

# Laboratorio II – 1° modulo

## Lezione 9

### Analisi dei dati e rappresentazione grafica con ROOT

# Laboratorio II – 1° modulo

## Lezione 9

### Analisi dei dati e rappresentazione grafica con ROOT

# Indice

---

- Rappresentazione grafica dei dati con ROOT
  - TF1
  - TH1
- Problema: calcolo di un intervallo di confidenza utilizzando la funzione di likelihood
  - Ricerca di zeri della funzione
  - Costruzione della Likelihood
  - Calcolo dell'intervallo di confidenza
- Approfondimenti
  - ROOT in interattivo
  - Analisi dati

# Programma ROOT “prototipo”

Costruiamo un programma che utilizza **TApplication** per la creazione di un ambiente grafico e proviamo a disegnare una funzione

Esempio:

```
#include <TApplication.h>
#include <TF1.h>
int main()
{
    TApplication* myApp = new TApplication("myApp", NULL, NULL);
    TCanvas* myCanv = new TCanvas("myCanv","Titolo Canvas",0,0,700,500);
    TF1* myFun = new TF1("myFun","[0]*TMath::Exp(-x)-[1]",0,10);
    myFun->SetParameters(1,0.5);
    myFun->Draw();
    myApp->Run();
    return 0;
}
```

La classe **TApplication()** ci consente di implementare un ambiente grafico facilmente gestibile

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 myApp li gestisce tutti

# TF1: funzioni predefinite di ROOT

---

In ROOT ci sono alcune **funzioni pre-definite** pronte per essere usate nel fit:

- **gaus** Normale con 3 parametri:

$$f(x) = p_0 * \exp(-0.5 * ((x-p_1)/p_2)^2)$$

- **expo** Esponenziale con 2 parametri:

$$f(x) = \exp(p_0 + p_1 * x)$$

- **polN** Polinomio di grado N:

$$f(x) = p_0 + p_1 * x + p_2 * x^2 + \dots$$

Il loro utilizzo è immediato, non è nemmeno necessario inizializzare i parametri perché lo fa ROOT in automatico, e.g.  
`myHisto->Fit ("gaus");`

---

# TF1: Funzioni definiti dall'utente

---

E` inoltre possibile definire una qualsivoglia funzione di fit utilizzando la classe **TF1** (funzioni di una variabile)

Ci sono due modi per definire un oggetto della classe **TF1** da passare come argomento al metodo Fit:

- Scrivendo esplicitamente la funzione nella dichiarazione della **TF1**:

```
TF1* myFun = new TF1 ("myFun", "[0]*x+[1]", 0, 10);
```

- Passando al costruttore della **TF1** una funzione C++:

```
double retta (double* x, double* par)
```

```
{
```

```
    return par[0]*x[0]+par[1];
```

```
}
```

```
...
```

```
TF1* myFun = new TF1 ("myFun", retta, 0, 10, 2);
```

# TF1: alcuni metodi

---

- Inizializzazione dei parametri:

```
myFun->SetParameter(0, 0.5);  
myFun->SetParameter(1, 3.3);
```

- Impostazione del nome dei parametri:

```
myFun->SetParName(0, "a");  
myFun->SetParName(1, "b");
```

- Eseguire il fit di un istogramma (definito con un puntatore di nome myHisto):

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

- Accedere ai risultati del Fit:

```
double chi2 = myFun->GetChiSquare();  
double p0 = myFun->GetParameter(0);  
double e0 = myFun->GetParError(0);
```

...

---

# Istogrammi

Qualunque tipo di variabile, classi incluse, può essere istanziata nella memoria dinamica del calcolatore mediante l'istruzione `new`. Vediamo il caso di alcune classi di ROOT:

```
// Creazione di un istogramma

TH1D* myHisto = new TH1D("myHisto","Titolo istogramma",
                          100,-10,10);

...
myHisto->Fill(0.4);    // Fill di un istogramma
myHisto->Draw();       // Draw di un istogramma
myHisto->Fit("gaus"); // Fit di un istogramma

ecc...
```

Ovviamente quanto detto vale per qualunque tipo di classe, sia creata da voi che di ROOT (*cfr. slide 26 Lezione 5*)

# Visualizzazione degli istogrammi

In ROOT è anche possibile visualizzare a schermo istogrammi, funzioni, grafici, etc... Le classi ed i metodi di ROOT da utilizzare sono i seguenti:

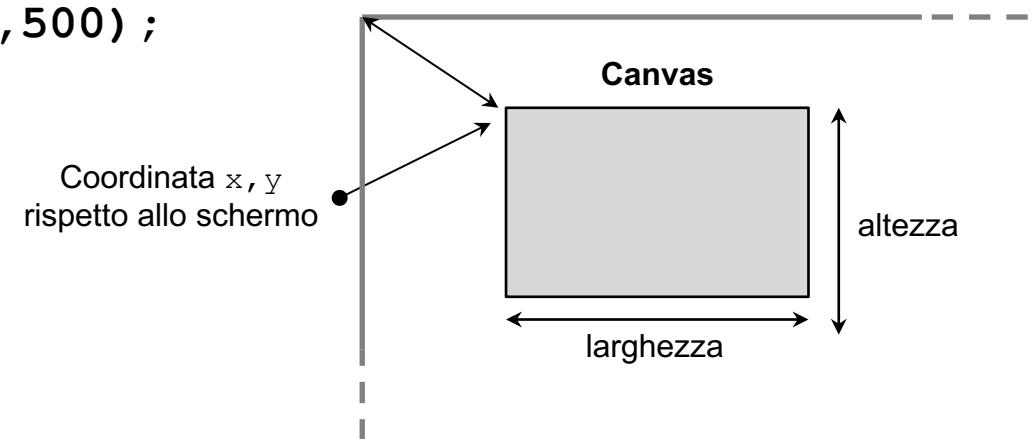
**TCanvas**: genera la finestra in cui verrà disegnato l'istogramma (funzione, grafico, etc...)

```
#include <TCanvas.h>
```

```
...
```

```
// TCanvas("nome", "titolo", x, y, larghezza, altezza)
```

```
TCanvas* myCanv = new TCanvas("myCanv","Titolo canvas",  
                           Schermo  
                           0,0,700,500);
```



# Visualizzazione degli istogrammi

## Esempio:

```
TCanvas* myCanv = new TCanvas ("myCanv", "Titolo  
canvas", 0, 0, 700, 500);  
  
TH1D* myHisto = new TH1D ("myHisto", "Titolo istogramma", 100,-  
10,10);  
  
...  
  
myCanv->cd(); // Ci si "sposta" nel canvas  
  
myHisto->Draw();  
  
myCanv->Modified(); } } // Esegue il "refresh" del canvas  
myCanv->Update(); } }  
  
...
```

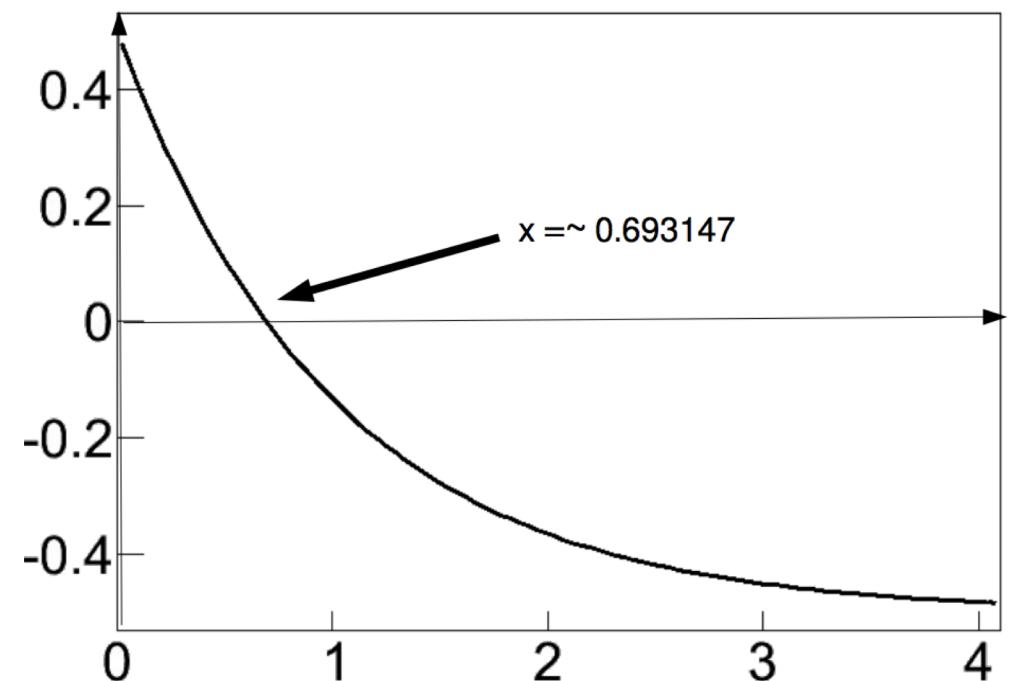
# Ricerca di zeri di funzione

È possibile utilizzare tecniche di analisi numerica per trovare gli zeri di una funzione a piacere

- Assumiamo **funzioni regolari** (continue su un compatto)
- Assumiamo che vi sia **uno e un solo zero** nell'intervallo di studio
- Va definita la precisione (distanza dello zero) a cui siamo interessati

Prendiamo come esempio la funzione  $f(x) = e^{-x} - 0.5$

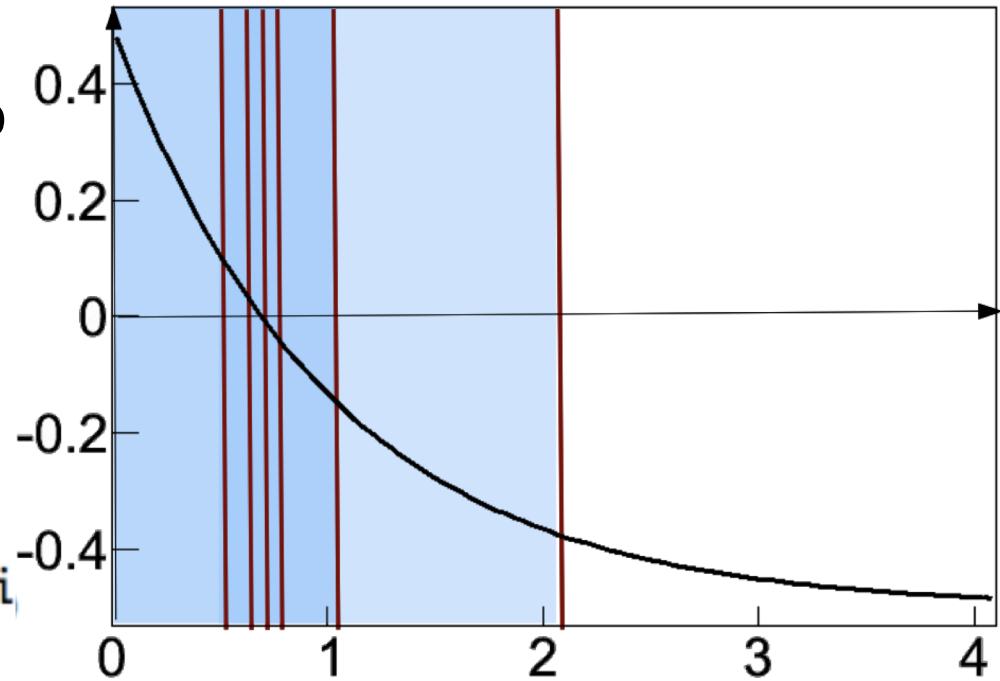
- NB: le assunzioni sopra sono per semplicità. In linea di principio non sono strettamente necessarie, ma se non sono verificate occorrerà particolare cura nell'impostare i parametri di ricerca.



# Metodo della bisezione

- Dimezziamo l'ampiezza dell'intervallo che contiene lo zero fino a quando non sia minore della precisione desiderata.
- La funzione è approssimata all'ordine 0.

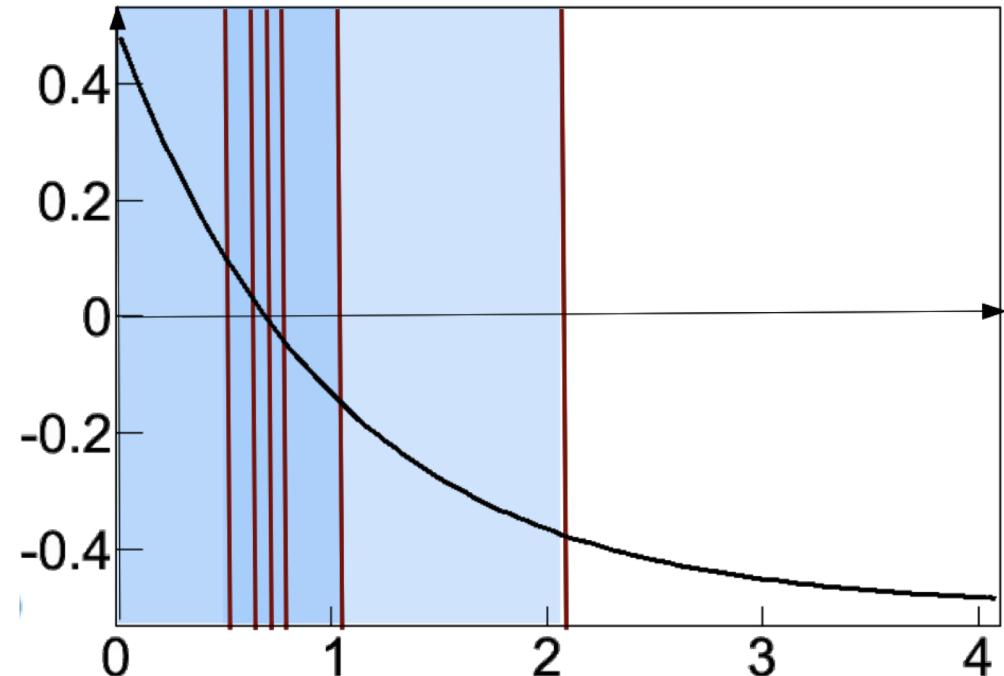
```
for (int i = 0 ; i < num_iterations ; ++i)
{
    double x_m = 0.5 * (xMin + xMax) ;
    if (f (xMin) * f (x_m) < 0)
        xMax = x_m ;
    else
        xMin = x_m ;
}
return 0.5 * (xMax + xMin) ;
```



- L'intervallo che contiene lo zero si riconosce controllando la **concordanza delle y**.
- Si restituisce il **punto medio** dell'intervallo dopo un certo numero di iterazioni.

# Metodo della bisezione ricorsiva

- Possiamo impostare l'algoritmo della bisezione in maniera ricorsiva (ovvero che richiami se stesso)
- L'algoritmo continuerà a girare fino a quando non sarà fermato da un if di controllo, dettato dalla precisione desiderata



```
double BisezioneRicorsiva (TF1* f, double min, double max, double precision) {  
    double xm = 0.5*(min+max);  
    if (max-min<precision) return xm;  
  
    if (f->Eval(xm)*f->Eval(min) < 0)  
        return BisezioneRicorsiva(f, min, xm, precision);  
    else  
        return BisezioneRicorsiva(f, xm, max, precision); }
```

# Determinazione del parametro $\mu$ dai campionamenti di una pdf esponenziale - metodo Likelihood

---

Data una distribuzione esponenziale

$$\text{pdf}(x, \mu) = 1/\mu \exp(-1 x/\mu)$$

vogliamo stimare l'unico parametro che la descrive usando il metodo della likelihood.

Si chiede quindi di scrivere un programma che simuli un esperimento in cui la pdf è campionata N volte.

Il massimo della likelihood è determinabile in modo analitico (visto a lezione) e corrisponde alla media aritmetica dei campionamenti. Il programma deve usare una tecnica numerica per:

1. determinare la pdf associata all'estimatore
2. calcolare l'intervallo di confidenza che il metodo della likelihood associa alla stima del parametro ( con il metodo della bisezione)
3. calcolare la “copertura” associata all'intervallo dato dalla likelihood

## ... in dettaglio

1. **determinare la pdf associata all'estimatore.** Si ripete l'esperimento 10000 volte (potete integrare questo conto nel punto 3) e si costruisce l'istogramma corrispondente alla media aritmetica dei campionamenti. [questa è la pdf associata alla media aritmetica di N campionamenti, in quali casi è una gaussiana ?]
2. **calcolare l'intervallo di confidenza che il metodo della likelihood associa alla stima del parametro ( con il metodo della bisezione)** si estraggono N campionamenti casuali (un esperimento) che consentono di fissare i parametri della  $\mathcal{L}$ , a quel punto si disegna la funzione  $\log \mathcal{L}$  e si determina l'intervallo che corrisponde a  $\log \mathcal{L} - 0.5$
3. **calcolare la “copertura” associata all'intervallo dato dalla likelihood** si effettuano 10000 esperimenti (ciascuno di N campionamenti) e per ognuno si ripete il punto precedente determinando quante volte l'intervallo contiene il valore vero

impostiamo un valore per  $\mu$  (sarà il valore vero della parametro) e per il numero dei campionamenti

l'estimatore è la media aritmetica, il valore atteso per l'estimatore è  $\mu$  vero, l'errore atteso sulla media aritmetica è la deviazione standard della pdf esponenziale (pari a  $\mu$ ) divisa per  $\sqrt{N}$

```
double mu=5;           //media distrib. esponenziale
int nsample;           //numero di campionamenti della pdf
cout<<"Numero di campionamenti "<<endl;
cin>>nsample;

TF1 *fpdf = new TF1("fpdf", "1/[0]*exp(-x/[0])", 0, 1000);
fpdf->SetParameter(0,mu);
TRandom1 casuale(time(NULL));
```

→ Imposto  $\mu = 5$

→ Voglio stimare  $\mu$  dato un certo numero di campionamenti nsample

→ Definisco la PDF

```
int nbin=100;
double sigma_attesa=mu/sqrt(nsample*1.);
double min=mu-5*sigma_attesa, max=mu+5*sigma_attesa;
```

→ Errore atteso  
Intervallo per visualizzazione istogramma

# PDF dell'estimatore

Verifichiamo la validità dell'approssimazione Gaussiania, calcolando **numericamente** la PDF dell'estimatore

```
//determino la pdf dell'estimatore numericamente
TH1F *hmedia=new TH1F("hmedia","pdf numerica per l'estimatore",nbin,min,max);
double sum;
int nexp=10000; //numero di volte in cui ripeto l'esperimento
for (int j=0;j<nexp;j++) {
    sum=0;
    for (int i=0;i<nsample;i++) sum+=fpdf.GetRandom(0, 1000);
    hmedia->Fill(sum/(nsample*1.));
}
hmedia->Scale(1/(nexp*1.));
hmedia->SetLineColor(1);hmedia->SetFillColor(1);hmedia->SetFillStyle(3001);

//pdf numerica dell'estimatore
c1->cd(1);
hmedia->Draw("histo E");
TF1 *fgauss = new TF1("fgauss", "gausn", min, max);
hmedia->Fit(fgauss);
fgauss->Draw("same");
TF1 *fgamma = new TF1("fgamma", "[2]*TMath::GammaDist(x,[0],0,[1])", min, max);
fgamma->SetParameters(mu*sqrt(nsample),1/sqrt(nsample),.1);
hmedia->Fit(fgamma,"LR+");
fgamma->SetLineColor(4);
fgamma->Draw("same");
...
```

Estraggo nsample numeri casuali distribuiti esponenzialmente con  $\mu = 5$  e ne faccio la media. Ripeto questa procedura 10000 volte per avere una **distribuzione di  $\mu$**

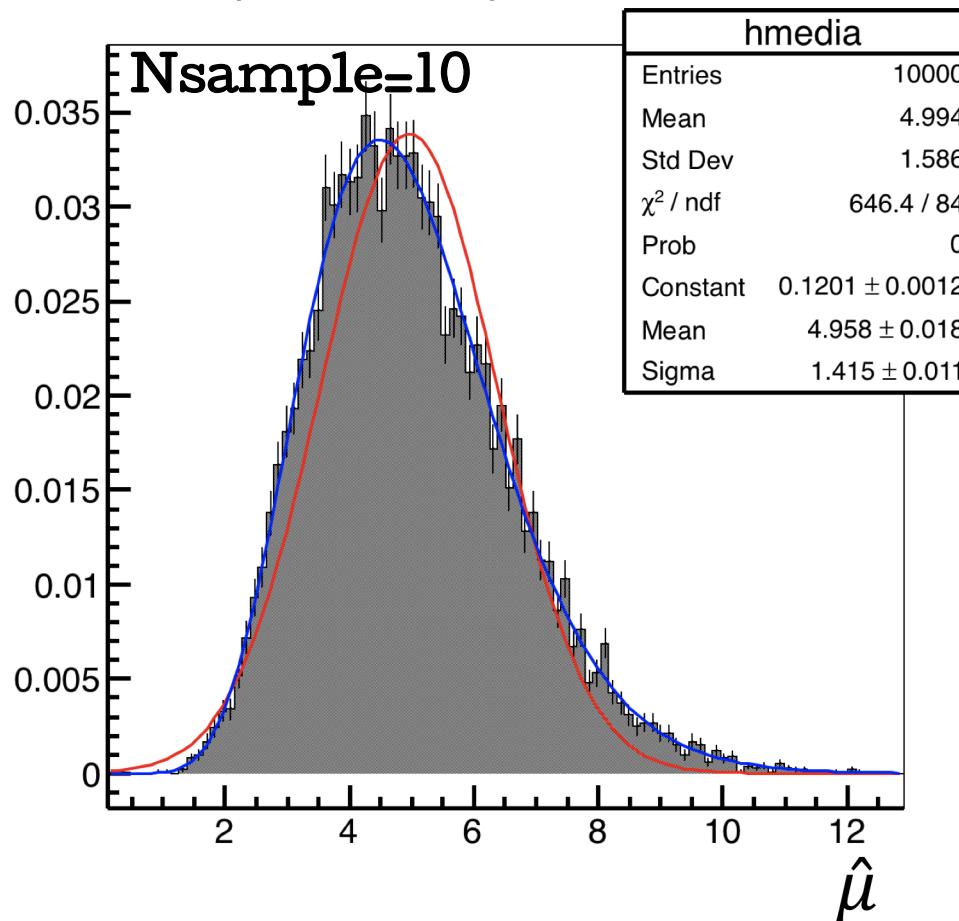
Fitto la distribuzione della media con una funzione gamma e una gaussiana.

Quale funziona meglio per piccoli nsample? E grandi nsample?

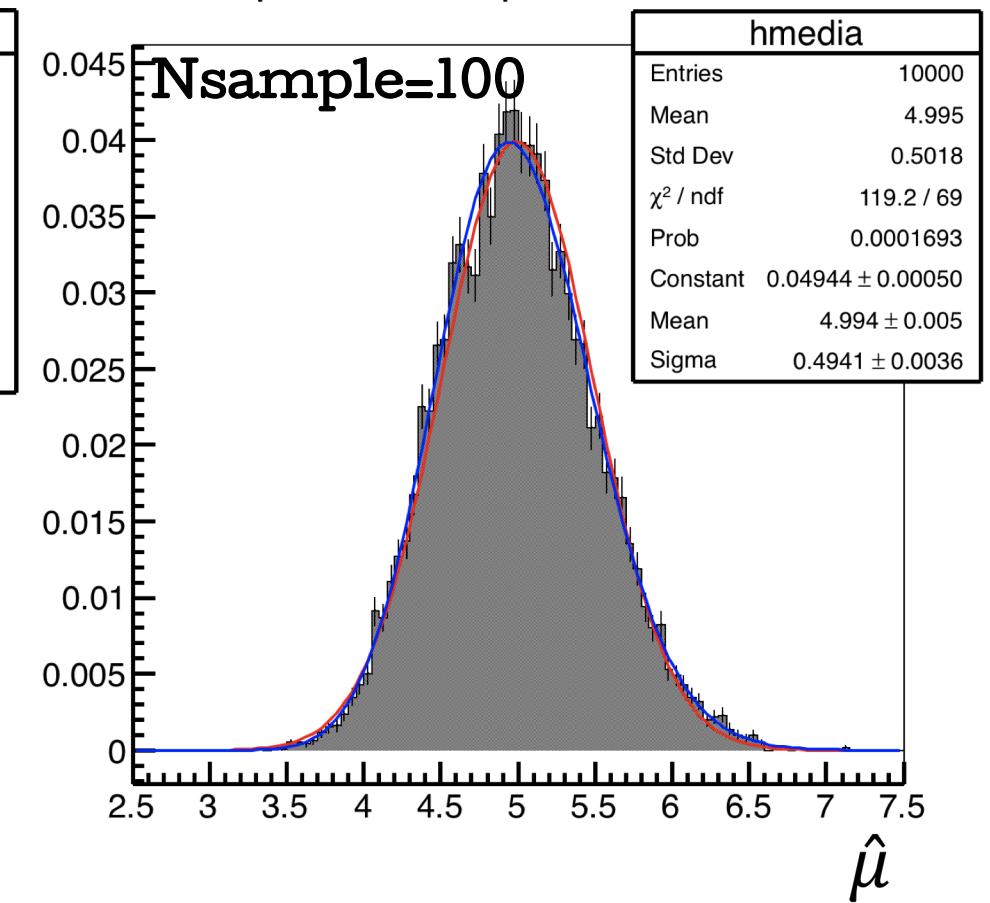
# PDF estimatore

Verifichiamo la validità dell'approssimazione Gaussiana, calcolando **numericamente** la PDF dell'estimatore

pdf numerica per l'estimatore



pdf numerica per l'estimatore



Quale funziona meglio per piccoli nsample? E grandi nsample?

# Report (1)

---

Iniziamo a riportare un po' di risultati:

```
cout<<"\nEstimatore media aritmetica (e' anche il MLE), per "<<nsample<<" campionamenti\n";
cout<<"                           Media attesa per l'estimatore = "<<mu<<endl;
cout<<"   Deviazione standard attesa per l'estimatore = "<<sigma_attesa<<endl;
cout<<" Intervallo a 1 sigma "<<mu-mu/sqrt(nsamp)<<"<mu<< mu+mu/sqrt(nsamp)<<endl<<endl;

cout<<"\n Toy MC con "<<nexp<<" esperimenti, usato per costruire la pdf dell'estimatore \n";
cout<<"   Media estimatore da Toy MC = "<<hmedia->GetMean()<<endl;
cout<<"   Deviazione standard da Toy MC = "<<hmedia->GetRMS()<<endl;
int binmin=(mu-sigma_attesa-min)/(max-min)*nbin+1;
int binmax=(mu+sigma_attesa-min)/(max-min)*nbin+1;
cout<<binmin<<endl;
cout<<"Copertura intervallo da Var[x_medio] = "<<hmedia->Integral(binmin,binmax)<<"%"<<endl<<endl;
```

# Funzione di Likelihood

Possiamo ottenere il valore centrale dell'estimatore e l'intervallo di confidenza minimizzando la funzione di likelihood

Per le proprietà dei logaritmi, minimizzare il logaritmo della likelihood o la likelihood è equivalente, ma i logaritmi sono più semplici da trattare.

Scriviamo la log-likelihood:

$$\mathcal{L} = \prod \frac{1}{\mu} e^{\frac{-x_i}{\mu}} \rightarrow \log \mathcal{L} = \sum \left( \log \frac{1}{\mu} - \frac{x_i}{\mu} \right) \cong -N \log \mu - \frac{N \hat{\mu}}{\mu}$$

```
//loglikelihood calcolata per l'ultimo set di dati campionati      x=μ
//c1->cd(2);                                                       Parametro [0] = ̂μ
double muhat;                                                       Parametro [1] = nsample
double lmin=1, lmax=20;
lmin=min, lmax=max;
TF1 *floglike = new TF1("floglike", "-[1]*TMath::Log(x)-[1]/x*[0]", lmin, lmax);
double interval[2];
```

La maximum likelihood corrisponde alla media aritmetica già calcolata prima (potete verificare calcolando  $\frac{\partial}{\partial x} \log \mathcal{L} = 0 \Rightarrow x = \hat{\mu}$  ).

Devo calcolare invece l'intervallo di confidenza del parametro.

Posso farlo trovando gli zeri della funzione floglike + 0.5 prima e dopo  $\hat{\mu}$

# Intervallo per la Likelihood

Vogliamo trovare gli zeri della funzione ( $\text{loglike} + 0.5$ ) prima e dopo  $\hat{\mu}$

```

double BisezioneRicorsiva(TF1 * func, double xMin, double xMax, double precision) {
    double x_m = 0.5 * (xMin + xMax);
    if ((xMax - xMin) < precision) return x_m;
    if (func->Eval(x_m)*func->Eval(xMin)< 0) {
        return BisezioneRicorsiva(func, xMin, x_m, precision);
    }
    return BisezioneRicorsiva(func, x_m, xMax, precision);
}

void intervallo(double muhat, int nsample, TF1 *floglike, double *interval, double lmin, double lmax) {
    floglike->SetParameters(muhat,nsample*1.);
    //floglike->Draw();
    double lkmax=floglike->Eval(muhat);
    TF1 *func = new TF1("func", "floglike+0.5-[2]", lmin, lmax);
    func->SetParameter(2,lkmax)
    interval[0]=BisezioneRicorsiva(func, lmin, muhat,.0000001);
    interval[1]=BisezioneRicorsiva(func, muhat, lmax, .0000001);
}

```

Metodo della bisezione

Applico la bisezione  
tra  $(\min; \hat{\mu})$  e  
tra  $(\hat{\mu}; \max)$   
Per identificare gli estremi

# Likelihood: copertura

Vogliamo trovare gli zeri della funzione (loglike + 0.5) prima e dopo  $\hat{\mu}$

```
//calcolo copertura dell'intervallo ottenuto da lnL-0.5
double copertura=0;
int ncicli=10000;
for (int j=0;j<ncicli;j++) {
    sum=0;
    for (int i=0;i<nsample;i++) sum+=fpdf.GetRand(0,1000);
    muhat=sum/nsample;
    intervallo(muhat, nsample, floglike, interval, lmin, lmax);
    if (interval[0]<mu && interval[1]>mu) copertura++;
}
TLine lmedia(muhat,floglike->Eval(lmin),muhat,floglike->Eval(muhat));
lmedia.SetLineColor(3);
TLine llmezzi(lmin,floglike->Eval(muhat)-0.5,lmax,floglike->Eval(muhat)-0.5);
llmezzi.SetLineColor(3);

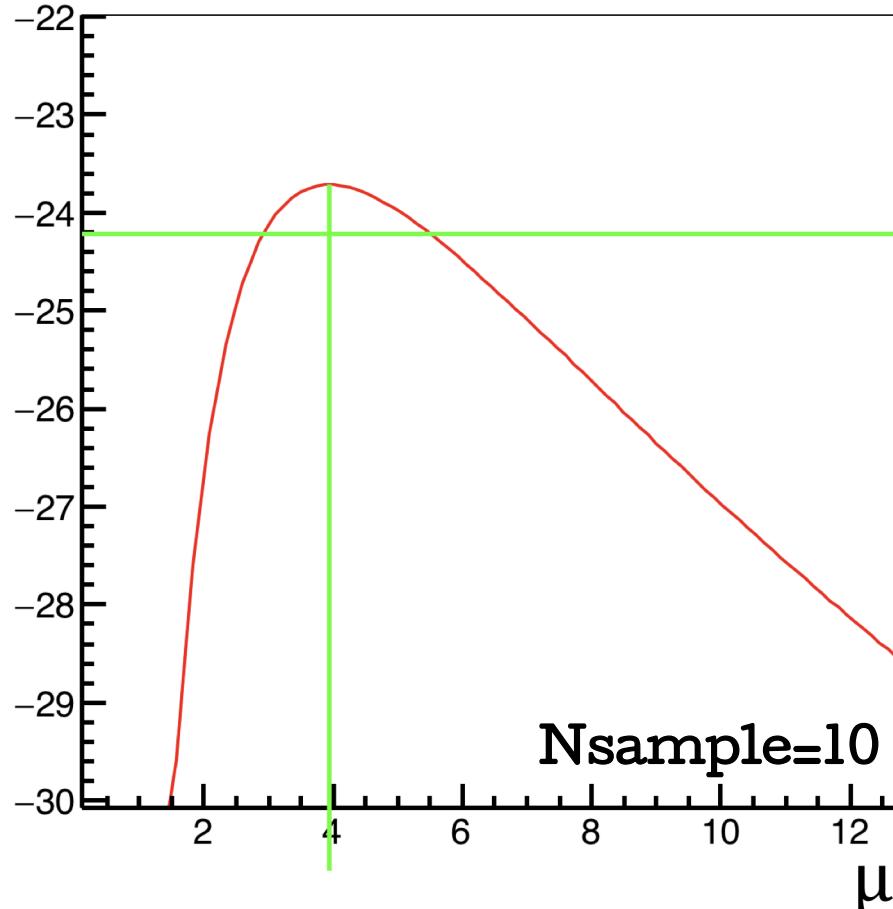
floglike->Draw("");
lmedia.Draw("same");
llmezzi.Draw("same");
```

Come è distribuita la likelihood per piccoli nsample? E grandi nsample? A cosa si deve la differenza?

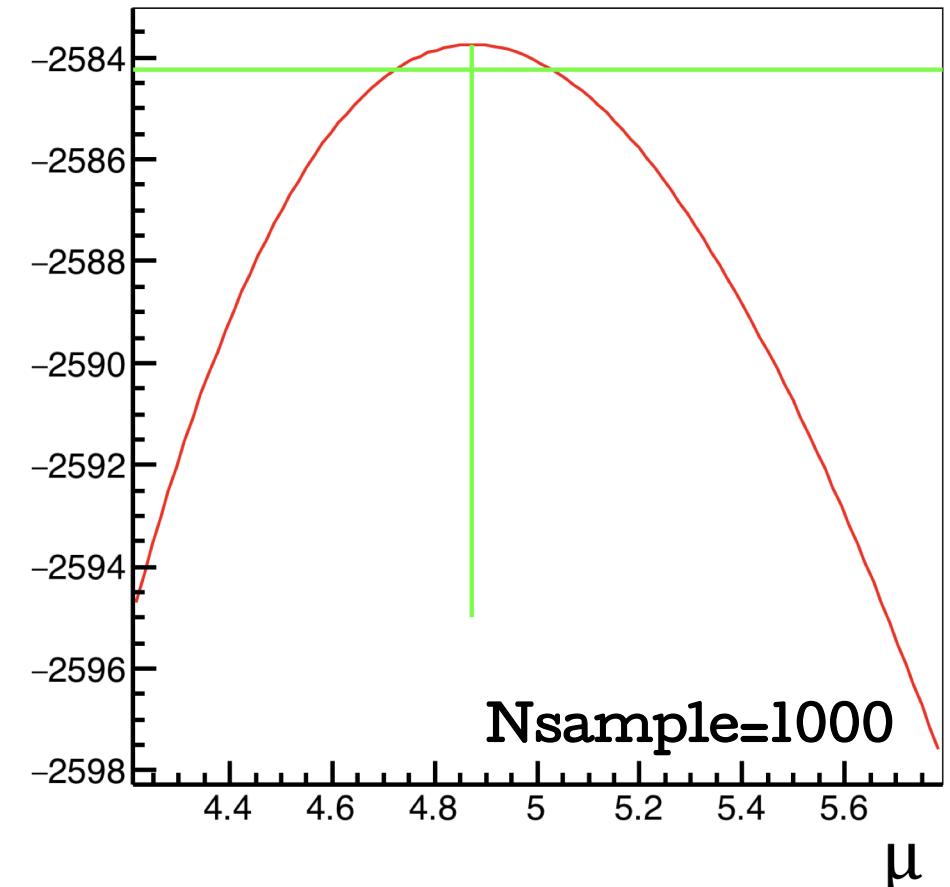
# Likelihood: distribuzione

Vogliamo trovare gli zeri della funzione ( $\text{loglike} + 0.5$ ) prima e dopo  $\hat{\mu}$

$$-[1]*\text{TMath::Log}(x)-[1]/x*[0]$$



$$-[1]*\text{TMath::Log}(x)-[1]/x*[0]$$



Come è distribuita la likelihood per piccoli nsample? E grandi nsample? A cosa si deve la differenza?

# Report (2)

---

Riportiamo i risultati ottenuti da MLE, e confrontiamoli con quelli ottenuti in precedenza:

```
cout<<"Copertura intervallo ln(L)-0.5 = "<<copertura/(ncicli*1.)<<"%"<<endl;  
  
cout<<"\nStima da un singolo data-set\n";  
cout<<"      Valore di mu da MLE = "<<muhat<<endl;  
cout<<"      Intervallo di confidenza da Likelihood+1/2 "<<interval[0]<<"<mu<"<<interval[1]<<endl;
```

# Approfondimenti

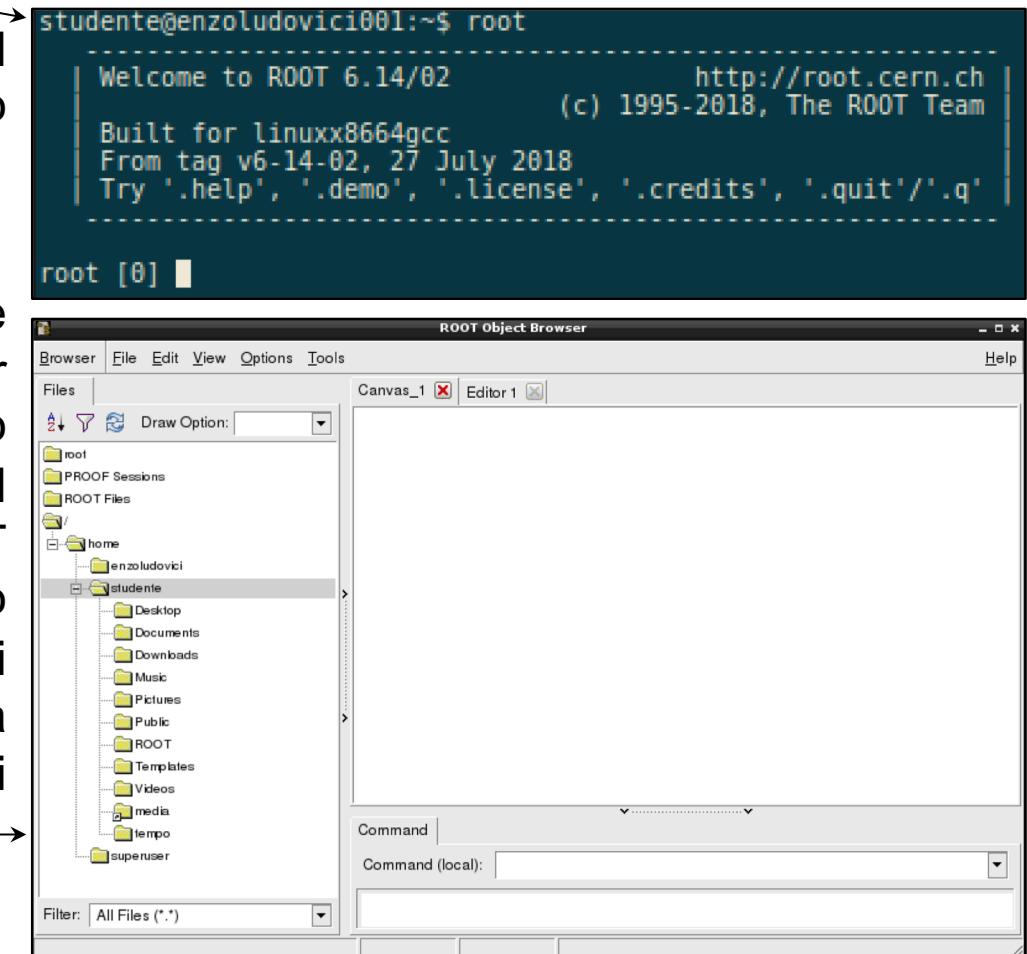
- **ROOT in interattivo**

# ROOT in interattivo

Scrivendo semplicemente `root` al *prompt* del terminale si entra nell'ambiente interattivo di ROOT. Dall'ambiente interattivo è possibile eseguire varie operazioni tra cui scrivere codice C/C++ che viene immediatamente interpretato dall'interprete C/C++ di ROOT chiamato **CINT**

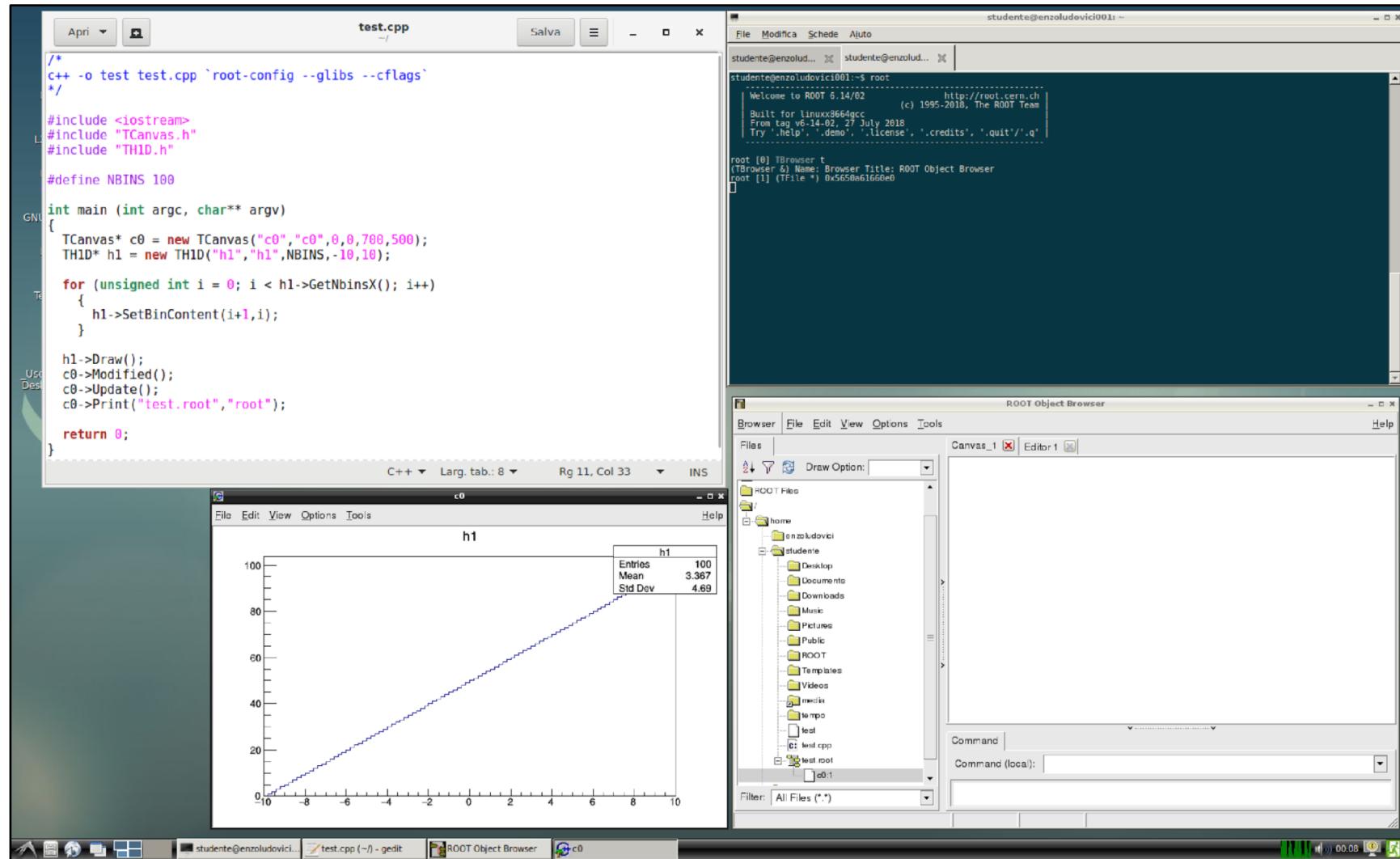
Per cominciare è possibile scrivere il comando **.help** che stampa a schermo alcuni suggerimenti di base

Tipicamente però è utile entrare nell'ambiente interattivo di ROOT per lanciare l'applicativo **TBrowser**. Questo applicativo consente di **aprire** ed **esplorare file** salvati in formato ROOT (i.e. con estensione `root`) che possono contenere anche svariati istogrammi (magari suddivisi in sotto-cartelle). Dalla riga di comando scrivere quindi **TBrowser t**



# ROOT in interattivo

Dall'interno di TBrowser è possibile aprire i file con estensione `root` semplicemente cliccando su di essi. Qui di seguito è riportato un banale esempio



# ROOT in interattivo

Da ROOT in interattivo è anche possibile eseguire degli script scritti in C++. Come vedrete dall'esempio lo script non è un programma C++ completo, mancano infatti gli include, ma CINT è in grado di capire cosa gli serve e li include automaticamente. Inoltre ROOT istanzia automaticamente una `TApplication` per gestire il `TCanvas` in interattivo

Per eseguire uno script l'istruzione è :`.x nome_del_file.C`

```
{  
    TCanvas* c0 = new TCanvas("c0","c0",0,0,700,500);  
    TH1D* h1      = new TH1D("h1","h1",100,-10,10);  
  
    for (unsigned int i = 0; i < h1->GetNbinsX(); i++)  
    {  
        h1->SetBinContent(i+1,i);  
    }  
  
    h1->Draw();  
    c0->Modified();  
    c0->Update();  
    c0->Print("test.root","root");  
}
```

```
| Welcome to ROOT 6.14/04          http://root.cern.ch |  
| (c) 1995-2018, The ROOT Team   |  
| Built for macosx64           |  
| From tags/v6-14-04@v6-14-04, Aug 23 2018, 17:00:44 |  
| Try '.help', '.demo', '.license', '.credits', '.quit'/.q' |  
  
[root [0] .x test.C  
Info in <TCanvas::SaveAs>: ROOT file test.root has been created  
root [1] ]
```

# **Analisi dei dati**

# 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**,  $f(x_1 \dots x_n; \theta_1 \dots \theta_m)$ , è una **funzione parametrica** ( $\theta_j$ ), 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

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

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

# Il metodo dei minimi quadrati

Tipicamente i funzionali, cioè i metodi di fit, sono di due tipi:

- **Minimi quadrati** (problema di minimizzazione di una certa “distanza” del modello dai dati)
- **Massima verosimiglianza** (problema di massimizzazione della probabilità, descritta dal modello, di ottenere i dati)

**Il funzionale dei minimi quadrati e`:**

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

Che espressione assume  $\sigma_i$  nel caso di un istogramma?

$n$  = numero di misure / conteggi  
 $y_i$  = misura-i / conteggio-i  
 $f()$  = modello con  $m$  parametri  $\theta_j$   
 $\sigma_i$  = errore sulla misura / conteggio

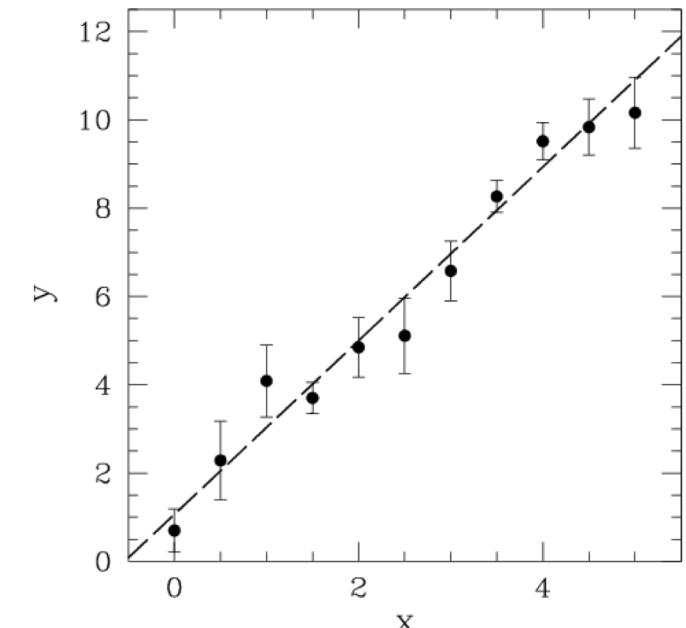
Perché proprio questa relazione matematica? → **Teorema di Gauss-Markov**: per modelli lineari nei parametri il metodo dei minimi quadrati fornisce degli stimatori lineari, *unbiased*, a varianza minima, cioè per i quali l'incertezza nella stima dei  $\theta_j$  è più piccola possibile (nella classe degli stimatori lineari e *unbiased*)

# Il metodo dei minimi quadrati

- Prendiamo per esempio un modello lineare:  $y = ax + b$ , e.g. l'andamento della velocità di un grave in caduta libera, nel vuoto, in un campo gravitazionale in funzione del tempo
- Il funzionale dei minimi quadrati avrà espressione:

$$\text{Min}^2(a, b) = \sum_{i=1}^n \left( \frac{y_i - (ax_i + b)}{\sigma_i} \right)^2$$

Dove le  $x_i$  rappresentano i diversi tempi a cui sono state effettuate le misure di velocità  $y_i$ . Le  $\sigma_i$  sono le incertezze associate alle diverse misure di velocità  $y_i$

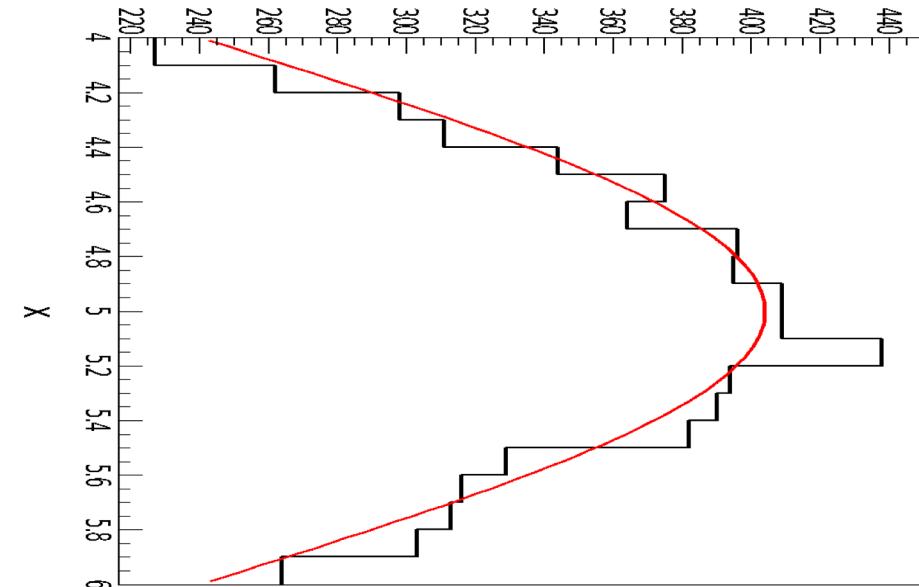


# Il metodo dei minimi quadrati

- Prendiamo per esempio un istogramma di una misura affetta da una incertezza avente distribuzione Normale
- Il funzionale dei minimi quadrati avrà espressione:

$$\text{Min}^2(A, \mu, \sigma) = \sum_{i=1}^n \left( \frac{y_i - A \cdot e^{-\frac{1}{2} \left( \frac{x_i - \mu}{\sigma} \right)^2}}{\sqrt{y_i}} \right)^2$$

Dove le  $x_i$  rappresentano i centri dei diversi bin. Le  $y_i$  sono i conteggi per bin. Le incertezze sui conteggi,  $\sigma_i$ , sono semplicemente date dalla radice quadrata dei conteggi medesimi, i.e.  $\sqrt{y_i}$



# Il metodo dei minimi quadrati

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

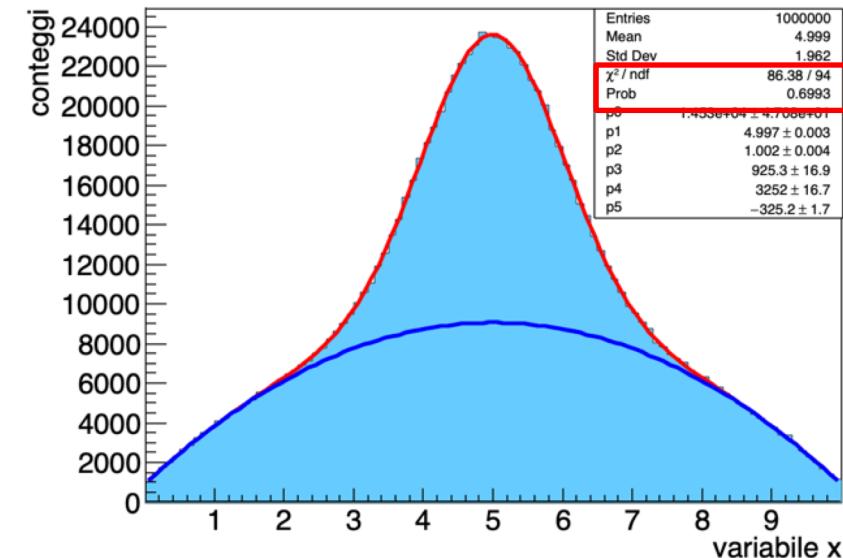
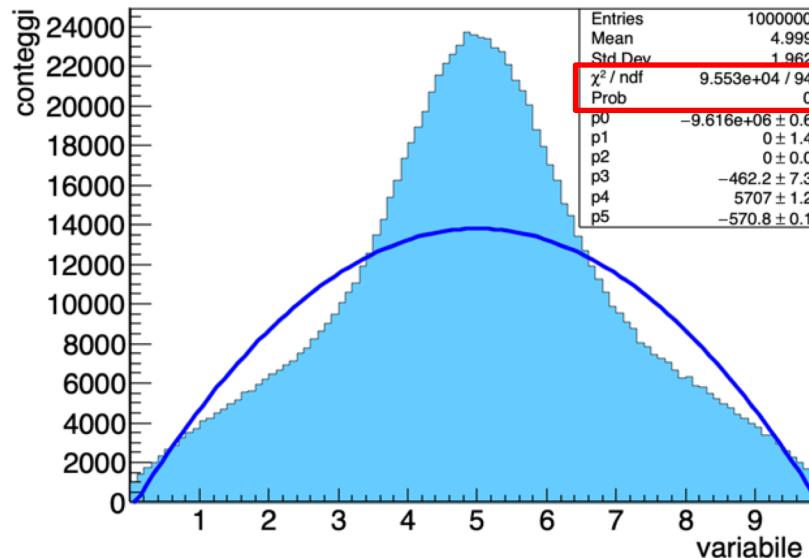
- ✓ Problema di ottimizzazione del modello rispetto ai dati (e.g. minimizzazione minimi quadrati) → miglior stima dei parametri
- ✓ Quantificare la qualità della regressione → effettuare un test statistico che sia in grado di dire quale sia la probabilità che il modello descriva i dati
- ☐ Quantificare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

Nel caso dei **minimi quadrati**, e sotto l'ipotesi di incertezze aventi distribuzione di probabilità Normale, il funzionale ha **distribuzione di probabilità  $\chi^2_{n-m}$** ,  $n$  numero di misure,  $m$  numero dei parametri del modello (per una dimostrazione si veda "Statistical Methods in Data Analysis," W.J. Metzger, capitolo 3.11)

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

# Verificare la bontà di un fit

Esempio di “cattiva” (sinistra) e “buona” (destra) descrizione dell’andamento dei dati da parte del modello



- Con metodo dei minimi quadrati è quindi possibile effettuare il **test del Chi-2** per capire se il modello fornisce una “buona” (o “cattiva”) descrizione dei dati
- Il test ci fornisce uno strumento per giudicare se le differenze tra i dati sperimentali ed il modello sono compatibili con le fluttuazioni statistiche