Introduzione a ROOT
(II parte)

Analisi dati con ROOT

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

ROOT

An Object-Oriented
Data Analysis Framework

Fino ad ora

- Oggetti incontrati:
  - **`TH1F`**, **`TH2F, TF1`**
  - **`TGraph, TGraphErrors`**
  - **`TCanvas`**, **`TFile`**, **`TBrowser`**
- Cosa sappiamo fare:
  - Salvare istogrammi e riempirli a partire da un file di testo
  - Creare grafici (Y vs X entrambe variabili fisiche) con gli errori per ogni punto
  - 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

TNtuple : public TTree

- Consente di salvare in un unico oggetto la lista di tutte le misure
- Ogni singola azione di misura è chiamata evento
- In un singolo evento può essere misurata più di una variabile (nella `TNtuple` c'e' un **`branch`** per ogni variabile)

```
// definisco la ntupla
TNtuple ntu("ntu", "ntu", "x1:x2");

// importo i dati dal file
std::ifstream inFile(argv[1], std::ios::in);

while(!inFile.eof())
{
  // leggo i valori (x1,x2) dal file
  double num1, num2;
  inFile >> num1 >> num2;

  // inserisco le coppie (x1,x2) in una ntupla
  ntu.Fill(num1, num2);
}
```

I nomi delle singole variabili sono definiti in una stringa di caratteri, separati da ":" - in questo caso "`x1:x2`"

Per riempire la ntupla, uso il metodo **`Fill`**, a cui passo tanti valori *float* quante sono le colonne delle variabili (*branch)* che ho definito

Disegnarne il contenuto

- Esistono diverse possibilità:

```
// stampo su file l'istogramma 1D per x1
TCanvas c1("c1", "histo1D_x1");
ntu.Draw("x1");
c1.Print("histo1D_x1.png", "png");
```

Uso il metodo **`Draw`**, a cui passo il nome della variabile di cui voglio l'istogramma

NB: in questo modo non posso scegliere il binning dell'istogramma – è scelto in maniera automatica

```
// stampo su file l'istogramma 1D per x2
TH1F histo1D_x2("histo1D_x2", "", 400, 0., 20.);
TCanvas c2("c2", "histo1D_x2");
ntu.Draw("x2 >> histo1D_x2");
histo1D_x2.SetFillColor(5);
c2.Print("histo1D_x2.png", "png");
```

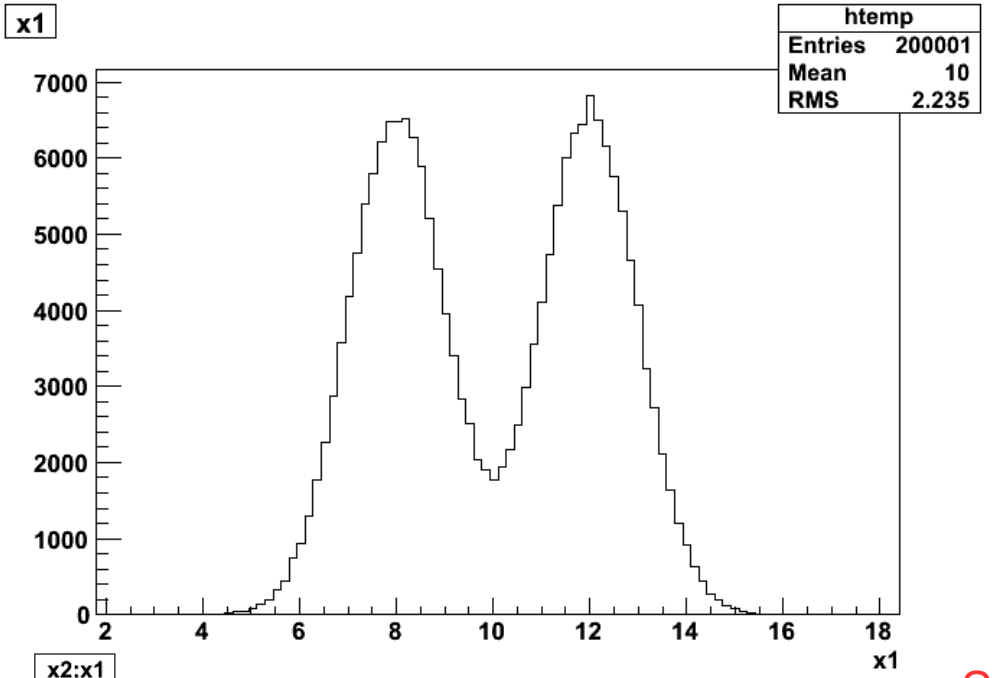
Istanzio un oggetto **`TH1F`** (con le caratteristiche desiderate)

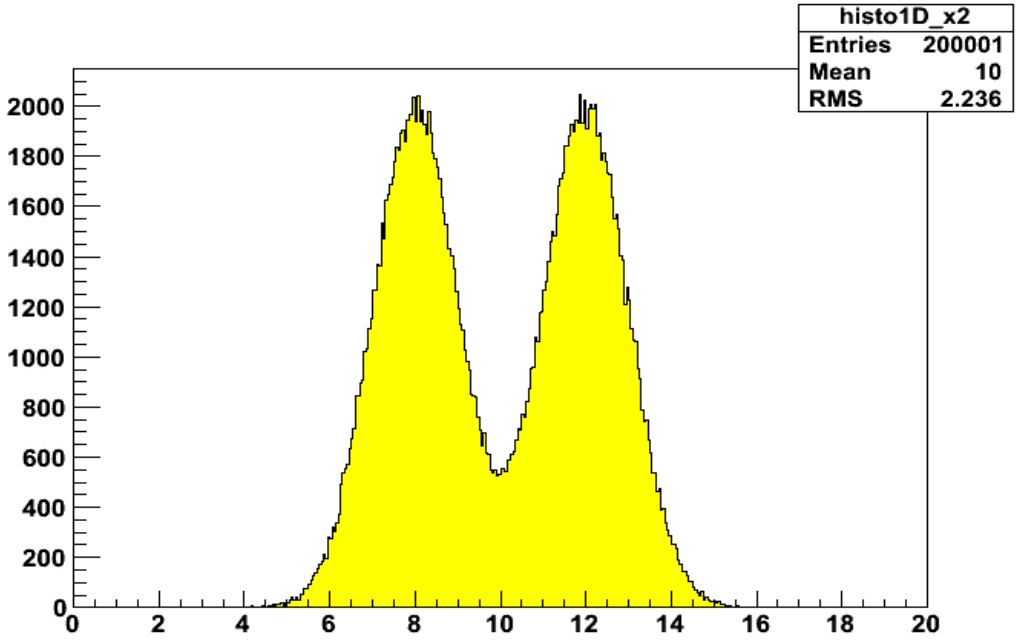
Chiamo il metodo **`Draw`**, a cui passo il nome della variabile che voglio disegnare, e metto il risultato nell'istogramma precedentemente definito con il simbolo

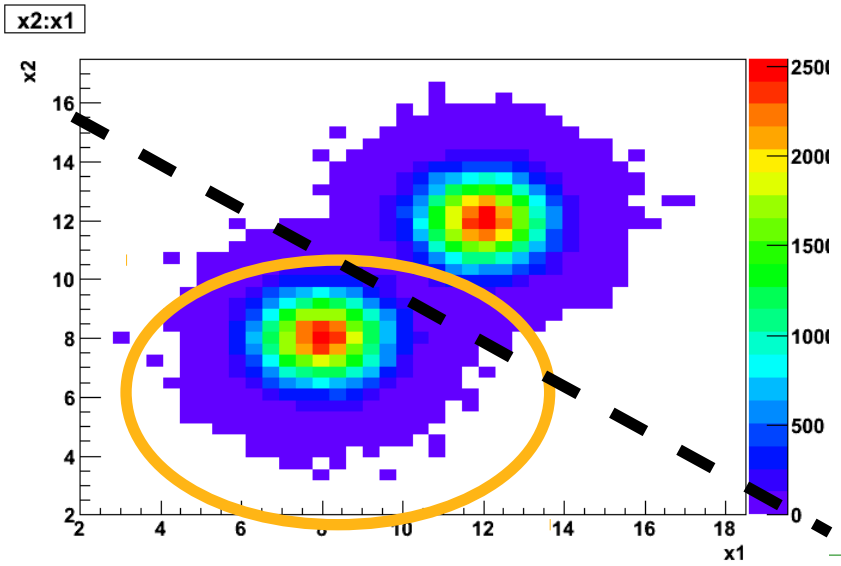
```
// stampo su file l'istogramma 2D per x2 vs x1
TCanvas c3("c3", "histo2D_x2_vs_x1");
ntu.Draw("x2:x1", "", "COLZ");
c3.Print("histo2D_x2_vs_x1.png", "png");
```

Posso anche disegnare una variabile verso l'altra semplicemente scrivendo **`Draw("x2:x1")`**

Il risultato







Supponiamo di voler conoscere la forma della distribuzione di *x1* per gli eventi che cadono nella regione evidenziata

L'istogramma qui sopra non mi aiuta, la distribuzione è alterata dagli eventi che non cadono in quella regione → devo effettuare un **taglio per eliminare gli eventi che non mi interessano**

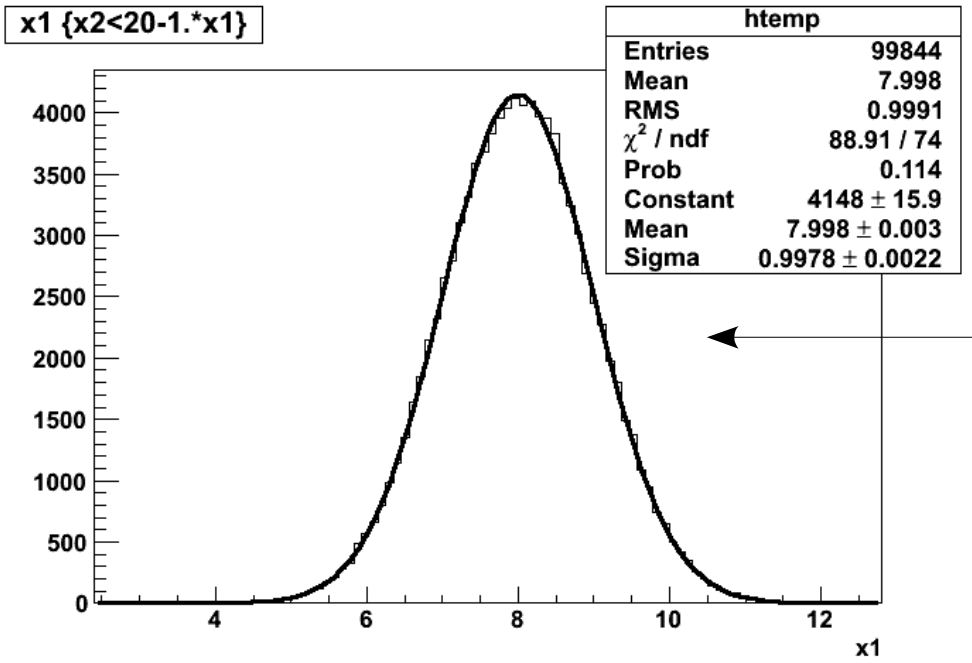
Fit di variabili di una ntupla

- Vogliamo *fittare* la distribuzione di *x1* con una gaussiana, solo per gli eventi che appartengono al *blob* inferiore:

```
// fitto x1 dopo aver effettuato il taglio
TCanvas c4("c4", "histo1D_x1_CUT");

TF1 fitFunc("fitFunc", "gaus", 0., 20.);
ntu.Fit("fitFunc", "x1", "x2<20-1.*x1", "");

c4.Print("histo1D_x1_CUT.png", "png");
```



Chiamo il metodo **`TTree::Fit`**, a cui passo il nome della funzione di fit, il nome del *branch* da *fittare* e, come terzo argomento, la **condizione che deve essere rispettata**

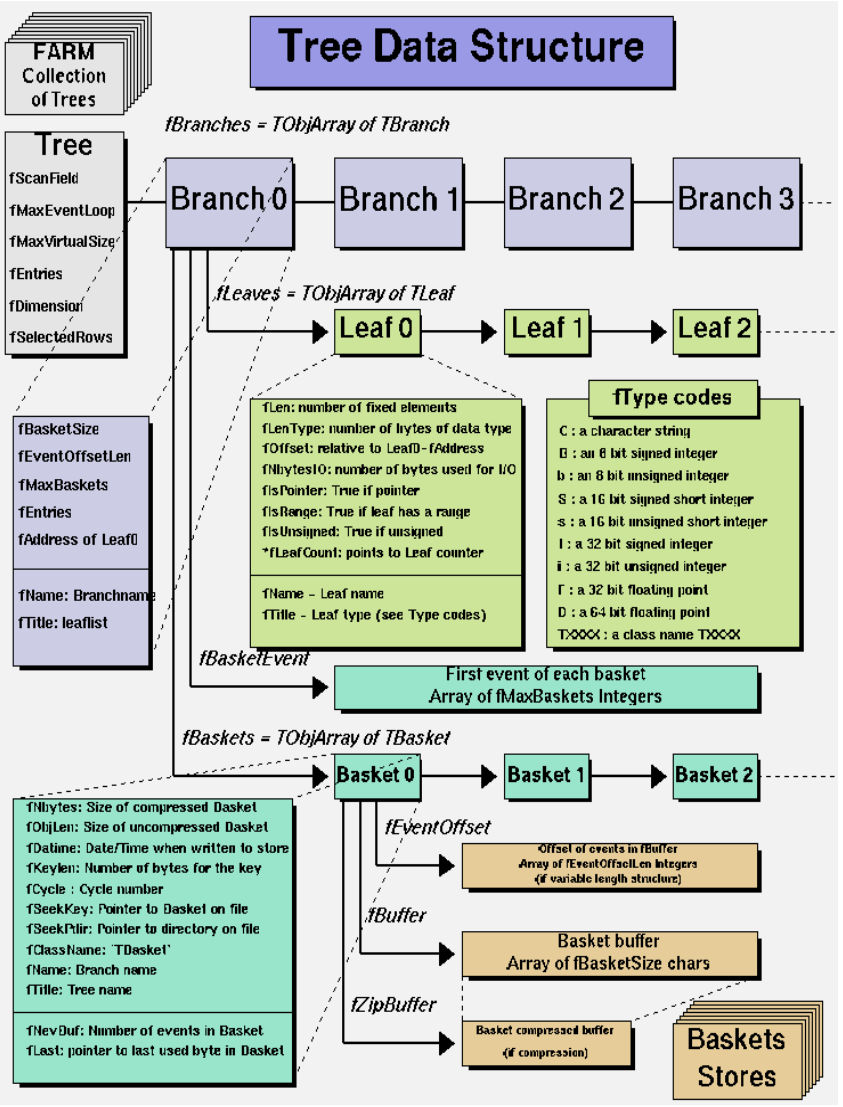
La condizione è un'espressione logica: solo se il risultato è **`true`**, l'operazione viene eseguita

Solo gli eventi in cui x2 < 20 – x1 (sotto la retta tratteggiata del plot di pag. 30) sono considerati

Più in generale: i TTree

- Per salvare oggetti generici (non solo `float`) si usano i `TTRee` (non più le `TNtuple`)

- Sono una soluzione comoda per salvare grandi quantità di dati
  - Spazio disco limitato
  - Maggiore velocità di accesso ai dati
- Le diverse quantità salvate nel *Tree* costituiscono i *branch* del *Tree*
  - Ogni branch può essere letto in maniera indipendente dagli altri



Unbinned fit

- Quando si riempie un istogramma, alcune informazioni sono perse:
  - Non si conosce il valore di ogni singola *entry* dell'istogramma
  - Si conosce solo la frequenza associata ad ogni bin
- Una ntupla fornisce un metodo di semplice utilizzo per effettuare un **fit non binnato,** cioè che tiene conto dell'informazione contenuta in ogni singolo evento
  - Viene costruita, evento per evento, una funzione di verosimiglianza

$$L(\theta) = \prod_{i=1}^{N} f(x_i|\theta)$$

    dove f è la pdf attesa per la variabile in questione
  - Viene minimizzata *L* per ottenere la migliore stima di *θ*

Unbinned fit

```
// unbinned fit
TF1 fitFunc2("fitFunc2", "1./[1]/sqrt(2.*3.14159)*exp(-1.*(x-[0])*(x-[0])/2./[1]/[1])", 0., 20.);
fitFunc2.SetParameters(8., 1.);
ntu.UnbinnedFit("fitFunc2", "x1", "x2<20-1.*x1", "");

std::cout << "Mean = " << fitFunc2.GetParameter(0);
std::cout << "   +/- " << fitFunc2.GetParError(0) << std::endl;

std::cout << "Sigma = " << fitFunc2.GetParameter(1);
std::cout << "   +/- "  << fitFunc2.GetParError(1) << std::endl;
```

È sufficiente chiamare il metodo **`TTree::UnbinnedFit`**

NB: la funzione di fit, in questo caso, deve **necessariamente** essere una pdf (**normalizzata ad area = 1**)

```
 FCN=141582 FROM MIGRAD    STATUS=CONVERGED      31 CALLS          32 TOTAL
                     EDM=8.0684e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                          STEP         FIRST
  NO.   NAME      VALUE            ERROR                 SIZE      DERIVATIVE
   1  p0           7.99802e+00   3.16189e-03   8.21554e-04  -2.74546e-06
   2  p1           9.99096e-01   2.23579e-03   5.82240e-04   4.01756e-02
                               ERR DEF= 0.5
Mean = 7.99802   +/- 0.00316189
Sigma = 0.999096   +/- 0.00223579
[andrea@despero:~] 
```