Lezione 1

Argomenti della lezione:

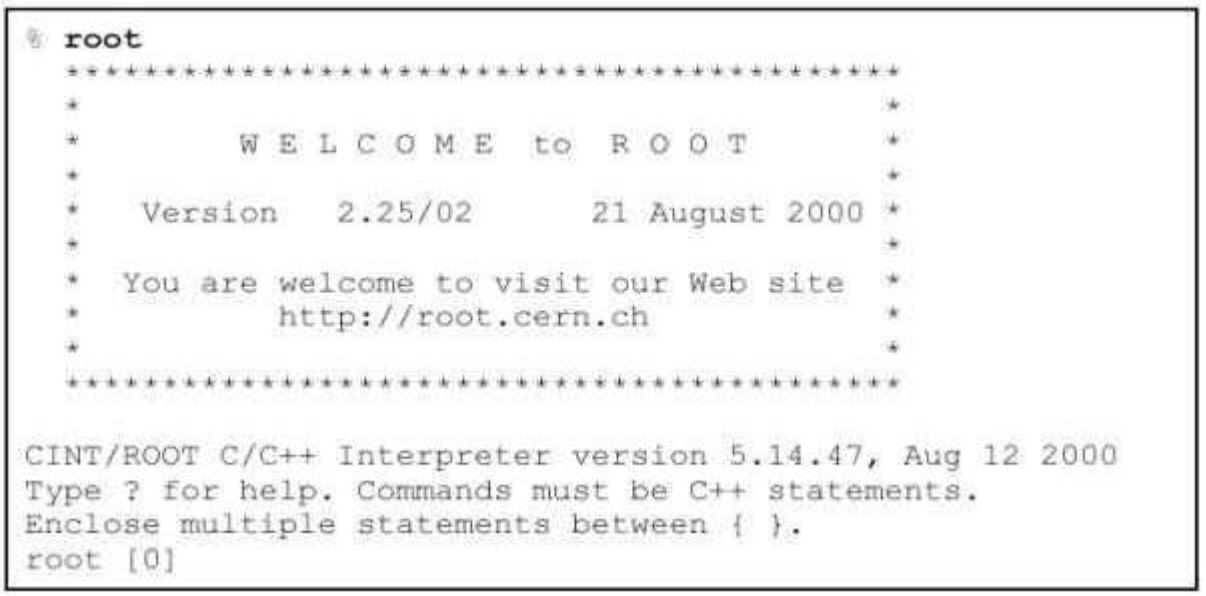
Introduzione a ROOT

Informazioni generali

- ROOT è un Framework sviluppato al CERN (vedi sito ufficiale http://root.cern.ch) interamente sviluppato in linguaggio C++
- fornisce un insieme di classi che principalmente servono per l'analisi statistica dei dati
- mette a disposizione un ambiente di sviluppo che si basa sull'interprete C++, CINT
- dal sito del CERN è possibile scaricare versioni di ROOT per vari sistemi operativi (come Linux e Windows32). La versione stabile più recente rilasciata è la 3.03
- nel sito è possibile trovare un'esauriente guida in formato pdf e la completa descrizione di tutte le classi di ROOT

Schermata iniziale

Digitando `root` al prompt dei comandi e premendo invio si avvia la sessione di ROOT.



Comandi di ROOT

Tutti i comandi sono preceduti da un punto (`"."`). Vediamone alcuni:

- `".q"` per uscire dalla sessione corrente di ROOT
- `".x nome-file.C"` (oppure `.cc`) per eseguire un programma scritto precedentemente. E' altresì possibile eseguire un programma all'avvio della sessione di ROOT digitando al prompt dei comandi

```
root nome-file.C
```

- `".!comando-shell"` per eseguire un comando della shell

  ES: `".!ls"` visualizza il listato dei file della directory corrente

- `".?"` per vedere la lista di tutti i comandi

CINT

- CINT è un interprete C++
- consente di eseguire programmi scritti in C++, interpretando ed eseguendo un'istruzione alla volta
- è competibile con quesi tutto il codice ANSI-C e ANSI-C++
- essendo un interprete è particolarmente utile nella fase di sviluppo delle applicazioni, comunque l'esecuzione dei programmi è più lenta rispetto a quella dei programmi compilati
- al prompt di ROOT è possibile digitare direttamente le istruzioni C++

```
ES: root[] cout << "Hello World" << endl;
    Hello World
    root[] int a = 10;
    root[] cout << a << endl;
    10
```

- per eseguire blocchi di istruzioni, si inizia digitando `{` e si finisce il blocco con `}`

```
ES: root[] {
     End with '}'> cout << "a" << endl;
     End with '}'> cout << "b" << endl;
```

```
End with '}'> }
a
b
```

- per eseguire un blocco di istruzioni scritte in un file (supponiamo che il file si chiami `esempio1.cc`):

```
{
#include <iostream.h>

   cout << " Hello" << endl;
   float x = 3.;
   float y = 5.;
   int   i = 101;
   cout <<" x = "<<x<<" y = "<<y<<" i = "<<i<< endl;
}
```

Per eseguirlo digitare

```
root[] .x esempio1.cc
```

(NOTA: il blocco di istruzioni scritto nel file è uno solo).

- per eseguire un programma scritto in un file (supponiamo che il file si chiami `script3.cc`):

```
#include <iostream.h>
int script3(int j = 10)
{
   cout << " Hello" << endl;
   float x = 3.;
   float y = 5.;
   int   i = j;
   cout <<" x = "<<x<<" y = "<<y<<" i = "<<i<<endl;
   return 0;
}
```

Per eseguirlo digitare

```
root[] .x script3.cc(4)
```

(NOTA: il nome della funzione principale deve essere lo stesso di quello del file).

La funzione `script3()` può essere eseguita nuovamente:

```
root [] script3()
 Hello
 x = 3 y = 5 i = 10
(int)0
root [] script3(33)
 Hello
 x = 3 y = 5 i = 33
(int)0
```

- CINT conosce tutte le funzioni, variabili e classi dichiarate. Consente in questo modo il completamento delle parole con il tasto <TAB>:

```
root [] l = new TL<TAB>
TLeaf
TLeafB
TLeafC
TLeafD
TLeafF
TLeafI
TLeafObject
TLeafS
TLine
TLatex
TLegendEntry
TLegend
TLink
TList
TListIter
TLazyMatrix
TLazyMatrixD
```

- Possibilità di visualizzare la dichiarazione di una classe (nell'esempio la classe `TLine`) e indirizzare l'uscita di un'istruzione su un file di testo:

```
root [] .class TLine
==========================================================
class TLine //A line segment
 size=0x28
List of base class-------------------------------
0x0        public: TObject //Basic ROOT object
0xc        public: TAttLine //Line attributes
List of member variable--------------------------
Defined in TLine
0x0        protected: Double_t fX1 //X of 1st point
0x0        protected: Double_t fY1 //Y of 1st point
0x0        protected: Double_t fX2 //X of 2nd point
0x0        protected: Double_t fY2 //Y of 2nd point
0x0        private: static class TClass* fgIsA
List of member function--------------------------
Defined in TLine
filename   line:size busy function type and name
(compiled) 0:0    0 public: class TLine TLine(void);
(compiled) 0:0    0 public: Double_t GetX1(void);
(compiled) 0:0    0 public: Double_t GetX2(void);
(compiled) 0:0    0 public: Double_t GetY1(void);
(compiled) 0:0    0 public: Double_t GetY2(void);
...
...
(compiled) 0:0 public: virtual void SetX1(Double_t x1);
(compiled) 0:0 public: virtual void SetX2(Double_t x2);
(compiled) 0:0 public: virtual void SetY1(Double_t y1);
(compiled) 0:0 public: virtual void SetY2(Double_t y2);
(compiled) 0:0    0 public: void ~TLine(void);
root [] l.Print(); > test.log
root [] l.Dump(); >> test.log
root [] ?
```

Convenzioni

- Le classi iniziano con `T`: `TTree`, `TBrowser`

- I tipi finiscono con `_t`: `Int_t`, `Double_t`
- Le costanti iniziano con la `k`: `kTRUE`, `kRed`
- Le variabili globali iniziano con la `g`: `gROOT`, `gSystem`
- Le funzioni membro iniziano con la lettera maiuscola: `Draw()`, `Fill()`
- Variabili e parametri iniziano con la lettera minuscola: `nLine`, `nBytes`

Variabili globali

- `gROOT`: puntatore all'oggetto `TROOT` della sessione corrente. Permette di accedere ad una serie di liste che puntano agli oggetti creati (es: lista delle finestre, lista dei file aperti) e consente di modificare le proprietà comuni a tutta la sessione (es: stile grafico)
- `gFile`: puntatore al file attualmente aperto (oggetto `TFile`)
- `gRandom`: puntatore al corrente generatore di numeri casuali. Di default punta ad un oggetto `TRandom` che genera numeri a partire da una distribuzione uniforme
- `gDirectory`: puntatore alla directory corrente della sessione di `ROOT` (classe `TDirectory`)

Classi, Metodi e Costruttori

- Esempi di dichiarazione:

1. ```
   TF1 *f = new TF1("f","sin(x)/x",0,10);
   ```
2. ```
   TF1 f  (“f”,”sin(x)/x”,0,10);
   ```

   Costruttore classe `TF1:`

   TF1 (nome, funzione, min, max)
   nome: stringa che identifica la funzione all'interno della sessione di ROOT
   funzione: stringa che descrive la funzione
   min e max: double che rappresentano l'intervallo di variabilità della funzione

- Per riferirsi ad un metodo dell'oggetto:

1. Se si usa un puntatore (caso 1):

   puntatore_oggetto`->`nome_metodo(parametri);

2. Se si usa una variabile normale (caso 2):

   oggetto`.` nome_metodo(parametri);

- Esempi:

1. ```
   f->Draw(); // f puntatore all'oggetto
   ```
2. ```
   f.Draw(); // f variabile normale
   ```

Disegnare Funzioni Matematiche

La classe TF1

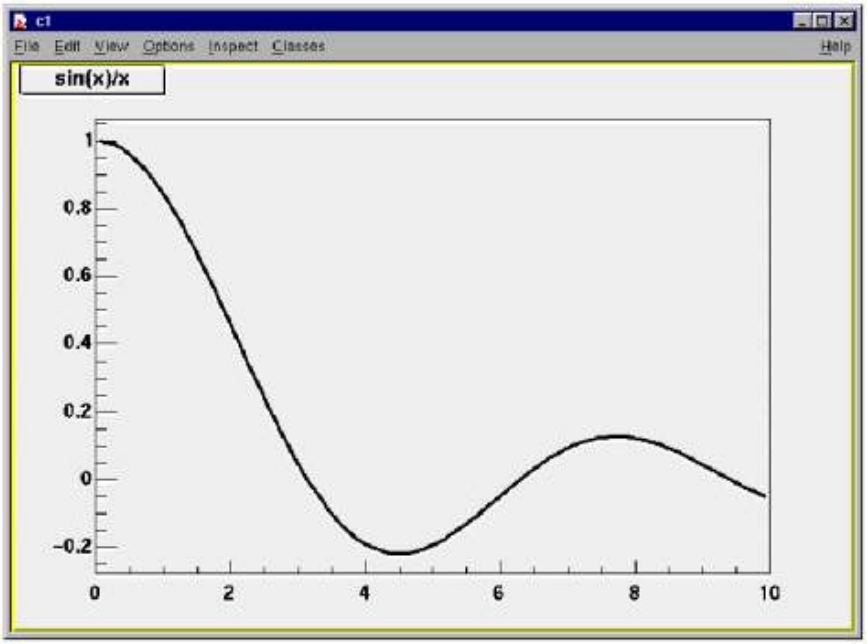
Per disegnare funzioni matematiche ad una sola variabile si fa uso della classe TF1 (vedi esempio costruttore sopra).

Esempio introduttivo

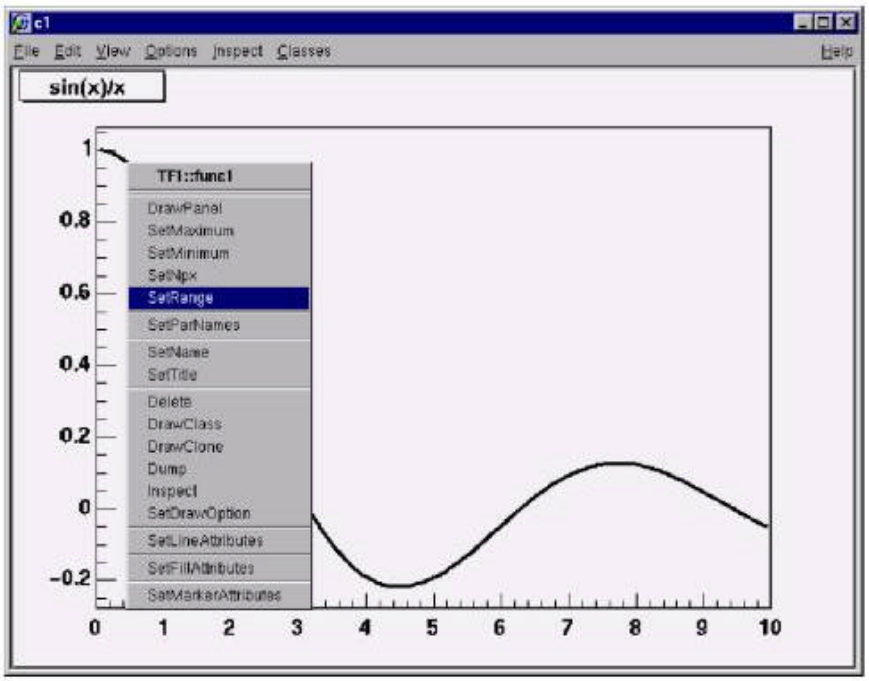
- Esempio per disegnare una funzione:

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

Disegna la funzione sin(x)/x con x che varia tra 0 e 10. L'uscita grafica dovrebbe essere:



- E' possibile modificare l'aspetto grafico (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. Per esempio selezionare `SetRange`, che serve per modificare il range della funzione (che supponiamo voler modificare in `[-10, +10]`), equivale a scrivere l'istruzione `f1.SetRange(-10,10)`

Calcolare, derivare ed integrare

```
root [] f1.Eval(3)
(Double_t)4.70400026866224020e-02
root [] f1.Derivative(3)
(Double_t)(-3.4567505671992330e-01)
root [] f1.Integral(0,3)
(Double_t)1.84865252799946810e+00
root [] f1.Draw()
```

Programma di esempio per l'impiego delle funzioni

- Disegniamo le seguenti funzioni:
  - Poisson
  - Gaussiana
  - Doppia gaussiana
- Classi usate
  - `TCanvas`: classe per la gestione delle finestre
  - `TGraph`: classe per disegnare funzioni per punti
  - `TF1`: classe per la gestione delle funzioni
  - `TH2F`: classe per la gestione degli istogrammi bidimensionale
  - `TMath`: classe con funzioni matematiche
  - `TLegend`: classe per la gestione delle legenda
- Programma da scaricare: funzioni.cc

Alfio Lazzaro, 2/12/2002