

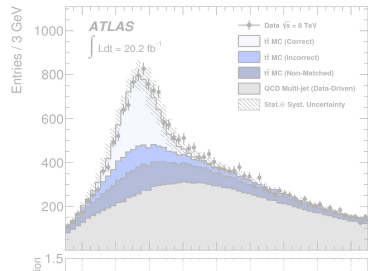


UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Analisi dati in fisica subnucleare

Speedrun degli analysis tools:
C++, programmazione a oggetti, ROOT

Alberto Orso Maria Iorio



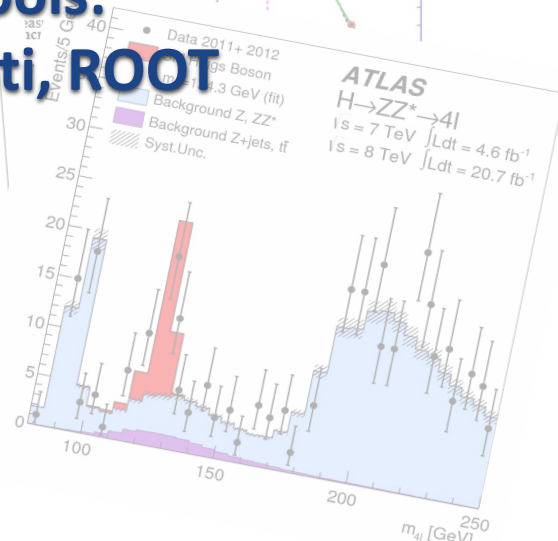
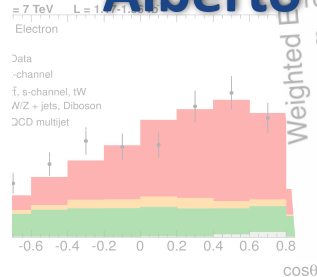
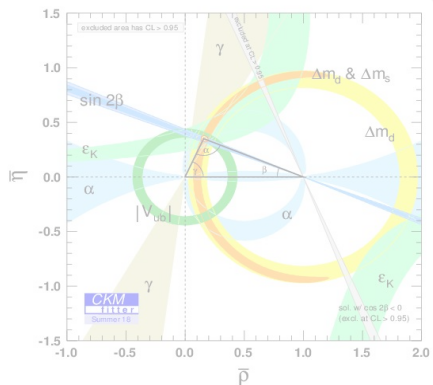
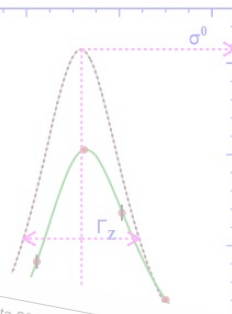
liminary

Axial-vector mediator
Dirac DM
 $g_{\gamma} = 1.0$
 $g_{DM} = 0.25$
 $g_{\gamma} = 0$

LHCP 20

40

ALEPH
OPAL



Linguaggi di programmazione: definizioni

Linguaggio di programmazione: traduce delle istruzioni da **linguaggio umano** a **linguaggio macchina**

Esistono diversi tipi di linguaggi!

Linguaggi compilati:

C/C++, julia

Linguaggi in cui scrivo una lista di operazioni e c'è un programma, **un compilatore**, che

- 1) verifica la correttezza di quanto scritto
- 2) Traduce in un con le operazioni scritte in **linguaggio macchina**
- 3) Crea tipicamente un file eseguibile per lanciare il comando

Pros: high performance, bugs perlopiù visibili

Cons: implementazione rigida, scrittura time consuming

Linguaggi di programmazione: definizioni (cont'd)

Linguaggi interpretati:

python, perl, R

Linguaggi in cui scrivo una lista di operazioni e c'è un **interprete** che

- 1) traduce il **linguaggio in operazioni**
- 2) Le esegue appena lanciato run time

Pros: si può eseguire run time, rendendo più facile la realizzazione di prototipi

Cons: performances più basse dei linguaggi compilati, bugs a volte poco chiari

Linguaggi di programmazione: definizioni (cont'd'd)

Altre possibilità:

A volte i linguaggi interpretati vengono compilati, ma non sono altrettanto efficienti di solito.

Interpreti di linguaggi compilati

Trattano il linguaggio compilato come uno interpretato, con i pros e cons che ne conseguono

Compilazione run-time:

Linguaggi permettono la scrittura di un file macchina e la sua esecuzione in catena.

Linguaggio di scripting:

Uso un linguaggio interpretato (e.g. python) per eseguire delle operazioni scritte in un linguaggio compilato (e.g. C++).

→ Tipico nelle collaborazioni - si scrive un core in C++ e ci si interfaccia con python.

Il programma Root

Root è un “**Object oriented analysis framework**”
basato su C++:

- è una collezione di librerie che mettono a disposizione strumenti utili
- Contiene un **interprete di C++**: funziona intrinsecamente da linea di comando.
- non pensato per la compilazione, ma **i suoi oggetti possono essere compilati!**

Ha degli innegabili **punti di forza**:

- **Programmazione ad oggetti**
- Accessibilità e velocità di esecuzione
- Facile da includere in altri framework
- Mantenuto e integrato costantemente

ROOT

An Object-Oriented
Data Analysis Framework



Il programma Root

Root è un “**Object oriented analysis framework**”
basato su C++:

- è una collezione di librerie che mettono a disposizione strumenti utili
- Contiene un **interprete di C++**: funziona intrinsecamente da linea di comando.
- non pensato per la compilazione, ma **i suoi oggetti possono essere compilati!**

Ha degli innegabili **punti di forza**:

- **Programmazione ad oggetti**
- Accessibilità e velocità di esecuzione
- Facile da includere in altri framework
- Mantenuto e integrato costantemente

ROOT

An Object-Oriented
Data Analysis Framework



Ma **non è la risposta a tutto**:

- Gestione della memoria poco trasparente
- Molte aggiunte necessarie
- Funzioni spesso misleading
- Induce cattive abitudini!

Getting started: apriamo ROOT come interprete

Se abbiamo svolto l'esercizio precedente, possiamo aprire root da terminale:

root

```
(base) orso@orso-VirtualBox:~$ root
```

```
-----  
| Welcome to ROOT 6.24/02                               https://root.cern |  
| (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers |  
| Built for linuxx86_64gcc on Jun 28 2021, 09:28:51 |  
| From tags/v6-24-02@v6-24-02 |  
| With |  
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.' |  
-----
```

```
root [0]
```

Da qui possiamo svolgere “live” delle operazioni di C++ grazie al compilatore

Ad esempio:

```
root [0] double a = 130.0  
(double) 130.00000  
root [1] sqrt(a)  
(double) 11.401754  
root [2] 11.401754
```

O anche cose più complicate:

```
root [6] double b[10]  
(double [10]) { 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000  
, 0.0000000, 0.0000000, 0.0000000, 0.0000000 }  
root [7] b[0]=13.3  
(double) 13.300000  
root [8] for (int i=0;i<10;++i){cout<<b[i]<<" ";}cout<<endl;  
13.3 0 0 0 0 0 0 0 0 0  
root [9]
```


La programmazione ad oggetti e le classi

Preso da qui: <https://root.cern.ch/doc/master/classTH1D.html>

Basata su classi:

La classe è una variabile che al suo interno:

Attributi (altre variabili) e **metodi** (funzioni)

Un istogramma quindi ha al suo interno

→ **Oggetti concreti**, ad es:

dei numeri caratteristici, **nome**, **titolo**, **nbins**

un vettore di bins

Altri oggetti di root (o puntatori ad essi)

→ Metodi per:

Modificare gli oggetti interni

Fare operazioni di input/output (incl. Stampare, disegnare)

Effettuare operazioni statistiche

TH1F Class Reference

Histogram Library

1-D histogram with a float per channel (see [TH1](#) documentation))

Definition at line 571 of file [TH1.h](#).

Public Member Functions

TH1F ()

Constructor. [More...](#)

TH1F (const char ***name**, const char ***title**, **Int_t** nbinsx, **Double_t** xlow, **Double_t** xup)

Create a 1-Dim histogram with fix bins of type float (see [TH1::TH1](#) for explanation of parameters) [More...](#)

TH1F (const char ***name**, const char ***title**, **Int_t** nbinsx, const **Float_t** *xbins)

Create a 1-Dim histogram with variable bins of type float (see [TH1::TH1](#) for explanation of parameters) [More...](#)

TH1F (const char ***name**, const char ***title**, **Int_t** nbinsx, const **Double_t** *xbins)

Create a 1-Dim histogram with variable bins of type float (see [TH1::TH1](#) for explanation of parameters) [More...](#)

TH1F (const **TVectorF** &**v**)

Create a histogram from a TVectorF by default the histogram name is "TVectorF" and title = "". [More...](#)

TH1F (const **TH1F** &**h1f**)

Copy Constructor. [More...](#)

Esempio di classe in root: l'istogramma TH1D

Al di là degli oggetti di C++ (float, double, int etc) possiamo accedere a tutti i tipi definiti all'interno di ROOT

Tipico esempio: l'istogramma

TH1D (istogrammi in 1 dimensione di Double)

Nota: ci sono istogrammi 1D, 2D, con diversi contenuti etc...

Per definire un istogramma

```
TH1D a = new TH1D("nome", "titolo", 100, 0, 100)
```

Nome: una "tag", un'etichetta attaccata a questo istogramma; **Titolo:** da mettere nei grafici
Terzo campo (primo 100) = **numero bins** ; **quarto campo** (0) = minimo ; quinto campo (100) = **massimo**

```
root [9] TH1D h1d("h1name", "h1 title", 100, 0, 100)
(TH1D &) Name: h1name Title: h1 title NbinsX: 100
```

Questa funzione che abbiamo definito è un **costruttore**

Classi e costruttori

Un costruttore è un metodo fondamentale:

- Ha lo stesso nome dell'oggetto
- Riempie tutti i campi utili della classe
- Esempio: da linea di comando root ci dà le opzioni disponibili
- Iniziamo a scrivere un istogramma, TH1D::TH1D (e premiamo TAB

```
root [10] TH1D::TH1D(  
TH1D TH1D()  
TH1D TH1D(const TH1D& h1d)  
TH1D TH1D(const TVectorD& v)  
TH1D TH1D(const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)  
TH1D TH1D(const char* name, const char* title, Int_t nbinsx, const Double_t* xbins)  
TH1D TH1D(const char* name, const char* title, Int_t nbinsx, const Float_t* xbins)
```

- **Nota bene:** sono funzioni diverse ma con lo stesso nome → questo è l'overloading!
- **Per vedere tutte le funzioni di una classe si può scrivere la** CLASSE : : **e usare** TAB

Istogrammi: creare un istogramma e riempirlo

Gli istogrammi possono essere riempiti con `Fill (valore)`, incrementando il bin corrispondente a `valore` di 1 unità

```
root [10] h1d.Fill(10.1)
(int) 11
```

Possiamo recuperare il contenuto bin-per-bin usando:

`FindBin (float)` : trova il bin corrispondente ad un numero

`GetBinContent (int)` : prende l'altezza del bin i-esimo

```
root [11] h1d.FindBin(10.1)
(int) 11
root [12] h1d.GetBinContent(11)
(double) 1.0000000
```

Ci sono molte operazioni utili per gli istogrammi, ad es. si possono scalare con `Scale (numero)`:

```
root [13] h1d.Scale(1.3)
root [14] h1d.GetBinContent(11)
(double) 1.3000000
```

Istogrammi: disegnare un istogramma

Gli istogrammi hanno diverse funzionalità di disegno:

```
root [15] h1d.Draw()  
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```

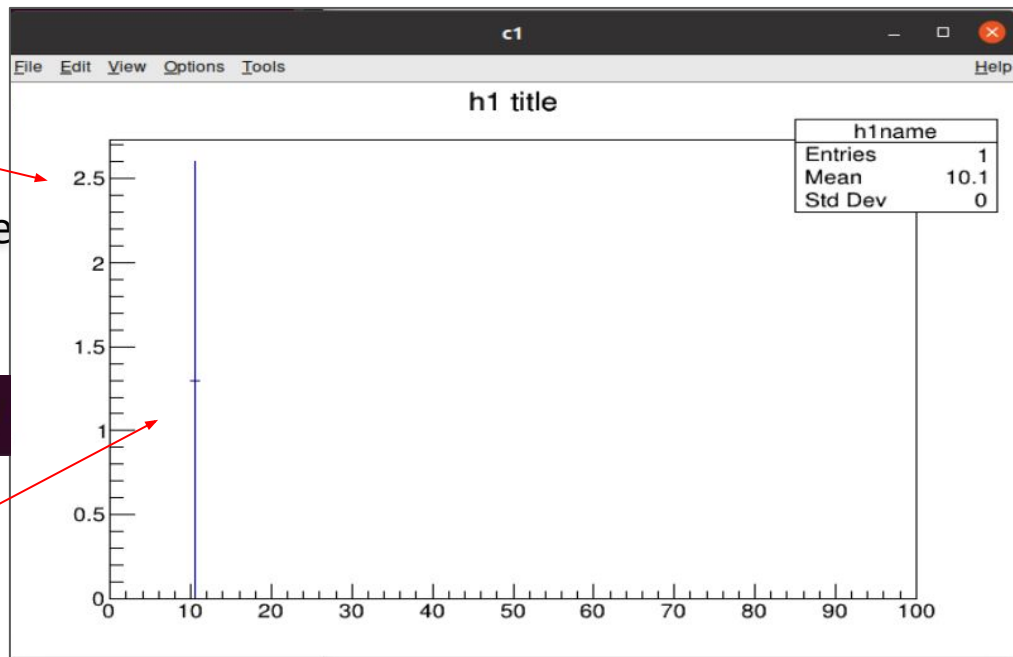
→ Disegnando un istogramma **nascerà una canvas** (oggetto: TCanvas)

→ Questa canvas può essere salvata come file in formato .png, .pdf, o molto altro

```
root [16] c1->SaveAs("h1d.png")  
Info in <TCanvas::Print>: file h1d.png has been created  
root [17]
```

Troverete il png nella cartella da cui avete lanciato root.

Domanda: che errore è questo?



Membri pubblici e privati

Non tutti i metodi sono accessibili dall'esterno

→ **Membri protected**: non accessibili dall'esterno

Protected Member Functions

virtual **Double_t** **RetrieveBinContent** (Int_t bin) const

Raw retrieval of bin content on internal data structure see convention for numbering bins in **TH1::GetBin**. More...

virtual **void** **UpdateBinContent** (Int_t bin, **Double_t** content)

Raw update of bin content on internal data structure see convention for numbering bins in **TH1::GetBin**. More...

Protected Member Functions inherited from TH1


→ **Che significa?**

```
root [17] h1d.RetrieveBinContent(11)
ROOT_prompt_17:1:5: error: 'RetrieveBinContent' is a protected member of 'TH1D'
h1d.RetrieveBinContent(11)
      ^
/home/orso/root/include/TH1.h:647:21: note: declared protected here
    virtual Double_t RetrieveBinContent(Int_t bin) const { return fArray[bin]; }
                        ^
root [18] h1d.GetBinContent(11)
(double) 1.3000000
root [19]
```

Ereditarietà

Una classe può avere delle classi derivate:

- > Esempio di implementazione nel codice :
- > Le classi figlie vedono i membri pubblici e protected delle classi da cui provengono (= Ereditano)!



```
using namespace std;

// Base class
class Shape {
public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }

protected:
    int width;
    int height;
};

// Derived class
class Rectangle: public Shape {
public:
    int getArea() {
        return (width * height);
    }
};
```

Ereditarietà

Una classe può avere delle classi derivate:

-> Esempio di implementazione nel codice :
-> Le classi figlie vedono i membri pubblici e protected delle classi da cui provengono (= Ereditano)!

-> Un istogramma 2D è “figlio” di un istogramma 1D

```
////////////////////////////////////  
//  
// TH2  
//  
// 2-Dim histogram base class.  
//  
////////////////////////////////////  
  
#include "TH1.h"  
#include "TMatrixFBasefwd.h"  
#include "TMatrixDBasefwd.h"  
  
class TProfile;  
  
class TH2 : public TH1 {  
protected:  
    Double_t      fScalefactor;    //Scale factor  
    Double_t      fTsumwy;        //Total Sum of weight*Y  
    Double_t      fTsumwy2;       //Total Sum of weight*Y*Y  
    Double_t      fTsumwxy;       //Total Sum of weight*X*Y  
  
    TH2() {
```

```
using namespace std;  
  
// Base class  
class Shape {  
public:  
    void setWidth(int w) {  
        width = w;  
    }  
    void setHeight(int h) {  
        height = h;  
    }  
  
protected:  
    int width;  
    int height;  
};  
  
// Derived class  
class Rectangle: public Shape {  
public:  
    int getArea() {  
        return (width * height);  
    }  
};
```


Ereditarietà

Gli istogrammi 1D, 2D, le funzioni varie hanno dei metodi comuni:

Protected Member Functions

virtual **Double_t** **RetrieveBinContent** (**Int_t** bin) const

Raw retrieval of bin content on internal data structure see convention for numbering bins in **TH1::GetBin**. More...

virtual **void** **UpdateBinContent** (**Int_t** bin, **Double_t** content)

Raw update of bin content on internal data structure see convention for numbering bins in **TH1::GetBin**. More...

► **Protected Member Functions inherited from TH1**

Esempi di classi sorelle:

TH**1D** (istogrammi in **1** dimensione di **Double**); TH**1F** (istogrammi in **1** dimensione di **Float**) etc

TF1 (funzioni in una dimensione) TF2 funz. In due dimensioni etc.

Altra classe utile: Funzioni

Conosciamo bene le funzioni TF1 (“nomeFunzione”, “formula”, xmin, xmax)

Ad esempio possiamo costruire una funzione del tipo:

```
TF1 mypoly("polynomial", "[0]+x*[1]", 0, 1);
```

```
root [16] TF1 mypoly("polynomial", "[0]+x*[1]", 0, 1000)
(TF1 &) Name: polynomial Title: [0]+x*[1]
```

→ dove:

mypoly è l’oggetto di tipo TF1;

polynomial è il nome dell’oggetto di tipo TF1;

x è la variabile

[0] e **[1]** sono parametri della funzione. Si possono inizializzare con:

→ **mypoly**.SetParameters(a, b)

```
root [17] mypoly.SetParameters(10, -0.008)
```

→ che darà la funzione $a+bx$

Altra cosa utile:

mypoly.Eval(x_0) valuterà la funzione nel valore x_0

Istogrammi e Funzioni

Un istogramma può “fittare” una funzione - **in realtà è la funzione che fa il fit all’istogramma:**

```
h1d-> Fit("polynomial") #Nota bene: ho usato il nome della  
funzione!
```

I parametri della funzione **possono essere fissati** (prima o dopo il fit) con:

```
mypoly->FixParameter(0)
```

L’istogramma può essere riempito casualmente secondo **una p.d.f. che viene da una funzione o un altro istogramma:**

```
h1d->FillRandom("polynomial")
```

Esempio di somma di istogrammi:

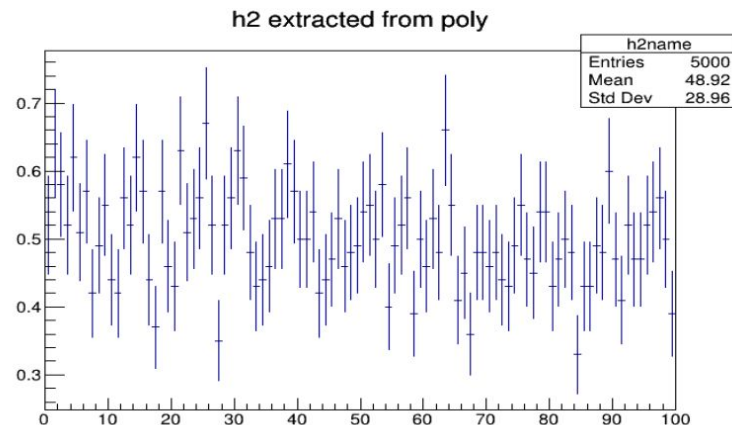
Istogrammi possono essere sommati tra di loro. Generiamo un secondo istogramma dalla polinomiale:

```
root [18] TH1D h2d("h2name","h2 extracted from poly",100,0,100)
          (TH1D &) Name: h2name Title: h2 extracted from poly NbinsX: 100
```

Con FillRandom(5000) otteniamo 5000 entries:

```
root [52] h2d.FillRandom("polynomial",5000)
root [53] h2d.Scale(0.01)
root [54] h2d.Draw()
```

Lo scaliamo per ottenere un grafico come questo →



Esempio di somma di istogrammi:

Istogrammi possono essere sommati tra di loro. Generiamo un secondo istogramma dalla polinomiale:

```
root [18] TH1D h2d("h2name","h2 extracted from poly",100,0,100)
(TH1D &) Name: h2name Title: h2 extracted from poly NbinsX: 100
```

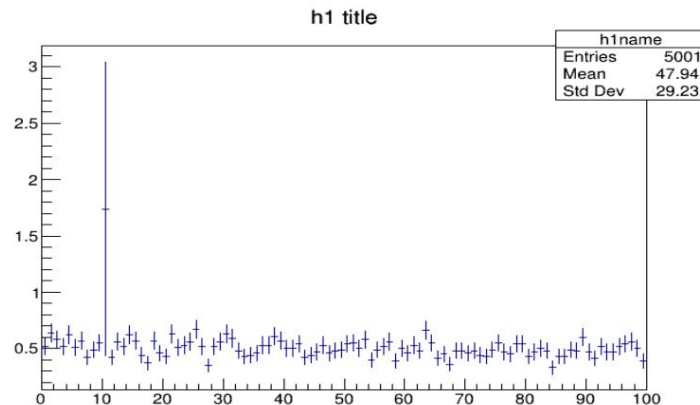
Con FillRandom(5000) otteniamo 5000 entries:

```
root [52] h2d.FillRandom("polynomial",5000)
root [53] h2d.Scale(0.01)
root [54] h2d.Draw()
```

Lo scaliamo per ottenere un grafico come questo

Sommiamo al primo, ottenendo questo

```
root [66] h1d.Add(&h2d)
(bool) true
root [67] h1d.Draw()
```



Scripting: scrivere una macro

I comandi che abbiamo usato li possiamo mettere in una macro per evitare di riscriverli, ad es

```
void Macro_example(){
    TCanvas c1("c1");
    TH1D * h1d = new TH1D ("h1d","h1d title",100,0,100);
    //Nota: affinché gli istogrammi persistano è necessario fare puntatori #rootthings
    h1d->Fill(10.1);
    h1d->Scale(1.3);
    TH1D * h2d = new TH1D("h2d","h2d extracted from poly",100,0,100);
```

...etc. Esempio salvato qui:

https://github.com/oiorio/AnalisiDati/blob/main/3.%20Data%20handling%20in%20root/macro_example.C

Posso eseguire una macro dall'esterno:

```
root -l macro.C
```

Oppure da root

```
(base) orso@orso-VirtualBox:~/macros$ root -l
root [0] .x macro_example.C
Info in <TCanvas::Print>: file Sum_Example.png has been created
```

Nelle prossime lezioni vedremo come compilare/fare cose più civili, per ora good enough!

Esercizio di riscaldamento #1:

E2.1.1) Proviamo a scrivere una TF1 gaussiana tra 0 e 1000 con valori che vogliamo noi di media e dev.st..

E2.1.2) Proviamo a riempire un istogramma prendendo casualmente 100 punti dalla gaussiana.

Esercizio di riscaldamento #2:

E2.2) Usiamo ifstream per prendere i numeri dal file di testo.

Prendiamo i dati salvati qui:

<https://github.com/oiorio/AnalisiDati/tree/main/3. Data handling in root/data>

Usando l'esempio qui:

https://github.com/oiorio/AnalisiDati/blob/main/3.%20Data%20handling%20in%20root/macro_getfromfile.C

```
{  
#include <iostream.h>  
#include <TH1.h>  
    ifstream f("exercise_1_Likelihood_gr1.txt");  
  
    float x=0.;  
    while(!f.eof()){  
        f>>x;  
        cout << " x is " <<x<<endl;  
    }  
}
```

E ora facciamo qualcosa di più divertente...

E2.3.1) Scriviamo una PDF “extended” su x (0 - 1000).

→ Definiamo una funzione:

$$\frac{e^{-(s+b)}}{N!} \prod_{i=1}^N sP_s(x_i; s, \Theta) + bP_b(x_i; b, \Theta)$$

con **s** eventi di segnale, **b** eventi di fondo e distribuzione P_s gaussiana, P_b di fondo esponenziale

2.3.2) Valutiamola in ogni punto del dataset preso.

2.3.3) Facciamo un **likelihood scan** dei parametri s , b e della gaussiana fissando i parametri a quelli forniti.

2.3.4) Facciamo un **profile likelihood** scan del parametro s lasciando gli altri parametri floating e presi dal metodo “fit” della funzione.

Nota Bene: questa esercitazione non è solo per oggi! Non fuggite urlando!