

Laboratorio II – 1° modulo

Lezione 11

Grafici e regressione

Laboratorio II – 1° modulo

Lezione 11

Grafici e regressione

Indice

- Rappresentazione grafica di relazioni tra grandezze fisiche: le classi `TGraph` e `TGraphErrors` di ROOT
- **Grafici e regressione**
- Regressione con `TGraphErrors`
- Incertezze sia in x che in y
- Interpolazione/Estrapolazione
- Relazione tra curva di livello per $\Delta M_{\text{in}^2} = 1$ ed incertezze sulla stima dei parametri
- Alcuni esempi patologici
- Test di bontà della regressione
- Esercizi
- HESSE vs MINOS
- Referenze

TGraph di ROOT

Spesso si ha la necessità di mettere in relazione due grandezze fisiche, e.g. posizione (y_i) – tempo (x_i), tensione (y_i) – frequenza (x_i), etc... per fare questo ROOT mette a disposizione due classi: [TGraph](#) e [TGraphErrors](#)

- **TGraph**: consente di mettere in relazione due grandezze (x e y) ma non contempla la possibilità di attribuirne le incertezze

```
#include "TGraph.h"
```

Possiede diversi tipi di costruttori:

```
TGraph myGraph; \\ Default constructor
```

```
TGraph myGraph(const char* filename); \\ Il costruttore legge i dati dal file contenente due colonne con le coordinate x e y dei punti
```

```
TGraph myGraph(int n, const double* x, const double* y); \\ Il costruttore legge i dati dai due array contenenti le coordinate x e y dei punti (n = numero di punti)
```

Esempio di file di dati

x	y
0	1
0.1	0.998029
0.2	0.992123
0.3	0.982305
...	

TGraph di ROOT

```
#include <iostream>
#include <cmath>
#include "TApplication.h"
#include "TCanvas.h"
#include "TGraph.h"
#include "TAxis.h"

using namespace std;

int main(int argc, char** argv)
{
    if (argc < 2)
    {
        cout << "Not enough parameters: ./myTGraph filename.txt" << endl;
        return 1;
    }

TApplication* myApp = new TApplication("myApp", NULL, NULL);
TCanvas* myC = new TCanvas("myC", "myC", 0, 0, 700, 500);

// TGraph default constructor and SetPoint method
double x,y;
int nPoints = 100;
double period = 10;
TGraph myGraph0;
for (int i = 0; i < nPoints; i++)
{
    x = static_cast<double>(i) / nPoints * period;
    y = sin(2.*3.14/period * x);
    myGraph0.SetPoint(myGraph0.GetN(),x,y);
}
myGraph0.GetXaxis()->SetTitle("x (units)");
myGraph0.GetYaxis()->SetTitle("y (units)");
myGraph0.SetMarkerStyle(20);
myGraph0.SetMarkerSize(0.5);
```

• Metodo per inserire le misure

Imposta attributi:

- titolo degli assi
- tipo di simbolo del marker
- dimensione del marker

TGraph di ROOT

```
// TGraph constructor from file
TGraph myGraph1(argv[1]);
myGraph1.SetMarkerStyle(21);
myGraph1.SetMarkerSize(0.5);
myGraph1.SetMarkerColor(kRed);

myC->cd();
myGraph0.Draw("AP");
myGraph1.Draw("P same");
myC->Modified();
myC->Update();

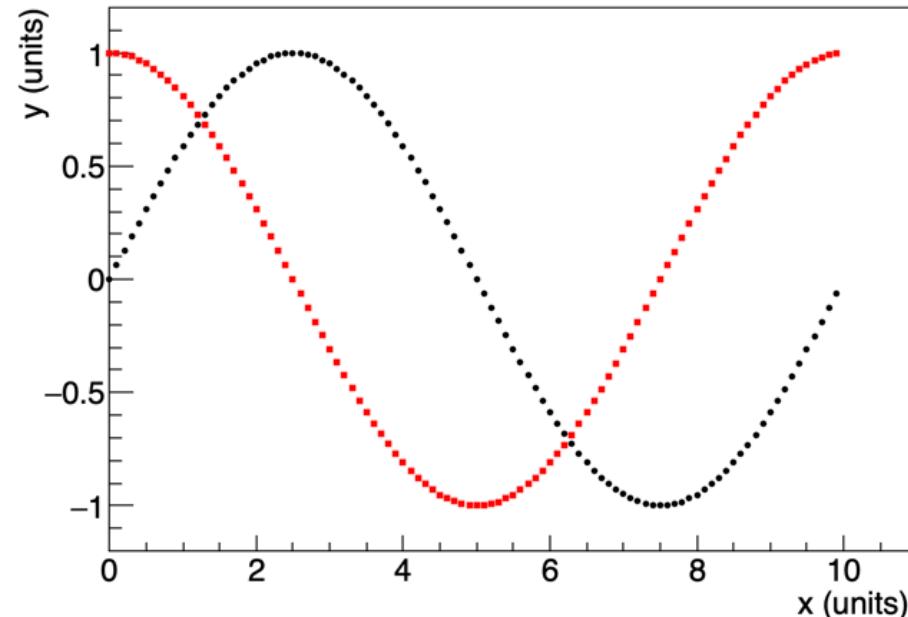
myApp->Run();
return 0;
}
```

Imposta attributi:

- tipo di simbolo del marker
- dimensione del marker
- colore del marker

- Opzione “**A**” per disegnare gli assi
- Opzione “**P**” per disegnare i marker
- Opzione “**same**” per disegnare sullo stesso canvas senza cancellare ciò che è stato disegnato precedentemente

Risultato del programma



TGraphErrors di ROOT

Se si ha la necessità di impostare anche le incertezze dei punti sperimentali è necessario usare la classe [TGraphErrors](#)

- **TGraphErrors**: consente di mettere in relazione due grandezze (x e y) attribuendone anche le incertezze

```
#include "TGraphErrors.h"
```

Possiede diversi tipi di costruttori:

```
TGraphErrors myGraph; \\ Default constructor
```

TGraphErrors myGraph(const char* filename); \\ Il costruttore legge i dati dal file contenente quattro colonne con le coordinate x e y dei punti e le incertezze, in x e y , da attribuire a ciascun punto

```
TGraphErrors myGraph(int n, const double* x, const double* y,
const double* ex, const double* ey); \\ Il costruttore legge i dati dai quattro array contenenti le coordinate  $x$  e  $y$  dei punti e le incertezze da attribuire a ciascun punto (n = numero di punti)
```

Esempio di file di dati

x	y	errX	errY
0	1	0.25	0.05
0.25	0.987701	0.25	0.05
0.5	0.951106	0.25	0.05
0.75	0.891115	0.25	0.05
1	0.809204	0.25	0.05
...			

TGraphErrors di ROOT

```
#include <iostream>
#include <cmath>
#include "TApplication.h"
#include "TCanvas.h"
#include "TGraphErrors.h"
#include "TAxis.h"

using namespace std;

int main(int argc, char** argv)
{
    if (argc < 2)
    {
        cout << "Not enough parameters: ./myTGraphErrors filename.txt" << endl;
        return 1;
    }

TApplication* myApp = new TApplication("myApp", NULL, NULL);
TCanvas* myC = new TCanvas("myC", "myC", 0, 0, 700, 500);

// TGraphErrors default constructor and SetPoint method
double x,y;
int nPoints = 40;
double period = 10;
double errX = period / nPoints;
double errY = 0.05;
TGraphErrors myGraph0;
for (int i = 0; i < nPoints; i++)
{
    x = static_cast<double>(i) / nPoints * period;
    y = sin(2.*3.14/period * x);
    myGraph0.SetPoint(myGraph0.GetN(),x,y); ←
    myGraph0.SetPointError(myGraph0.GetN()-1,errX,errY);
}
```

Metodi per inserire le misure
e le incertezze

TGraphErrors di ROOT

```

myGraph0.GetXaxis()->SetTitle("x (units)");
myGraph0.GetYaxis()->SetTitle("y (units)");
myGraph0.SetMarkerStyle(20);
myGraph0.SetMarkerSize(0.5);

// TGraphErrors constructor from file
TGraphErrors myGraph1(argv[1]);
myGraph1.SetMarkerStyle(21);
myGraph1.SetMarkerSize(0.5);
myGraph1.SetMarkerColor(kRed);

myC->cd();
myGraph0.Draw("AP");
myGraph1.Draw("P same");
myC->Modified();
myC->Update();

myApp->Run();
return 0;
}

```

- Opzione “**A**” per disegnare gli assi
- Opzione “**P**” per disegnare i marker
- Opzione “**same**” per disegnare sullo stesso canvas senza cancellare ciò che è stato disegnato precedentemente

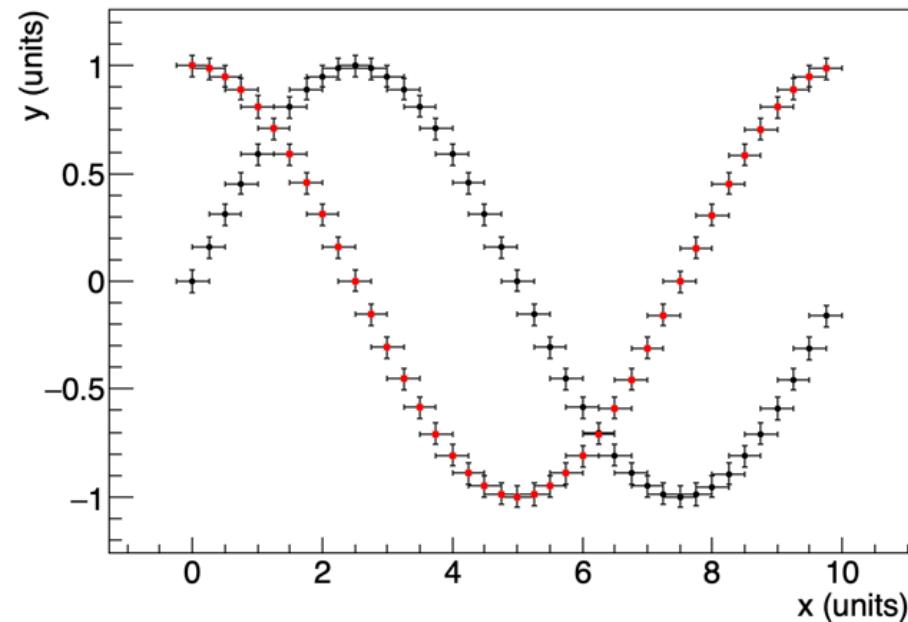
Imposta attributi:

- titolo degli assi
- tipo di simbolo del marker
- dimensione del marker

Imposta attributi:

- tipo di simbolo del marker
- dimensione del marker
- colore del marker

Risultato del programma



Grafici e regressione

Nel caso dei **grafici** i funzionali dei **minimi quadrati** e della **massima verosimiglianza** assumono la seguente forma

Minimi quadrati:

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

Massima verosimiglianza (nell'ipotesi di **errori** di misura che seguono una **distribuzione Normale**):

Dove: $L(\vec{\theta}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(y_i - f(x_i; \vec{\theta}))^2}{2\sigma_i^2}}$

n: numero di misure

x_i: variabile indipendente

y_i: osservabile oggetto della misura a cui è attribuita un'incertezza **σ_i**

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

Nel caso di errori che seguono una distribuzione Normale i due metodi sono equivalenti, la $-\log(\text{Likelihood})$ infatti si riduce alla sommatoria dei minimi quadrati (a meno di un fattore $\frac{1}{2}$)

Regressione con TGraphErrors

```
...
double cauchy(double* x, double* par)
{
    double val = par[0] + par[1]/(x[0]*x[0]);
    return val;
}
```

Modello per fissare i dati: $n(\lambda) = a + b/\lambda^2$, relazione di Cauchy che lega l'indice di rifrazione di un mezzo dispersivo alla lunghezza d'onda dell'onda elettromagnetica che lo attraversa

```
void computeChi2(TGraphErrors* myData, TF1* myFun, double& Chi2, double& NDF, double& pvalue)
{
    double result = 0;

    for (int i = 0; i < myData->GetN(); i++)
    {
        result += pow(myData->GetY()[i] - myFun->Eval(myData->GetX()[i]), 2.) /
        (pow(myData->GetErrorY(i), 2.) + pow(myFun->Derivative(myData->GetX()[i]) * myData->GetErrorX(i), 2.));
    }

    Chi2 = result;
    NDF = myData->GetN() - myFun->GetNpar();
    pvalue = TMath::Prob(Chi2, NDF);
}
```

```
int main(int argc, char** argv)
{
    if (argc < 2)
    {
        cout << "Not enough parameters: ./fitTGraphErrors filename.txt" << endl;
        return 1;
    }
    gStyle->SetOptFit(1112);
    TApplication* myApp = new TApplication("myApp", NULL, NULL);
    TCanvas* myC = new TCanvas("myC", "myC", 0, 0, 700, 500);
    ...
}
```

Funzione per il calcolo del Chi-2, cioè del funzionale Min^2 , con abuso di linguaggio a volte i due nomi vengono utilizzati per riferirsi alla stessa cosa

Regressione con TGraphErrors

```
...
TF1* myFun = new TF1("myFun", cauchy, 400, 1000, 2);
myFun->SetParameter(0, 1.6);
myFun->SetParameter(1, 8e3);
myFun->SetParName(0, "Const");
myFun->SetParName(1, "Scale");

// TGraphErrors constructor from file
TGraphErrors* myGraph = new TGraphErrors(argv[1]);
myGraph->GetXaxis()->SetTitle("lambda (nm)");
myGraph->GetYaxis()->SetTitle("Refractive index");
myGraph->SetMarkerStyle(20);
myGraph->SetMarkerSize(0.5);

myC->cd();
myGraph->Draw("AP");
myGraph->Fit("myFun");
myC->Modified();
myC->Update();

cout << "\nReduced Chi-2: "
     << myFun->GetChisquare() / myFun->GetNDF() << endl;
cout << "p-value: " << myFun->GetProb() << endl;

double Chi2, NDF, pvalue;
computeChi2(myGraph, myFun, Chi2, NDF, pvalue);
cout << "\nMy Chi-2: " << Chi2 << endl;
cout << "My NDF: " << NDF << endl;
cout << "My reduced Chi-2: " << Chi2/NDF << endl;
cout << "My p-value: " << pvalue << endl;

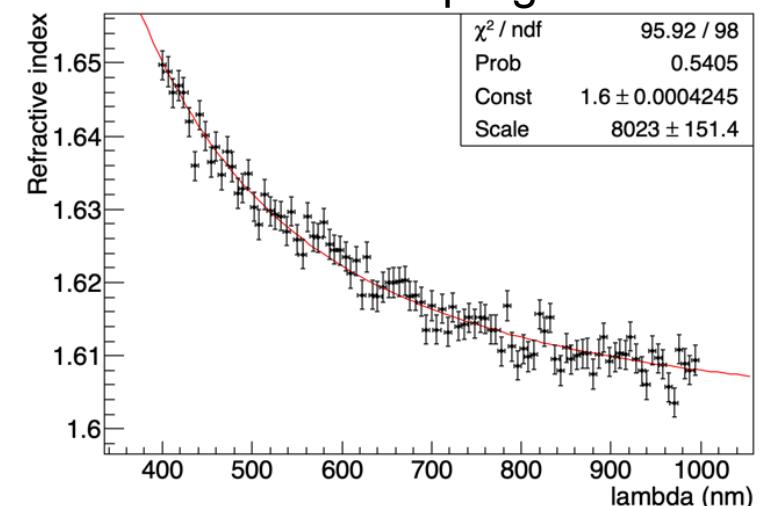
myApp->Run();
return 0;
}
```

Abbiamo visto i metodi del **TGraphErrors**:

- **GetN()**: restituisce il numero di punti del **TGraphErrors**
- **GetX()**: restituisce un array di double (i.e. **double***)
- **GetY()**: restituisce un array di double (i.e. **double***)
- **GetErrorX(i)**: restituisce l'incertezza in x del punto i -esimo del **TGraphErrors**
- **GetErrorY(i)**: restituisce l'incertezza in y del punto i -esimo del **TGraphErrors**

Inoltre abbiamo visto il metodo **Eval(double x)** di **TF1**: restituisce il valore della funzione in corrispondenza di x ed il metodo **Derivative(double x)** che computa la derivata numerica della **TF1** in x

Risultato del programma



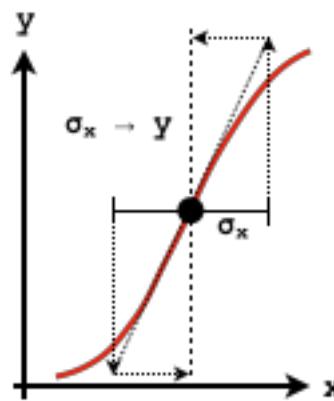
Incertezze sia in x che in y

A volte oltre alle incertezze sulla variabile dipendente (y) possono essere presenti anche le incertezze sulla variabile indipendente (x). Nel computo del funzionale Min^2 è necessario considerarle entrambe. Questo viene fatto osservando che i denominatori della serie Min^2 non sono altro che le varianze dei numeratori. Possiamo quindi calcolarli tramite la formula per la **propagazione degli errori** e sotto l'ipotesi che le misure di x e y siano statisticamente indipendenti:

$$\text{Var}[y - f(x; \vec{\theta})] = \sigma_y^2 + \left(\frac{\partial f(x; \vec{\theta})}{\partial x} \right)^2 \sigma_x^2$$

Di conseguenza il funzionale dei minimi quadrati diventa:

$$\text{Min}^2(\vec{\theta}) = \sum_{i=1}^n \frac{(y_i - f(x_i; \vec{\theta}))^2}{\sigma_{y_i}^2 + \left(\frac{\partial f(x; \vec{\theta})}{\partial x} \Big|_{x_i} \right)^2 \sigma_{x_i}^2}$$



L'interpretazione geometrica può essere ottenuta dalla figura, dove le incertezze in x sono trasportate sull'asse delle y tramite la tangente, e.g. se il modello ha tangente uguale a zero in x_i , allora l'incertezza in x non conta nel computo di Min^2 .

Modelli lineari nei parametri

La ricerca di un estremante, in particolare del minimo, del funzionale Min^2 non è in generale un problema che può essere scritto con un'espressione semplice, i.e. chiusa, analitica, sono quindi necessarie tecniche numeriche di ricerca degli estremanti. Se però il **modello** è **lineare nei parametri**, il problema si riconduce alla semplice inversione di una matrice. Si prenda per esempio un modello a due parametri $f(x; \theta_1, \theta_2) = \theta_1 h_1(x) + \theta_2 h_2(x)$:

$$\text{Min}^2(\vec{\theta}) = \sum_{i=1}^n \left(\frac{y_i - (\theta_1 h_1(x_i) + \theta_2 h_2(x_i))}{\sigma_i} \right)^2$$

Derivando rispetto ai parametri si ottiene:

Nella pratica, al posto di definire:

$$\begin{cases} -2 \sum_{i=1}^n \left(\frac{y_i - (\theta_1 h_1(x_i) + \theta_2 h_2(x_i))}{\sigma_i} \frac{h_1(x_i)}{\sigma_i} \right) = 0 \\ -2 \sum_{i=1}^n \left(\frac{y_i - (\theta_1 h_1(x_i) + \theta_2 h_2(x_i))}{\sigma_i} \frac{h_2(x_i)}{\sigma_i} \right) = 0 \end{cases}$$

```
TF1* myFun = new TF1 ("myFun", cauchy, 400, 1000, 2);
```

basta semplicemente utilizzare l'operatore **++**, come segue:

```
TF1* myFun = new TF1 ("myFun", "1 ++ 1/(x*x)", 400, 1000);
```

automaticamente viene interpretato come **[0] + [1]/(x*x)**, inoltre poiché il modello è lineare nei parametri, evita di utilizzare un algoritmo numerico di ricerca dell'estremante del funzionale (il tutto non è più valido nel caso di incertezze anche sulla variabile x poiché rendono il problema non lineare)

Interpolazione/Estrapolazione

Dopo aver eseguito il fit ed aver quindi stimato i parametri del modello con le relative incertezze può essere necessario stimare il valore della variabile dipendente (y) del modello per un punto arbitrario dell'intervallo di validità della variabile indipendente (x). In generale dato un modello lineare nei parametri:

$$f(x; \vec{\theta}) = \theta_1 h_1(x) + \theta_2 h_2(x) + \cdots + \theta_m h_m(x) = y$$

l'incertezza associata a y , applicando la formula per la propagazione degli errori, risulta:

$$\begin{aligned} \sigma_y^2 &= \sum_{i=1}^m \left(\frac{\partial f}{\partial \theta_i} \right)^2 \sigma_{\theta_i}^2 + 2 \sum_{j=1, j \neq i}^m \frac{\partial f}{\partial \theta_i} \frac{\partial f}{\partial \theta_j} \text{Cov}[\theta_i, \theta_j] = \\ &= \sum_{i=1}^m h_i^2(x) \sigma_{\theta_i}^2 + 2 \sum_{j=1, j \neq i}^m h_i(x) h_j(x) \text{Cov}[\theta_i, \theta_j] \end{aligned}$$

E.g.: dato un modello parabolico $y = \theta_1 + \theta_2 x + \theta_3 x^2$ si vuole stimare il valore dei parametri dai dati, ed **interpolare/estrapolare** il valore del modello a x_p

- Eseguire fit con l'opzione “**S**” → salva la matrice di covarianza (**cfr. Lezione 10**)
- Fissato x_p calcolare y_p con il metodo `Eval(x_p)`
- L'incertezza sulla stima di y_p risulta:

$$\sigma_y^2 = \sigma_{\theta_1}^2 + x_p^2 \sigma_{\theta_2}^2 + x_p^4 \sigma_{\theta_3}^2 + 2(x_p \text{Cov}[\theta_1, \theta_2] + x_p^2 \text{Cov}[\theta_1, \theta_3] + x_p^3 \text{Cov}[\theta_2, \theta_3])$$

Relazione tra curva di livello per $\Delta\text{Min}^2 = 1$ ed incertezze sulla stima dei parametri

L'incertezza sulla stima dei parametri è intimamente legata alla curvatura del funzionale nell'intorno del minimo, infatti la matrice di covarianza è proporzionale all'inverso della matrice Hessiana. Per modelli lineari nei parametri la radice quadrata degli elementi diagonali della matrice di covarianza, i.e. le varianze, coincidono con la proiezione della curva di livello per $\Delta\text{Min}^2 = 1$ sugli assi coordinati

Si prenda l'esempio della relazione di Cauchy ($n(\lambda) = \mathbf{a} + \mathbf{b}/\lambda^2$). Il risultato del fit fornisce la seguente stima dei parametri:

Curva di livello per $\Delta\text{Min}^2 = 1$

Const (i.e. \mathbf{a}) = 1.60001 ± 0.0004

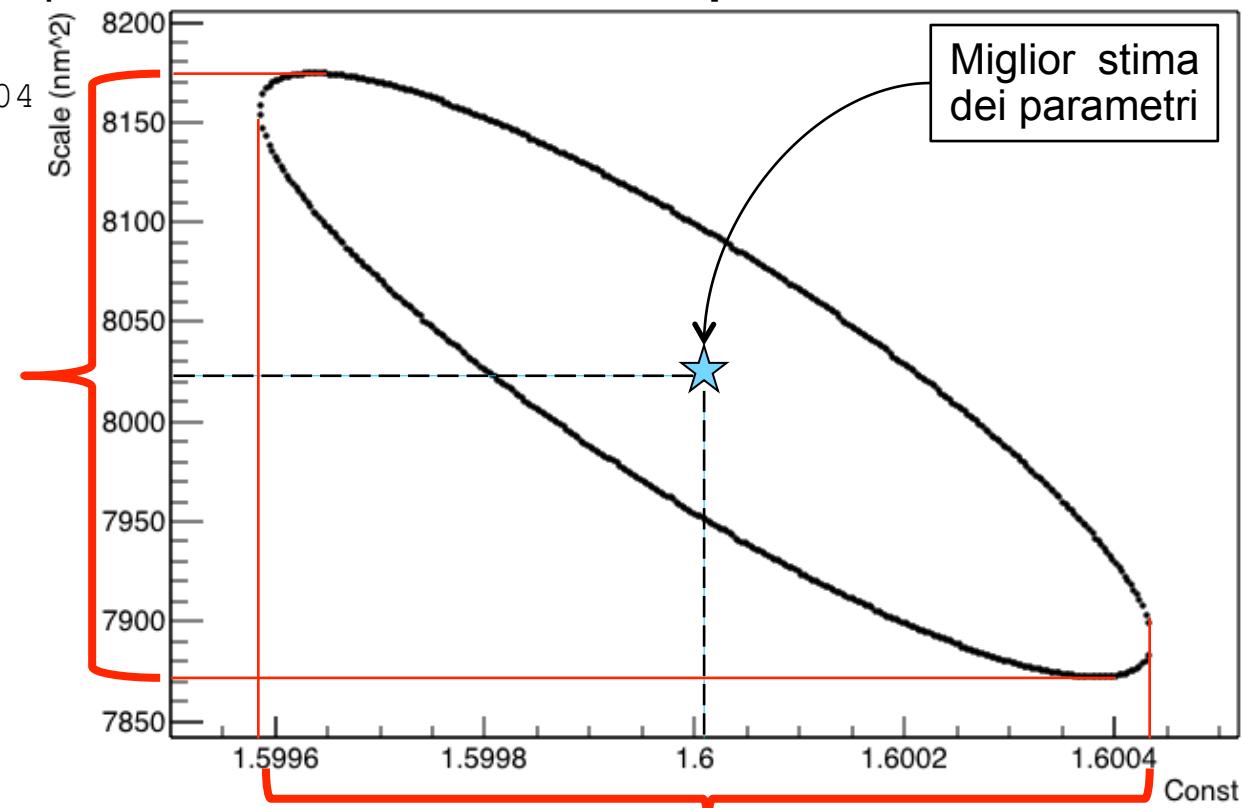
range: $1.59961 \div 1.60041$

Scale (i.e. \mathbf{b}) = 8023 ± 151

range: $7872 \div 8174$

$\rho = -0.87805$ (coefficiente
di correlazione)

Il cui range coincide esattamente con la proiezione sugli assi della curva di livello per $\Delta\text{Min}^2 = 1$



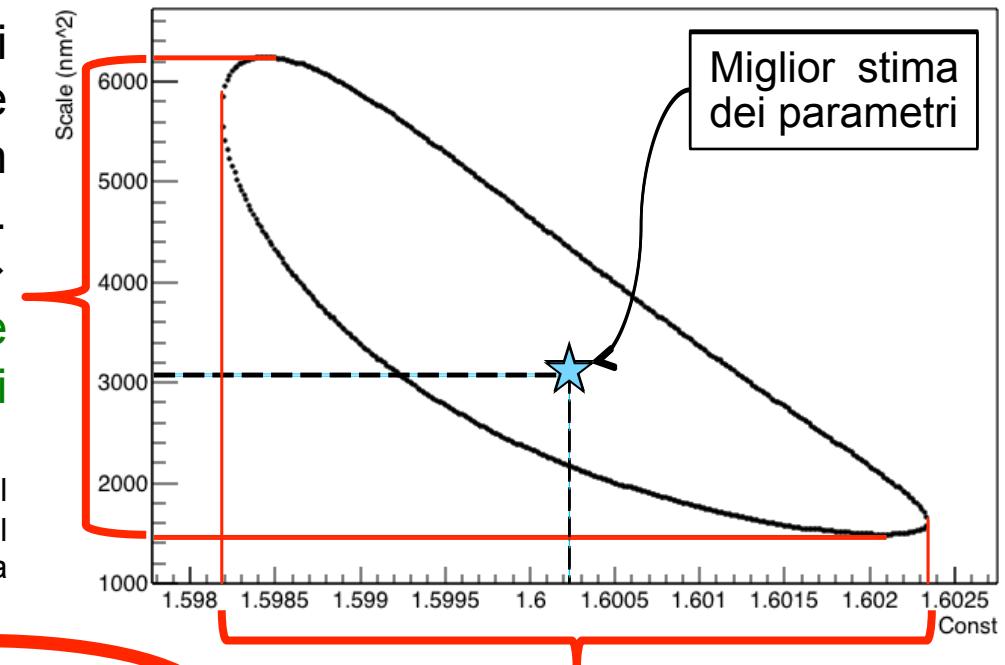
Alcuni esempi patologici

Si prenda ancora come esempio la legge di Cauchy ($n(\lambda) = a + b/\lambda^2$) dove però abbiamo volutamente “**non-linearizzato**” il modello: $n(\lambda) = a + 10^3 \cdot \log(b)/\lambda^2$

Se si esegue il programma di esempio con il file di dati “fittGraphErrors_nonLin.txt” si ottiene la seguente curva di livello per $\Delta \text{Min}^2 = 1$

La curva risulta distorta e questo si ripercuote sulla stima delle incertezze dei parametri (le incertezze possono non aver più distribuzione Normale, e.g. possono non essere più simmetriche) → **quando possibile è consigliabile utilizzare modelli lineari nei parametri** (cfr. sezione “*HESSE VS MINOS*”)

Questo non vuol dire che dovete linearizzare/approssimare, il modello, semplicemente se c’è la possibilità di riscrivere il modello in maniera che sia lineare nei parametri, ma formalmente equivalente a quello non-lineare, è preferibile



```
FCN=98.9501 FROM MIGRAD   STATUS=CONVERGED
                           EDM=5.67466e-08  STRATEGY= 1
EXT PARAMETER                STEP         FIRST
NO. NAME        VALUE       ERROR      SIZE      DERIVATIVE
 1  Const        1.60026e+00  2.07160e-03  4.88494e-06  1.70097e-01
 2  Scale        3.03235e+03  2.18166e+03  5.07053e+00  2.74680e-07
```

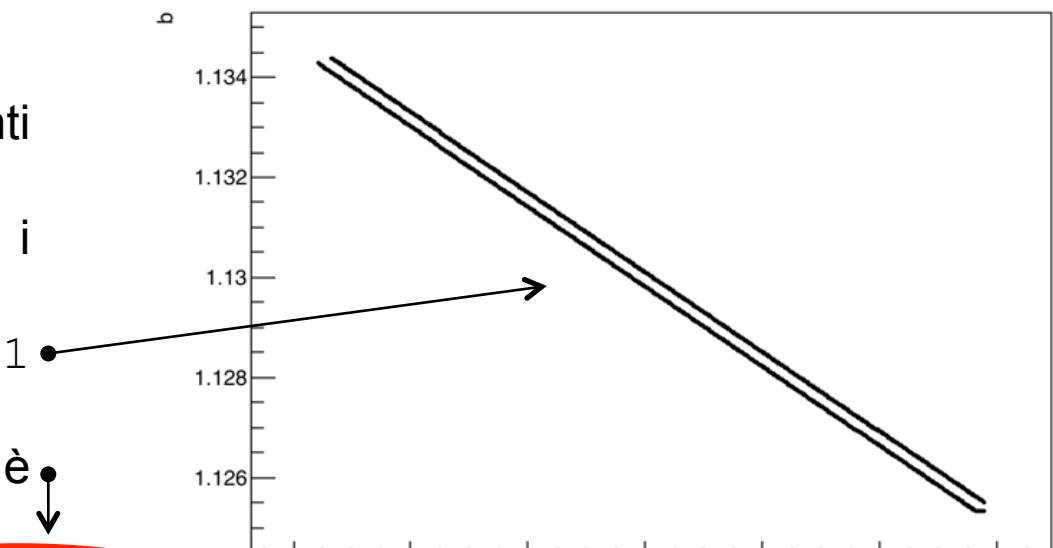
Alcuni esempi patologici

Un esempio di errata impostazione del problema di fit lo si ha quando nel modello sono presenti due o più parametri altamente correlati

Per simulare questo fattiamo i dati con il modello $n(\lambda) = a*b + 8023/\lambda^2$, dove a e b sono ovviamente degeneri perché lo stesso modello potrebbe essere scritto come $n(\lambda) = c + 8023/\lambda^2$

In questo caso si osservano le seguenti patologie:

- il coefficiente di correlazione tra i parametri a e b risulta: -0.998
- la curva di livello per $\Delta \text{Min}^2 = 1$ risulta degenere
- la matrice di covarianza non è definita positiva



```
FCN=95.9226 FROM HESSE      STATUS=NOT POSDEF    10 CALLS      88 TOTAL
                           EDM=1.61044e-12   STRATEGY= 1      ERR MATRIX NOT POS-DEF
EXT PARAMETER APPROXIMATE STEP FIRST
 NO. NAME     VALUE      ERROR      SIZE      DERIVATIVE
 1 Const     1.41611e+00  2.84448e-03  3.37627e-07  9.99519e-03
 2 Scale     1.12986e+00  2.26951e-03  4.87655e-07  1.25275e-02
```

Quando possibile evitare elevate correlazioni tra i parametri ed evitare sempre che due o più parametri siano degeneri come nell'esempio riportato

Test di bontà della regressione

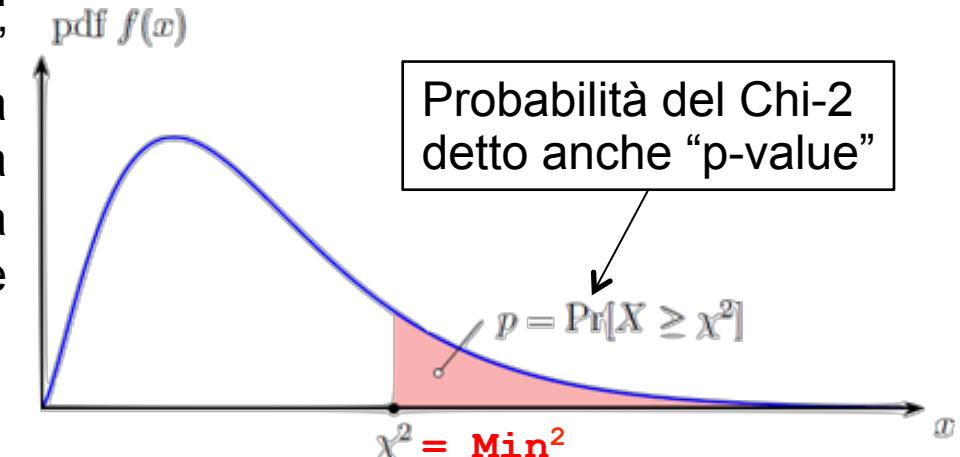
Il funzionale dei **minimi quadrati**, sotto l'ipotesi di incertezze aventi distribuzione di probabilità Normale, ha **distribuzione di probabilità χ^2_{n-m}** , n numero di misure, m numero dei parametri del modello

$$\text{Min}^2(\vec{\hat{\theta}}) = \sum_{i=1}^n \left(\frac{y_i - f(x_i; \vec{\hat{\theta}})}{\sigma_i} \right)^2 \sim \chi^2_{n-m}$$

Infatti se è vero che i dati seguono il modello $f(x; \theta)$ allora $f(x_i; \theta)$ è il “vero” valore che avrebbe dovuto assumere la grandezza fisica (chiamiamola μ_i), ma a causa delle incertezze (rappresentate da σ_i) ha assunto i valori y_i . Se le incertezze hanno distribuzione Normale, allora

$$\frac{y_i - \mu_i}{\sigma_i} \sim N(0,1)$$

il cui quadrato segue una distribuzione χ^2_1 . A seguito del processo di ottimizzazione Min^2 è un numero. Calcolare l'area sottesa dalla distribuzione χ^2_{n-m} da Min^2 a infinito (i.e. il “p-value”) equivale a chiedersi quale sia la probabilità che il funzionale assuma un qualunque valore nell'intervallo $[\text{Min}^2, \infty]$. Se il p-value è “piccolo” allora è poco probabile che il modello descriva i dati



Riassumendo...

L'operazione di regressione (i.e. fit) consta di tre passaggi distinti:

- ✓ Problema di ottimizzazione del modello rispetto ai dati (e.g. minimizzazione minimi quadrati) → miglior stima dei parametri
- ✓ Quantificare l'incertezza sulla stima dei parametri
- ✓ Quantificare la qualità della regressione → effettuare un test statistico che sia in grado di dire quale sia la probabilità che il modello descriva i dati

E` importante notare che anche se MIGRAD è stato in grado di trovare il minimo del funzionale non è detto che sia stato anche in grado di calcolare opportunamente la **matrice di covarianza**, che, si ricorda, è proporzionale all'**inverso** della **matrice Hessiana** → prima ancora di guardare al risultato grafico del fit è necessario guardare l'output di MIGRAD per verificare STATUS=CONVERGED

Quando si implementa un modello per descrivere i dati, **se possibile**, bisogna scriverlo in maniera tale che sia **lineare nei parametri**

Bisogna inoltre **evitare** che ci siano **parametri degeneri**

Esercizi

Esercizio 1: Inserire i dati contenuti nel file `ese01Fit.txt` in un `TGraphErrors` ed eseguirne il fit secondo il modello:

$$f(x; \tau) = V_0 [1 - \exp(-x/\tau)]$$

- Stampate a schermo i risultati ottenuti per i parametri V_0 e τ , unitamente ai loro errori

Esercizio 2 (interpolazione/estrapolazione): Fittare i dati contenuti nel file `ese02Extr.txt` con una parabola ed estrapolare il valore della variabile y_p in corrispondenza di $x_p = 32$

- Stimare quindi l'incertezza associata a y_p tenendo conto delle correlazioni associate ai parametri di fit

Esercizi

Esercizio 3: Aggiungere al codice di slide 10 e 11 una funzione che calcola la curva di livello per $\Delta\text{Min}^2 = \text{delta}$ (dove delta è un parametro della funzione)

- Il prototipo della funzione è:

```
TGraph* computeContour (TGraphErrors* myData, TF1*  
myFun, double delta);
```

Riceve quindi in ingresso il **TGraphErrors** contenente i dati, la **TF1** che contiene la funzione di fit ed il valore di ΔMin^2 a cui calcolare la curva di livello. Restituisce un puntatore a **TGraph** nel quale la funzione avrà salvato i punti che costituiscono la curva di livello

La funzione deve ricercare nello spazio (**Const**; **Scale**) il valore di ΔMin^2 che più si avvicina a **delta**. Deve quindi implementare due cicli **for** (uno dentro l'altro, quello più esterno sulla variabile **Const**, quello più interno sulla variabile **Scale**) con i quali realizzare la ricerca

Esercizi

Entrambi i cicli devono partire dal valore ottimo del parametro meno due sigma e devono arrivare fino al valore ottimo più due sigma: $(\text{Const}-2\sigma; \text{Const}+2\sigma)$ e $(\text{Scale}-2\sigma; \text{Scale}+2\sigma)$. I cicli devono essere divisi in un numero di passi pari a 600

Per ogni valore di Const (chiamiamolo $\text{Const}_{i-\text{esimo}}$) il ciclo for più interno ricerca il valore di Scale tale per cui ΔMin^2 si avvicina di più a delta. Quando questo punto viene trovato (chiamiamolo $\text{Scale}_{j-\text{esimo}}$) salva il punto $\text{Const}_{i-\text{esimo}}; \text{Scale}_{j-\text{esimo}}$ in un [TGraph](#)

N.B.:

- i valori ottimi dei parametri sono quelli restituiti dal fit
- il ΔMin^2 è calcolato come differenza tra il valore del funzionale Min^2 ottenuto al termine del fit e quello ottenuto per ogni punto $\text{Const}_{i-\text{esimo}}; \text{Scale}_{j-\text{esimo}}$ durante l'esecuzione del ciclo for più interno

HESSE VS MINOS

Nel caso di modelli con dipendenza lineare dai parametri il calcolo delle incertezze sui parametri del modello mediante l'approssimazione parabolica del funzionale e mediante la proiezione delle curva di livello per $\Delta \text{Min}^2 = 1$ coincidono (si ricorda infatti che la matrice di covarianza è proporzionale all'inverso della matrice Hessiana e che gli elementi diagonali della matrice di covarianza sono le varianze sui parametri del modello)

Nel caso della likelihood questo è rigorosamente vero solo se $\log(f(x; \theta_1 \dots \theta_m))$ è lineare in $\theta_1 \dots \theta_m$, nel qual caso si prende come curva di livello $\Delta \log(\text{Likelihood}) = 0.5$

MIGRAD di *default* utilizza la matrice Hessiana per il calcolo delle incertezze sui parametri perché è il metodo più veloce

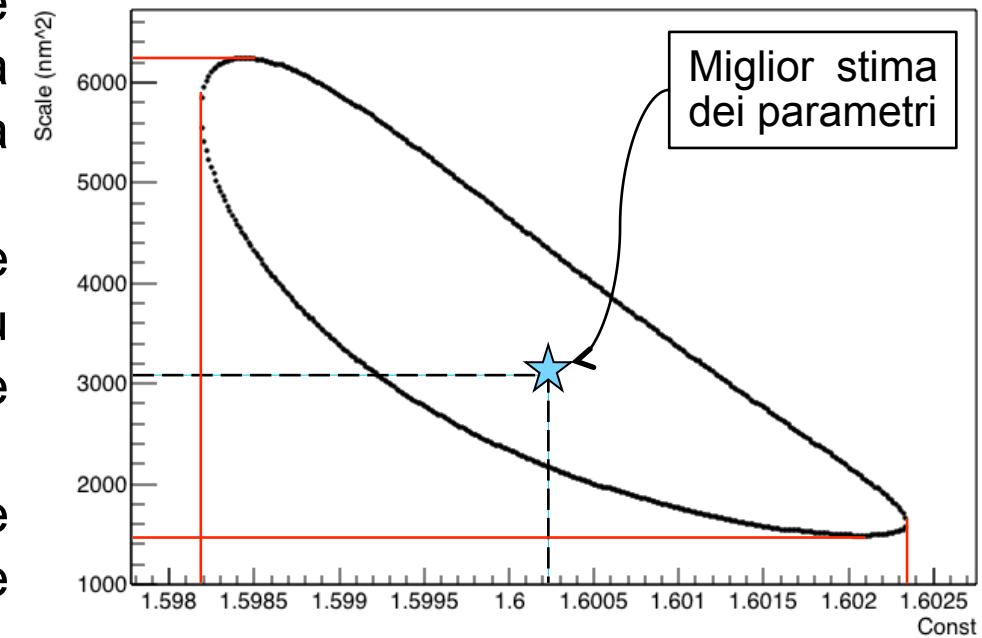
HESSE VS MINOS

Per forzare l'uso del metodo basato sulla proiezione della **curva di livello** per $\Delta\text{Min}^2 = 1$, ovvero $\Delta\log(\text{Likelihood}) = 0.5$, (i.e. **metodo più corretto per trattare le incertezze**) bisogna utilizzare l'opzione di fit “**E**” (in associazione ai metodi **LowerError** (`unsigned int i`) e **UpperError** (`unsigned int i`) della classe `TFitResult.h`)

In ROOT il metodo della proiezione della curva di livello per $\Delta\text{Min}^2 = 1$, ovvero $\Delta\log(\text{Likelihood}) = 0.5$, è detto **MINOS**, mentre quello basato sulla matrice Hessiana è detto **HESSE**. **Riassumendo:**

- Nel caso in cui l'approssimazione parabolica del funzionale nell'intorno del minimo sia valida, le incertezze possono essere calcolate sia utilizzando la matrice Hessiana sia la proiezione della curva di livello
- Nel caso in cui l'approssimazione parabolica non sia valida è più opportuno utilizzare la proiezione della curva di livello

N.B.: la matrice di covarianza è comunque sempre calcolata tramite l'inverso della matrice Hessiana



Referenze

Il pacchetto di ROOT per l'ottimizzazione si chiama MINUIT. Il programma di MINUIT che ricerca il minimo si chiama MIGRAD, mentre i programmi che calcolano gli errori si chiamano HESSE e MINOS

Per un approfondimento si veda:

- “*MINUIT - A system for function minimization and analysis of the parameter errors and correlations*” F. James and M. Roos, Computer Physics Communications **10** (1975) 343-367
- “*Interpretation of the shape of the likelihood function around its minimum*” F. James, Computer Physics Communications **20** (1980) 29-35
- “*MINUIT - Function minimization and error analysis - Reference manual*” F. James
- “*MINUIT - User's guide*” F. James and M Winkler

Come si può vedere da questo link non esistono molti software per risolvere problemi di ottimizzazione: [en.wikipedia.org/wiki/
List_of_optimization_software](https://en.wikipedia.org/wiki/List_of_optimization_software)

, uno di questi è proprio il pacchetto MINUIT di ROOT