 100 particelle ciascuno secondo proporzioni predefinite. Le particelle possono essere pioni, kaoni o protoni, tutte di carica positiva o negativa. In aggiunta è possibile produrre kaoni di risonanza, dal cui decadimento si ottengono un pione e un kaone di carica opposta.

Durante la fase di analisi si verifica che la generazione sia coerente con i parametri impostati. Si controllano l'abbondanza dei vari tipi di particelle, la distribuzione del modulo dell'impulso e la sua direzione. Si cerca inoltre di risalire alla massa e alla vita media delle K^* decadute. Ciò avviene analizzando gli istogrammi della massa invariante calcolata tra vari gruppi di particelle. La massa invariante tra i prodotti delle K^* decadute si distribuisce infatti secondo una distribuzione gaussiana la cui media è legata alla massa delle K^* e la deviazione standard all'inverso della vita media.

2 Struttura del codice

Alla base del programma ci sono tre classi: ParticleType, ResonanceType e Particle. La classe ParticleType (list. 1-2) contiene la massa, la carica e il nome di ogni tipo di particella. Dispone inoltre dei rispettivi getters e di un metodo Print che permette di stampare a schermo ogni informazione. La classe ResonanceType (list. 3-4) eredita da ParticleType e aggiunge la larghezza di risonanza con il rispettivo getter. Sovrascrive inoltre il metodo Print, in modo da stampare anche quest'ultimo attributo. Si è scelto di fare ereditare ResonanceType da ParticleType perché le due classi sono in una relazione di tipo "is-a". Ogni particella di risonanza è anche una particella in senso generale e ne possiede tutti gli attributi (nome, massa e carica).

La classe Particle (list. 5-6) aggiunge le proprietà cinematiche tipiche di una particella fisica, cioè le tre componenti della quantità di moto. Per integrare le informazioni sulla specie si è scelto di non fare ereditare Particle da ParticleType. Se si fosse sfruttata l'ereditarietà, per ogni istanza di Particle sarebbero stati salvati in memoria nome, massa e carica, sprecando risorse. Si è preferito utilizzare l'aggregazione e includere nella classe Particle un array statico di puntatori ParticleType e un indice intero. L'array è riempito tramite il metodo statico AddParticleType all'inizio del programma. Ogni istanza di Particle viene inizializzata con un indice che indica il tipo di particella. Oltre ai canonici getters e setters, Particle dispone di un metodo Decay2Body che permette ai K^* di decadere in un pione e in un kaone di carica opposta.

3 Generazione

La generazione avviene nel file main.cpp (list. 7). Inizialmente vengono generati 10^5 eventi di 100 particelle ciascuno. Ad ogni particella viene assegnata una specie casuale attraverso un blocco if-else if-else secondo le proporzioni definite in tab. 1. La quantità di moto è stabilita estraendone il modulo da una distribuzione esponenziale di media 1 GeV. Gli angoli azimutali e polari si ricavano invece da due distribuzioni uniformi definite sugli intervalli $[0, \pi]$ e $[0, 2\pi]$. Le particelle K^* vengono fatte decadere in un π^+ e un K^- o in π^- e un K^+ con pari probabilità. Tutte le estrazioni si basano sulle distribuzioni standard di ROOT, richiamate attraverso il puntatore globale gRandom.

Durante la generazione vengono riempiti una serie di istogrammi. Vengono salvati i tipi di particella generati, il modulo e la direzione dell'impulso, l'energia e la massa invariante. In particolare per la massa invariante vengono creati cinque istogrammi. Nel primo la massa invariante viene calcolata tra tutte le particelle di ogni evento; in altri due solo tra le particelle di carica uguale e di carica opposta; negli ultimi solo tra un pione e un kaone di carica uguale e di carica opposta. Tutti gli istogrammi vengono poi salvati in un file ROOT per essere analizzati successivamente.

Tabella 1: Proporzioni e caratteristiche delle particelle generate

Particella	Carica (e)	Simbolo	Percentuale
Pione	+1	π^+	40%
	-1	π^-	40%
Kaone	+1	K^+	5%
	-1	K^-	5%
	0	K^*	1%
Protone	+1	p^+	4.5%
	-1	p^-	4.5%

Tabella 2: Abbondanza attesa e osservata delle particelle generate divise per tipo

Specie	Occorrenze osservate (10^5)	Occorrenze attese (10^5)
π^+	40.01 ± 0.02	40
π^-	40.00 ± 0.02	40
K^+	4.995 ± 0.007	5.0
K^-	4.996 ± 0.007	5.0
p^+	4.493 ± 0.007	4.5
p^-	4.495 ± 0.007	4.5
K^*	1.007 ± 0.003	1.0

4 Analisi

L'analisi è codificata nel file analysis.cpp (list. 8). Come si può osservare nella tab. 2 e nel primo istogramma in fig. 1, l'abbondanza delle varie specie è compatibile con i valori attesi. Il modulo dell'impulso si distribuisce secondo una distribuzione esponenziale di media compatibile con il valore atteso 1 GeV e con un χ^2 accettabile. Allo stesso modo gli angoli azimutali e polari dell'impulso sono compatibili con una distribuzione uniforme definita sui rispettivi domini.

Per risalire alle caratteristiche delle K^* decadute si è sfruttato il fatto che ogni decadimento produca un kaone e un pione di carica opposta. L'insieme delle particelle di carica concorde contiene quindi solo particelle che non sono state originate da un decadimento, mentre quello delle particelle di carica opposta contiene le particelle originate dai decadimenti più le altre. Sottraendo il secondo dal primo si elimina gran parte del fondo delle particelle non originate dai decadimenti. Fittando l'istogramma delle masse invarianti tra questo gruppo di particelle si ottiene una gaussiana con media pari alla massa delle K^* e deviazione standard pari alla larghezza di risonanza. In tab. 4 è riportato l'esito del fit.

Per migliorare la precisione della misura si è riutilizzato il metodo appena descritto restringendolo ai soli pioni e kaoni, dato che i decadimenti non producono protoni. In tab. 4 sono riportati l'esito del fit e un fit gaussiano di controllo eseguito sulla massa invariante tra le coppie πK prodotte dai decadimenti. In fig. 2 sono riportati i tre istogrammi della massa invariante coi rispettivi fit.

Gli esiti di tutti e tre i metodi di calcolo restituiscono χ^2 accettabili e sono compatibili con i valori attesi di media (0.89166 GeV/c^2) e deviazione standard (0.050 GeV/c^2). È infine rilevante notare come restringendo il calcolo ai soli pioni e kaoni l'errore si riduca sensibilmente.

Tabella 3: Fit del modulo e degli angoli azimutali e polari dell'impulso delle particelle generate

Variabile	Distribuzione	Parametro del fit	χ^2	DOF	χ^2/DOF
Modulo	expo	(1.0001 ± 0.0003) GeV	23.79	18	1.32
Angolo azimutale	pol0	$(10000 \pm 3) \cdot 10^2$ rad^{-1}	9.50	9	1.06
Angolo polare	pol0	$(10000 \pm 3) \cdot 10^2$ rad^{-1}	14.03	9	1.56

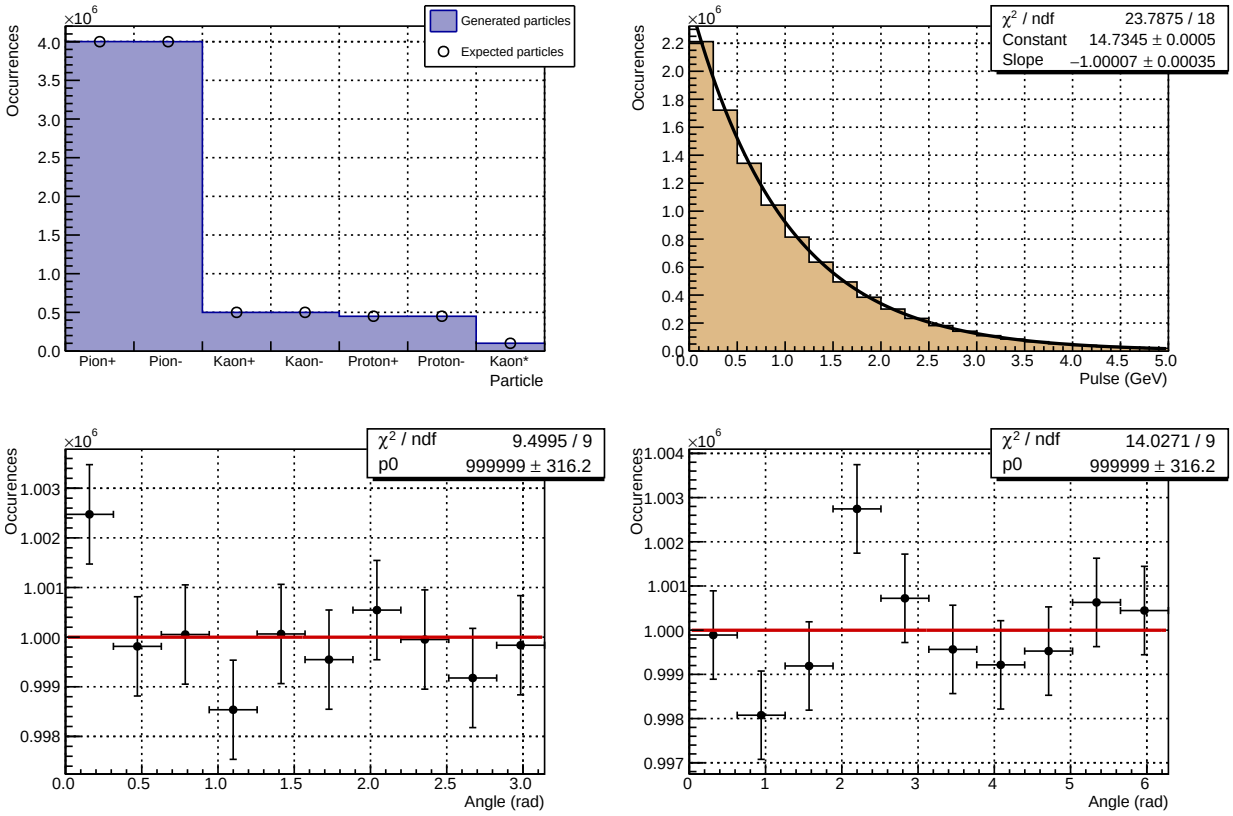

Figura 1: Istogrammi delle particelle generate e attese divise per specie (in alto a sx), del modulo dell'impulso con fit esponenziale (in alto a dx) e degli angoli azimutali e polari con fit pol0 (rispettivamente in basso a sx e a dx)

Tabella 4: Fit degli istogrammi della massa invariante calcolata tra varie combinazioni di particelle

Distribuzione gaussiana	Media (Gev/c^2)	Sigma (GeV/c^2)	Ampiezza (10^4)	χ^2/DOF
Massa invariante ottenuta dalla differenza delle combinazioni di particelle di carica discorde e concorde	0.888 ± 0.005	0.054 ± 0.006	3.8 ± 0.3	0.43
Massa invariante ottenuta da differenza delle combinazioni di particelle πK di carica discorde e concorde	0.892 ± 0.002	0.049 ± 0.002	1.99 ± 0.07	1.36
Massa invariante ottenuta dalle particelle generate dai decadimenti delle K^*	0.89146 ± 0.00016	0.05015 ± 0.00011	1.335 ± 0.005	1.57

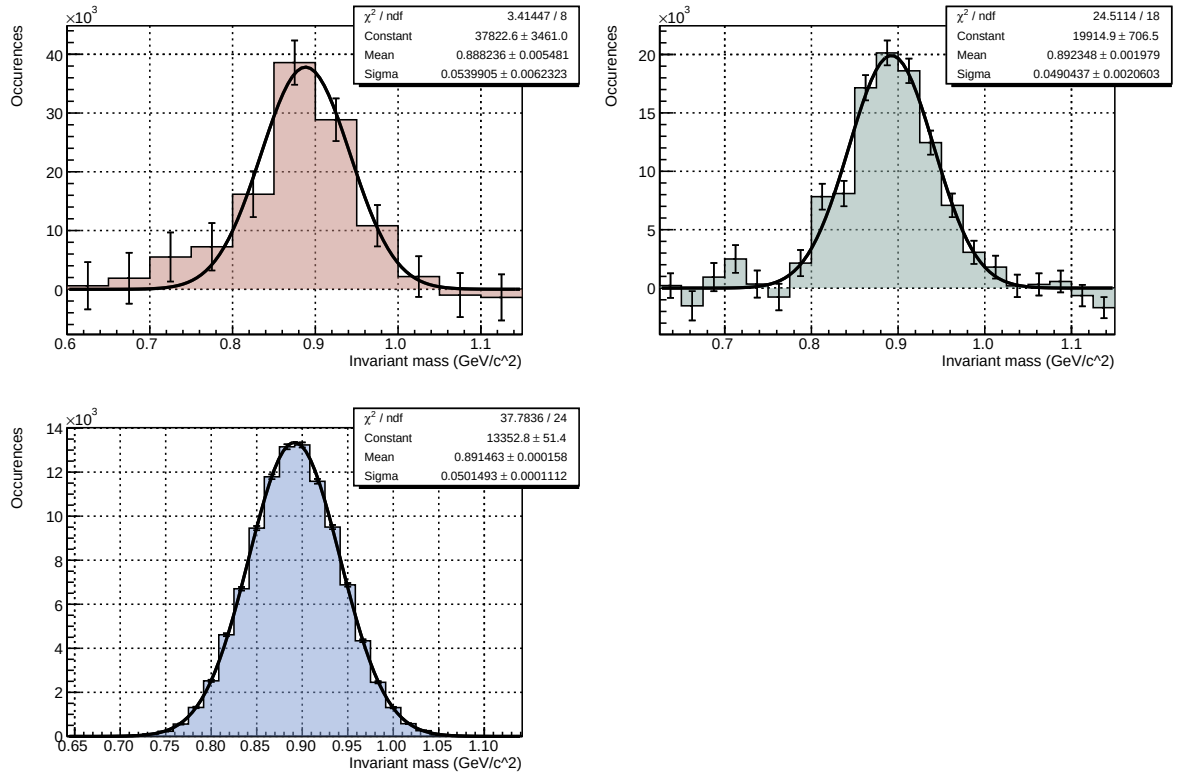

Figura 2: Istogrammi della massa invariante ottenuta dalla differenza di combinazioni di particelle di carica uguale ed opposta (in alto a sx), dalla differenza delle combinazioni πK di carica uguale ed opposta (in alto a dx) e dalle coppie πK generate dai decadimenti delle K^* (in basso a sx)

Indice dei listati

Listato 1: particleType.hpp

```
#ifndef PARTICLE_TYPE_HPP
#define PARTICLE_TYPE_HPP

#include <string>

class ParticleType
{
 public:
  ParticleType(std::string name = std::string{},
               double mass = 0.,
               int charge = 0);
  std::string GetName() const { return fName; }
  double GetMass() const { return fMass; }
  double GetCharge() const { return fCharge; }
  virtual void Print() const;
  virtual double GetWidth() const;

 private:
  const std::string fName;
  const double fMass;
```

```
  const int fCharge;
};

#endif  // PARTICLE_TYPE_HPP
```

Listato 2: particleType.cpp

```
#include "ParticleType.hpp"
#include <iostream>

ParticleType::ParticleType(std::string name, double mass, int charge)
    : fName{name}, fMass{mass}, fCharge{charge}
{
}

void ParticleType::Print() const
{
  std::cout << "Name:" << fName << " Mass:" << fMass << " Charge:" << fCharge
            << '\n';
}

double ParticleType::GetWidth() const { return 0.; }
```

Listato 3: resonanceType.hpp

```
#ifndef RESONANCE_TYPE_HPP
#define RESONANCE_TYPE_HPP

#include <string>
#include "ParticleType.hpp"

class ResonanceType : public ParticleType
{
 public:
  ResonanceType(std::string name = std::string{},
                double mass = 0.,
                int charge = 0,
                double width = 0.);
  double GetWidth() const override { return fWidth; }
  void Print() const override;

 private:
  const double fWidth;
};

#endif  // RESONANCE_TYPE_HPP
```

Listato 4: resonanceType.cpp

```
#include "ResonanceType.hpp"
#include <iostream>

ResonanceType::ResonanceType(std::string name,
                             double mass,
                             int charge,
                             double width)
    : ParticleType{name, mass, charge}, fWidth{width}
{
}

void ResonanceType::Print() const
{
  std::cout << "Name:" << GetName() << " Mass:" << GetMass()
            << " Charge:" << GetCharge() << " Width:" << fWidth << '\n';
}
```

Listato 5: particle.hpp

```
#ifndef PARTICLE_HPP
#define PARTICLE_HPP

#include "ResonanceType.hpp"
#include <string>
```

```

class Particle {
public:
  Particle(std::string name = {}, double Px = 0., double Py = 0.,
           double Pz = 0.);

  int GetIParticle() const { return fIParticle; }
  void SetParticleType(int IParticle) { fIParticle = IParticle; }
  void SetParticleType(std::string name);

  double GetPx() const { return fPx; }
  double GetPy() const { return fPy; }
  double GetPz() const { return fPz; }

  void SetP(double x, double y, double z) {
    fPx = x;
    fPy = y;
    fPz = z;
  }

  std::string GetName() const { return fParticleTypes[fIParticle]->GetName(); }
  double GetCharge() const { return fParticleTypes[fIParticle]->GetCharge(); }
  double GetMass() const { return fParticleTypes[fIParticle]->GetMass(); }
  double GetEnergy() const;
  double InvMass(Particle const &particle2) const;

  void Print() const;

  int Decay2body(Particle &dau1, Particle &dau2) const;

  static void AddParticleType(std::string name, double mass, int charge,
                              double width = 0.);
  static void PrintParticleTypes();

private:
  int fIParticle;

  double fPx;
  double fPy;
  double fPz;

  static const int fMaxNumParticleType = 10;
  static int fNParticleType;
  static ParticleType *fParticleTypes[];
  static int FindParticle(std::string name);

  void Boost(double bx, double by, double bz);
};

#endif // PARTICLE_HPP
```

Listato 6: particle.cpp

```
#include "Particle.hpp"
#include <cmath>   // for M_PI
#include <cstdlib> //for RAND_MAX
#include <iostream>

int Particle::fNParticleType = 0;

ParticleType *Particle::fParticleTypes[fMaxNumParticleType] = {};

Particle::Particle(std::string name, double Px, double Py, double Pz)
    : fPx{Px}, fPy{Py}, fPz{Pz} {
  fIParticle = FindParticle(name);
  if (fIParticle == -1 && name != "") {
    std::cout << "Uknown ParticleType passed to Particle constructor" << '\n';
  }
}

void Particle::SetParticleType(std::string name) {
  fIParticle = FindParticle(name);
  if (fIParticle == -1) {
    std::cout << "Uknown ParticleType passed to SetParticleType" << '\n';
  }
```

```
23 }
24
25 double Particle::GetEnergy() const {
26   ParticleType *particalType = fParticleTypes[fIParticle];
27   double mass = particalType->GetMass();
28   return std::sqrt(mass * mass + fPx * fPx + fPy * fPy + fPz * fPz);
29 }
30
31 double Particle::InvMass(Particle const &particle2) const {
32   double E1 = GetEnergy();
33   double E2 = particle2.GetEnergy();
34   double Px2 = particle2.GetPx();
35   double Py2 = particle2.GetPy();
36   double Pz2 = particle2.GetPz();
37
38   return std::sqrt(
39       // clang-format off
40       (E1 + E2) * (E1 + E2) -
41       ((fPx + Px2) * (fPx + Px2) +
42       (fPy + Py2) * (fPy + Py2) +
43       (fPz + Pz2) * (fPz + Pz2))
44       // clang-format on
45   );
46 }
47
48 void Particle::Print() const {
49   ParticleType *particalType = fParticleTypes[fIParticle];
50   std::cout << "IParticle:" << fIParticle << " Name:" << particalType->GetName()
51             << " Pulse:(" << fPx << ',' << fPy << ',' << fPz << ")\n";
52 }
53
54 void Particle::AddParticleType(std::string name, double mass, int charge,
55                                double width) {
56   if (fMaxNumParticleType != fNParticleType && FindParticle(name) == -1) {
57     if (width == 0.) {
58       fParticleTypes[fNParticleType] = new ParticleType(name, mass, charge);
59     } else {
60       fParticleTypes[fNParticleType] =
61           new ResonanceType(name, mass, charge, width);
62     }
63     ++fNParticleType;
64   }
65 }
66
67 void Particle::PrintParticleTypes() {
68   for (int i = 0; i != fNParticleType; ++i) {
69     fParticleTypes[i]->Print();
70   }
71 }
72
73 int Particle::FindParticle(std::string name) {
74   for (int i = 0; i != fNParticleType; ++i) {
75     if (name == fParticleTypes[i]->GetName()) {
76       return i;
77     }
78   }
79   return -1;
80 }
81
82 int Particle::Decay2body(Particle &dau1, Particle &dau2) const {
83   if (GetMass() == 0.0) {
84     printf("Decayment cannot be preformed if mass is zero\n");
85     return 1;
86   }
87
88   double massMot = GetMass();
89   double massDau1 = dau1.GetMass();
90   double massDau2 = dau2.GetMass();
91
92   if (fIParticle > -1) { // add width effect
93
94     // gaussian random numbers
95
96     double x1, x2, w, y;
97
```

```
    double invnum = 1. / RAND_MAX;
    do {
      x1 = 2.0 * rand() * invnum - 1.0;
      x2 = 2.0 * rand() * invnum - 1.0;
      w = x1 * x1 + x2 * x2;
    } while (w >= 1.0);

    w = sqrt((-2.0 * log(w)) / w);
    y = x1 * w;

    massMot += fParticleTypes[fIParticle]->GetWidth() * y;
  }

  if (massMot < massDau1 + massDau2) {
    printf("Decayment cannot be preformed because mass is too low in this "
           "channel\n");
    return 2;
  }

  double pout =
      sqrt(
          (massMot * massMot - (massDau1 + massDau2) * (massDau1 + massDau2)) *
          (massMot * massMot - (massDau1 - massDau2) * (massDau1 - massDau2))) /
      massMot * 0.5;

  double norm = 2 * M_PI / RAND_MAX;

  double phi = rand() * norm;
  double theta = rand() * norm * 0.5 - M_PI / 2.;
  dau1.SetP(pout * sin(theta) * cos(phi), pout * sin(theta) * sin(phi),
            pout * cos(theta));
  dau2.SetP(-pout * sin(theta) * cos(phi), -pout * sin(theta) * sin(phi),
            -pout * cos(theta));

  double energy = sqrt(fPx * fPx + fPy * fPy + fPz * fPz + massMot * massMot);

  double bx = fPx / energy;
  double by = fPy / energy;
  double bz = fPz / energy;

  dau1.Boost(bx, by, bz);
  dau2.Boost(bx, by, bz);

  return 0;
}

void Particle::Boost(double bx, double by, double bz) {
  double energy = GetEnergy();

  // Boost this Lorentz vector
  double b2 = bx * bx + by * by + bz * bz;
  double gamma = 1.0 / sqrt(1.0 - b2);
  double bp = bx * fPx + by * fPy + bz * fPz;
  double gamma2 = b2 > 0 ? (gamma - 1.0) / b2 : 0.0;

  fPx += gamma2 * bp * bx + gamma * bx * energy;
  fPy += gamma2 * bp * by + gamma * by * energy;
  fPz += gamma2 * bp * bz + gamma * bz * energy;
}
```

Listato 7: main.cpp

```
#include "Particle.hpp"
#include "TFile.h"
#include "TH1.h"
#include "TH2.h"
#include "TMath.h"
#include "TRandom.h"
#include <array>
#include <iostream>

int constexpr maxNumParticle = 100;
int constexpr maxNumDecay = 20;

void AddParticleTypes() {
```

```
  Particle::AddParticleType("pion+", 0.13957, 1);
  Particle::AddParticleType("pion-", 0.13957, -1);
  Particle::AddParticleType("kaon+", 0.49367, 1);
  Particle::AddParticleType("kaon-", 0.49367, -1);
  Particle::AddParticleType("proton+", 0.93827, 1);
  Particle::AddParticleType("proton-", 0.93827, -1);
  Particle::AddParticleType("kaon*", 0.89166, 0, 0.05);
}

TList *FillHistos(int nEvents = 1e5) {
  auto hParticleTypes = new TH1D("hParticleTypes", "Particle types", 7, 0, 7);
  auto hAzimutalAngles = new TH1F(
      "hAzimutalAngles", "Azimutal angles distribution", 10, 0, TMath::Pi());
  auto hPolarAngles = new TH1F("hPolarAngles", "Polar angles distribution", 10,
                               0, 2 * TMath::Pi());
  auto hAngles = new TH2F("hAngles", "Angles distribution", 10, 10, 0,
                          TMath::Pi(), 0, 2 * TMath::Pi());
  auto hPulse = new TH1D("hPulse", "Pulse", 20, 0, 5);
  auto hTransversePulse =
      new TH1D("hTransversePulse", "Transverse pulse", 20, 0, 5);
  auto hEnergy = new TH1D("hEnergy", "Energy", 20, 0, 5);
  auto hInvMass = new TH1D("hInvMass", "Invariant mass", 20, 0, 5);
  auto hConcordantInvMass = new TH1D(
      "hConcordantInvMass",
      "Invariant mass of particles with concordant charge sign", 200, 0, 5);
  auto hDiscordantInvMass = new TH1D(
      "hDiscordantInvMass",
      "Invariant mass of particles with discordant charge sign", 200, 0, 5);
  auto hConcordantPionKaonInvMass =
      new TH1F("hConcordantPionKaonInvMass",
               "Invariant mass of kaons and pions with concordant charge sign",
               200, 0, 5);
  auto hDiscordantPionKaonInvMass =
      new TH1F("hDiscordantPionKaonInvMass",
               "Invariant mass of kaons and pions with discordant charge sign",
               200, 0, 5);
  auto hResonanceCoupleInvMass = new TH1F(
      "hResonanceCoupleInvMass", "Invariant mass of decayed particles couples",
      60, 0.89166 - 0.25, 0.89166 + 0.25);

  for (int k = 0; k != nEvents; ++k) {
    std::array<Particle, maxNumParticle + maxNumDecay> particles;
    int nDecay = 0;

    for (int i = 0; i != maxNumParticle; ++i) {
      double phi = gRandom->Uniform(2 * TMath::Pi());
      double theta = gRandom->Uniform(TMath::Pi());
      double P = gRandom->Exp(1);

      double Px = P * TMath::Sin(theta) * TMath::Cos(phi);
      double Py = P * TMath::Sin(theta) * TMath::Sin(phi);
      double Pz = P * TMath::Cos(theta);
      double transP = TMath::Sqrt(Px * Px + Py * Py);

      particles[i].SetP(Px, Py, Pz);

      hPolarAngles->Fill(phi);
      hAzimutalAngles->Fill(theta);
      hAngles->Fill(theta, phi);
      hPulse->Fill(P);
      hTransversePulse->Fill(transP);

      double x = gRandom->Rndm();
      if (x < 0.4) {
        particles[i].SetParticleType("pion+");
        hParticleTypes->Fill(0);
      } else if (x < 0.8) {
        particles[i].SetParticleType("pion-");
        hParticleTypes->Fill(1);
      } else if (x < 0.85) {
        particles[i].SetParticleType("kaon+");
        hParticleTypes->Fill(2);
      } else if (x < 0.9) {
        particles[i].SetParticleType("kaon-");
        hParticleTypes->Fill(3);
```

```
89        } else if (x < 0.945) {
90          particles[i].SetParticleType("proton+");
91          hParticleTypes->Fill(4);
92        } else if (x < 0.99) {
93          particles[i].SetParticleType("proton-");
94          hParticleTypes->Fill(5);
95        } else {
96          particles[i].SetParticleType("kaon*");
97          hParticleTypes->Fill(6);
98
99          double y = gRandom->Rndm();
100         auto &childParticle1 = particles[maxNumParticle + nDecay];
101         auto &childParticle2 = particles[maxNumParticle + nDecay + 1];
102
103         if (y < 0.5) {
104           childParticle1.SetParticleType("pion+");
105           childParticle2.SetParticleType("kaon-");
106         } else {
107           childParticle1.SetParticleType("pion-");
108           childParticle2.SetParticleType("kaon+");
109         }
110
111         particles[i].Decay2body(childParticle1, childParticle2);
112
113         double invMass = childParticle1.InvMass(childParticle2);
114         hResonanceCoupleInvMass->Fill(invMass);
115
116         nDecay += 2;
117       }
118       hEnergy->Fill(particles[i].GetEnergy());
119     }
120
121     for (int i = 0; i != maxNumParticle + nDecay; ++i) {
122       auto &particle = particles[i];
123
124       for (int j = i + 1; j != maxNumParticle + nDecay; ++j) {
125         auto &particle2 = particles[j];
126         auto name = particle.GetName();
127         auto name2 = particle2.GetName();
128         double invMass = particle.InvMass(particle2);
129
130         hInvMass->Fill(invMass);
131
132         if (particle.GetCharge() * particle2.GetCharge() > 0) {
133           hConcordantInvMass->Fill(invMass);
134           if ((name == "pion+" && name2 == "kaon+") ||
135               (name == "pion-" && name2 == "kaon-")) {
136             hConcordantPionKaonInvMass->Fill(invMass);
137           }
138         } else if (particle.GetCharge() * particle2.GetCharge() < 0) {
139           hDiscordantInvMass->Fill(invMass);
140           if ((name == "pion+" && name2 == "kaon-") ||
141               (name == "pion-" && name2 == "kaon+")) {
142             hDiscordantPionKaonInvMass->Fill(invMass);
143           }
144         }
145       }
146     }
147   }
148
149   auto listHistos = new TList{};
150   listHistos->SetOwner();
151
152   listHistos->Add(hParticleTypes);
153   listHistos->Add(hAzimutalAngles);
154   listHistos->Add(hPolarAngles);
155   listHistos->Add(hAngles);
156   listHistos->Add(hPulse);
157   listHistos->Add(hTransversePulse);
158   listHistos->Add(hEnergy);
159   listHistos->Add(hInvMass);
160   listHistos->Add(hConcordantInvMass);
161   listHistos->Add(hDiscordantInvMass);
162   listHistos->Add(hConcordantPionKaonInvMass);
163   listHistos->Add(hDiscordantPionKaonInvMass);
```

```
  listHistos->Add(hResonanceCoupleInvMass);

  return listHistos;
}

int main() {
  gRandom->SetSeed();
  AddParticleTypes();

  auto listHistos = FillHistos(1e5);

  auto file = new TFile("Histograms.root", "RECREATE");
  for (auto const histo : *listHistos) {
    histo->Write();
  }
  file->Close();

  delete listHistos;
}
```

Listato 8: analysis.cpp

```
#include <iostream>

#include "TCanvas.h"
#include "TF1.h"
#include "TFile.h"
#include "TGaxis.h"
#include "TH1F.h"
#include "TLegend.h"
#include "TList.h"
#include "TROOT.h"
#include "TStyle.h"

#include "myStyle.hpp"

void analysis(char *filePath) {
  setMyStyle();
  TGaxis::SetMaxDigits(3);
  gROOT->SetStyle("myStyle");

  auto fileHistograms = new TFile(filePath);
  auto cGeneration = new TCanvas("cGeneration", "Particle Generation");
  auto cInvMass = new TCanvas("cInvMass", "Invariant mass");
  cInvMass->Divide(2, 2);
  cGeneration->Divide(2, 2);

  {
    cGeneration->cd(1);

    auto hParticleTypes = (TH1D *)fileHistograms->Get("hParticleTypes");

    std::cout << "PARTICLE TYPES\n";
    for (int i = 1; i != 8; ++i) {
      std::cout << "IParticle: " << i << '\t'
                << "Occurrences: " << hParticleTypes->GetBinContent(i)
                << " +/- " << hParticleTypes->GetBinError(i) << '\n';
    }
    std::cout << '\n';

    hParticleTypes->UseCurrentStyle();
    hParticleTypes->SetTitle("Generated particles");
    hParticleTypes->SetXTitle("Particle");
    hParticleTypes->SetYTitle("Occurrences");
    hParticleTypes->GetXaxis()->SetBinLabel(1, "Pion+");
    hParticleTypes->GetXaxis()->SetBinLabel(2, "Pion-");
    hParticleTypes->GetXaxis()->SetBinLabel(3, "Kaon+");
    hParticleTypes->GetXaxis()->SetBinLabel(4, "Kaon-");
    hParticleTypes->GetXaxis()->SetBinLabel(5, "Proton+");
    hParticleTypes->GetXaxis()->SetBinLabel(6, "Proton-");
    hParticleTypes->GetXaxis()->SetBinLabel(7, "Kaon*");
    hParticleTypes->SetFillColor(kBlue - 8);
    hParticleTypes->SetLabelSize(0.06, "x");
    hParticleTypes->SetStats(0);
    hParticleTypes->Draw();
```

```
54
55      auto hExpectedParticleTypes = new TH1D(*hParticleTypes);
56      int entries = hParticleTypes->GetEntries();
57
58      hExpectedParticleTypes->SetBinContent(1, 0.4 * entries);
59      hExpectedParticleTypes->SetBinContent(2, 0.4 * entries);
60      hExpectedParticleTypes->SetBinContent(3, 0.05 * entries);
61      hExpectedParticleTypes->SetBinContent(4, 0.05 * entries);
62      hExpectedParticleTypes->SetBinContent(5, 0.045 * entries);
63      hExpectedParticleTypes->SetBinContent(6, 0.045 * entries);
64      hExpectedParticleTypes->SetBinContent(7, 0.01 * entries);
65
66      hExpectedParticleTypes->SetMarkerStyle(kCircle);
67      hExpectedParticleTypes->SetMarkerSize(0.7);
68      hExpectedParticleTypes->Draw("P,same");
69
70      auto legend = new TLegend(0.7, 0.8, 0.95, 0.95);
71      legend->AddEntry(hParticleTypes, "Generated particles", "f");
72      legend->AddEntry(hExpectedParticleTypes, "Expected particles", "p");
73      legend->Draw();
74    }
75
76    {
77      cGeneration->cd(2);
78
79      auto hPulse = (TH1D *)fileHistograms->Get("hPulse");
80      hPulse->Fit("expo", "Q");
81      auto fitFunc = hPulse->GetFunction("expo");
82      double Chi = fitFunc->GetChisquare();
83      int dof = fitFunc->GetNDF();
84
85      std::cout << "PULSE FIT\n"
86                << "Tau: " << -fitFunc->GetParameter(1) << " +/- "
87                << fitFunc->GetParError(1) << '\n'
88                << "Chi^2: " << Chi << '\n'
89                << "dof: " << dof << '\n'
90                << "Chi^2/dof: " << Chi / dof << "\n\n";
91
92      hPulse->UseCurrentStyle();
93      hPulse->GetXaxis()->SetTitle("Pulse (GeV)");
94      hPulse->GetYaxis()->SetTitle("Occurences");
95      hPulse->SetFillColor(42);
96      hPulse->SetLineColor(kBlack);
97      fitFunc->SetLineColor(kBlack);
98      fitFunc->SetLineWidth(2);
99      hPulse->Draw();
100   }
101
102   {
103     cGeneration->cd(3);
104
105     auto hAzimutalAngles = (TH1F *)fileHistograms->Get("hAzimutalAngles");
106     auto hPolarAngles = (TH1F *)fileHistograms->Get("hPolarAngles");
107
108     auto listAngles = new TList();
109     listAngles->Add(hAzimutalAngles);
110     listAngles->Add(hPolarAngles);
111
112     for (int i = 0; i != 2; ++i) {
113       cGeneration->cd(i + 3);
114       auto h = (TH1F *)listAngles->At(i);
115
116       h->Fit("pol0", "Q");
117       auto fitFunc = h->GetFunction("pol0");
118       double Chi = fitFunc->GetChisquare();
119       int dof = fitFunc->GetNDF();
120
121       std::cout << h->GetTitle() << " FIT\n"
122                 << "Parameter: " << fitFunc->GetParameter(0) << " +/- "
123                 << fitFunc->GetParError(0) << '\n'
124                 << "Chi^2: " << Chi << '\n'
125                 << "dof: " << dof << '\n'
126                 << "Chi^2/dof: " << Chi / dof << "\n\n";
127
128       h->UseCurrentStyle();
```

```
129       h->SetXTitle("Angle (rad)");
130       h->SetYTitle("Occurences");
131       h->GetYaxis()->SetTitleOffset(1.26);
132       h->SetMarkerStyle(kFullCircle);
133       h->SetMarkerSize(0.6);
134       h->SetMarkerColor(kBlack);
135       h->SetLineColor(kBlack);
136       fitFunc->SetLineColor(kRed + 1);
137       h->Draw("E1P");
138     }
139   }
140
141   {
142     auto hConcordantInvMass = (TH1D *)fileHistograms->Get("hConcordantInvMass");
143     auto hDiscordantInvMass = (TH1D *)fileHistograms->Get("hDiscordantInvMass");
144     auto hConcordantPionKaonInvMass =
145         (TH1F *)fileHistograms->Get("hConcordantPionKaonInvMass");
146     auto hDiscordantPionKaonInvMass =
147         (TH1F *)fileHistograms->Get("hDiscordantPionKaonInvMass");
148     auto hResonanceCoupleInvMass =
149         (TH1F *)fileHistograms->Get("hResonanceCoupleInvMass");
150     hResonanceCoupleInvMass->Rebin(2);
151
152     hDiscordantPionKaonInvMass->Sumw2();
153     auto hDifferencePionKaonInvMass =
154         (TH1F *)hDiscordantPionKaonInvMass->Clone("hDifferencePionKaonInvMass");
155     hDifferencePionKaonInvMass->Add(hDiscordantPionKaonInvMass,
156                                     hConcordantPionKaonInvMass, 1, -1);
157     hDifferencePionKaonInvMass->GetXaxis()->SetRangeUser(0.89166 - 0.25,
158                                                          0.89166 + 0.25);
159     hDiscordantInvMass->Sumw2();
160     auto hDifferenceInvMass = (TH1D *)hDiscordantInvMass->Clone("hDiff"
161                                                                 "erenc"
162                                                                 "eInvM"
163                                                                 "ass");
164     hDifferenceInvMass->Add(hDiscordantInvMass, hConcordantInvMass, 1, -1);
165     hDifferenceInvMass->Rebin(2);
166     hDifferenceInvMass->GetXaxis()->SetRangeUser(0.89166 - 0.25,
167                                                  0.89166 + 0.25);
168
169     auto listInvMass = new TList();
170     listInvMass->Add(hDifferenceInvMass);
171     listInvMass->Add(hDifferencePionKaonInvMass);
172     listInvMass->Add(hResonanceCoupleInvMass);
173
174     int colors[3] = {45, 31, 38};
175
176     for (int i = 0; i != 3; ++i) {
177       cInvMass->cd(i + 1);
178       auto h = (TH1D *)listInvMass->At(i);
179
180       h->Fit("gaus", "Q");
181       auto fitFunc = h->GetFunction("gaus");
182       double Chi = fitFunc->GetChisquare();
183       int dof = fitFunc->GetNDF();
184
185       std::cout << "GAUS FIT " << i << '\n'
186                 << "Amplitude : " << fitFunc->GetParameter(0) << " +/- "
187                 << fitFunc->GetParError(0) << '\n'
188                 << "Mean : " << fitFunc->GetParameter(1) << " +/- "
189                 << fitFunc->GetParError(1) << '\n'
190                 << "Sigma : " << fitFunc->GetParameter(2) << " +/- "
191                 << fitFunc->GetParError(2) << '\n'
192                 << "Chi^2: " << Chi << '\n'
193                 << "dof: " << dof << '\n'
194                 << "Chi^2/dof: " << Chi / dof << "\n\n";
195
196       h->UseCurrentStyle();
197       h->SetXTitle("Invariant mass "
198                    "(GeV/c^2)");
199       h->SetYTitle("Occurences");
200       h->SetFillColorAlpha(colors[i], 0.5);
201       h->SetLineColor(kBlack);
202       fitFunc->SetLineColor(kBlack);
203       fitFunc->SetLineWidth(2);
```

```
      h->Draw("E1, X0");
      h->Draw("HIST, same");
      h->Draw("E1, X0, same");
    }
  }

  cGeneration->SaveAs("Generation.pdf");
  cGeneration->SaveAs("Generation.C");
  cGeneration->SaveAs("Generation.root");

  cInvMass->SaveAs("InvMass.pdf");
  cInvMass->SaveAs("InvMass.C");
  cInvMass->SaveAs("InvMass.root");
}

int main(int argc, char *argv[]) { analysis(argv[1]); }
```

Listato 9: myStyle.hpp

```
// Mainframe macro generated from application: /home/egg/ROOT-6.22/bin/root.exe
// By ROOT version 6.22/02 on 2020-11-07 11:33:37

#if !defined(__CINT__) || defined(__MAKECINT__)

#ifndef ROOT_TStyle
#include "TStyle.h"
#endif

#endif

void setMyStyle() {
  // Add the saved style to the current ROOT session.

  delete gROOT->GetStyle("myStyle");

  TStyle *tmpStyle = new TStyle("myStyle", "My Style");
  tmpStyle->SetNdivisions(510, "x");
  tmpStyle->SetNdivisions(510, "y");
  tmpStyle->SetNdivisions(510, "z");
  tmpStyle->SetAxisColor(1, "x");
  tmpStyle->SetAxisColor(1, "y");
  tmpStyle->SetAxisColor(1, "z");
  tmpStyle->SetLabelColor(1, "x");
  tmpStyle->SetLabelColor(1, "y");
  tmpStyle->SetLabelColor(1, "z");
  tmpStyle->SetLabelFont(42, "x");
  tmpStyle->SetLabelFont(42, "y");
  tmpStyle->SetLabelFont(42, "z");
  tmpStyle->SetLabelOffset(0.005, "x");
  tmpStyle->SetLabelOffset(0.005, "y");
  tmpStyle->SetLabelOffset(0.005, "z");
  tmpStyle->SetLabelSize(0.045, "x");
  tmpStyle->SetLabelSize(0.045, "y");
  tmpStyle->SetLabelSize(0.045, "z");
  tmpStyle->SetTickLength(0.03, "x");
  tmpStyle->SetTickLength(0.03, "y");
  tmpStyle->SetTickLength(0.03, "z");
  tmpStyle->SetTitleOffset(1, "x");
  tmpStyle->SetTitleOffset(1.2, "y");
  tmpStyle->SetTitleOffset(1, "z");
  tmpStyle->SetTitleSize(0.045, "x");
  tmpStyle->SetTitleSize(0.045, "y");
  tmpStyle->SetTitleSize(0.045, "z");
  tmpStyle->SetTitleColor(1, "x");
  tmpStyle->SetTitleColor(1, "y");
  tmpStyle->SetTitleColor(1, "z");
  tmpStyle->SetTitleFont(42, "x");
  tmpStyle->SetTitleFont(42, "y");
  tmpStyle->SetTitleFont(42, "z");
  tmpStyle->SetBarWidth(1);
  tmpStyle->SetBarOffset(0);
  tmpStyle->SetDrawBorder(0);
  tmpStyle->SetOptLogx(0);
  tmpStyle->SetOptLogy(0);
  tmpStyle->SetOptLogz(0);
```

```
  tmpStyle->SetOptDate(0);
  tmpStyle->SetOptStat(0);
  tmpStyle->SetOptTitle(kFALSE);
  tmpStyle->SetOptFit(111);
  tmpStyle->SetNumberContours(20);
  tmpStyle->GetAttDate()->SetTextFont(62);
  tmpStyle->GetAttDate()->SetTextSize(0.025);
  tmpStyle->GetAttDate()->SetTextAngle(0);
  tmpStyle->GetAttDate()->SetTextAlign(11);
  tmpStyle->GetAttDate()->SetTextColor(1);
  tmpStyle->SetDateX(0.01);
  tmpStyle->SetDateY(0.01);
  tmpStyle->SetEndErrorSize(2);
  tmpStyle->SetErrorX(0.5);
  tmpStyle->SetFuncColor(2);
  tmpStyle->SetFuncStyle(1);
  tmpStyle->SetFuncWidth(2);
  tmpStyle->SetGridColor(0);
  tmpStyle->SetGridStyle(3);
  tmpStyle->SetGridWidth(1);
  tmpStyle->SetLegendBorderSize(1);
  tmpStyle->SetLegendFillColor(0);
  tmpStyle->SetLegendFont(42);
  tmpStyle->SetLegendTextSize(0);
  tmpStyle->SetHatchesLineWidth(1);
  tmpStyle->SetHatchesSpacing(1);
  tmpStyle->SetFrameFillColor(0);
  tmpStyle->SetFrameLineColor(1);
  tmpStyle->SetFrameFillStyle(1001);
  tmpStyle->SetFrameLineStyle(1);
  tmpStyle->SetFrameLineWidth(1);
  tmpStyle->SetFrameBorderSize(1);
  tmpStyle->SetFrameBorderMode(0);
  tmpStyle->SetHistFillColor(0);
  tmpStyle->SetHistLineColor(602);
  tmpStyle->SetHistFillStyle(1001);
  tmpStyle->SetHistLineStyle(1);
  tmpStyle->SetHistLineWidth(1);
  tmpStyle->SetHistMinimumZero(kFALSE);
  tmpStyle->SetCanvasPreferGL(kFALSE);
  tmpStyle->SetCanvasColor(0);
  tmpStyle->SetCanvasBorderSize(2);
  tmpStyle->SetCanvasBorderMode(0);
  tmpStyle->SetCanvasDefH(500);
  tmpStyle->SetCanvasDefW(700);
  tmpStyle->SetCanvasDefX(10);
  tmpStyle->SetCanvasDefY(10);
  tmpStyle->SetPadColor(0);
  tmpStyle->SetPadBorderSize(2);
  tmpStyle->SetPadBorderMode(0);
  tmpStyle->SetPadBottomMargin(0.1);
  tmpStyle->SetPadTopMargin(0.1);
  tmpStyle->SetPadLeftMargin(0.1);
  tmpStyle->SetPadRightMargin(0.1);
  tmpStyle->SetPadGridX(kTRUE);
  tmpStyle->SetPadGridY(kTRUE);
  tmpStyle->SetPadTickX(0);
  tmpStyle->SetPadTickY(0);
  tmpStyle->SetPaperSize(20, 26);
  tmpStyle->SetScreenFactor(1);
  tmpStyle->SetStatColor(0);
  tmpStyle->SetStatTextColor(1);
  tmpStyle->SetStatBorderSize(2);
  tmpStyle->SetStatFont(42);
  tmpStyle->SetStatFontSize(0.040);
  tmpStyle->SetStatStyle(1001);
  tmpStyle->SetStatFormat("g");
  tmpStyle->SetStatX(1.);
  tmpStyle->SetStatY(0.95);
  tmpStyle->SetStatW(0.22);
  tmpStyle->SetStatH(0.19);
  tmpStyle->SetStripDecimals(kFALSE);
  tmpStyle->SetTitleAlign(23);
  tmpStyle->SetTitleFillColor(0);
  tmpStyle->SetTitleTextColor(1);
```

```
  tmpStyle->SetTitleBorderSize(0);
  tmpStyle->SetTitleFont(42);
  tmpStyle->SetTitleFontSize(0.05);
  tmpStyle->SetTitleStyle(0);
  tmpStyle->SetTitleX(0.5);
  tmpStyle->SetTitleY(0.995);
  tmpStyle->SetTitleW(0);
  tmpStyle->SetTitleH(0);
  tmpStyle->SetLegoInnerR(0.5);

  Int_t fPaletteColor[255] = {
      924,  925,  926,  927,  928,  929,  930,  931,  932,  933,  934,  935,
      936,  937,  938,  939,  940,  941,  942,  943,  944,  945,  946,  947,
      948,  949,  950,  951,  952,  953,  954,  955,  956,  957,  958,  959,
      960,  961,  962,  963,  964,  965,  966,  967,  968,  969,  970,  971,
      972,  973,  974,  975,  976,  977,  978,  979,  980,  981,  982,  983,
      984,  985,  986,  987,  988,  989,  990,  991,  992,  993,  994,  995,
      996,  997,  998,  999,  1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007,
      1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019,
      1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031,
      1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043,
      1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055,
      1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067,
      1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079,
      1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091,
      1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103,
      1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115,
      1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127,
      1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139,
      1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151,
      1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163,
      1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175,
      1176, 1177, 1178};
  tmpStyle->SetPalette(255, fPaletteColor);

  TString fLineStyleArrayTmp[30] = {"",
                                    "  ",
                                    " 12 12",
                                    " 4 8",
                                    " 12 16 4 16",
                                    " 20 12 4 12",
                                    " 20 12 4 12 4 12 4 12",
                                    " 20 20",
                                    " 20 12 4 12 4 12",
                                    " 80 20",
                                    " 80 40 4 40",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  ",
                                    "  "};
  for (Int_t i = 0; i < 30; i++)
    tmpStyle->SetLineStyleString(i, fLineStyleArrayTmp[i]);

  tmpStyle->SetHeaderPS("");
  tmpStyle->SetTitlePS("");
  tmpStyle->SetFitFormat("g");
  tmpStyle->SetPaintTextFormat("g");
  tmpStyle->SetLineScalePS(3);
  tmpStyle->SetJoinLinePS(0);
  // tmpStyle->SetCapLinePS(0);
```

```
207   tmpStyle->SetColorModelPS(0);
208   tmpStyle->SetTimeOffset(788918400);
209
210   tmpStyle->SetLineColor(1);
211   tmpStyle->SetLineStyle(1);
212   tmpStyle->SetLineWidth(1);
213   tmpStyle->SetFillColor(19);
214   tmpStyle->SetFillStyle(1001);
215   tmpStyle->SetMarkerColor(1);
216   tmpStyle->SetMarkerSize(1);
217   tmpStyle->SetMarkerStyle(1);
218   tmpStyle->SetTextAlign(11);
219   tmpStyle->SetTextAngle(0);
220   tmpStyle->SetTextColor(1);
221   tmpStyle->SetTextFont(42);
222   tmpStyle->SetTextSize(0.05);
223 }
```