Laboratorio di Elettromagnetismo e Ottica– C++/ROOT
Lezione VI

Silvia Arcelli

Generazione Monte Carlo in ROOT

ROOT ci offre diverse modalità di generare numeri casuali secondo definite distribuzioni. I principali:

– **TH1::FillRandom(const char*f, Int_t N) : riempire istogramma con occorrenze generate secondo la funzione f ("implicito")**

```
TF1 *f1 = new TF1("f1","abs(sin(x)/x)*sqrt(x)",0,10);
histo->FillRandom("f1",n); //Histo conterrà n estrazioni di una variabile distribuita secondo la pdf definita da f1
```

–**TF1::GetRandom(): estrazioni casuali secondo la funzione TF1, ritorna la singola estrazione ("esplicito")**

```
double r = f1->GetRandom(); //r sarà una variabile distribuita secondo la pdf definita da f1
```

–**TRandom Class via methods, ritorna la singola estrazione ("esplicito")**

PRNG in ROOT: distribuzioni generiche

- Le classi TRandom forniscono vari metodi per generare secondo distribuzioni generiche:
  - Le seguenti funzioni sono ad es. generate esplicitamente:
    - **Exp(tau)**
    - **Integer(imax)**
    - **Uniform(a,b)**
    - **Gaus(media,sigma)**
    - **Landau(moda,sigma)**
    - **Poisson(media)**
    - **Binomial(ntot,prob)**

  Disponibili attraverso il puntatore globale **gRandom**

Controllo del seme di Generazione

- Effettuando una generazione Monte Carlo occorre tenere anche in conto il terzo importante parametro di un algoritmo PRNG**: il seme (seed).**

- gRandom nasce con un seed iniziale di default. Potete verificarlo in una qualunque macro facendo una stampa a schermo (prima di avviare ogni generazione) di **gRandom->GetSeed()**.

- Per essere sicuri ogni volta di fare generazioni indipendenti, prima e al di fuori del ciclo di generazione (una sola volta!) inserire la chiamata:

gRandom->SetSeed()

Generare secondo definite proporzioni

- **Si utilizza la proprietà della distribuzione uniforme**: la probabilità della variabile di avere un'occorrenza in un intervallo è **proporzionale alla dimensione dell'intervallo stesso. Il flusso del codice è quindi:**
  - Estrazione di un numero random uniformemente distribuito fra 0 e 1, con gRandom->Rndm()
  - For & if-else if-else block structure con condizioni sulle percentuali di popolazione

Generare secondo definite proporzioni

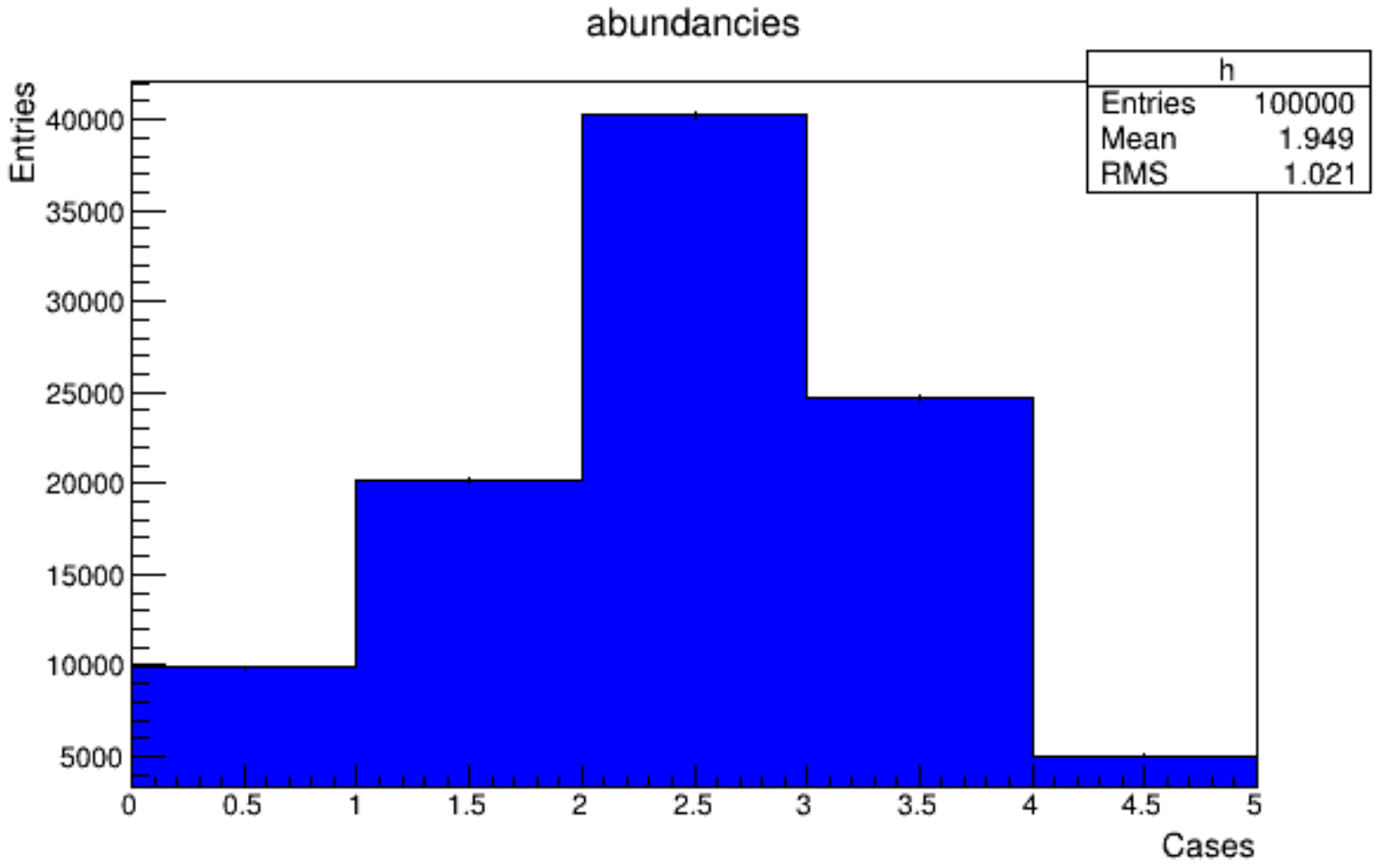
- Esempio: generare una popolazione di 1.E5 soggetti, divisa in 5 categorie, con rispettive probabilità di occorrenza: (10%, 20%,40%,25%,5%). Inserire in un istogramma le occorrenze per verificare che siano state generate in maniera opportuna.

```
void proportions(Int_t nGen){

TH1F *h =new TH1F("h","abundancies",5,0,5);
Double_t x=0;
for(Int_t i=0;i<nGen;i++){
  x=gRandom->Rndm(); // the uniform random number in [0,1]
  if(x<0.1)h->Fill(0); //10%
  else if(x<0.3)h->Fill(1); //20%
  else if(x<0.7)h->Fill(2);//40%
  else if(x<0.95)h->Fill(3);//25%
  else h->Fill(4);//5%
}
TCanvas *c1 = new TCanvas("c1","Generate proportions,
Example",200,10,600,400);
h->Draw("H");
h->Draw("E,SAME");
}
```

Macro proportions.C

Generare secondo definite proporzioni



Generare secondo definite proporzioni

- Modificare la macro precedente per generare una popolazione di 1.E5 soggetti, divisa in 4 categorie, con rispettive probabilità di occorrenza: (20%, 15%,30%,35%).

Generare variabili indipendenti

Altra cosa da tener presente:

–Dovendo generare più di una variabile (assunte per ipotesi indipendenti), alla generazione di ciascuna variabile **deve corrispondere un'estrazione indipendente** di un numero random. Altrimenti si introducono correlazioni fittizie.

–ES: popolazione di punti nella spazio con distribuzione uniforme in angolo polare θ in $[0,\pi]$ e angolo azimutale φ in $[0,2\pi]$

```
TH2F *h =new TH2F("h","",1000,0,TMath::Pi(),1000,0,2*TMath::Pi());

Double_t phi,theta=0;
for(Int_t i=0;i<nGen;i++){
   theta=gRandom->Rndm()*TMath::Pi();
   phi= gRandom->Rndm()*2*TMath::Pi();
   h->Fill(theta,phi);
}
TCanvas *c1 = new TCanvas("c1","theta-phi
correlation",200,10,600,400);
h->Draw();

}
```

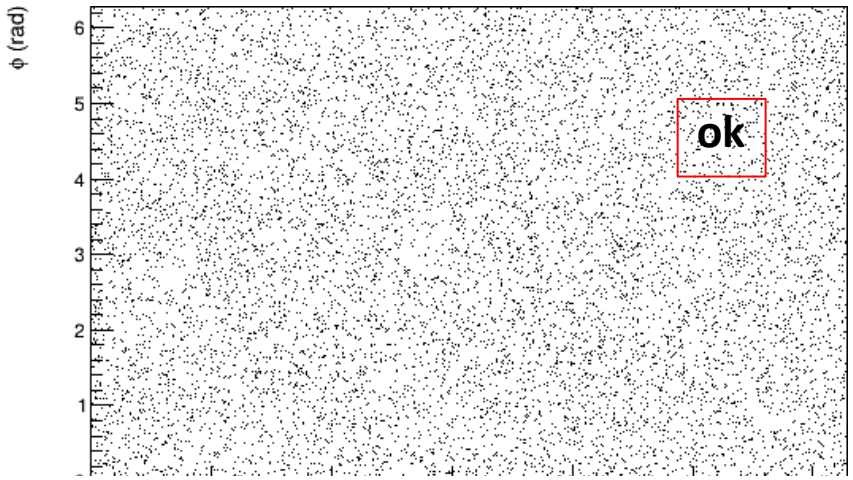
ok

Macro testCorrelation.C

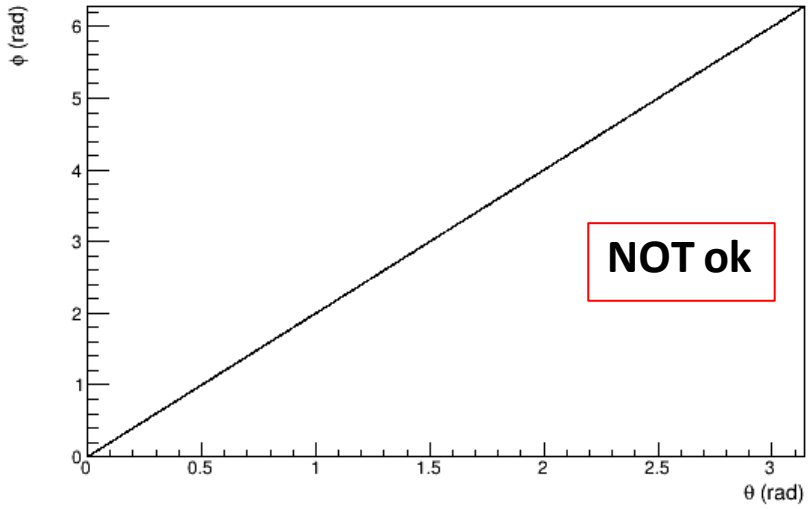
```
Double_t phi,theta=0;
for(Int_t i=0;i<nGen;i++){
   theta=gRandom->Rndm()*TMath::Pi();
   phi= 2*theta;
   h->Fill(theta,phi);
}
```

NOT ok

Generare variabili indipendenti



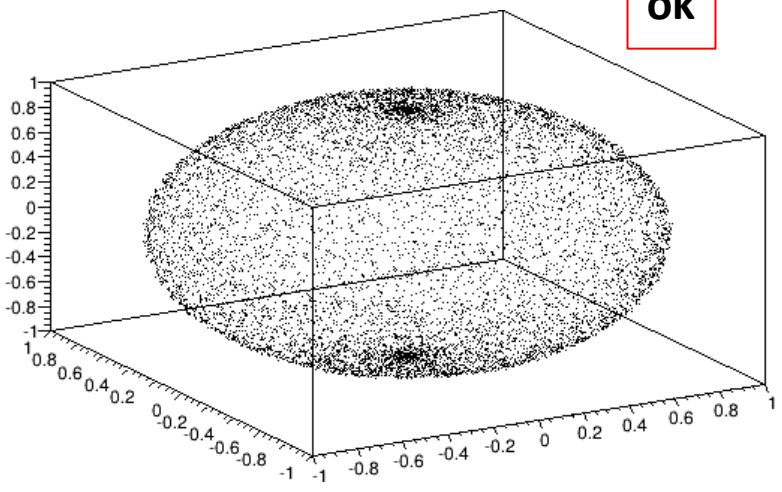
Distribuzione degli angoli corretta (non correlati)



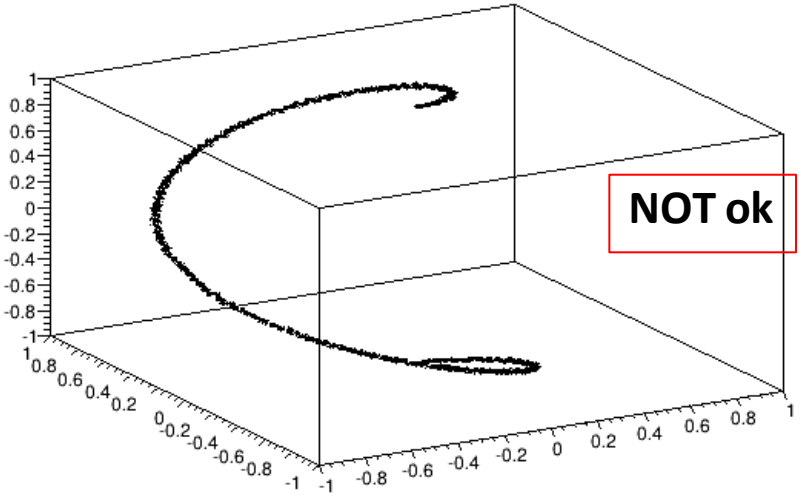
Theta e phi totalmente correlati

Generare variabili indipendenti

ok



In 3D,
Distribuzione nello spazio



NOT ok

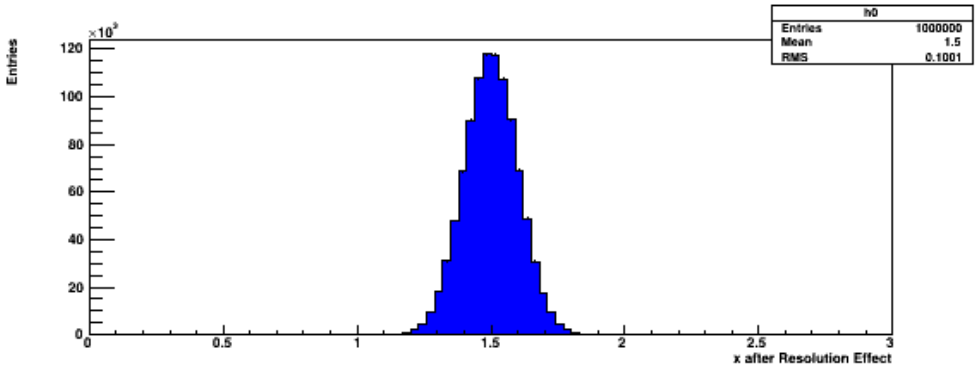
Come simulare un effetto di risoluzione

Caso tipico: effettuare uno "***smearing***", assumendo un effetto di risoluzione di tipo gaussiano, sulla variabile generata secondo una certa distribuzione a priori.

Esempio con due diverse distribuzione «a priori»:

- **Valore fisso** (fixedValue=1.5)
- **Distribuzione uniforme U(1,2)** (con gRandom->Uniform(a,b))

Come simulare un effetto di risoluzione

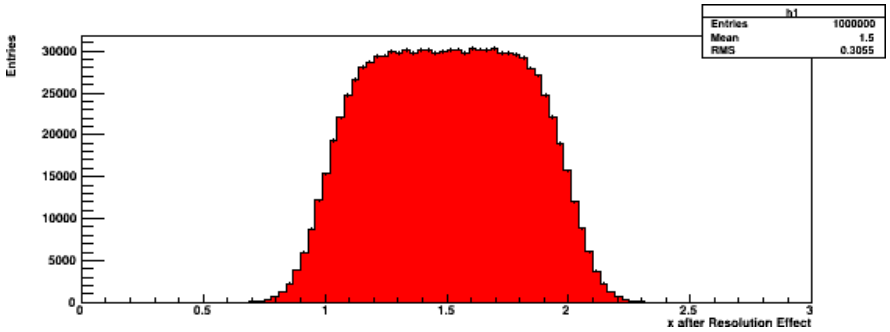



```
{
Double_t res=0.1, Int_t nGen=1E6

//first case: fixed value smeared
Double_t fixedValue =1.5;
for( Int_t i=0;i<nGen;i++)h[0]->Fill(gRandom->Gaus(fixedValue,res) );

//second case: Uniform distribution smeared
for( Int_t i=0;i<nGen;i++)h[1]->Fill(gRandom->Gaus(gRandom->Uniform(1,2),
res) );

}
```



Macro resolution.C

Come simulare un effetto di risoluzione

- Considerazioni sulla forma, sulla media e sulla RMS delle distribuzioni osservate
- Modificare il valore di default del primo parametro (la risoluzione sperimentale dell'apparato) e settarlo per esempio a 0.01, e quindi a 0.3

Come simulare un'efficienza

Predefinendo il profilo di efficienza secondo una funzione scelta, e applicando "hit or miss". Esempio: **Distribuzione Gaussiana** G(2.5,1) nell'intervallo [0,5]con un'efficienza costante di osservare l'estrazione pari al 70% ($\varepsilon(x)=70\%$, costante)

```
{
TH1F *hAcc=new TH1F("hAcc", " " , 500,0,5); hAcc->Sumw2();
TH1F *hGen=new TH1F("hGen", " " , 500,0,5); hGen->Sumw2();
Float_t x,y;
for(Int_t i=0;i<1.E6;i++){
  x=gRandom->Gaus(2.5,1); //the variable x
  y=gRandom->Rndm(); //generate a uniform random number in [0,1]
  if(y<0.7)hAcc->Fill(x);//fill the histo with accepted entries
  hGen->Fill(x); //fill the histo with generated entries
}
TH1F *hEff=new TH1F(*hGen);
hEff->Divide(hAcc,hGen,1,1,"B"); //the efficiency histogram
}
```

Istogrammi-divisione:

Per dividere i due istogrammi abbiamo utilizzato:

- Il Copy Constructor di TH1F: TH1F::TH1F (const TH1F & *h1*)
- Il Metodo Divide, virtual Bool_t Divide (const TH1 *h1, const TH1 *h2, Double_t c1=1, Double_t c2=1, Option_t *option="") : Replace contents of this histogram by the division of h1 by h2: this= c1*h1/c2*h2
- Siamo nel caso di applicare una statistica di tipo binomiale (il numeratore è sottoinsieme del numeratore, k successi su n prove, in ogni bin)
- Per il calcolo degli errori opzione "B". Prima di riempire entrambi gli istogrammi, invocare il metodo TH1F::Sumw2(). Invocare questo metodo è opportuno ogni qualvolta si facciano operazioni su istogrammi, per una corretta valutazione delle incertezze sui contenuti dei bin.

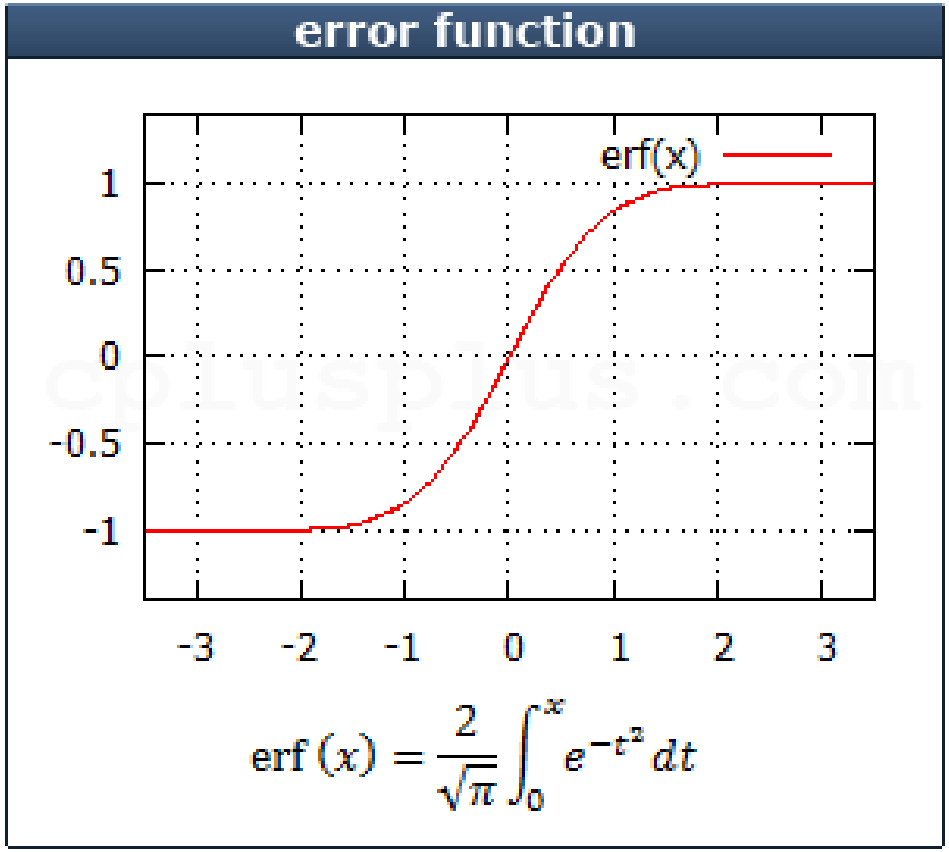
Caso con efficienza dipendente da x

Distribuzione Gaussiana G(2.5,1) con un'efficienza di osservare l'estrazione che è funzione di x secondo **$\varepsilon(x)=0.2\,x$**

```
{
TF1 *eff=new TF1("eff","0.2*x",0,5);
TH1F *hAcc=new TH1F("hAcc", " " , 500,0,5);
TH1F *hGen=new TH1F("hGen", " " , 500,0,5);
Float_t x,y;
for(Int_t i=0;i<1.E6;i++){
  x=gRandom->Gaus(2.5,1); //the variable x
  y=gRandom->Rndm(); //generate a uniform random number in [0,1]
  if(y<eff->Eval(x))hAcc->Fill(x);//fill the histo with accepted entries
  hGen->Fill(x); //fill the histo with generated entries
}
TH1F *hEff=new TH1F(*hGen);
hEff->Divide(hAcc,hGen,1,1,"B"); //the efficiency histogram
}
```

```
hAcc->Sumw2();
hGen->Sumw2();
```

Caso con efficienza descritta da TMath::Erf(x)



Erf(x) («funzione degli errori») è connessa alla distribuzione cumulativa Φ(x) della gaussiana G(0,1)

$$\Phi(x) = \frac{1}{2}\left(1 + erf\left(\frac{x}{\sqrt{2}}\right)\right)$$

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt$$

Molto comunemente usata nel simulare profili di efficienza "a soglia"

Caso con efficienza descritta da TMath::Erf(x)

Macro efficiency.C

- **Distribuzione Uniforme della variabile,** con profilo di efficienza **Erf(x)**
  - Disegnare distribuzione generata e distribuzione osservata
  - Fare il rapporto fra generata e osservata
  - Valutare l'efficienza integrale e l'efficienza differenziale
- Stessa prova con **Distribuzione Esponenziale Decrescente (μ=1),** sempre con profilo di efficienza Erf(x)

Notate che **l'efficienza differenziale**:

$$\varepsilon_{diff}(x) = \frac{Nacc(x)}{Ngen(x)}$$

(quella in funzione della variabile) **non dipende** dalla distribuzione a priori, mentre **quella integrale**:

$$\varepsilon_{int} = \frac{Nacc(tot)}{Ngen(tot)}$$

Dipende fortemente dalla distribuzione a priori.

Benchmarking Code

TBenchmark: Utility di ROOT per valutare la performance in termini di tempo di esecuzione di una macro/programma:

- virtual void **Start** (const char *name): Starts Benchmark with the specified name
- virtual void **Stop** (const char *name): Terminates Benchmark with specified name.
- virtual void TBenchmark::**Print** (Option_t * *name* = "") const : Prints parameters of Benchmark name.

- virtual void **Show** (const char *name) : Stops Benchmark name and Prints results (=**Stop+Print**)

Benchmarking Code

Esempio di generazione di una gaussiana G(0,1) con:

- il metodo TH1::FillRandom("gaus") (implicito, trasformazione inversa numerica)
- il metodo TRandom::Gaus(…) (esplicito, trasformazione inversa esatta)

Fare 10^8 estrazioni $\rightarrow$ confronto tempi di CPU.

Benchmarking Code

Macro benchmark.C

```
void benchmark(Int_t nGen){
char *histName=new char[10]; TH1F *h[2];
for(Int_t i=0;i<2;i++){
sprintf(histName,"h%d",i);
h[i] =new TH1F(histName,"test histogram",100,0,10.);
}

//filling the histo
gBenchmark->Start("With TH1::FillRandom");

//with numerical inverse transform
h[0]->FillRandom("gaus",nGen);
//stop and show benchmark
gBenchmark->Show("With TH1::FillRandom");

//with pure inverse transform
Double_t x=0;
gBenchmark->Start("Direct TRandom::Gaus invocation");
for (Int_t i =0;i<nGen;i++){
  x=gRandom->Gaus(0,1);
  h[1]->Fill(x);
}
//Stop and show benchmark
gBenchmark->Show("Direct TRandom::Gaus invocation");

}
```