

Introduzione a ROOT

Corso di Introduzione alla Fisica SubNucleare

a.a. 2023/24

gianluca.lamanna@unipi.it

Installare ROOT

- Sorgenti e istruzioni: <https://root.cern/install/>
 - Di preferenza useremo versioni pre-compilate → «Download a pre-compiled binary distribution»
- ROOT nasce nativamente per sistemi LINUX, esistono versioni utilizzabili anche in altri OS, in particolare Windows e MacOS
- La versione non ci importa molto perché useremo delle feature molto basilari presenti da sempre in ROOT (le nuove versioni di solito hanno nuove funzionalità):
 - L'ultima versione stabile è del 13.4.2022: Release 6.26/02
 - <https://root.cern/releases/release-62602/>
- LINUX:
 - Pacchetti pre-compilati per diverse distribuzioni: Fedora, Centos, Ubuntu (nel seguito istruzioni per ubuntu)
 - Possibilità di compilare i sorgenti se non trovate la vostra distribuzione
 - Scaricare i packages per risolvere le dipendenze: `sudo apt-get install dpkg-dev cmake g++ gcc binutils libx11-dev libxpm-dev libxft-dev libxext-dev python libssl-dev`
 - Altri pacchetti utili: `sudo apt-get libtiff`
 - Python (ma anche altri) potrebbe essere già installato
 - Scaricare la versione voluta: `tar -xvf nome_file.tar.gz`
 - Andare in bin e fare `source thisroot.sh`
- MacOS:
 - Disponibili da 10.15 a 12.3
 - Possibile anche installare con homebrew: `brew install root`
- WINDOWS con VISUAL STUDIO:
 - Installare VISUAL STUDIO versione FREE da : <https://visualstudio.microsoft.com/it/vs/>
 - Scaricare il file exe corrispondente alla versione di visual studio installata: `root_v6.26.02.win64.vc17.exe`
- WINDOWS con WSL
 - Installare WSL per Windows: in una finestra di terminale (o powershell) digitare `wsl --install`
 - Riavviare
 - Installare XMING da <https://sourceforge.net/projects/xming/>
 - Spuntare `access control` nel setup di XLaunch e avviare XMING
 - Fare `export DISPLAY=$(cat /etc/resolv.conf | grep nameserver | awk '{print $2}'):0` (o anche `export DISPLAY=0:0`)
 - Seguire istruzione per linux

Installazione in WLS (windows)

- Metodo piu' facile (installazione automatica):
→ `sudo snap install root-framework`
- METODO ALTERNATIVO (installazione manuale):
- Installare le dipendenze (circa 300 MB):
→ `sudo apt-get install binutils cmake dpkg-dev g++ gcc libssl-dev git libx11-dev libxext-dev libxft-dev libxpm-dev python3`
→ `sudo apt-get install libtbb-dev`
→ Vedi <https://root.cern/install/dependencies/> per distribuzioni differenti da Ubuntu
- Eventualmente installare i pacchetti aggiuntivi (circa 900 MB):
→ `sudo apt-get install gfortran libpcre3-dev xlibmesa-glu-dev libglew-dev libftgl-dev libmysqlclient-dev libfftw3-dev libcfitsio-dev graphviz-dev libavahi-compat-libdnssd-dev libldap2-dev python3-dev python3-numpy libxml2-dev libkrb5-dev libgsl0-dev qtwebengine5-dev nlohmann-json3-dev`
- Anche se non si installano tutti i pacchetti aggiuntivi può comunque essere una buona idea installare `python3-dev` e `python3-numpy`

Installazione WLS (windows) (2)

- Controllare la versione di Ubuntu della WLS: `lsb_release -a`
- Scaricare la versione del precompilato idonea:
→ `wget https://root.cern/download/root_v6.30.06.Linux-ubuntu22.04-x86_64-gcc11.4.tar.gz`
- `tar xvf root_v6.30.06.Linux-ubuntu22.04-x86_64-gcc11.4.tar.gz`
- Andare in `root/bin` e fare `source thisroot.sh`

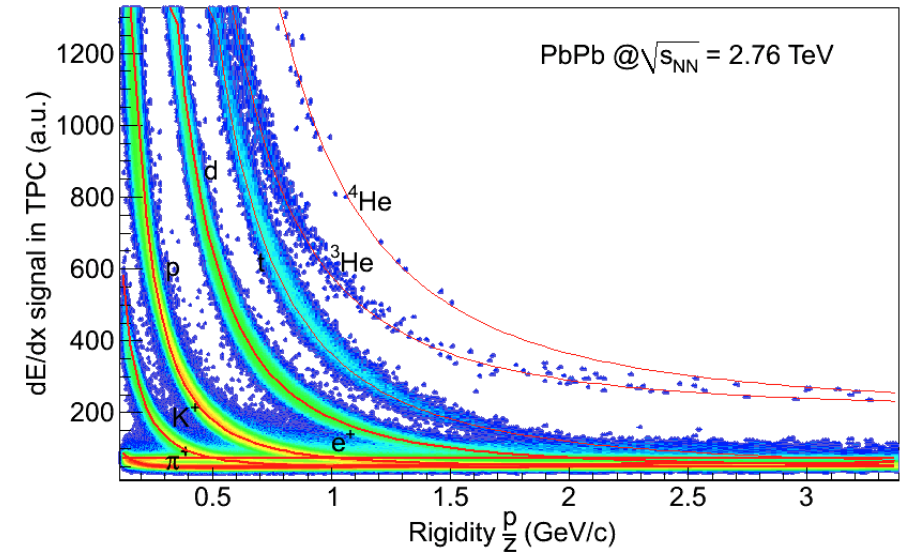
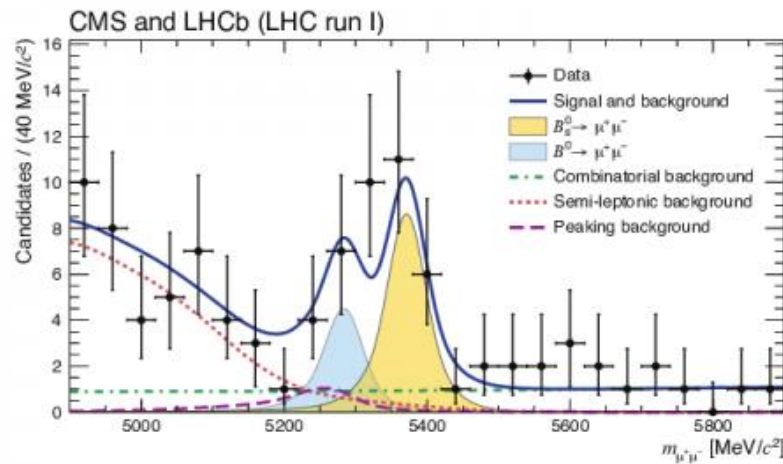
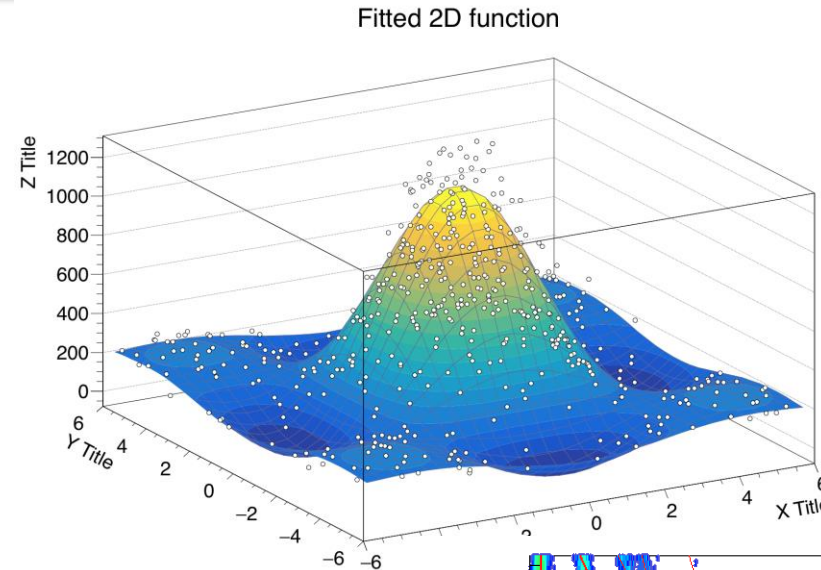
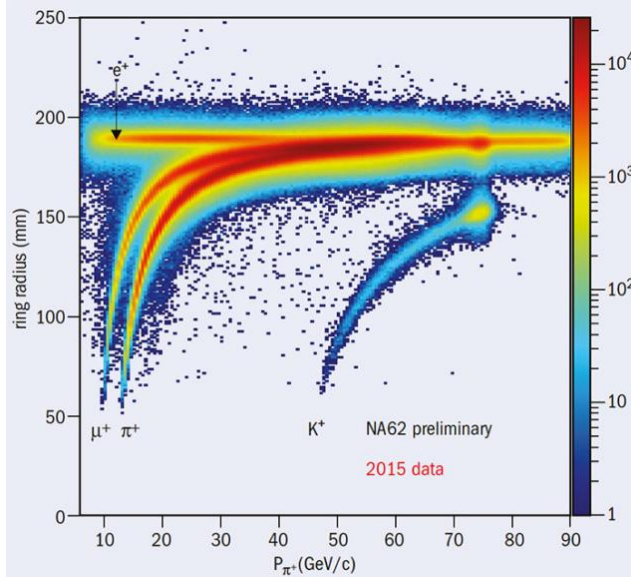
Qualche comando della shell di linux

- `ls`
 - Lista i file in una directory
- `mkdir`
 - Crea una nuova directory
- `cd mydir/mysubdir`
 - Cambia directory
- `cp myfile.txt mydir`
 - Copia un file
- `mv myfile.txt mydir`
 - Sposta un file
- `rm myfile.txt`
 - Cancella un file
- `cat myfile.txt, more myfile.txt, less myfile.txt`
 - Visualizza il contenuto di un file
- `pwd`
 - Directory corrente
- `grep pippo myfile.txt`
 - Cerca in un file
- Editors di testo
 - vi, nano, emacs, xemacs, gedit
 - `sudo apt update`
 - `sudo apt install emacs`

Cos'è ROOT?

- Root è un framework orientato agli oggetti per analisi dei dati
 - Legge i dati
 - Scrive i dati
 - Permette di selezionare i dati in base a qualche criterio
 - Produce i risultati come numeri o come plots
- Supporta una programmazione “interattiva” (come la shell di Python) oppure anche una versione compilata
- Si basa su C++, ma permette di utilizzare un wrapping di python: pyroot
- Integra moltissimi tools: generazione di numeri casuali, minimizzazione, Neural networks, analisi multivariate, ...

Cosa si può fare con ROOT

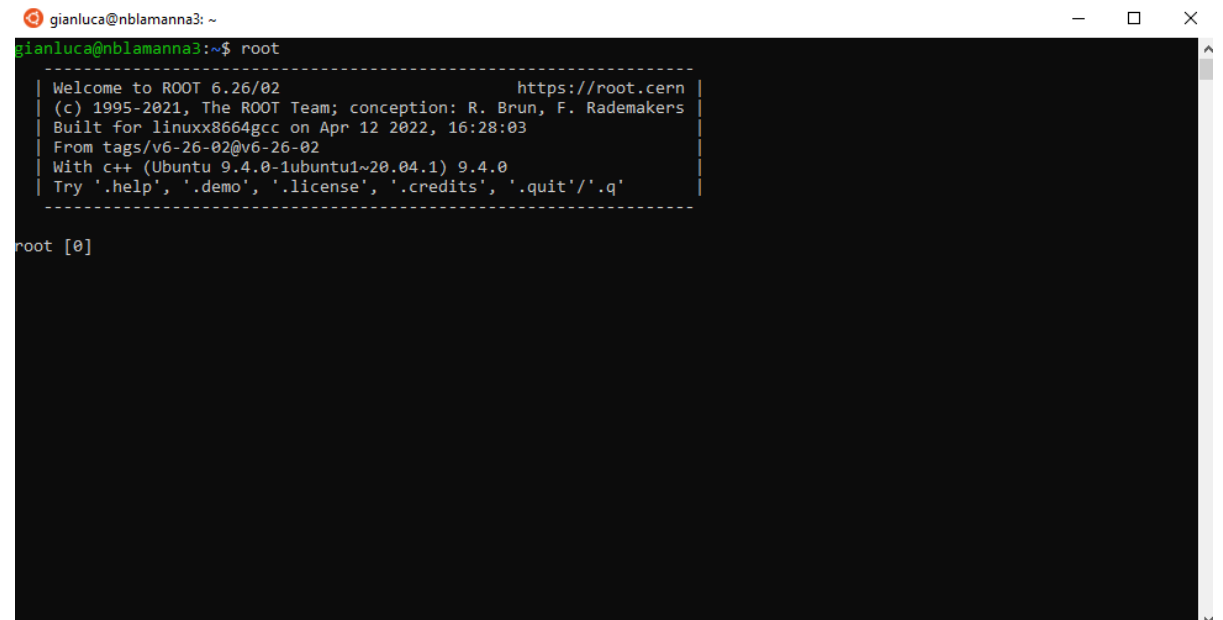


Documentazione

- Il riferimento è: <http://root.cern.ch>
- Molti manuali e molta documentazione
 - <https://root.cern/primer/>
 - https://root.cern/doc/master/group__Tutorials.html
 - <https://github.com/root-project/training/tree/master/BasicCourse>
- Pyroot
 - <https://root.cern/manual/python/>
 - https://root.cern.ch/doc/master/group__tutorial__pyroot.html
- Reference principale:
 - <https://root.cern/doc/master/index.html>

Console interattiva

- Lanciare root nella console interattiva basta scrivere “root”
 - Con `root --help` si ha qualche opzione al lancio
- L'interprete si chiama CLING
 - JIT (Just in time compilation)
- Possibilità di scrivere e eseguire “macros”
- Root può essere usato come calcolatrice
 - Vedere istruzioni della classe TMath:
[https://root.cern/doc/master/names
paceTMath.html](https://root.cern/doc/master/namespaceTMath.html)



```
gianluca@nblamanna3: ~  
gianluca@nblamanna3:~$ root  
-----  
Welcome to ROOT 6.26/02                                     https://root.cern  
| (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers  
| Built for linuxx86_64gcc on Apr 12 2022, 16:28:03  
| From tags/v6-26-02@v6-26-02  
| With c++ (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0  
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.'q'  
-----  
root [0]
```

Comandi base

- Come uscire?
 - `.q`
 - `.qqqqqqqq`
- Eseguire un comando della shell
 - `.[command]`
- Come caricare un file macro esterno? (in C)
 - `.L mymacro.C`
 - `.x myfunction.C`
- Nella console è supportato il TAB per il completamento dei comandi

Sintassi del C++

- I file si chiamano con .C o .cpp
- Compilatore per C++ è g++
- Accesso alla memoria basato sui puntatori
- Linguaggio orientato agli oggetti
 - Classi: collezioni di data type e azioni su essi
 - Data members
 - Class methods
- gli oggetti sono istanze di una classe definite da un costruttore
- Gli oggetti sono create con new e distrutti con delete

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello world!" << endl;
    return 0;
}
```

```
int a = 5
int i = a //crea un oggetto copia di «a»
int *b = &a //assegna a «b» l'indirizzo di «a»
int &r = a // «r» è una referenza a «a»
```

Alcune delle classi più comuni

TH1	Istogramma 1D (TH1S, TH1F, TH1D)
TH2	Istogramma 2D
TProfile	Istogramma profilo
TF1	Funzioni 1D
TLine	Linea nel piano
TMarker	Marker
TGraph	Vettore di punti nel piano
TLatex	Scrittura testo
TLegend	Legenda
TFile	File ROOT
TCanvas	Finestra grafica di disegno
TTree	Ntuple dati
TRandom	Numeri casuali

Tutti i metodi sono
ampiamente descritti
sul sito di ROOT

Funzioni

- Si usa la classe TF1 per definire una funzione

```
TF1 f1("f1", "sin(x)/x", 0.,10.)  
f1.Draw()
```

- Si possono anche aggiungere dei parametri

```
TCanvas *c2 = new TCanvas("c2", "Funzione2")  
TF1 f2("f2", "[0]*sin([1]*x)/x", 0.,10.)  
f2.SetParameters(1,1)  
f2.Draw()
```

- Premere “View” → “Editor” nel menu a tendina per interagire con il plot
- Premere anche su Toolbar

Funzioni (in Python)

- Si usa la classe TF1 per definire una funzione

```
import ROOT  
f1=ROOT.TF1("f1", "sin(x)/x", 0.,10.)  
f1.Draw()
```

- Si possono anche aggiungere dei parametri

```
c2=ROOT.TCanvas("c2","Funzione2")  
f2=ROOT.TF1("f2","[0]*sin([1]*x)/x",0,10)  
f2.SetParameters(1,2)  
f2.Draw()  
c2.Update()
```

- Premere “View” → “Editor” nel menu a tendina per interagire con il plot
- Premere anche su Toolbar

Plot con gli errori (in C)

- Proviamo a plottare dei punti con gli errori

```
TGraphErrors gr("ExampleData.txt")  
gr.Draw("AP")
```

- Altro esempio di grafico

```
TGraph g  
g.SetTitle("My graph;myX;myY")  
g.SetPoint(0,1,0)  
g.SetPoint(1,2,3)  
g.SetMarkerStyle(kFullSquare)  
g.SetMarkerColor(kRed)  
g.SetLineColor(kOrange)  
g.Draw("APL")
```

```
gianluca@nblamanna3:~$ more ExampleData.txt  
# fake data to demonstrate the use of TGraphErrors  
  
# x    y    ex    ey  
1.    0.4  0.1   0.05  
1.3   0.3  0.05  0.1  
1.7   0.5  0.15  0.1  
1.9   0.7  0.05  0.1  
2.3   1.3  0.07  0.1  
2.9   1.5  0.2   0.1  
gianluca@nblamanna3:~$
```

Plot con gli errori (in Python)

- Proviamo a plottare dei punti con gli errori

```
gr=ROOT.TGraphErrors("FakeData.txt")  
gr.Draw("AP")
```

- Altro esempio di grafico

```
gr.SetTitle("My graph;myX;myY")  
gr.SetPoint(0,1,0)  
gr.SetPoint(1,2,3)  
gr.SetMarkerStyle(ROOT.kFullSquare)  
gr.SetMarkerColor(ROOT.kRed)  
gr.SetLineColor(ROOT.kOrange)  
gr.Draw("APL")
```

```
gianluca@nblamanna3:~$ more ExampleData.txt  
# fake data to demonstrate the use of TGraphErrors  
  
# x    y    ex    ey  
1.    0.4  0.1    0.05  
1.3   0.3  0.05   0.1  
1.7   0.5  0.15   0.1  
1.9   0.7  0.05   0.1  
2.3   1.3  0.07   0.1  
2.9   1.5  0.2    0.1  
gianluca@nblamanna3:~$
```


Lettura dei dati (in C)

- Root può leggere i dati da varie sorgenti:
→ File, network, database
- Normalmente i dati vengono conservati in oggetti che si chiamano TTree oppure Tntuple
→ I trees e le ntuples possono essere viste come delle tabelle, ogni riga rappresenta un “evento”, ogni colonna rappresenta una quantità/proprietà associata a quell’evento
- Esempio:

```
TNtuple dati("dati", "dati", "volts:ampere:width")
dati.ReadFile("dati.txt")
dati.Scan()
dati.Scan("ampere")
```

```
gianluca@nblamanna3:~$ more dati.txt
#volts ampere width
3.1 5. 2.
4.3 7. 2.
5.4 3. 1.
1.3 2. 3.2
4.5 3. 1.2
2.3 1. 2.1
gianluca@nblamanna3:~$
```

Lettura dei dati (in Python)

- Root può leggere i dati da varie sorgenti:
→ File, network, database
- Normalmente i dati vengono conservati in oggetti che si chiamano TTree oppure Tntuple
→ I trees e le ntuples possono essere viste come delle tabelle, ogni riga rappresenta un “evento”, ogni colonna rappresenta una quantità/proprietà associata a quell’evento
- Esempio:

```
dati=ROOT.TNtuple("dati", "dati", "volts:ampere:width")
dati.ReadFile("dati.txt")
dati.Scan()
dati.Scan("ampere")
```

```
gianluca@nblamanna3:~$ more dati.txt
#volts ampere width
3.1  5.  2.
4.3  7.  2.
5.4  3.  1.
1.3  2.  3.2
4.5  3.  1.2
2.3  1.  2.1
gianluca@nblamanna3:~$
```

Salvataggio di un Ntupla (in C)

- Possiamo salvare un ntupla in un file

```
TFile f("rootfile.root", "CREATE")  
f.cd()  
dati.Write()  
f.Close()
```

- E anche rileggerla dal file

```
TFile f("rootfile.root")  
f.ls()
```

Salvataggio di un Ntupla (in Python)

- Possiamo salvare un ntupla in un file

```
f=ROOT.TFile("rootfile.root", "CREATE")  
f.cd()  
dati.Write()  
f.Close()
```

- E anche rileggerla dal file

```
f=ROOT.TFile("rootfile.root")  
f.ls()
```

Draw di un Ntupla (in C)

- Avendo un ntupla è possibile fare il Draw di una Colonna
→ Questo genera un istogramma
- E' possibile applicare dei tagli per "selezionare" alcuni eventi
→ Caricare `simpletree.root` : anche con `root simpletree.root`

```
tree1->Draw("px")  
tree1->Draw("px", "py>2.")
```

- Possiamo fare istogrammi bidimensionali con "px:py"
- Normalmente si producono istogrammi
→ E' buon norma definire gli istogrammi a priori e poi proiettare il risultato dell'ntupla sopra

Draw di un Ntupla (in Python)

- Avendo un ntupla è possibile fare il Draw di una Colonna
→ Questo genera un istogramma
- E' possibile applicare dei tagli per "selezionare" alcuni eventi
→ Caricare `simpletree.root` : anche con `root simpletree.root`

```
tree1.Draw("px")  
tree1.Draw("px","py>2.")
```

- Possiamo fare istogrammi bidimensionali con "px:py"
- Normalmente si producono istogrammi
→ E' buon norma definire gli istogrammi a priori e poi proiettare il risultato dell'ntupla sopra

I Tree

- I tree sono molto simili alle ntuple
→ Organizzazione gerarchica delle informazioni (Tree, Branch, leaf)
- Lettura:

Metodo veloce

```
TFile *f = new TFile("tree2.root");  
TTree *tree = (TTree*)f->Get("t2");  
tree->Print();  
tree->Draw("vect");
```

Metodo preciso

```
void main(){  
    TFile *f = new TFile("tree2.root");  
    TTree *tree = (TTree*)f->Get("t2");  
    // associa i TBranch alle variabili del programma  
    double varA, varB;  
    tree->SetBranchAddress("var1",&varA);  
    tree->SetBranchAddress("var2",&varB);  
    // legge le entry  
    for (int i=0; i<tree->GetEntries(); i++)  
    {  
        tree->GetEntry(i);  
        // ... altre operazioni con la entry ...  
    }
```

I Tree (in Python)

- I tree sono molto simili alle ntuple
 - Organizzazione gerarchica delle informazioni (Tree, Branch, leaf)
- Lettura:

```
f=ROOT.Tfile("simpletree.root")  
mytree=f.Get("tree1")  
mytree.Draw("px")  
mytree.Print()
```

```
f=ROOT.Tfile("simpletree.root")  
mytree=f.Get("tree1")  
nn=mytree.GetEntries()  
For in in range(nn):  
    mytree.GetEntry(i)  
    my_px=mytree.px  
    my_py=mytree.py
```


Tree

- Scrittura

```
TTree* t = new TTree("t","nuovo  
tree");  
  
double var1, var2;  
  
t->Branch("var1",&var1,"var1/D");  
t->Branch("var2",&var2,"var2/D");  
  
var1 = 5.0;  
var2 = 1.2;  
  
t->Fill();  
  
var1 = 4.5;  
var2 = 1.8;  
  
t->Fill();  
  
t->Write();
```

- Molte cose sui tree sono semplificate dal metodo MakeClass che genera automaticamente una classe che può operare sul tree, con dei metodi base come il metodo Loop → Consiglio di guardare la documentazione

Tree

- Scrittura

```
TTree* t = new TTree("t","nuovo  
tree");  
  
double var1, var2;  
  
t->Branch("var1",&var1,"var1/D");  
t->Branch("var2",&var2,"var2/D");  
  
var1 = 5.0;  
var2 = 1.2;  
  
t->Fill();  
  
var1 = 4.5;  
var2 = 1.8;  
  
t->Fill();  
  
t->Write();
```

```
mytree=ROOT.TTree()  
from array import array  
ampere = array('f',[0])  
mytree.Branch('ampere',ampere,'ampere/F')  
volts = array('f',[0])  
mytree.Branch('volts',volts,'volts/F')  
for i in range(tot_events):  
    mytree.Fill()
```

- Molte cose sui tree sono semplificate dal metodo MakeClass che genera automaticamente una classe che può operare sul tree, con dei metodi base come il metodo Loop → Consiglio di guardare la documentazione

Booking di istogrammi (in C)

- Le classi sono TH1F, TH2F, TH3F, TH1D, ...

```
TH1F hist("hist", "hist", 20,0,400)  
tree1->Draw("px>>hist")
```

- Ora che abbiamo il plot su un istogramma ci possiamo fare tante cose
→ <https://root.cern/manual/histograms/>
- Possiamo avere la media (hist.GetMean()), il bin con il Massimo (hist.GetMaximum()), il contenuto di un bin (hist.GetBinContent(int bin_number)).
- Possiamo cambiare il colore, la linea etc. etc.

Booking di istogrammi (in Python)

- Le classi sono TH1F, TH2F, TH3F, TH1D, ...

```
hist=ROOT.TH1F("hist", "hist", 20,0,400)  
tree1.Draw("px>>hist")
```

- Ora che abbiamo il plot su un istogramma ci possiamo fare tante cose
→ <https://root.cern/manual/histograms/>
- Possiamo avere la media (hist.GetMean()), il bin con il Massimo (hist.GetMaximum()), il contenuto di un bin (hist.GetBinContent(int bin_number)).
- Possiamo cambiare il colore, la linea etc. etc.

Fit di histogrammi (in C)

- Avendo un istogramma ci sono molte funzioni per fare il fit
- Le più semplici sono del tipo

```
hist->Fit("gauss")  
hist->GetFunction("gauss")->GetParameter("Mean")  
hist->GetFunction("gauss")->GetParError(1)
```

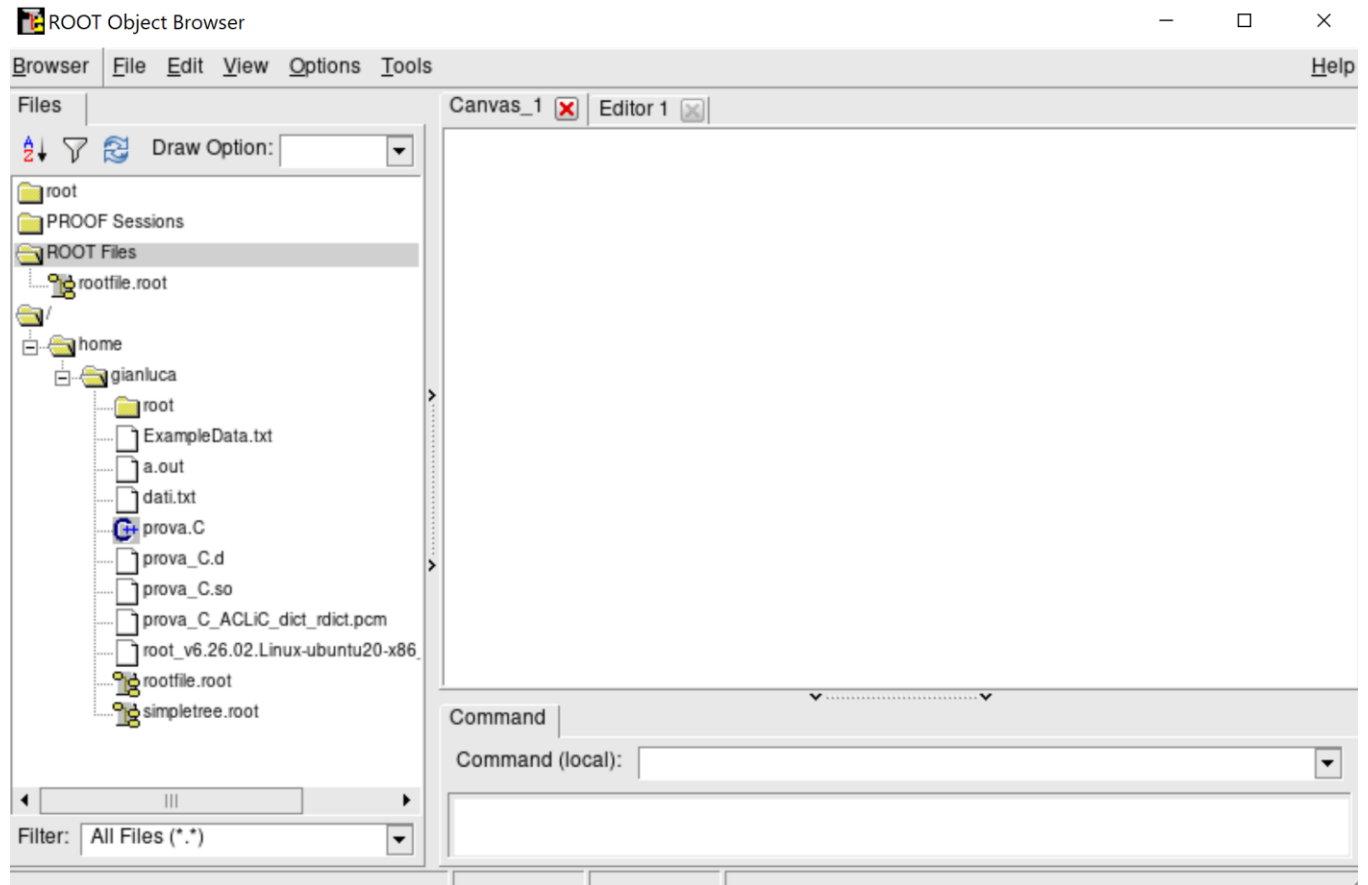
Fit di histogrammi (in Python)

- Avendo un istogramma ci sono molte funzioni per fare il fit
- Le più semplici sono del tipo

```
hist->Fit("gauss")  
hist->GetFunction("gauss")->GetParameter("Mean")  
hist->GetFunction("gauss")->GetParError(1)
```

TBrowser

- All'interno di una sessione di ROOT si può aprire un Tbrowser
 - TBrowser b
 - new TBrowser()
- Questo permette di guardare la struttura di un file, con all'interno i tree e gli istogrammi etc.



Macros

- Le macro di ROOT sono dei programmi leggeri che permettono di eseguire del codice in sequenza
- Le stesse cose che scriviamo in interattivo possono essere salvate in una macro

```
void MacroName() {  
  
    //lines of C++ code  
  
}
```


Macros

- La macro si esegue in un terminale
→ `root MacroName.C`
- Oppure al prompt di ROOT
→ `.x MacroName.C`
- Oppure si carica dentro root e poi si esegue come funzione
→ `.L MacroName.C; MacroName()`
- Per migliorare le prestazioni si possono compilare invece di interpretare le macros
→ Basta aggiungere un + → `.L MacroName.C+`
- Si possono anche produrre dei programmi che runnano da soli a partire dalle macro
→ `g++ myMacro.C 'root-config --cflag --libs' -o myMacro`

Esempio di Macro

- Vediamo un esempio di come si potrebbe fare una Macros per fare la massa di un bosone Z e fare un fit con una gaussiana

```
void fitZmacros() {  
    TFile *inf = TFile::Open("Zbosons.root");  
    TTree *ztree = (TTree*)inf->Get("ztree");
```

Apri il file e apri il tree

```
    float Zmass;  
    int Zcharge;  
    ztree->SetBranchAddress("Zmass", &Zmass);  
    ztree->SetBranchAddress("Zcharge", &Zcharge);
```

Inizializza il tree

```
    TH1F *hmass = new TH1F("hmass", "", 60, 60, 120);
```

Definisci l'istogramma

```
    for(int j=0; j<ztree->GetEntries(); j++) {  
        ztree->GetEntry(j);
```

Loopa sul tree

```
if (Zcharge != 0) continue;  
hmass->Fill(Zmass);  
}
```

Riempi l'istogramma

```
TCanvas *c1 = new TCanvas("c1", "Dimuon mass",  
600, 600);
```

Definisci la canvas

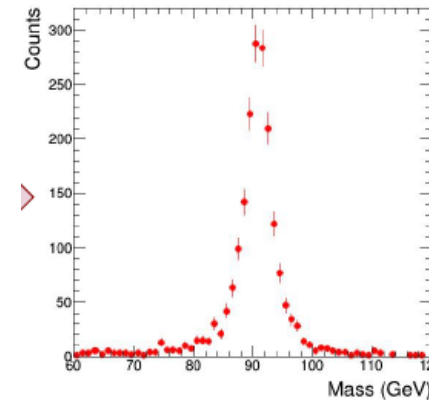
```
hmass->Draw("ep");
```

Disegna l'istogramma con gli errori

```
c1->SaveAs("./Zpeak.png");
```

Produci il plot

- A questo punto si può eseguire
→ Ad esempio: root fitZboson.C+



- Ulteriori migliorie

```
hmass->GetXaxis()->SetTitle("Mass (GeV)");  
hmass->GetYaxis()->SetTitle("Counts");  
hmass->GetXaxis()->SetTitleSize(0.05);  
hmass->GetYaxis()->SetTitleSize(0.05);  
hmass->GetYaxis()->SetTitleOffset(1.2);
```

```
gStyle->SetOptStat(0);
```

- Ora vogliamo fare il fit con una Breit-Wigner e una gaussiana
→ Definiamo la funzione che ci interessa

```
Double_t RBWGaus(Double_t *x, Double_t *par) {  
    //Fit parameters: ...  
    //Setup for the integral ...  
    for(Double_t i=1.0; i<=np/2; i++) {  
        xx = xlow + (i-.5) * step;  
        fbw = TMath::BreitWigner(xx,par[1],par[0]);  
        sum += fbw * TMath::Gaus(x[0],xx,par[3]);  
        //other side...  
    }  
    return (par[2] * step * sum * (1./sqrt  
(2*TMath::Pi())) / par[3]);  
}
```

```
TF1 *f = new TF1("f",RBWGaus,60,120,4);
```

- Prepariamo il fit

```
f->SetParameters(2.495, 91.0, 2000.0, 2.0);  
f->SetParNames("BW width", "BW mean", "Area",  
"Sigma");
```

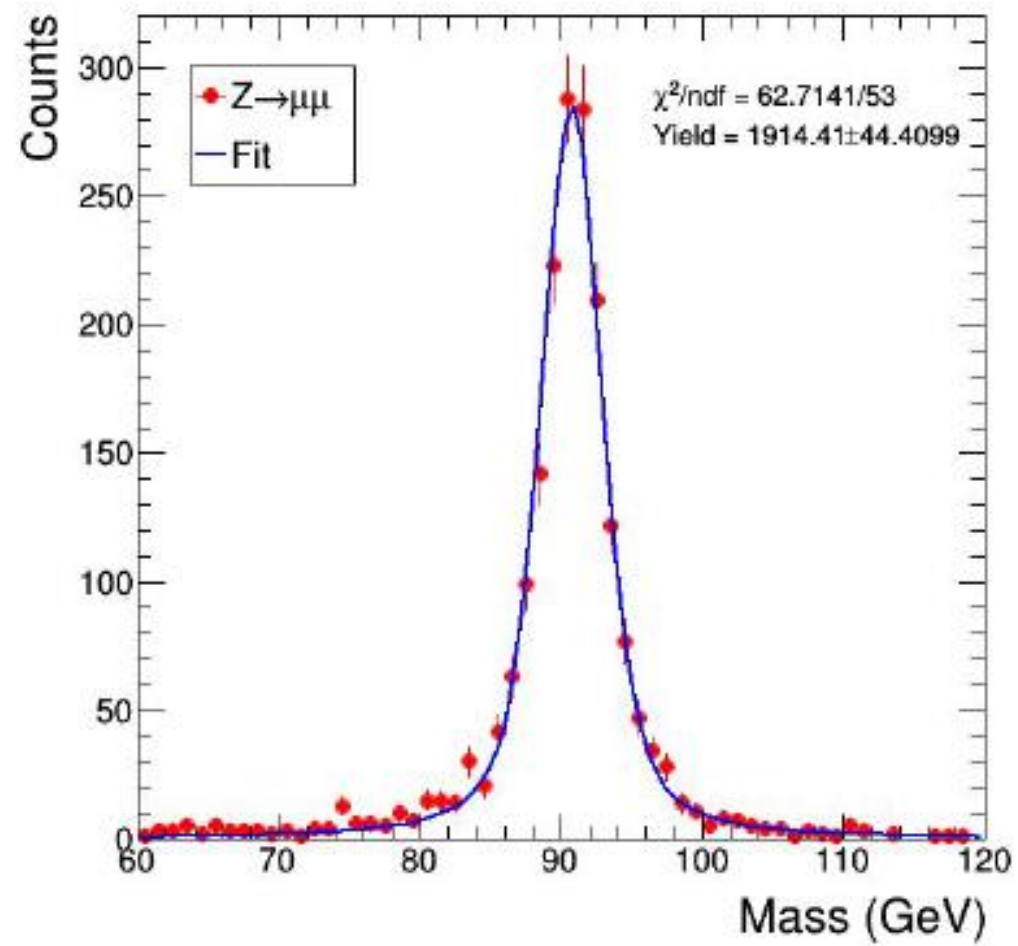
```
f->FixParameter(0, 2.495); //PDG value  
f->SetParLimits(1, 86, 96);
```

```
TFitResultPtr fitr = hmass->Fit(f, "RNS", "");
```

```
TLegend *l = new TLegend(0.18,0.78,0.34,0.90);  
l->SetTextSize(0.04);  
l->AddEntry(hmass,"Z#rightarrow#mu#mu","lp");  
l->AddEntry(f,"Fit","l");  
l->Draw();
```

```
TLatex *tx = new TLatex();  
tx->SetTextSize(0.03);  
tx->SetTextAlign(12);  
tx->SetFont(42);  
tx->SetNDC(kTRUE);
```

```
tx->DrawLatex(0.63,0.87,Form("#chi^{2}/ndf = %  
g/%d",fitr->Chi2(),fitr->Ndf()));
```



- Cosa fare:

- Familiarizzare con Root
- Provare a creare un istogramma, un grafico, fare un fit dai dati della sorgente radioattiva acquisiti in aula (vedi sito elearning)
- Provare a fare un'ntupla dai dati (segnale e fondo)
- Provare a sottrarre il fondo dal segnale
- Fare qualche esempio sul sito della documentazione di root
- (Provare a scrivere una makeclass per creare istogrammi a partire da un tree)