Collezione di macro di ROOT

Luca Zoppetti

17/01/2024

Indice

Introduzione

Questo documento contiene una collezione di macro di ROOT che ho scritto per esercitarmi in vista degli esami di Laboratorio del primo e del secondo anno della laurea in Fisica all'Università di Bologna. Tutto il codice è pubblicato anche in una *repository* pubblica su GitHub, per cui è possibile scaricarlo ed eseguirlo se si vuole capire come funziona ciascun esempio riportato di seguito. Tutto quello che pubblico sulla mia pagina è gratuito e ad uso libero per il proprio studio, tuttavia se apprezzassi il mio lavoro e ti andasse di offrirmi un caffè, questo è il mio profilo PayPal: paypal.me/lucazoppetti. Buono studio!

1 Esami del primo anno

1.1 08/06/2018 (Istogramma)

Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma unidimensionale (classe `TH1D`) con 1000 bin, in un range da -1. a 1.;
2. In un ciclo lo si riempie con 10^6 occorrenze di una variabile casuale, estratte da una distribuzione gaussiana con media 0 e deviazione standard 1 (attraverso l'opportuno metodo della classe `TRandom`);
3. Al termine del ciclo si stampano a schermo:
    - La media e la RMS dell'istogramma, con i loro errori;
    - Il contenuto dei bin di underflow e overflow (occorrenze fuori range). Dire anche se ci si aspetta che questi ultimi (underflow e overflow) siano diversi da 0, motivando la risposta.

Risposta

È corretto aspettarsi che i bin di underflow e overflow siano diversi da 0. Il motivo risiede nel fatto che l'intervallo $[-1, 1]$ contiene solo il 68.2% delle occorrenze di una gaussiana con $\mu = 0$ e $\sigma = 1$. Di conseguenza, ci si può aspettare che circa il 15.9% delle occorrenze cada in underflow e il 15.9% cada in overflow.

```
#include <iostream>

#include "TH1D.h"
#include "TRandom.h"

void macro() {
  TH1D *h = new TH1D("h", "Histogram", 1000, -1., 1.);

  Double_t x;
  for (Int_t i = 0; i < 1E6; ++i) {
    x = gRandom->Gaus(0., 1.);
    h->Fill(x);
  }

  std::cout << "Histogram mean: " << h->GetMean() << "+-" << h->GetMeanError()
            << '\n'
            << "Histogram RMS: " << h->GetRMS() << "+-" << h->GetRMSError()
            << '\n'
            << "Underflow: " << h->GetBinContent(0) << '\n'
            << "Overflow: " << h->GetBinContent(1001) << '\n';
}
```

1.2 07/02/2019 (`TGraphErrors`)

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un grafico (class `TGraphErrors`) atto a rappresentare le 5 coppie di misure (x, y) che dovranno essere opportunamente inserite in forma di arrays:
    - Valori in x: `{1,2,3,4,5}`;
    - Valori in y: `{3,9,19,33,51}`;
    - Errori in y: `{0.1,0.4,0.9,1.6,2.5}`.
2. Si disegna il graifco;

3. Si esegue sul grafico un fit secondo una dipendenza funzionale del tipo $f(x) = Ax^2 + B$, dove A e B sono parametri liberi del fit.

```
#include "TCanvas.h"
#include "TF1.h"
#include "TGraphErrors.h"

void macro() {
  Double_t x[]{1, 2, 3, 4, 5};
  Double_t y[]{3, 9, 19, 33, 51};
  Double_t y_err[]{0.1, 0.4, 0.9, 1.6, 2.5};

  TGraphErrors* g = new TGraphErrors(5, x, y, nullptr, y_err);

  TCanvas* c = new TCanvas("c");
  g->Draw();
  TF1* f = new TF1("f", "[0] * x * x + [1]");
  f->SetParameters(1, 1);
  g->Fit(f);
}
```

1.3 10/06/2019 (`TGraphErrors`)

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un grafico atto a rappresentare le 7 coppie di misure (x, y) che dovranno essere opportunamente inserite in forma di arrays:
   - Valori in x: `{-0.75,-0.5,-0.25,0.,0.25,0.5,0.75}`;
   - Valori in y: `{3.1, 0.1 ,-3.2, -0.1, 2.9, 0.1,-3.2}`;
   - Errori in y: `{0.2,0.1,0.2,0.1,0.2,0.1,0.2}`.
2. Si disegna il grafico con l'opzione per visualizzare anche le barre di errore;
3. Si esegue sul grafico un fit secondo una dipendenza funzionale del tipo $f(x) = A\sin(Bx)$, dove A e B sono parametri liberi del fit. A tal fine istanziare un oggetto funzione `TF1` in modo opportuno.
4. Si stampano a schermo i valori dei due parametri e le relative incertezze, e il valore del χ^2 ridotto.

```
#include "TCanvas.h"
#include "TF1.h"
#include "TGraphErrors.h"
#include "TMath.h"

Double_t fitFunction(Double_t *x, Double_t *par) {
  Double_t xx = x[0];
  Double_t val = par[0] * TMath::Sin(par[1] * xx);
  return val;
}

void macro() {
  Double_t x[]{-0.75, -0.5, -0.25, 0., 0.25, 0.5, 0.75};
  Double_t y[]{3.1, 0.1, -3.2, -0.1, 2.9, 0.1, -3.2};
  Double_t y_err[]{0.2, 0.1, 0.2, 0.1, 0.2, 0.1, 0.2};
  TGraphErrors *g = new TGraphErrors(7, x, y, nullptr, y_err);

  TF1 *f = new TF1("f", fitFunction, -1., 1., 2);
  f->SetParameters(3., 6.);
  TCanvas *c = new TCanvas("c");
  g->Fit(f, "Q");
  g->Draw();

  std::cout << "par0: " << f->GetParameter(0) << "+-" << f->GetParError(0)
            << '\n'
            << "par1: " << f->GetParameter(1) << "+-" << f->GetParError(1)
            << '\n'
            << "Reduced chi square: " << f->GetChisquare() / f->GetNDF()
            << '\n';
}
```

1.4 23/06/2021 (`TGraphErrors` con dati da file)

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un grafico (classe `TGraphErrors`) atto a rappresentare le 10 coppie di misure (x,y) e l'incertezza sui valori in y che sono, nell'ordine (x,y,dy), salvati nelle tre colonne del file di testo denominato `data.txt` come segue: `txt 1 6 0.6 2 17 1.7 3 34 3.4 4 57 5.7 5 86 8.6 6 121 12.1 7 162 16.2 8 209 20.9 9 262 26.2 10 321 32.1`
2. Si disegna il grafico, inserendo i nomi degli assi;
3. Si esegue sul grafico un fit secondo una dipendenza funzionale del tipo $Ax^2 + Bx + C$, dove A, B e C sono parametri liberi del fit. Si stampi a schermo il valore dei parametri dopo il fit con le relative incertezze e il χ^2 ridotto.

```
#include "TF1.h"
#include "TGraphErrors.h"
#include "TMath.h"

Double_t fitFunction(Double_t *x, Double_t *par) {
  Double_t xx = *x;
  Double_t val = par[0] * xx * xx + par[1] * xx + par[2];
  return val;
}

void macro() {
  TGraphErrors *g = new TGraphErrors("data.txt", "%lg %lg %lg");
  g->SetTitle("Grafico;x;y");

  TF1 *f = new TF1("f", fitFunction, 0., 10., 3);
  f->SetParameters(1., 1., 1.);
  g->Fit(f, "Q");
  g->Draw();

  std::cout << "A: " << f->GetParameter(0) << "+-" << f->GetParError(0) << '\n'
            << "B: " << f->GetParameter(1) << "+-" << f->GetParError(1) << '\n'
            << "C: " << f->GetParameter(2) << "+-" << f->GetParError(2) << '\n'
            << "Reduced chi square: " << f->GetChisquare() / f->GetNDF()
            << '\n';
}
```

1.5 12/07/2022 (`TGraphErrors`)

Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un grafico (classe `TGraphErrors`) atto a rappresentare le 5 coppie di misure (x,y) che dovranno essere opportunamente inserite in forma di arrays:

   - Valori in x: `{1,4,9,16,25}`;
   - Valori in y: `{10,20,30,40,50}`;
   - Errori in x: `{0.1,0.4,0.9,1.6,2.5}`;
   - Errori in y: `{0.1,0.2,0.3,0.4,0.5}`.

2. Si disegna il grafico, inserendo come titoli degli assi "Misure in x" e "Misure in y".
3. Si esegue sul grafico un fit secondo una dipendenza funzionale del tipo $f(x) = A + B\sqrt{x}$, dove A e B sono i parametri liberi del fit. Stampare a schermo i valori di A e B esito del fit, con rispettive incertezze, e il χ^2 ridotto.

```
#include "TF1.h"
#include "TGraphErrors.h"
#include "TMath.h"

Double_t fitFunction(Double_t *x, Double_t *par) {
  Double_t xx = *x;
  Double_t val = par[0] + par[1] * TMath::Sqrt(xx);
  return val;
}

```

```
void macro() {
  Double_t x[]{1, 4, 9, 16, 25};
  Double_t y[]{10, 20, 30, 40, 50};
  Double_t x_err[]{0.1, 0.4, 0.9, 1.6, 2.5};
  Double_t y_err[]{0.1, 0.2, 0.3, 0.4, 0.5};

  TGraphErrors *g = new TGraphErrors(5, x, y, x_err, y_err);
  g->SetTitle("Grafico;Misure in x;Misure in y");

  TF1 *f = new TF1("f", fitFunction, 0, 30, 2);
  f->SetParameters(1., 1.);
  g->Fit(f, "Q");
  g->Draw();

  std::cout << "A: " << f->GetParameter(0) << "+-" << f->GetParError(0) << '\n'
            << "B: " << f->GetParameter(1) << "+-" << f->GetParError(1) << '\n'
            << "Reduced chi square: " << f->GetChisquare() / f->GetNDF()
            << '\n';
}
```

1.6 06/09/2022 (Istogramma)

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma unidimensionale con 1000 bin, in un range da 0. a 10.;
2. In un ciclo si riempie l'istogramma con 10^7 occorrenze di una variabile casuale estratta da una distribuzione gaussiana con media $\mu = 5$ e $\sigma = 1$;
3. Al termine del ciclo si esegue il fit dell'istogramma con una funzione gaussiana esplicitamente definita attraverso un `TF1` (tre parametri liberi: ampiezza, media e deviazione standard) e si stampano a schermo:

    - La media e la RMS dell'istogramma risultante con le rispettive incertezze;
    - Il valore dei tre parametri della funzione dopo il fit, con rispettive incertezze, e il χ^2 ridotto del fit.

```
#include "TCanvas.h"
#include "TF1.h"
#include "TH1F.h"
#include "TMath.h"
#include "TRandom.h"

Double_t fitFunction(Double_t *x, Double_t *par) {
  Double_t xx = *x;
  Double_t val = par[0] * TMath::Exp(-(xx - par[1]) * (xx - par[1]) / 2. /
                                     par[2] / par[2]);
  return val;
}

void macro() {
  TH1F *h = new TH1F("h", "Histogram", 1000, 0., 10.);

  Double_t x;
  for (Int_t i = 0; i < 1E7; ++i) {
    x = gRandom->Gaus(5., 1.);
    h->Fill(x);
  }

  TF1 *f = new TF1("f", fitFunction, 0., 10., 3);
  f->SetParameters(1, 1, 1);
  // Q disables console output
  h->Fit(f, "Q");

  Double_t hMean = h->GetMean();
  Double_t hMeanErr = h->GetMeanError();
  Double_t hRMS = h->GetRMS();
  Double_t hRMSErr = h->GetRMSError();
  Double_t par0 = f->GetParameter(0);
  Double_t par0Err = f->GetParError(0);
  Double_t par1 = f->GetParameter(1);
```

```
  Double_t par1Err = f->GetParError(1);
  Double_t par2 = f->GetParameter(2);
  Double_t par2Err = f->GetParError(2);
  Double_t chiSquare = f->GetChisquare();
  Double_t NDOF = f->GetNDF();

  std::cout << "Histogram mean: " << hMean << "+-" << hMeanErr << '\n'
            << "Histogram RMS: " << hRMS << "+-" << hRMSErr << '\n'
            << "par0: " << par0 << "+-" << par0Err << '\n'
            << "par1: " << par1 << "+-" << par1Err << '\n'
            << "par2: " << par2 << "+-" << par2Err << '\n'
            << "Reduced chi square: " << chiSquare / NDOF << '\n';
}
```

1.7 15/06/2023 (Somma di istogrammi)

Creare due istogrammi con 500 bin ciascuno fra 0 e 5. Riempire il primo istogramma con 10^4 occorrenze generate tramite una distribuzione gaussiana con media 2.5 e deviazione standard 0.1. Riempire il secondo con lo stesso numero di occorrenze estratte da una distribuzione uniforme. In seguito, eseguire la somma dei due istogrammi ed eseguire il fit dell'istogramma risultante a una funzione gaussiana sommata a una distribuzione uniforme. Infine, stampare a schermo la media e l'RMS dell'istogramma, i parametri risultanti dal fit e il χ^2 ridotto del fit.

```
#include <TCanvas.h>
#include <TF1.h>
#include <TH1F.h>
#include <TMath.h>
#include <TRandom.h>

Double_t fitFunction(Double_t *x, Double_t *par) {
  Double_t xx = x[0];
  Double_t val = par[0] * TMath::Exp(-(xx - par[1]) * (xx - par[1]) / 2. /
                                     par[2] / par[2]) +
                 par[3];
  return val;
}

void macro() {
  constexpr Int_t nGen = 1E4;
  TH1F *g = new TH1F("g", "Gauss", 500, 0., 5.);
  TH1F *u = new TH1F("u", "Uniform", 500, 0., 5.);

  Double_t x;
  for (Int_t i = 0; i < nGen; ++i) {
    x = gRandom->Gaus(2.5, 0.1);
    g->Fill(x);

    x = gRandom->Uniform(0., 5.);
    u->Fill(x);
  }

  TH1F *sum = new TH1F(*g);
  sum->SetName("sum");
  sum->SetTitle("Sum");
  sum->Add(g, u);

  TF1 *f = new TF1("f", fitFunction, 0, 5, 4);
  f->SetParameters(1, 1, 1, 1);

  TCanvas *canvas1 = new TCanvas("canvas1");
  sum->Fit(f, "Q");
  sum->Draw();

  TCanvas *canvas2 = new TCanvas("canvas2");
  canvas2->Divide(2, 1);
  canvas2->cd(1);
  g->Draw();
  canvas2->cd(2);
  u->Draw();

  TF1 *fitFunc = sum->GetFunction("f");
```

```
  Double_t hMean = sum->GetMean();
  Double_t hMeanErr = sum->GetMeanError();
  Double_t hRMS = sum->GetRMS();
  Double_t hRMSErr = sum->GetRMSError();
  Double_t par0 = fitFunc->GetParameter(0);
  Double_t par0Err = fitFunc->GetParError(0);
  Double_t par1 = fitFunc->GetParameter(1);
  Double_t par1Err = fitFunc->GetParError(1);
  Double_t par2 = fitFunc->GetParameter(2);
  Double_t par2Err = fitFunc->GetParError(2);
  Double_t par3 = fitFunc->GetParameter(3);
  Double_t par3Err = fitFunc->GetParError(3);
  Double_t chiSquare = fitFunc->GetChisquare();
  Double_t NDOF = fitFunc->GetNDF();

  std::cout << "Histogram mean: " << hMean << "+-" << hMeanErr << '\n'
            << "Histogram RMS: " << hRMS << "+-" << hRMSErr << '\n'
            << "par0: " << par0 << "+-" << par0Err << '\n'
            << "par1: " << par1 << "+-" << par1Err << '\n'
            << "par2: " << par2 << "+-" << par2Err << '\n'
            << "par3: " << par3 << "+-" << par3Err << '\n'
            << "Reduced chi square: " << chiSquare / NDOF << '\n';
}
```

2 Esami del secondo anno

2.1 15/01/2018

Quesito 1 (Efficienza)
Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definiscono 2 istogrammi monodimensionali di 1000 bin in un range da 0 a 5.
2. Si riempie il primo istogramma con 10^7 occorrenze di una variabile casuale x generate esplicitamente e singolarmente (attraverso `gRandom`) e distribuite secondo una distribuzione esponenziale decrescente con media $\mu = 1$ (campione totale).
3. Su tali occorrenze, si simula (attraverso un criterio di reiezione di tipo "hit or miss") un'efficienza di rivelazione dipendente dalla variabile casuale x secondo la forma: $\varepsilon(x) = x/5$. Riempire il secondo istogramma con le occorrenze accettate (campione accettato).
4. Si effettua la divisione fra i due istogrammi per ottenere l'efficienza di rivelazione osservata, utilizzando il metodo Divide della classe degli istogrammi e inserendo l'opportuna opzione per la valutazione degli errori secondo la statistica binomiale.
5. Si disegna l'istogramma dell' efficienza visualizzando le incertezze sui contenuti dei bin.

Quesito 2 (Fit)
Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definiscono 2 istogrammi monodimensionali di 500 bin in un range da 0 a 5.
2. Si riempe il primo istogramma con 10^6 occorrenze generate esplicitamente e singolarmente secondo una distribuzione gaussiana con media $\mu = 2.5$ e deviazione standard $\sigma = 0.25$.
3. Si riempe il secondo istogramma con 10^4 occorrenze generate esplicitamente e singolarmente secondo una distribuzione uniforme del range [0,5].
4. Si fa la somma dei due istogrammi, e si effettua il Fit dell'istogramma somma secondo una forma funzionale consistente di una gaussiana (3 parametri: ampiezza,media e deviazione standard) e un polinomio di grado 0 (1 parametro), per un totale di 4 parametri liberi.
5. Si stampa a schermo il valore dei parametri dopo il fit, con relativo errore, e il χ^2 ridotto.

Quesito 3 (`FillRandom`)
Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma monodimensionale di 100 bin in un range da 0 a 10.
2. Si riempie l'istogramma con 10^7 occorrenze di una variabile casuale x distribuita secondo la p.d.f. $f(x) = \sqrt{x} + x^2$ nel range [0,10] , utilizzando il metodo `FillRandom(const char\* f,Int_t N)` della classe di istogrammi.

2.1.1 Risposta al quesito 1 (Efficienza)

```
#include "TH1F.h"
#include "TRandom.h"

void macro1() {
  gRandom->SetSeed();

  TH1F* h_gen = new TH1F{"h_gen", "Histogram 1", 1000, 0., 5.};
  h_gen->Sumw2();
  TH1F* h_acc = new TH1F{"h_acc", "Histogram 2", 1000, 0., 5.};
  h_acc->Sumw2();

  for (int j{}; j < 1e7; ++j) {
    auto x = gRandom->Exp(1);
    h_gen->Fill(x);

    auto y = gRandom->Uniform(0, 1);
    if (y < x / 5) {
      h_acc->Fill(x);
    }
  }

  TH1F* h_eff = new TH1F(*h_acc);
  h_eff->SetName("h_eff");
  h_eff->SetTitle("Efficiency");
  h_eff->Divide(h_acc, h_gen, 1, 1, "B");
  h_eff->Draw("E");
}
```

2.1.2 Risposta al quesito 2 (Fit)

```
#include "TF1.h"
#include "TH1F.h"
#include "TMath.h"
#include "TRandom.h"

Double_t fitFunc(Double_t* x, Double_t* par) {
  double xx = x[0];
  double y = par[0] * TMath::Exp(-(xx - par[1]) * (xx - par[1]) /
                                 (2 * par[2] * par[2])) +
             par[3];
  return y;
}

void macro2() {
  gRandom->SetSeed();

  TH1F* histo1 = new TH1F{"histo1", "Histogram 1", 500, 0, 5};
  histo1->Sumw2();
  TH1F* histo2 = new TH1F{"histo2", "Histogram 2", 500, 0, 5};
  histo2->Sumw2();

  for (int i{}; i < 1e6; ++i) {
    auto x1 = gRandom->Gaus(2.5, 0.25);
    histo1->Fill(x1);

    if (i < 1e4) {
      auto x2 = gRandom->Uniform(0, 5);
      histo2->Fill(x2);
    }
  }

  TF1* f = new TF1{"f", fitFunc, 0, 5, 4};
  f->SetParameter(0, 16000);
  f->SetParameter(1, 2.5);
  f->SetParameter(2, 0.25);
  f->SetParameter(3, 20);

  TH1F* sum = new TH1F{*histo1};
  sum->SetName("sum");
  sum->SetTitle("Sum");
  sum->Add(histo1, histo2);
```

```
42
43   sum->Fit(f, "Q");
44   std::cout << "Parameter 0: " << f->GetParameter(0) << " +- "
45             << f->GetParError(0) << '\n'
46             << "Parameter 1: " << f->GetParameter(1) << " +- "
47             << f->GetParError(1) << '\n'
48             << "Parameter 2: " << f->GetParameter(2) << " +- "
49             << f->GetParError(2) << '\n'
50             << "Parameter 3: " << f->GetParameter(3) << " +- "
51             << f->GetParError(3) << '\n'
52             << "Reduced chi square: " << f->GetChisquare() / f->GetNDF()
53             << '\n';
54 }
```

2.1.3 Risposta al quesito 3 (`FillRandom`)

```
1  #include "TF1.h"
2  #include "TH1F.h"
3  #include "TMath.h"
4
5  Double_t fitFunc(Double_t* xx) {
6    double x = xx[0];
7    return TMath::Sqrt(x) + x * x;
8  }
9
10 void macro3() {
11   TH1F* histo = new TH1F{"histo", "Histogram", 100, 0, 10};
12
13   TF1* f = new TF1{"f", "sqrt(x) + x^2", 0, 10};
14
15   histo->FillRandom("f", 1e7);
16
17   histo->Draw();
18 }
```

2.2 26/06/2018

Quesito 1 (Efficienza)
Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definiscono 2 istogrammi monodimensionali di 1000 bin in un range da 0 a 10.
2. Si riempe il primo istogramma con 10^7 occorrenze di una variabile casuale x generate esplicitamente e singolarmente e distribuite secondo una distribuzione gaussiana con $\mu = 5$ e deviazione standard $\sigma = 1$ (campione totale).
3. Su tali occorrenze, si simula (attraverso un criterio di reiezione di tipo "hit or miss") un'efficienza di rivelazione dipendente dalla variabile casuale x secondo la forma: $\varepsilon(x) = 0.1xe^{-x}$. Riempire il secondo istogramma con le occorrenze accettate (campione accettato).
4. Si effettua la divisione fra i due istogrammi per ottenere l'efficienza di rivelazione osservata, utilizzando il metodo Divide della classe degli istogrammi e inserendo l'opportuna opzione per la valutazione degli errori secondo la statistica binomiale.
5. Si disegna l'istogramma dell' efficienza visualizzando le incertezze sui contenuti dei bin.

Quesito 2 (Fit)
Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definiscono 2 istogrammi monodimensionali di 500 bin in un range da 0 a 5.
2. Si riempie il primo istogramma con 10^6 occorrenze generate esplicitamente e singolarmente secondo una distribuzione gaussiana con media $\mu = 2$ e deviazione standard $\sigma = 0.5$.
3. Si riempie il secondo istogramma con 10^5 occorrenze generate esplicitamente e singolarmente secondo una distribuzione esponenziale decrescente con media $\mu = 1$.
4. Si fa la somma dei due istogrammi, e si effettua il Fit dell'istogramma somma secondo una forma funzionale consistente di una gaussiana (3 parametri) e un esponenziale (2 parametri), per un totale di 5 parametri liberi.
5. Si stampa a schermo il valore dei parametri dopo il fit, con relativo errore, e il χ^2 ridotto.

Quesito 3 (`FillRandom`)
Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma monodimensionale di 100 bin in un range da 0 a 10.
2. Si riempie l'istogramma con 10^5 occorrenze di una variabile casuale x distribuita secondo la p.d.f. $f(x) = \sin(x) + x^2$ nel range [0,10], utilizzando il metodo `FillRandom(const char* f,Int_t N)` della classe di istogrammi.

2.2.1 Risposta al quesito 1 (Efficienza)

```
#include "TCanvas.h"
#include "TH1F.h"
#include "TMath.h"
#include "TRandom.h"

void macro1() {
  gRandom->SetSeed();

  TH1F* h_gen = new TH1F{"h_gen", "Histogram 1", 1000, 0, 10};
  h_gen->Sumw2();
  TH1F* h_acc = new TH1F{"h_acc", "Histogram 2", 1000, 0, 10};
  h_acc->Sumw2();

  for (int j{}; j < 1e7; ++j) {
    auto x = gRandom->Gaus(5, 1);
    h_gen->Fill(x);

    auto y = gRandom->Uniform(0, 1);
    if (y < 0.1 * x * TMath::Exp(-x)) {
      h_acc->Fill(x);
    }
  }

  TH1F* h_eff = new TH1F{*h_acc};
  h_eff->SetName("h_eff");
  h_eff->SetTitle("Division");
  h_eff->Divide(h_acc, h_gen, 1, 1, "B");
  h_eff->Draw("E");
}
```

2.2.2 Risposta al quesito 2 (Fit)

```
#include "TF1.h"
#include "TH1F.h"
#include "TMath.h"
#include "TRandom.h"

Double_t fitFunc(Double_t* xx, Double_t* par) {
  double x = xx[0];
  return par[0] *
              TMath::Exp(-(x - par[1]) * (x - par[1]) / (2 * par[2] * par[2])) +
         par[3] * TMath::Exp(-x / par[4]);
}

void macro2() {
  gRandom->SetSeed();

  TH1F* gauss = new TH1F{"gauss", "Gaussian", 500, 0, 5};
  TH1F* exp = new TH1F{"uniform", "Uniform", 500, 0, 5};

  for (int i{}; i < 1e6; ++i) {
    auto x1 = gRandom->Gaus(2, 0.5);
    gauss->Fill(x1);

    if (i < 1e5) {
      auto x2 = gRandom->Exp(1);
      exp->Fill(x2);
    }
  }

  TF1* f = new TF1{"f", fitFunc, 0, 5, 5};
```

```
  f->SetParameter(0, 8000);
  f->SetParameter(1, 2);
  f->SetParameter(2, 0.6);
  f->SetParameter(3, 1500);
  f->SetParameter(4, 1);

  TH1F* sum = new TH1F{*gauss};
  sum->SetName("sum");
  sum->SetTitle("Sum");
  sum->Add(gauss, exp);
  sum->Fit(f, "Q");

  std::cout << "Parameter 0: " << f->GetParameter(0) << " +- "
            << f->GetParError(0) << '\n'
            << "Parameter 1: " << f->GetParameter(1) << " +- "
            << f->GetParError(1) << '\n'
            << "Parameter 2: " << f->GetParameter(2) << " +- "
            << f->GetParError(2) << '\n'
            << "Parameter 3: " << f->GetParameter(3) << " +- "
            << f->GetParError(3) << '\n'
            << "Parameter 4: " << f->GetParameter(4) << " +- "
            << f->GetParError(4) << '\n'
            << "Reduced chi square: " << f->GetChisquare() / f->GetNDF()
            << '\n';
}
```

2.2.3 Risposta al quesito 3 (`FillRandom`)

```
#include "TF1.h"
#include "TH1F.h"

void macro3() {
  TH1F* histo = new TH1F{"histo", "Histogram", 100, 0, 10};

  TF1* f = new TF1{"f", "sin(x) + x^2", 0, 10};

  histo->FillRandom("f", 1e5);

  histo->Draw();
}
```

2.3 19/01/2024

Quesito 1 (Istogrammi e `TList`)

Importante: in tutti i punti del quesito, sfruttare quanto più possibile la possibilità di scrivere il codice in forma di cicli.

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definiscono in un ciclo 4 istogrammi monodimensionali di tipo `TH1D`, aventi 1000 bin in un range da 0 a 10, e con nomi root `{"histo0","histo1","histo2","histo3"}`, attraverso un array di puntatori di tipo `TH1*` e successiva allocazione dinamica degli oggetti.
2. Si impostano i titoli dell'asse x e y rispettivamente a "x variable" e "entries".
3. Si riempie ciascuno degli istogrammi con 10^6 occorrenze di una variabile casuale generata *esplicitamente e singolarmente* (i.e. attraverso `gRandom`) secondo delle distribuzioni gaussiane con media, rispettivamente, pari a $\mu = 2.5, 4.5, 6.5, 8.6$ e deviazione standard σ pari al 10% della media corrispondente.
4. Si inseriscono i quattro istogrammi in un oggetto contenitore di tipo `TList`.
5. Successivamente, sempre utilizzando un ciclo, si disegnano su una `TCanvas` divisa in 4 pad i quattro istogrammi recuperandoli dalla `TList`, e si stampano a schermo media e RMS con relative incertezze.

Quesito 2 (Categorie)

Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma monodimensionale di 5 bin in un range da 0 a 5, finalizzato a visualizzare il numero di conteggi osservati nelle categorie che contraddistinguono la popolazione di cui a punto successivo.

2. Si genera una popolazione di $N = 10^7$ elementi appartenenti a 5 categorie distinte, con rispettive probabilità di occorrenza:

- Caso 0: 5%
- Caso 1: 15%
- Caso 2: 15%
- Caso 3: 35%
- Caso 4: 30%

e si riempie l'istogramma con le occorrenze osservate nei diversi casi.

3. Si stampano a schermo *le frazioni* di popolazione osservate nei cinque bin dell'istogramma, utilizzando i metodi espliciti degli istogrammi per estrarre il numero di ingressi dell'istogramma e il contenuto dei bin.

4. Si salva l'istogramma su un file ROOT.

Quesito 3 (PDF definita a tratti)
Si scriva la parte rilevante e autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma monodimensionale di 100 bin in un range da 0 a 5.

2. Si riempie l'istogramma con 10^6 occorrenze di una variabile casuale x distribuita secondo la p.d.f.:

   - $f(x) = 0.2$ per $0 \leq x < 1$
   - $f(x) = x/5$ per $1 \leq x < 5$

   nel range $[0, 5]$, utilizzando il metodo `FillRandom(const char* f, Int_t N)` della classe degli istogrammi.

3. Monitorare il tempo di esecuzione del punto 2. attraverso la funzionalità `TBenchmark`.

2.3.1 Risposta al quesito 1 (Istogrammi e `TList`)

```
#include <iostream>
#include <vector>

#include "TCanvas.h"
#include "TH1D.h"
#include "TList.h"
#include "TRandom.h"
#include "TString.h"

void macro1() {
  TString histName = "histo";
  TH1* histo[4];

  std::vector<double> mu{2.5, 4.5, 6.5, 8.5};

  TList* list = new TList();

  TCanvas* canvas = new TCanvas();
  canvas->Divide(2, 2);

  for (int j{}; j < 4; ++j) {
    histo[j] = new TH1D{histName + j, histName + j, 1000, 0, 10};
    histo[j]->GetXaxis()->SetTitle("x variable");
    histo[j]->GetYaxis()->SetTitle("entries");

    auto mean = mu[j];
    auto stddev = mean / 10.;

    for (int i{}; i < 1e6; ++i) {
      auto x = gRandom->Gaus(mean, stddev);
      histo[j]->Fill(x);
    }

    list->Add(histo[j]);

    canvas->cd(j + 1);
```

```
37
38      auto current_histo = (TH1*)list->At(j);
39      current_histo->Draw();
40
41      auto histo_mean = current_histo->GetMean();
42      auto histo_mean_err = current_histo->GetMeanError();
43      auto histo_rms = current_histo->GetRMS();
44      auto histo_rms_err = current_histo->GetRMSError();
45
46      std::cout << "HISTOGRAM " << j << ": " << '\n'
47                << "Mean: " << histo_mean << " +- " << histo_mean_err << '\n'
48                << "RMS: " << histo_rms << " +- " << histo_rms_err << '\n';
49    }
50  }
```

2.3.2 Risposta al quesito 2 (Categorie)

```
1  #include <iostream>
2
3  #include "TFile.h"
4  #include "TH1I.h"
5  #include "TRandom.h"
6
7  void macro2() {
8    TH1I* histo = new TH1I{"histo", "Histogram", 5, 0, 5};
9
10   for (int j{}; j < 1e7; ++j) {
11     auto x = gRandom->Uniform(0, 1);
12
13     if (x <= 0.05) {
14       histo->Fill(0);
15     } else if (x <= 0.20) {
16       histo->Fill(1);
17     } else if (x <= 0.35) {
18       histo->Fill(2);
19     } else if (x <= 0.70) {
20       histo->Fill(3);
21     } else {
22       histo->Fill(4);
23     }
24   }
25
26   double entries = histo->GetEntries();
27
28   for (int j{}; j < 5; ++j) {
29     double occurrences = histo->GetBinContent(j + 1);
30     double frac = occurrences / entries;
31
32     std::cout << "Caso " << j << ": " << frac << '\n';
33   }
34
35   TFile* file = new TFile("macro2.root", "recreate");
36   histo->Write();
37   file->Close();
38 }
```

2.3.3 Risposta al quesito 3 (PDF definita a tratti)

```
1  #include "TBenchmark.h"
2  #include "TF1.h"
3  #include "TH1F.h"
4
5  double pdfFunc(double *xx, double *par) {
6    double x = xx[0];
7    if (0 <= x && x < 1) {
8      return 0.2;
9    } else {
10     return x / 5.;
11   }
12 }
13
14 void macro3() {
15   TH1F *histo = new TH1F{"histo", "Histogram", 100, 0, 5};
```

```
  TF1 *pdf = new TF1{"pdf", pdfFunc, 0, 5, 0};

  gBenchmark->Start("FillRandom");
  histo->FillRandom("pdf", 1e6);
  gBenchmark->Show("FillRandom");
}
```

3 Esercizi

3.1 Benchmark

Si scriva la parte rilevante ed autoconsistente di codice di una macro di ROOT rivolto a monitorare con la classe `TBenchmark` il tempo di CPU rispettivamente impiegato per fare le due seguenti operazioni, utilizzando anche l'opportuno metodo per stampare a schermo i tempi di esecuzione:

1. Generare 10^5 occorrenze generate esplicitamente e singolarmente di una gaussiana con media $\mu = 0$ e deviazione standard $\sigma = 1$, riempiendo un istogramma `h1` che si assume già opportunamente definito.
2. Generare 10^5 occorrenze di una gaussiana `G(0,1)` con il metodo `FillRandom`, riempiendo un istogramma `h2` che si assume già opportunamente definito.

```
#include "TBenchmark.h"
#include "TCanvas.h"
#include "TF1.h"
#include "TH1F.h"
#include "TRandom.h"

void macro() {
  TH1F* h1 = new TH1F{"h1", "Histogram 1", 1000, -5, 5};
  TH1F* h2 = new TH1F{"h2", "Histogram 2", 1000, -5, 5};

  gBenchmark->Start("With loop");

  for (int j{}; j < 1e5; ++j) {
    auto x = gRandom->Gaus(0, 1);
    h1->Fill(x);
  }

  gBenchmark->Show("With loop");

  gBenchmark->Start("With FillRandom");

  h2->FillRandom("gaus", 1e5);

  gBenchmark->Show("With FillRandom");
}
```

3.2 Categorie

Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma monodimensionale di 4 bin in un range da 0 a 4.
2. Si generi una popolazione di 10^6 elementi appartenenti a 4 categorie distinte, con rispettive probabilità di occorrenza:
   - Caso 0: 60%
   - Caso 1: 30%
   - Caso 2: 9%
   - Caso 3: 1%
3. Si stampi a schermo il contenuto dei 4 bin dell'istogramma e relativa incertezza usando i metodi espliciti degli istogrammi `GetBinContent()` e `GetBinError()`.

```cpp
#include <iostream>

#include "TH1I.h"
#include "TRandom.h"

void macro() {
  TH1I* histo = new TH1I{"histo", "Categories", 4, 0, 4};

  for (int j{}; j < 1e6; ++j) {
    auto x = gRandom->Uniform(0, 1);

    if (x <= 0.6) {
      histo->Fill(0);
    } else if (x <= 0.9) {
      histo->Fill(1);
    } else if (x <= 0.99) {
      histo->Fill(2);
    } else {
      histo->Fill(3);
    }
  }

  for (int j{}; j < 4; ++j) {
    std::cout << "Case " << j << " occurrences: " << histo->GetBinContent(j + 1)
              << " +- " << histo->GetBinError(j + 1) << '\n';
  };
}
```

3.3 Efficienza

Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definiscono 2 istogrammi monodimensionali di 100 bin in un range da 0 a 10.
2. Si riempie il primo istogramma con 10^5 occorrenze di una variabile casuale x generate esplicitamente e singolarmente e distribuite secondo una distribuzione gaussiana con media $\mu = 5$ e deviazione standard $\sigma = 1$ (campione totale).
3. Su tali occorrenze, si simula (attraverso un criterio di reiezione di tipo "hit or miss") un'efficienza di rivelazione dipendente dalla variabile casuale x secondo la forma:

   - $\varepsilon(x) = 0.3$ per $0 \leq x \leq 3$
   - $\varepsilon(x) = 0.7$ per $x \geq 3$

   Riempire il secondo istogramma con le occorrenze accettate (campione accettato).
4. Si effettua la divisione fra i due istogrammi per ottenere l'efficienza di rivelazione osservata, utilizzando il metodo `Divide` della classe degli istogrammi e inserendo l'opportuna opzione per la valutazione degli errori secondo la statistica binomiale.
5. Si disegnano i tre istogrammi (campione totale, campione accettato, efficienza) su un'unica canvas divisa in tre pad.

```cpp
#include "TCanvas.h"
#include "TH1F.h"
#include "TRandom.h"

double efficiency(double x) {
  if (0 <= x && x <= 3) {
    return 0.3;
  } else {
    return 0.7;
  }
}

void macro() {
  gRandom->SetSeed();

```

```
  TH1F* hgen = new TH1F{"hgen", "Campione totale", 100, 0, 10};
  hgen->Sumw2();
  TH1F* hacc = new TH1F{"hacc", "Campione accettato", 100, 0, 10};
  hacc->Sumw2();

  for (int j{}; j < 1e5; ++j) {
    auto x = gRandom->Gaus(5, 1);
    hgen->Fill(x);

    auto y = gRandom->Uniform(0, 1);
    if (y < efficiency(x)) {
      hacc->Fill(x);
    }
  }

  TH1F* heff = new TH1F{*hgen};
  heff->SetName("heff");
  heff->SetTitle("Efficienza");
  heff->Divide(hacc, hgen, 1, 1, "B");

  TCanvas* canvas = new TCanvas();
  canvas->Divide(2, 2);
  canvas->cd(1);
  hgen->Draw();
  canvas->cd(2);
  hacc->Draw();
  canvas->cd(3);
  heff->Draw();
}
```

3.4 `FillRandom`

Si scriva la parte rilevante ed autoconsistente del codice di una macro di ROOT in cui:

1. Si definisce un istogramma monodimensionale di 100 bin in un range da 0 a 10.
2. Si riempie l'istogramma utilizzando il metodo `FillRandom(const char* f, Int_t N)` degli istogrammi con 10^5 occorrenze di una variabile casuale x distribuita secondo la p.d.f. definita come segue:
   - $f(x) = x/5$ per $0 \leq x < 5$
   - $f(x) = 1$ per $5 \leq x \leq 10$

```
#include "TF1.h"
#include "TH1F.h"

// Define a step PDF
double pdfFunc(double* xx, double* par) {
  double x = xx[0];

  if (0 <= x && x < 5) {
    return x / 5;
  } else {
    return 1;
  }
}

void macro() {
  TH1F* histo = new TH1F{"histo", "Histogram", 100, 0, 10};
  TF1* pdf = new TF1{"pdf", pdfFunc, 0, 10, 0};

  histo->FillRandom("pdf", 1e5);

  histo->Draw();
}
```

4 Macro realizzate a lezione

4.1 Riempimento e fit di istogrammi

```
void setStyle() {
  gROOT->SetStyle("Plain");
  gStyle->SetPalette(57);
  gStyle->SetOptTitle(0);
}

// Gaussiana 1-D definita nella funzione utente
Double_t myFunction(Double_t *x, Double_t *par) {
  Double_t xx = x[0];
  Double_t val = par[0] * TMath::Exp(-(xx - par[1]) * (xx - par[1]) / 2. /
                                     par[2] / par[2]);
  return val;
}

void myMacro(Int_t nGen = 1E5) {
  gStyle->SetOptStat(2210);
  gStyle->SetOptFit(1111);

  TString histName = "h";
  TH1F *h[2];
  for (Int_t i = 0; i < 2; i++) {
    h[i] = new TH1F(histName + i, "test histogram", 100, -5, 5.);
    // cosmetics
    h[i]->SetMarkerStyle(20);
    h[i]->SetMarkerSize(0.5);
    h[i]->SetLineColor(1);
    h[i]->GetYaxis()->SetTitleOffset(1.2);
    h[i]->GetXaxis()->SetTitleSize(0.04);
    h[i]->GetYaxis()->SetTitleSize(0.04);
    h[i]->GetXaxis()->SetTitle("x");
    h[i]->GetYaxis()->SetTitle("Entries");
  }

  h[0]->SetFillColor(kBlue);
  h[1]->SetFillColor(kRed);

  // filling the histos with FillRandom

  h[0]->FillRandom("gaus", nGen);  // gaus predefined function (G(0,1))

  TF1 *f = new TF1("f", myFunction, -10, 10, 3);  // user defined function
  f->SetParameter(0, 1);
  f->SetParameter(1, 0);
  f->SetParameter(2, 1);
  h[1]->FillRandom("f", nGen);

  // drawing

  TCanvas *c1 = new TCanvas("c1", "TF1 examples", 200, 10, 600, 400);
  h[0]->Fit("gaus");
  h[0]->Draw("H");
  h[0]->Draw("E,P,SAME");

  TCanvas *c2 = new TCanvas("c2", "TF1 examples", 200, 10, 600, 400);
  f->SetParameter(0, 4000);
  f->SetParameter(1, 0);
  f->SetParameter(2, 1);
  //  f->SetParName(0,"A");
  //  f->SetParName(1,"#mu");
  //  f->SetParName(2,"#sigma");
  f->SetLineColor(kCyan);
  TFitResultPtr fRes = h[1]->Fit(f, "S", "");
  h[1]->Draw("H");
  h[1]->Draw("E,P,SAME");
  TLegend *leg = new TLegend(.1, .7, .3, .9, "test Fit ");
  leg->SetFillColor(0);
  leg->AddEntry(h[1], "Punti Sperimentali");
  leg->AddEntry(f, "Fit Gaussiano");
  leg->Draw("S");
  c2->Print("testFit.gif");
  c2->Print("testFit.C");
  c2->Print("testFit.root");

```

```
  TCanvas *c3 = new TCanvas("c3", "TF1 examples", 200, 10, 600, 400);
  c3->Divide(1, 2);
  for (Int_t i = 0; i < 2; i++) {
    c3->cd(i + 1);
    h[i]->Draw("H");
    h[i]->Draw("E,P,SAME");
  }
  c3->Print("testHisto.gif");
  c3->Print("testHisto.C");
  c3->Print("testHisto.root");

  /*----Writing out fit results ---*/

  TF1 *fitFunc = h[1]->GetFunction("f");
  Double_t mean = fitFunc->GetParameter(1);
  Double_t meanErr = fitFunc->GetParError(1);
  Double_t sigma = fitFunc->GetParameter(2);
  Double_t sigmaErr = fitFunc->GetParError(2);
  Double_t chiSquare = fitFunc->GetChisquare();
  Double_t nDOF = fitFunc->GetNDF();
  Double_t Prob = fitFunc->GetProb();
  cout << "Mean = " << mean << " +/- " << meanErr << endl;
  cout << "Sigma = " << sigma << " +/- " << sigmaErr << endl;
  cout << "ChiSquare = " << chiSquare << " , nDOF " << nDOF << endl;
  cout << "ChiSquare Probability= " << Prob << endl;

  // Covariance and Correlation matrices

  TMatrixD cov = fRes->GetCovarianceMatrix();
  TMatrixD cor = fRes->GetCorrelationMatrix();
  cov.Print();
  cor.Print();
}
```

4.2 TList

```
void setStyle() {
  gROOT->SetStyle("Plain");
  gStyle->SetPalette(57);
  gStyle->SetOptTitle(0);
}
// Gaussiana 1-D definita nella funzione utente
Double_t myFunction(Double_t *x, Double_t *par) {
  Double_t xx = x[0];
  Double_t val = par[0] * TMath::Exp(-(xx - par[1]) * (xx - par[1]) / 2. /
                                     par[2] / par[2]);
  return val;
}
// Gaussiana 2-D definita nella funzione utente
Double_t myFunction2(Double_t *x, Double_t *par) {
  Double_t xx = x[0];
  Double_t yy = x[1];
  Double_t val =
      par[0] *
      TMath::Exp(-(xx - par[1]) * (xx - par[1]) / 2. / par[2] / par[2]) *
      TMath::Exp(-(yy - par[1]) * (yy - par[1]) / 2. / par[2] / par[2]);
  return val;
}
void testList(Int_t nGen = 1E5) {
  gStyle->SetOptStat(2210);
  gStyle->SetOptFit(1111);
  TH1 *h[3];
  TString histName = "h";
  for (Int_t i = 0; i < 3; i++) {
    if (i < 2)
      h[i] = new TH1F(histName + i, "test histogram", 100, -5, 5.);
    else
      h[2] = new TH2F(histName + i, "test histogram", 100, -5, 5., 100, -5, 5.);
    // cosmetics
    h[i]->SetMarkerStyle(20);
    h[i]->SetMarkerSize(0.5);
    h[i]->SetLineColor(1);
    h[i]->GetYaxis()->SetTitleOffset(1.2);
```

```
    h[i]->GetXaxis()->SetTitleSize(0.04);
    h[i]->GetYaxis()->SetTitleSize(0.04);
    h[i]->GetXaxis()->SetTitle("x");
    h[i]->GetYaxis()->SetTitle("Entries");
  }
  h[0]->SetFillColor(kBlue);
  h[1]->SetFillColor(kRed);

  // filling the histos with FillRandom

  h[0]->FillRandom("gaus",
                   nGen);  // fill the first histo with gaus predefined function

  TF1 *f = new TF1("f", myFunction, -10, 10, 3);  // user defined function
  f->SetParameter(0, 1);
  f->SetParameter(1, 0);
  f->SetParameter(2, 1);
  h[1]->FillRandom("f", nGen);  // Fill the second histo with f

  TF2 *f2 =
      new TF2("f2", myFunction2, -10, 10, -10, 10, 3);  // user defined function
  f2->SetParameter(0, 1);
  f2->SetParameter(1, 0);
  f2->SetParameter(2, 1);
  h[2]->FillRandom("f2", nGen);  // Fill the third histo with f2

  // The TGraphErrors
  // The values and the errors on the X and Y axis
  const int n_points = 10;
  double x_vals[n_points] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  double y_vals[n_points] = {6, 12, 14, 20, 22, 24, 35, 45, 44, 53};
  double y_errs[n_points] = {5, 5, 4.7, 4.5, 4.2, 5.1, 2.9, 4.1, 4.8, 5.43};
  double x_errs[n_points] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
  // Instance of the graph
  TGraphErrors *graph = new TGraphErrors(n_points, x_vals, y_vals, 0, y_errs);
  graph->SetTitle("Example Graph; X (a.u.); Y(a.u.)");

  // Filling The TList
  TList *testList = new TList();
  for (Int_t i = 0; i < 3; i++) testList->Add(h[i]);
  testList->Add(graph);
  testList->Print();

  // drawing
  TCanvas *c1 = new TCanvas("c1", "TList example", 200, 10, 600, 400);
  c1->Divide(2, 2);
  for (Int_t i = 0; i < 4; i++) {
    c1->cd(i + 1);
    if (!testList->At(i)->InheritsFrom("TGraph")) {  // RTTI feature of ROOT
      if (!testList->At(i)->InheritsFrom("TH2")) {
        testList->At(i)->Draw("H");
        testList->At(i)->Draw("E,P,SAME");
      } else
        testList->At(i)->Draw("SURF1");
    } else
      testList->At(i)->Draw("APE");
  }
  gDirectory->ls();
  TFile *file = new TFile("testList.root", "recreate");
  gDirectory->ls();
  file->Write();
  file->Close();
}
```

4.3 Benchmark

```
#include "TBenchmark.h"
#include "TH1F.h"
#include "TRandom.h"
#include "TStyle.h"

void benchmark(Int_t nGen) {
  gStyle->SetOptStat(111111);
```

```
  TString histName = "h";
  TH1F *h[2];
  for (Int_t i = 0; i < 2; i++) {
    h[i] = new TH1F(histName + i, "test histogram", 1000, -5, 5.);
    // cosmetics
    h[i]->SetMarkerStyle(20);
    h[i]->SetMarkerSize(0.5);
    h[i]->SetLineColor(1);
    h[i]->GetYaxis()->SetTitleOffset(1.5);
    h[i]->GetXaxis()->SetTitle("x");
    h[i]->GetYaxis()->SetTitle("Entries");
  }

  h[0]->SetFillColor(4);
  h[1]->SetFillColor(2);
  TBenchmark *b = new TBenchmark();
  // filling the histo
  b->Start("With TH1::FillRandom");

  // with numerical inverse transform
  h[0]->FillRandom("gaus", nGen);
  // stop and show benchmark
  b->Show("With TH1::FillRandom");

  // with inverse transform
  Double_t x = 0;
  b->Start("Direct TRandom::Gauss invocation");
  for (Int_t i = 0; i < nGen; i++) {
    x = gRandom->Gaus(0, 1);
    h[1]->Fill(x);
  }
  // Stop and show benchmark
  b->Show("Direct TRandom::Gauss invocation");
}
```

4.4 Verifica di correlazione

```
void testCorrelation(Int_t nGen) {
  gStyle->SetOptStat(0);
  TH2F *h = new TH2F("h", "", 1000, 0, TMath::Pi(), 1000, 0, 2 * TMath::Pi());
  TH3F *hSpace = new TH3F("hSpace", "", 100, -1, 1, 100, -1, 1, 100, -1, 1);
  // cosmetics
  h->SetLineColor(1);
  h->GetYaxis()->SetTitleOffset(1.2);
  h->GetXaxis()->SetTitleSize(0.04);
  h->GetYaxis()->SetTitleSize(0.04);
  h->GetXaxis()->SetTitle("#theta (rad)");
  h->GetYaxis()->SetTitle("#phi (rad)");

  Double_t phi, theta = 0;

  for (Int_t i = 0; i < nGen; i++) {
    theta = gRandom->Rndm() * TMath::Pi();
    // phi=gRandom->Rndm()*2*TMath::Pi();
    phi = 2 * theta;  // not ok
    h->Fill(theta, phi);
    hSpace->Fill(sin(theta) * cos(phi), sin(theta) * sin(phi), cos(theta));
  }

  TCanvas *c1 = new TCanvas("c1", "theta-phi correlation", 200, 10, 600, 400);
  h->Draw();

  TCanvas *c2 = new TCanvas("c2", "xyz distribution", 200, 10, 600, 400);
  hSpace->Draw();
}
```

4.5 Efficienza

```
Double_t effErf(double *x, double *p) {
  // p[0]==media
  // p[1]==sigma
  // p[2]==valore di saturazione
  // offset 1 e fattore 1/2 per riscalare Erf in [0,1]
```

```
6
7   return (TMath::Erf((x[0] - p[0]) / p[1]) + 1) / 2. * p[2];
8 }
9
10 void efficiency(Int_t nGen = 1E6) {
11   gRandom->SetSeed();
12   cout << "random generation seed: " << gRandom->GetSeed() << endl;
13   TH1F *h[2];
14   TString histName = "h";
15   TString title[2] = {"generated distribution", "observed distribution"};
16   for (Int_t i = 0; i < 2; i++) {
17     h[i] = new TH1F(histName + i, title[i], 100, 0, 10);
18     // cosmetics
19     h[i]->SetLineColor(1);
20     h[i]->GetYaxis()->SetTitleOffset(1.2);
21     h[i]->GetXaxis()->SetTitleSize(0.04);
22     h[i]->GetYaxis()->SetTitleSize(0.04);
23     h[i]->GetXaxis()->SetTitle("x");
24     h[i]->GetYaxis()->SetTitle("Entries");
25     h[i]->Sumw2();  // Important
26   }
27
28   h[0]->SetFillColor(4);
29   h[1]->SetFillColor(2);
30
31   // The efficiency profile
32   TF1 *myErf = new TF1("myErf", effErf, 0, 10., 3);
33   myErf->SetParameter(0, 5.);   // flexus
34   myErf->SetParameter(1, 0.5);  // width
35   myErf->SetParameter(2, 0.7);  // saturation value
36   TCanvas *cFunc =
37       new TCanvas("cFunc", "Efficiency Profile", 200, 10, 600, 400);
38   myErf->SetLineColor(1);
39   myErf->SetMaximum(1);
40   myErf->Draw();
41
42   // case: uniform distribution
43   Double_t x = 0, xRNDM = 0;
44
45   for (Int_t i = 0; i < nGen; i++) {
46     // x=gRandom->Uniform(0,10);
47     x = gRandom->Exp(1);  // negative exponential with mu=1
48     h[0]->Fill(x);        // generated
49     xRNDM = gRandom->Rndm();
50     if (xRNDM < myErf->Eval(x)) h[1]->Fill(x);  // observed
51   }
52
53   TCanvas *cHisto =
54       new TCanvas("cHisto", "Efficiency effects, Generated and Observed ", 200,
55                   10, 600, 400);
56   cHisto->Divide(1, 2);
57   for (Int_t i = 0; i < 2; i++) {
58     cHisto->cd(i + 1);
59     h[i]->SetMinimum(0);
60     h[i]->Draw("H");
61     h[i]->Draw("E,SAME");
62   }
63
64   TH1F *hEff = new TH1F(*h[0]);
65   hEff->SetTitle("Observed Efficiency");
66   hEff->SetName("hEff");
67   hEff->SetFillColor(3);
68   hEff->Divide(h[1], h[0], 1, 1, "B");
69   TCanvas *cEff = new TCanvas("cEff", "Observed Efficiency", 200, 10, 600, 400);
70   hEff->SetMaximum(1.);
71   hEff->Draw("H");
72   hEff->Draw("E,SAME");
73   cout << "Integral efficiency =" << h[1]->Integral() / h[0]->Integral()
74        << endl;
75 }
```

4.6 Proporzioni

```
void proportions(Int_t nGen) {
  TH1F *h = new TH1F("h", "abundancies", 5, 0, 5);
  // cosmetics
  h->SetLineColor(1);
  h->GetYaxis()->SetTitleOffset(1.2);
  h->GetXaxis()->SetTitleSize(0.04);
  h->GetYaxis()->SetTitleSize(0.04);
  h->GetXaxis()->SetTitle("Cases");
  h->GetYaxis()->SetTitle("Entries");
  h->SetFillColor(4);
  Double_t x = 0;

  for (Int_t i = 0; i < nGen; i++) {
    x = gRandom->Rndm();
    if (x < 0.1)
      h->Fill(0);
    else if (x < 0.3)
      h->Fill(1);
    else if (x < 0.7)
      h->Fill(2);
    else if (x < 0.95)
      h->Fill(3);
    else
      h->Fill(4);
  }
  TCanvas *c1 =
      new TCanvas("c1", "Generate proportions, Example", 200, 10, 600, 400);
  h->Draw("H");
  h->Draw("E,SAME");
}
```

4.7 Risoluzione

```
void resolution(Double_t res = 0.1, Int_t nGen = 1E6) {
  TString histName = "h";
  TH1F *h[2];
  for (Int_t i = 0; i < 2; i++) {
    h[i] = new TH1F(histName + i, "test histogram", 90, 0, 3);
    // cosmetics
    h[i]->SetLineColor(1);
    h[i]->GetYaxis()->SetTitleOffset(1.2);
    h[i]->GetXaxis()->SetTitleSize(0.04);
    h[i]->GetYaxis()->SetTitleSize(0.04);
    h[i]->GetXaxis()->SetTitle("x after Resolution Effect");
    h[i]->GetYaxis()->SetTitle("Entries");
  }

  h[0]->SetFillColor(4);
  h[1]->SetFillColor(2);

  // first case: fixed value smeared
  Double_t fixedValue = 1.5;
  for (Int_t i = 0; i < nGen; i++) h[0]->Fill(gRandom->Gaus(fixedValue, res));
  // second case: Uniform distribution smeared
  for (Int_t i = 0; i < nGen; i++)
    h[1]->Fill(gRandom->Gaus(gRandom->Uniform(1, 2), res));

  TCanvas *c1 =
      new TCanvas("c1", "Resolution Effects, Examples", 200, 10, 600, 400);
  c1->Divide(1, 2);
  for (Int_t i = 0; i < 2; i++) {
    c1->cd(i + 1);
    h[i]->Draw("H");
    h[i]->Draw("E,SAME");
  }
}
```

4.8 Scrivere su TTree

```
void makeTree() {
  TFile *file = new TFile("testTree.root", "recreate");  // opening the ROOT
                                                         // file
  TTree *T = new TTree("T", "test tree");  // creating the Tree
```

```
5
6   Float_t x, y, z;  // local variables
7
8   // Defining the branches
9   T->Branch("x", &x, "x/F");
10  T->Branch("y", &y, "y/F");
11  T->Branch("z", &z, "z/F");
12
13  for (Int_t i = 0; i < 100000; i++) {
14    x = gRandom->Gaus(3, 2);   // gaussian
15    y = gRandom->Gaus(10, 3);  // gaussian
16    z = gRandom->Exp(3.);      // expo
17    T->Fill();                 // filling the Tree
18  }
19
20  T->Write();     // writing the Tree on the ROOT file
21  file->Close();  // closing the ROOT file
22 }
```

4.9 Leggere da TTree

```
1  TCanvas *CreateC(Int_t i) {
2    TString cName = "c";
3    TString cTitle = "Tree examples ";
4    TCanvas *c;
5    c = new TCanvas(cName + i, cTitle + i, 200, 10, 600, 400);
6    return c;
7  }
8
9  void readTree() {
10   // TH1::AddDirectory(kFALSE);
11
12   TCanvas *c[10];
13
14   TFile *file = new TFile("testTree.root");  // opening the root file
15   file->ls();                                // listing the file content
16
17   TTree *Tout = (TTree *)file->Get("T");  // getting the Tree
18
19   Tout->Print();  // listing the Tree content
20
21   Int_t nentries = Tout->GetEntries();  // number of entries in the Tree
22   cout << " nentries in tree = " << nentries << endl;
23
24   // Drawing the Tree variables (automatic loop on Tree entries done by ROOT)
25
26   int i = 0;
27
28   CreateC(i)->cd();
29   i++;
30   Tout->Draw("x");  // Drawing x looping over all entries, on a temporary histo
31
32   CreateC(i)->cd();
33   i++;
34   Tout->Draw("x", "", "", 1000,
35              nentries - 1000);  // same as above, but for the last 1000 entries
36
37   CreateC(i)->cd();
38   i++;
39   Tout->Draw(
40       "y:x");  // Drawing y vs x, looping over all entries, on a temporary histo
41
42   TH1F *h1 = new TH1F("h1", "x distribution", 200, -10., 10.);
43   CreateC(i)->cd();
44   i++;
45   Tout->Draw("x>>h1", "y>10.");  // Drawing x over a predefined histo h1 and
46                                  // apply a selection on y.
47
48   CreateC(i)->cd();
49   i++;
50   Tout->Draw("sqrt(x**2+y**2)", "log(y)>1.");  // you may use functions
51
52   CreateC(i)->cd();
```

```
  i++;
  Tout->Draw("sqrt(x**2+y**2):log(z)",
             "y>1 && x<0");  // correlation plot with selection

  // in this mode, the user loops explicitly over the tree and can
  // recover/select each of the entries

  Float_t x, y, z;

  // connecting local variables x,y,z to the tree variables

  Tout->SetBranchAddress("x", &x);
  Tout->SetBranchAddress("y", &y);
  Tout->SetBranchAddress("z", &z);

  for (Int_t i = 0; i < nentries; i++) {
    if (i % 10000 == 0) {
      Tout->GetEntry(i);
      cout << " x = " << x << " y = " << y << " z = " << z << endl;
    }
  }
  file->Close();  // closing the file
}
```