

# Laboratorio di elettromagnetismo e ottica

## Relazione ROOT

Nicolò Montalti

9 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, tutti 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 nella prima fase. Si controllano l'abbondanza dei vari tipi di particella, 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 viene fatto 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 loro 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, spreco di 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  $\pi^+$  e un  $K^-$  o in  $\pi^-$  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	$\pi^+$	40%
	-1	$\pi^-$	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 ( $10^5$ )
$\pi^+$	$40.01 \pm 0.02$	40
$\pi^-$	$40.00 \pm 0.02$	40
$K^+$	$4.995 \pm 0.007$	5.0
$K^-$	$4.996 \pm 0.007$	5.0
$p^+$	$4.493 \pm 0.007$	4.5
$p^-$	$4.495 \pm 0.007$	4.5
$K^*$	$1.007 \pm 0.003$	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  $\chi^2$  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 risultante 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  $\pi K$  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  $\chi^2$  accettabili e sono compatibili con i valori attesi di media ( $0.89166 \text{ GeV}/c^2$ ) e deviazione standard ( $0.050 \text{ GeV}/c^2$ ). 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	$\chi^2$	DOF	$\chi^2/\text{DOF}$
Modulo	expo	$\tau = (0.9999 \pm 0.0003) \text{ GeV}$	23.79	18	1.32
Angolo azimutale	pol0	$p0 = (10000 \pm 3) \cdot 10^2 \text{ rad}^{-1}$	9.50	9	1.06
Angolo polare	pol0	$p0 = (10000 \pm 3) \cdot 10^2 \text{ rad}^{-1}$	14.03	9	1.56

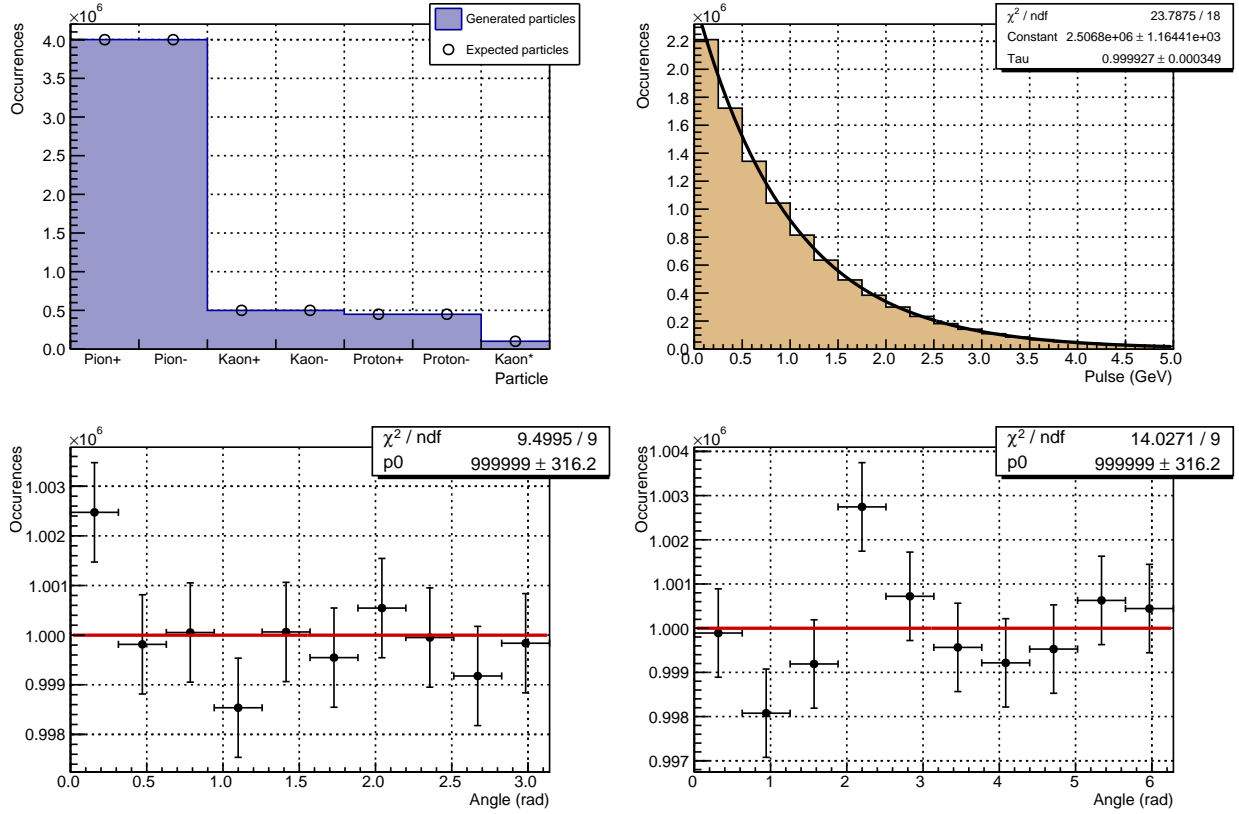


Figura 1: Istogrammi delle particelle generate e attese divise per specie (in alto a sx), del modulo dell'impulso con fit esponenziale (in alto a dx) e degli angoli azimutali e polari con fit pol0 (rispettivamente in basso a sx e a dx)

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

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

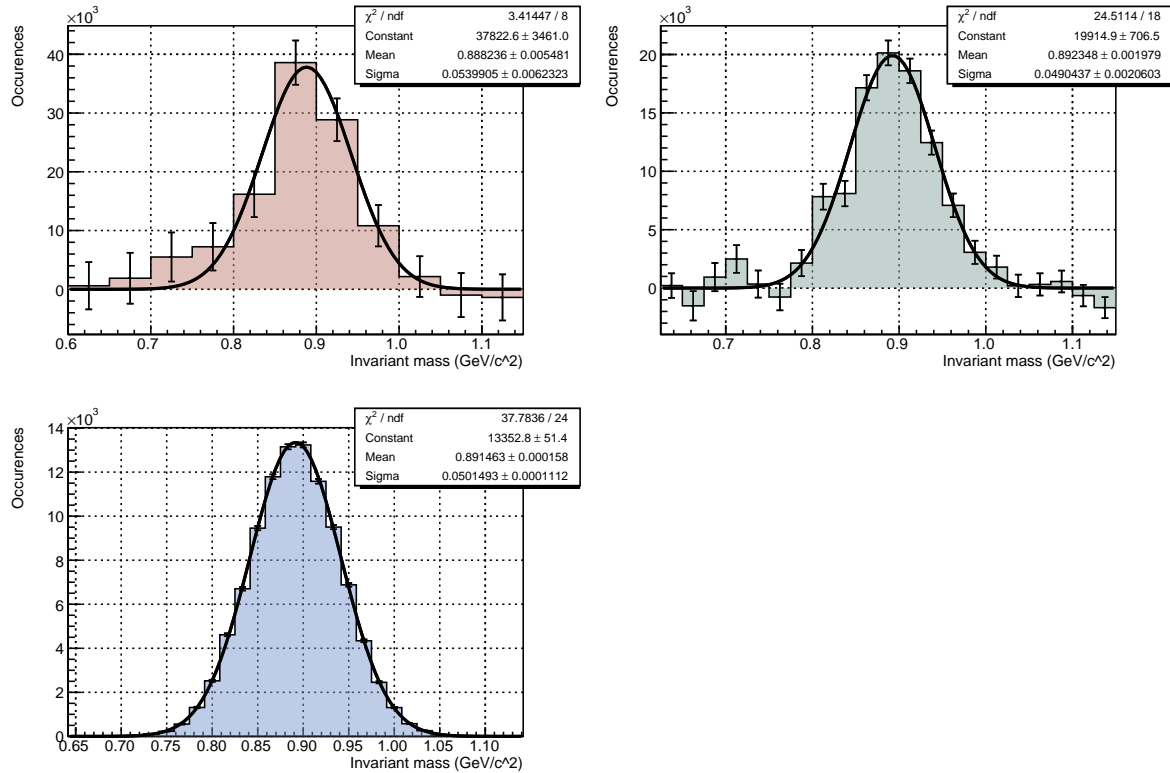


Figura 2: Istogrammi della massa invariante ottenuta dalla differenza di combinazioni di particelle di carica uguale ed opposta (in alto a sx), dalla differenza delle combinazioni  $\pi K$  di carica uguale ed opposta (in alto a dx) e dalle coppie  $\pi K$  generate dai decadimenti delle  $K^*$  (in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

## Indice dei listati

1	particleType.hpp . . . . .	4
2	particleType.cpp . . . . .	5
3	resonanceType.hpp . . . . .	5
4	resonanceType.cpp . . . . .	5
5	particle.hpp . . . . .	5
6	particle.cpp . . . . .	6
7	main.cpp . . . . .	8
8	analysis.cpp . . . . .	11
9	myStyle.hpp . . . . .	14

### Listato 1: particleType.hpp

```

1 #ifndef PARTICLE_TYPE_HPP
2 #define PARTICLE_TYPE_HPP
3
4 #include <string>
5
6 class ParticleType
7 {
8 public:
9     ParticleType(std::string name = std::string{},
10                 double mass = 0.,
11                 int charge = 0);
12     std::string GetName() const { return fName; }
13     double GetMass() const { return fMass; }
14     double GetCharge() const { return fCharge; }
15     virtual void Print() const;
16     virtual double GetWidth() const;
17
18 private:
19     const std::string fName;
20     const double fMass;

```

```

21     const int fCharge;
22 };
23
24 #endif // PARTICLE_TYPE_HPP

```

#### Listato 2: particleType.cpp

```

1 #include "ParticleType.hpp"
2 #include <iostream>
3
4 ParticleType::ParticleType(std::string name, double mass, int charge)
5     : fName{name}, fMass{mass}, fCharge{charge}
6 {
7 }
8
9 void ParticleType::Print() const
10 {
11     std::cout << "Name:" << fName << " Mass:" << fMass << " Charge:" << fCharge
12         << '\n';
13 }
14
15 double ParticleType::GetWidth() const { return 0.; }

```

#### Listato 3: resonanceType.hpp

```

1 #ifndef RESONANCE_TYPE_HPP
2 #define RESONANCE_TYPE_HPP
3
4 #include <string>
5 #include "ParticleType.hpp"
6
7 class ResonanceType : public ParticleType
8 {
9 public:
10     ResonanceType(std::string name = std::string{},
11         double mass = 0.,
12         int charge = 0,
13         double width = 0.);
14     double GetWidth() const override { return fWidth; }
15     void Print() const override;
16
17 private:
18     const double fWidth;
19 };
20
21 #endif // RESONANCE_TYPE_HPP

```

#### Listato 4: resonanceType.cpp

```

1 #include "ResonanceType.hpp"
2 #include <iostream>
3
4 ResonanceType::ResonanceType(std::string name,
5     double mass,
6     int charge,
7     double width)
8     : ParticleType{name, mass, charge}, fWidth{width}
9 {
10 }
11
12 void ResonanceType::Print() const
13 {
14     std::cout << "Name:" << GetName() << " Mass:" << GetMass()
15         << " Charge:" << GetCharge() << " Width:" << fWidth << '\n';
16 }

```

#### Listato 5: particle.hpp

```

1 #ifndef PARTICLE_HPP
2 #define PARTICLE_HPP
3
4 #include "ResonanceType.hpp"
5 #include <string>

```

```

6
7 class Particle {
8 public:
9     Particle(std::string name = {}, double Px = 0., double Py = 0.,
10             double Pz = 0.);
11
12     int GetIParticle() const { return fIParticle; }
13     void SetParticleType(int IParticle) { fIParticle = IParticle; }
14     void SetParticleType(std::string name);
15
16     double GetPx() const { return fPx; }
17     double GetPy() const { return fPy; }
18     double GetPz() const { return fPz; }
19
20     void SetP(double x, double y, double z) {
21         fPx = x;
22         fPy = y;
23         fPz = z;
24     }
25
26     std::string GetName() const { return fParticleTypes[fIParticle]->GetName(); }
27     double GetCharge() const { return fParticleTypes[fIParticle]->GetCharge(); }
28     double GetMass() const { return fParticleTypes[fIParticle]->GetMass(); }
29     double GetEnergy() const;
30     double InvMass(Particle const &particle2) const;
31
32     void Print() const;
33
34     int Decay2body(Particle &dau1, Particle &dau2) const;
35
36     static void AddParticleType(std::string name, double mass, int charge,
37                                double width = 0.);
38     static void PrintParticleTypes();
39
40 private:
41     int fIParticle;
42
43     double fPx;
44     double fPy;
45     double fPz;
46
47     static const int fMaxNumParticleType = 10;
48     static int fNParticleType;
49     static ParticleType *fParticleTypes[];
50     static int FindParticle(std::string name);
51
52     void Boost(double bx, double by, double bz);
53 };
54
55 #endif // PARTICLE_HPP

```

#### Listato 6: particle.cpp

```

1 #include "Particle.hpp"
2 #include <cmath> // for M_PI
3 #include <cstdlib> //for RAND_MAX
4 #include <iostream>
5
6 int Particle::fNParticleType = 0;
7
8 ParticleType *Particle::fParticleTypes[fMaxNumParticleType] = {};
9
10 Particle::Particle(std::string name, double Px, double Py, double Pz)
11     : fPx{Px}, fPy{Py}, fPz{Pz} {
12     fIParticle = FindParticle(name);
13     if (fIParticle == -1 && name != "") {
14         std::cout << "Uknown ParticleType passed to Particle constructor" << '\n';
15     }
16 }
17
18 void Particle::SetParticleType(std::string name) {
19     fIParticle = FindParticle(name);
20     if (fIParticle == -1) {
21         std::cout << "Uknown ParticleType passed to SetParticleType" << '\n';
22     }
23 }

```

```

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

```

```

98     double invnum = 1. / RAND_MAX;
99     do {
100         x1 = 2.0 * rand() * invnum - 1.0;
101         x2 = 2.0 * rand() * invnum - 1.0;
102         w = x1 * x1 + x2 * x2;
103     } while (w >= 1.0);
104
105     w = sqrt((-2.0 * log(w)) / w);
106     y = x1 * w;
107
108     massMot += fParticleTypes[fIParticle]->GetWidth() * y;
109 }
110
111 if (massMot < massDau1 + massDau2) {
112     printf("Decayment cannot be preformed because mass is too low in this "
113           "channel\n");
114     return 2;
115 }
116
117 double pout =
118     sqrt(
119         (massMot * massMot - (massDau1 + massDau2) * (massDau1 + massDau2)) *
120         (massMot * massMot - (massDau1 - massDau2) * (massDau1 - massDau2))) /
121     massMot * 0.5;
122
123 double norm = 2 * M_PI / RAND_MAX;
124
125 double phi = rand() * norm;
126 double theta = rand() * norm * 0.5 - M_PI / 2.;
127 dau1.SetP(pout * sin(theta) * cos(phi), pout * sin(theta) * sin(phi),
128           pout * cos(theta));
129 dau2.SetP(-pout * sin(theta) * cos(phi), -pout * sin(theta) * sin(phi),
130           -pout * cos(theta));
131
132 double energy = sqrt(fPx * fPx + fPy * fPy + fPz * fPz + massMot * massMot);
133
134 double bx = fPx / energy;
135 double by = fPy / energy;
136 double bz = fPz / energy;
137
138 dau1.Boost(bx, by, bz);
139 dau2.Boost(bx, by, bz);
140
141 return 0;
142 }
143
144 void Particle::Boost(double bx, double by, double bz) {
145     double energy = GetEnergy();
146
147     // Boost this Lorentz vector
148     double b2 = bx * bx + by * by + bz * bz;
149     double gamma = 1.0 / sqrt(1.0 - b2);
150     double bp = bx * fPx + by * fPy + bz * fPz;
151     double gamma2 = b2 > 0 ? (gamma - 1.0) / b2 : 0.0;
152
153     fPx += gamma2 * bp * bx + gamma * bx * energy;
154     fPy += gamma2 * bp * by + gamma * by * energy;
155     fPz += gamma2 * bp * bz + gamma * bz * energy;
156 }

```

#### Listato 7: main.cpp

```

1 #include "Particle.hpp"
2 #include "TFile.h"
3 #include "TH1.h"
4 #include "TH2.h"
5 #include "TMath.h"
6 #include "TRandom.h"
7 #include <array>
8 #include <iostream>
9
10 int constexpr maxNumParticle = 100;
11 int constexpr maxNumDecay = 20;
12
13 void AddParticleTypes() {

```



```

14 Particle::AddParticleType("pion+", 0.13957, 1);
15 Particle::AddParticleType("pion-", 0.13957, -1);
16 Particle::AddParticleType("kaon+", 0.49367, 1);
17 Particle::AddParticleType("kaon-", 0.49367, -1);
18 Particle::AddParticleType("proton+", 0.93827, 1);
19 Particle::AddParticleType("proton-", 0.93827, -1);
20 Particle::AddParticleType("kaon*", 0.89166, 0, 0.05);
21 }
22
23 TList *FillHistos(int nEvents = 1e5) {
24     auto hParticleTypes = new TH1D("hParticleTypes", "Particle types", 7, 0, 7);
25     auto hAzimutalAngles = new TH1F(
26         "hAzimutalAngles", "Azimutal angles distribution", 10, 0, TMath::Pi());
27     auto hPolarAngles = new TH1F("hPolarAngles", "Polar angles distribution", 10,
28         0, 2 * TMath::Pi());
29     auto hAngles = new TH2F("hAngles", "Angles distribution", 10, 10, 0,
30         TMath::Pi(), 0, 2 * TMath::Pi());
31     auto hPulse = new TH1D("hPulse", "Pulse", 20, 0, 5);
32     auto hTransversePulse =
33         new TH1D("hTransversePulse", "Transverse pulse", 20, 0, 5);
34     auto hEnergy = new TH1D("hEnergy", "Energy", 20, 0, 5);
35     auto hInvMass = new TH1D("hInvMass", "Invariant mass", 20, 0, 5);
36     auto hConcordantInvMass = new TH1D(
37         "hConcordantInvMass",
38         "Invariant mass of particles with concordant charge sign", 200, 0, 5);
39     auto hDiscordantInvMass = new TH1D(
40         "hDiscordantInvMass",
41         "Invariant mass of particles with discordant charge sign", 200, 0, 5);
42     auto hConcordantPionKaonInvMass =
43         new TH1F("hConcordantPionKaonInvMass",
44             "Invariant mass of kaons and pions with concordant charge sign",
45             200, 0, 5);
46     auto hDiscordantPionKaonInvMass =
47         new TH1F("hDiscordantPionKaonInvMass",
48             "Invariant mass of kaons and pions with discordant charge sign",
49             200, 0, 5);
50     auto hResonanceCoupleInvMass = new TH1F(
51         "hResonanceCoupleInvMass", "Invariant mass of decayed particles couples",
52         60, 0.89166 - 0.25, 0.89166 + 0.25);
53
54     for (int k = 0; k != nEvents; ++k) {
55         std::array<Particle, maxNumParticle + maxNumDecay> particles;
56         int nDecay = 0;
57
58         for (int i = 0; i != maxNumParticle; ++i) {
59             double phi = gRandom->Uniform(2 * TMath::Pi());
60             double theta = gRandom->Uniform(TMath::Pi());
61             double P = gRandom->Exp(1);
62
63             double Px = P * TMath::Sin(theta) * TMath::Cos(phi);
64             double Py = P * TMath::Sin(theta) * TMath::Sin(phi);
65             double Pz = P * TMath::Cos(theta);
66             double transP = TMath::Sqrt(Px * Px + Py * Py);
67
68             particles[i].SetP(Px, Py, Pz);
69
70             hPolarAngles->Fill(phi);
71             hAzimutalAngles->Fill(theta);
72             hAngles->Fill(theta, phi);
73             hPulse->Fill(P);
74             hTransversePulse->Fill(transP);
75
76             double x = gRandom->Rndm();
77             if (x < 0.4) {
78                 particles[i].SetParticleType("pion+");
79                 hParticleTypes->Fill(0);
80             } else if (x < 0.8) {
81                 particles[i].SetParticleType("pion-");
82                 hParticleTypes->Fill(1);
83             } else if (x < 0.85) {
84                 particles[i].SetParticleType("kaon+");
85                 hParticleTypes->Fill(2);
86             } else if (x < 0.9) {
87                 particles[i].SetParticleType("kaon-");
88                 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);

```

```

164 listHistos->Add(hResonanceCoupleInvMass);
165
166 return listHistos;
167 }
168
169 int main() {
170     gRandom->SetSeed();
171     AddParticleTypes();
172
173     auto listHistos = FillHistos(1e5);
174
175     auto file = new TFile("Histograms.root", "RECREATE");
176     for (auto const histo : *listHistos) {
177         histo->Write();
178     }
179     file->Close();
180
181     delete listHistos;
182 }

```

## Listato 8: analysis.cpp

```

1 #include <iostream>
2
3 #include "TCanvas.h"
4 #include "TF1.h"
5 #include "TFile.h"
6 #include "TGaxis.h"
7 #include "TH1F.h"
8 #include "TLegend.h"
9 #include "TList.h"
10 #include "TROOT.h"
11 #include "TStyle.h"
12
13 #include "myStyle.hpp"
14
15 void analysis(char *filePath) {
16     setMyStyle();
17     TGaxis::SetMaxDigits(3);
18     gROOT->SetStyle("myStyle");
19
20     auto fileHistograms = new TFile(filePath);
21     auto cGeneration = new TCanvas("cGeneration", "Particle Generation");
22     auto cInvMass = new TCanvas("cInvMass", "Invariant mass");
23     cInvMass->Divide(2, 2);
24     cGeneration->Divide(2, 2);
25
26     {
27         cGeneration->cd(1);
28
29         auto hParticleTypes = (TH1D *)fileHistograms->Get("hParticleTypes");
30
31         std::cout << "PARTICLE TYPES\n";
32         for (int i = 1; i != 8; ++i) {
33             std::cout << "IParticle: " << i << '\t'
34                 << "Occurrences: " << hParticleTypes->GetBinContent(i)
35                 << " +/- " << hParticleTypes->GetBinError(i) << '\n';
36         }
37         std::cout << '\n';
38
39         hParticleTypes->UseCurrentStyle();
40         hParticleTypes->SetTitle("Generated particles");
41         hParticleTypes->SetXTitle("Particle");
42         hParticleTypes->SetYTitle("Occurrences");
43         hParticleTypes->GetXaxis()->SetBinLabel(1, "Pion+");
44         hParticleTypes->GetXaxis()->SetBinLabel(2, "Pion-");
45         hParticleTypes->GetXaxis()->SetBinLabel(3, "Kaon+");
46         hParticleTypes->GetXaxis()->SetBinLabel(4, "Kaon-");
47         hParticleTypes->GetXaxis()->SetBinLabel(5, "Proton+");
48         hParticleTypes->GetXaxis()->SetBinLabel(6, "Proton-");
49         hParticleTypes->GetXaxis()->SetBinLabel(7, "Kaon*");
50         hParticleTypes->SetFillColor(kBlue - 8);
51         hParticleTypes->SetLabelSize(0.06, "x");
52         hParticleTypes->SetStats(0);
53         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 fitFunc =
80     new TF1("fitFuncExpo", "[0] * TMath::Exp(-x / [1])", 0., 10.);
81 fitFunc->SetParameters(1e6, 1);
82 fitFunc->SetParNames("Constant", "Tau");
83 fitFunc->SetLineColor(kBlack);
84 fitFunc->SetLineWidth(2);
85
86 auto hPulse = (TH1D *)fileHistograms->Get("hPulse");
87 hPulse->Fit("fitFuncExpo", "Q");
88 double Chi = fitFunc->GetChisquare();
89 int dof = fitFunc->GetNDF();
90
91 std::cout << "PULSE FIT\n"
92     << "Tau: " << fitFunc->GetParameter(1) << " +/- "
93     << fitFunc->GetParError(1) << '\n'
94     << "Chi^2: " << Chi << '\n'
95     << "dof: " << dof << '\n'
96     << "Chi^2/dof: " << Chi / dof << "\n\n";
97
98 hPulse->UseCurrentStyle();
99 hPulse->GetXaxis()->SetTitle("Pulse (GeV)");
100 hPulse->GetYaxis()->SetTitle("Occurrences");
101 hPulse->SetFillColor(42);
102 hPulse->SetLineColor(kBlack);
103 hPulse->Draw();
104 }
105
106 {
107 cGeneration->cd(3);
108
109 auto hAzimutalAngles = (TH1F *)fileHistograms->Get("hAzimutalAngles");
110 auto hPolarAngles = (TH1F *)fileHistograms->Get("hPolarAngles");
111
112 auto listAngles = new TList();
113 listAngles->Add(hAzimutalAngles);
114 listAngles->Add(hPolarAngles);
115
116 for (int i = 0; i != 2; ++i) {
117     cGeneration->cd(i + 3);
118     auto h = (TH1F *)listAngles->At(i);
119
120     h->Fit("pol0", "Q");
121     auto fitFunc = h->GetFunction("pol0");
122     double Chi = fitFunc->GetChisquare();
123     int dof = fitFunc->GetNDF();
124
125     std::cout << h->GetTitle() << " FIT\n"
126         << "Parameter: " << fitFunc->GetParameter(0) << " +/- "
127         << fitFunc->GetParError(0) << '\n'
128         << "Chi^2: " << Chi << '\n'

```

```

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

```

```

204     h->SetFillColorAlpha(colors[i], 0.5);
205     h->SetLineColor(kBlack);
206     fitFunc->SetLineColor(kBlack);
207     fitFunc->SetLineWidth(2);
208     h->Draw("E1, XO");
209     h->Draw("HIST, same");
210     h->Draw("E1, XO, same");
211 }
212 }
213
214 cGeneration->SaveAs("Generation.pdf");
215 cGeneration->SaveAs("Generation.C");
216 cGeneration->SaveAs("Generation.root");
217
218 cInvMass->SaveAs("InvMass.pdf");
219 cInvMass->SaveAs("InvMass.C");
220 cInvMass->SaveAs("InvMass.root");
221 }
222
223 int main(int argc, char *argv[]) { analysis(argv[1]); }

```

#### Listato 9: myStyle.hpp

```

1 // Mainframe macro generated from application: /home/egg/ROOT-6.22/bin/root.exe
2 // By ROOT version 6.22/02 on 2020-11-07 11:33:37
3
4 #if !defined(__CINT__) || defined(__MAKECINT__)
5
6 #ifndef ROOT_TStyle
7 #include "TStyle.h"
8 #endif
9
10 #endif
11
12 void setMyStyle() {
13     // Add the saved style to the current ROOT session.
14
15     delete gROOT->GetStyle("myStyle");
16
17     TStyle *tmpStyle = new TStyle("myStyle", "My Style");
18     tmpStyle->SetNdivisions(510, "x");
19     tmpStyle->SetNdivisions(510, "y");
20     tmpStyle->SetNdivisions(510, "z");
21     tmpStyle->SetAxisColor(1, "x");
22     tmpStyle->SetAxisColor(1, "y");
23     tmpStyle->SetAxisColor(1, "z");
24     tmpStyle->SetLabelColor(1, "x");
25     tmpStyle->SetLabelColor(1, "y");
26     tmpStyle->SetLabelColor(1, "z");
27     tmpStyle->SetLabelFont(42, "x");
28     tmpStyle->SetLabelFont(42, "y");
29     tmpStyle->SetLabelFont(42, "z");
30     tmpStyle->SetLabelOffset(0.005, "x");
31     tmpStyle->SetLabelOffset(0.005, "y");
32     tmpStyle->SetLabelOffset(0.005, "z");
33     tmpStyle->SetLabelSize(0.045, "x");
34     tmpStyle->SetLabelSize(0.045, "y");
35     tmpStyle->SetLabelSize(0.045, "z");
36     tmpStyle->SetTickLength(0.03, "x");
37     tmpStyle->SetTickLength(0.03, "y");
38     tmpStyle->SetTickLength(0.03, "z");
39     tmpStyle->SetTitleOffset(1, "x");
40     tmpStyle->SetTitleOffset(1.2, "y");
41     tmpStyle->SetTitleOffset(1, "z");
42     tmpStyle->SetTitleSize(0.045, "x");
43     tmpStyle->SetTitleSize(0.045, "y");
44     tmpStyle->SetTitleSize(0.045, "z");
45     tmpStyle->SetTitleColor(1, "x");
46     tmpStyle->SetTitleColor(1, "y");
47     tmpStyle->SetTitleColor(1, "z");
48     tmpStyle->SetTitleFont(42, "x");
49     tmpStyle->SetTitleFont(42, "y");
50     tmpStyle->SetTitleFont(42, "z");
51     tmpStyle->SetBarWidth(1);
52     tmpStyle->SetBarOffset(0);

```

```

53 tmpStyle->SetDrawBorder(0);
54 tmpStyle->SetOptLogx(0);
55 tmpStyle->SetOptLogy(0);
56 tmpStyle->SetOptLogz(0);
57 tmpStyle->SetOptDate(0);
58 tmpStyle->SetOptStat(0);
59 tmpStyle->SetOptTitle(kFALSE);
60 tmpStyle->SetOptFit(111);
61 tmpStyle->SetNumberContours(20);
62 tmpStyle->GetAttDate()->SetTextFont(62);
63 tmpStyle->GetAttDate()->SetTextSize(0.025);
64 tmpStyle->GetAttDate()->SetTextAngle(0);
65 tmpStyle->GetAttDate()->SetTextAlign(11);
66 tmpStyle->GetAttDate()->SetTextColor(1);
67 tmpStyle->SetDateX(0.01);
68 tmpStyle->SetDateY(0.01);
69 tmpStyle->SetEndErrorSize(2);
70 tmpStyle->SetErrorX(0.5);
71 tmpStyle->SetFuncColor(2);
72 tmpStyle->SetFuncStyle(1);
73 tmpStyle->SetFuncWidth(2);
74 tmpStyle->SetGridColor(0);
75 tmpStyle->SetGridStyle(3);
76 tmpStyle->SetGridWidth(1);
77 tmpStyle->SetLegendBorderSize(1);
78 tmpStyle->SetLegendFillColor(0);
79 tmpStyle->SetLegendFont(42);
80 tmpStyle->SetLegendTextSize(0);
81 tmpStyle->SetHatchesLineWidth(1);
82 tmpStyle->SetHatchesSpacing(1);
83 tmpStyle->SetFrameFillColor(0);
84 tmpStyle->SetFrameLineColor(1);
85 tmpStyle->SetFrameFillStyle(1001);
86 tmpStyle->SetFrameLineStyle(1);
87 tmpStyle->SetFrameLineWidth(1);
88 tmpStyle->SetFrameBorderSize(1);
89 tmpStyle->SetFrameBorderMode(0);
90 tmpStyle->SetHistFillColor(0);
91 tmpStyle->SetHistLineColor(602);
92 tmpStyle->SetHistFillStyle(1001);
93 tmpStyle->SetHistLineStyle(1);
94 tmpStyle->SetHistLineWidth(1);
95 tmpStyle->SetHistMinimumZero(kFALSE);
96 tmpStyle->SetCanvasPreferGL(kFALSE);
97 tmpStyle->SetCanvasColor(0);
98 tmpStyle->SetCanvasBorderSize(2);
99 tmpStyle->SetCanvasBorderMode(0);
100 tmpStyle->SetCanvasDefH(500);
101 tmpStyle->SetCanvasDefW(700);
102 tmpStyle->SetCanvasDefX(10);
103 tmpStyle->SetCanvasDefY(10);
104 tmpStyle->SetPadColor(0);
105 tmpStyle->SetPadBorderSize(2);
106 tmpStyle->SetPadBorderMode(0);
107 tmpStyle->SetPadBottomMargin(0.1);
108 tmpStyle->SetPadTopMargin(0.1);
109 tmpStyle->SetPadLeftMargin(0.1);
110 tmpStyle->SetPadRightMargin(0.1);
111 tmpStyle->SetPadGridX(kTRUE);
112 tmpStyle->SetPadGridY(kTRUE);
113 tmpStyle->SetPadTickX(0);
114 tmpStyle->SetPadTickY(0);
115 tmpStyle->SetPaperSize(20, 26);
116 tmpStyle->SetScreenFactor(1);
117 tmpStyle->SetStatColor(0);
118 tmpStyle->SetStatTextColor(1);
119 tmpStyle->SetStatBorderSize(2);
120 tmpStyle->SetStatFont(42);
121 tmpStyle->SetStatFontSize(0.040);
122 tmpStyle->SetStatStyle(1001);
123 tmpStyle->SetStatFormat("g");
124 tmpStyle->SetStatX(1.);
125 tmpStyle->SetStatY(0.95);
126 tmpStyle->SetStatW(0.22);
127 tmpStyle->SetStatH(0.19);

```



```

128 tmpStyle->SetStripDecimals(kFALSE);
129 tmpStyle->SetTitleAlign(23);
130 tmpStyle->SetTitleFillColor(0);
131 tmpStyle->SetTitleTextColor(1);
132 tmpStyle->SetTitleBorderSize(0);
133 tmpStyle->SetTitleFont(42);
134 tmpStyle->SetTitleFontSize(0.05);
135 tmpStyle->SetTitleStyle(0);
136 tmpStyle->SetTitleX(0.5);
137 tmpStyle->SetTitleY(0.995);
138 tmpStyle->SetTitleW(0);
139 tmpStyle->SetTitleH(0);
140 tmpStyle->SetLegoInnerR(0.5);
141
142 Int_t fPaletteColor[255] = {
143     924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935,
144     936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947,
145     948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959,
146     960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971,
147     972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983,
148     984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995,
149     996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007,
150     1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019,
151     1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031,
152     1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043,
153     1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055,
154     1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067,
155     1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079,
156     1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091,
157     1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103,
158     1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115,
159     1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127,
160     1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139,
161     1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151,
162     1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163,
163     1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175,
164     1176, 1177, 1178};
165 tmpStyle->SetPalette(255, fPaletteColor);
166
167 TString fLineStyleArrayTmp[30] = {"",
168     " ",
169     " 12 12",
170     " 4 8",
171     " 12 16 4 16",
172     " 20 12 4 12",
173     " 20 12 4 12 4 12 4 12",
174     " 20 20",
175     " 20 12 4 12 4 12",
176     " 80 20",
177     " 80 40 4 40",
178     " ",
179     " ",
180     " ",
181     " ",
182     " ",
183     " ",
184     " ",
185     " ",
186     " ",
187     " ",
188     " ",
189     " ",
190     " ",
191     " ",
192     " ",
193     " ",
194     " ",
195     " ",
196     " "};
197 for (Int_t i = 0; i < 30; i++)
198     tmpStyle->SetLineStyleString(i, fLineStyleArrayTmp[i]);
199
200 tmpStyle->SetHeaderPS("");
201 tmpStyle->SetTitlePS("");
202 tmpStyle->SetFitFormat("g");

```



```
203 tmpStyle->SetPaintTextFormat("g");
204 tmpStyle->SetLineScalePS(3);
205 tmpStyle->SetJoinLinePS(0);
206 // 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->SetFont(42);
222 tmpStyle->SetTextSize(0.05);
223 }
```