

Laboratorio II – 1° modulo

Lezione 10

Istogrammi e regressione

Laboratorio II – 1° modulo

Lezione 10

Istogrammi e regressione

Indice

- **Istogrammi e regressione**
- Svolgimento guidato di un esercizio “standard” di regressione con il metodo dei minimi quadrati:
 - Scelta del *binning* dell’istogramma
 - Funzione di fit e inizializzazione dei parametri
- Correlazione dei parametri della funzione di fit
- Metodi `GetBinContent()` / `SetBinContent()` per leggere e modificare il contenuto di un istogramma
- Esercizi

Cosa significa eseguire una regressione (i.e. fit)?

- Si esegue una regressione (o fit) quando si vuole trovare il **miglior adattamento** di un modello ad un set di dati affetto da delle incertezze
- Generalmente il **modello**, $f(x; \theta_1 \dots \theta_m)$, è una **funzione parametrica** (θ_j), con la quale si intende descrivere l'andamento dei **dati sperimentali**
- I dati sperimentali possono legare una osservabile ad un'altra, e.g. **grafico** posizione (y_i) – tempo (x_i) di un grave, oppure sono la registrazione dell'occorrenza (y_i) di una certa osservabile, e.g. **istogramma** delle altezze (x_i) di un campione di persone
- L'operazione di fit consiste nel **massimizzare** (o minimizzare) un certo **funzionale** (F). Questa operazione di **ottimizzazione** viene eseguita in **funzione dei parametri** del modello che vengono così ottimizzati per avere il miglio adattamento ai dati

$$F(\vec{\theta}; \vec{y}) \xrightarrow{\text{best } \vec{\theta}} \nabla_{\vec{\theta}} F(\vec{\theta}; \vec{y}) = 0 \quad \text{i.e.} \quad \frac{\partial F(\vec{\theta}; \vec{y})}{\partial \theta_i} = 0$$

\vec{y} vettore dei dati (i.e. misure), $\vec{\theta}$ vettore dei parametri del modello

Iistogrammi e regressione

Nel caso degli **istogrammi** i funzionali dei **minimi quadrati** e della **massima verosimiglianza** assumono la seguente forma (nell'ipotesi che i **conteggi** per ciascun bin seguano una **legge Poissoniana**)

Minimi quadrati: $\text{Min}^2(\vec{\theta}) = \sum_{i=1}^n \left(\frac{y_i - f(x_i; \vec{\theta})}{\sqrt{y_i}} \right)^2$

Questo metodo è il *default* in ROOT per fissare gli istogrammi:
e.g. `histo->Fit ("gaus")`

Massima verosimiglianza:

$$L(\vec{\theta}) = \prod_{i=1}^n \frac{f(x_i; \vec{\theta})^{y_i} e^{-f(x_i; \vec{\theta})}}{y_i!}$$

In questo caso è necessario specificarlo tramite l'opzione "**L**":
e.g. `histo->Fit ("gaus", "L")`

Istogrammi e regressione

Dove:

n: numero di misure

x_i: osservabile oggetto della misura

y_i: conteggi nel bin i-esimo

f(): modello con m parametri $\theta_1 \dots \theta_m$

Quando usare i minimi quadrati e quando la massima verosimiglianza:

- **bin con pochi conteggi (< 9)**: le ipotesi di validità del metodo dei minimi quadrati vengono meno, inoltre i bin con zero conteggi vengono ignorati → è più corretto in questi casi usare il metodo della massima verosimiglianza
- **bin con molti conteggi**: i due metodi danno lo stesso risultato, ma il metodo dei minimi quadrati è computazionalmente più semplice da risolvere, specialmente se il modello è lineare nei parametri → la minimizzazione in questi casi si riduce alla semplice inversione di una matrice

Esercizio di regressione

Il file `data1.txt` contiene i risultati di alcune misure di una certa osservabile fisica

Immaginiamo di **non** sapere né quanti dati contiene, né quali siano i valori minimo e massimo

- Si vogliono inserire i dati in un istogramma con binning opportuno per rappresentarne la distribuzione
- Infine si vuole fissare l'istogramma con una funzione data dalla somma di una Normale e di una parabola e accedere ai risultati del fit

Lettura del file di dati

```

...
double num, xMin, xMax;
vector<double> dataList;
bool first = true;
while (true)
{
    in >> num;
    if (in.eof() == true) break;
    dataList.push_back(num); ←
    if (first == true)
    {
        xMin = num;
        xMax = num;
        first = false;
    }
    else
    {
        if (num < xMin) xMin = num;
        if (num > xMax) xMax = num;
    }
}
cout << "The file " << argv[1] << " contains " << dataList.size()
    << " data" << endl;
cout << "Minimum: " << xMin << " - Maximum: " << xMax << endl;

```

Viene preventivamente costruito l'oggetto `in` della classe `ifstream` per aprire il file

Viene letto il file e i numeri in esso contenuti sono salvati in un `std::vector<double>`. Un semplice algoritmo cerca i numeri minimo e massimo da utilizzare successivamente per costruire l'istogramma.

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

Costruzione dell'istogramma

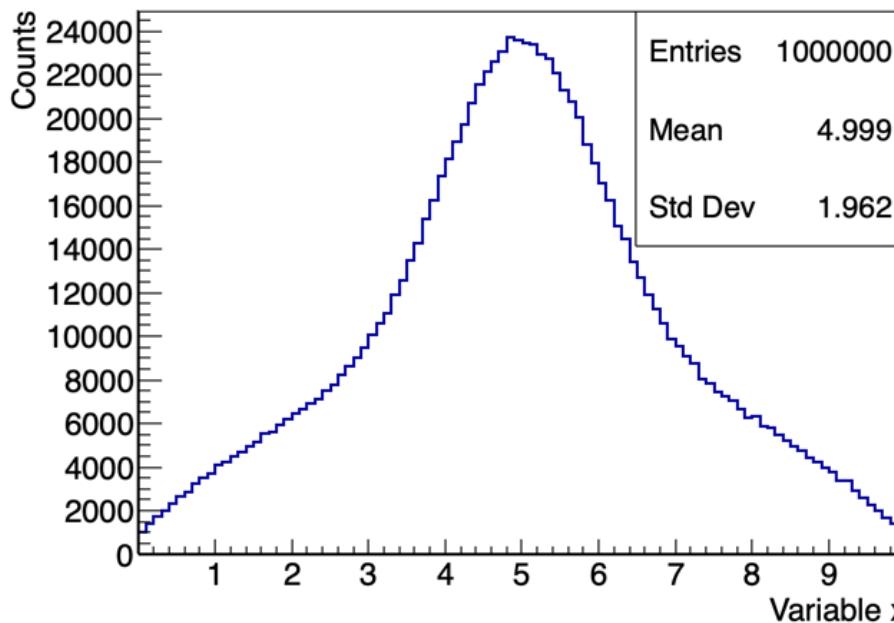
```

int nBins = 100;           ← Imposta il numero di bin dell'istogramma
TH1D* h1 = new TH1D("h1","Data distribution", nBins, xMin, xMax);
for (unsigned int i = 0; i < dataList.size(); i++)
    h1->Fill(dataList[i]);
TCanvas* myCanv1 = new TCanvas("myCanv1","myCanv1",0,0,700,500);
myCanv1->cd();
h1->Draw();

```

Vengono usati i valori di `xMin` e `xMax` per impostare il range dell'istogramma

Viene riempito l'istogramma con i numeri salvati nell'`std::vector<double> dataList`

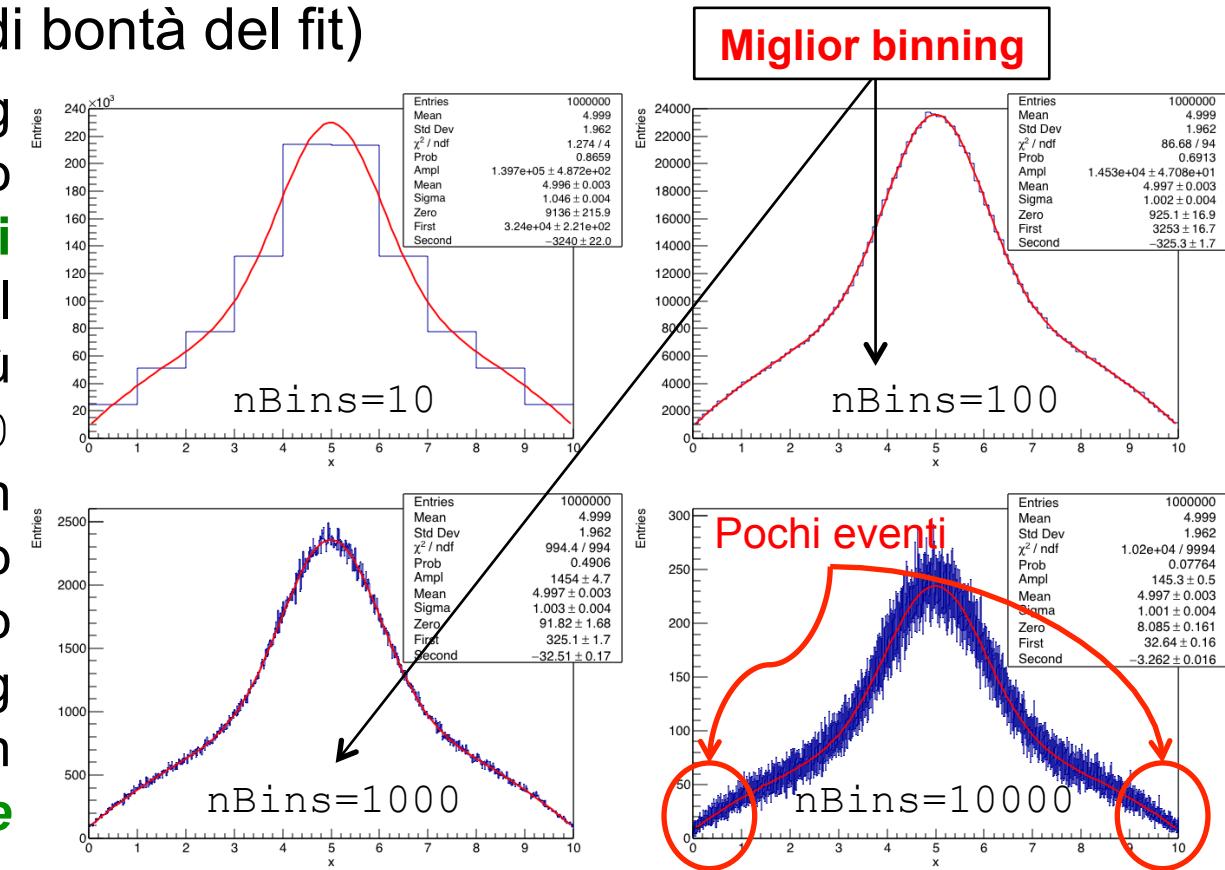


E` importante **visualizzare** l'istogramma **prima di procedere** con altre operazioni sia per rendersi conto se l'algoritmo di lettura ha operato correttamente, sia per capire se la funzione che verrà utilizzata per il fit è quella corretta e quali valori iniziali dare ai parametri (e.g. se si deve fittare una Normale guardare dove sono centrati i dati e la larghezza della distribuzione, se si deve fittare un polinomio di grado 2 e la concavità dei dati è verso il basso, inizializzare il coefficiente di x^2 ad un valore negativo, etc...)

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)

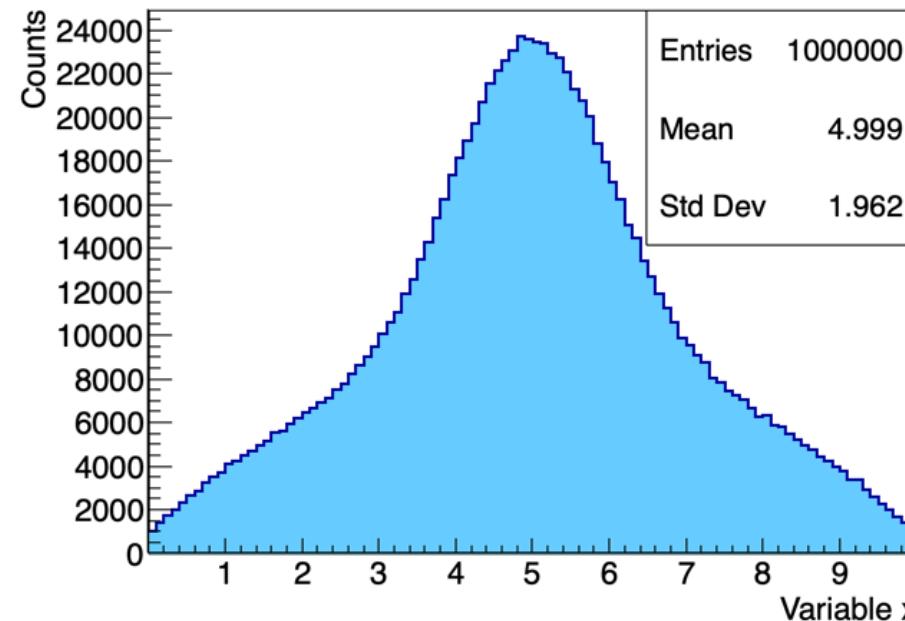
I seguenti istogrammi, con binning 10, 100, 1000 e 10000, forniscono un **errore percentuale sui parametri molto simile** ma il valore del Chi-2 ridotto risulta più prossimo a 1 nel caso nBins=100 e nBins=1000 → è preferibile un binning per il quale il Chi-2 ridotto è ~1 (nel caso in cui il Chi-2 ridotto dovesse essere simile tra binning diversi è preferibile il binning con bin più larghi) → *limite superiore della larghezza del bin*



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(kAzure+6);  
h1->GetXaxis()->SetTitle("Variable x");  
h1->GetYaxis()->SetTitle("Counts");
```



Definizione della funzione di fit

```

double myGauss (double* x, double* par) {
    return par[0] * exp(-0.5*((x[0]-par[1])*(x[0]-par[1])) /
        (par[2]*par[2])));
}

double myParabola (double* x, double* par) {
    return par[0] + par[1]*x[0] + par[2]*x[0]*x[0];
}

double sum (double* x, double* par) {
    double val = myGauss(x, par) + myParabola(x, &par[3]); ←
    return val;
}

```

Indirizzo del quarto elemento dell'array, infatti sum sarà usato nel main come TF1 a 6 parametri, di cui myGauss usa i primi tre mentre myParabola usa i secondi tre

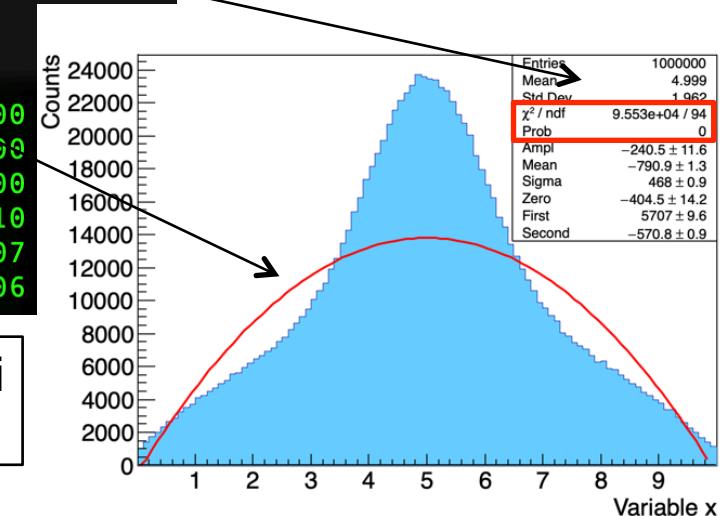
Funzioni utilizzate per definire delle TF1 di ROOT

E` stato scelto di definire separatamente la Normale e la Parabola per poter calcolare separatamente il numero di eventi sottostanti a ciascuna distribuzione

Fit dell'istogramma

Se ci si dimentica di inizializzare i parametri della funzione di fit (o non li si inizializza correttamente), è molto probabile che il fit non converga. → bisogna sempre ricordarsi di inizializzare i parametri del fit

```
FCN=95526.1 FROM HESSE STATUS=FAILED 11 CALLS 345 TOTAL
EDM=5.06542e-14 STRATEGY= 1 ERROR MATRIX UNCERTAINTY 10
0.0 per cent
EXT PARAMETER APPROXIMATE STEP FIRST
NO. NAME VALUE ERROR SIZE DERIVATIVE
1 Ampl -8.74194e+06 4.24264e+00 -0.00000e+00 0.00000e+00
2 Mean 0.00000e+00 1.41421e+00 -0.00000e+00 0.00000e+00
3 Sigma 0.00000e+00 1.41421e+00 -0.00000e+00 0.00000e+00
4 Zero -4.62196e+02 7.33703e+00 -6.47877e+03 8.80528e-10
5 First 5.70675e+03 1.16507e+00 5.51418e+03 -1.51786e-07
6 Second -5.70826e+02 1.31661e-01 -5.33934e+02 -1.57972e-06
```



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

`gStyle->SetOptFit(1112);`

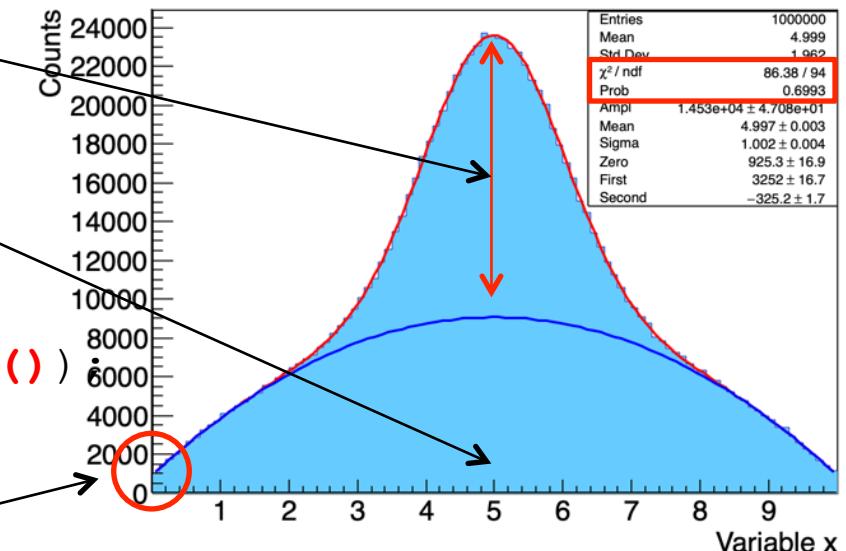
```
TF1* myFun = new TF1 ("myFun", sum, xMin, xMax, 6);
h1->Fit("myFun");
```

Fit dell'istogramma

```

TF1* myFun = new TF1 ("myFun", sum, xMin, xMax, 6);
// Gaussian maximum value
myFun->SetParameter(0, 10000);
// Gaussian mean value
myFun->SetParameter(1, 5.);
// Gaussian standard deviation
myFun->SetParameter(2, h1->GetRMS());
// Intercept of the parabola
myFun->SetParameter(3, 1000.);

```



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

```

FCN=86.3815 FROM MIGRAD   STATUS=CONVERGED
EDM=1.5791e-00  STRATEGY= 1  ERROR MATRIX UNCERTAINTY   0
.8 per cent
EXT PARAMETER                         STEP          FIRST
NO.   NAME      VALUE       ERROR      SIZE      DERIVATIVE
 1  Ampl      1.45346e+04  4.71230e+01 -1.34533e-02 -9.73667e-07
 2  Mean      4.99727e+00  2.92071e-03  2.70184e-07 -2.06981e-02
 3  Sigma     1.00243e+00  3.81134e-03  2.41859e-06 -3.14665e-02
 4  Zero      9.25323e+02  1.71416e+01 -1.60481e-04 -2.83385e-05
 5  First     3.25248e+03  1.67872e+01  7.05352e-03 -1.39538e-04
 6  Second    -3.25236e+02  1.66943e+00 -7.20685e-04 -1.21017e-03

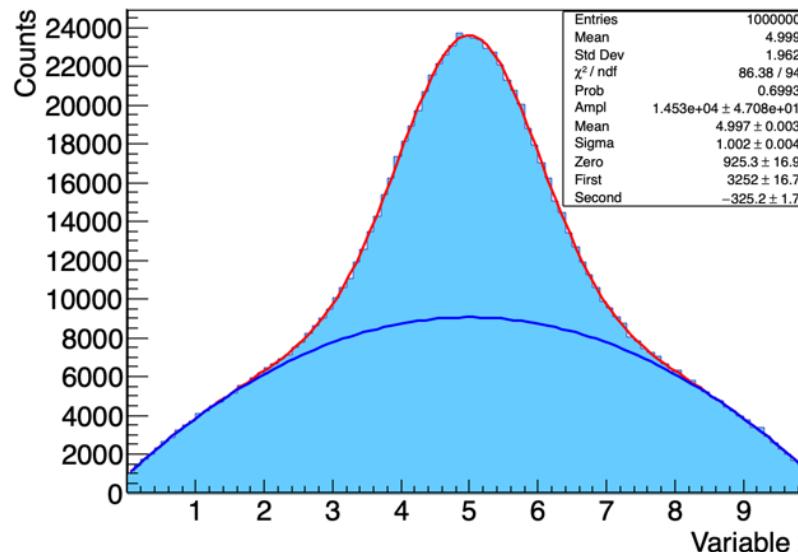
```

Get/setParameter di TF1

```

for (unsigned int i = 0; i < myFun->GetNpar(); i++)
    cout << "Parameter-" << i << " = " << myFun->GetParameter(i)
        << " +/- " << myFun->GetParError(i) << endl;
TF1* myBackground = new TF1("myBackground", myParabola,
                            xMin, xMax, 3);
myBackground->SetParameter(0,myFun->GetParameter(3));
myBackground->SetParameter(1,myFun->GetParameter(4));
myBackground->SetParameter(2,myFun->GetParameter(5));
myBackground->SetLineColor(kBlue);
myBackground->Draw("same");

```



Vengono usati i metodi **GetParameter** e **GetParError** della classe **TF1** per accedere ai valori dei parametri ed alle loro incertezze

Viene definita una nuova **TF1** per rappresentare la parabola ottenuta dal fit, separando così la componente Normale

Viene usata l'opzione grafica “**same**” per rappresentare la funzione **myBackground** sullo stesso canvas, senza cancellare il grafico precedente

Matrice di covarianza

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

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

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

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

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

Matrice di covarianza

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

```
FCN=86.3815 FROM MIGRAD      STATUS=CONVERGED      78 CALLS      79 TOTAL
                           EDM=1.23497e-08   STRATEGY= 1   ERROR MATRIX ACCURATE
EXT PARAMETER          VALUE        ERROR        STEP         SIZE      DERIVATIVE
NO.   NAME
 1  Ampl    1.45346e+04  4.70845e+01  1.53566e-01  1.28413e-06
 2  Mean     4.99727e+00  2.91935e-03  1.31252e-05  -9.19753e-03
 3  Sigma    1.00243e+00  3.80425e-03  9.65908e-06  1.61238e-02
 4  Zero     9.25323e+02  1.69269e+01  3.34888e-02  -4.92023e-06
 5  First    3.25248e+03  1.67048e+01  5.31782e-03  1.72316e-05
 6  Second   -3.25236e+02  1.66028e+00  6.00948e-04  1.01471e-04

*****
Minimizer is Minuit / Migrad
Chi2           =      86.3815
Ndf            =       94
Edm           =  1.23497e-08
NCalls         =       79
Ampl           =     14534.6 +/-  47.0845
Mean           =      4.99727 +/-  0.00291935
Sigma          =      1.00243 +/-  0.00380425
Zero           =      925.323 +/-  16.9269
First          =      3252.48 +/-  16.7048
Second         =     -325.236 +/-  1.66028

Covariance Matrix:

          Ampl      Mean      Sigma      Zero      First      Second
Ampl    2216.9  -0.00016654  -0.0070838  201.74  -353.07  35.332
Mean    -0.00016654  8.5226e-06  8.0064e-08  0.004957  -0.0011725  1.967e-05
Sigma   -0.0070838  8.0064e-08  1.4472e-05  0.019402  -0.042249  0.0042271
Zero    201.74   0.004957   0.019402   286.52  -201.88  18.323
First   -353.07  -0.0011725  -0.042249  -201.88  279.05  -27.547
Second  35.332   1.967e-05   0.0042271   18.323  -27.547  2.7565

Correlation Matrix:

          Ampl      Mean      Sigma      Zero      First      Second
Ampl    1  -0.0012116  -0.039548  0.25312  -0.44889  0.45197
Mean   -0.0012116    1  0.0072091  0.10031  -0.024043  0.0040581
Sigma  -0.039548   0.0072091    1  0.3013  -0.66483  0.66925
Zero   0.25312    0.10031   0.3013    1  -0.71396  0.65199
First  -0.44889   -0.024043  -0.66483  -0.71396    1  -0.99324
Second 0.45197   0.0040581   0.66925  0.65199  -0.99324    1
```

Matrice di covarianza

Se si ha bisogno di accedere alla matrice di covarianza (ad esempio per effettuare calcoli di propagazione delle incertezze), è possibile salvarla 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() ;
// or TMatrixDSym cov = r->GetCorrelationMatrix() ;
```

```
cout << "\nRetrieved covariance matrix" << endl;
for (int i = 0; i < myFun->GetNpar(); i++) {
    for (int j = 0; j < myFun->GetNpar(); j++) {
        double sigma_ij = cov(i,j); ↴
        cout << setw(15) << sigma_ij;
    }
    cout << endl;
}
```

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

Ancora sui TH1: metodi Get/Set

La classe degli histogrammi di ROOT mette a disposizione tutta una serie di **metodi di tipo Get** per **accedere ai data member** 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...

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

Ancora sui TH1: metodi Get/Set

A differenza degli array, per i quali il primo elemento ha indice zero, gli histogrammi di ROOT seguono la seguente convenzione

- bin = 0; *underflow bin*
- bin = 1; primo bin con il *low-edge INCLUSO*
- bin = nBins; ultimo bin con l'*upper-edge ESCLUSO*
- bin = nBins+1; *overflow bin*

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 i data member** di un oggetto istogramma

- `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...

Cosa fare se il fit non converge

Prima ancora di guardare il risultato del fit è necessario guardare l'output di **MIGRAD** a schermo. MIGRAD è il programma di ROOT che effettua la procedura di minimizzazione del funzionale (minimi quadrati o –log-likelihood)

```
FCN=63.2621 FROM MIGRAD   STATUS=CONVERGED    215 CALLS      216 TOTAL
                           EDM=3.00000e-00  STRATEGY= 1    ERROR MATRIX ACCURATE
EXT PARAMETER          VALUE        ERROR        STEP         FIRST
NO.    NAME      VALUE        ERROR        SIZE      DERIVATIVE
 1  Amp1      4.04111e+02  5.04127e+00  1.58692e-02  5.16625e-05
 2  Mean      5.00040e+00  9.87541e-03  3.86068e-05 -1.03630e-03
 3  Sigma     9.81257e-01  7.30460e-03  2.29747e-05  3.85261e-02
```

Solo quando si è sicuri che MIGRAD ha prodotto un risultato credibile ha senso guardare il plot, Chi-2, p-value, etc ...

Se lo STATUS non è CONVERGED, potrebbe essere necessario:

- cambiare i valori di inizio dei parametri
- evitare che i parametri assumano valori troppo grandi o troppo piccoli moltiplicandoli per dei valori opportuni (e.g. se un parametro (θ) ha un valore nell'intorno di 10^{-9} , scrivere il modello come $\theta \cdot 10^{-9}$, così θ varierà nell'intorno delle unità)

Cosa fare se il fit non converge

- “guidare” il fit fissando uno o più parametri e lasciando liberi gli altri (e.g. se il modello ha due parametri (θ_1 , θ_2) si può lanciare il fit fissando θ_1 ad un valore ragionevole (metodo **FixParameter(...)** di TF1) e lasciando θ_2 libero; se il fit converge si esegue un secondo fit immediatamente dopo il primo, in maniera tale che θ_2 abbia come valore di partenza quello trovato dal primo fit, lasciando liberi sia θ_1 che θ_2 (metodo **ReleaseParameter(...)** di TF1))

Inoltre, per essere sicuri che MINUIT abbia trovato un **massimo assoluto** e non uno relativo, è consigliato:

- lanciare tante volte (i.e. N) il fit con dei valori di inizio dei parametri, $\theta_1 \dots \theta_m$, scelti a caso in un range opportuno
- alla fine di ogni fit si registra, i.e. si salva, il valore del funzionale (Min^2 o $-\log(\text{likelihood})$)
- alla fine della sequenza degli N fit, si sceglie tra tutti quello che ha fornito il valore più piccolo del funzionale

Esercizi

Esercizio 1: Realizzare un programma che legga e fitti i dati presenti nel file `data1.txt`. In particolare il programma deve:

1. Ricevere in input il nome del file di dati
2. Implementare una funzione che legge il file di dati e li salva in un istogramma di 100 bin il cui minimo e massimo devono essere dedotti dai dati. Prototipo della funzione:

```
bool readData (char* fileName, vector<double>&  
dataList, double& xMin, double& xMax)
```

Dove:

- `dataList` è il vettore contenente i dati
- `xMin` è il più piccolo numero presente nei dati
- `xMax` è il più grande numero presente nei dati

3. Implementare mediante `TF1` la funzione di fit: somma di una Normale e di un polinomio del second'ordine
4. Fittare i dati

Esercizi

-
5. Scrivere a schermo il Chi-2 (metodo `GetChisquare()` di `TF1`), il numero di gradi di libertà (metodo `GetNDF()` di `TF1`), e il p-value (metodo `GetProb()` di `TF1`) computati da ROOT
 6. Scrivere a schermo il Chi-2, il numero di gradi di libertà ed il p-value (vedere metodo `Prob(...)` della classe `TMath.h`) **computati da voi** mediante una opportuna funzione, e verificare che coincidano con quelli computati da ROOT. Prototipo della funzione:

```
void computeChi2(TH1D* myHisto, TF1* myFun, double& Chi2, double& NDF, double& pvalue);
```

7. Dopo aver eseguito il fit calcolare quanti conteggi ci sono sotto la Normale e quanti sotto il polinomio del second'ordine

Esempio di formula per il computo del numero di conteggi:

```
nBins / (xMax - xMin) *myBackground->Integral(xMin, xMax)
```

Esercizi

Esercizio 2: Modificare il programma realizzato all'esercizio 1 aggiungendo le seguenti operazioni:

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 fissare in precedenza `h1`
5. Verificare di ottenere i medesimi risultati
6. Stampare a schermo la matrice di covarianza

Esercizi

Esercizio 3: Realizzate un programma che legga e fitti i dati presenti nel file `data2.txt`. Svolgere i medesimi punti dell'esercizio 1 con l'unica differenza che in questo caso non vi viene fornito nessun suggerimento ne` sul numero di bin, ne` sulla funzione con cui fissare i dati

Commento: in linea di principio potreste fissare qualunque cosa con un polinomio di grado sufficientemente elevato, ma fatevi guidare dal vostro senso fisico ... e dal rasoio di Occam: "*Pluralitas non est ponenda sine necessitate*" (i.e. a parità di fattori la spiegazione più semplice è da preferire)

Esercizi

Esercizio 4: 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 in un range $[0, 4]$ di momento
2. Definire tre funzioni, a due parametri (`a` e `b`) della forma

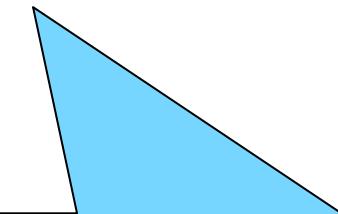
`double dist(double* E, double* par)`

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

$$\text{dist}_1(E) = aE^2 e^{-bE}$$

$$\text{dist}_2(E) = aE e^{-bE}$$

$$\text{dist}_3(E) = aE^2 e^{-bE^2}$$



Usare l'opzione `"+"` del metodo `Fit` per mantenere il disegno di tutte le tre funzioni sul grafico, e.g. `h1->Fit("MyFunc", "+") ;`

Usare `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 come test di bontà della regressione. Si usino le formule statistiche (NON il fit di ROOT) per determinare il Chi-2 ridotto:
 - calcolare il valore teorico della $dist_n(E)$ per ciascun bin (si prenda $E =$ valore centrale del bin)
 - estrarre il valore misurato della $dist_n(E)$ 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 quale 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