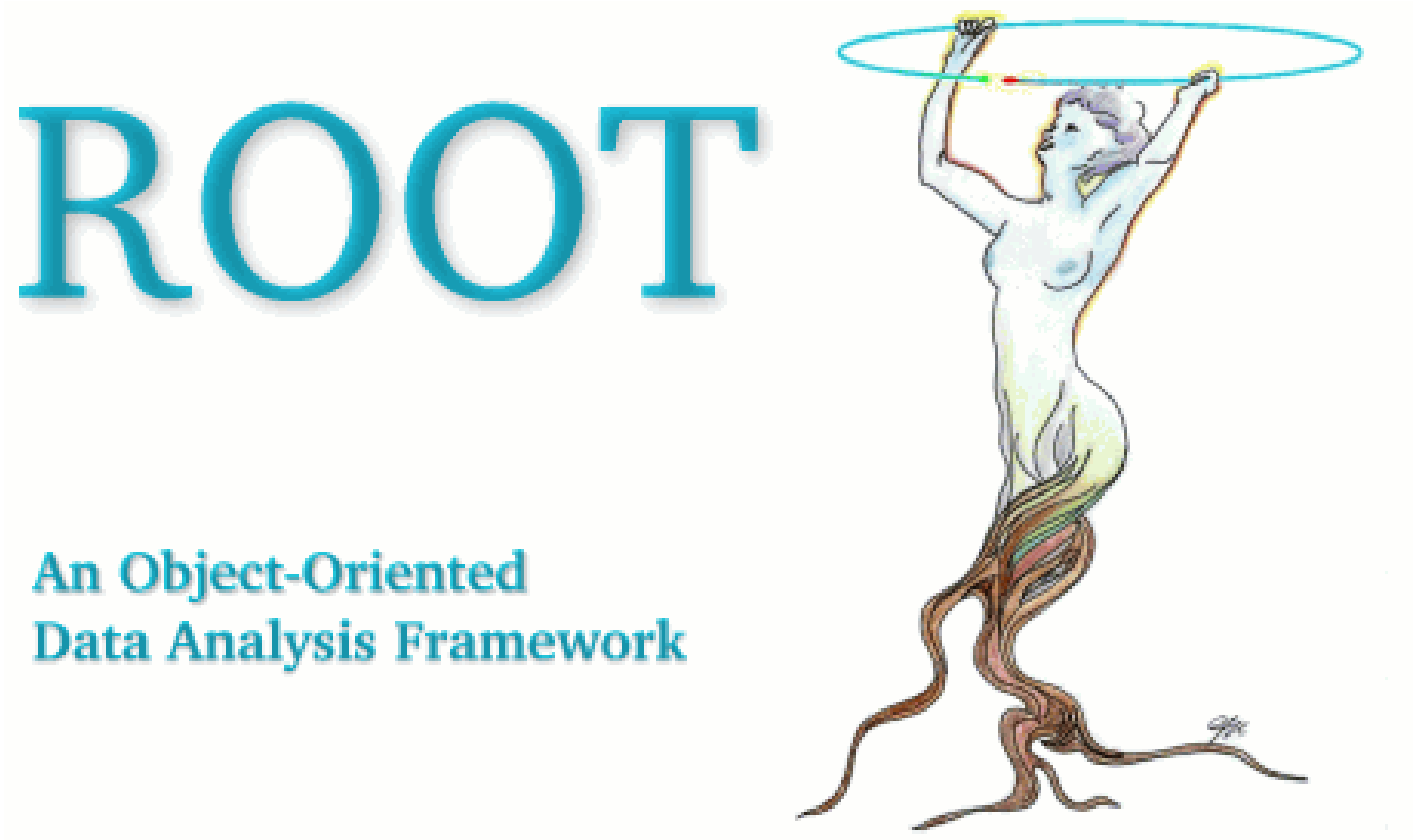
Introduzione a ROOT

Analisi dati con ROOT

- Pacchetto di analisi dati scritto in **C++** (*object-oriented*)
  - È nato al CERN!

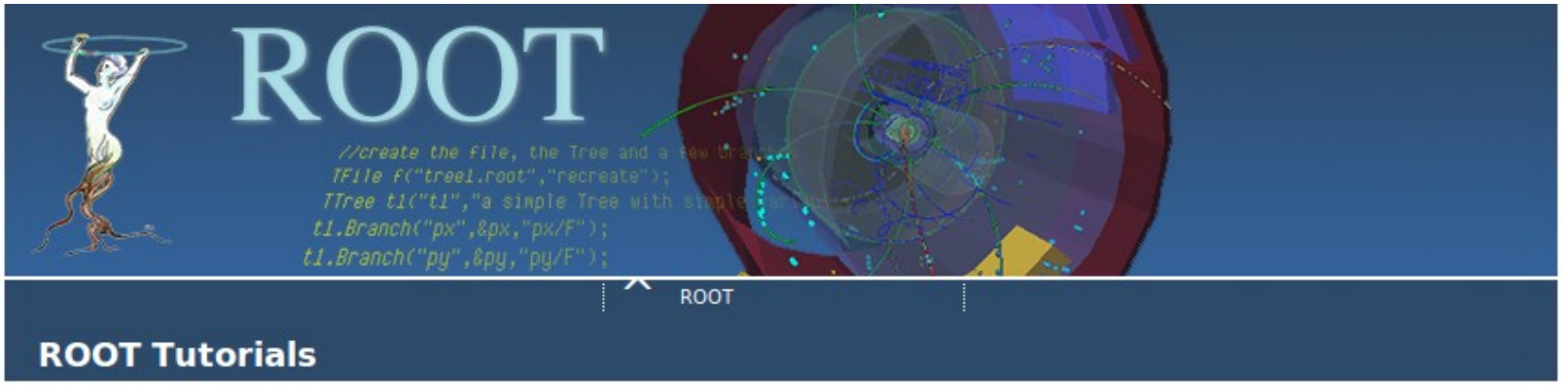


Dove lo trovo?

- Si trova tutto su web: sia **l'ultima versione del programma** (ad oggi la 5.32) che la **utilissima documentazione**:
  - ***Home page***: http://root.cern.ch/drupal
  - ***User's guide***: http://root.cern.ch/drupal/content/users-guide
  - ***Tutorials***: http://root.cern.ch/root/html/tutorials/

I *tutorial* di ROOT

- Per imparare a fare qualcosa in ROOT:
  - Si cerca il *tutorial* con un esempio già fatto
  - Si copia ed incolla la parte interessante nel proprio programma



From $ROOTSYS/tutorials/

hist	Histograms
graphics	Basic Graphics
graphs	TGraph, TGraphErrors, etc
gui	Graphics User Interface
fit	Fitting tutorials
io	Input/Output
tree	Trees I/O, Queries, Graphics
math	Math tutorials
matrix	Matrix packages tutorials
geom	Geometry package
gl	OpenGL examples
eve	Event Display
fft	Fast Fourier Transforms
foam	TFoam example
image	Image Processing
mlp	Neural Networks
net	Network, Client/server

Come installare ROOT

- Diversi modi di installazione **su Linux**:
  - Scaricate i **pacchetti precompilati** (*binaries*) compatibili con la vostra distribuzione e *unzip*-ate il contenuto del file → ROOT è (quasi) pronto per essere eseguito
  - Scaricate il **file sorgente** del programma (*source*), *unzip*-ate e compilate ROOT seguendo le istruzioni contenute in README/INSTALL o qui
- Dopo aver installato, aggiungete le seguenti **variabili di sistema** (necessarie per eseguire ROOT e per *linkarne* le librerie!) al file di configurazione della *shell* (`~/.bashrc` oppure `~/.(t)cshrc`)

(ba)sh :

```
. bin/thisroot.sh
```

ESEGUE:

```
export ROOTSYS=<percorso-base-di-root>
export PATH=${PATH}:${ROOTSYS}/bin
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${ROOTSYS}/lib
```

(t)csh :

```
source bin/thisroot.csh
```

ESEGUE:

```
setenv ROOTSYS <percorso-base-di-root>
setenv PATH ${PATH}:${ROOTSYS}/bin
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${ROOTSYS}/lib
```

Compilare un programma con ROOT

```
c++ -o testROOT `root-config --cflags --glibs` testROOT.cpp
```

- **`c++`**: il comando di compilazione è sempre lo stesso
- **`-o testROOT`**: nome del file eseguibile
- **`` `root-config --cflags –glibs` ``**: comando per *linkare* le librerie di ROOT facilmente
- **`testROOT.cpp`**: nome del file da compilare

Le classi di ROOT

- Per vedere quali sono gli **oggetti esistenti** e quali siano i loro **metodi** → cercate i prototipi *online* → Google Italia

nome della classe **nome della classe(i) da cui eredita**

```
class TH1F: public TH1, public TArrayF
```

```
TH1F methods
TH1F : histograms with one float per channel.  Maximum precision 7 digits
```

Function Members (Methods)

public:

costruttori

```
TH1F ()
TH1F (const TVectorF& v)
TH1F (const TH1F& h1f)
TH1F (const char* name, const char* title, Int_t nbinsx, const Float_t* xbins)
TH1F (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins)
TH1F (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)
```

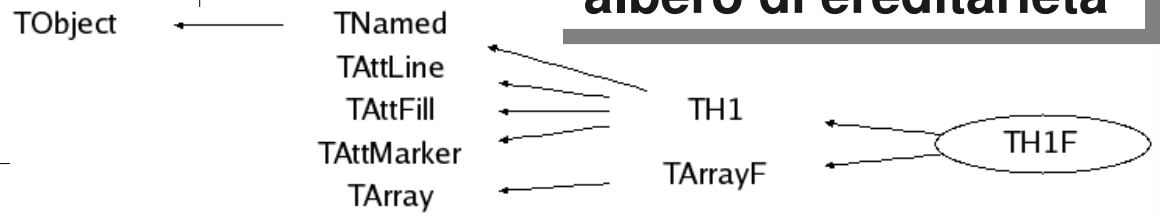
metodi

```
virtual ~TH1F ()
virtual void AddBinContent (Int_t bin)
virtual void AddBinContent (Int_t bin, Double_t w)
static TClass* Class ()
virtual void Copy (TObject& hnew) const
virtual TH1* DrawCopy (Option_t* option = "") const
virtual Double_t GetBinContent (Int_t bin) const
virtual Double_t GetBinContent (Int_t bin, Int_t) const
virtual Double_t GetBinContent (Int_t bin, Int_t, Int_t) const
virtual TClass* IsA () const
TH1F& operator= (const TH1F& h1)
virtual void Reset (Option_t* option = "")
virtual void SetBinContent (Int_t bin, Double_t content)
virtual void SetBinContent (Int_t bin, Int_t, Double_t content)
virtual void SetBinContent (Int_t bin, Int_t, Int_t, Double_t content)
virtual void SetBinsLength (Int_t n = -1)
virtual void ShowMembers (TMemberInspector& insp, char* parent)
virtual void Streamer (TBuffer& b)
void StreamerNVirtual (TBuffer& b)
```

protected:

- Cliccate sui diversi costruttori / metodi per avere una breve descrizione
- Consultate il manuale per maggiori informazioni / esempi!

albero di ereditarietà



Come aprire e chiudere ROOT

- Per entrare in ROOT si digita **`root`**
- Compare la finestra di benvenuto ed un prompt:

```
[andrea@despero:~] root
   ******************************************
   *                                        *
   *        W E L C O M E  to  R O O T      *
   *                                        *
   *    Version   5.25/04  23 November 2009 *
   *                                        *
   *  You are welcome to visit our Web site *
   *          http://root.cern.ch           *
   *                                        *
   ******************************************

ROOT 5.25/04 (trunk@31401, Nov 23 2009, 23:28:12 on linuxx8664gcc)

CINT/ROOT C/C++ Interpreter version 5.17.00, Dec 21, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] 5+7
(const int)12
root [1] std::cout << "output di test" << std::endl;
output di test
root [2] double a = 10;
root [3] double b = 2;
root [4] std::cout << "somma = " << a+b << std::endl;
somma = 12
root [5] .q
[andrea@despero:~]
```

- Si presenta così: **`root [0]`** N.B. tra [ ] è indicato il numero progressivo dell'operazione eseguita
- È **l'interprete** di C++
- Si possono digitare comandi di C++ (tasto invio per eseguirli)
- Per chiudere ROOT si digita **`.q`**

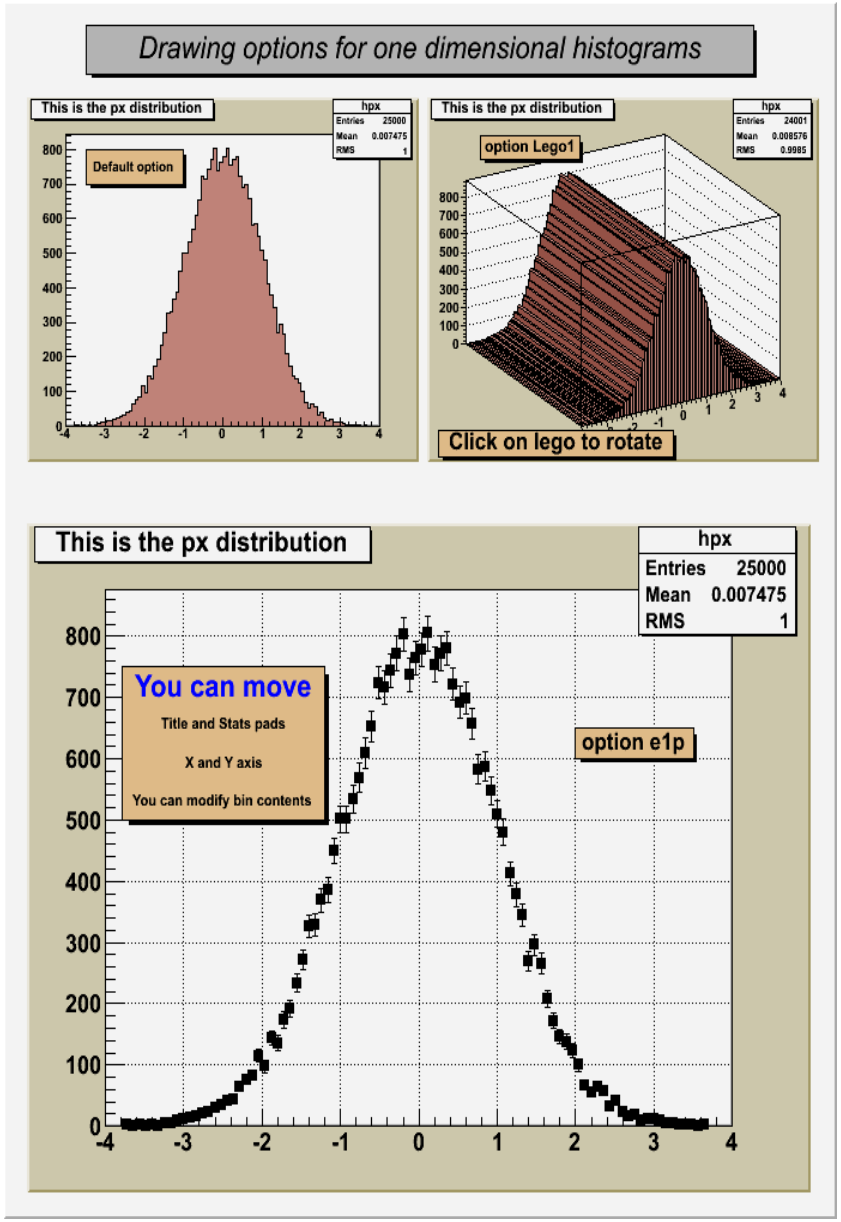
Una *macro* di ROOT

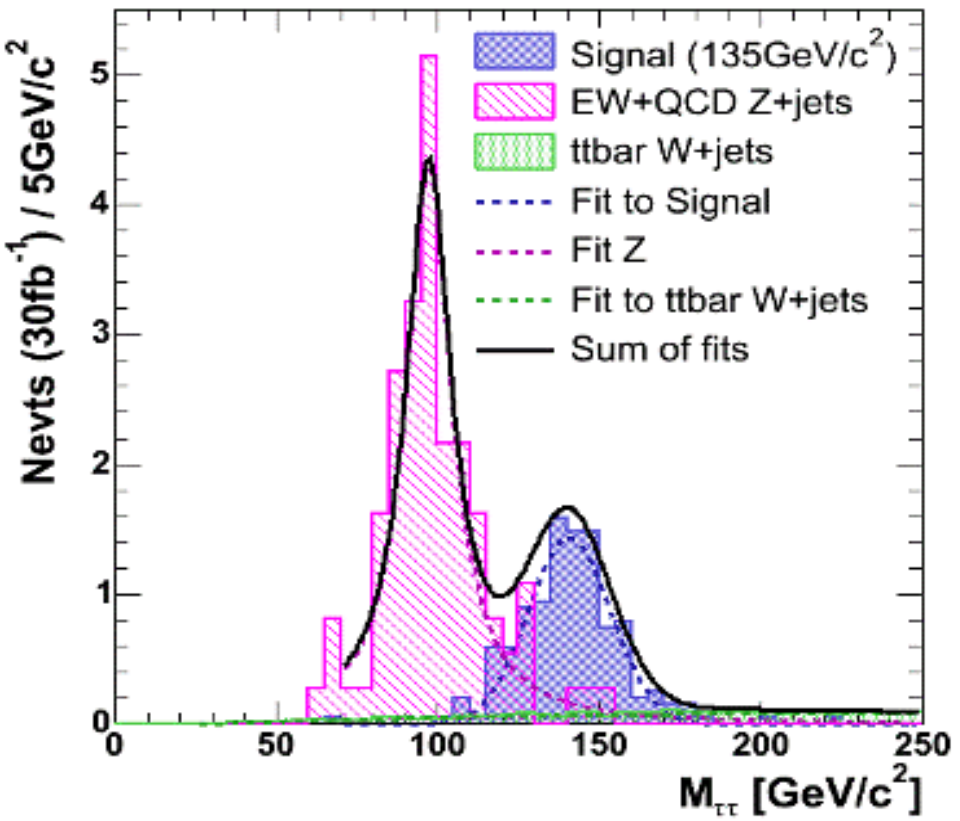
```
int testMacro(int ingresso = 4)
{
  std::cout << "calcolo il doppio di "
            << ingresso
            << std::endl;

  return 2*ingresso;
}


int testMacro2(int ingresso = 4)
{
  std::cout << "calcolo il triplo di "
            << ingresso
            << std::endl;

  return 3*ingresso;
}
```

```
root [0] .x testMacro.C
calcolo il doppio di 4
(int)8
root [1] .x testMacro.C(6)
calcolo il doppio di 6
(int)12
root [2] .L testMacro.C
root [3] testMacro(8)
calcolo il doppio di 8
(int)16
root [4] testMacro2(10)
calcolo il triplo di 10
(int)30
root [5]
```

- Scrivo queste funzioni (in linguaggio C++!) in un file chiamato `testMacro.C`
- Posso eseguirle in due modi differenti:
  - **`.x testMacro.C`** → ROOT esegue solo la funzione trovata nel file che ha lo **stesso nome** del file (senza estensione `.C`)
  - **`.L testMacro.C`** → ROOT carica le *macro* come **libreria**; le funzioni vengono poi chiamate da riga di comando
- Ecco un esempio di entrambi gli utilizzi:

Istogrammi 1D - TH1F





Riempire un istogramma

- Leggo da un file di testo i valori che mi interessano e li uso per riempire un istogramma
- E' un oggetto di tipo **`TH1F`**
- Non dimenticare gli `#include` e le istruzioni per la compilazione

```
TH1F histo ("histo","",100,-3,2) ;
std::ifstream leggo (argv[1], std::ios::in) ;
while (!leggo.eof ())
  {
    double var ;
    leggo >> var ;
    histo.Fill (var) ;
  }
leggo.close () ;
```

devo conoscere l'intervallo nel quale mi aspetto di trovare i dati per definire l'istogramma

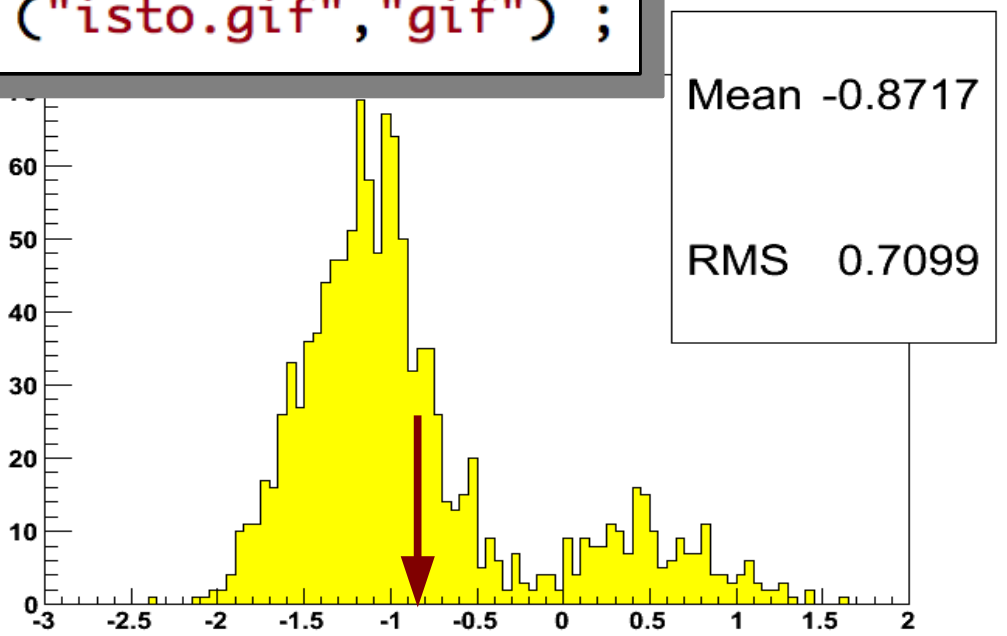
il nome del file e' un argomento di **`int main(int argc, char** argv)`**

ogni variabile letta viene passata al metodo **`Fill`** dell'istogramma

Disegnare l'istogramma

- ROOT disegna l'istogramma con il metodo **`TH1F::Draw`**
- serve una tavolozza dove disegnarlo (**`TCanvas`**)
- il **`TCanvas`** puo' essere stampato su un file grafico

```
TCanvas c1 ;
histo.SetFillColor (5) ;
histo.Draw () ;
c1.Print ("isto.gif","gif") ;
```



l'istogramma sara' riempito di giallo

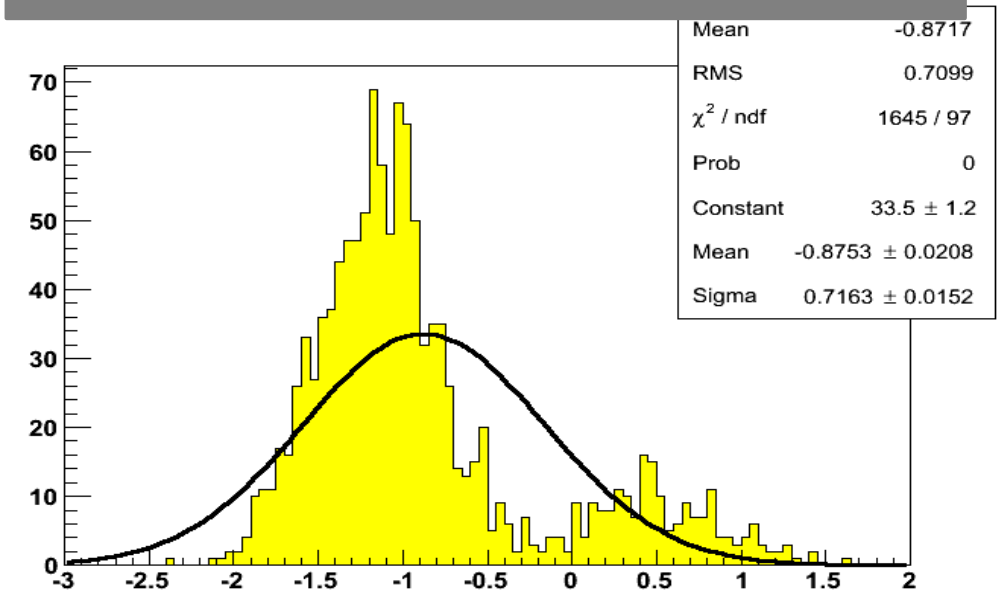
il file gif si chiama **isto.gif**

la media non si sovrappone al picco perché e' calcolata su tutta la distribuzione

Fit dell'istogramma

- cercare i parametri di una distribuzione nota, in modo che si adatti all'istogramma
- utilizzo una gaussiana per cercare il picco: $p_0 \times \exp\left(-\frac{(x-p_1)^2}{2p_2^2}\right)$

```
TF1 gauss ("gauss","gaus",-3,1) ;
gauss.SetParameter (1,histo.GetMean ()) ;
gauss.SetParameter (2,histo.GetRMS ()) ;

histo.Fit ("gauss","L") ;
histo.Draw () ;
c1.Print ("istoFit1.gif","gif") ;
```



la funzione di fit in ROOT e' di tipo **TF1**

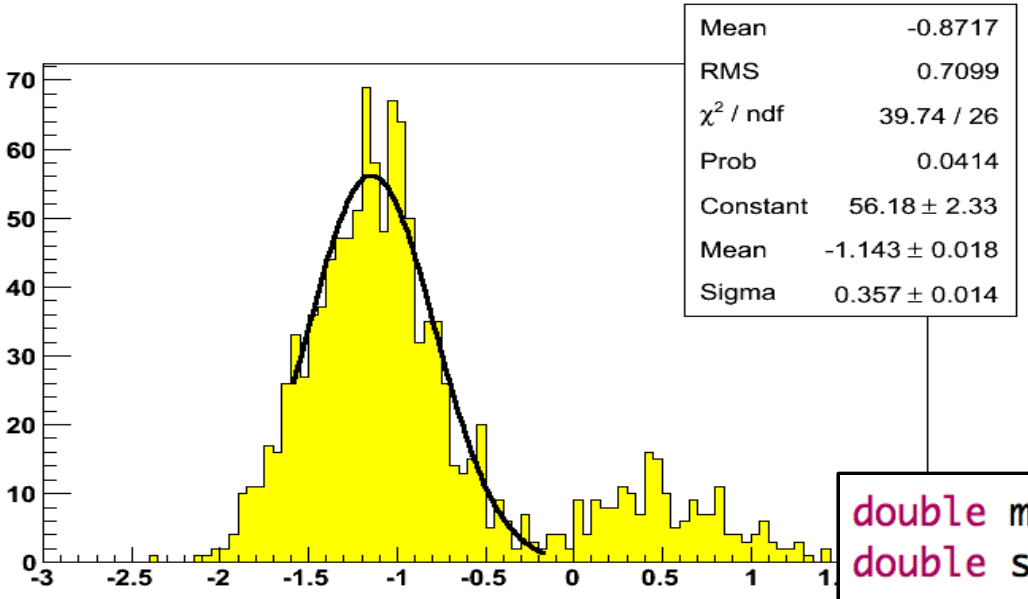
i parametri iniziali servono per guidare il fit verso la direzione giusta, quindi uso i parametri dell'istogramma il piu' possibile

l'opzione “L” impone a ROOT di fare il fit massimizzando una Likelihood

anche il fit risente della presenza del picco minore

Fit del picco

- Per trovare il picco, faccio il fit soltanto nella regione di interesse
- Estraggo la regione di interesse da parametri noti, cioè il fit precedente



estraggo dalla funzione il valore dei suoi parametri

faccio il fit soltanto fra (μ-σ e μ+σ)

i nuovi parametri, ottenuti dal secondo fit, danno μ e σ per il picco principale

```
double mean = gauss.GetParameter (1) ;
double sigma = gauss.GetParameter (2) ;

histo.Fit ("gauss","L","",mean-sigma, mean+sigma) ;
histo.Draw () ;
c1.Print ("istoFit2.gif","gif") ;
```

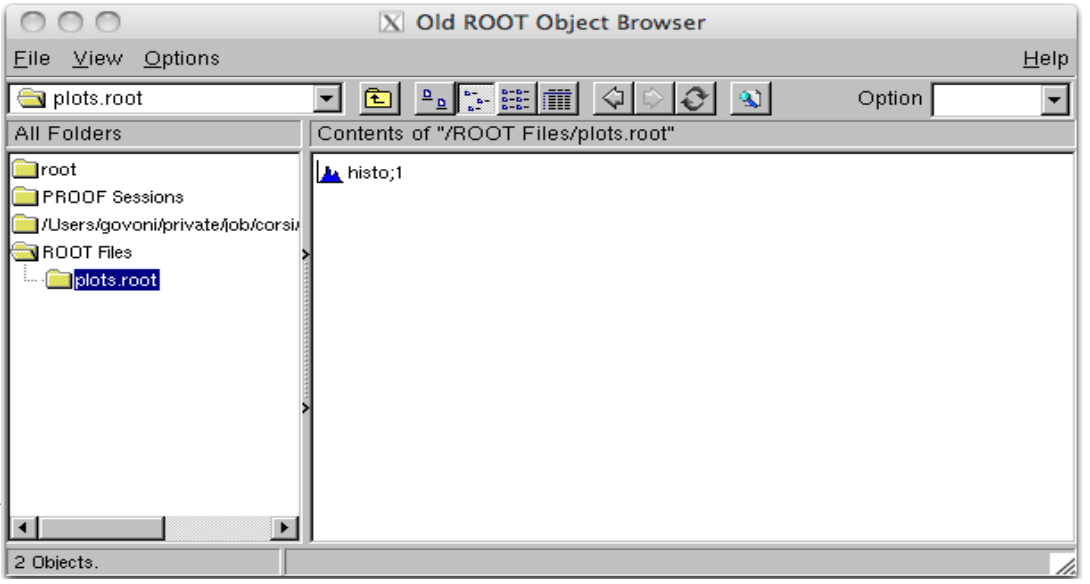
Salvataggio dell'istogramma

- nello stesso programma, l'istogramma può essere salvato su un file di ROOT, per rileggerlo senza bisogno di riempirlo ogni volta

```
TFile saving ("plots.root","recreate") ;
histo.Write () ;
saving.Close () ;
```

- Per rileggere l'istogramma, aprire il file di ROOT eseguendo il comando (da *shell*) **`root plots.root`**
  - Il file viene caricato col nome **`_file0`**

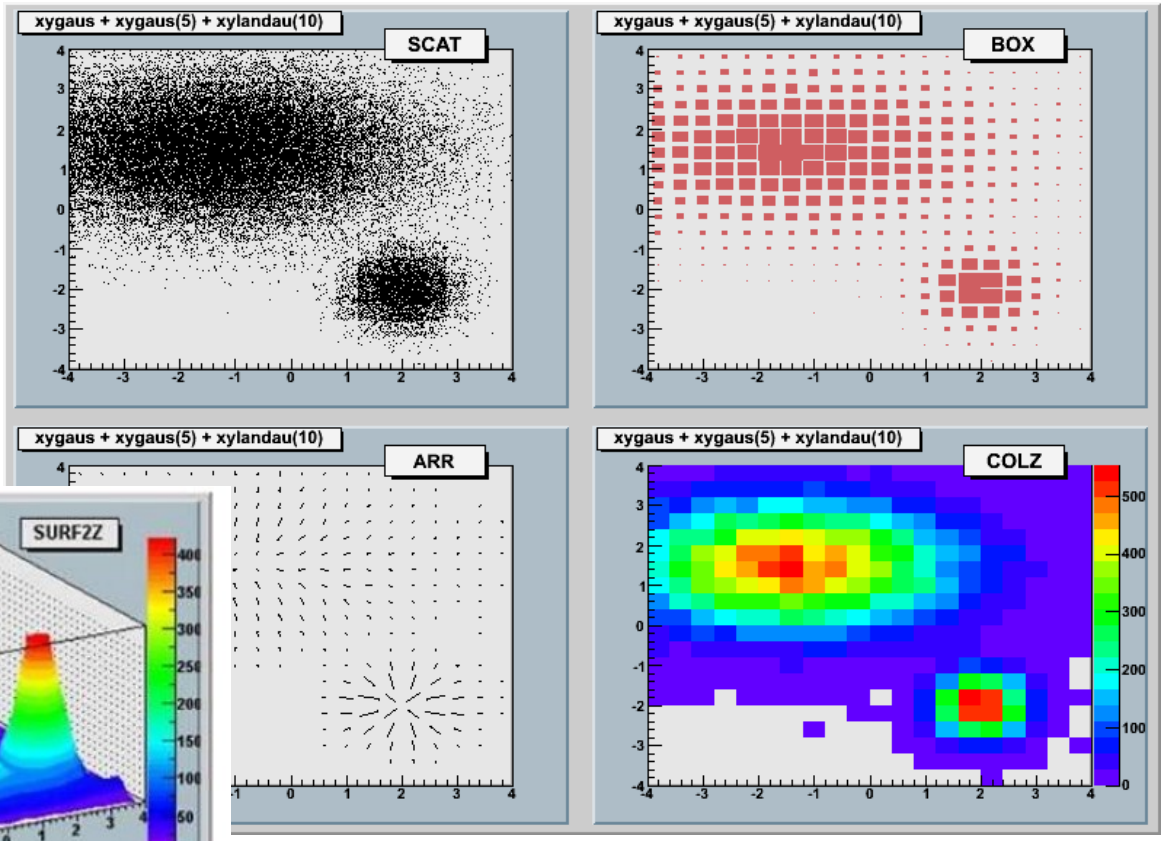
```
root [0]
root [2] _file0->ls ()
TFile**         plots.root
 TFile*         plots.root
  KEY: TH1F     histo;1
root [3] histo->Draw ()
root [2] TBrowser b
```

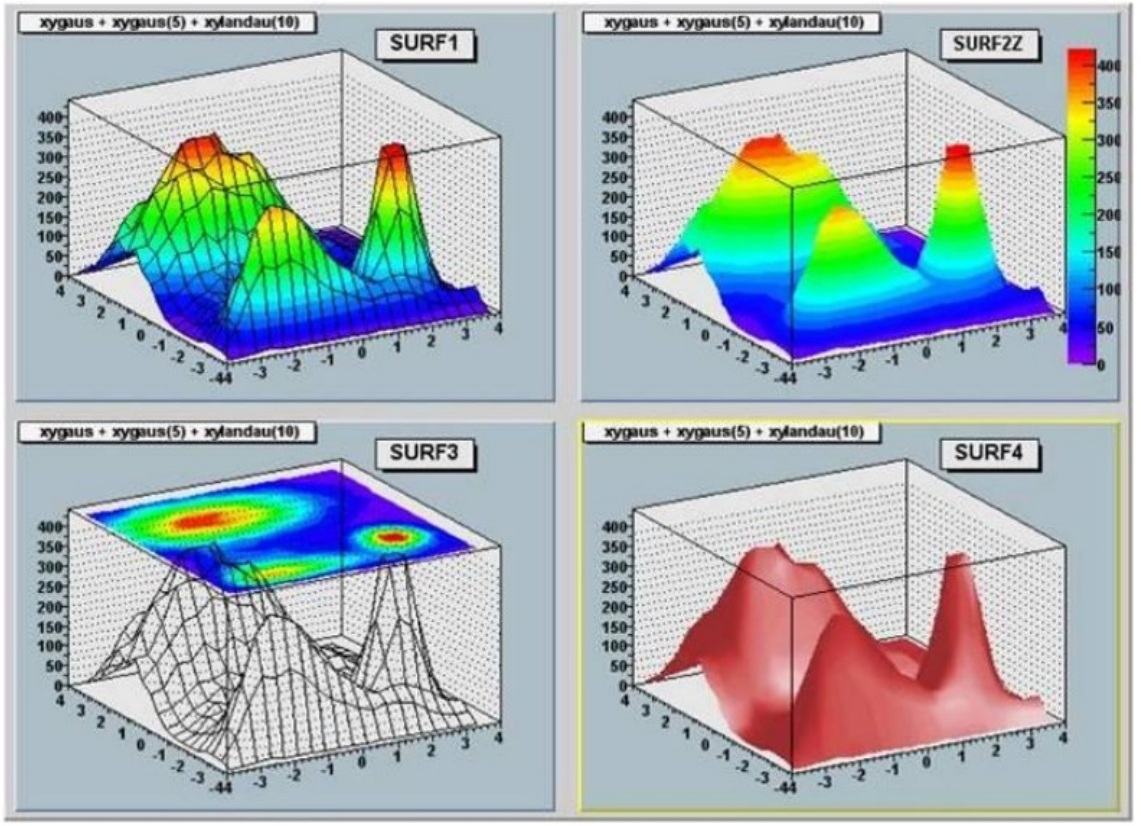


Istogrammi 2D - TH2F

```
#include "TH2F.h"
```

In un **TH2F** ci sono **due assi** per le variabili → ogni **bin** è un **rettangolo**!





Insieme di Mandelbrot

- Possiamo rappresentare l'insieme di **mandelbrot**:
  - Ad ogni punto *c* del piano complesso (nel range [-2, 1]x[-1,1]) associamo un numero *N* che indica la rapidità della divergenza della successione $z_{n+1} = z_n^2 + c$, con $z_0 = 0$
  - Per ogni *n*, la successione diverge se $|z_n| > 2$
- Quindi noi possiamo fare così;
  - Scegliamo un numero massimo di iterazioni N = ~100
  - Controlliamo, per ogni *c*, a quale step *n* la successione diverge
  - Questo valore di *n* è il contenuto del bin associato a *c*

Riempire un istogramma 2D

```
TH2F histo("histo", "", 600, atof(argv[1]), atof(argv[2]), 400, atof(argv[3]), atof(argv[4]));

// loop sui bin - asse x
for(int ix = histo.GetXaxis() -> GetFirst();
    ix <= histo.GetXaxis() -> GetLast(); ++ix)
{
  // loop sui bin - asse y
  for(int iy = histo.GetYaxis() -> GetFirst();
      iy <= histo.GetYaxis() -> GetLast(); ++iy)
  {
    complessi c(histo.GetXaxis() -> GetBinCenter(ix), histo.GetYaxis() -> GetBinCenter(iy));

    complessi z_old(0., 0.);
    complessi z_new(0., 0.);

    int n = 0;
    for( ; n < 100; ++n)
    {
      z_new = z_old*z_old + c;

      if(z_new.Mod() > 2.) break;

      else z_old = z_new;
    }

    histo.Fill(c.Re(), c.Imm(), n);
  }
}
```

Gli estremi dell'istogramma (xMin, xMax) e (yMin, yMax) sono argomenti di **int main**

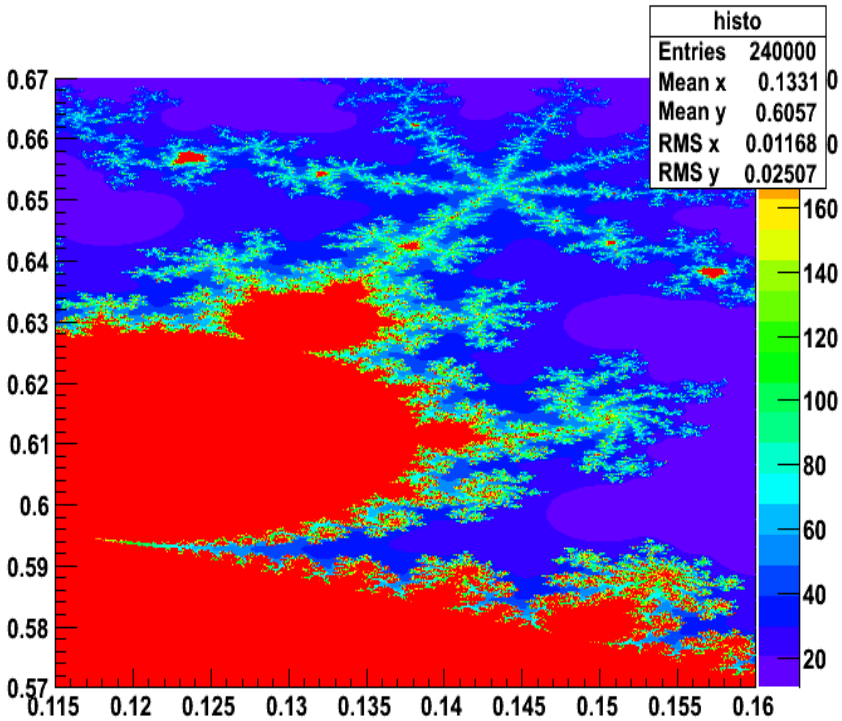
Per effettuare il *loop* sui bin dell'istogramma, uso il metodo **`GetXaxis()`**, che restituisce un oggetto di tipo **`TAxis`**, al quale applico i metodi **`GetFirst()`** e **`GetLast()`**

Riempio l'istogramma con il metodo **`Fill`**, al quale passo tre valori: la **coppia x,y** che identifica il bin da riempire, e un **terzo valore**, che indica il peso da dare a quel bin

Disegnare un istogramma 2D

- Esistono diversi modi per rappresentarlo:
  - Densità di punti diversa a seconda del contenuto del bin (“SCAT”)
  - Colori diversi a seconda del contenuto di ciascun bin (“COLZ”)
  - Contenuto del bin rappresentato sull'asse *z* (“LEGO”)

```
TCanvas c1;
c1.cd();
histo.Draw("COLZ");
c1.Print("mandelbrot.png", "png");
```



Decido quale visualizzazione adottare tramite l'argomento (opzionale!) **`*char`** del metodo **`Draw`**

Consigli grafici

- Le opzioni di visualizzazione che ROOT usa di *default* sono "brutte"
- E' possibile cambiarle scrivendo nel codice le seguenti righe:

- `gROOT → SetStyle("Plain");`

Fa sì che i plot abbiano sfondo bianco anziché grigino. Serve **`#include "TROOT.h"`**

- `gStyle → SetPalette(1);`

Fa sì che la tavolozza di colori (usata per esempio nella visualizzazione "COLZ") segua una scala termica

- `gStyle → SetOptStat(1);`

Stampa in un riquadro le statistiche dell'istogramma e i risultati del fit (vedere la classe per la legenda dei parametri)

- `gStyle → SetOptFit(1);`

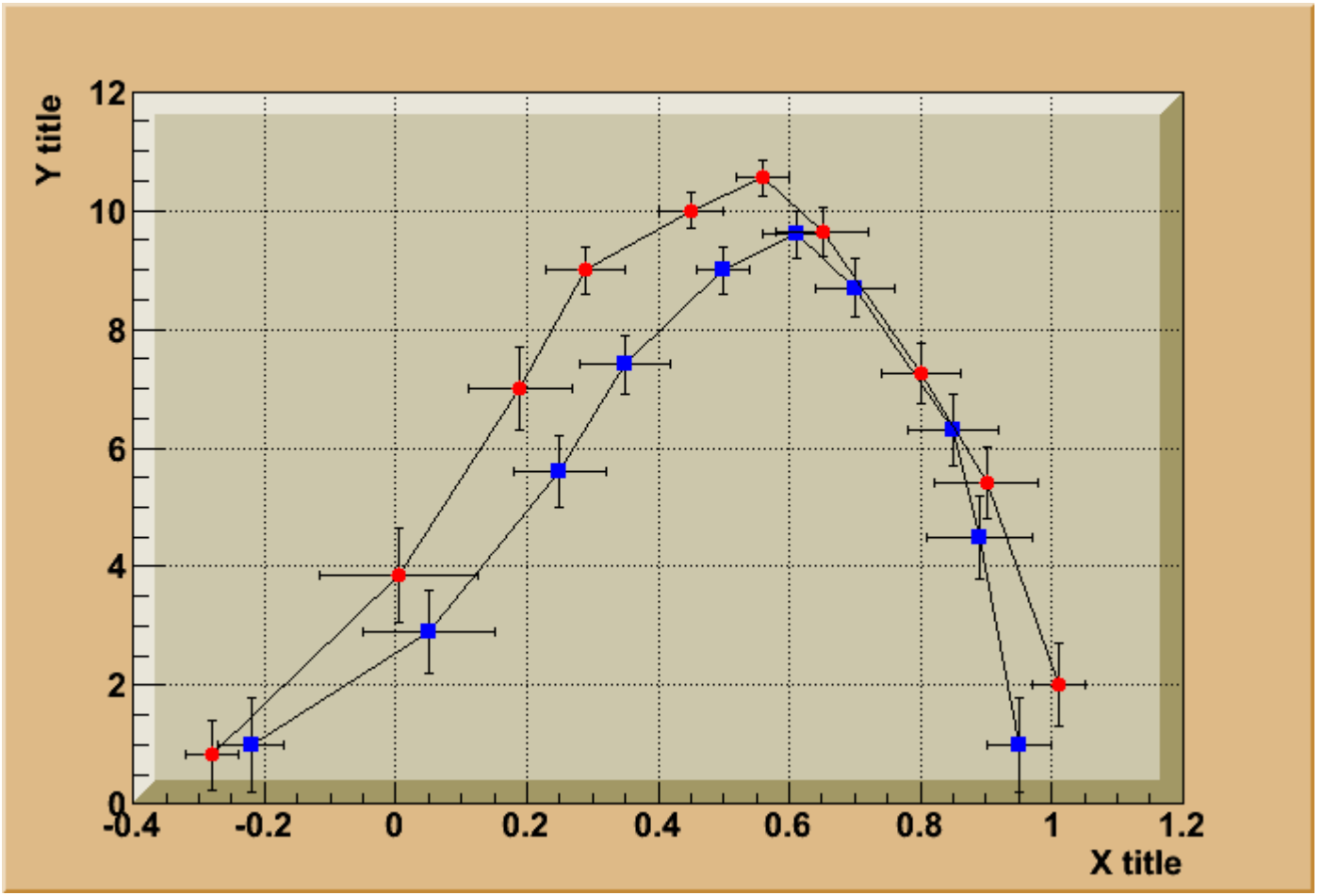
Serve **`#include "TStyle.h"`**

Fino ad ora

- Oggetti incontrati:
  - TH1F, TH2F, TF1, TCanvas, TFile, TBrowser
- Cosa sappiamo fare:
  - Salvare istogrammi e riempirli a partire da un file di testo
  - Fare un fit dei dati con funzioni predefinite in ROOT
  - Disegnare gli oggetti in un canvas
  - Salvare gli oggetti in un file .root e navigarci

Grafici - TGraph

```
#include "TGraph.h"
```



Costruire un grafico

- Oggetto **`TGraph`**: è costituito da due *array X* ed *Y* di *N* punti ciascuno
  - Consente la rappresentazione di ***Y*** vs. ***X*** entrambe variabili fisica e di vedere eventuali correlazioni
    - *reminder*: TH1F → X è variabile fisica, Y è una frequenza
    - *reminder*: TH2F → X e Y variabili fisiche, su Z c'è una frequenza

```
TGraph graph;
TGraph graph2;

int nPoints = 10;
for(int i = 0; i < nPoints; ++i)
  {
    double x = i * 2*3.14 / nPoints;

    graph.SetPoint(i, x, sin(x));
    graph2.SetPoint(i, x, sin(x - 0.5));
  }
```

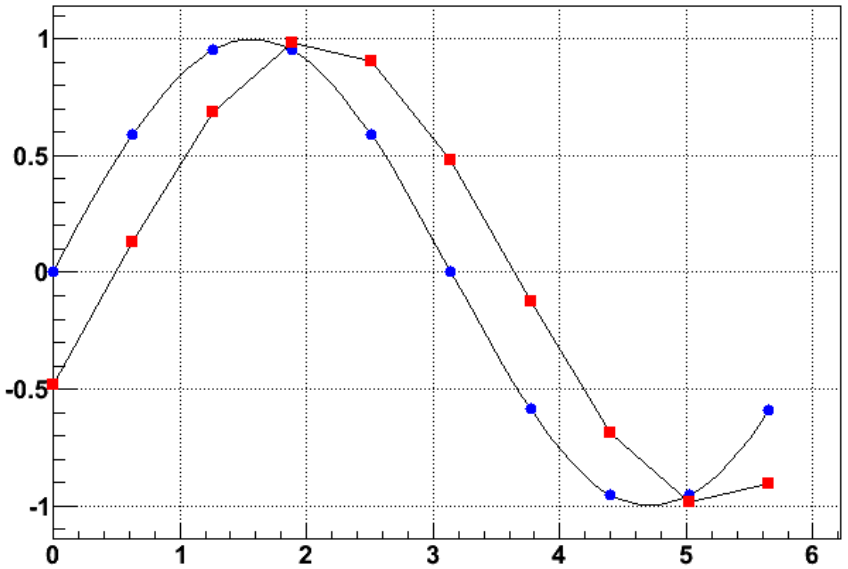
Costruisco l'oggetto **`TGraph`** e inserisco le coppie *(x,y)* con il metodo **`SetPoint`**

Esistono altri costruttori (→ vedere guida)

NB: il primo punto del grafico ha indice 0!

Rappresentare un grafico

```
graph.SetMarkerStyle(20);
graph.SetMarkerColor(kBlue);
graph.GetXaxis()->SetTitle("X");
graph.GetYaxis()->SetTitle("Y");

graph2.SetMarkerStyle(21);
graph2.SetMarkerColor(kRed);

TCanvas c1;
c1.SetGridx();
c1.SetGridy();
graph.Draw("APC");
graph2.Draw("PL,same");
c1.Print("graph.gif","gif");
```



Impostate le proprietà del grafico con i metodi opportuni (colori, forme dei *marker*, ecc.)

Per disegnare un grafico bisogna passare alcune opzioni al metodo **`Draw`**:

- **`"A"`** → vengono disegnati gli assi x e y del grafico
- **`"P"`** → vengono disegnati i punti
- **`"C"`** o **`"L"`** → i punti vengono collegati con una linea

Qui ed altrove, l'opzione **`"same"`** consente di disegnare più di un oggetto sullo stesso **`TCanvas`**

Grafico con errori

- E' l'oggetto **`TGraphErrors`**: utile per rappresentare il risultato di una **misura sperimentale** – valori misurati + errori di misura (vedi Lab. di Fisica 2° sem.)
- In questo esempio, il file di dati contiene un possibile risultato della misura del ΔV ai capi di un condensatore di un circuito *RC* in carica

```
TGraphErrors data_graph(argv[1]);

TF1 fitFunc("fitFunc", "[0]*(1. - exp(-x/[1]))", 0., 0.01);
fitFunc.SetParameters(10., 0.002);
fitFunc.SetLineColor(kRed);

data_graph.Fit("fitFunc");

gStyle -> SetOptFit(1111);
```

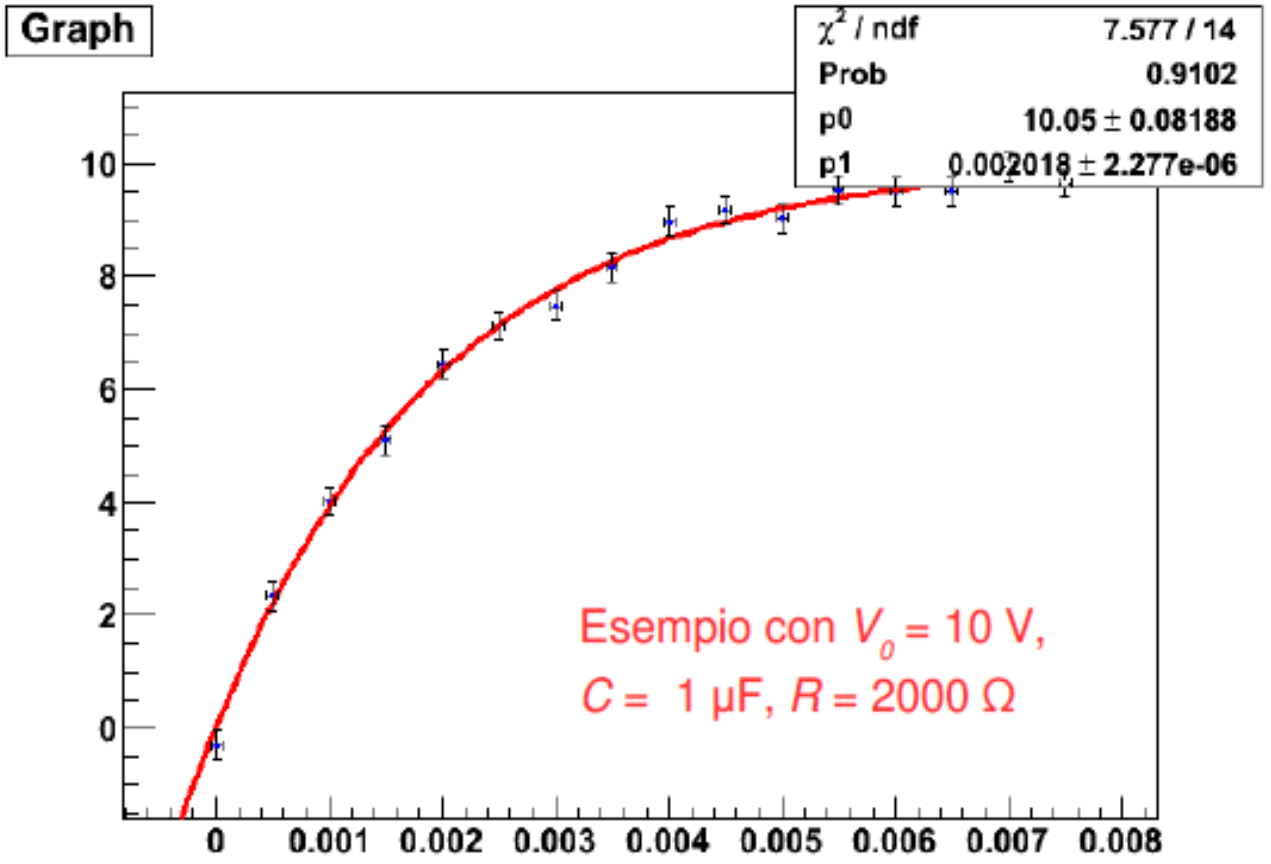
Al costruttore del **`TGraphErrors`** si passa il nome del file che contiene i dati. Il file è costituito da 4 colonne di valori numerici:

x y xErr yErr

Definiamo la funzione di fit con l'andamento atteso: $V_0\,[1 - \exp(-x/\tau)]$.

La forma funzionale costituisce il secondo argomento del costruttore del **`TF1`**:
per le variabili si utilizzano le lettere *x*,*y*,*z*..., i parametri di fit sono indicati come *[0]*, *[1]*, *[2]*,...

Il fit di un grafico



NB: gli errori sull'asse delle ascisse vengono tenuti in considerazione da ROOT durante la procedura di minimizzazione del χ^2. (vengono trasportati sull'asse delle ordinate in base alla pendenza della curva in ciascun punto)

```
std::cout << " Parametro p0 = " << fitFunc.GetParameter(0)
          << " errore su p0 = " << fitFunc.GetParError(0)
          << std::endl;
std::cout << " Parametro p1 = " << fitFunc.GetParameter(1)
          << " errore su p1 = " << fitFunc.GetParError(1)
          << std::endl;
```

Si utilizzano i metodi **GetParameter** e **GetParError** per estrarre i parametri dalla TF1