Corso di Laboratorio 2 – Programmazione C++

Silvia Arcelli

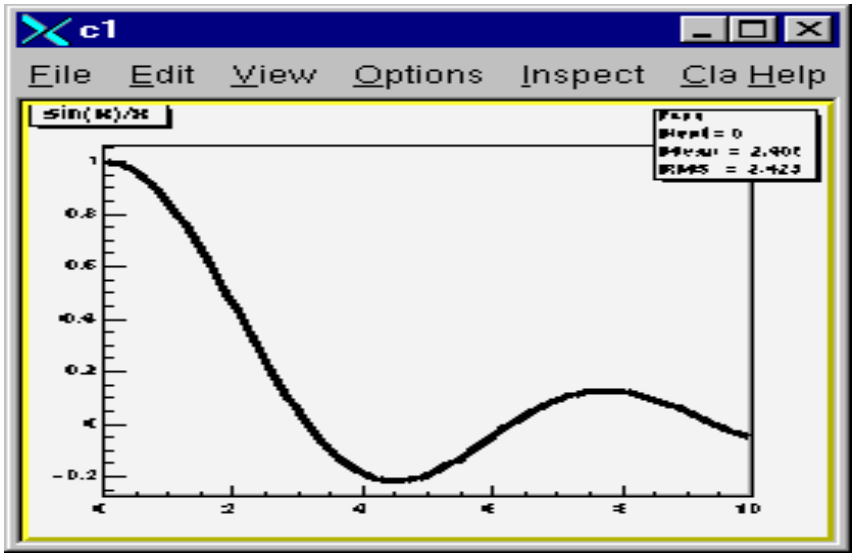
10 Novembre 2014

Funzioni in ROOT

- Function Objects (TF1)
  - Tre tipi di costruttori per TF1
  - Sono disponibili anche dei “built in” function objects (gaus, expo, poln,…), vedere in: http://root.cern.ch/root/html/TFormula.html
  - Funzioni definite dall’utente
  - Corrispondenti in 2 e 3 dimensioni: TF2, TF3

Funzioni in ROOT-TF1

1. Consente di rappresentare una espressione C++ -like in cui la variabile è x




```
TF1 *f1 = new TF1("f1","sin(x)/x",0,10);
f1->Draw(); // disegna la funzione
TF1 *f2 = new TF1("f2","f1 * 2",0,10);// potete
  usare funzioni precedentemente definite
```

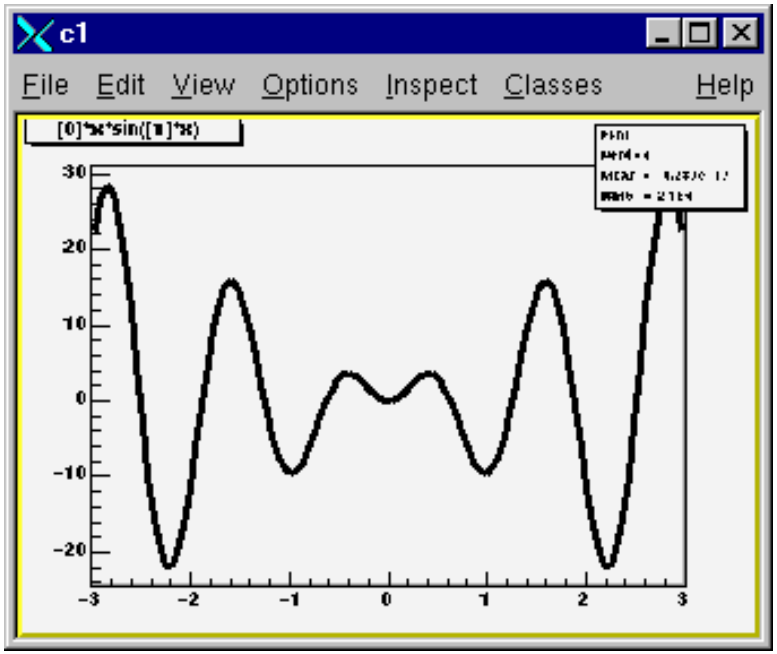
Funzioni in ROOT-TF1

2. Come prima, ma con parametri:

```
TF1 *f1 = new TF1("f1","[0]*x*sin([1]*x)",-3,3);
```

Per inizializzare i parametri:

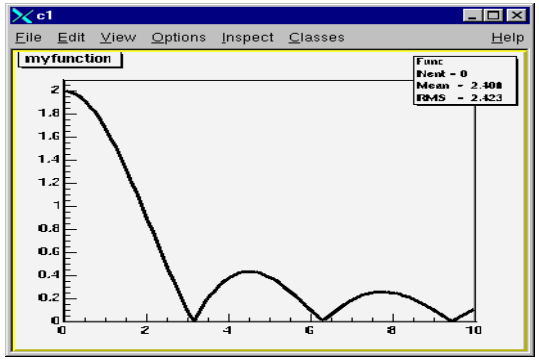
```
f1->SetParameter(0,10);
f1->SetParameter(1,5);

f1->Draw(); // disegna f1
```



Funzioni in ROOT-TF1

3. Funzione definita dall'utente (in una macro ad esempio):

```
Double_t MyFunction(Double_t *x, Double_t *par){
 Float_t  xx     =  x[0];
 Double_t val    =
    TMath::Abs(par[0]*sin(par[1]*xx)/xx);
 return val;
}
```



TF1 Constructor:

```
TF1 *f1 = new TF1("f1",MyFunction,0,10,2);
```

(**NOTA: il 2 indica il numero di parametri in MyFunction.**)

Setting dei parametri:

```
f1->SetParameters(2,1); // inizializza i due parametri a
2 e 1
```

Esempi

Macro che grafica due funzioni definite dall'utente(1D e2D)

makeFunctions.C e myFunctions.C

Fitting in ROOT

- Fitting
  - Fit con una funzione definita dall'utente
  - Fitting in range separati e sottorange, combinando funzioni

Per fare un fit di un istogramma con la funzione scelta:

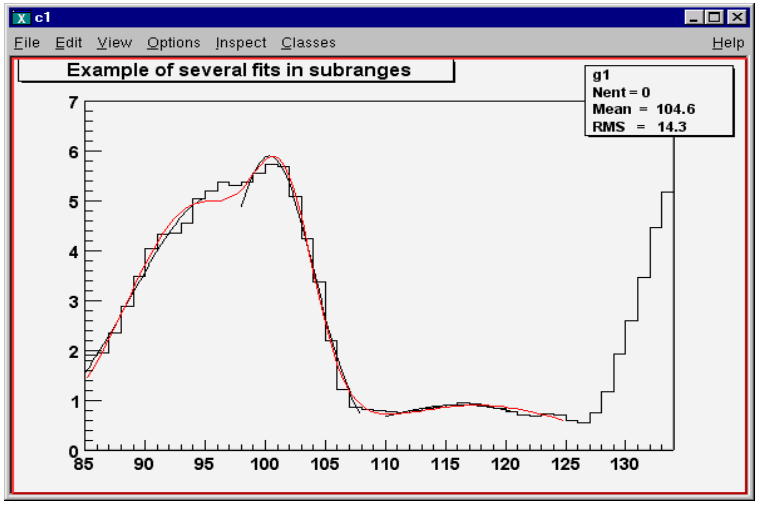
```
TF1 *fn1 = new TF1("f1","[0] *x*sin([1]*x)",-3,3);
f1->SetParameters(10,5); // inizializzo i due parametri a 10, 5

aHistogram->Fit("f1"); //con il nome della funzione
aHistogram->Fit(fn1); // o con il puntatore
```

Fitting in Sotto-Range

- Definire il range nel TF1 constructor.

```
TF1 *g1 = new TF1("g1", "gaus", 85,95);
```

- Per default TH1::Fit() fitta sull'intero range dell'istogramma h. Per fittare nel sotto-range, usare l'opzione "R" nella chiamata a Fit(), `h->Fit("g1", "R");`



Esempio:

Fare un fit con tre gaussiane di questo istogramma, nel range [85 ,125]

Fitting in Sotto-Range

1. Definire tre gaussiane e una funzione somma (total):

```
TF1 * g1    = new TF1("g1","gaus",85,95);
TF1 * g2    = new TF1("g2","gaus",98,108);
TF1 * g3    = new TF1("g3","gaus",110,121);
TF1 * total = new
    TF1("total","gaus(0)+gaus(3)+gaus(6)",85,125);
```

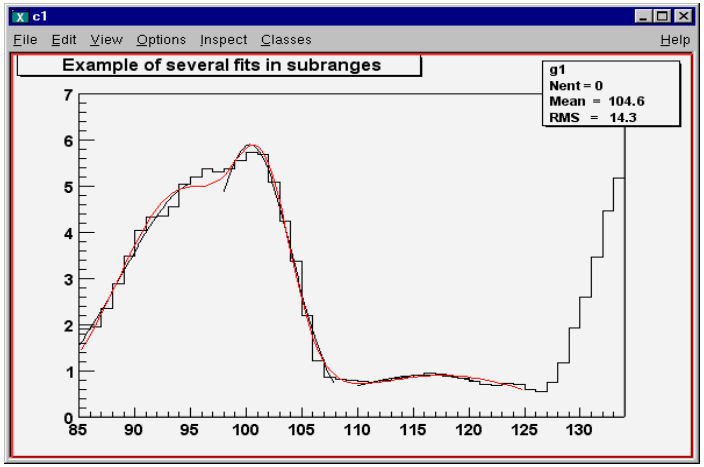
NOTA: gaus(0) è equivalente a : [0]*exp(-0.5*((x-[1])/[2])**2)

e (0) indica che l'indicizzazione dei parametri parte da 0

Fitting in Sotto-Range

2. Fittare in ogni sotto-range e mettere i parametri risultanti in “total”

```
h->Fit(g1,"R");
h->Fit(g2,"R");
h->Fit(g3,"R");
g1->GetParameters(&par[0]);
g2->GetParameters(&par[3]);
g3->GetParameters(&par[6]);
total->SetParameters(par);
h->Fit(total,"R")
```



Risultato del fit: usare I metodi GetParameters e GetChisquare “recuperando” la funzione di fit dall’istogramma:
TF1 *fitFunc = h->GetFunction(“total"). Disponibile anche informazione più dettagliata (matrice di covarianza)

Esempi

Macro che fitta un istogramma con una componente di segnale (gaussiano) e una di fondo (esponenziale):

fitSignal.C

Generazione di numeri casuali

Numeri casuali: valore assunto da una variabile aleatoria, il cui valore è per definizione impredicibile sulla base delle occorrenze passate.

•Alla base del metodo Monte Carlo ci sono sequenze di numeri casuali, "difficilmente" ottenibili nel concreto (processi fisici: lancio di una moneta, tempi di arrivo dei raggi cosmici,..)

•Per questioni di efficienza, i generatori di Monte Carlo si basano su sequenze di numeri Pseudo-Casuali (PRNG) che hanno alla base un algoritmo matematico, quindi teoricamente perfettamente deterministiche e predicibili:

$$x_{n+1} = f(x_n)$$

Generazione di numeri casuali

Generatori di numeri pseudo casuali (PRNG) sul mercato basati su diversi algoritmi. Generalmente producono sequenze di interi uniformemente distribuiti o numeri reali (sempre uniformemente distribuiti) in [0,1]. Gli algoritmi disponibili si possono qualificare:

- in termini di prestazioni: ad esempio la loro velocità, o in termini del periodo, ovvero dopo quante iterazioni si ricomincia a ripercorrere nella sequenza stati già generati

Generazione di distribuzioni non uniformi

•Gli algoritmi di generazione delle sequenze quindi producono numeri secondo una distribuzione uniforme in un certo intervallo. E se si vogliono generare numeri distribuiti secondo altre pdf (pdf= probability density function)?

•Numeri casuali di distribuzioni non uniformi possono essere ottenuti a partire da numeri casuali uniformemente distribuiti applicando alcuni metodi di trasformazione. Due classi di metodi:

1. Metodo del rigetto (hit-or-miss)

2. Metodo della trasformazione inversa

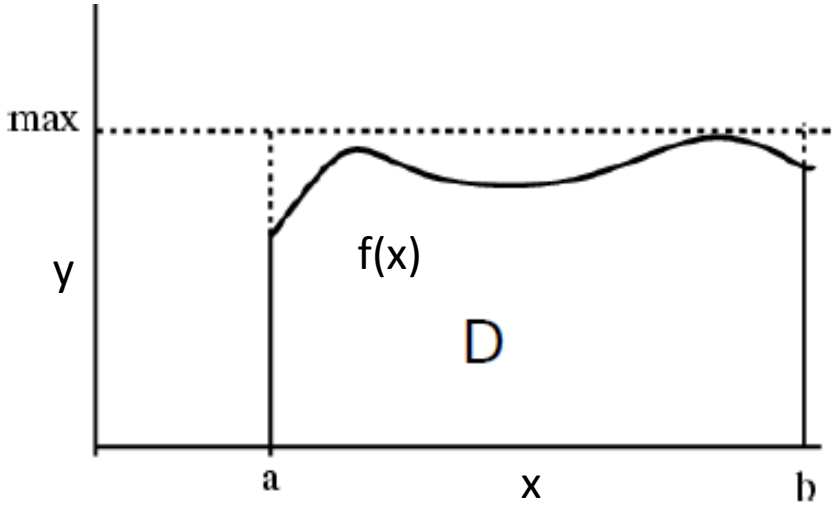
Metodo del Rigetto (von Neumann, 1951):

Supponete di voler generare random una variabile x distribuita secondo una distribuzione di probabilità f(x). Si può usare un metodo del tipo *"hit or miss"* (funziona sempre, anche se può essere molto inefficiente a seconda del tipo di funzione f(x)).

•Generate due variabili casuali x,y distribuite uniformemente negli opportuni domini, utilizzando il vostro PNRG:

$$x = U(a,b),$$
$$y = U(0,max)$$

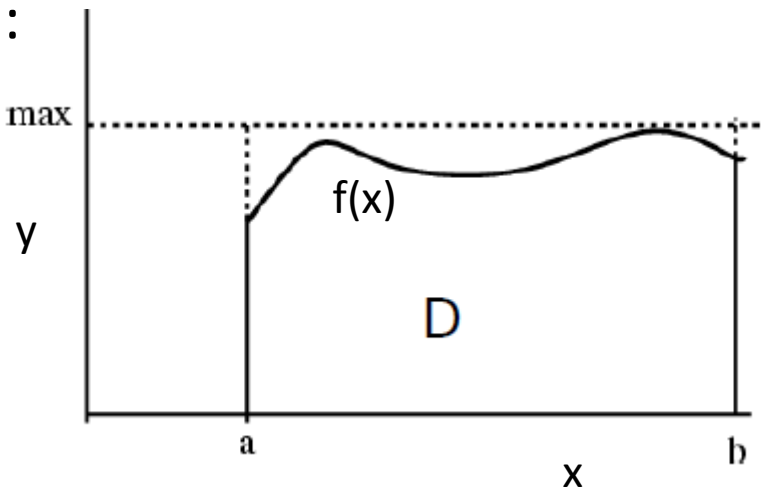
$$max = max(f(x)) \ in \ [a,b]$$



Metodo del Rigetto

•Definite una funzione h(x,y) tale per cui:

$$h(x,y) = 0 \quad \text{se} \quad y > f(x)$$
$$h(x,y) = 1 \quad \text{se} \quad y < f(x)$$



hit or miss:

•Se h(x,y) = 0 scartate l'estrazione

• se h(x,y)=1 accettate l'estrazione

$$x = U(a,b),$$
$$y = U(0,\max)$$

In questo modo la variabile x risulterà distribuita secondo f(x).

Metodo del Rigetto

per aumentare l'efficienza del metodo:

a) determinate un'altra PDF *g(x)* per la quale la generazione di punti casuali è semplice e veloce, e tale che:

$$c \cdot g(x) \geq f(x)$$

g(x) "copre" f(x), modulo un fattore di scala

b) Generate punti x secondo la g(x) e un numero casuale $y \in U(0,1)$

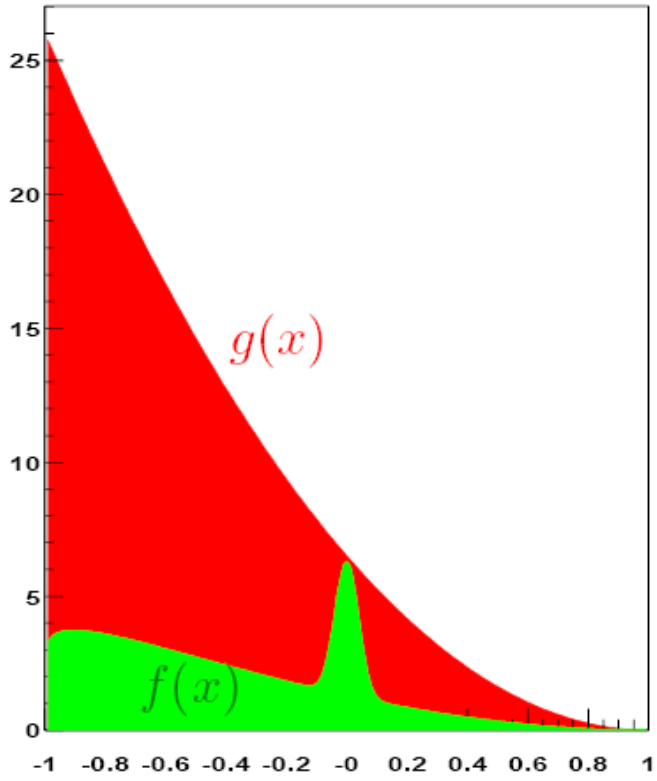
c) Se $y \cdot c \cdot g(x) < f(x)$ l'estrazione viene accettata, altrimenti viene rigettata e si reitera il punto b).

In sostanza, "ridefinite" il max del caso precedente per diminuire il *rate* (frequenza) di rigetto

Limitazioni del Metodo del Rigetto

-in linea di principio, è applicabile solo a pdf definite su un dominio finito.

- E' in generale un metodo poco efficiente, a seconda della forma della distribuzione



Presenza di un picco nella f(x)
→ alto rate di rigetto

Metodo della Trasformazione Inversa:

Se la funzione f(x) che definisce la pdf è integrabile e la sua distribuzione cumulativa F(x) è invertibile è molto più efficiente usare il metodo della *Trasformazione (o ripartizione) inversa*:

$$\int_a^b f(x)dx = 1 \qquad P(x < X) = F(X) = \int_a^X f(x)dx$$

Il metodo si basa sul fatto che la variabile y definita come:

$$y = F^{-1}(U) \qquad \text{dove:} \qquad U \in U(0,1)$$

è distributa secondo f(X)

Metodo della Trasformazione Inversa, dimostrazione:

Cosa vogliamo dimostrare:

$$P(y < X) = P(F^{-1}(U) < X) = F(X)$$

Dim:

$$P(F^{-1}(U) < X) = P(F(F^{-1}(U)) < F(X)) = P(U < F(X)) = F(X)$$

Applico F

U è U(0,1)

Quindi, usando questa proprietà genero la variabile aleatoria y attraverso la funzione cumulativa di probabilità F, e automaticamente y risulta distribuita secondo la pdf f

Metodo della Trasformazione Inversa:

Nel caso dell'esponenziale:

$$f(x) = ke^{-kx} \qquad y = F(x) = \int_0^x f(y)dy = 1 - e^{-kx}$$

Generate y in U(0,1) e invertite per avere x:

$$x = F^{-1}(y) = -\frac{\ln(1-y)}{k}$$

x sarà distribuito secondo una pdf di tipo esponenziale

Metodo della Trasformazione Inversa:

Nel caso della gaussiana unitaria di media 0, N(0,1):

$$f(x) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2}}$$

La pdf gaussiana 1-D non è integrabile, ma in due dimensioni lo è:

$$\iint_{x,y} \frac{1}{2\pi} \cdot e^{-\frac{x^2+y^2}{2}} dxdy = \iint_{r,\varphi} \frac{1}{2\pi} \cdot e^{-\frac{r^2}{2}} rdrd\varphi = 1$$

E la cumulativa ha la forma:

$$F(\frac{R^2}{2}, \phi) = (1 - e^{-\frac{R^2}{2}}).\frac{1}{2\pi}\phi$$

Prodotto di due cumulative indipendenti, una di una distribuzione esponenziale e una di una distribuzione uniforme

Metodo della Trasformazione Inversa:

Per cui se genero due numeri random n1 e n2 e determino x e y come:

(Metodo diBox-Muller)

$$x = \sqrt{-2\pi \ln(n_1)}\cos(2\pi n_2),\ \ y = \sqrt{-2\pi \ln(n_1)}\sin(2\pi n_2)$$

avrò che x,y sono distribuite secondo una PDF gaussiana 2-D.

•Per ottenere una gaussiana 1-D, essendo x,y indipendenti, basta "ignorare" una delle variabili.

•Per poi generare secondo una gaussiana generica N(μ,σ), è sufficiente fare una trasformazione del tipo: $x' = \sigma \cdot x + \mu$

Limitazioni del metodo di trasformazione inversa:

- Solitamente richiede che la funzione di distribuzione cumulativa sia conosciuta analiticamente e che sia invertibile analiticamente. Solo un ristretto numero di funzioni soddisfano queste richieste.

- In teoria è possibile integrare numericamente la funzione densità di probabilità e tabulare (ad esempio in un istogramma) la funzione di distribuzione cumulativa. In questo modo è possibile invertire la funzione di distribuzione cumulativa numericamente. Questo procedimento è però spesso molto lento

PRNG in ROOT

- Classe TRandom:
  - LCG ($P \approx 10^9$)
  - Velocità: 34 ns/chiamata
- Classe TRandom1:
  - Algoritmo RANLUX ($P \approx 10^{171}$)
  - Velocità: 242 ns/chiamata
- Classe TRandom2:
  - Algoritmo Tausworthe ($P \approx 10^{26}$)
  - Velocità: 37 ns/chiamata
- Classe TRandom3: (default)
  - Algoritmo Mersenne Twister ($P \approx 10^{6000}$)
  - Velocità: 45 ns/chiamata

- L'inizializzazione viene effettuata col metodo SetSeed()

- La generazione di numeri U(0,1) viene effettuata con Rndm()

PRNG in ROOT: distribuzioni generiche

- Le classi TRandom forniscono vari metodi per generare secondo distribuzioni generiche:
  - Le seguenti funzioni sono generate esplicitamente:
    - Exp(tau)
    - Integer(imax)
    - Gaus(media,sigma)
    - Landau(moda,sigma)
    - Poisson(media)
    - Binomial(ntot,prob)
  - In generale si può generare da qualunque funzione analitica:

    Esempio:

    ```
    TF1 *f1 = new TF1("f1","abs(sin(x)/x)*sqrt(x)",0,10);
    double r = f1->GetRandom();
    ```
  - Si può anche riempire direttamente un istogramma:
    - Histo->FillRandom(“f1”,n);