Corso di Laboratorio di Fisica

III anno LT in Fisica

Analisi dati con ROOT

A.A. 2018-2019

Alberto Garfagnini, Marcello Lunardon

Analisi dati con ROOT

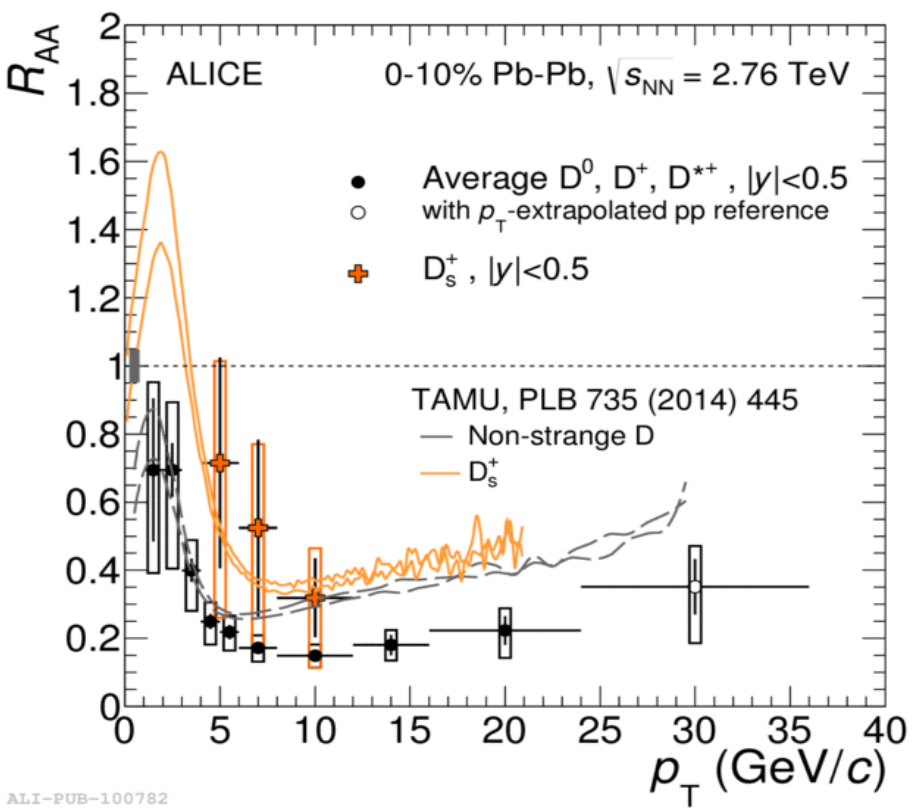
Prima parte

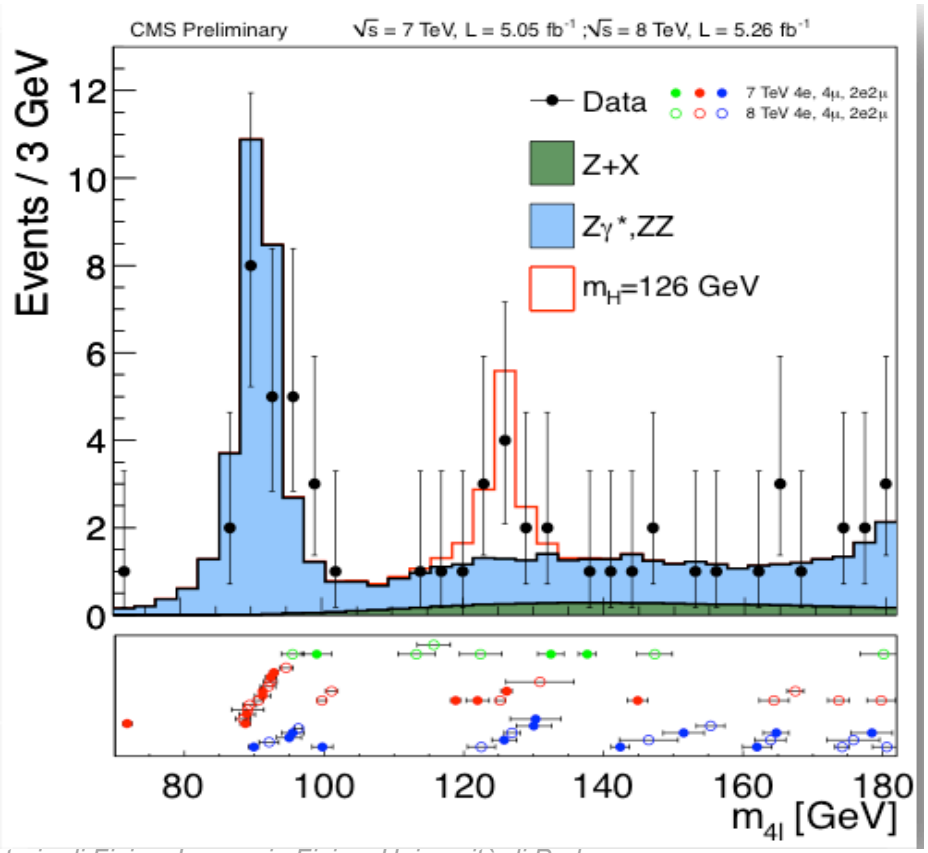
Cos'è ROOT e a cosa serve

Analisi dati con ROOT

Cos'è ROOT

- pacchetto di librerie in C++ per l'analisi dei dati in Fisica sviluppato e mantenuto dal CERN. E' il software ufficiale della maggior parte degli esperimenti a di fisica delle particelle (acceleratori, ricerca di materia oscura, fisica del neutrino, …)

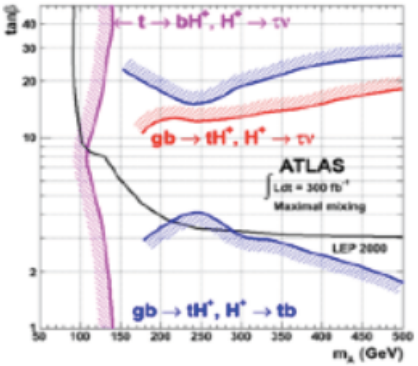


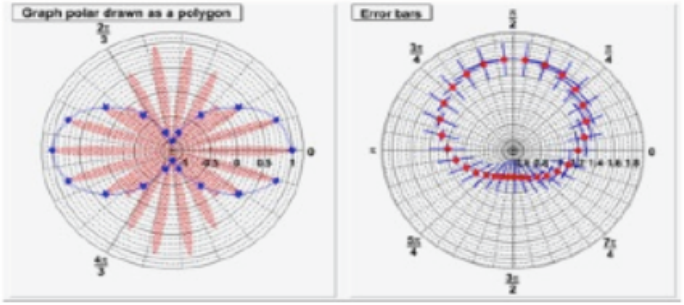


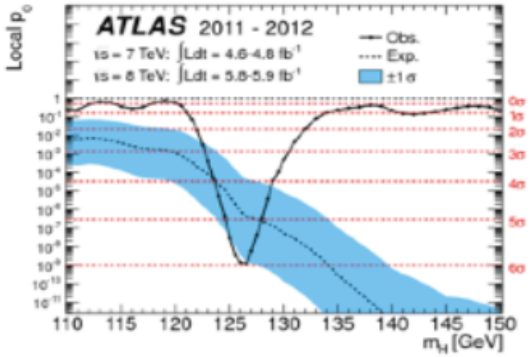
Analisi dati con ROOT

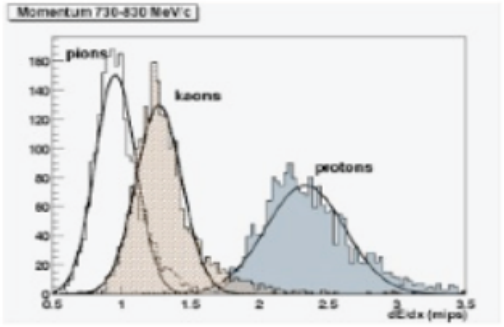
Cos'è ROOT

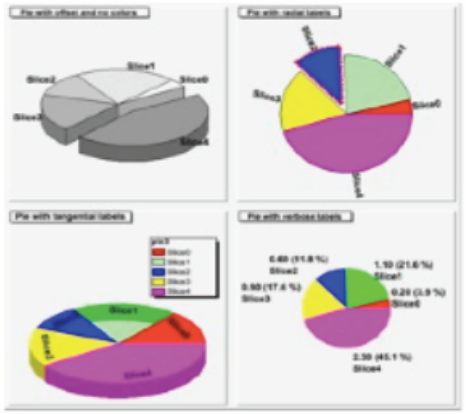
- sono **disponibili tutti i principali oggetti per l'analisi**: *grafici, istogrammi, funzioni, strutture dati complesse, oggetti grafici* per la rappresentazione dei dati, *strutture geometriche* per la simulazione degli apparati...

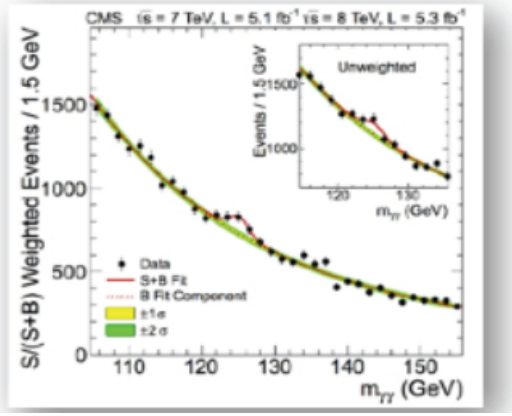










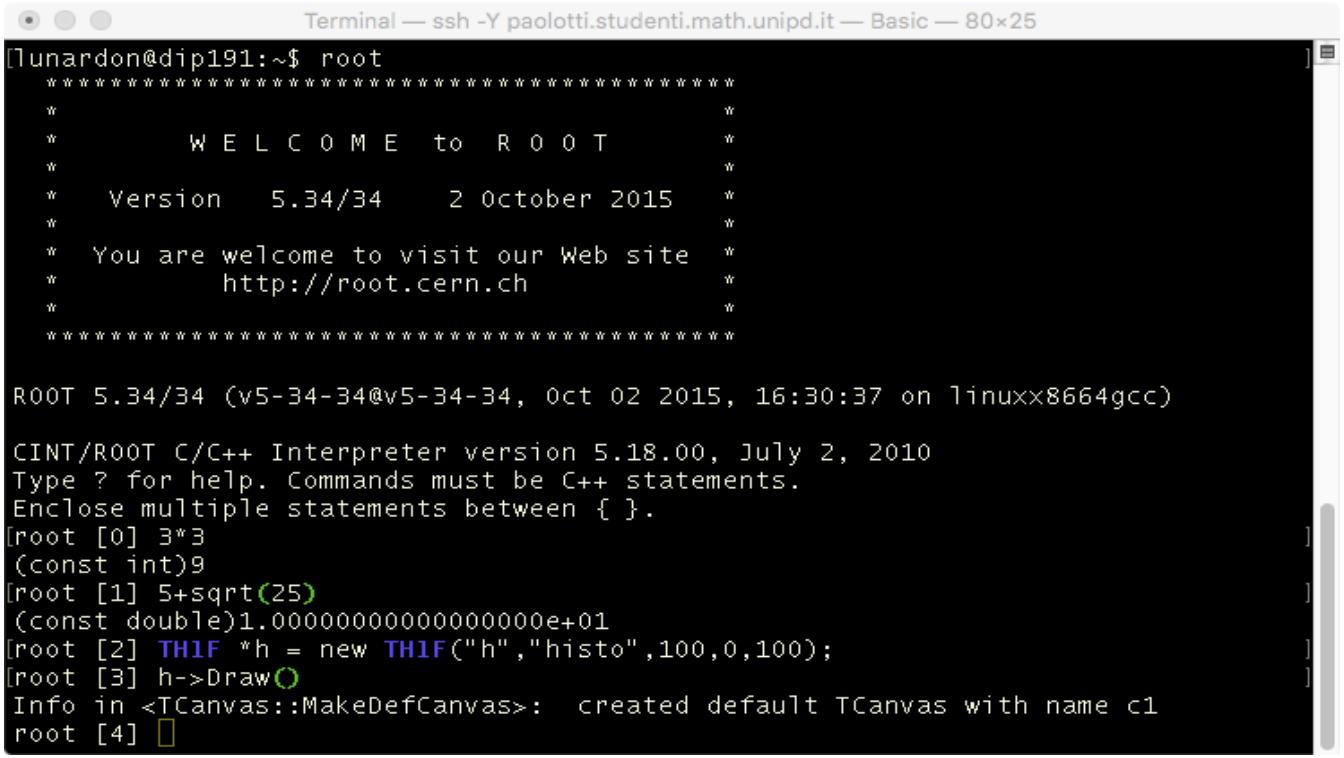


Analisi dati con ROOT

Cos'è ROOT

- è dotato di un **interprete C++** (CINT/CLING) per l'esecuzione di istruzioni "al volo" e semplici programmi di analisi (***macro***).

- è anche interfacciabile direttamente con Python

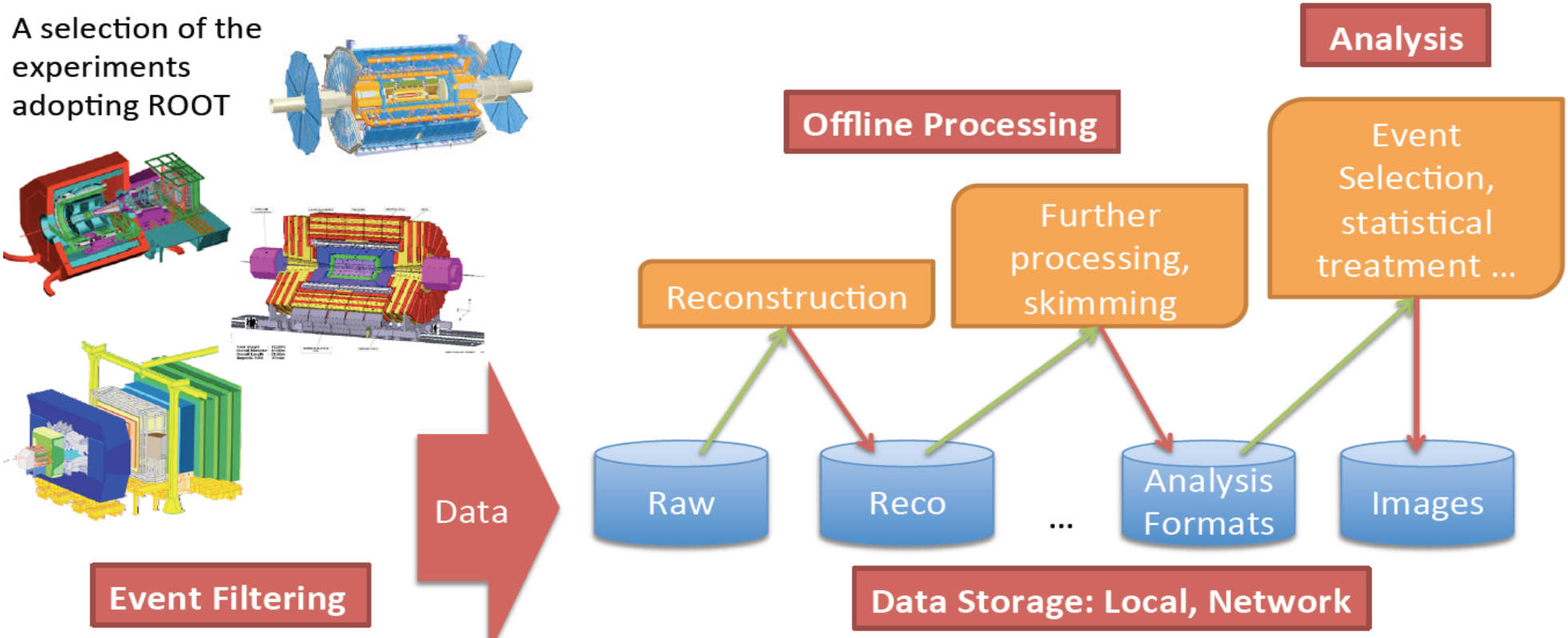


Analisi dati con ROOT

Cos'è ROOT

- software **open-source**, completamente **customizzabile** (*ALICE -> aliroot = root + librerie di analisi dell'esperimento*)

- grazie al **linguaggio standard C++** è di facile integrazione con i sistemi di acquisizione (DAQ) degli esperimenti



Analisi dati con ROOT

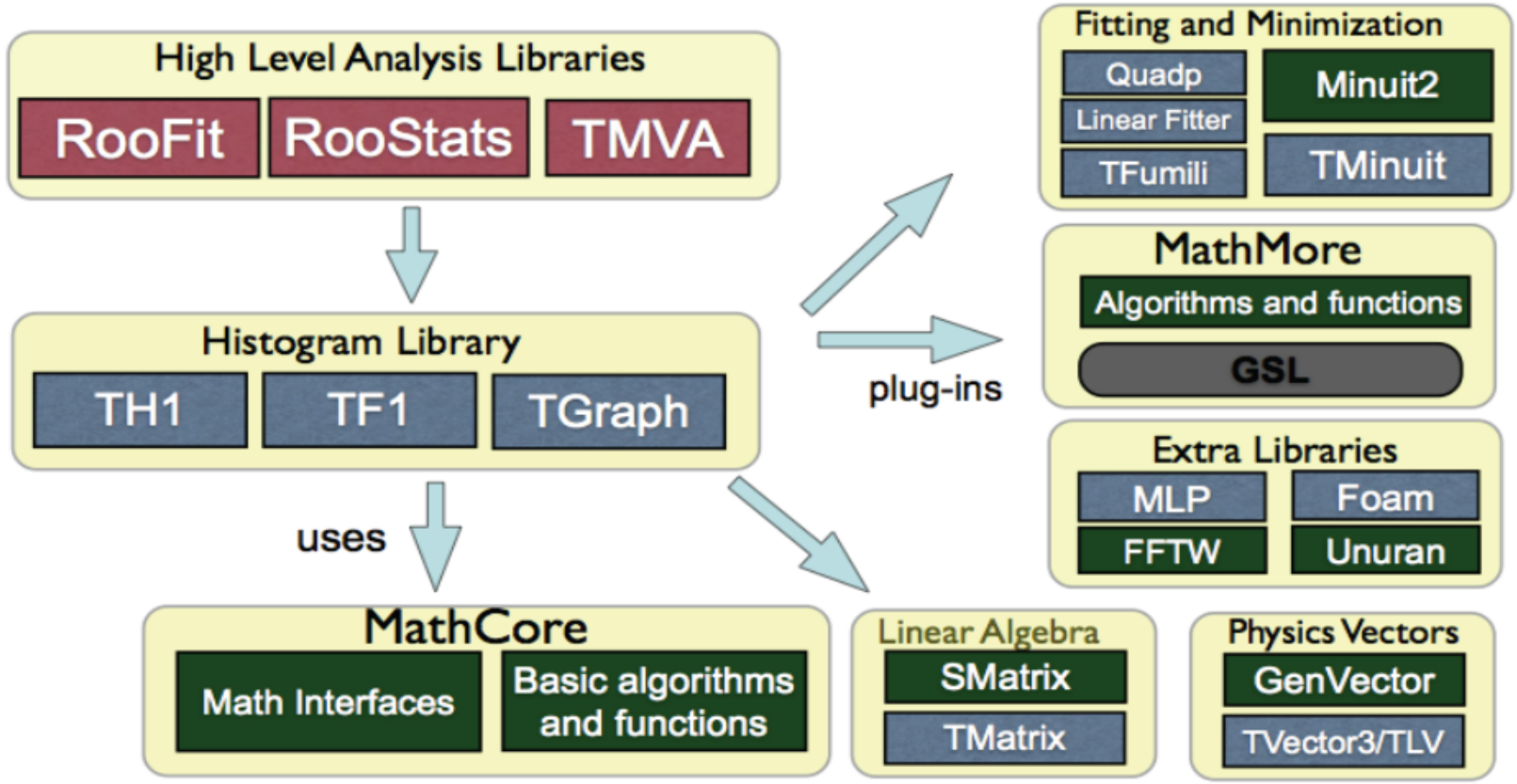
Cos'è ROOT

- E' particolarmente adeguato per la **gestione di grandi quantità di dati** di fisica come quelle acquisite dai moderni esperimenti (-> interfacciato con GRID), ma molto utile anche per la gestione dei dati in piccoli o medi esperimenti.

Analisi dati con ROOT

Cos'è ROOT

- ROOT offre un gran numero di **librerie e strumenti matematici** per l'analisi statistica avanzata

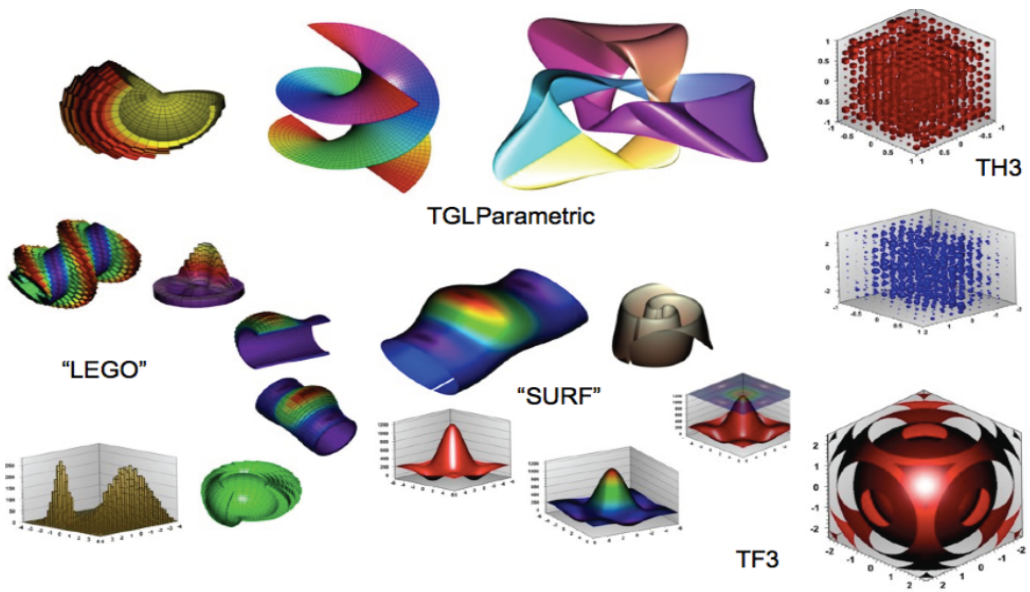


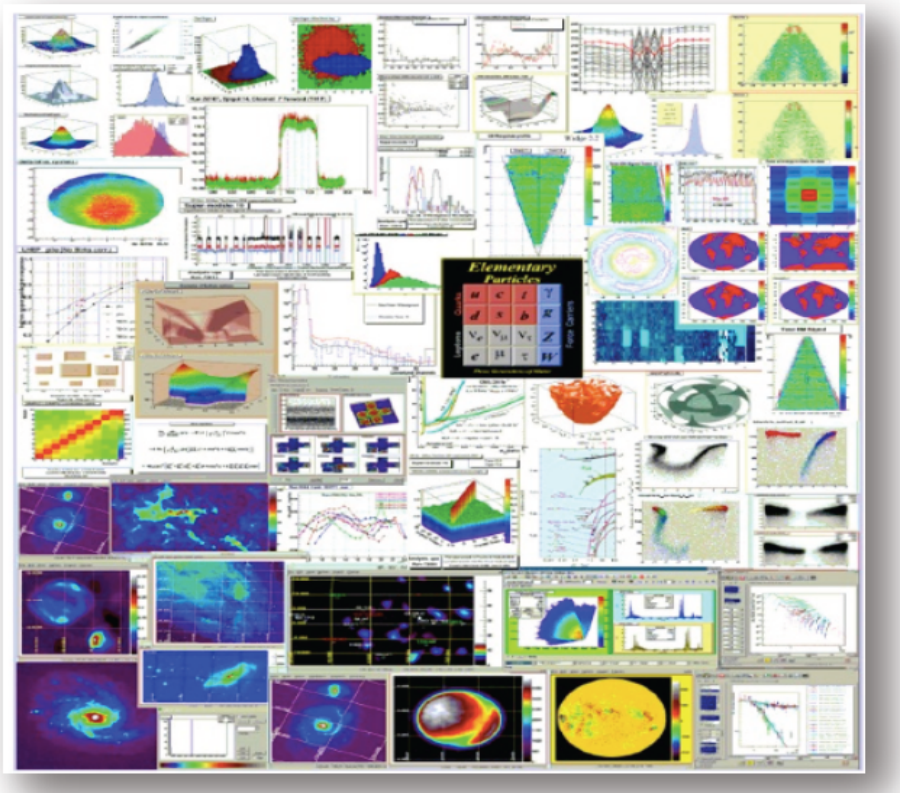
Analisi dati con ROOT

Cos'è ROOT

- ROOT è in **continuo aggiornamento**
- il progetto, iniziato negli anni ’90, è ora alla versione 6 (*last stable release 6.14/06, del 5-Nov-2018*). Noi in laboratorio abbiamo la 6.10/08.

- molte nuove funzionalità per la rappresentazione dei dati in 1, 2 e 3 dimensioni





Analisi dati con ROOT

Perchè ROOT ?

- ROOT è ***un*** tool per l'analisi statistica dei dati.
- Usato a Lab. Fisica LT e Lab. Fisica LM.
- Scritto da fisici per i fisici: particolare attenzione per gli oggetti e i metodi per l'analisi in fisica, inclusa la rappresentazione dei dati.
- Sono disponibili diversi altri programmi di analisi statistica, qualcuno ottimizzato in ambiti diversi, altri molto generali...
  - *MATLAB*
  - *Mathematica*
  - *R*
  - *SAS*
  - *Origin*
  - *...*

Analisi dati con ROOT

Risorse online:

Molte informazioni online (qualche volta anche troppe... ☺)

=> ROOT main page: **https://root.cern.ch//**

ROOT
Data Analysis Framework

Google Custom Search

Download Documentation News Support About Development Contribute

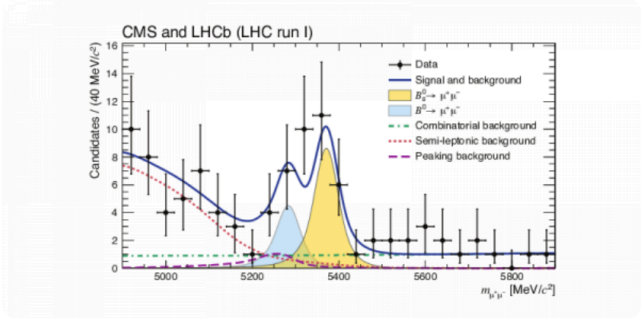
Getting Started | Reference Guide | Forum | Gallery

ROOT is ...

A modular scientific software toolkit. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

Start from examples or try it in your browser!

Download ROOT or Read More ...



Previous Pause Next

Under the Spotlight

2018-01-17 **ROOT Users' Workshop 2018**

The ROOT team would like to invite you to the 11th ROOT Users' Workshop.

2017-08-03 **The ROOT Docker container (alpha version)**

Do you like Docker? Would you like to use ROOT? We provide an *alpha* version of the ROOT Docker container!

2016-09-05 **Get the most out of the ROOT tutorials!**

Other News

2018-07-11 RDataFrame session at CHEP 2018

2018-06-15 ROOT::RWhy!

2017-03-08 Development release 6.09/02 is out!

2016-04-16 The status of reflection in C++

Latest Releases

Release 6.14/06 - 2018-11-05

Analisi dati con ROOT

Risorse online

- Download ROOT : https://root.cern.ch/downloading-root
  - -> Release 6.14/06 – 2018-11-05
  - -> Release 5.34/38 – 2018-03-12
    - Source, Binary distributions…

- Getting started: https://root.cern.ch/getting-started
  - ROOT Primer (https://root.cern.ch/guides/primer)
  - Code Examples
  - ROOT Tutorial for Summer Students
    by D. Piparo and O. Couet
    https://indico.cern.ch/event/395198/

- Reference Guide : https://root.cern.ch/doc/v614/
  -> 6.14/06, 5.34/38, ...

Analisi dati con ROOT

Installazione

- Metodo raccomandato per sistemi unix-based (linux, osx,...) : *installing from source -> https://root.cern.ch/building-root*
  - *check prerequisites*
  - *Quick Start using Cmake*
- Oppure
  - *unpack the source archive*
  - *configure with script*
  - *make / install*
    - *standard installation (qualche directory visibile a tutti, tipo /usr/local)*
    - *user installation (-> set environmental variables)*
      - *ROOTSYS*
      - *PATH*
      - *LD_LIBRARY_PATH*
      - *source with # source bin/thisroot.sh*
- Metodo alternativo, più veloce:

  installing from precompiled binaries, available for most OS

Analisi dati con ROOT

Installazione

- Run ROOT in Docker containers : *installing from source*
  ->https://hub.docker.com/r/rootproject/root-ubuntu16/

MAC OSX:

```
1) start Xquartz
$ open -a XQuartz
2) check the local ip (enX identifies the interface)
$ ip=$(ifconfig enX | grep inet | awk '$1=="inet" {print $2}')
3) authorize the ip to get X connections
$ xhost + $ip
4) start the docker container
docker run --rm -it -e DISPLAY=$ip:0 -v /mac_osx:/data \
           rootproject/root-ubuntu16 /bin/bash
```

Analisi dati con ROOT

Gli oggetti di ROOT

Classi:

- è una struttura di memoria complessa che può contenere sia dati (*data members*) che funzioni (*class methods*);
- i dati e le funzioni possono essere pubblici o privati. Dall'esterno sono accessibili sono i contenuti pubblici;
- spesso le classi usate sono derivate da classi via via più generiche;
- struttura e codice di tutte le classi di ROOT sono accessibili nella Reference Guide.

```
class TH1F : public TH1, public TArrayF {
public:
   TH1F();
   TH1F(const char *name,const char *title,Int_t nbinsx,Double_t xlow,Double_t xup);
   ....
   virtual ~TH1F();
   ....
   virtual void     SetBinContent(Int_t bin, Double_t content);
};
```

Analisi dati con ROOT

Gli oggetti di ROOT

Oggetti:

- sono delle variabili di *tipo classe xyz;*
- sono creati da un metodo speciale (il *costruttore*) che ha lo stesso nome della classe;
- In genere ci sono più di un costruttore per una classe (cioè è possibile istanziare l'oggetto in più modi diversi...);
- La grande maggioranza degli oggetti di ROOT ha un "nome" che è una keyword che permette all'interprete di riconoscerlo all'interno della shell interattiva

```
TH1F h; // oggetto istogramma 1D con valori float
TH1F h2("name","title",100,0,1000); // costruttore con parametri iniziali
TH1F h3(h); // costruttore che crea una copia di h
```

Analisi dati con ROOT

Gli oggetti di ROOT

Oggetti:

- si possono creare direttamente come oggetti (*TH1F h;*)
  oppure utilizzando i puntatori (*TH1F *h = new TH1F;*)
- Nel primo caso si accede ai contenuti attraverso il punto (*h.Draw();*), nel secondo con la freccetta (*h->Draw();*)
- nella ROOT shell è disponibile il *tab-completion*, molto utile anche come help on-line...

Esempi:

```
TH1F *h = new TH1F(”h",”htitle",100,0,1000); // crea “h”
for (int i=0; i<1000; i++) h->Fill(val[i]) // riempie con i dati
h->Draw(); // disegna nella finestra grafica (canvas)
TH1F *h2 = (TH1F*) h->Clone(“h2”); // crea una copia
h2->SetName(“h2”); h2->SetLineColor(2); h2->Add(hbkg, -1); // modifica
h2->Draw(“same”); // disegna nella stessa canvas
```

Analisi dati con ROOT

Grafici, funzioni, istogrammi, strutture dati

- Sono i principali oggetti utilizzati nell'analisi dei dati

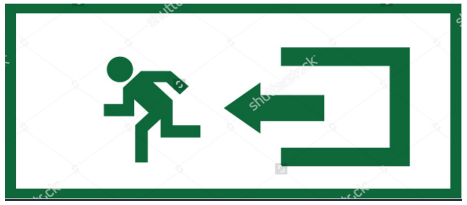
- Le classi principali sono:
  - TGraph, TGraphErrors, TGraph2D... grafici a 1 e 2 dimensioni
    https://root.cern.ch/doc/master/classTGraph.html
  - TF1, TF2, funzioni parametriche di 1 o 2 variabili
    https://root.cern.ch/doc/master/classTF1.html
  - TH1(S/I/F/D), TH2, TH3... istogrammi a 1 o più dimensioni
    https://root.cern.ch/doc/master/TH1F.html
  - TTree, TNtuple, ... strutture dati complesse
    https://root.cern.ch/doc/master/TTree.html
  - TFile file per memorizzare dati e risultati dell'analisi
    https://root.cern.ch/doc/master/TFile.html

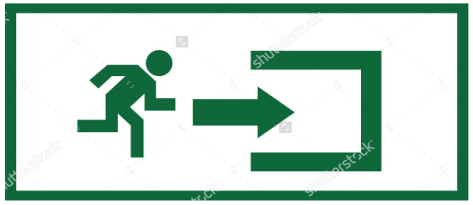
Analisi dati con ROOT

Troubleshooting

- Durante l'analisi dati è normale incorrere in qualche crash del programma, soprattutto se si lavora a lungo all'interno della shell o con gli strumenti di editing grafico...

- Qualche piccolo consiglio:
  - **usare le macro** : si può ricostruire l'analisi in ogni momento
  - quando si lavora in grafica, **salvare frequentemente** la canvas
  - **chiudere e riaprire la sessione interattiva** quando si può ...





Analisi dati con ROOT

Prima di iniziare...

1. loggarsi col proprio account e aprire un terminale
2. spostarsi nella directory di lavoro “labfis”

   # cd labfis
3. connettersi alla mia pagina del corso e scaricarsi il **materiale per oggi**:
   1. **ROOTintro_AA1819.pdf** : presentazione aggiornata
   2. le macro e dati per gli esempi di oggi
4. creare una sottodirectory data e copiare alcuni files:

   # mkdir data

   # cp BraggAnalysis2.C bragg

   # cp ch4_500mb_thr96.root bragg
5. *Lanciare root*

Analisi dati con ROOT

Seconda parte

- **Alcune funzionalità di base per partire**
- **I principali oggetti di ROOT per l'analisi dei dati in Laboratorio**

Analisi dati con ROOT

Esercizio 1 : ROOT come calcolatrice

- Apriamo una sessione interattiva di ROOT (ROOT shell):

```
# docker run –rm –it -e DISPLAY=localhost:0 \
   rootproject/root-ubuntu16
```

Proviamo qualche semplice operazione tipo calcolatrice:

```
alberto — docker run --rm -it -e DISPLAY=localhost:0 rootproject/root-ubuntu...
Last login: Thu Dec 21 14:51:30 on ttys004
➜  ~ docker run --rm -it -e DISPLAY=localhost:0  rootproject/root-ubuntu16
   ------------------------------------------------------------------
  | Welcome to ROOT 6.11/03                        http://root.cern.ch |
  |                                (c) 1995-2017, The ROOT Team      |
  | Built for linuxx8664gcc                                          |
  | From heads/master@v6-11-02-814-gc896bbf, Nov 15 2017, 17:09:00   |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
   ------------------------------------------------------------------

[root [0] 1+1
 (int) 2
[root [1] 1+1.5
 (double) 2.5
[root [2] sqrt(12) + sqrt(27.5)
 (double) 8.70815
[root [3] 2 * TMath::Pi() * 4.5
 (double) 28.2743
 root [4]
```

Analisi dati con ROOT

Esercizio 1 : ROOT come calcolatrice

- Nella shell possiamo anche definire variabili e inserire righe di codice in linguaggio C/C++

```
alberto — docker run --rm -it -e DISPLAY=localhost:0 rootproject/root-ubuntu...
[→  ~ docker run --rm -it -e DISPLAY=localhost:0  rootproject/root-ubuntu16
   ------------------------------------------------------------------
  | Welcome to ROOT 6.11/03                        http://root.cern.ch |
  |                                  (c) 1995-2017, The ROOT Team |
  | Built for linuxx8664gcc                                         |
  | From heads/master@v6-11-02-814-gc896bbf, Nov 15 2017, 17:09:00 |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'      |
   ------------------------------------------------------------------

[root [0] 1+1
 (int) 2
[root [1] 1+1.5
 (double) 2.5
[root [2]
[root [2] double x = 1.2
 (double) 1.2
[root [3] double y = TMath::Pi()*x
 (double) 3.76991
[root [4] double z = 0
 (double) 0
[root [5] for (int i=0; i<10; i++) z += x*i;
[root [6] z
 (double) 54
 root [7]
```

Analisi dati con ROOT

Esercizio 1 : i comandi al prompt

- la shell riceve un certo numero di comandi che iniziano con “.”

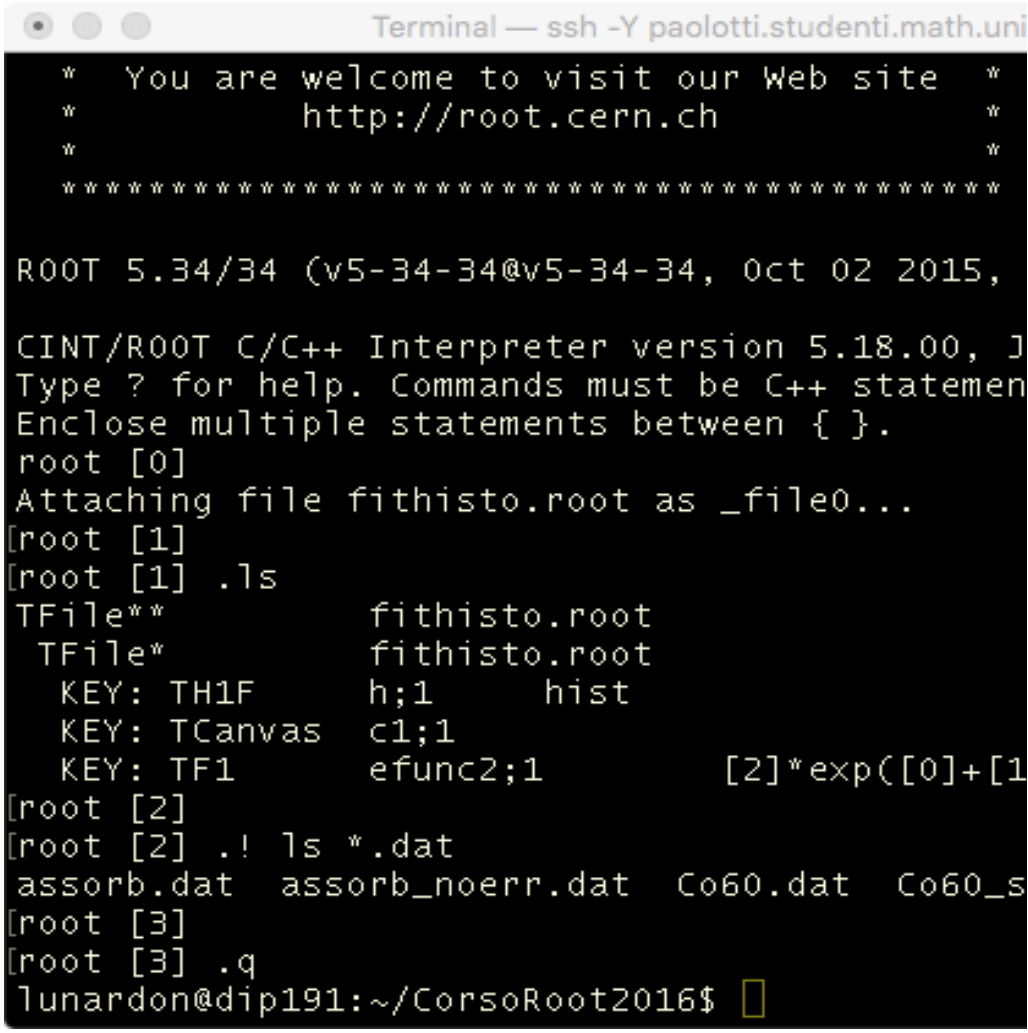
.demo : lancia tutorial demo

.q : esce

.! <OS command>

.help : full list

- è attivo il tab-completion

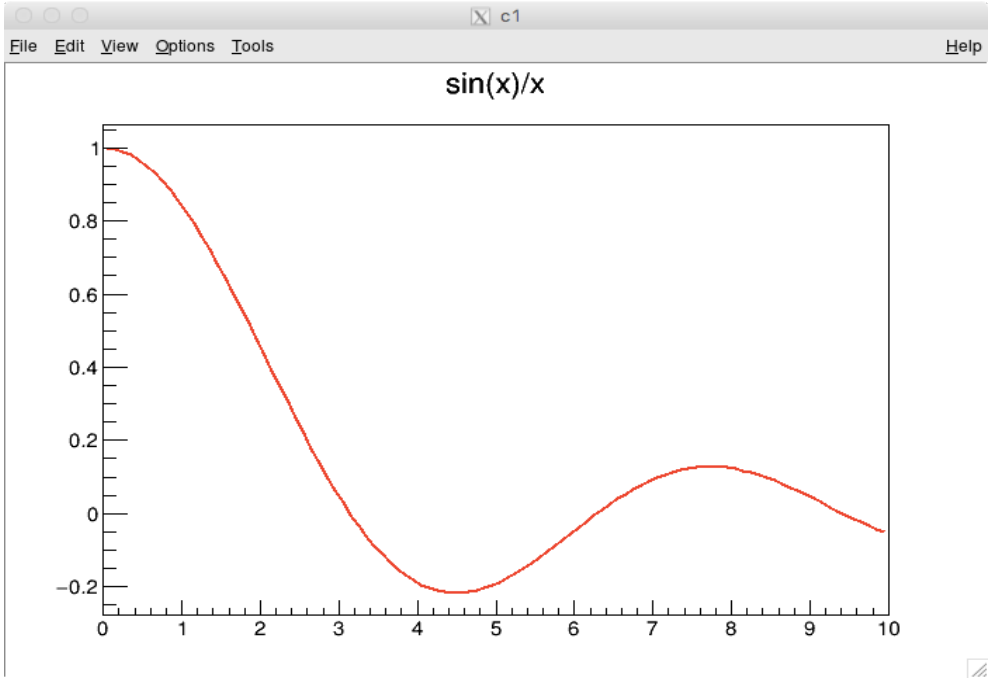


Analisi dati con ROOT

Esercizio 1 : ROOT per disegnare una funzione

- definiamo ora una funzione da riga di comando e disegnamola

```
[root [0] TF1 f1("f1","sin(x)/x",0.,10.);
[root [1] f1.Draw()
 Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```



- *forma alternativa con la sintassi dei puntatori:*

```
[root [0] TF1 *f1 = new TF1("f1","sin(x)/x",0.,10.);
[root [1] f1->Draw()
 Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

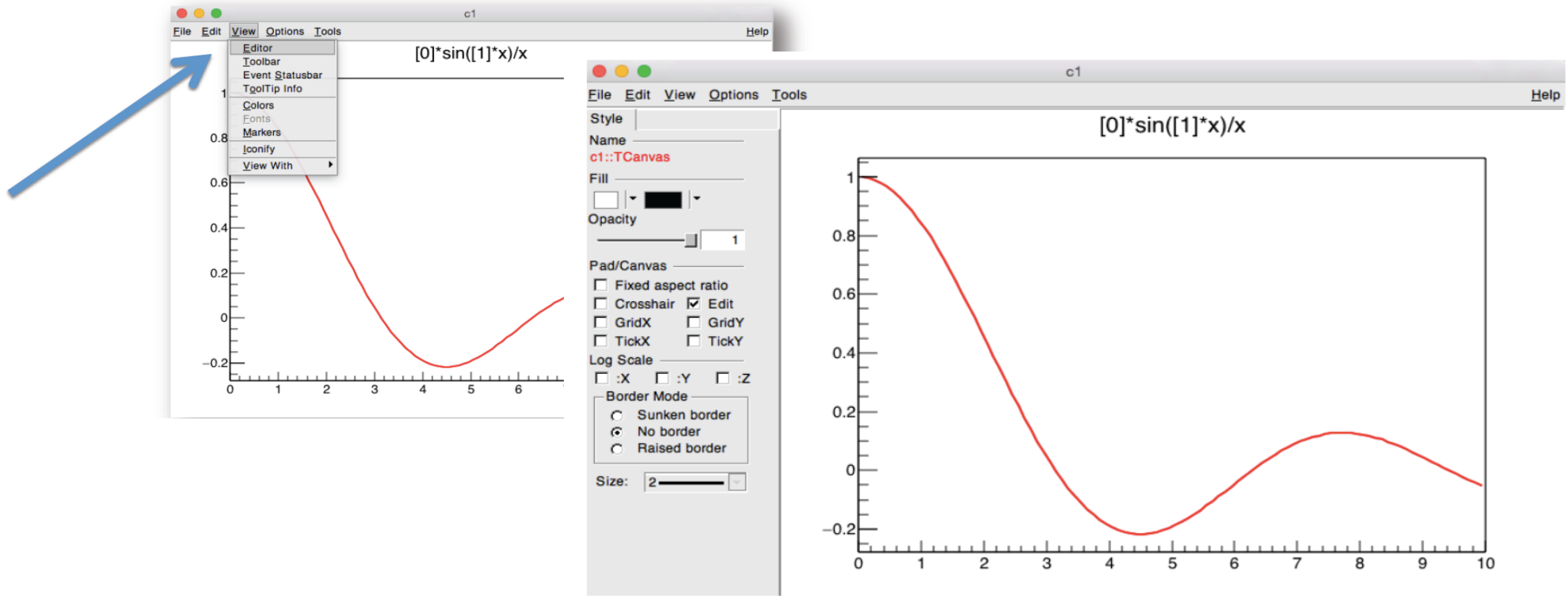
Analisi dati con ROOT

Esercizio 1 : ROOT per disegnare una funzione

- usiamo ora una forma più generale

```
[root [2] TF1 f2("f2","[0]*sin([1]*x)/x",0.,10.);
[root [3] f2.SetParameters(1,1);
[root [4] f2.Draw();
```

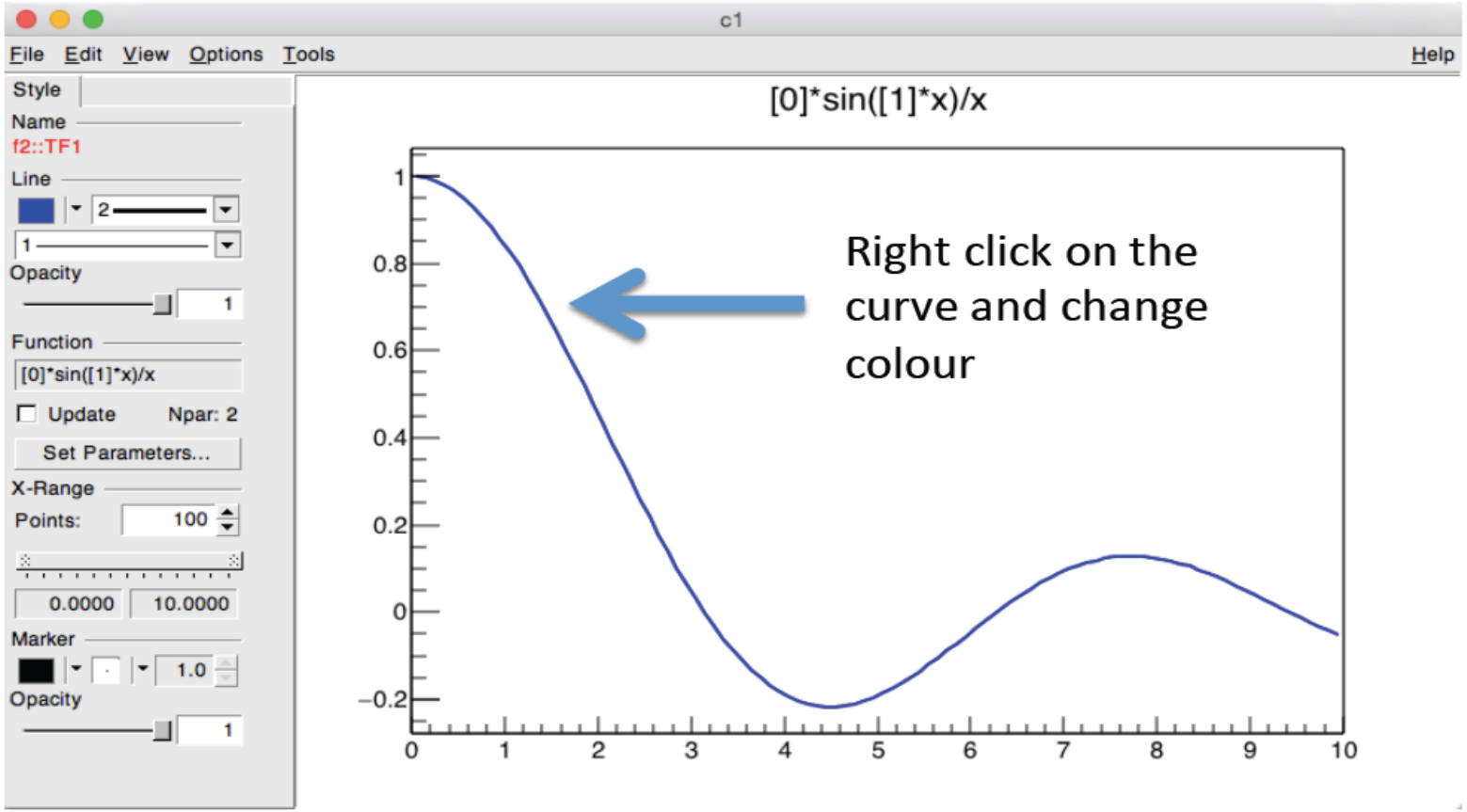
- e usiamo gli strumenti grafici per modificare il disegno (***Editor***)



Analisi dati con ROOT

Esercizio 1 : ROOT per disegnare una funzione

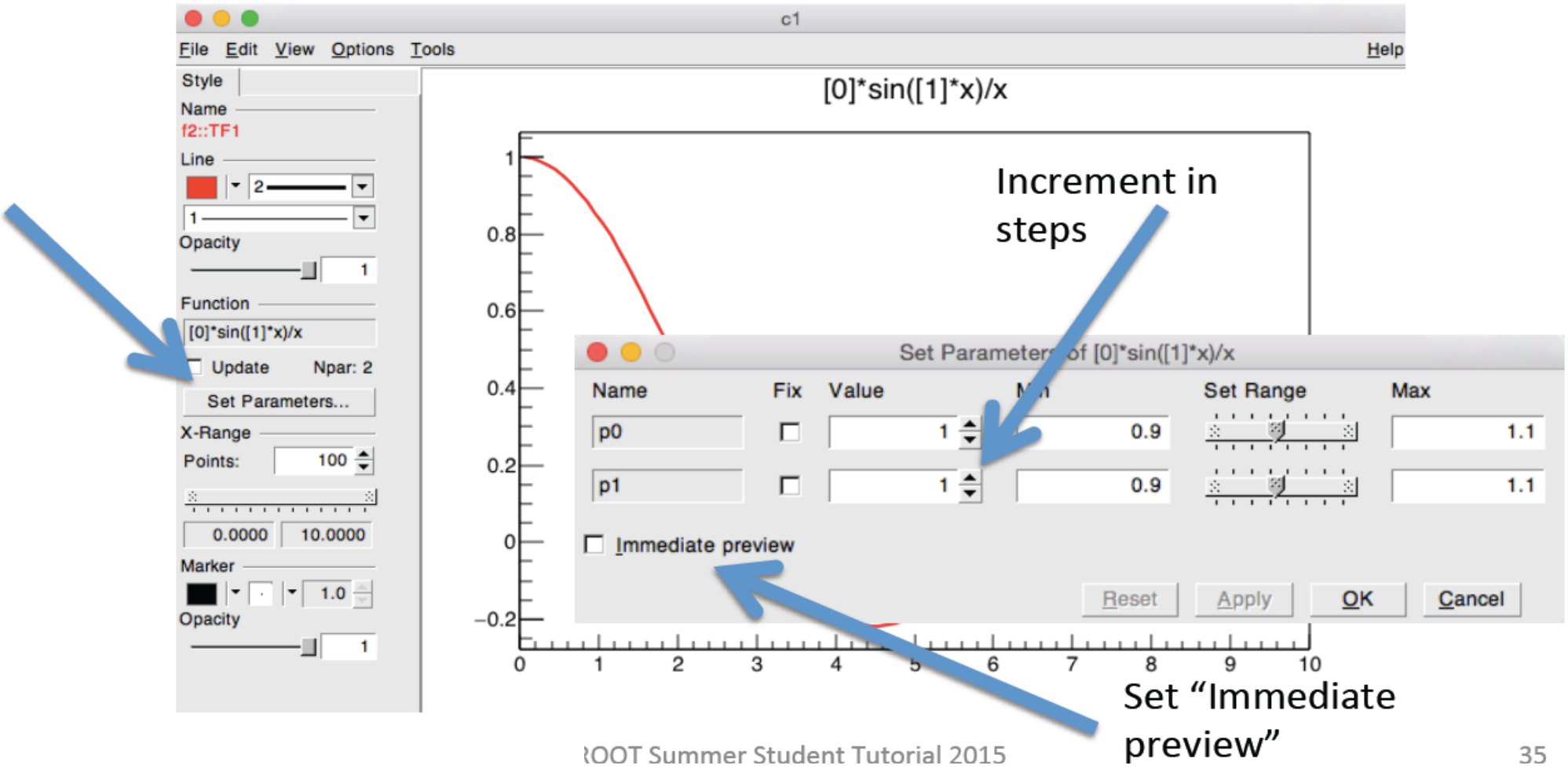
- proviamo a cambiare gli attributi grafici (colore, tipo di linea, spessore...)



Analisi dati con ROOT

Esercizio 1 : ROOT per disegnare una funzione

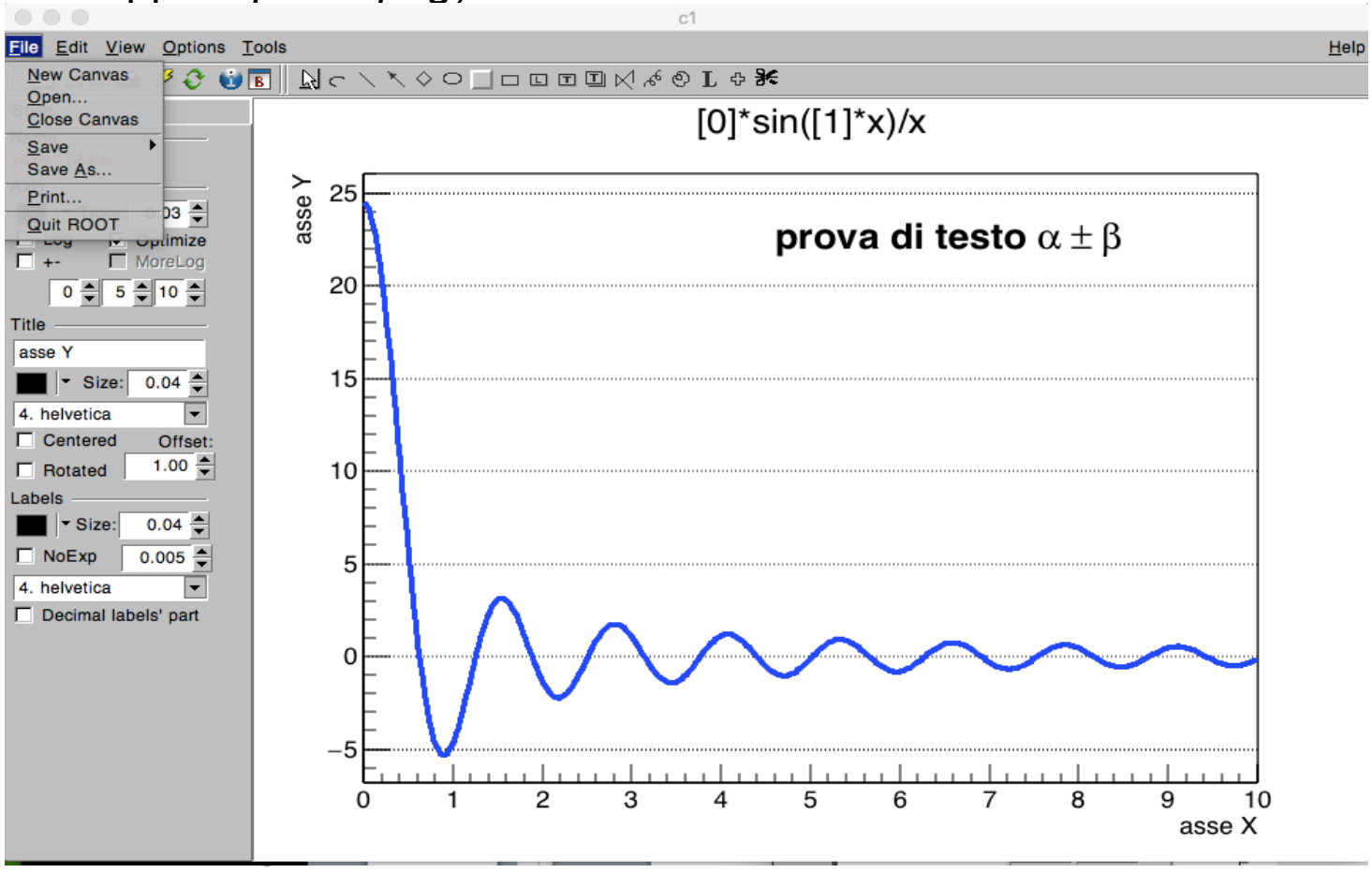
- proviamo a modificare i parametri



Analisi dati con ROOT

Esercizio 1 : ROOT per disegnare una funzione

- aggiungiamo il ***Toolbar***, inseriamo testo e altre informazioni, quindi salviamo su file: File → SaveAs → scegliere un formato grafico (.jpg, .gif, .png, .pdf, ...) e/o modificabile (.root) scrivendo esplicitamente l'estensione nel nome del file (es. *prova.root* oppure *prova.png*)



Analisi dati con ROOT

Esercizio 2 : analizziamo dei dati con un grafico

- carichiamo i dati di una misura di assorbimento gamma salvati su un file di testo in un oggetto grafico di tipo TGraphErrors

```
root [0] TGraphErrors *g = new TGraphErrors("assorb.dat");
root [1] g->SetName("assorb"); // optional but useful
root [2] g->Draw("ap")
```

file assorb.dat :

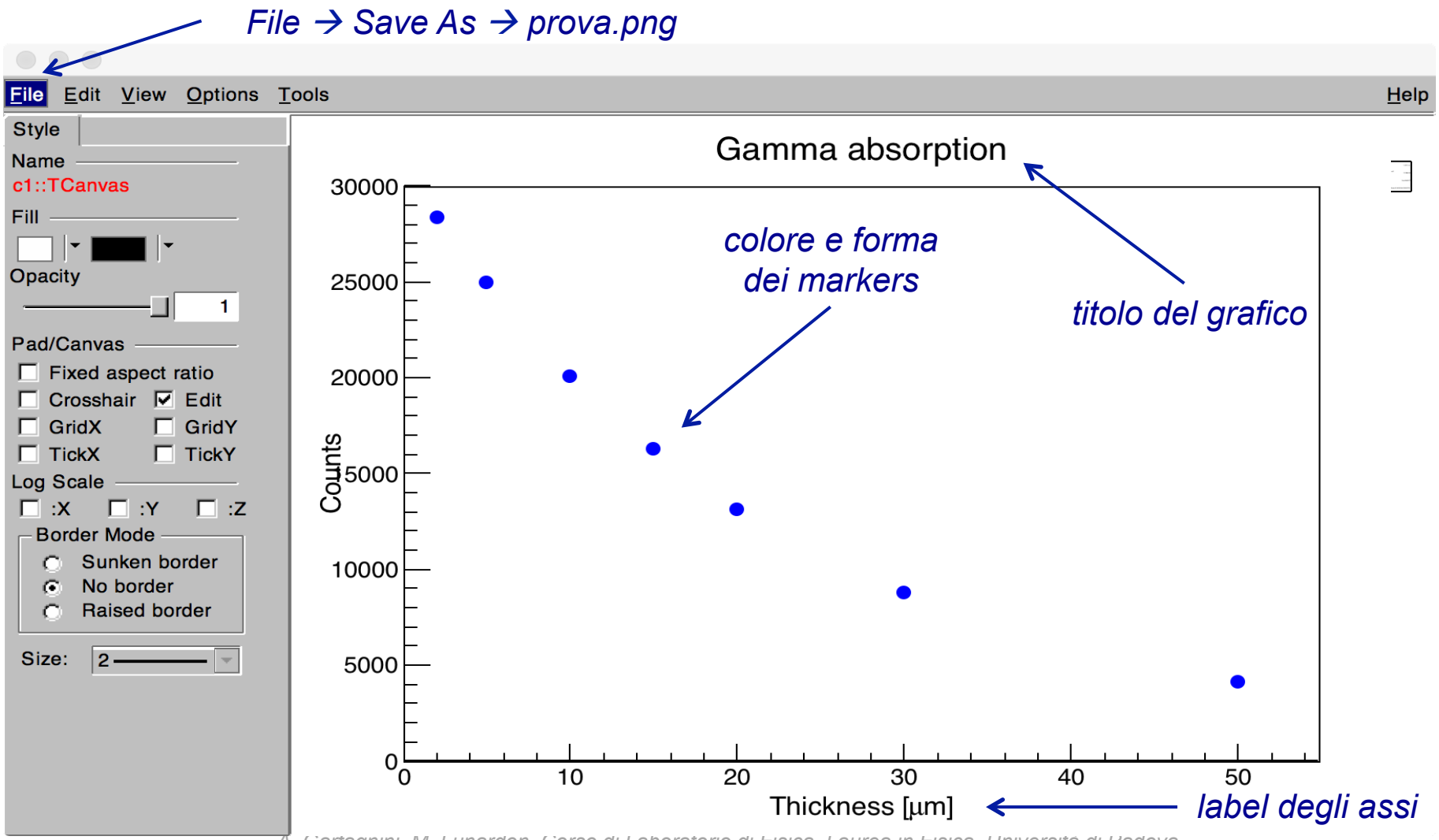
```
# graph data:
# thick    counts    errTk    errCnt
2.0        28420     0.02     50
5.0        24960     0.02     40
10.0       20130     0.02     40
15.0       16270     0.02     30
20.0       13130     0.02     50
30.0       8780      0.02     20
50.0       4160      0.02     20
```

settando il nome, l'oggetto sarà visibile all'interno della shell con questo nome. L'etichetta "assorb" diventa in qualche modo un puntatore all'oggetto appena creato

Analisi dati con ROOT

Esercizio 2 : analizziamo dei dati con un grafico

- sistemiamo con l'Editor la figura e poi salviamola in formato grafico



Analisi dati con ROOT

Esercizio 2 : analizziamo dei dati con un grafico

- Proviamo a fittare la distribuzione con una funzione esponenziale + fondo costante

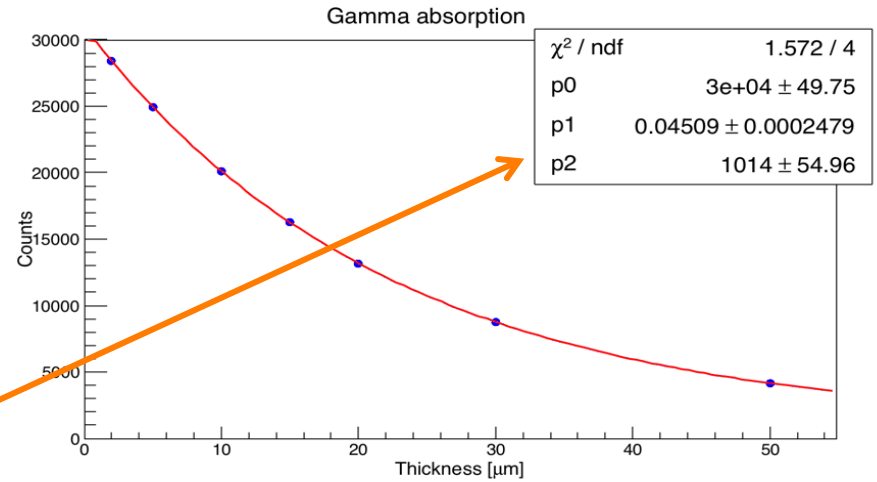
funzione da fittare

```
root [1] TF1 *f1 = new TF1("f1","[0]*exp(-[1]*x)+[2]",0.,60.);
root [2] f1->SetParameters(30000,0.1,0);
root [3] assorb->Fit(f1,"R");
 FCN=1.57211 FROM MIGRAD    STATUS=CONVERGED     109 CALLS         110 TOTAL
                     EDM=1.03677e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           2.99950e+04   4.97482e+01   2.20003e-02   9.99411e-06
   2  p1           4.50853e-02   2.47902e-04   4.68745e-08  -4.76586e+00
   3  p2           1.01443e+03   5.49556e+01   9.15876e-03   3.09368e-05
```

inizializzazione con valori ragionevoli

Fit del grafico "assorb" con la funzione f1 nella regione in cui è definita f1

Output testo di MIGRAD e output grafico



Analisi dati con ROOT

Le macro

- I comandi (istruzioni C++) dell'analisi possono essere raggruppati in veri e propri "programmi" che possono essere eseguiti dall'interprete (procedura più semplice) o compilati (più preciso e veloce in esecuzione).

- Questi programmi (**macro**) hanno la tipica estensione ".C" e devono contenere una funzione principale (il "main") che ha lo stesso nome del file. Possono inoltre ricevere degli argomenti dall'esterno.

- Esempio: *il file mymacro.C :*

```
double func1(double x) {
   return TMath::Sqrt(x)/(1+x);
}

void mymacro(double x) {
   std::cout << "func = << func1(x) << std::endl;
}
```

Analisi dati con ROOT

Le macro

- dall'interno del CINT/CLING si può :

- eseguire la macro direttamente con ".x":

```
root [0] .x mymacro.C(10)
func = 0.28748
```

- caricare la macro in memoria e usarla come funzione:

```
root [0] .L mymacro.C
root [1] mymacro(10)
func = 0.28748
```

- caricare la macro in memoria e usarla come funzione:

```
root [0] .L mymacro.C++
    Info in <TUnixSystem::ACLiC>: creating shared library /
    server0/0/2015/lunardon/CorsoRoot2016/./mymacro_C.so
root [1] mymacro(10)
func = 0.28748
```

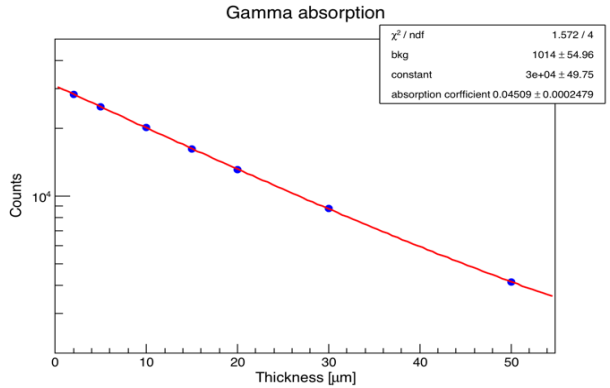
includere Riostream.h e TMath.h

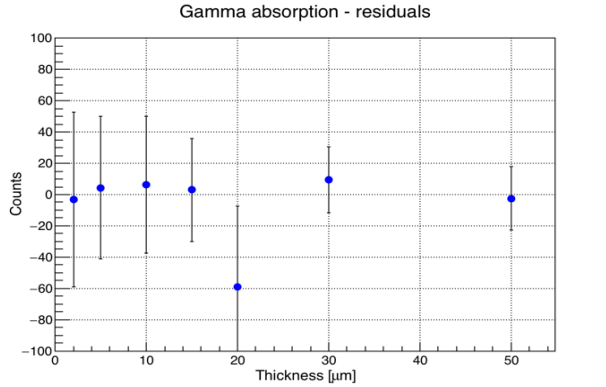
Analisi dati con ROOT

Esercizio 3 : analisi completa dei dati precedenti con una macro

- **macro FitAssorb.C**
  - carica il grafico “assorb.dat”
  - imposta le caratteristiche grafiche
  - fitta con una funzione esponenziale decrescente + fondo cost.
  - recupera tutti i risultati del fit (parametri, chi2, matrice di covarianza...)
  - produce il grafico dei residui

```
Gamma absorption coefficient mu 0.0450853 +- 0.000247903

COVARIANCE MATRIX :

3x3 matrix is as follows

     |      0    |      1    |      2    |
-----------------------------------------
   0 |        3020       -1429      0.01269
   1 |       -1429        2475   -0.003129
   2 |    0.01269   -0.003129   6.146e-08

chi2 : 1.57211
sigma_mu = 0.000247903
```





Analisi dati con ROOT

Altri esercizi...

- **macro macro1.C** : grafico e fit lineare descritto nella sezione 3.2 del Primer

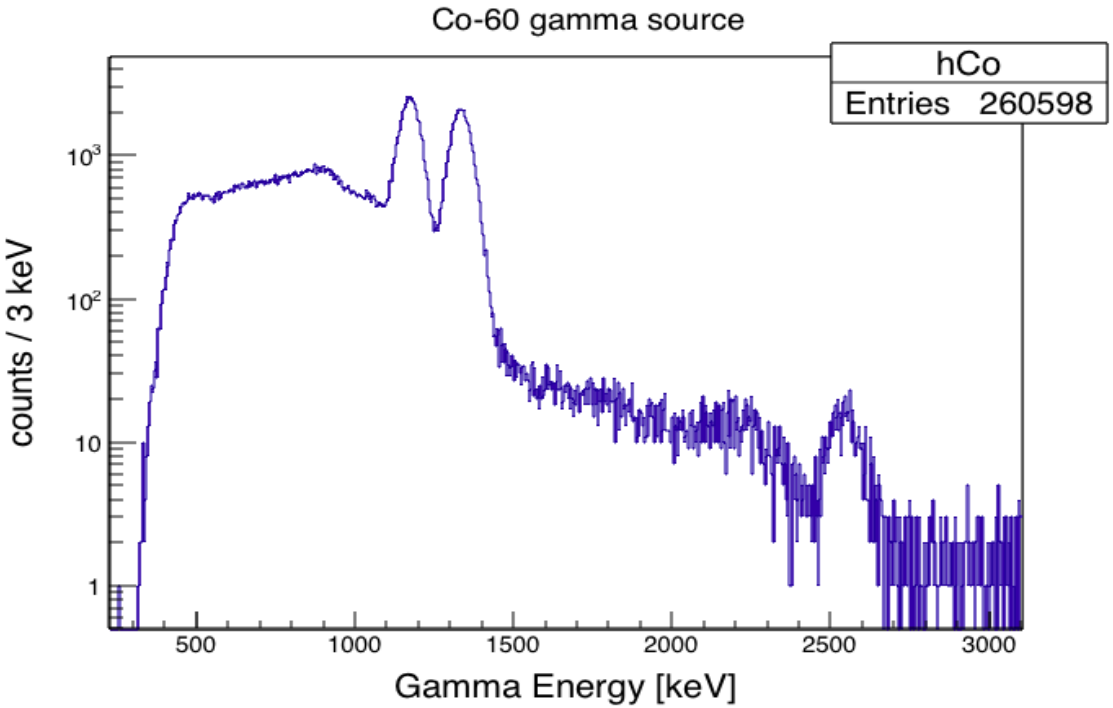
- **file 1N4148.txt** : caratteristica I-V di un diodo 1N4148. Creare una macro che carica il grafico, lo disegna con asse Y logaritmica, esegue il fit esponenziale (verificare la regione di fit...) e stampa sullo schermo i risultati. Potete anche usare i vostri dati raccolti in laboratorio (meglio!)

- **file CD4007UB.txt** : caratteristica I_D-V_{DS} di un nMOS. Plottare e fittare linearmente la regione di saturazione per estrarre λ_n e r_o . Attenzione alla regione del fit...

Analisi dati con ROOT

Gli istogrammi

- Uno degli oggetti più utilizzati nell'analisi dati in fisica
- In generale l'istogramma 1D rappresenta il **grafico delle frequenze**: in x c'è una certa grandezza divisa in intervalli (*bins*) e in y il numero di misure entro quell'intervallo.
- Il profilo dell'istogramma normalizzato tende quindi alla funzione densità di probabilità della variabile x.
- La spaziatura di default degli intervalli (*binning*) è tutta uguale, ma è possibile costruire istogrammi con binning variabile.



Analisi dati con ROOT

Gli istogrammi

- La classe base di ROOT è TH1, con le implementazioni a numeri di diversa precisione. L'istogramma di float è il TH1F
- Il costruttore principale è :

  TH1F *h = new TH1F("hname", "htitle", nbinsx, xlow, xup)

  hname = char*, nome dell'oggetto

  htitle = char*, titolo dell'istogramma

  nbinsx = int, numero di bin dell'asse x

  xlow, xup = double, limiti dell'asse x

- Esempio:

  *TH1F *h = new TH1F("h","histo prova",400,0,2000); // crea un istogramma di 2048 canali con asse da 0 a 2000 (=> 5 a.u. per bin)*

  h->Fill(502); // incrementa di 1 il 101-esimo bin corrispondente a 500<=x<505.

  h->Draw(); // disegna l'istogramma

Analisi dati con ROOT

Gli istogrammi

- Alcuni metodi utili per l'analisi:
  - **Fill**(valore, peso) : incrementa il bin di "peso" anziché di 1
  - **SetBinContent**(bin, content) : inserimento manuale del contenuto
  - **GetBinContent**(bin) : recupera il contenuto del bin
  - **Integral**(binx1, binx2) : somma il contenuto da binx2 a binx2
  - **Fit**(func/formula, "opt") : fitta con una func o una formula
  - **Clone**("newname") : crea una copia con un nuovo nome
  - **Add**(...) / **Divide**(...) / **Scale**(...) : operazioni tra istogrammi
  - **Rebin**(n) : compatta l'istogramma
  - **Clear**() / **Reset**() : resetta l'istogramma
  - **GetXaxis**()/**GetYaxis**() : ritorna il puntatore agli assi
  - ...

Analisi dati con ROOT

La funzione Gaussiana

- La **gaussiana** ha un particolare interesse nell'analisi dati fisica (non a caso è la prima opzione proposta dal FitPanel...).
- Il significato fisico dei tre parametri dipende dal contesto, ma generalmente è del tipo:
  - **mean = valore dell'osservabile** (energia, massa, lunghezza d'onda, ...)
  - **area = intensità dell'osservabile** (intensità luminosa, sezione d'urto di produzione...)
  - **sigma = risoluzione** dell'apparato o intrinseca del metodo di misura o della fisica dell'esperimento

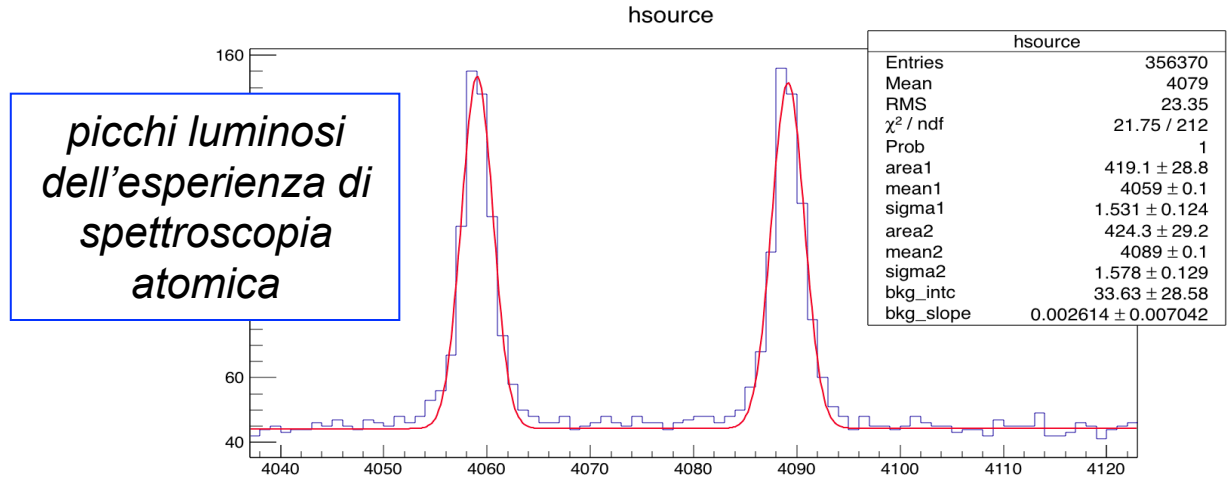
$$f(x) = \frac{A}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

A = area
μ = mean value
σ = sigma

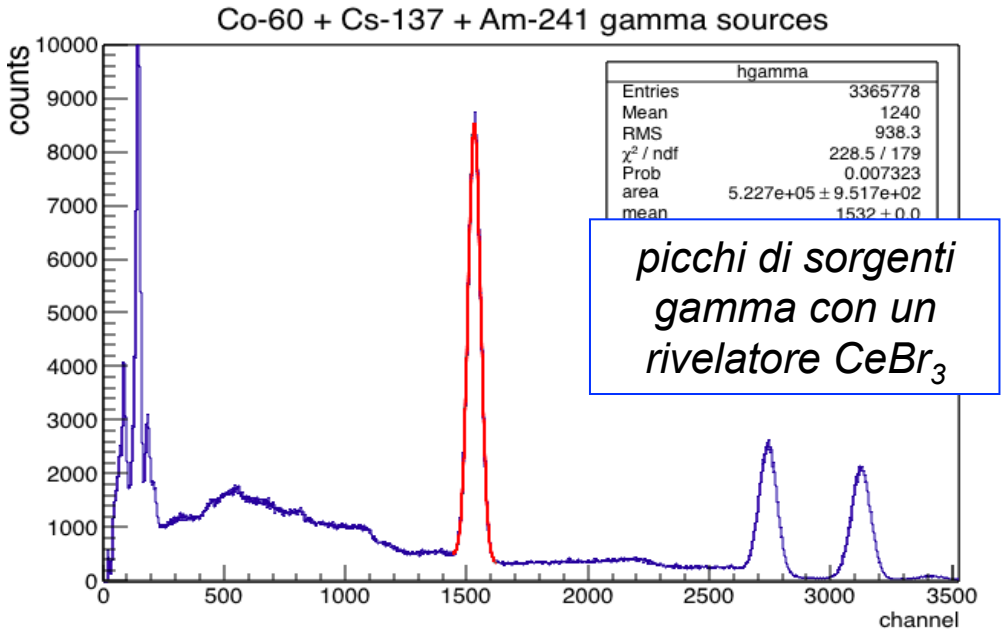
Analisi dati con ROOT

La funzione Gaussiana

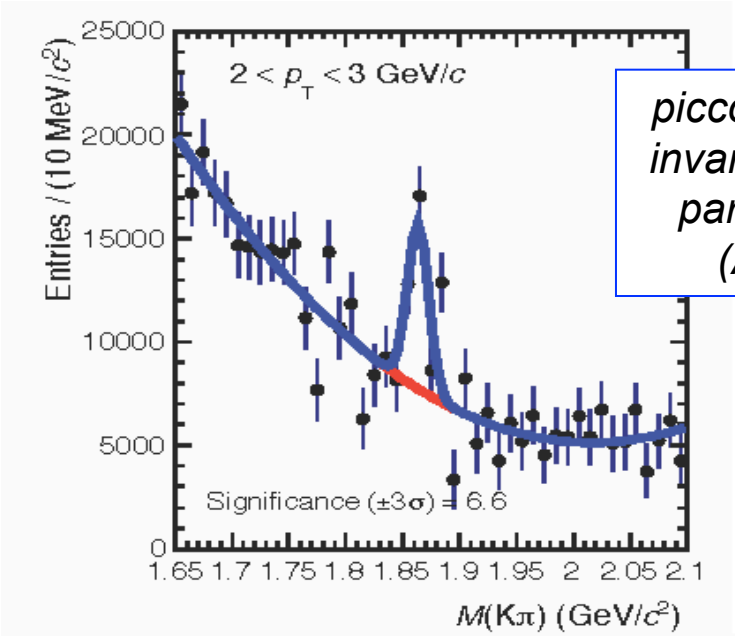
- Qualche esempio di istogrammi con gaussiane...



picchi luminosi dell'esperienza di spettroscopia atomica



picchi di sorgenti gamma con un rivelatore $CeBr_3$

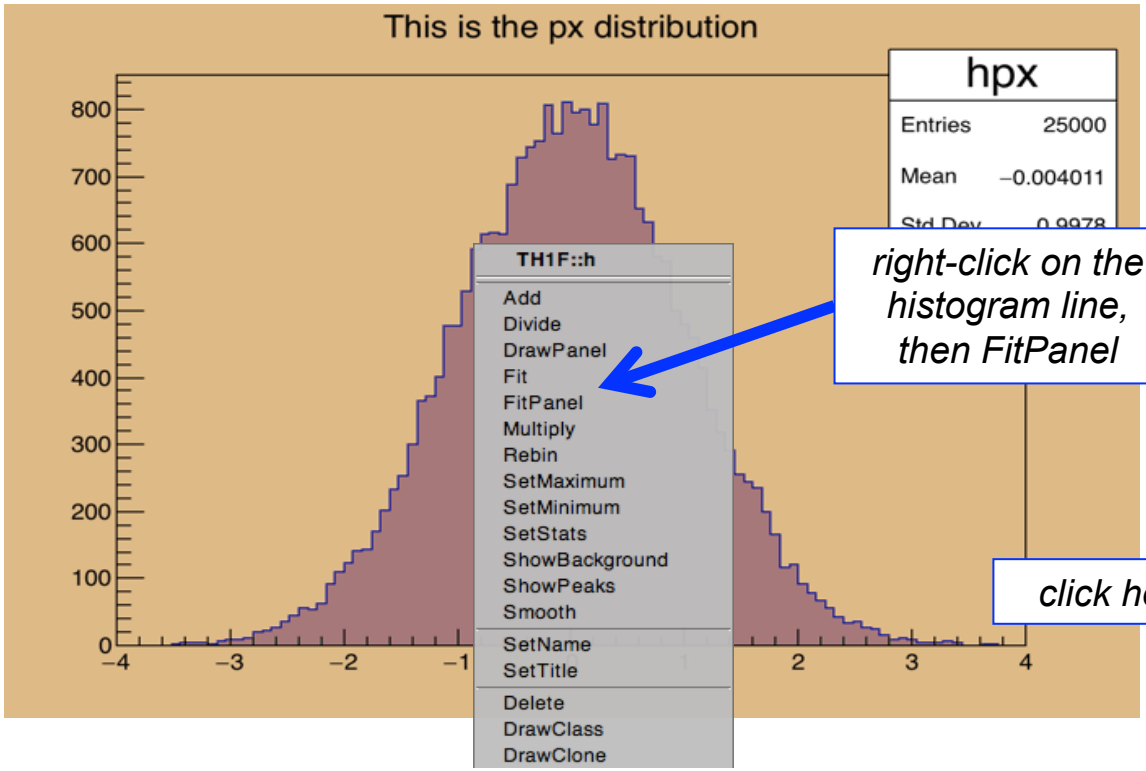


picco di massa invariante della particella D^0 (ALICE)

Analisi dati con ROOT

Esercizio 4 : creiamo e analizziamo un istogramma

- Eseguiamo la macro **$ROOTSYS/tutorials/hsimple.C** e fittiamo l'istogramma con una gaussiana usando il **FitPanel**



Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd

- Proviamo ora con una gaussiana di dati “veri”. Carichiamo l’istogramma HgCd.csv (*file di testo – il numero nella riga i-esima è il contenuto dell’i-esimo bin*) con la macro **ReadHistoFromTextFile.C** che legge un file di testo e crea un oggetto istogramma:

  TH1F ****ReadHistoFromTextFile*(const char *fname, const char *histname=NULL, bool draw=1)*

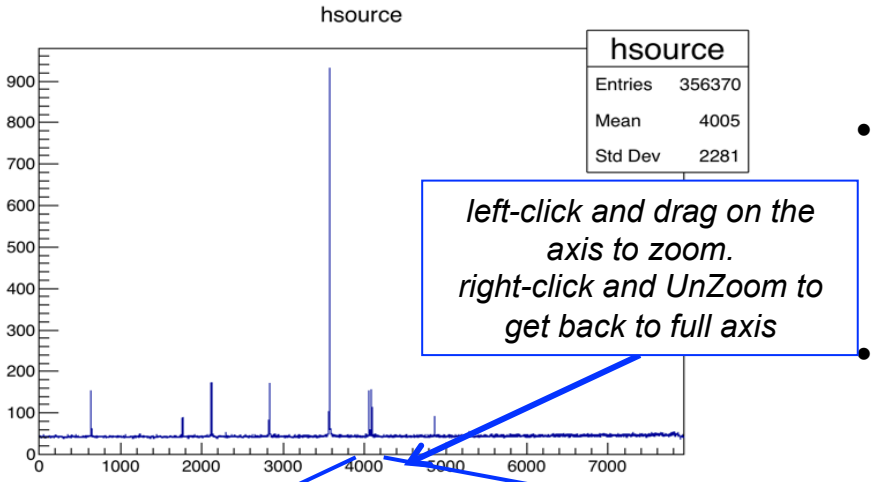
  - *legge il file di testo fname (riga i-esima = contenuto del bin i)*
  - *assegna eventualmente il nome histname all’istogramma*
  - *disegna (draw=1) o no l’istogramma caricato*
  - *ritorna il puntatore all’oggetto creato di tipo TH1F*

```
[root [0] .L ReadHistoFromTextFile.C
 root [1] ReadHistoFromTextFile(

TH1F* ReadHistoFromTextFile(const char* fname, const char* histname = __null, bo
ol draw = 1)
[root [1] ReadHistoFromTextFile("HgCd.csv","hsource")                          ]
creating new histo with 7926 bins...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(TH1F *) 0x7f80cb2f9280
root [2]
```

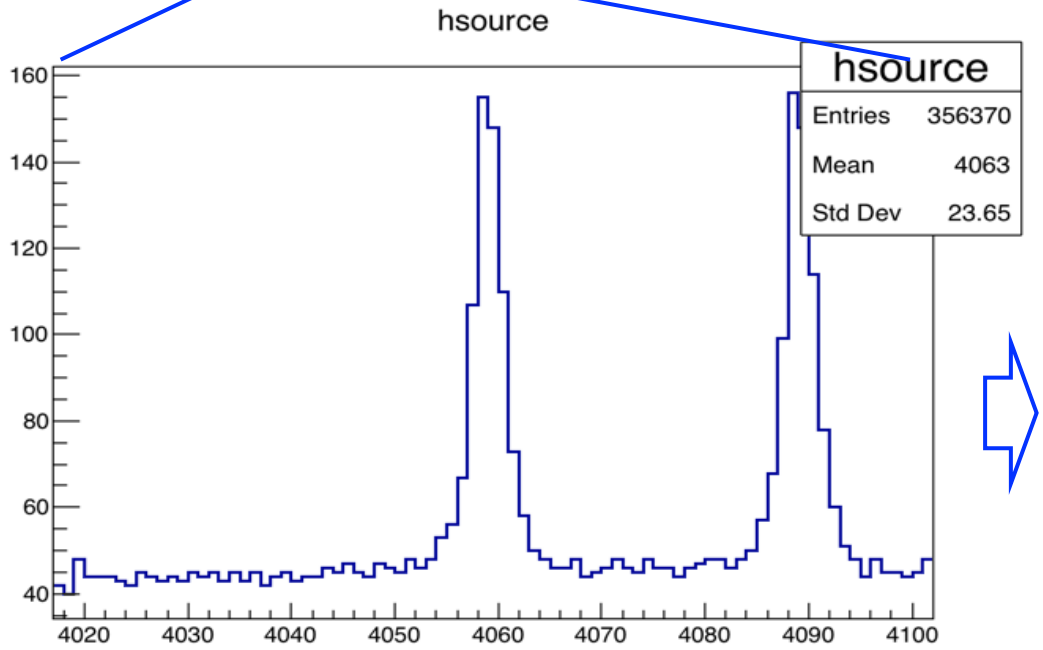
hint: il TAB completa il comando e fa anche vedere la struttura della funzione/classe

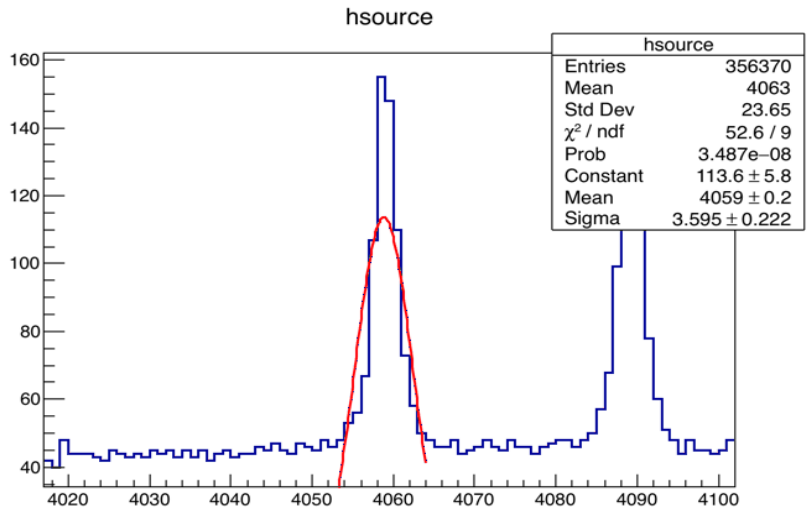
Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd



- Espandiamo per vedere il picco alla posizione **4059** (576.96 nm)...
- ... e proviamo a fare un fit gaussiano col FitPanel...





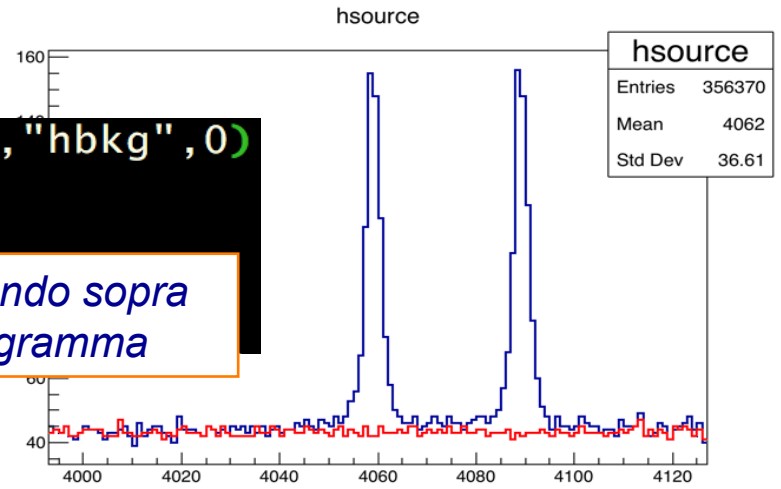
Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd

- Il fit viene male per la presenza del fondo. Si può definire una funzione con un parametro in più (gaussiana + fondo costante) oppure utilizzare l'istogramma del fondo, sottraendolo dai dati della sorgente:

```
root [2] ReadHistoFromTextFile("HgCd_bkg.csv","hbkg",0)
creating new histo with 7926 bins...
(TH1F *) 0x7fd5a6563de0
root [3] hbkg->SetLineColor(2);
root [4] hbkg->Draw("same");
```

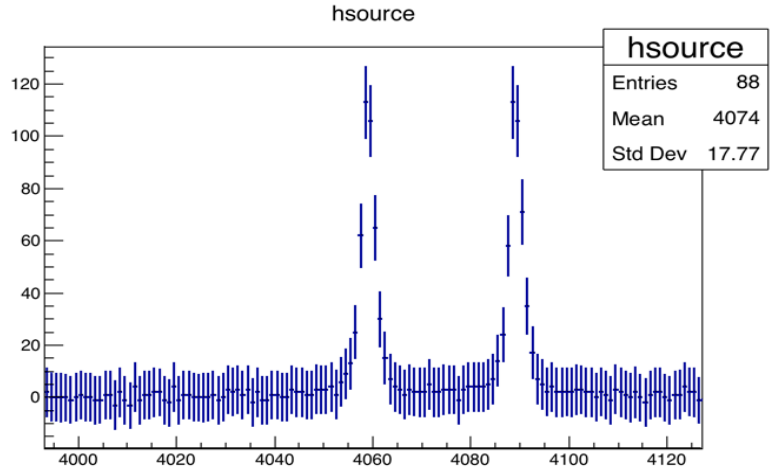
disegna il fondo sopra l'altro istogramma



abilita il calcolo automatico degli errori statistici

```
root [6] hsource->Sumw2(1);
root [7] hsource->Add(hbkg,-1);
root [8] hsource->Draw()
```

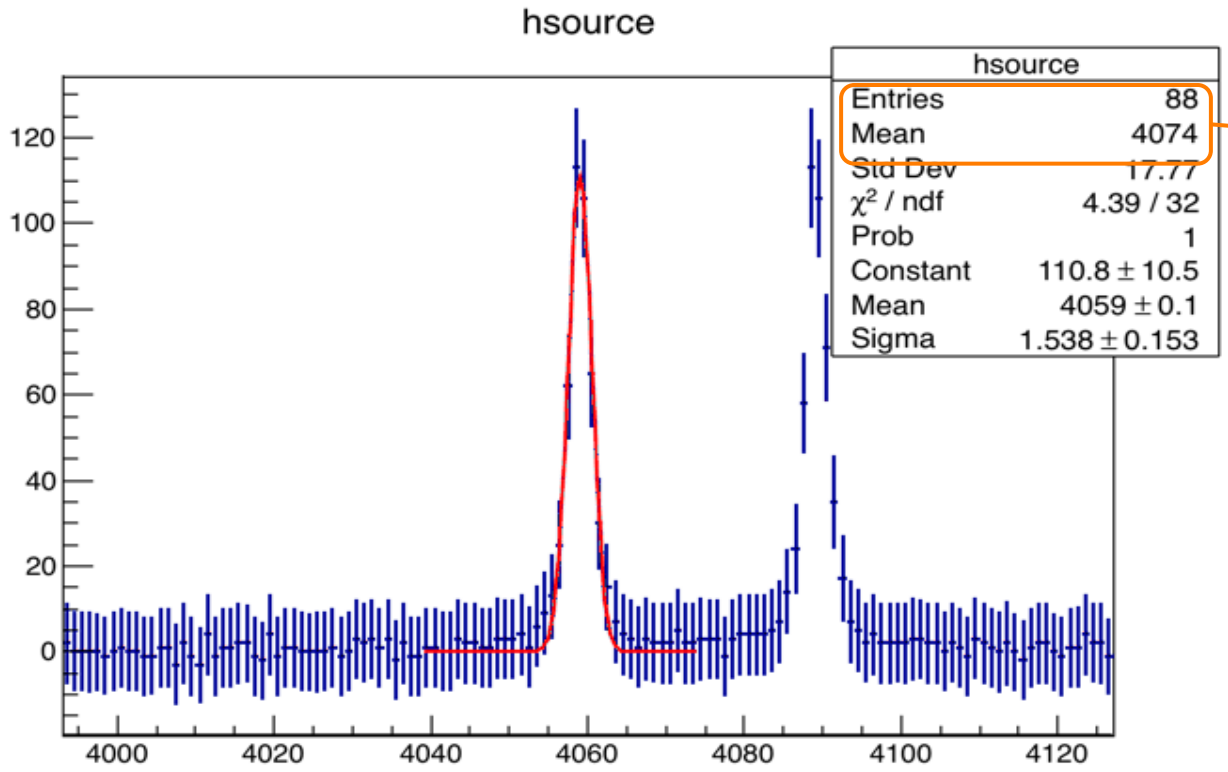
sottrae il fondo bin a bin



Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd

- E' ora possibile fittare con la gaussiana semplice. Il fit viene bene (Prob=1) a causa degli errori statistici molto grandi (pochi conteggi per bin). E' possibile comunque vedere già qui che le code presentano qualche problema...



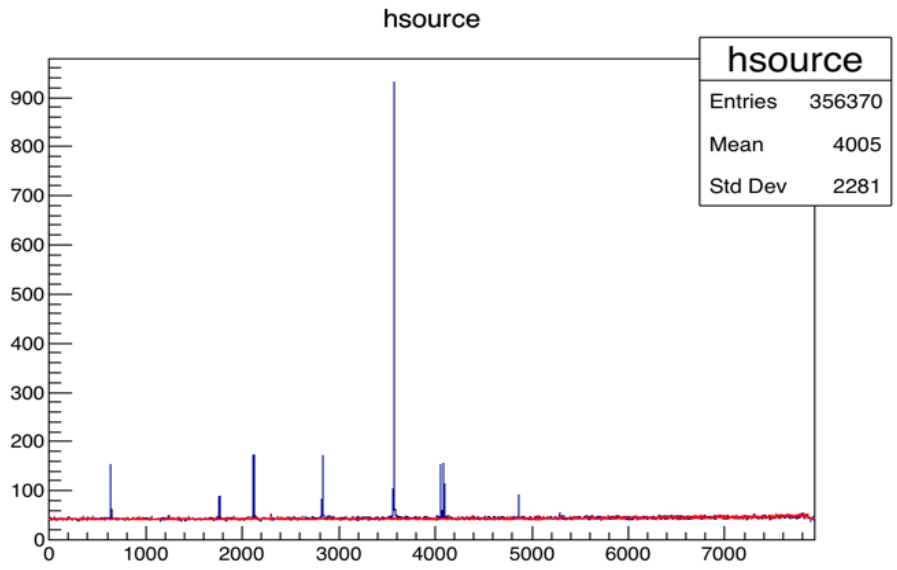
ATTENZIONE: queste info sono relative a tutto il pezzo di istogramma visualizzato, non solo al picco analizzato!

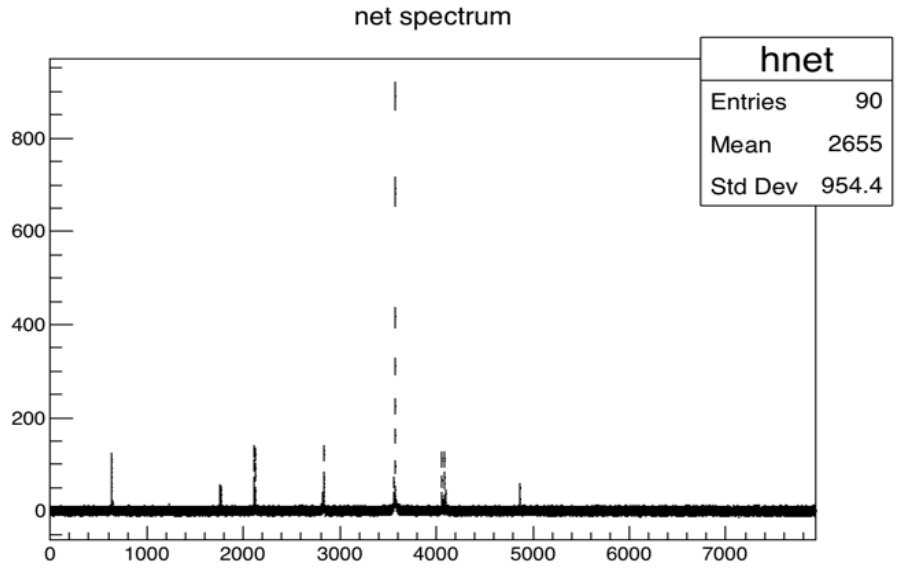
Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd

La macro **Prisma1.C** carica gli spettri della sorgente e il fondo da file di testo e crea l'istogramma con fondo sottratto.

*void **Prisma1**(const char *hname="HgCd.csv", const char *hnamebkg="HgCd_bkg.csv")*

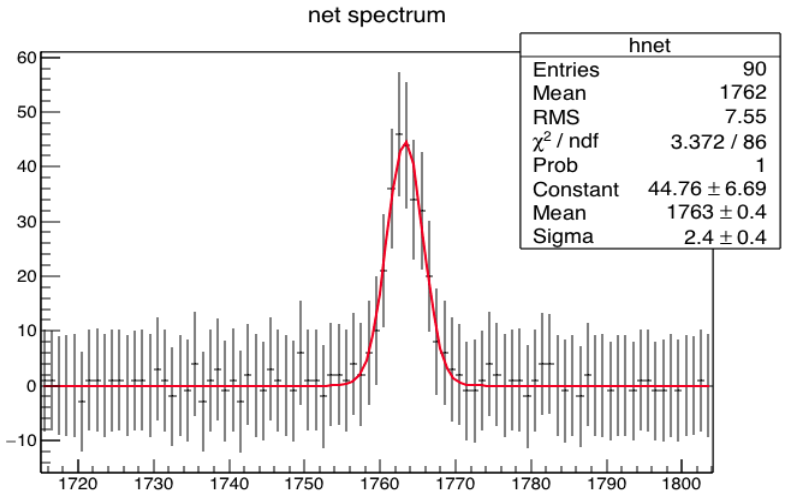




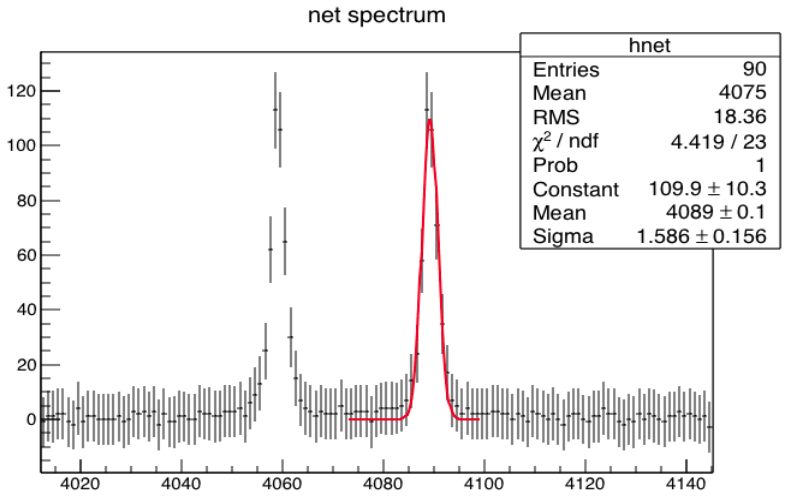
Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd

- Possiamo fittare altri picchi, per esempio alle posizioni **1763** (riga a 467.82 nm) e **4089** (579.07 nm)...



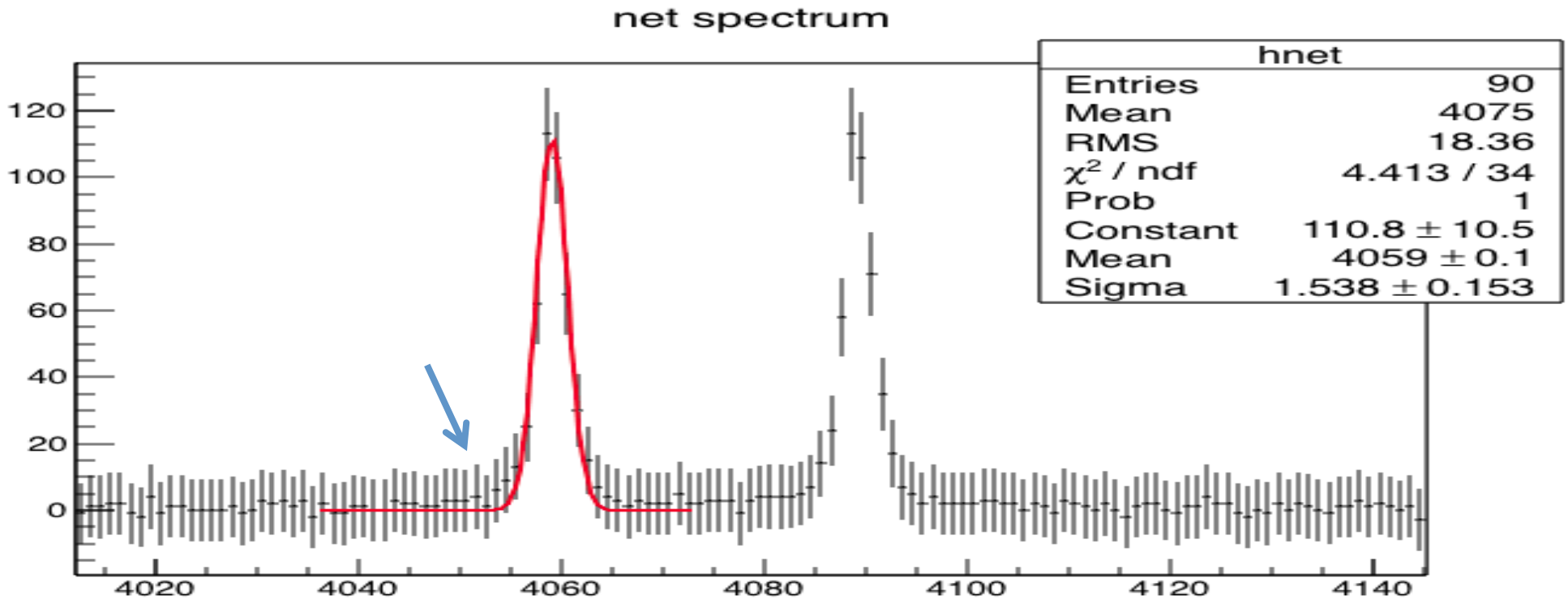
- **Attenzione *all'inizializzazione dei parametri*** nei fit successivi al primo: cliccare *Set Parameters*, inserire il valore approssimato e premere *enter*...



Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd

- **Ricaviamo l'area** del picco a 577 nm:
  1) dal **fit** : A = Constant * sigma * sqrt(2*π) = **427.2 ± (?) counts**
  2) col "**bin counting**" : A = integral(mean +- 3*σ) = integral(4054,4064) = **451 ± (?) counts** => *il metodo 2 da' un valore più alto → code....*



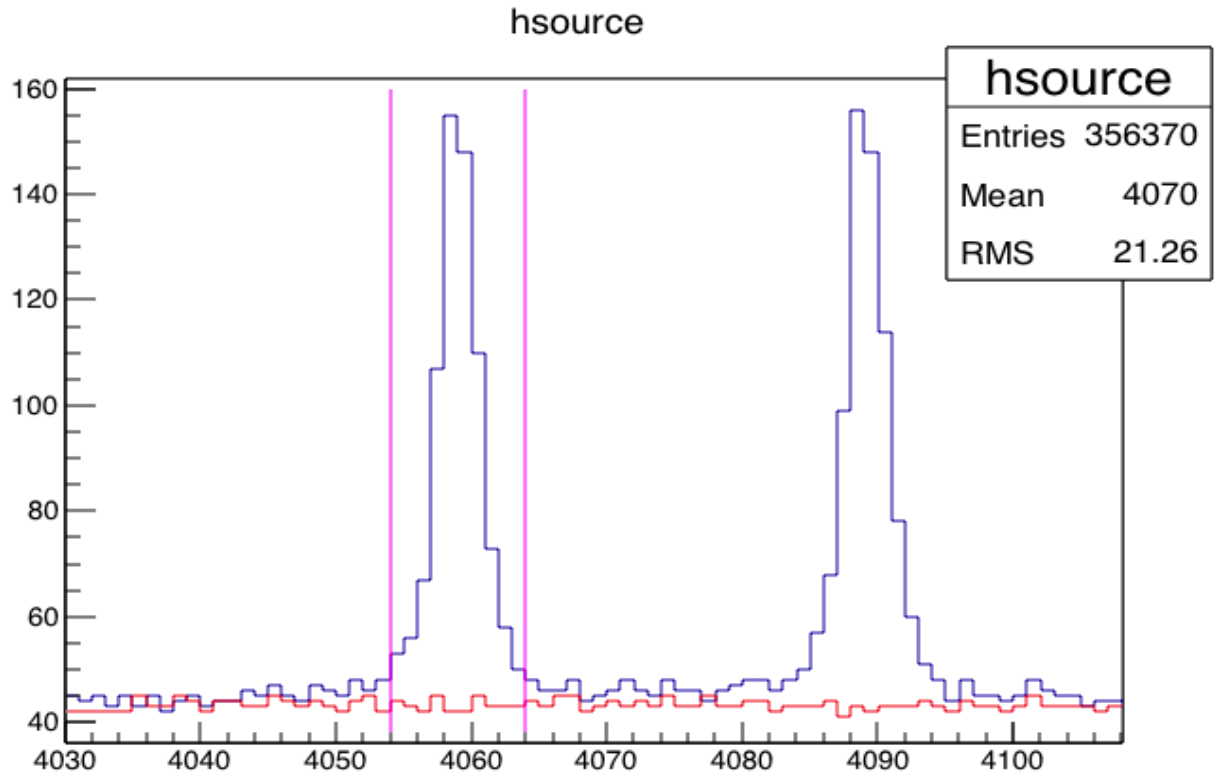
Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd

- Possiamo calcolare correttamente l'incertezza dell'area del picco a 577 nm usando gli integrali degli spettri sorgente_con_fondo e del fondo:
  - hsource->Integral(4054,4064) = 925 = N
  - hbkg->Integral(4054,4064) = 474 = B

- A = N – B = 451
- σ_A = sqrt(N+B) = 37

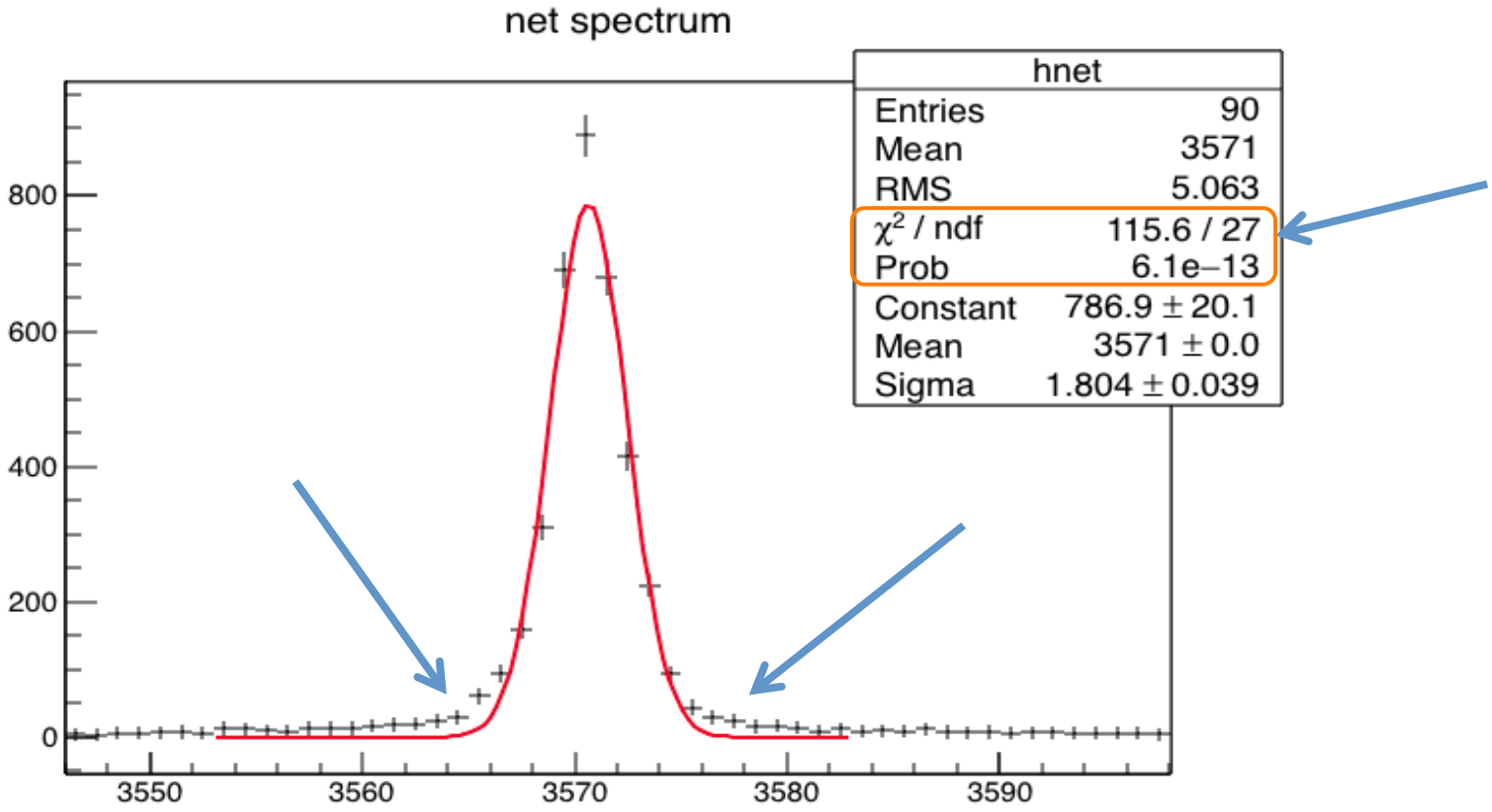
=> A = 451 ± 37 counts



Analisi dati con ROOT

Esercizio 5 : Fit gaussiano picchi spettro CCD lampada HgCd

- L'aspetto delle code è ancora più evidente sulla riga di massima intensità...

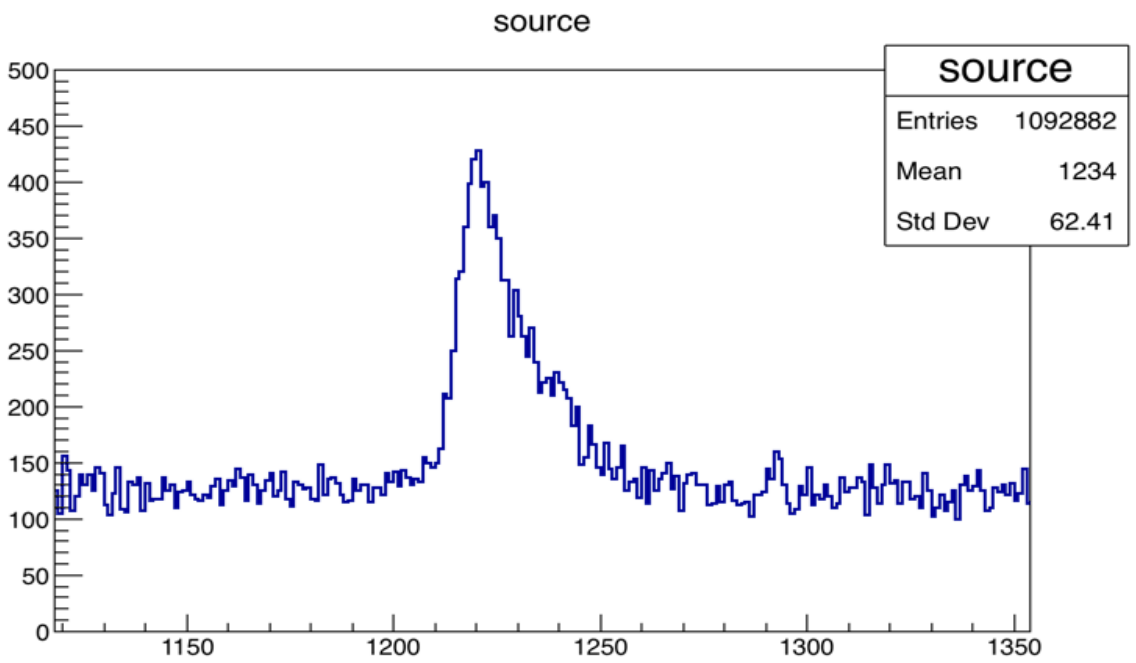


→ provare a calcolare l'area (con errore) dal fit e col bin counting per questo picco.

Analisi dati con ROOT

Nota: centroide della distribuzione e valor medio della gaussiana

- Il picco in figura è chiaramente **non gaussiano** (in questo caso la deformazione della riga è dovuta alle impostazioni non ottimali dello spettrometro)

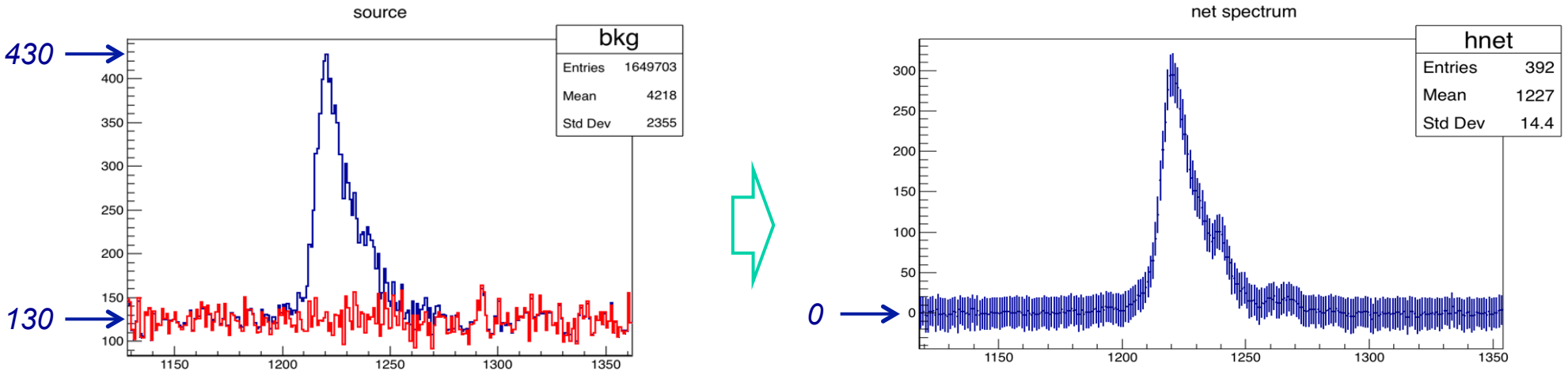


- Come facciamo a estrarre in modo corretto le informazioni di interesse (posizione, area, larghezza) ?

Analisi dati con ROOT

Nota: centroide della distribuzione e valor medio della gaussiana

1. Notiamo innanzi tutto che il segnale che vogliamo analizzare (la riga) è sovrapposto a un fondo non trascurabile, che dovrà quindi essere tenuto in conto nella valutazione dei parametri. In questo caso il fondo è stato misurato nelle stesse condizioni e possiamo quindi sottrarlo direttamente prima di procedere con l'analisi

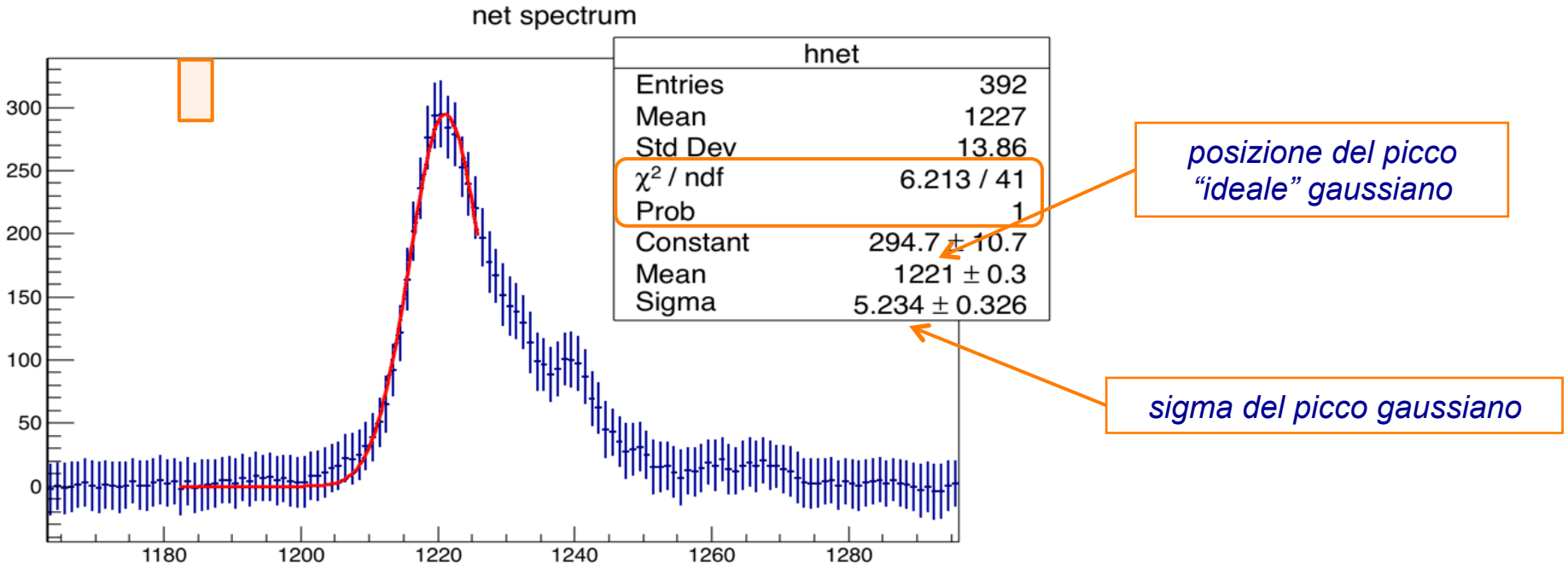


Analisi dati con ROOT

Nota: centroide della distribuzione e valor medio della gaussiana

2. La corretta stima della posizione (*mean*) **dipende dalla situazione fisica**.

   a) può essere *la posizione del massimo* → si può fittare solo la parte più regolare della curva, o magari solo la parte alta del picco...

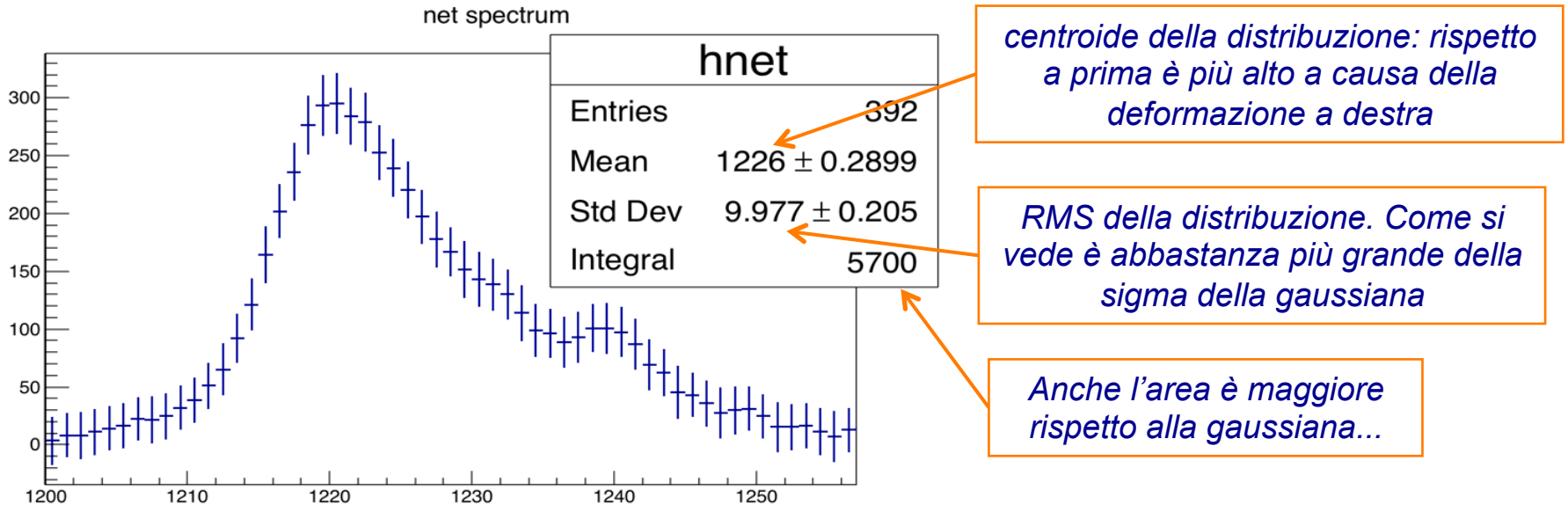


Analisi dati con ROOT

Nota: centroide della distribuzione e valor medio della gaussiana

2. La corretta stima della posizione (*mean*) **dipende dalla situazione fisica**.

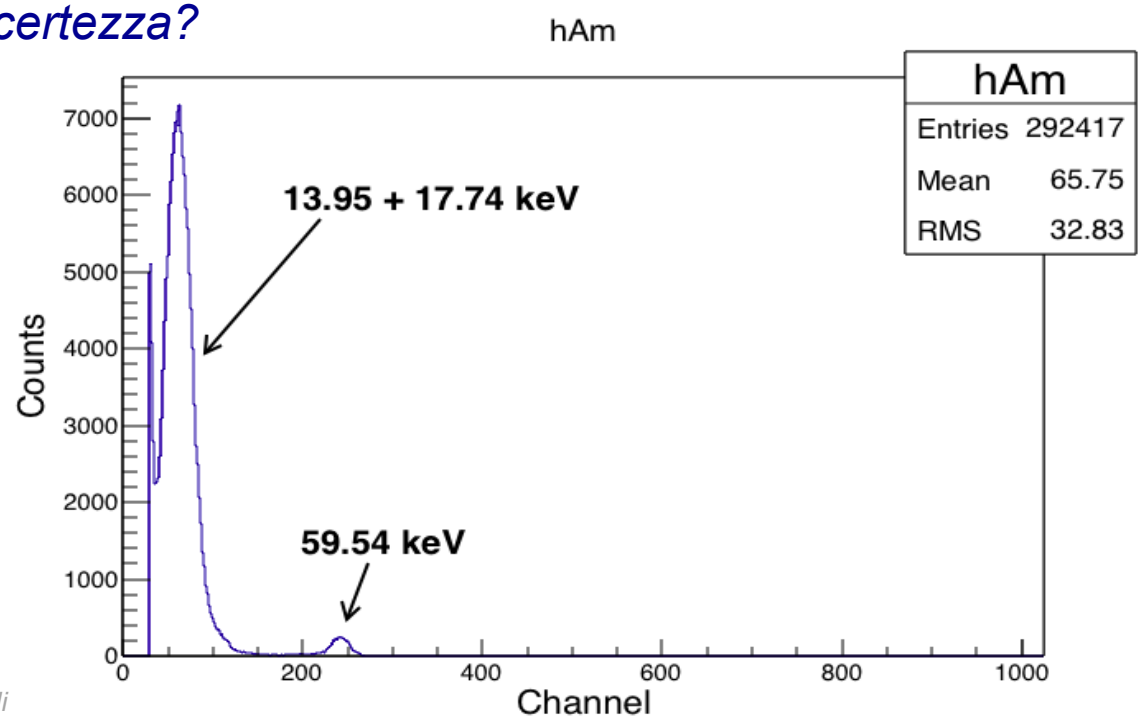
   b) può essere *il centroide*, cioè il valore medio della distribuzione → ROOT fornisce direttamente l'informazione del centroide del pezzo di istogramma visualizzato nella Canvas. Occorre quindi visualizzare il picco in un intervallo ragionevole (*nb: i risultati, in particolare l'RMS, dipenderanno da questa scelta...*)



Analisi dati con ROOT

Esercizio 6 : Spettro della sorgente X con un Fotodiodo al Silicio

- Carichiamo il file Am241_FD.dat (spettro in formato testo) usando la macro precedente ReadHistoFromTextFile.C
- Espandiamo e fittiamo il picco a 59.5 keV
  - *con che funzione?*
  - *in che range?*
  - *quanto vale l'energia e la sua incertezza?*
  - *quanto vale l'area e la sua incertezza?*
  - *quanto la larghezza?*



Analisi dati con ROOT

Input / Output su files

- In un'analisi complessa ci sono spesso diversi step ed è utile poter salvare i risultati su file per poterli rivedere o modificare in seguito.

- Scrittura su files:

```
TFile *output_file = new TFile(fileName,"RECREATE");
output_file->cd();
....
generic_object->Write();
output_file->Close();
```

- Lettura da files

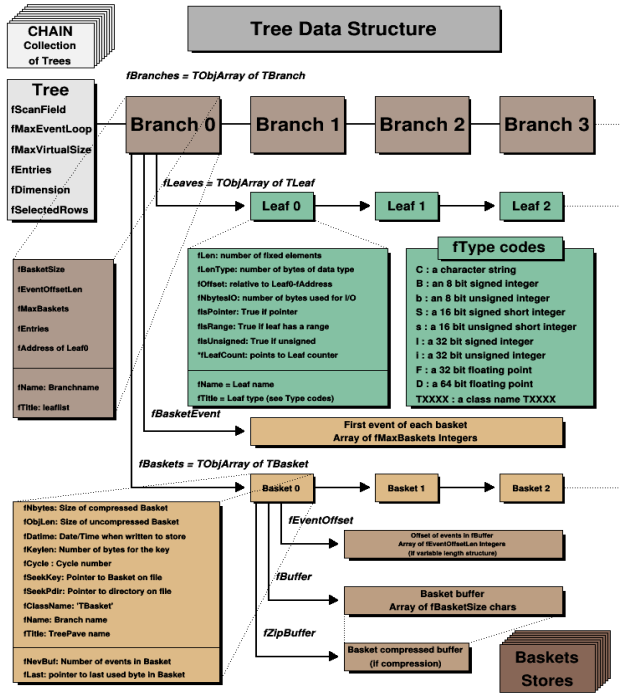
```
TFile *infile = new TFile(fileName,"READ");
TH1F *h = (TH1F*)infile->Get("h");
h->Draw();
...
input_file->Close(); // non necessario in generale...
```

Analisi dati con ROOT

Strutture dati complesse : le classi TTree e TNtuple

- La classe **TTree** permette di gestire **grandi quantità di dati del medesimo tipo** (tipo quelli provenienti dai grandi esperimenti di fisica subnucleare) in modo versatile ed efficiente.

- I dati possono essere strutturati su più livelli (***branches***, ***leaves***). Ogni *branch* ha la sua definizione e la sua lista di buffers che contengono i dati veri e propri (*leaves*). Il Tree è una lista di branches indipendenti.

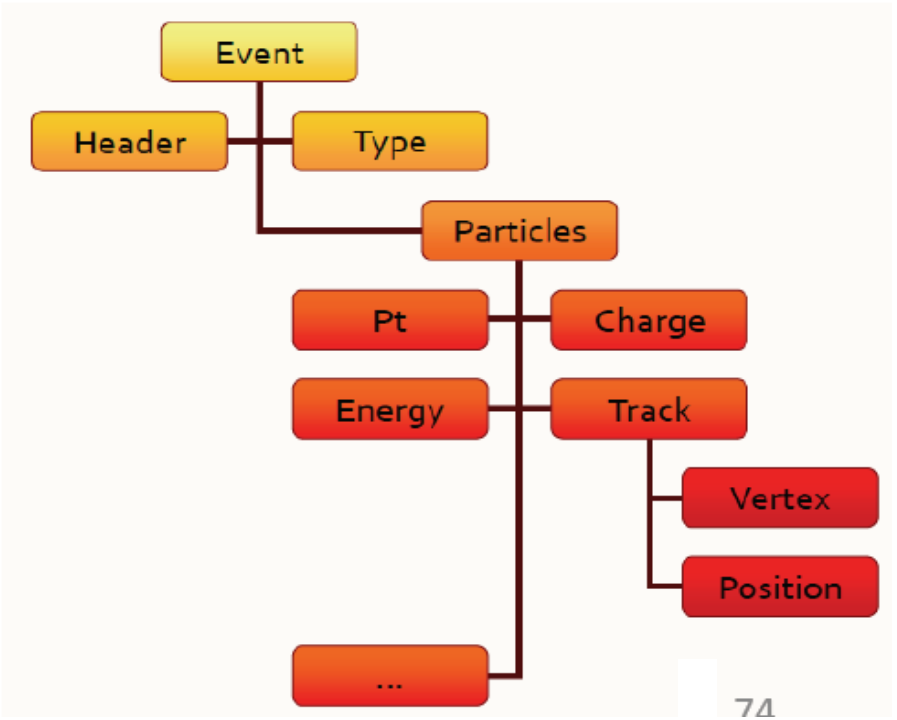


Analisi dati con ROOT

Strutture dati complesse : le classi TTree e TNtuple

- La struttura base complessa viene spesso indicata con il termine **evento** (per esempio un evento di collisione tra due protoni).

- Il *tree* permette l'accesso alle informazioni sia **orizzontalmente** (tutte le informazioni di un certo evento) che **verticalmente** (i dati di una certa *leaf* di una certa *branch* per tutti gli eventi)



Analisi dati con ROOT

Strutture dati complesse : le classi TTree e TNtuple

- L'*ntupla* (class **TNtuple**) è una **versione semplificata del TTree** con una **lista semplice di variabili float** (o double in TNtupleD)

- **Potente strumento di analisi** che permette di fare in modo rapido i plot di una data variabile o di due variabili correlate (A vs. B), con selezioni sulle altre variabili (analisi multi-parametrica)

- Ideale per l'analisi multi-parametrica preliminare da riga di comando

  root[0] nt->Draw("D0mass","pt>2 && pt<4")

Analisi dati con ROOT

Strutture dati complesse : le classi TTree e TNtuple

- **Creazione di un'ntupla :**

  TNtuple *nt = new TNtuple("name","title","varlist");

  "varlist" = "var1:var2:var3:..."

- **Riempimento:**

  nt->Fill(value1, value2, value3);

  oppure:

  float array[3];

  array[0]=value1; array[1]=value2; array[2]=value3;

  nt->Fill(array);

Analisi dati con ROOT

Strutture dati complesse : le classi TTree e TNtuple

- **Analisi**: esempi di comandi di uso frequente
  - *nt->GetEntries(); // ritorna il numero di eventi*
  - *nt->Print(); // lista le variabili definite*
  - *nt->Draw("var1"); // disegna la distribuzione di var1 su tutti gli eventi*
  - *nt->Draw("var1:var2"); // disegna il plot di var1 vs. var2*
  - *nt->Draw("var1","var2>0 && var2<1"); // plot di var1 condizionato a var2 compresa tra 0 e 1*
  - *nt->Draw("var2","var2>0","same",1000); // plotta var2>0 sopra il precedente istogramma usando solo le prime 1000 entries*

Analisi dati con ROOT

Strutture dati complesse : le classi TTree e TNtuple

- **Analisi**: esempi di comandi di uso frequente
  - *TH1F *h=new TH1F("h","var1",1000,0,1000);*
  - *nt->Draw("var1>>h"); // disegna riempiendo h con il binning scelto*
  - *nt->Project("h","var1*var2","var2>20"); // riempie h con var1*var2 con selezione su var2, senza disegnare.*
- Recupero delle informazioni:

```
{ ...
    float v1,v2;
    nt->SetBranchAddress("var1",&v1); // connette la brach "var1" con la variabile v1
    nt->SetBranchAddress("var2",&v2);
    ...
    nt->GetEntry(i); // recupera l'evento i-esimo
    ...
}
```

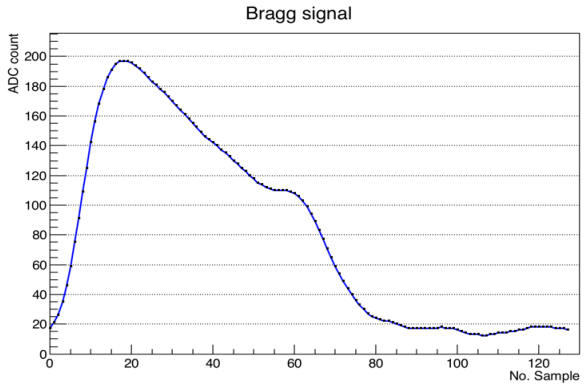
Analisi dati con ROOT

Strutture dati complesse : le classi TTree e TNtuple

Un esempio di uso di TTree

- Facciamo un esempio con un tree di strutture semplici (eventi di acquisizione dell'esperienza di Bragg):
- struttura dati:

```
struct bragg_signal {
  short int s[128];    // ogni "evento" è l'insieme dei 128
};                     campioni del segnale
```



- codice:

```
{ ...
      TTree *tree = new TTree("bragg","Bragg Signals");
      bragg_signal signal;
      TBranch *branch = tree->Branch("signals",&signal,"s[128]/s");
    ...
}
```

tree name

branch name

Analisi dati con ROOT

Strutture dati complesse : le classi TTree e TNtuple

Un esempio di uso di TTree

- riempimento :

```
{ ...
      for (int i=0; i<nev; i++) {
         for (int j=0; j<128; j++) signal.s[j] = ... ;
         branch->Fill();
      }
      ...
      tree->Write(); // salvataggio del file
      fout->Close();
}
```

Analisi dati con ROOT

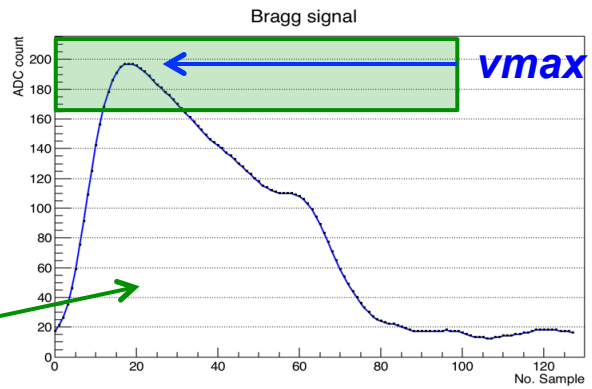
Strutture dati complesse : le classi TTree e TNtuple

Un esempio di uso di TTree

- lettura:

```
{ ...
    bragg_signal signal; // definisce un oggetto tipo bragg_signal
    TFile *fin=new TFile(filename); // apre il file
    TTree *tree = (TTree*)fin->Get("bragg"); // cerca il tree "bragg"
    TBranch *br = tree->GetBranch("signals"); // recupera la branch
    br->SetAddress(&signal); // connette la branch all'oggetto bragg_signal
    ...
     for (int i=0; i<br->GetEntries(); i++) {
          br->GetEntry(i); // carica in memoria l'evento i-esimo
          DoSomething();  // analizza l'evento usando i dati in signal
    }
}
```

ANALISI EVENTO x EVENTO



Analisi dati con ROOT

Esercizi 7: Analisi dei segnali della Camera di Bragg

La macro AnaBragg.C

- La macro AnaBragg.C legge i segnali campionati della camera registrati in un tree (di nome “bragg”) di strutture semplici (di nome “signals”, un array di 128 short)
- Fa una preanalisi dei segnali calcolando per ogni evento l’integrale del segnale in una data zona, la tensione massima e la baseline, e inserisce questi valori in un’ntupla. Infine salva l’ntupla su rootfile.
- Proviamo...

Carica la macro e compila. Funziona anche senza compilare...

```
[root [0] .L AnaBragg.C++                                                    ]
Info in <TMacOSXSystem::ACLiC>: creating shared library /Users/lunardon/Document
s/didattica/lab3/CorsoRoot/./AnaBragg_C.so
root [1] AnaBragg(
int AnaBragg(const char* filename, int intto = 100, float blfix = 13, int nsig =
 0)
[root [1] AnaBragg("ch4_500mb_thr96.root")                                   ]
Number of events in file : 3000
3000 events analyzed...
(int) 0
```

tab-completion: va inserito il nome del file (stringa). Ci sono poi altri 3 parametri con valori di default

Analisi dati con ROOT

Esercizi 7: Analisi dei segnali della Camera di Bragg

- Apriamo il file di output e vediamo cosa c'è dentro...

```
root [0] f = TFile::Open("anabragg_ch4_500mb_thr96.root");
root [1] .ls
TFile**         anabragg_ch4_500mb_thr96.root
 TFile*         anabragg_ch4_500mb_thr96.root
  KEY: TNtuple  nt;1
root [2] nt->Print()
******************************************************************************
*Tree    :nt        :                                                        *
*Entries :     3000 : Total =            63056 bytes  File  Size =      20522 *
*        :          : Tree compression factor =   3.04                       *
******************************************************************************
*Br    0 :ev        : F                                                      *
*Entries :     3000 : T                ytes  File Size  =       4294 *
*Baskets :        1 : B                ytes  Compression=   2.81     *
*.............................................................................*
*Br    1 :vmax      : Float_t                                                *
*Entries :     3000 : Total  Size=      12537 bytes  File Size  =       3798 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   3.18     *
*.............................................................................*
*Br    2 :width     : Float_t                                                *
*Entries :     3000 : Total  Size=      12542 bytes  File Size  =        155 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=  77.87     *
*.............................................................................*
*Br    3 :integral  : Float_t                                                *
*Entries :     3000 : Total  Size=      12557 bytes  File Size  =       7059 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.71     *
*.............................................................................*
*Br    4 :baseline  : Float_t                                                *
*Entries :     3000 : Total  Size=      12557 bytes  File Size  =       4568 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.64     *
*.............................................................................*
```

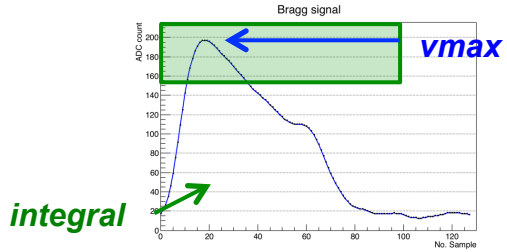
Apre il file ROOT

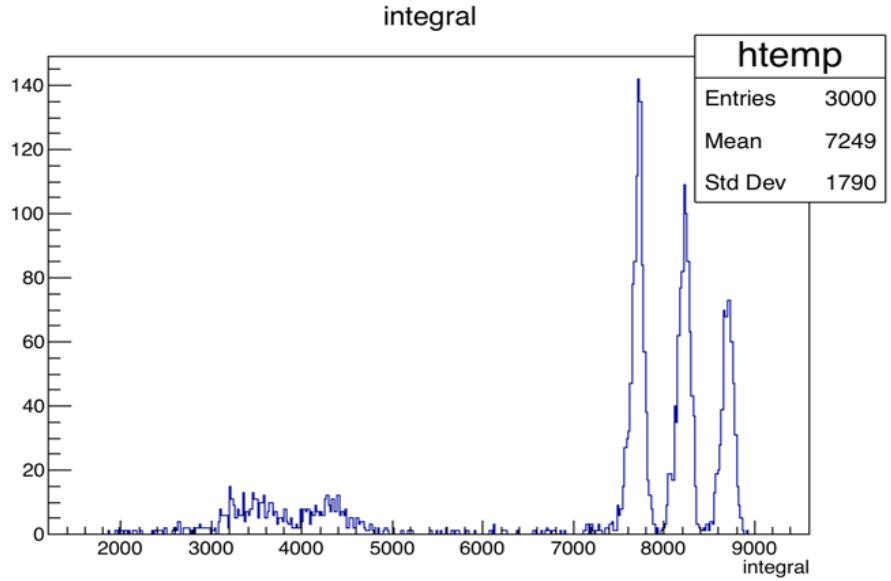
stampa la struttura del tree

branch 0 di nome "ev" (numero di evento)

Analisi dati con ROOT

Esercizi 7: Analisi dei segnali della Camera di Bragg



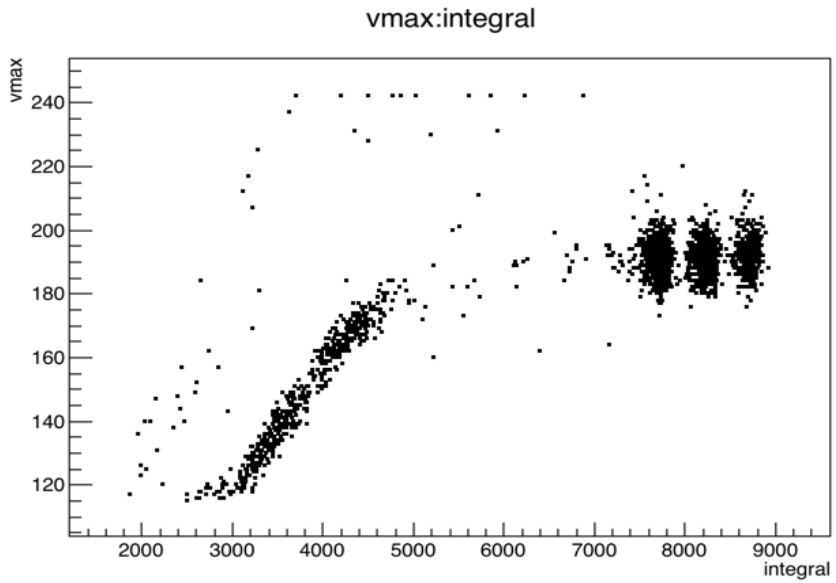


- Costruiamo l'istogramma (TH1F) degli integrali di tutti gli eventi

```
[root [3] nt->Draw("integral")
```

- Costruiamo il bidimensionale (TH2F) dei massimi dei segnali rispetto all'integrale

```
[root [9] nt->Draw("vmax:integral")
```

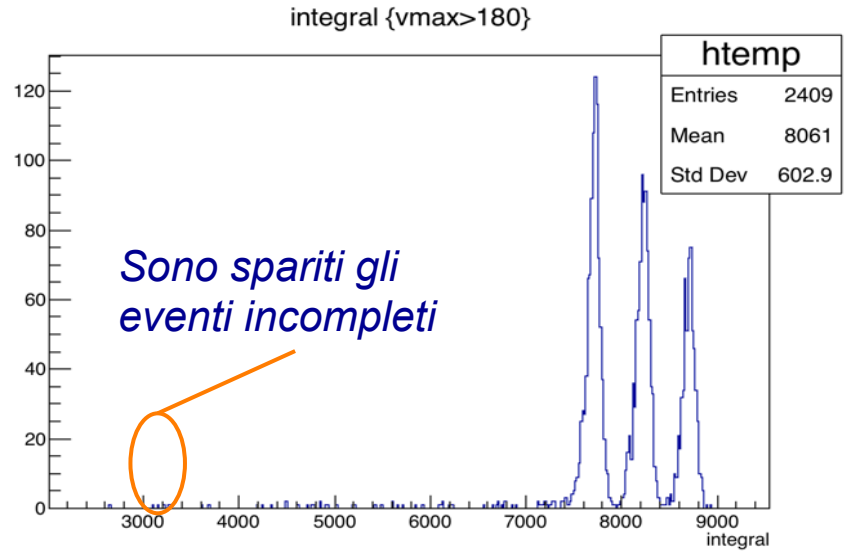


Analisi dati con ROOT

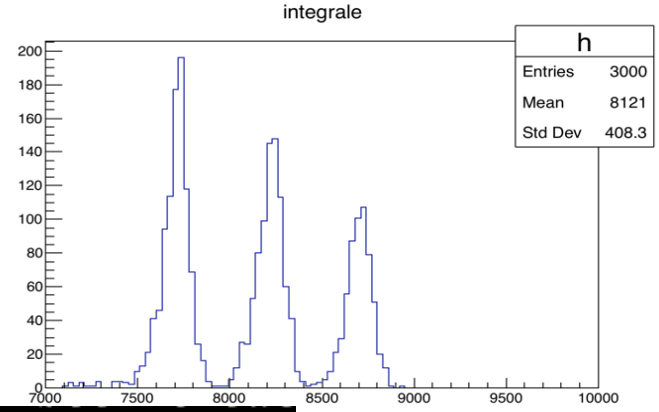
Esercizi 7: Analisi dei segnali della Camera di Bragg

- Costruiamo l'istogramma (TH1F) degli integrali di tutti gli eventi **condizionato** al fatto di avere vmax > 180

```
[root [10] nt->Draw("integral","vmax>180")
```



- Riempiamo con i dati dell'ntupla un istogramma definito in precedenza (così possiamo scegliere binning e range)




```
[root [12] TH1F *h = new TH1F("h","integrale",100,7000,10000)
 (TH1F *) 0x7fd34ce58730
[root [13] nt->Draw("integral>>h")
```

Analisi dati con ROOT

Esercizi 7: Analisi dei segnali della Camera di Bragg

Provare le seguenti operazioni:

- Costruire l'istogramma di vmax
- Costruire l'istogramma di vmax con binning adeguato (128 bin, un'unità per bin)
- Costruire il bidimensionale integrale in funzione del numero di evento
- Analizzare uno dei tre picchi dell'istogramma degli integrali (fit, centroide, larghezza...)
- ...