Algoritmi in C++
(prima parte)

Alcuni algoritmi in C++

- Far risolvere al calcolatore, in modo approssimato, problemi analitici
- Diverse tipologie di problemi
  - generazione di sequenze di numeri casuali
  - ricerca degli zeri di una funzione
  - integrazione numerica di una funzione
  - ricerca degli estremanti di una funzione
- Si lavora in condizioni semplici
- Scrivere gli algoritmi in modo più comodo possibile dal punto di vista del programma
- Qualche funzionalità di ROOT (librerie di analisi dati)

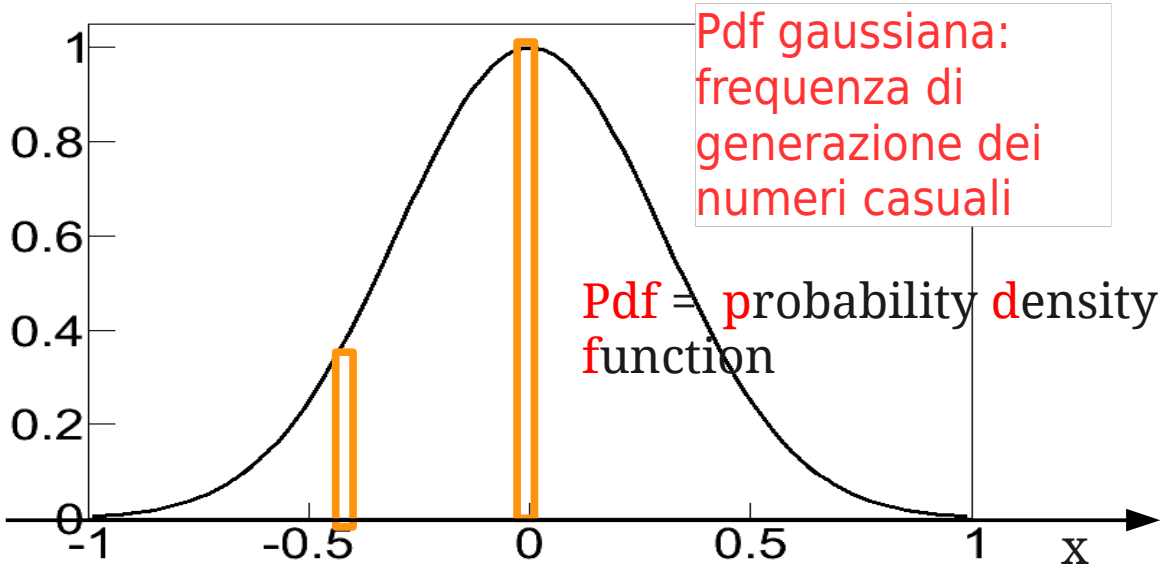
generazione di sequenze di numeri casuali

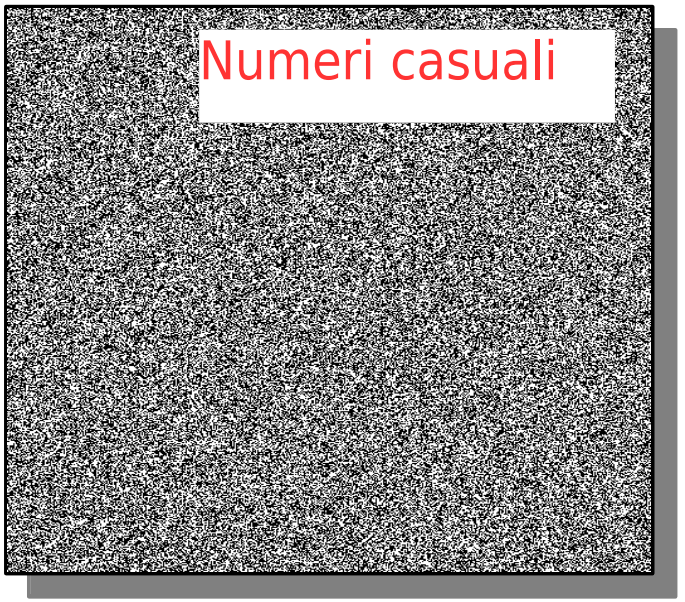
Numeri casuali

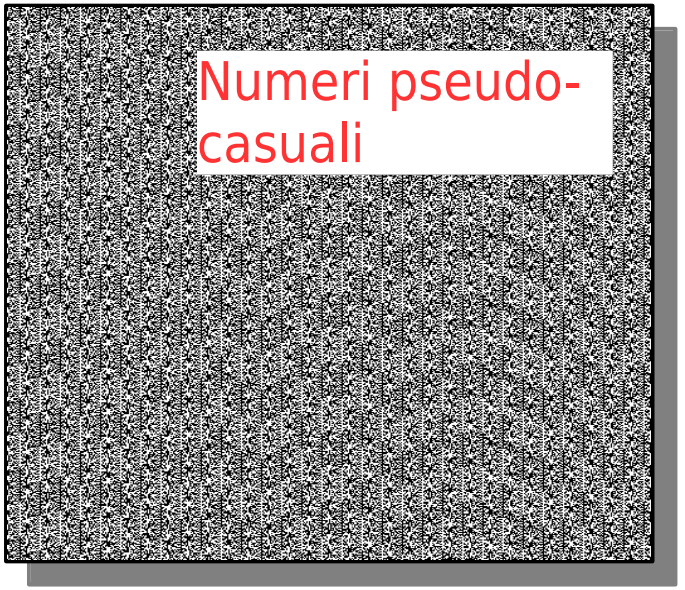
Una sequenza di numeri casuali e' una successione di elementi non correlati storicamente

Il C++ fornisce sequenze di numeri pseudo-casuali

I numeri casuali seguono una distribuzione di probabilità (pdf)







Generazione in C++

La funzione **rand** genera numeri interi fra **0** e **RAND_MAX**

E' necessario includere la libreria

```
#include <cstdlib>
```

```
int max_num = 10 ;
for (int it = 0; it < max_num; ++it)
  {
    std::cout << "numero casuale intero: "
              << rand () << "\n" ;
  }
```

```
double min = -1 ;
double max = 1 ;
for (int it = 0; it < max_num; ++it)
  {
    std::cout << "numero casuale razionale: "
              << min + (max - min) * rand ()
                 / (1. * RAND_MAX) << "\n" ;
  }
```

```
#include <ctime>
srand (time(NULL));
```

La funzione time restituisce il tempo in secondi dal 1 gennaio 1970.

Genera una sequenza di 10 numeri casuali distribuiti uniformemente, con la funzione **rand**

La funzione non ha argomenti e ritorna i numeri casuali

Cambio l'intervallo di generazione dei numeri casuali con una traslazione (**min +**) ed uno zoom:
(**(max-min)/RAND_MAX**)

$x \in [0, \text{RAND\_MAX}] \rightarrow x' \in [\min, \max]$

Provare a rieseguire il programma: i numeri generati sono sempre gli stessi?

Riprovare inizializzando il seme usando la funzione srand() **prima** di generare i numeri casuali.

come fare il disegno

```
int maxHisto = getHistoMax (histo, nBins) ;
int maxColumnsNum = 60 ;
int scale = maxHisto / maxColumnsNum + 1 ;
for (int i = 0 ; i<nBins ; ++i)
  {
    for (int j = 0 ; j<histo[i]/scale ; ++j)
      std::cout << "#" ;
    std::cout << "\n" ;
  }
```

disegniamo a schermo tanti "#" quanti sono i conteggi per ogni bin (oppure un numero proporzionale)

```
int getHistoMax (int * histo, int nBins)
{
  int max = 0 ;
  for (int i = 0 ; i<nBins ; ++i)
    if (histo[i] > max) max = histo[i] ;
  return max ;
}
```

```
#
###
#######
#############
####################
##################################
##########################################
##################################################
##############################################
########################################
##################################
####################
#############
#######
###
#
```

Disegno una distribuzione

ROOT permette di disegnare istogrammi in modo veloce ed efficace

```
TH1F pdf ("pdf","distribuzione",100,-1,1) ;
for (int it = 0; it < max_num; ++it)
  {
    double numero_casuale = min +
               (max - min) * rand ()
               / (1. * RAND_MAX) ;
    pdf.Fill (numero_casuale) ;
  }
```

TH1F e' un istogramma definito da -1 ad 1 con 100 intervalli (bin) sull'asse x

L'oggetto **pdf** e' di tipo **TH1F**

L'istogramma viene riempito con la funzione **Fill**, che viene chiamata sull'istogramma: e' un oggetto di C++!

```
TCanvas c1 ;
pdf.SetFillColor (5) ;
pdf.Draw () ;
c1.Print ("pdf.gif","gif") ;
```

Il disegno e' fatto su una tavolozza (**TCanvas**), che stampa (**print**) su un file (**pdf.gif**) la pdf (l'istogramma)

L'oggetto **c1** e' di tipo **TCanvas**

```
#include "TH1.h"
#include "TCanvas.h"
```

Le funzioni sono chiamate sugli oggetti
Bisogna includere le librerie di ROOT necessarie

Documentazione di ROOT

- User's guide all'indirizzo: http://root.cern.ch/drupal/content/users-guide
- Reference guide: http://root.cern.ch/root/html/
- Se sono interessato alla classe TH1, digito su google TH1 root e trovo direttamente il link che mi interessa: http://root.cern.ch/root/html/TH1.html

```
TH1 (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)

-*-*-*-*-*-*-*Normal constructor for fix bin size histograms*-*-*-*-*-*-*


  Creates the main histogram structure:
     name   : name of histogram (avoid blanks)
     title  : histogram title
              if title is of the form "stringt;stringx;stringy;stringz"
              the histogram title is set to stringt,
              the x axis title to stringy, the y axis title to stringy, etc.
     nbins  : number of bins
     xlow   : low edge of first bin
     xup    : upper edge of last bin (not included in last bin)

   When an histogram is created, it is automatically added to the list
   of special objects in the current directory.
   To find the pointer to this histogram in the current directory
   by its name, do:
   TH1F *h1 = (TH1F*)gDirectory->FindObject(name);

-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

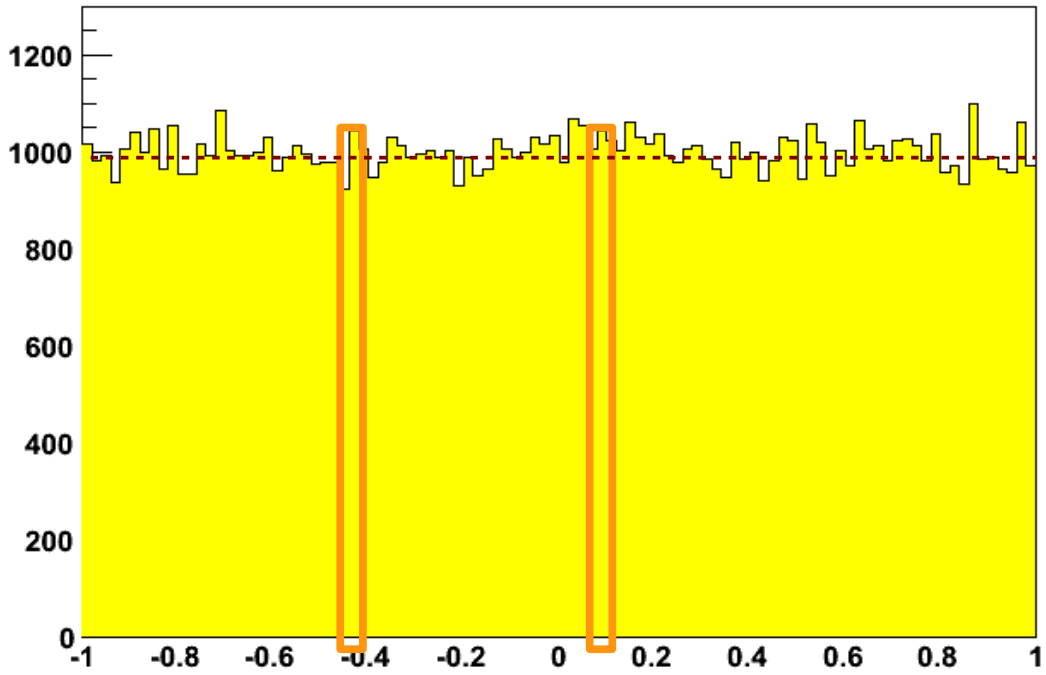
- Istruzioni sui parametri che è necessario inserire per costruire un oggetto, nel nostro esempio un istogramma 1D

La distribuzione risultante

- Il disegno di **pdf** visualizza il risultato della generazione di numeri casuali

```
bug:Lezione2 $ c++ -o esempio_2_01 \
>      `root-config --glibs --cflags` \
>      esempio_2_01.cpp
bug:Lezione2 $
```

Per compilare il C++ deve trovare le librerie di ROOT, il programma **root-config** scrive automaticamente dove sono



Ogni *bin* riporta il numero di volte in cui un evento casuale ha avuto valore fra il suo minimo e massimo (sull'asse x!)

La funzione **rand** quindi genera in maniera uniforme, perché i conteggi nei diversi *bin* sono circa uguali

statistiche di un *sample*

Su un *sample* di numeri casuali posso calcolare le statistiche che lo caratterizzano

```
double sum = 0 ;
double sumSq = 0 ;
int max_num = 100000 ;
for (int it = 0; it < max_num; ++it)
  {
    double numero_casuale = min +
           (max - min) * rand ()
           / (1. * RAND_MAX) ;
    sum += numero_casuale ;
    sumSq += numero_casuale *
             numero_casuale ;
  }
double mean = sum / max_num ;
double variance = sumSq / max_num
                  - mean * mean ;
```

La media e la varianza si calcolano a partire dalla somma degli elementi e dalla somma dei quadrati degli elementi di un insieme

$$\sigma^2 = E[x^2] - \mu^2, \; E[x^2] = \frac{\sum x_i^2}{N}$$

Le statistiche si possono calcolare anche nota la pdf sotto forma di istogramma

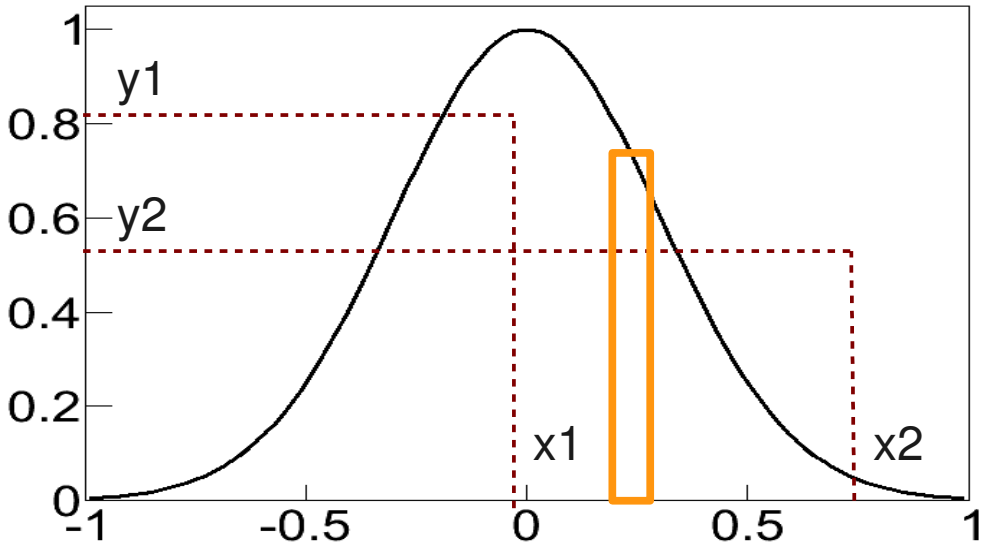
$$\sigma^2 = E[x^2] - \mu^2, \; E[x^2] = \frac{\sum x_i^2 \times N_i}{\sum N_i}$$

```
std::cout << "mean : " << pdf.GetMean () << "\n" ;
std::cout << "variance : " << pdf.GetRMS () *
                             pdf.GetRMS () << "\n" ;
```

Gli istogrammi di ROOT hanno funzioni che fanno il calcolo per noi

Try and catch

Generare un insieme di numeri casuali distribuiti non uniformement




```
double x = 0. ;
double y = 0. ;
do
  {
    x = rand_range (xMin, xMax) ;
    y = rand_range (yMin, yMax) ;
  } while (y > f(x)) ;
```

Il metodo del *try and catch* genera coppie di numeri casuali sull'asse x ed y con distribuzione uniforme

Se la coppia di punti si trova sotto la linea (x, f(x)) il numero x e' il numero cercato, altrimenti si continua la generazione

La frazione di x presi in un certo intervallo dipende dall'area di f(x) i quell'intervallo

L'algoritmo si può scrivere così, sfruttando una funzione (**rand_range**) che genera numeri casuali uniformemente

In dettaglio

```
double rand_range (double min, double max)
{
  return min + (max - min) *
       rand () / (1. * RAND_MAX) ;
}
```

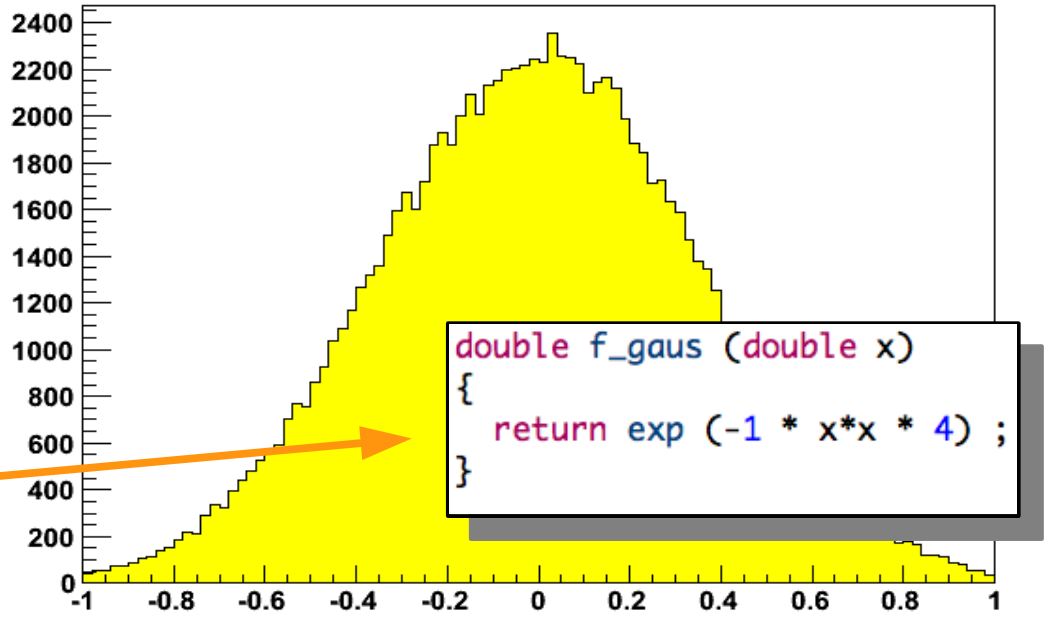
```
double rand_TAC (double f (double),
                 double xMin, double xMax,
                 double yMin, double yMax)
{
  double x = 0. ;
  double y = 0. ;
  do
    {
      x = rand_range (xMin, xMax) ;
      y = rand_range (yMin, yMax) ;
    } while (y > f(x)) ;
  return x ;
}
```

```
int max_num = 100000 ;
for (int it = 0; it < max_num; ++it)
  {
    double numero_casuale =
       rand_TAC (f_gaus, -1, 1, 0, 1) ;
    pdf.Fill (numero_casuale) ;
  }
```

Nel **main** viene chiamata **rand_TAC**, che chiama **rand_range** e si generano dati secondo **f_gaus**

Deve essere noto il massimo di **f_gaus**

Le funzioni possono essere messe in una libreria, da includere nel main




```
double f_gaus (double x)
{
  return exp (-1 * x*x * 4) ;
}
```

Teorema centrale del limite

Per generare numeri secondo una gaussiana si può utilizzare il teorema centrale del limite

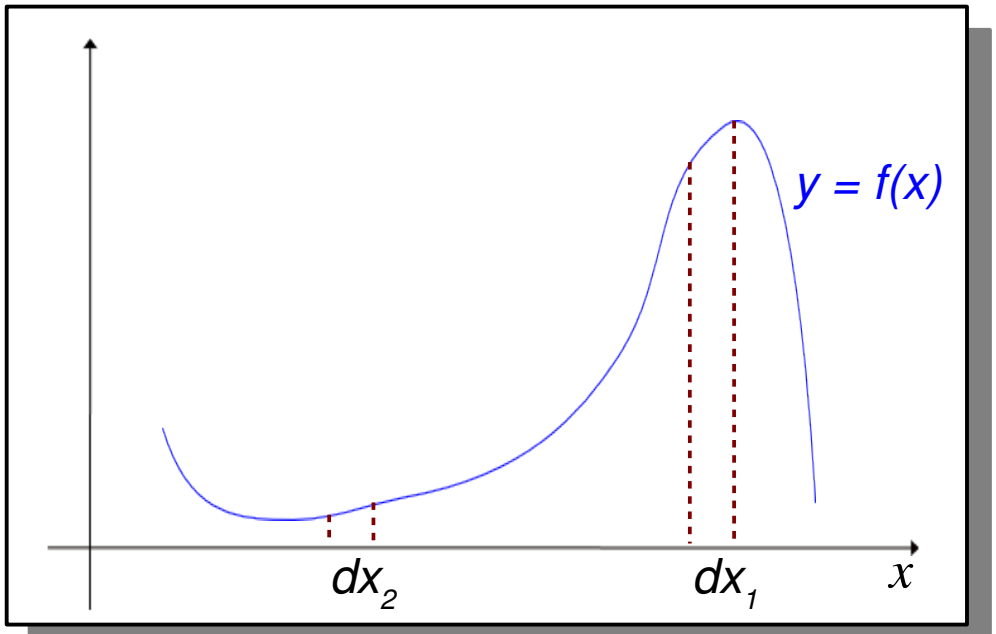
“La somma di N distribuzioni uniformi da' una gaussiana”

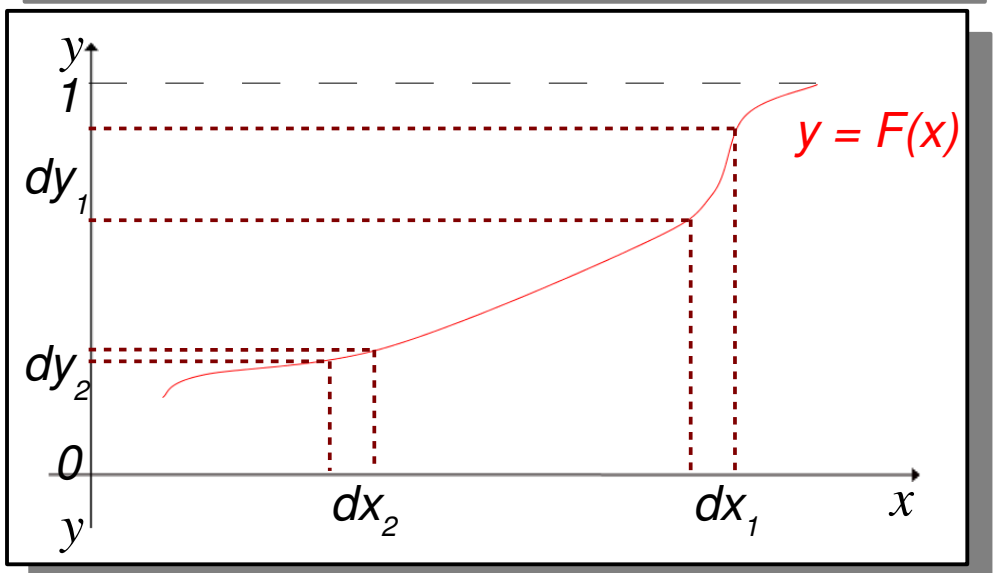
```
double rand_CLT (double xMin, double xMax,
                 int tries)
{
  double x = 0. ;

  for (int i = 0 ; i < tries ; ++i)
    {
      x += rand_range (xMin, xMax) ;
    }
  return x / tries ;
}
```

Nel **main** viene chiamata questa volta **rand_CLT**, che chiama **rand_range** per la generazione di numeri casuali secondo una distribuzione piatta

Per mantenere l'intervallo di interesse, bisogna dividere per il numero di tentativi

La funzione inversa





E' nota la forma funzionale della pdf secondo la quale vogliamo generare i numeri casuali, f(x)

Si calcola la primitiva della pdf, F(x)

- E' definita su R
- E' monotona crescente
- Ha valori compresi fra 0 ed 1

Generando uniformemente *y* in *[0,1)* osserveremo più frequentemente valori di *y* nell'intervallo dy_1 che in dy_2

Applicando l'inversa, verranno generati più frequentemente valori in dx_1 che in dx_2

$$dy = dF(x) = (dF/dx) \cdot dx = f(x) \cdot dx$$

Distribuzione esponenziale

- Voglio generare numeri casuali distribuiti come

$$f(x) = e^{-x/\mu}/\mu \qquad x \in [0,+\infty]$$

- Calcolo la funzione cumulativa

$$F(x) = \int_0^x f(s)ds = -e^{-x/\mu}+1$$

- Invertendo la *F(x)*, si trova

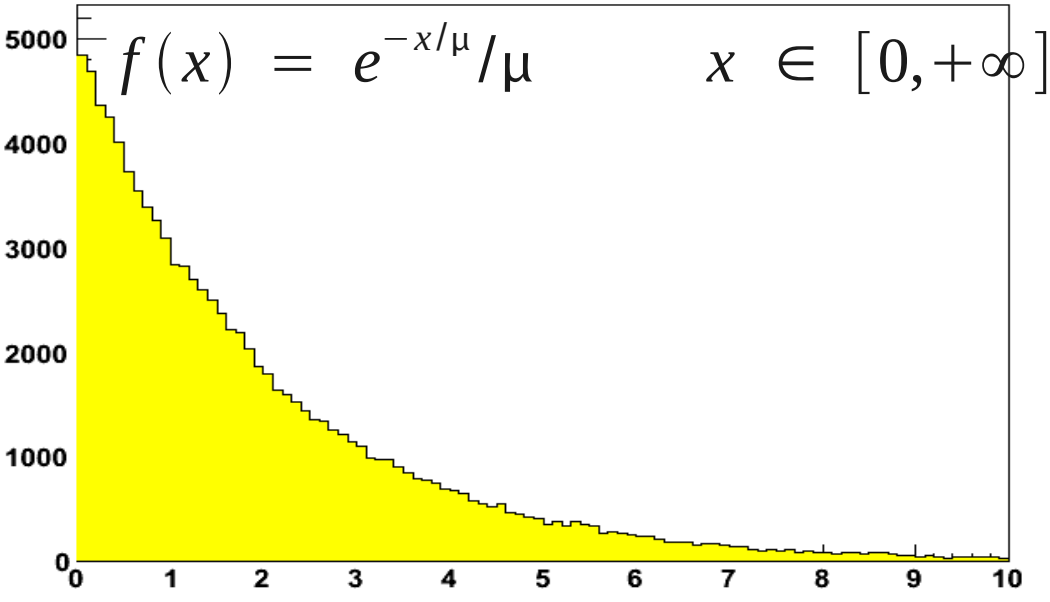
$$y = F(x) \quad \rightarrow \quad x = F^{-1}(y) = -\mu \log(1-y)$$

- Riassumendo: si generano numeri casuali uniformi *y* tra *[0,1]* e si calcola *x* dalla formula precedente: *x* risulta avere distribuzione esponenziale, con media *μ*

in dettaglio

Nel MAIN (file .cpp)

```
TH1F pdf("pdf","distribuzione",100,xmin,xmax);
for (int i=0;i<n_num;i++)
{
  num_rand=rand_exp(0,1,1);
  pdf.Fill(num_rand);
}
```



Funzioni (file .cc)

```
double rand_range (double min, double max)
{
  return min+(rand()*(max-min)/double(RAND_MAX));
}
```

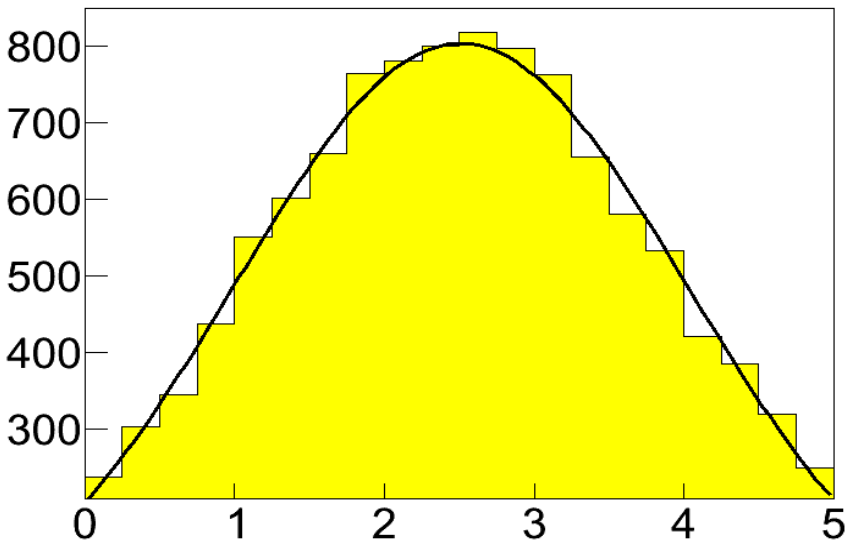
```
double rand_exp (double ymin,double ymax, double mu)
{
  double x,y;
  y=rand_range(ymin,ymax);
  x=-mu*log(1-y);
  return x;
}
```

esercizi

- **Esercizio 1:** Stimare la media e la varianza di una distribuzione uniforme sull'intervallo [a,b] e verificare che la varianza è uguale a: $\frac{(b-a)^2}{12}$
- **Esercizio 2:** Scrivere un programma che generi numeri casuali distribuiti secondo una gaussiana con le tecniche di *try and catch* e del teorema centrale del limite
- **Esercizio 3:** "verificare" il teorema centrale del limite con una distribuzione esponenziale invece di una distribuzione uniforme
  - quale è la larghezza della gaussiana prodotta con la tecnica del teorema centrale del limite?

scrivere gli algoritmi che generano i numeri casuali
come **funzioni** del C++

una semplice visualizzazione



un istogramma conta il numero di eventi che cascano in un singolo bin dell'asse di definizione, quindi e' una pdf integrata a tratti su ogni singolo bin

per rappresentare un istogramma utilizziamo un vettore di interi (i conteggi in ogni bin)

per riempire l'istogramma, per ogni numero casuale decidiamo quale bin incrementare

```
int nBins = 30 ;
int * histo = new int[nBins] ;
for (int i = 0 ; i<nBins ; ++i)
  histo[i] = 0 ;
```

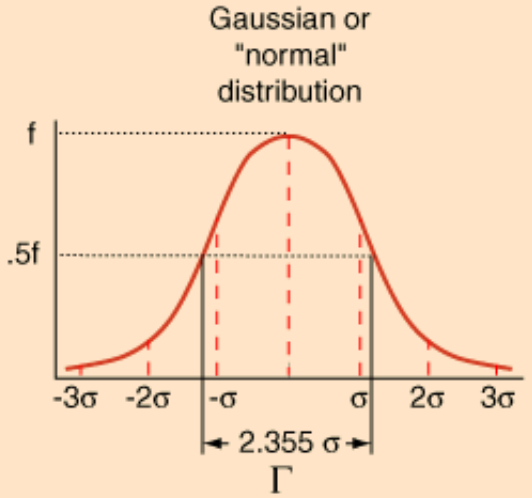
```
for (int it = 0; it < max_num; ++it)
  {
    double numero_casuale =
       rand_CLT (xMin, xMax, nTries) ;
    int index = floor ((numero_casuale - xMin)/step) ;
    ++histo[index] ;
  }
```

Larghezza di una gaussiana

Metà larghezza della distribuzione Gaussiana

La larghezza della curva Gaussiana a metà del massimo può essere ottenuta dalla fuzione nel seguente modo.

$$f(x) = Ce^{(-x^2/2\sigma^2)}$$



Sia x=h a metà dell'altezza massima.

$$0.5C = Ce^{(-h^2/2\sigma^2)}$$

Prendendo il logaritmo naturale su ambedue i lati:

$$\ln(.5) = -h^2 / 2\sigma^2$$

$$h^2 = -2\ln(.5)\sigma^2 = 2\ln 2\sigma^2$$

La larghezza totale è 2h.

$$\Gamma = 2\sqrt{2\ln 2}\sigma = 2.355\sigma$$