Introduzione a ROOT

1. Informazioni generali

ROOT è un ambiente visualizzazione e trattamento dati interattivo sviluppato al CERN (si veda il sito ufficiale `http://root.cern.ch` interamente sviluppato in linguaggio C++ e basato su un interprete di C/C++ chiamato CINT.

Oltre a mettere a disposizione un'interfaccia interattiva, esso fornisce un insieme di classi utili per l'analisi statistica dei dati.

2. Utilizzo di ROOT

Root puo' essere utilizzato essenzialmente in tre modi: in interattivo, con una macro interpretata e in un programma C/C++ compilato. Nell'ultima modalita' (in generale preferita) ROOT viene essenzialmente utilizzato come una libreria di funzioni utili che vengono incluse nel proprio programma C (o C++).

2.1 Interattivo

Si attiva una sessione di ROOT digitando `root` nel prompt dei comandi. Si può uscire dalla sessione digitando `.q` sulla linea di comando di ROOT.

2.2 Con macro non compilata

Si possono scrivere direttamente in ROOT quasi tutte le istruzioni di C e C++, che CINT interpreta ed esegue una alla volta.

Una serie di istruzioni possono essere scritte in un file racchiuse da delle parentesi graffe (questo tipo di scipts sono detti "unamed"), ad esempio

```
{
  int a = 10;
  int b = 5;
  cout << a + b << endl;
}
```

e successivamente eseguite semplicemente con `.x nome_file`. Si noti che le istruzioni sono ugualmente eseguite una per volta (da un programma che si chiama CINT) , rendendo la loro esecuzione più lenta rispetto a quella di un programma compilato.

Per facilitare la possibilita' di compilare le macro di ROOT che scriviamo e' consigliabile scrivere le istruzioni in "named" scripts, nel modo seguente:

```
int myMacro(){
  int a = 10;
  int b = 5;
  cout << a + b << endl;
  return 0;
}
```

Si può anche scrivere una o più funzioni in un file, con lo stessa sintassi delle normali funzioni di C/C++, e poi caricarle nella sessione corrente di ROOT digitando `.L nome_file`.

2.3 In programmi C/C++ compilati

In questo caso il prototipo e' un “named” script con alcune differenze:

1) Occorre includere tutti i files necessari
2) Occorre definire una TApplication : questo serve a far si che al termine dell'esecuzione del programma le finestre grafiche restino visibili.

Ecco un esempio di come appare un programma C che includa librarie di ROOT

```
#include "TCanvas.h"
#include "TF1.h"
#include <iostream>
#include "Tapplication.h"
....
using namespace std;
int main (int argc, char**argv)
{
  TApplication app("App",&argc, argv);
  int a = 10;
  int b = 5;
  cout << a + b << endl;
  TF1 *f1 = new TF1("f1","sin(x)/x",0.,10.)
  .....
  app.Run();
  return 0;
}
```

Ovviamente nella fase di compilazione dovremo indicare al compilatore dove deve andare a cercare le librerie di ROOT richieste dal nostro programma (prog.C):

g++ -o prog.x prog.C `root-config --cflags` `root-config --libs`

3. Le classi di ROOT

Tra le tante classi di ROOT prendiamo in considerazione quelle utili per creare e disegnare funzioni matematiche, grafici e istogrammi.

3.1 La classe TF1

Con l'istruzione

```
TF1 *f1 = new TF1("f1","sin(x)/x",0.,10.)
```

si dichiara un puntatore ad una variabile di tipo `TF1`, cioè una funzione monodimensionale, di nome `f1` e che corrisponde alla funzione *f(x) = sin(x)/x*. La variabilità della funzione viene limitata all'intervallo *0 < x < 10*.

Una classe contiene in sé una serie di funzioni, chiamati metodi, che permettono di gestire le variabili della classe. I metodi si possono chiamare con la sintassi:

```
f1->Draw()
```

In questo caso in metodo `Draw()` disegna la funzione `f1` appena dichiarata.

E' possibile modificare l'aspetto grafico, ad esempio il colore della linea, lo spessore, la dimensione della finestra, semplicemente cliccando con il tasto destro del mouse sugli oggetti che si vogliono modificare. Apparirà un menù a tendina con le varie caratteristiche che è possibile modificare. Ogni modifica che è possibile fare in modo interattivo, si può anche fare utilizzando del codice. Il grafico creato si può salvare utilizzando il menu `File -> Save As`; si può salvare in diversi formati, il più comodo per poter successivamente stampare è il formato PostScript (.ps).

Alcuni metodi di TF1:

```
f1->SetRange(-2.,15.) //Cambia l'intervallo di definizione
f1->Eval(1.)          // Valutarla in un punto
f1->Derivative(1.)    // Valutarne la derivata in un punto
f1->Integral(0.,1.)   // Calcolare l'integrale definito
```

Si può definire una nuova funzione a partire da `f1`:

```
TF1 *f2 = new TF1("f2","2.*f1",0.,10.)
```

Ci saranno utili per eseguire fit funzioni dipendenti da parametri:

```
TF1 *f3 = new TF1("f3","[0]*x*sin([1]*x)",-3.,3.)
f3->SetParameter(0,10.)  // Definire il parametro [0] a 10.
f3->SetParameters(10.,5.)// Definire [0] a 10. e [1] a 5.
```

3.2 La classe TGraph

Due vettori del medesimo tipo possono descrivere dei punti un piano bidimensionale. La classe TGraph gestisce i grafici a punti.

```
{
  int n = 20;
  double x[20], y[20];
  for (int i = 0; i < n; i++){
    x[i] = i*0.5;
    y[i] = 10.*sin(x[i]+0.2);
  }
  TGraph *g = new TGpraph(n,x,y); // Definisce il grafico
  g->Draw("A*");    // A = Disegna gli assi
                    // * = Segna i punti con un *
}
```

3.3 La classi TH1F e TH2F

Per la gestione di istogrammi esistono le classi `TH1F` (istogramma monodimensionale riempito con `float`) e `TH2F` (istogramma bidimensionale riempito con `float`):

- Definizione:

  ```
  TH1F *h1 = new TH1("h1","Isto 1D",100,-5.,5.)
  TH2F *h2 = new TH2("h2","Isto 2D",100,-5.,5.,100,-5.,5.)
  ```

- Riempimento (dove `x` e `y` sono `float`):

  ```
  h1->Fill(x)
  h2->Fill(x,y)
  ```

- Grafico:

  ```
  h1->Draw()
  h2->Draw()
  ```

3.4 La classe TCanvas

In ognuna delle classi precedenti, la chiamata del metodo Draw() produce una finestra grafica in cui viene disegnato il grafico desiderato. Ogni successiva chiamata di un metodo Draw() sovrascrive il nuovo grafico sulla precedente finestra grafica.

Per evitare ciò si possono creare finestre grafiche definendo variabili di tipo `TCanvas`.

La sequenza di comandi

```
TCanvas *c1 = new TCanvas("c1","Istogramma 1D",600,400)
h1->Draw()
TCanvas *c2 = new TCanvas("c2","Istogramma 2D",600,400)
h2->Draw()
```

produce due finestre distinte ciascuna con l'istogramma richiesto.

4. Eseguire fit con ROOT

Supponiamo di avere due vettori `x` e `y` di dimensione `N` e di voler trovare la retta che meglio descrive l'andamento di *y(x)*.

```
TGraph *retta = new TGraph(N,x,y)
retta->Draw("A*")
retta->Fit("pol1")    // polN = polinomio di grado N
                      // gaus = gaussiana
```

Ulteriormente, possedendo anche i valori degli errori su `x` e `y`, si può eseguire un fit pesato:

```
TGraphErrors *retta_err = new TGraphErrors(N,x,y,errx,erry)
retta_err->Draw("A*")
retta_err->Fit("pol1")
```

Al posto di una funzione predefinita di ROOT come `polN` o `gaus`, è consigliabile utilizzare una funzione definita dall'utente, ad esempio:

```
TF1 *f1 = new TF1("f1","[0]+[1]*x",0.10.)
retta_err->Draw("A*")
retta_err->Fit("f1")
```

In questo modo, non solo è possibile fittare con qualsiasi funzione, ma permette anche di recuperare in modo immediato il risultato del fit:

```
f1->GetParameter(0)
f1->GetParameter(1)
f1->GetChisquare()
```

Si noti che per recuperare i parametri ottenuti fittando con la funzione predefinita `gaus` avremmo dovuto prima recuperare la funzione usata per il fit e successivamente recuperare i parametri.

```
TF1 *f1 = retta_err->GetFunction("gaus")
f1->GetParameter(0)
f1->GetParameter(1)
```

```
f1->GetChisquare()
```

Nel caso di fit di un istogramma, il funzionamento del metodo `Fit()` è del tutto identico.