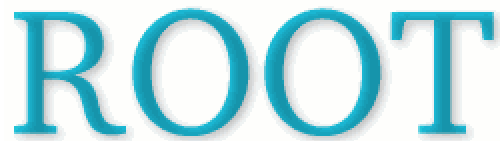


An Object-Oriented
Data Analysis Framework

Luciano Pandola

INFN-LNGS

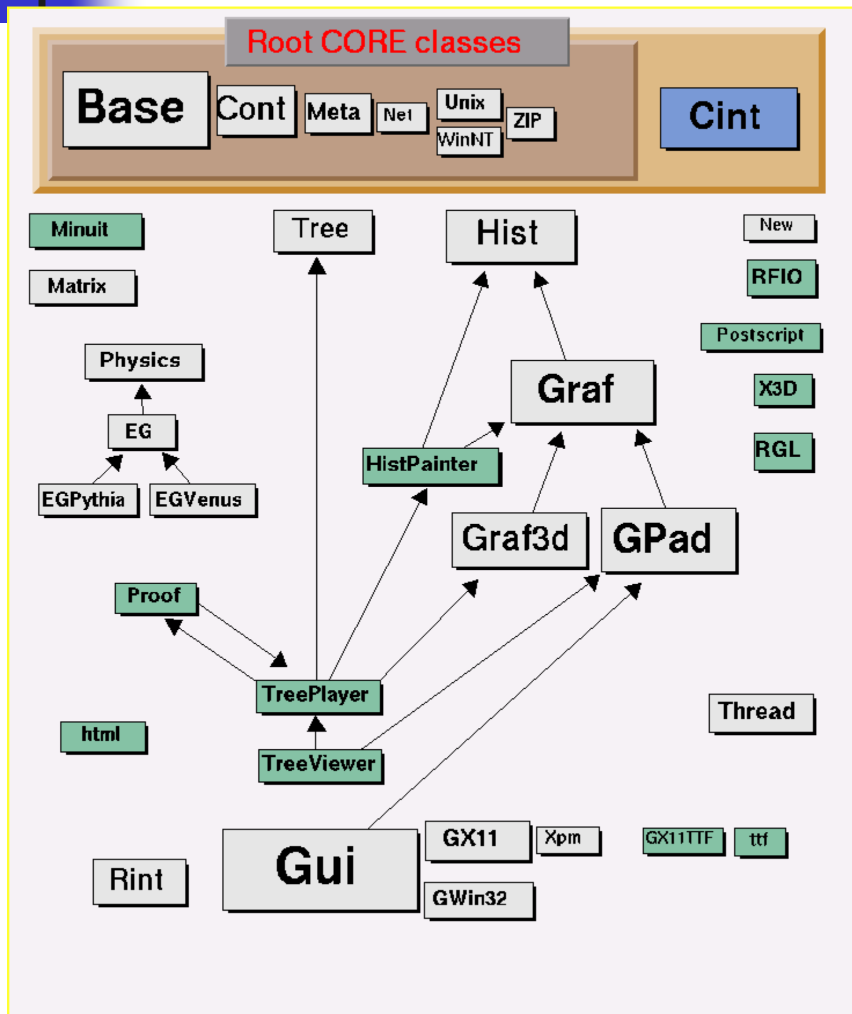
Corso INFN su C++, ROOT e Geant4

Presentazione basata da S. Panacek e N. Di Marco

Cosa può fare ROOT

- Istogrammi e fit
- Grafici e scatter plot (2D, 3D)
- **I/O** su file
  - specializzato per istogrammi ed Ntuple (TTrees)
- Supporto per analisi dati
- Interfaccia utente
  - GUI: Browsers, Panelli, Tree Viewer
  - Interfaccia a linea di comando: interprete C++ (CINT)
  - Processore per gli script (C++ compilato ⇔ C++ interpretato)
  - Possibilità di usare le classi di ROOT in programmi esterni

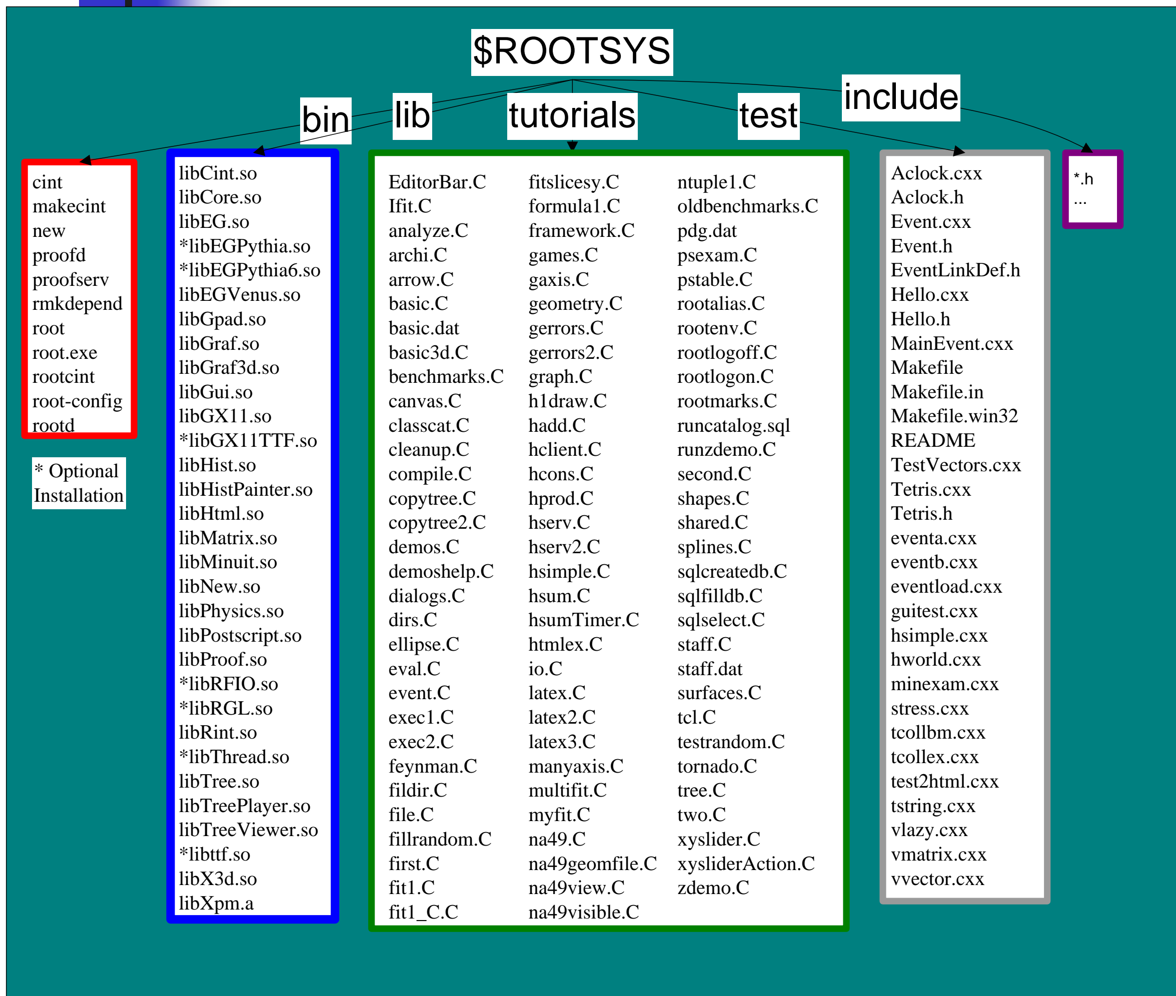
Le librerie di ROOT



- Oltre **350 classi**

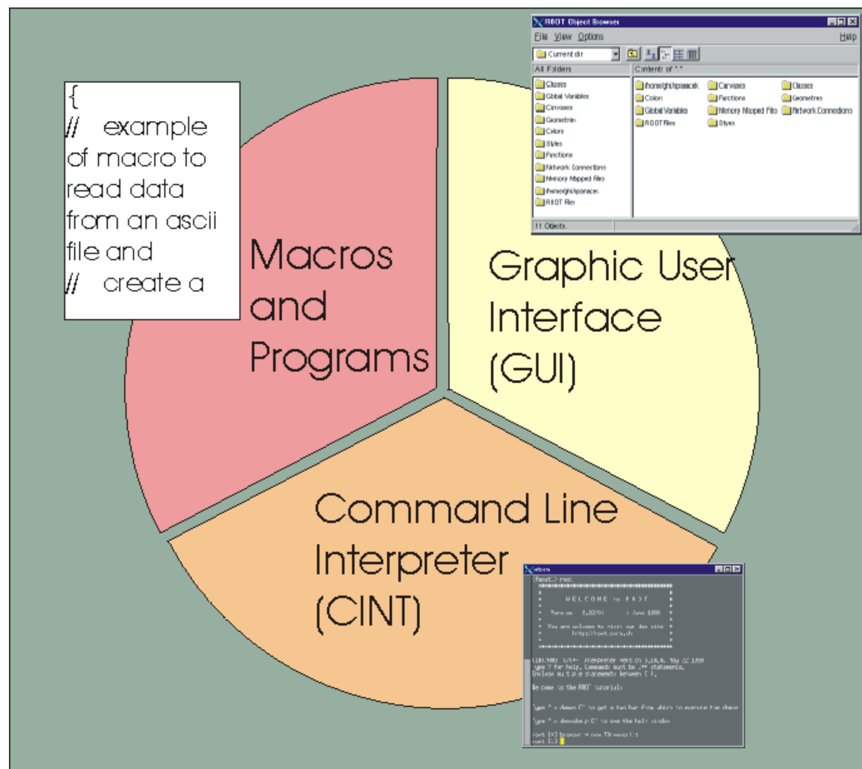
- Kernel (Core di ROOT)
- Interprete CINT
- Librarie caricate all'inizio: Hist, Tree ...
- Librarie caricate quando ce n'è necessità: HistPainter, TreePlayer,...
- Librerie specifiche: EG (event generator), Physics, Minuit...

L’organizzazione del framework



- Il tutto è controllato dalla variabile di ambiente **$ROOTSYS**
- directories per i *binari*, gli *includes* e le *librerie* compilate
- tutorial specifici per varie applicazioni utente

Come si parla con ROOT?



- **GUI** interattiva
  - bottoni, menu grafici, etc.
- Linea di comando interattiva di ROOT via CINT (**interprete** C++)
- Macro, applicazioni esterne con classi di ROOT, librarie (**compilatore** C++)

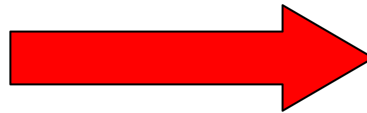
Cosa fare con la GUI?

- Ricerca ed apertura di files ROOTs
- Disegno di istogrammi
- Menu contestuali con i tre tasti del mouse
- Pannello di Disegno
  - scelta dei parametri, colori, etc.
- Pannello di Fit
  - scelta di parametri, limiti, etc.
- Aggiunta di testo, legende e altri oggetti
- Divisione della finestra (`TCanvas`) in più parti
  - Selezione di scale logaritmiche

La GUI di ROOT

Entrata in root

> > **root**



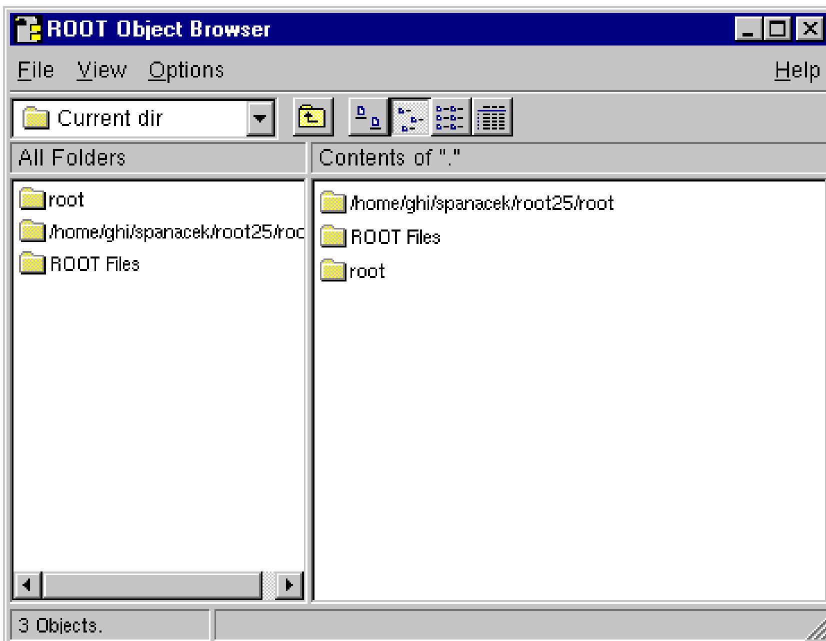
```
xterm
  *******************************************
  *                                         *
  *        W E L C O M E  to  R O O T       *
  *                                         *
  *   Version   3.01/00     23 April 2001   *
  *                                         *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch            *
  *                                         *
  *******************************************

FreeType Engine v1.x used to render TrueType fonts.

CINT/ROOT C/C++ Interpreter version 5.14.83, Apr 5 2001
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

Uscita da root

> **root[0] .q**



Apertura del browser

> **TBrowser b;**

Elementi di base di ROOT

c1

File Edit View Options Inspect Classes Help

finestra grafica

File Edit View Options Inspect Classes Help

ROOT session

```
C:\Documents and Sett
  ********************
  *
  *        W E L C O
  *
  *   Version    5.08/
  *
  *  You are welcome
  *          http://r
  *
  ********************

Compiled on 15 Decemb

CINT/ROOT C/C++ Inter
Type ? for help. Comm
Enclose multiple stat
root [0] TCanvas c
root [1] TCanvas *d = new TCanvas("mycanvas"
root [2] d->Divide(2,2)
root [3]
```

Funzioni e istogrammi

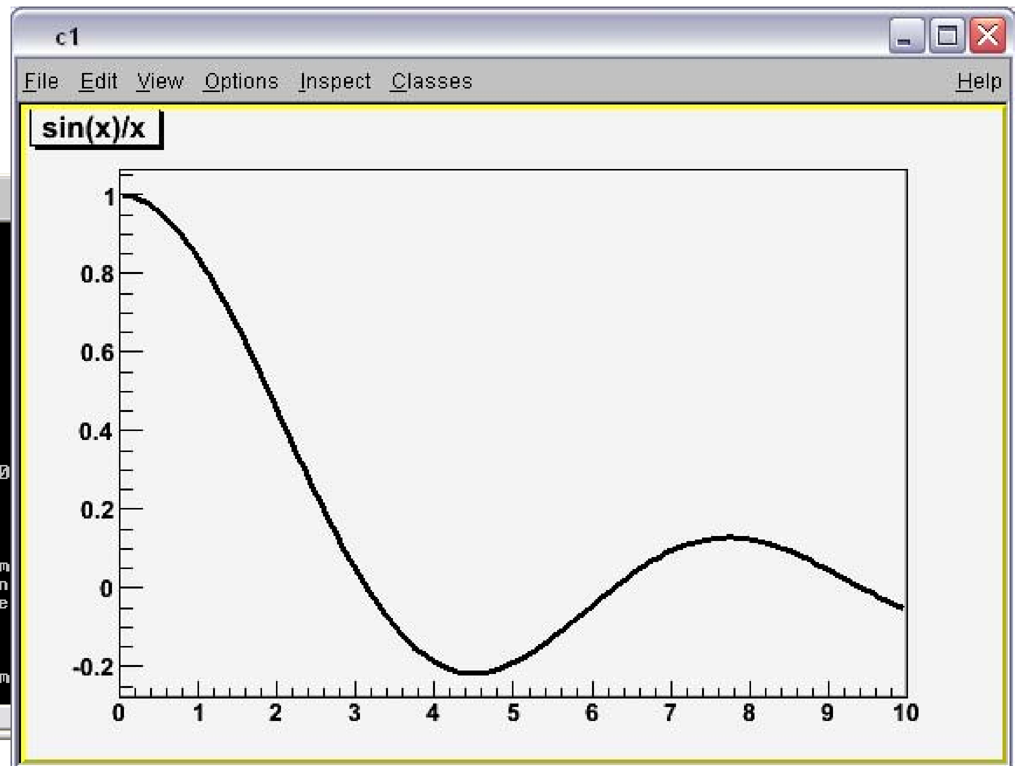
Funzioni 1D

- **TF1** → funzioni 1D *f(x)*
  - ogni funzione individuata da un nome (stringa)
  - ci sono alcune funzioni predefinite
    - "gaus", "expo", "pol0", ..., "polN"
- Le funzioni utente possono essere definite
  - a partire da quelle esistenti, e.g. "gaus+expo"
  - dando le formule esplicite in formato testo, e.g. "sin(x)/x"
  - per funzioni complesse, scrivendo una function che restituisce *f(x)* dato x (e i parametri liberi)

Plot di una funzione

nome formula range

```
[ ] TF1 f1("func1","sin(x)/x",0,10)
[ ] f1.Draw()
```



ROOT session

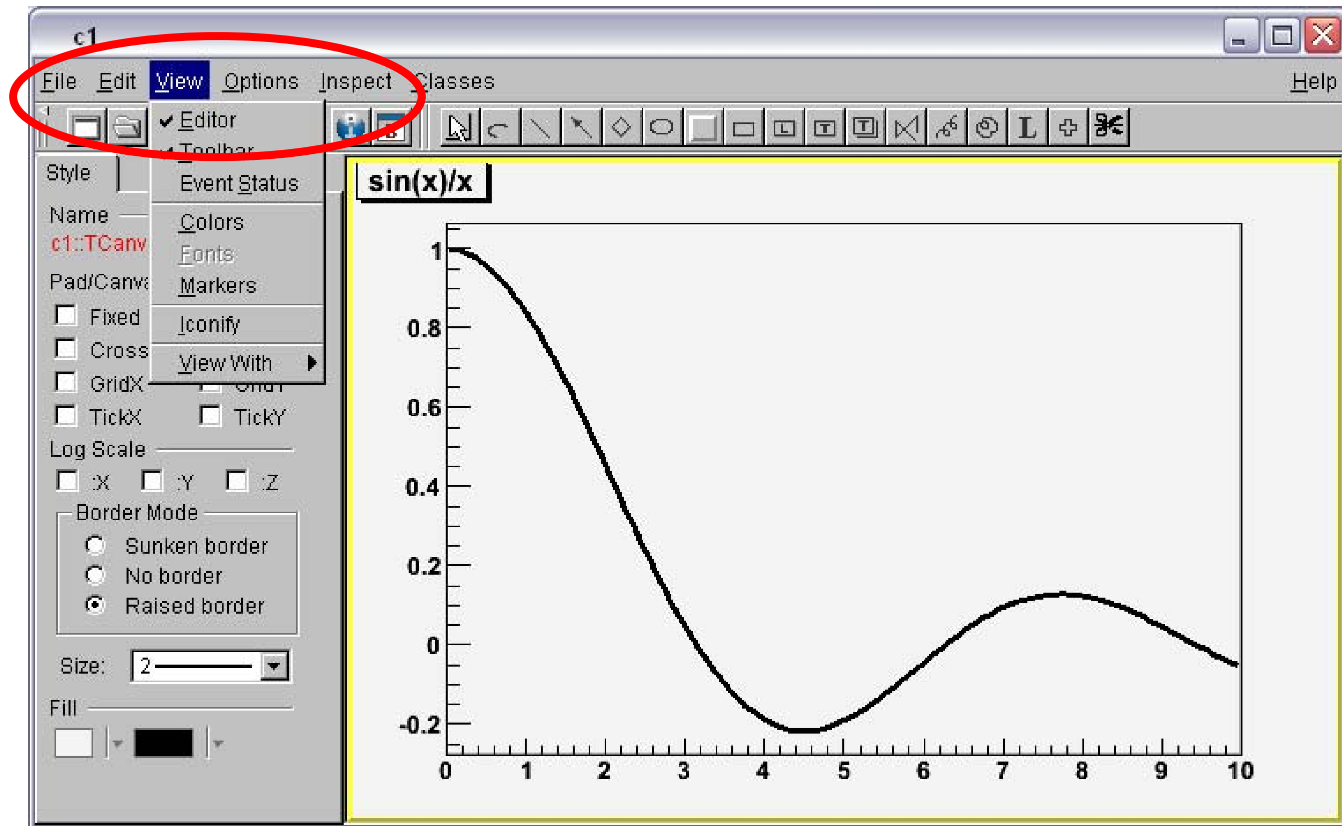
```
  *******************************************
  *                                         *
  *        W E L C O M E  to  R O O T       *
  *                                         *
  *   Version   5.08/00   13 December 2005  *
  *                                         *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch            *
  *                                         *
  *******************************************

Compiled on 15 December 2005 for win32.

CINT/ROOT C/C++ Interpreter version 5.16.5, November 30 20
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] TF1 f1(
TF1 TF1()
TF1 TF1(const char* name, const char* formula, Double_t xm
TF1 TF1(const char* name, Double_t xmin, Double_t xmax, In
TF1 TF1(const char* name, void* fcn, Double_t xmin, Double
TF1 TF1(const TF1& f1)
root [0] TF1 f1("func1", "sin(x)/x", 0, 10)
root [1] f1.Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with nam
root [2] _
```

Modificare un grafico esistente

- Selezione interattiva di:
  - dimensione assi
  - titoli assi
  - colori
  - stili
  - spessori
  - scale log
  - griglie

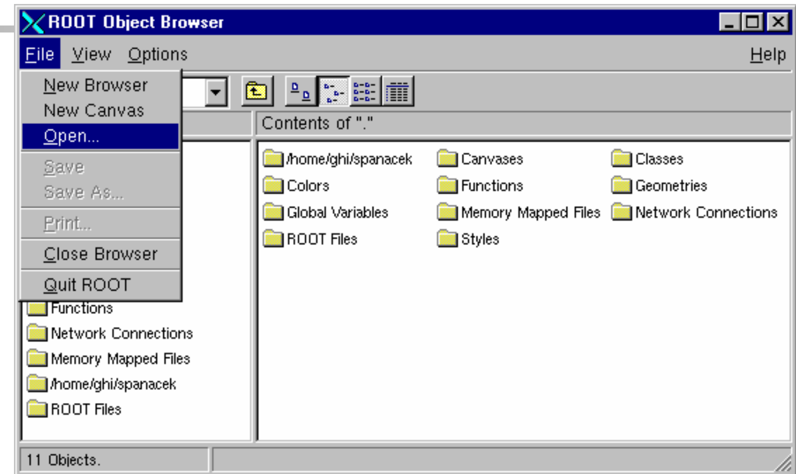


Istogrammi

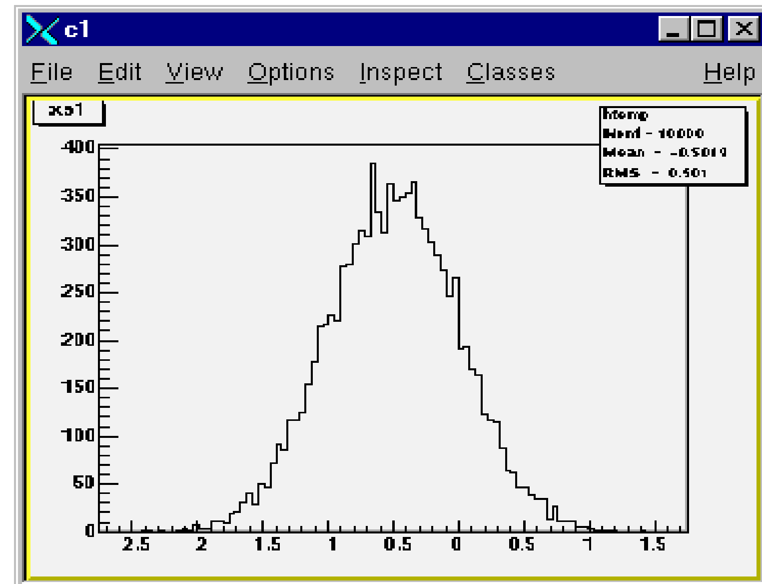
- `TH1D`, `TH1F`, `TH1I` → istogrammi 1D in ROOT
  - D = double, F = float, I = int
  - si specifica nome (stringa), titolo, numero di bins e range
- Esistono TH2* e TH3* per istogrammi a più dimensioni
- Metodi principali da linea di comando:
  - `->Draw();`
  - `->Fill(value,weigth);`
  - `->SetBinContent(i,value);`
    - Nota: il bin 0 contiene gli underflow e il (N+1) gli overflow

Disegno di un istogramma

Aprire il file di ROOT con il Browser, cercare l'istogramma dal file



Il doppio clic sull'istogramma apre automaticamente la canvas



Creazione di un istogramma

nome — titolo — bins — range

```
[ ] TH1F h1("hist,"Histogram from a gaussian",100,-3,3);
[ ] h1.FillRandom("gaus",100000);
[ ] h1.Draw()
```

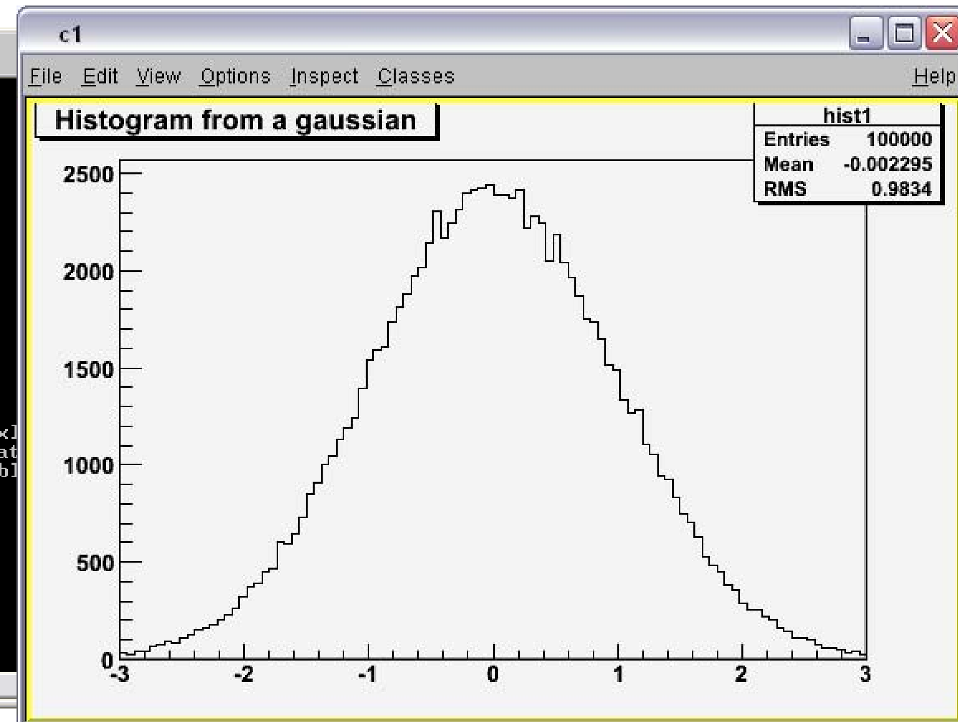
← riempie con 10^5 valori da "gaus"

ROOT session

```
  ******************************************
  *                                        *
  *        W E L C O M E  to  R O O T      *
  *                                        *
  *   Version   5.08/00  13 December 2005  *
  *                                        *
  *  You are welcome to visit our Web site *
  *          http://root.cern.ch           *
  *                                        *
  ******************************************

Compiled on 15 December 2005 for win32.

CINT/ROOT C/C++ Interpreter version 5.16.5, November 30 2005
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] TH1F h1(
TH1F TH1F()
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, Double_t x]
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, const Float
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, const Doubl
TH1F TH1F(const TVectorF& v)
TH1F TH1F(const TH1F& h1f)
root [0] TH1F h1("hist1", "Histogram from a gaussian", 100, -3., 3.)
root [1] h1.FillRandom(
void FillRandom(const char* fname, Int_t ntimes = 5000)
void FillRandom(TH1* h, Int_t ntimes = 5000)
root [1] h1.FillRandom("gaus", 100000)
root [2] h1.Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [3]
```

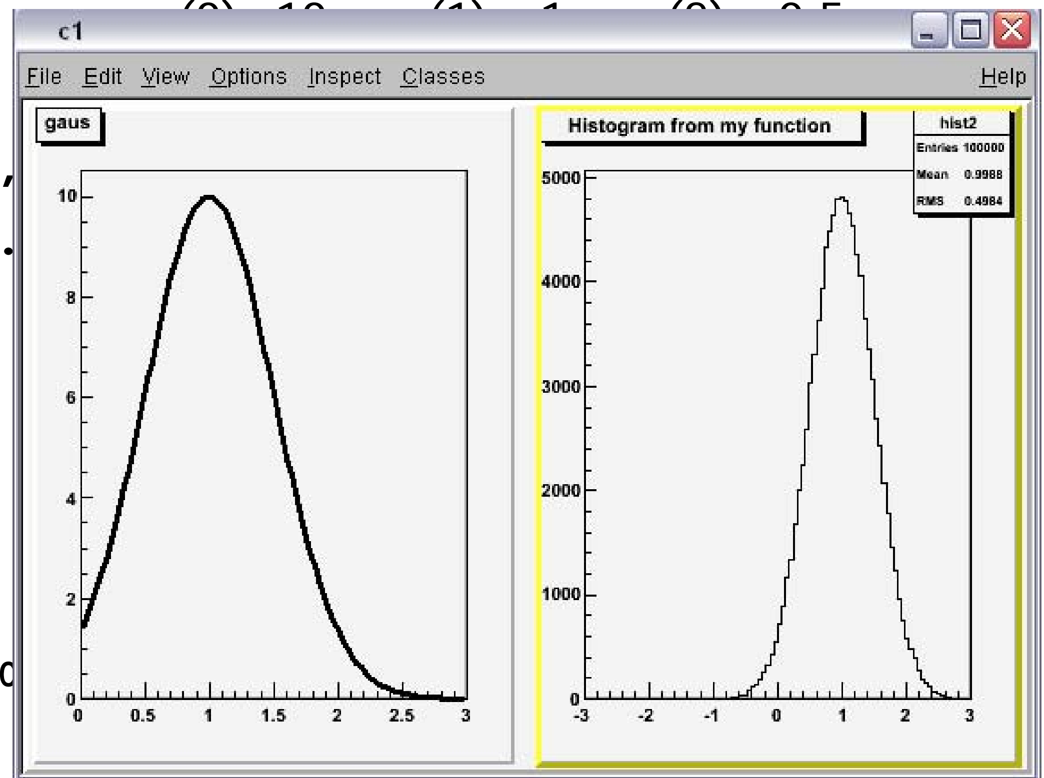


Esempio

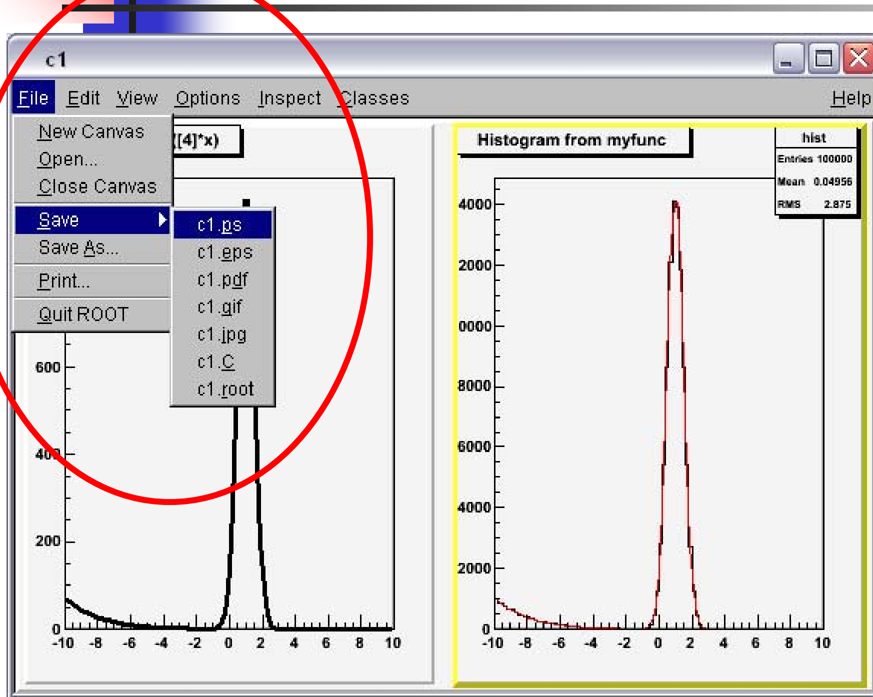
Riempire un istogramma estraendo 100000 numeri random dalla funzione:

$$par(0)e^{-0.5\left(\frac{x-par(1)}{par(2)}\right)^2}$$

```
[ ] TF1 myfunc("myfunc","gaus",
[ ] myfunc.SetParameters(10.,1.
[ ] TCanvas c;
[ ] c.Divide(2,1);
[ ] c.cd(1);
[ ] myfunc.Draw();
[ ] TH1F h2("hist","Histo from
[ ] h2.FillRandom("myfunc",1000
[ ] c.cd(2);
[ ] h2.Draw();
```

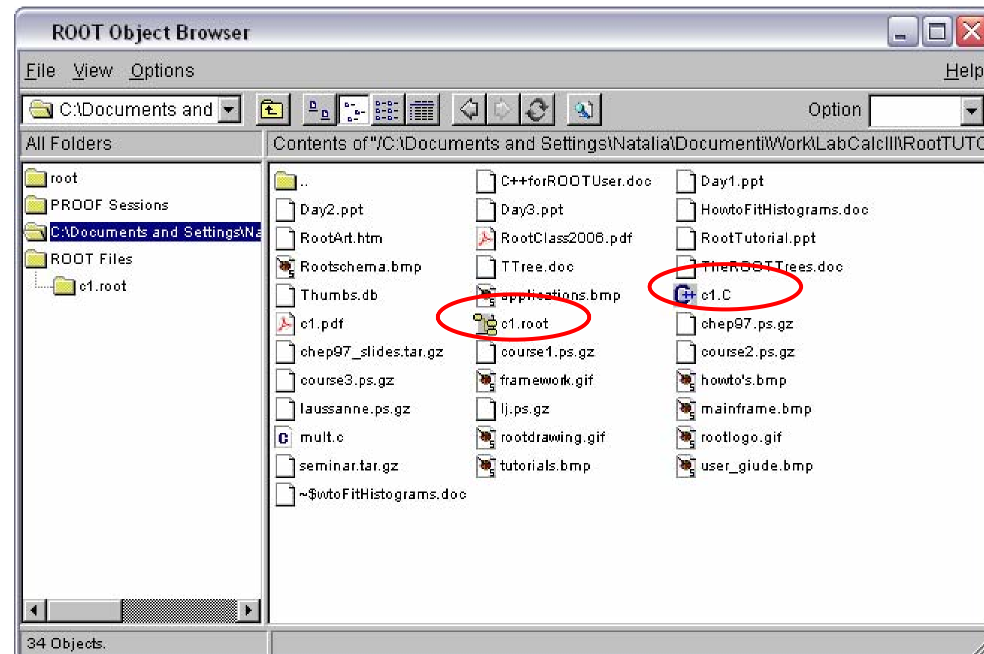


Salvare il proprio lavoro con TBrowser



1) **c1.C**
Per riutilizzare il plot:
root[0]> .x c1.C

2) **c1.pdf**



3) **c1.root**
Per riutilizzare il plot:
root[0]> TBrowser tb

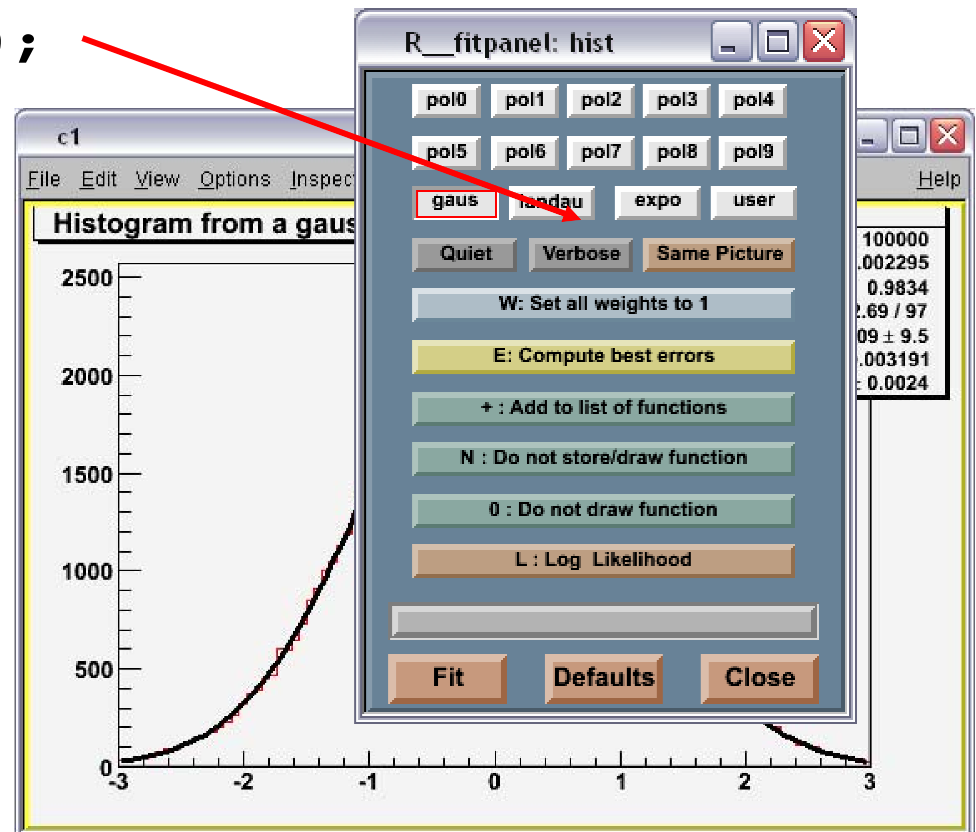
Fit di un istogramma - 1

```
[ ] TH1F h1("hist,"Histogram from a gaussian",100,-3,3);
[ ] h1.FillRandom("gaus",100000);
[ ] gStyle->SetOptFit(111);
[ ] h1.Fit("gaus");
```

oppure

```
[ ] h1.FitPanel();
```

Il FitPanel consente di scegliere parametri e funzioni in modo interattivo

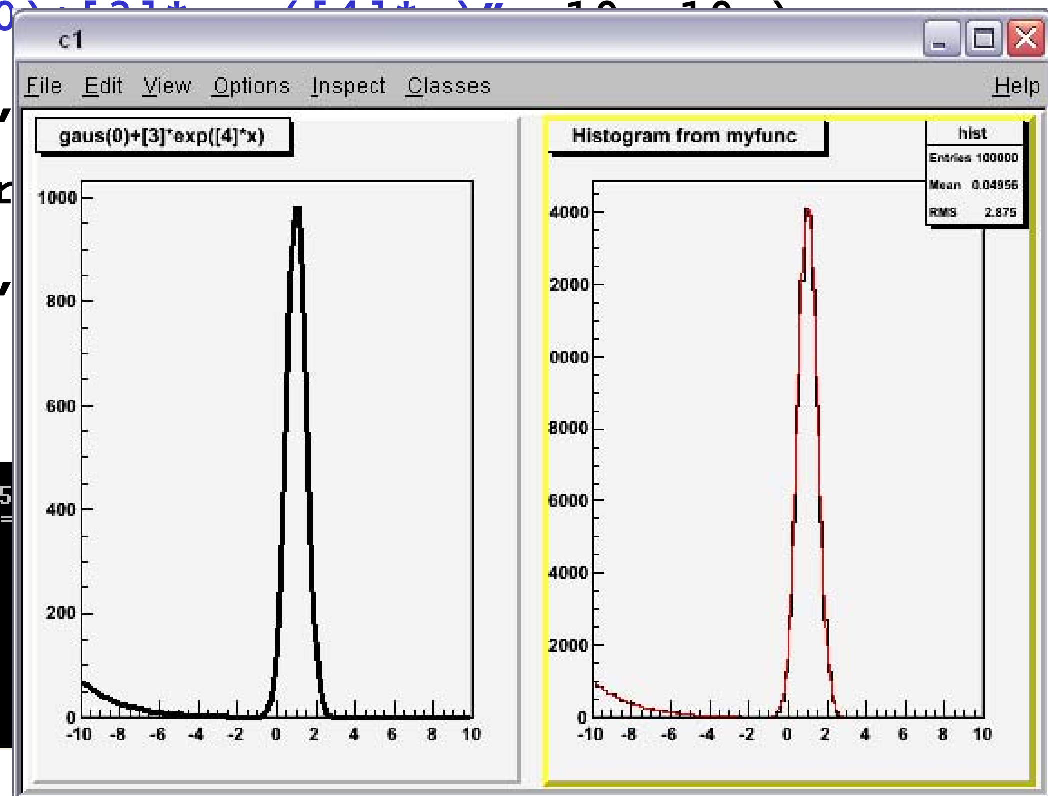


Fit di un istogramma - 2

Fittare un istogramma con la funzione: $par(0)e^{-\left(\frac{x-par(1)}{par(2)}\right)^2} + par(3)e^{par(4)x}$

```
[ ] TF1 f1("myfunc","gaus(0
[ ] f1.SetParameters(1000.,
[ ] TH1F h1("hist","Histogr
[ ] h1.FillRandom("myfunc",
[ ] h1.Fit("myfunc");
```

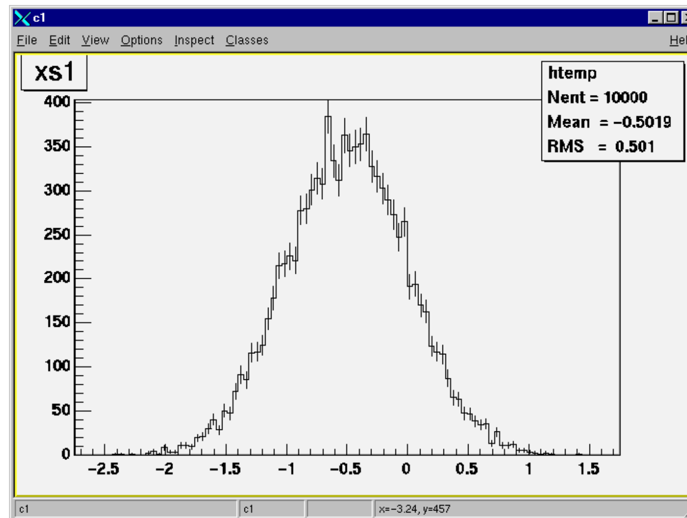
```
root [10] h1.Fit("myfunc")
 FCN=58.0027 FROM MIGRAD    STATUS=CONVERGED      5
                   EDM=1.40487e-008    STRATEGY=
  EXT PARAMETER
  NO.   NAME      VALUE            ERROR
   1  p0          1.43217e+004   5.83272e+001
   2  p1          9.98341e-001   1.66154e-003
   3  p2          4.98132e-001   1.17738e-003
   4  p3          7.32169e+000   3.23833e-001
   5  p4         -4.96196e-001   5.34834e-003
(Int_t)0
root [11]
```

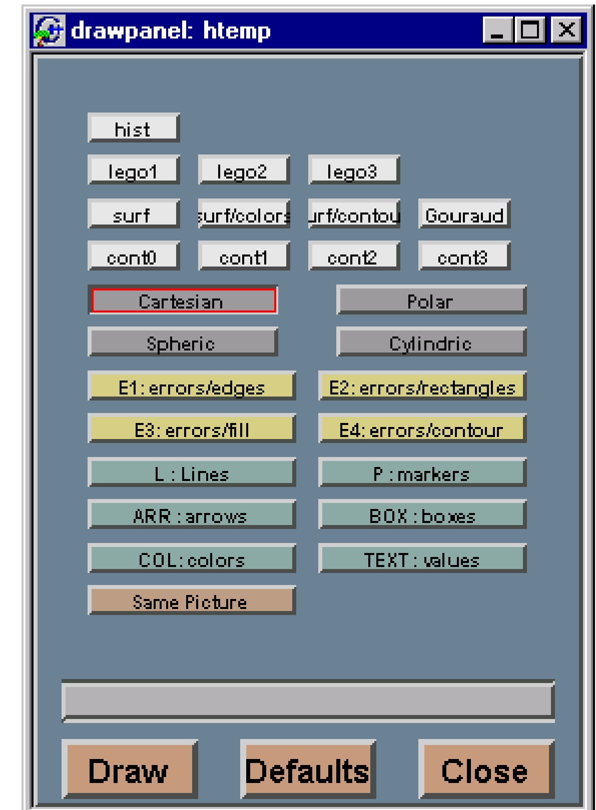


Il DrawPanel()

- Pannello interattivo
  - **`h1->DrawPanel();`**

Consente di cambiare i parametri/tipo di istogramma, le coordinate, gli errori, i colori, etc.





I comandi e gli script

Tre tipi di comandi

1. Comandi CINT, iniziano con "."

   `root[0].?`

   lista di tutti i comandi CINT

   `root[1].X [filename]`

   carica [filename] ed esegue la funzione [filename] (stesso nome del file)

   `root[2].L [filename]`

   carica [filename]

2. Comandi di SHELL, iniziano con ".!", e.g.

   `root[3] .! ls`

Tre tipi di comandi

3. Sintassi C++ (o quasi)

```
root [0] TBrowser *b = new TBrowser()
```

oppure

```
root [0] TBrowser *b = new TBrowser();
```

Il “;” è **opzionale**:

Se non c’è, ROOT mostra il valore di ritorno del comando:

```
root [0] 23+5   // mostra il valore di ritorno
(int)28
root [1] 23+5;  // nessun valore di ritorno
root [2]
```

Convenzioni su coding e nomi

Basato su Taligent

Classi	Iniziano con T	TTree, TBrowser
Tipi diversi dalle classi	Finiscono con _t	Int_t
Dati membri delle classi	Iniziano con f	fTree
Funzioni membro delle classi	Iniziano con maiuscola	Loop()
Costanti	Iniziano con k	kInitialSize, kRed
Variabili statiche	Iniziano con g	gEnv
Dati membri statici delle classi	Iniaziano con fg	fgTokenClient

Gli script – script senza nome

- Adatti a fare piccoli task
  - Iniziano con "{" e finiscono con "}"
  - Tutte le variabili in global scope
  - Nessuna definizione di classi o funzioni
  - Nessun parametro

Script **senza nome**: hello.C

```
{
  cout << "Hello" << endl;
}
```

Gli script – script con nome

- Adatti a task più complessi, ma che ancora non richiedono un eseguibile ad-hoc (è pur sempre una macro!)
  - Funzioni del C++
  - Regole di scopo secondo il C++ standard
  - La funzione che ha lo **stesso nome del file** può essere eseguita con .x

    ```
    root [3] .x myMacro.C
    ```
  - Supporta parametri e definizione di classi

Script con nome: say.C

```
void say(TString what="Hello")
{
  cout << what << endl;
}
```

```
root [3] .x say.C
Hello
root [4] .x say.C("Hi")
Hi
```

ACLiC: Automatic Compiler of Libraries for CINT

- Gli script con nome possono essere **interpretati** riga per riga (interprete CINT)

```
root [3] .x myMacro.C;
```

- **Compilati** per produrre una shared library via ACLiC (e poi eventualmente eseguiti)

```
root [3] .L myMacro.C++;  //ricompila sempre
root [3] .L myMacro.C+;   //ricompila solo se necessario
root [3] .x myMacro.C++;  //ricompila ed esegue
root [3] .L myMacro_C.so;  //ricarica la shared library
root [3] myMacro(); //esegue la funzione
root [3] .U myMacro_C.so;  //unload della libreria
```

I vantaggi dell'ACLiC

- Vantaggi
  - controllo **preventivo** della sintassi
    - l'errore viene individuato prima di girare, non dopo aver eseguito tutte le righe precedenti!
  - circa **5 volte più veloce** della macro interpretata
  - supporta per intero le caratteristiche del C++
- Svantaggi
  - Con il compilatore KCC, si può caricare soltanto una shared library per volta
    - non si può usare il comando .U per fare l'unload di una shared library
- Consiglio: compilare sempre

I TFile

Aprire un file

- Apertura di un file per leggerlo:

```
root[] TFile f("Example.root")
```

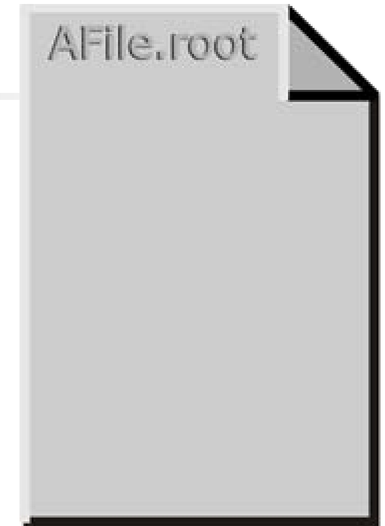
- Controllo del contenuto di un file

```
root[] f.ls()
TFile**          Example.root    ROOT file
 TFile*          Example.root    ROOT file
  KEY: TTree     myTree;1        Example ROOT tree
  KEY: TH1F      totalHistogram;1        Total Distribution
  KEY: TH1F      mainHistogram;1 Main Contributor
  KEY: TH1F      s1Histogram;1   First Signal
  KEY: TH1F      s2Histogram;1   Second Signal
```

- Recupero di un contenuto tramite il nome

```
root[] totalHistogram->Draw();
root[] TH1F* myHisto = (TH1F*)
  f.Get(“totalHistogram”);
```

I files di ROOT (TFile)



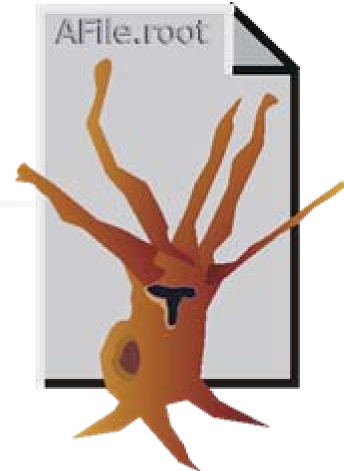
- Quando un file di ROOT viene aperto, questo diventa la directory corrente
  - manualmente: **`file.cd();`**
  - se non ci sono files aperti, la directory corrente è

  la memoria
- Gli istrogrammi e i TTree creati sono **salvati automaticamente** in quel file
- Quando il file viene chiuso, gli oggetti istogramma e tree associati con quel file vengono rimossi dalla memoria
- Ogni oggetto derivato da **`TObject`** può essere scritto su in file ROOT
  - Va aggiunto **esplicitamente**

```
myObject->Write("name");
```

I TTree

Trees di ROOT (TTree)



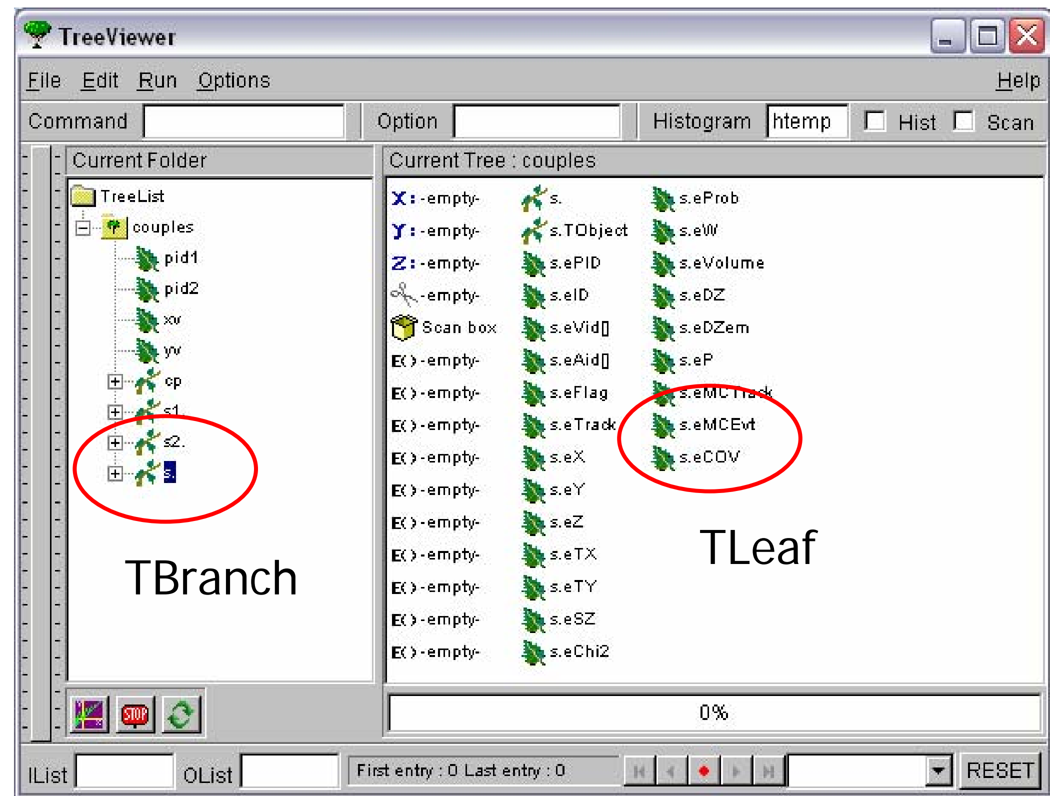
- E' l'implementazione ROOT di una **ntupla**
  - tabella di valori/oggetti correlati
    - es. energia, tempo, rise time di uno stesso evento
  - gli oggetti nel tree non sono necessariamente numeri
    - possono essere vettori o **qualsiasi oggetto** di **ROOT**
    - i vettori possono avere lunghezza variabile (la lunghezza è un altro campo dello stesso TTree)
- consente di salvare un grandissimo numero di entries (soluzione di storage!)
- ha una struttura **gerarchica** di rami (TBranch) e foglie (TLeaf)
  - è possibile leggere selettivamente solo una parte dei rami (anziché tutto il tree) → guadagno di tempo

Contenuto di un TTree

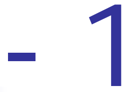
- Un TTree si può recuperare da un TFile esattamente come un istogramma (nome o funzione **Get**)

```
[] TTree* myTree =
  (TTree*) f.Get("name");
[] myTree->StartViewer();
```

Il viewer consente l'accesso interattivo al TTree e alle sue TLeaf



Analisi interattiva di un TTree - 1



Lista variabili (leafs e branches):

```
[ ]> tree->Print()
```

Plot 1D di una variabile

```
[ ]> tree->Draw(“varname”)
```

Scatter plot di due variabili

```
[ ]> tree->Draw(“varname1:varname2”)
```

Con un'opzione grafica (lego2)

```
[ ]> tree->Draw(“varname1:varname2”, “”, “lego2”)
```

Con un taglio su una terza variabile

```
[ ]> tree->Draw(“varname1:varname2”, “varname>3”, “lego”)
```

Scatter plot di tre variabili

```
[ ]> tree->Draw(“varname1:varname2:varname3”)
```

Analisi interattiva di un TTree - 2

Specifica di un evento (tutte le foglie)

```
[ ]> tree->Show(eventNumber);
```

Fit della distribuzione di una variabile

```
[ ]> tree->Fit("func", "varname")
```

Fit di una variabile con selezione

```
[ ]> tree->Fit("func", "varname", "varname > 10")
```

Uso della classe TCut per definire tagli

```
[ ]> TCut cut1="varname1>0.3"
[ ]> tree->Draw("varname1:varname2",cut1)
[ ]> TCut cut2="varname2<0.3*varname1+89"
[ ]> tree->Draw("varname1:varname2",cut1 && cut2)
```

Come costruire un TTree

- Un po' laborioso, richiede 5 passaggi:
  1. Creare un TFile
  2. Creare un TTree
  3. Aggiungere TBranch al TTree
  4. Riempire il TTree
  5. Scrivere il file
- Vediamoli più in dettaglio

Costruiamo un Tree - 1

- Step 1: Creare un oggetto TFile

```
TFile *myfile = new TFile("test.root","RECREATE");
```

Il costruttore TFile ha come argomenti:

- ✓ file name (i.e. "test.root ")
- ✓ opzioni: NEW, CREATE, RECREATE, UPDATE, o READ

- Step 2: Creare un oggetto TTree

```
TTree *tree = new TTree("myTree","A ROOT tree");
```

Il costruttore TTree ha come argomenti:

- ✓ Tree Name (e.g. "myTree")
- ✓ Titolo del TTree (possibilmente descrittivo!)

Costruiamo un Tree - 2

- Step 3: Aggiungere i Branches

```
Event *event = new  Event();
myTree->Branch("EventBranch","Event",&event);
```

- ✓ Nome del Branch
- ✓ Nome della Classe
- ✓ Indirizzo (puntatore) dell'oggetto (nel nostro caso, Event)
  - ✓ La classe Event può contenere diversi parametri (ad esempio, Ntrack, Flag)
  - ✓ Ciascuno di essi diventa una TLeaf

Costruiamo un Tree - 3

- Step 3b: Aggiungere i Branches

La stessa cosa si può fare dichiarando il TBranch direttamente come una **lista di variabili** (TLeaf)

```
Event *event = new  Event();
myTree->Branch
("Ev_Branch",&event,"ntrack/I:nseg:nvtex:flag/I:temp/F");
```

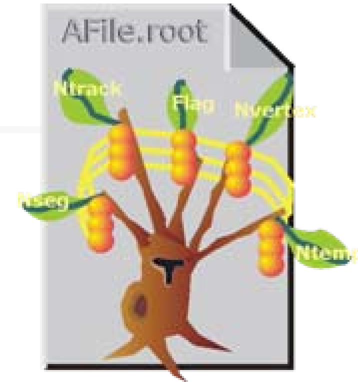
lista delle variabili con il loro tipo

Per avere **TBranch = TLeaf** (→ branches con una sola variabile)

```
Int_t ntrack;
myTree->Branch
("NTrack",&ntrack,"ntrack/I");
```

indirizzo di memoria da cui leggere il valore

Costruiamo un Tree - 4



- Step 4: Riempire il TTree

✓ Creare un loop "for"

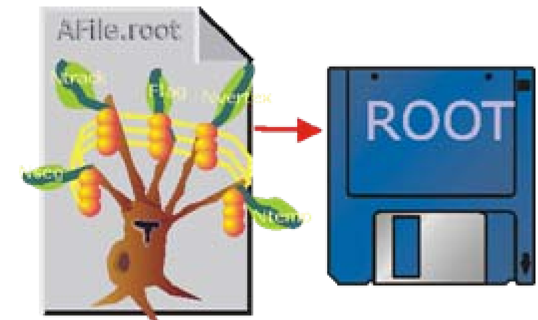
✓ Assegnare i valori corretti alle variabili/oggetti i cui indirizzi sono stati dichiarati nei branch

```
myTree->Fill();
```

- Step 5: Salvare il TTree su TFile

Il metodo Write() di TFile **scrive automaticamente** tutti i TTree e tutti gli istogrammi correnti.

```
myFile->Write();
```

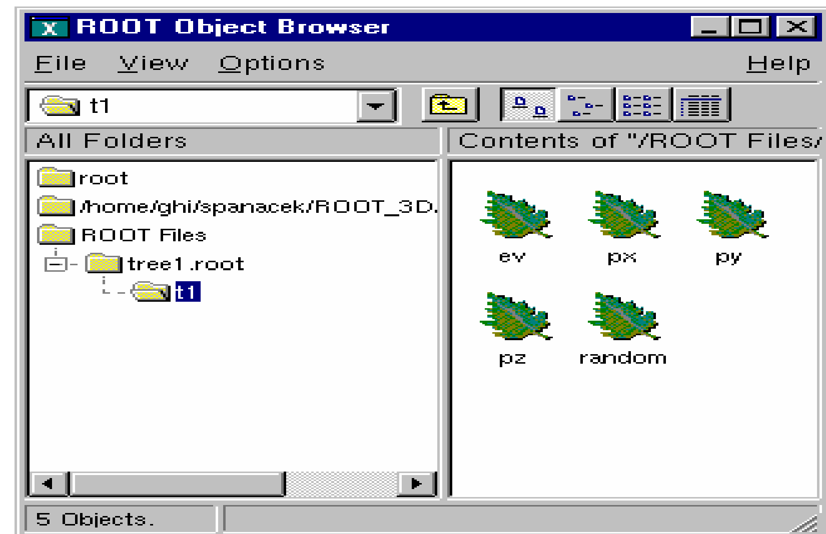


Come rileggere un TTree - 1

- Già visto come si apre e legge un TTree in modalità interattiva
- E come si recuperano le informazioni delle singole TLeaf da una macro o da un programma C++?
  - necessario per fare analisi complessa o di una grande mole di dati
- C'è un tutorial sotto

`$ROOTSYS/tutorials/tree1.C`

Come rileggere un TTree - 2



1. Aprire il TFile

```
TFile f("tree1.root")
```

2. Recuperare il tree (via il nome)

```
TTree * t1 =
   (TTree*)f.FindObject("t1")
```

Il tree dell'esempio ha 5 campi: ev, px, py, pz e random

Come rileggere un TTree - 3

3. Creare le variabili appropriate per contenere i dati

```
Float_t px, py;
```

4. Associare i branch che si vogliono rileggere con le variabili (o meglio i loro puntatori) via `SetBranchAddress("name", address)`
   - non è necessario rileggere tutti i branches

```
t1->SetBranchAddress("px", &px)
t1->SetBranchAddress("py", &py)
```

5. Leggere un'entry nel TTree con `GetEntry(nr)`

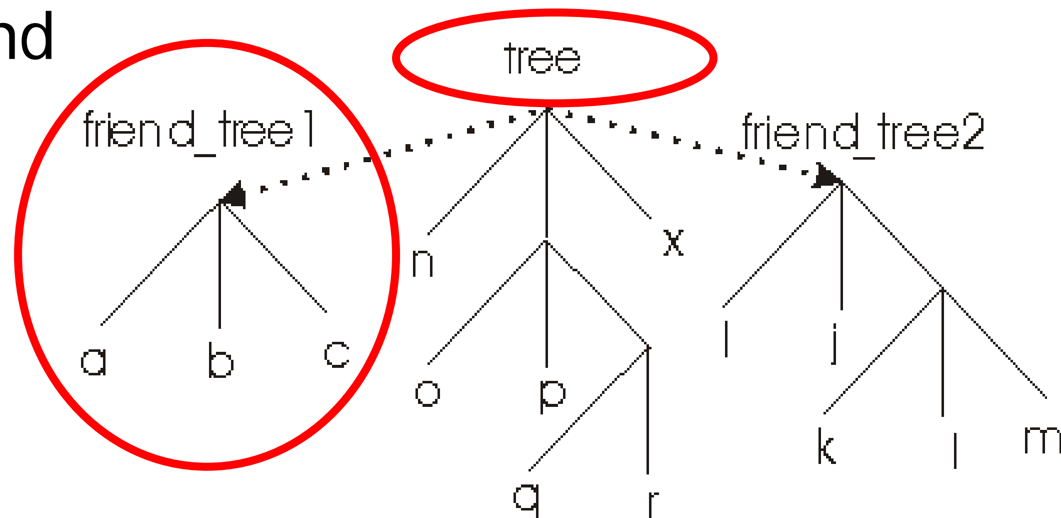
```
t1->GetEntry(0) // first entry
```

   - alle variabili mappate viene assegnato il valore effettivo
   - si può loopare su tutte le entry, da 0 a `t1->GetEntries()`

Friends di TTree

- In alcuni casi è utile/necessario aggiungere branch a dei TTree esistenti (e.g. risultati aggiuntivi dell'analisi)
  - Spesso il tree originale è readonly
  - Rischio di danneggiare il TTree originario
- Soluzione: aggiungere un TTree friend
- Ogni TTree ha accesso senza restrizioni a tutti i dati/campi dei suoi friend

Di fatto, equivalente ad un TTree **unico** che comprende tree, friend_tree1 e friend_tree2



Aggiungere friends ad un TTree

- `AddFriend("treeName", "fileName")`

```
mytree->AddFriend("ft1","ff.root")
```

  - se il nome del file non c'è, si assume che il friend sia nello stesso TFile del tree di partenza
- Se i TTree hanno lo stesso nome, è necessario dare al friend un "alias", in modo che i nomi siano **univoci**

```
tree.AddFriend("tree1 = tree","ff.root")
```

Accesso ai friends

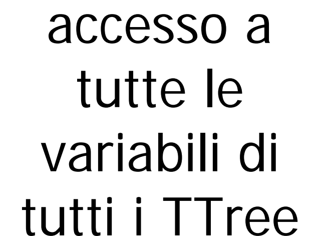
- **Accesso**:

  `treeName.branchName.leafName`

  basta anche solo la leafName, se è sufficiente a individuarla univocamente

- Esempio:

```
mytree->Draw("t2.px")
mytree->Draw("t2.pz","t1.px>0")
mytree->SetBranchAddress("t2.px")
```

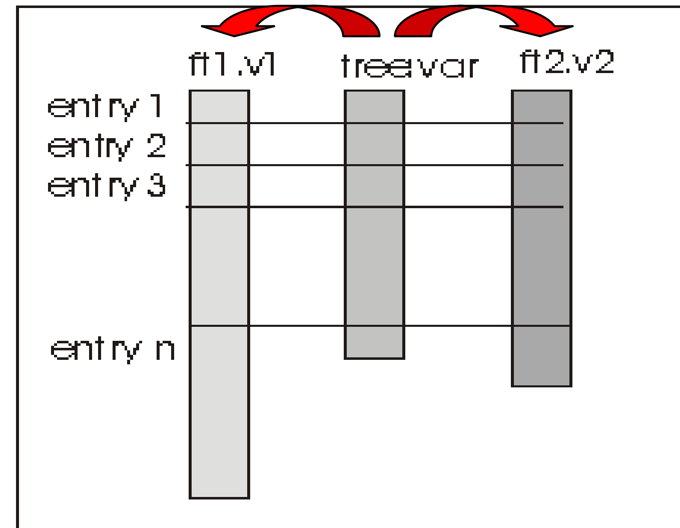


- Lista di tutte le branches:

```
mytree->Print("all")
```

La friend list

Il numero di entries del friend deve essere maggiore o uguale di quello del tree “principale”



`ft1` può essere friend di `tree`, ma `tree` non può essere friend di `ft1` → il “main tree” deve essere `tree`

Per accedere alla lista dei friends:

`TTree::GetListOfFriends()`

Usare ROOT in altri programmi C++

Usare ROOT in altri programmi - 1

- I TTree/TH1 di ROOT sono tipicamente generati dall'analisi dati o simulazioni (e.g. un programma di Geant4)
- Per scrivere dei files di ROOT nella vita reale spesso è necessario utilizzare le librerie di ROOT all'interno di programmi esterni C++
- Per far funzionare la cosa è necessario che il Makefile/linea di comando del compilatore contenga:
  - Per la compilazione: l'area dove sono contenuti gli header (.h) delle classi di ROOT
  - Per il linking: l'area dove sono contenute le librerie compilate (.so) di ROOT e i nomi delle librerie da linkare

Usare ROOT in altri programmi - 2

- Esiste un comando di ROOT che restituisce librerie e includes "pronte" per il compilatore
  - **root-config --cflags**

    Sul mio sistema risponde:

    ```
    -pthread -m64 -I/usr/local/root/include
    ```

  - **root-config --libs**

    Sul mio sistema:

    ```
    -L/usr/local/root/lib -lCore -lCint -lRIO -lNet -
      lHist -lGraf -lGraf3d -lGpad -lTree -lRint -
      lPostscript -lMatrix -lPhysics -lMathCore -lThread
      -pthread -lm -ldl -rdynamic
    ```

includes

Directory libs

Nomi libs

Esempio: GNUmakefile di Geant4

- Per utilizzare ROOT in applicazioni Geant4 è sufficiente aggiungere allo **GNUmakefile** dell'applicazione Geant4

```
CPPFLAGS += `root-config --cflags`
LDFLAGS  += `root-config --libs`
```

- Le **`CPPFLAGS`** sono le **opzioni** date al compilatore in fase di compilazione, le **`LDFLAGS`** in fase di linking
  - In altri Makefile/sistemi, i nomi possono essere diversi

Definire classi di ROOT-utente

Ancora un passo avanti: “rootificare” le proprie classi

- E’ possibile fare in modo che le **proprie classi** vengano viste come classi di ROOT (e.g. richiamabili da linea di comando, scrivibili su files ROOT o come campi di un TTree, etc.).
- Ad esempio, si possono creare nuovi “contenitori” e nuovi tipi di oggetti (es. **`MyTRun`**)
- Tre modi possibili:
  - Dall’interprete

    No I/O (non tanto utile!)
  - Come shared libraries (librerie compilate)
  - Con l’ACLiC

    Piena funzionalità

Definire la propria classe in ROOT

- Step 1: la classe deve ereditare da **`TObject`** (o anche dalla sua figlia **`TNamed`**)
  - Eredita tutte le caratteristiche degli oggetti ROOT, inclusa una stringa per il nome e vari metodi per I/O e gestione
- Step 2: aggiungere al codice le linee

```
ClassDef(ClassName,ClassVersionID)
```

Alla fine della definizione (.h)

```
ClassImp(ClassName)
```

All'inizio dell'implementazione (.c)

ClassDef e ClassImp

- **`ClassDef()`** e **`ClassImp()`** sono **macro** definite da ROOT
- Sono necessarie per gestire l'I/O degli oggetti. Hanno l'effetto di produrre:
  - Metodi di streamer per scrivere gli oggetti in **files di ROOT** o come **campi di un TTree**.
  - ShowMembers()
  - Operatore >> è in overload

Un esempio concreto

```
class MyTRun : public TNamed, public MyRun
{
public:
   MyTRun() {;};
   virtual ~MyTRun(){;};

   ClassDef(MyTRun, 1) // Run class
};
```

.h

Doppia ereditarietà

```
#include "MyTRun.hh"

ClassImp(MyTRun);
```

.C

- Step 3: creare un file **LinkDef.h**. Serve a notificare a ROOT l'esistenza di una nuova classe-utente da inserire nel dizionario

```
#ifdef __CINT__
#pragma link off all globals;
#pragma link off all classes;
#pragma link off all functions;
#pragma link C++ class MyTRun;
#endif
```

Ma ancora non basta...

- Step 4 (e ultimo): scrivere un Makefile e chiamare il comando **rootcint** per aggiungere la classe al dizionario

```
$(ROOTSYS)/bin/rootcint -f
   MyDictionary.cxx -c MyTRun.h
   LinkDef.h
```

- **LinkDef.h** deve essere l'ultimo argomento della linea di comando **rootcint**
- Il nome del file LinkDef deve contenere la stringa LinkDef.h o linkdef.h:
  - **MyNice_LinkDef.h** va bene