

Laboratorio II – 1° modulo

Lezione 10

Esercitazione sul fit di istogrammi

Indice

- Svolgimento guidato di un esercizio “standard” di fit di un istogramma:
 - Scelta del *binning* dell’istogramma
 - Definizione della funzione di fit e inizializzazione dei suoi parametri
- Correlazione dei parametri di fit
- Metodi `GetBinContent()` / `SetBinContent()` per leggere / modificare il contenuto dei bin di un istogramma
- Esercizi

Esercizio di fit

- Riprendiamo il file `data1.txt` già utilizzato per l'esercizio 1 della scorsa lezione
- Immaginiamo di non sapere né quanti dati contiene, né quali siano i valori minimo e massimo
- Vogliamo inserire i dati in un istogramma con binning opportuno per rappresentare la distribuzione dei dati
- Infine vogliamo fittare l'istogramma con una funzione data dalla somma di una Normale e di una parabola e accedere ai risultati del fit

Impostazione codice

```
// c++ -o FitHisto FitHisto.cpp `root-config --glibs --cflags`
```

```
// ./FitHisto data1.txt
```

```
#include <iostream>
#include <fstream>
#include <cmath>
```

```
#include <TApplication.h>
#include <TCanvas.h>
#include <TH1.h>
#include <TF1.h>
#include <TStyle.h>
```

```
using namespace std;
```

```
int main (int argc, char **argv) {
    TApplication *myApp = new TApplication("app", 0, 0);
```

```
// qui scrivo il codice dell'esercizio
```

```
myApp -> Run();
return 0;
```

```
}
```

• Compilazione ed esecuzione

• Librerie C++

Librerie ROOT:

- **TH1** è la classe degli istogrammi 1-dim: comprende sia TH1F, sia TH1D, a seconda che vogliamo lavorare con precisione float o double
- **TF1** è la classe delle funzioni a una variabile

Con il metodo Run() della classe TApplication viene lanciata l'esecuzione del codice ROOT che gestisce le finestre. **N.B.:** dovete istanziare una ed una sola variabile TApplication. Anche se istanziate più canvas (cioè più finestre) myApp è in grado di gestirli tutti

Lettura file di dati

```
int main (int argc, char **argv) {  
    TApplication *myApp = new TApplication("app", 0, 0);
```

```
    ifstream in (argv[1]);  
    if (in.good()==false) {  
        cout << "Errore apertura file" << endl;  
        return 1;  
    }
```

```
    double num, min, max;  
    vector<double> ListNum;  
    int Ndata=0;  
    while (true) {  
        in>>num;  
        if (in.eof()==true) break;  
        ListNum.push_back(num);  
        if (Ndata == 0) {  
            min=num;  
            max=num;  
        }  
        else {  
            if (num<min) min=num;  
            if (num>max) max=num;  
        }  
        Ndata++;  
    }  
    cout << "Il file " << argv[1] << " contiene " << Ndata << " dati" << endl;  
    cout << "Minimo: " << min << endl;  
    cout << "Massimo: " << max << endl;
```

• Costruisco l'oggetto `in` della classe `ifstream` per aprire il file il cui nome (`argv[1]`) è passato da riga di comando

Leggo il file, salvo i numeri in esso contenuti in un `vector<double>` e scrivo un semplice algoritmo per cercare i numeri minimo e massimo, da utilizzare successivamente per costruire un istogramma

N.B.: questo passaggio può essere omissso se si conoscono a priori min e max. In questo caso, si può definire l'istogramma prima del ciclo di lettura e riempirlo direttamente (senza necessità di salvare i dati in un `vector`)

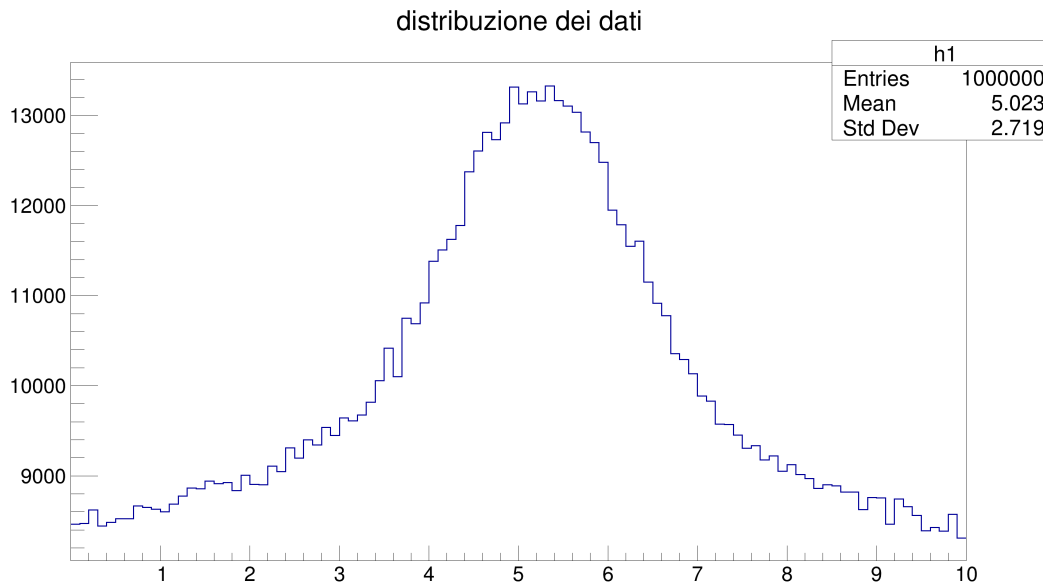
Costruzione istogramma

Imposto il numero di bin
dell'istogramma

Uso i valori ottenuti per `min` e
`max` per impostare il range
dell'istogramma

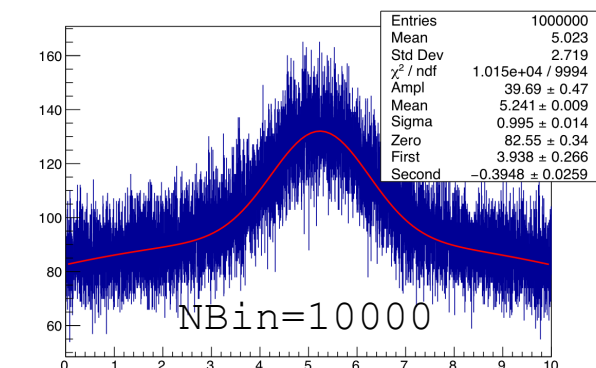
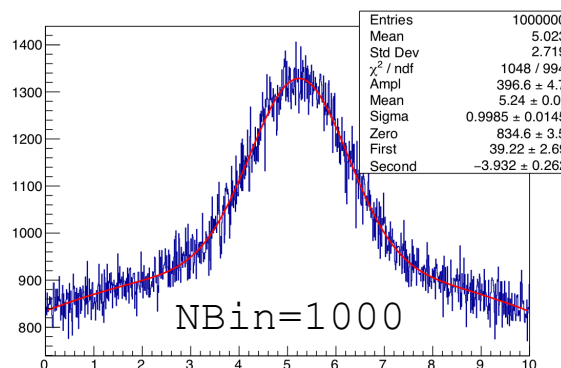
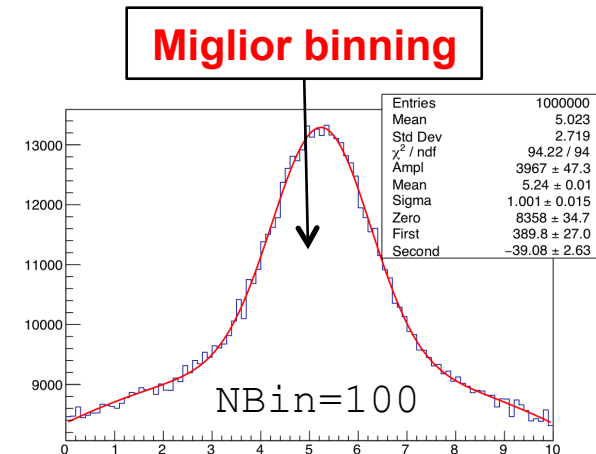
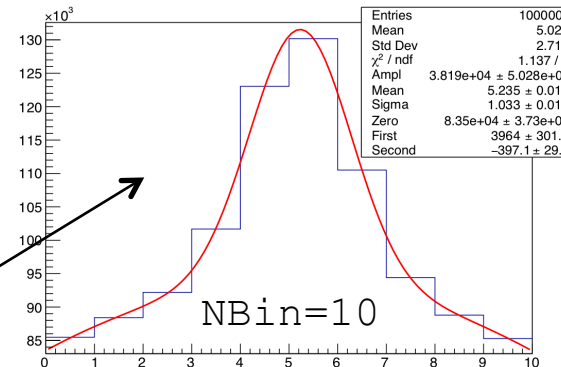
```
int NBin = 100;  
TH1F *h1 = new TH1F ("h1", "distribuzione dei dati", NBin, min, max);  
for (int i=0; i<Ndata; i++) {  
    h1->Fill(ListNum[i]);  
}  
TCanvas *c1 = new TCanvas ();  
h1->Draw();
```

Riempio l'istogramma con i numeri salvati
nel `vector<double> ListNum`



Scelta del binning

- Al fine di utilizzare l'approssimazione Normale della statistica binomiale del bin è necessario che il numero di conteggi per bin sia maggiore di **9** → **limite inferiore della larghezza del bin** (l'approssimazione Normale è desiderabile perché ci permette di giustificare il test del Chi-2 di bontà del fit)
- I seguenti istogrammi, con binning 10, 100, 1000 e 10000, forniscono un **errore percentuale sui parametri molto simile** ma il valore del Chi-2 ridotto è molto diverso da 1 nel primo caso (per gli altri si stabilizza a ~ 1) → è preferibile un binning dopo il quale il Chi-2 ridotto si è stabilizzato → **limite superiore della larghezza del bin**

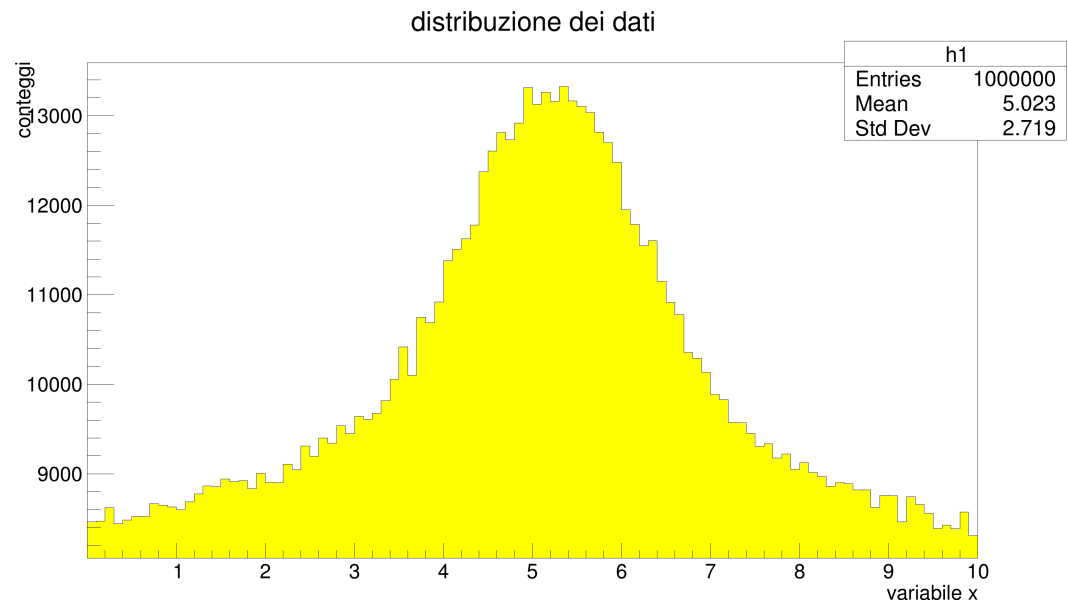
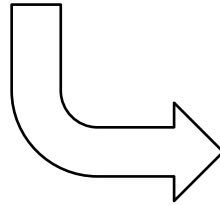


Miglior binning

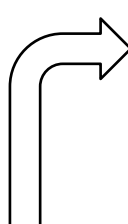
Opzioni grafiche

- Esistono opportuni metodi per inserire i titoli degli assi, cambiare il colore dell'istogramma, della sua linea, etc ...
- È anche possibile effettuare modifiche in modalità interattiva, cliccando sui diversi oggetti all'interno della finestra del canvas

```
h1->SetFillColor(kYellow);  
h1->GetXaxis() -> SetTitle("variabile x");  
h1->GetYaxis() -> SetTitle("conteggi");
```



Definizione funzione di fit



```
double fitfunc (double *x, double *p) {  
    double arg = (x[0]-p[1])/p[2];  
    double gaus = p[0]*exp(-0.5*arg*arg);  
    double parab = p[3] + p[4]*x[0] + p[5]*x[0]*x[0];  
    return gaus + parab;  
}
```

Funzione esterna al `main()`, utilizzata per definire una TF1 di ROOT

È necessario che abbia due puntatori come parametri di input:

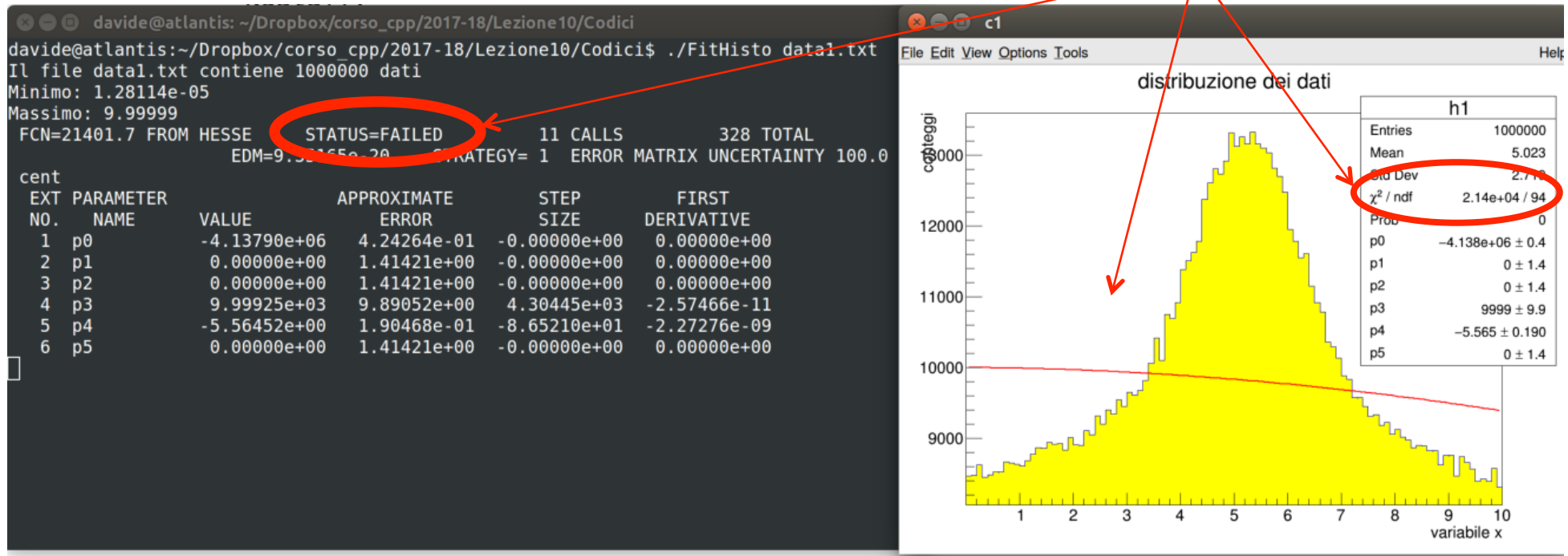
- il primo punta all'array delle variabili indipendenti (se la funzione è a una sola variabile utilizzeremo solo `x[0]`)
- il secondo punta all'array dei parametri della funzione

Nel `main()` costruisco la TF1 di ROOT che uso successivamente per fittare l'istogramma

```
int nPar = 6;  
TF1 *f1 = new TF1 ("myFunz", fitfunc, min, max, nPar);
```

Fit dell'istogramma

- Se mi dimentico di inizializzare i parametri della funzione di fit (o non li inizializzo correttamente), è molto probabile che il fit non converga



```
gStyle -> SetOptFit(1111);
int nPar = 6;
TF1 *f1 = new TF1 ("myFunz", fitfunc, min, max, nPar);
h1->Fit("myFunz");
```

Istruzione per far comparire i risultati del fit nella legenda dell'istogramma

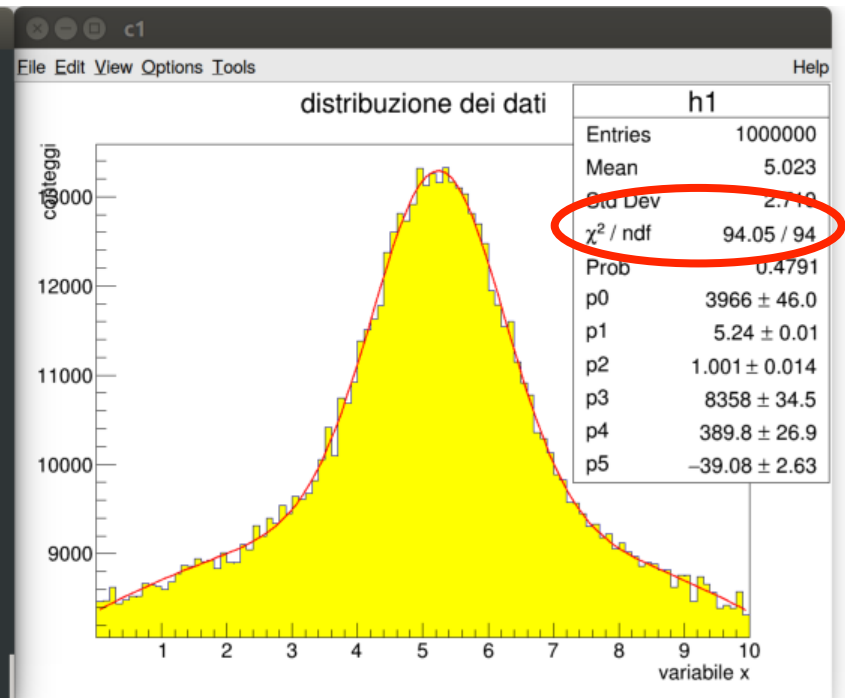
Fit dell'istogramma

```
gStyle -> SetOptFit(1111);
int nPar = 6;
TF1 *f1 = new TF1 ("myFunz", fitfunc, min, max, nPar);
f1->SetParameter(0, 10000); // normalizzazione gaussiana (valore massimo)
f1->SetParameter(1, 5.); // valor medio della gaussiana
f1->SetParameter(2, h1->GetRMS() );
f1->SetParameter(3, 8500.); // intercetta della parabola

h1->Fit("myFunz");
```

```
davide@atlantis: ~/Dropbox/corso_cpp/2017-18/Lezione10/Codici
davide@atlantis:~/Dropbox/corso_cpp/2017-18/Lezione10/Codici$ ./FitHisto data1.txt
Il file data1.txt contiene 1000000 dati
Minimo: 1.28114e-05
Massimo: 9.99999
FCN=94.0501 FROM MIGRAD STATUS=CONVERGED 265 CALLS 266 TOTAL
EDM=8.1615e-09 STRATEGY= 1 ERROR MATRIX UNCERTAINTY 2.3 per
cent
```

EXT NO.	PARAMETER NAME	VALUE	ERROR	STEP SIZE	FIRST DERIVATIVE
1	p0	3.96637e+03	4.59866e+01	2.46019e-02	1.99321e-06
2	p1	5.23976e+00	9.40567e-03	-1.45835e-05	-2.03164e-03
3	p2	1.00097e+00	1.44725e-02	-4.23990e-06	2.16079e-02
4	p3	8.35751e+03	3.44881e+01	-6.83774e-03	9.13140e-06
5	p4	3.89821e+02	2.69152e+01	-5.91232e-03	6.03957e-05
6	p5	-3.90841e+01	2.62760e+00	9.03438e-04	3.93460e-04



Get/SetParameter di TF1

```
double a = f1->GetParameter(5);
double b = f1->GetParameter(4);
double c = f1->GetParameter(3);
cout << "Parametri della parabola: " << endl;
cout << " a = " << a << " +- " << f1->GetParError(5) << endl;
cout << " b = " << b << " +- " << f1->GetParError(4) << endl;
cout << " c = " << c << " +- " << f1->GetParError(3) << endl;
```

Metodi **GetParameter** e **GetParError** della classe TF1 per accedere ai valori dei parametri (e loro incertezze)

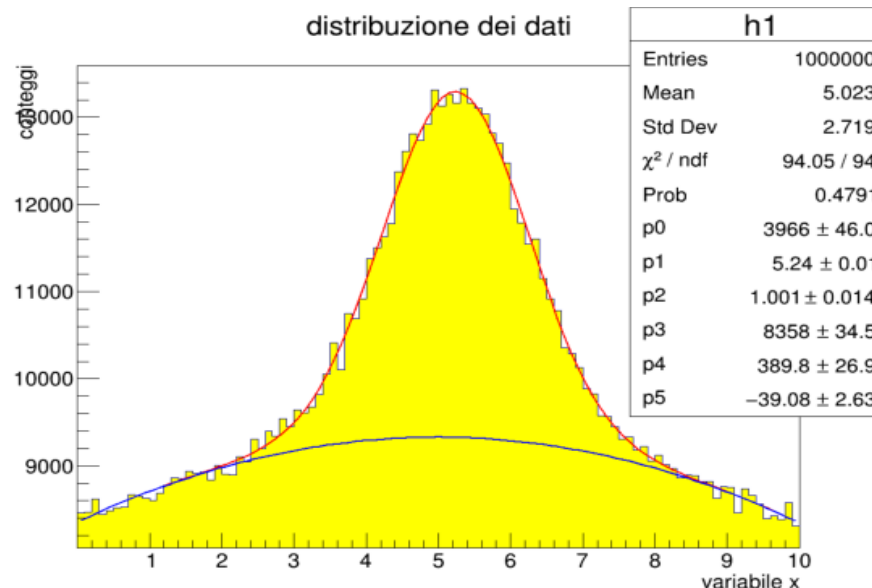
```
Parametri della parabola:
a = -39.0841 +- 2.6276
b = 389.821 +- 26.9152
c = 8357.51 +- 34.4881
```

```
TF1 *f2 = new TF1 ("parabola","pol2", min, max);
f2->SetParameter(0, c);
f2->SetParameter(1, b);
f2->SetParameter(2, a);
```

Definisco una nuova TF1 (usando la funzione predefinita pol2), per rappresentare la parabola ottenuta dal fit, separando così la componente Normale

```
f2->SetLineColor(kBlue);
f2->Draw("same");
```

Opzione grafica **"same"** per rappresentare la funzione f2 sullo stesso canvas, senza cancellare il grafico precedente



Correlazioni tra parametri

- Quando si esegue un fit con una funzione a più parametri, i valori stimati per i **parametri** possono essere tra loro **correlati**
- Per consentire di accedere a **tutti** i risultati del fit (compresi i coefficienti di correlazione dei parametri), la funzione di fit, se invocata specificando l'opzione "S", restituisce in output un oggetto della classe `TFitResultPtr`, che si comporta come un puntatore ai risultati del fit

Come si usa il `TFitResultPtr`?

- Includere la classe `TFitResult.h`: **`#include <TFitResult.h>`**
- Eseguire il fit con opzione "S", memorizzando l'output in un `TFitResultPtr`:

```
TFitResultPtr r = h1 -> Fit("myFunz", "S");
```

- Posso stampare a schermo i risultati completi del fit (compresa la matrice di covarianza):

```
r -> Print("V");
```

Correlazioni tra parametri

```
FCN=94.0501 FROM MIGRAD STATUS=CONVERGED 78 CALLS 79 TOTAL
EDM=6.90892e-09 STRATEGY= 1 ERROR MATRIX ACCURATE

EXT  PARAMETER      STEP      FIRST
NO.   NAME        VALUE      ERROR    SIZE    DERIVATIVE
  1   p0          3.96637e+03  4.73271e+01  1.26289e-01 -1.23502e-06
  2   p1          5.23976e+00  9.41762e-03  4.29449e-05 -2.50640e-03
  3   p2          1.00097e+00  1.45487e-02  3.40179e-05  6.44695e-03
  4   p3          8.35751e+03  3.47295e+01  4.70831e-02 -2.68839e-06
  5   p4          3.89821e+02  2.70086e+01  8.07215e-03 -9.57404e-06
  6   p5          -3.90841e+01  2.63508e+00  1.01806e-03 -9.60480e-05
```

Minimizer is Minuit / Migrad

```
Chi2      = 94.0501
Ndf       = 94
Edm       = 6.90892e-09
NCalls    = 79
p0        = 3966.37 +/- 47.3271
p1        = 5.23976 +/- 0.00941762
p2        = 1.00097 +/- 0.0145487
p3        = 8357.51 +/- 34.7295
p4        = 389.821 +/- 27.0086
p5        = -39.0841 +/- 2.63508
```

Covariance Matrix:

	p0	p1	p2	p3	p4	p5
p0	2239.9	-0.018222	0.21521	801.93	-886.25	86.793
p1	-0.018222	8.8692e-05	-5.1896e-06	0.024554	0.0084021	-0.001705
p2	0.21521	-5.1896e-06	0.00021167	0.22067	-0.27742	0.02706
p3	801.93	0.024554	0.22067	1206.1	-791.9	72.492
p4	-886.25	0.0084021	-0.27742	-791.9	729.47	-70.602
p5	86.793	-0.001705	0.02706	72.492	-70.602	6.9437

Correlation Matrix:

	p0	p1	p2	p3	p4	p5
p0	1	-0.040883	0.31256	0.4879	-0.69334	0.69595
p1	-0.040883	1	-0.037876	0.075073	0.033033	-0.068707
p2	0.31256	-0.037876	1	0.43674	-0.706	0.70584
p3	0.4879	0.075073	0.43674	1	-0.84425	0.79213
p4	-0.69334	0.033033	-0.706	-0.84425	1	-0.99203
p5	0.69595	-0.068707	0.70584	0.79213	-0.99203	1

```
TFitResultPtr r = h1->Fit("myFunz", "S");
r->Print("V");
```


Correlazioni tra parametri

- Se ho bisogno di recuperare all'interno del mio programma i coefficienti di correlazione (ad esempio per effettuare calcoli di propagazione delle incertezze), è possibile salvare la matrice di correlazione in un oggetto di ROOT della classe delle matrici simmetriche (TMatrixDSym)
- Includere la classe TMatrixDSym.h: `#include <TMatrixDSym.h>`
- Usare il metodo seguente per ottenere la matrice di covarianza a partire dal TFitResultPtr r:

```
TMatrixDSym cov = r -> GetCovarianceMatrix();
```

Posso accedere a ciascuno degli elementi della matrice di covarianza grazie all'overloading dell'operator(int row, int col):

```
double sigma_ij = cov(i,j);
```

```
TMatrixDSym cov = r->GetCovarianceMatrix();  
cout << "\nMatrice di covarianza " << endl;  
for (int i=0; i<nPar; i++) {  
    for (int j=0; j<nPar; j++) {  
        double sigma_ij = cov(i,j);  
        cout << setw(15) << sigma_ij;  
    }  
    cout << endl;  
}
```

Ancora sui TH1: metodi Get

La classe degli istogrammi di ROOT mette a disposizione tutta una serie di **metodi di tipo Get** per **accedere** agli **attributi** dell'istogramma:

- `double GetBinContent (int i):` restituisce il numero di conteggi dell'i-esimo bin
- `double GetBinError (int i):` restituisce l'incertezza associata ai conteggi dell'i-esimo bin (di default è pari alla radice quadrata dei conteggi)
- `double GetBinCenter(int i):` restituisce il valore dell'ascissa su cui è centrato il bin
- `double GetBinWidth(int i):` restituisce la larghezza (o step) del bin i-esimo
- `int GetNbinsX():` restituisce il numero di bin dell'istogramma
- etc...

ATTENZIONE: a differenza degli array, per i quali il primo elemento ha indice zero, gli istogrammi di ROOT seguono la seguente convenzione

- `bin = 0;` underflow bin
- `bin = 1;` first bin with low-edge xlow INCLUDED
- `bin = nbins;` last bin with upper-edge xup EXCLUDED
- `bin = nbins+1;` overflow bin

Fate attenzione all'uso degli indici: con gli istogrammi bisogna partire a contare da **1** invece che da **0**

Ancora sui TH1: metodi Set

Oltre ai metodi di tipo `Get`, che non modificano il contenuto dell'istogramma, esistono anche i **metodi di tipo `Set`**, con cui è possibile **modificare** gli **attributi** di un oggetto istogramma. Ad esempio:

- `void SetBinContent (int i, double content):` imposta i conteggi dell'*i*-esimo bin al valore della variabile `content`
- `void SetBinError (int i, double error):` imposta l'incertezza dell'*i*-esimo bin al valore della variabile `error`
- etc...

Esercizio 1: Riempite, con il metodo `Fill`, un istogramma `h1` con i dati contenuti nel file `data1.txt` ed esercitatevi con i metodi `Get/Set` della classe istogramma come segue:

1. Create un nuovo istogramma `h2` con lo stesso binning di `h1`
2. Leggete in un ciclo `for` il contenuto e l'incertezza dei bin di `h1`
3. Usate i metodi di tipo `Set` per impostare il contenuto (e l'incertezza) dei bin di `h2`, in modo che coincidano con quelli di `h1`
4. Eseguite infine il fit dell'istogramma `h2` con la medesima funzione utilizzata per fittare in precedenza `h1`
5. Verificate di ottenere i medesimi risultati

Esercizio 2:

In un esperimento di fisica, si eseguono misure di momento riportate nel file `momenta.txt`

Si chiede di:

1. Leggere i dati dal file e riempire un istogramma `TH1F` con un numero congruo di bin ed in un range opportuno di impulso
2. Definire tre funzioni, a due parametri (a e b) della forma

`double dist(double *E, double *par)`

che rappresentino tre possibili distribuzioni che si vogliono testare, elencate qui sotto, e fittare l'istogramma:

- $dist_1(E) = aE^2e^{-bE}$
- $dist_2(E) = aEe^{-bE}$
- $dist_3(E) = aE^2e^{-bE^2}$

Usate l'opzione "+" del metodo `Fit` per mantenere il disegno di tutte tre le funzioni sul grafico. Es: `h1->Fit("MyFunc", "+");`
Usate `SetLineColor` della classe `TF1` per differenziare i colori delle tre funzioni di fit

Esercizi

3. Stabilire quali delle tre distribuzioni si adatta meglio ai dati utilizzando il test del Chi-2. Si usino le formule statistiche (NON il fit di ROOT) per determinare il Chi-2 ridotto:
 - calcolare il valore teorico della $\text{dist}_n(E)$ per ciascun bin (si prenda E = valore centrale del bin)
 - estrarre il valore misurato della $\text{dist}_n(E)$ per ciascun bin dell'istogramma costruito con i dati, per fare questo si utilizzi il metodo `GetBinContent(i)`
 - scrivere a schermo il valore del Chi-2 e il numero di gradi di libertà
4. Ripetere il punto precedente facendo il fit dell'istogramma con ROOT e facendosi restituire Chi-2 e numero di gradi di libertà. Confrontare i valori calcolati da voi con quelli restituiti da ROOT
5. Concludere l'esercizio scrivendo a terminale qual è la funzione che meglio si adatta ai dati sperimentali, i valori (e le incertezze) stimati per i parametri, la matrice di covarianza dei parametri e il Chi-2 ridotto