



Corso di Laboratorio 2 – Programmazione C++

Silvia Arcelli

9 Novembre 2015

Generazione di numeri casuali

Numeri casuali: valore assunto da una variabile aleatoria, il cui valore è per definizione imprevedibile 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 in linea teorica perfettamente deterministiche e predicibili:

$$X_{n+1} = f(X_n)$$

Generazione di numeri casuali

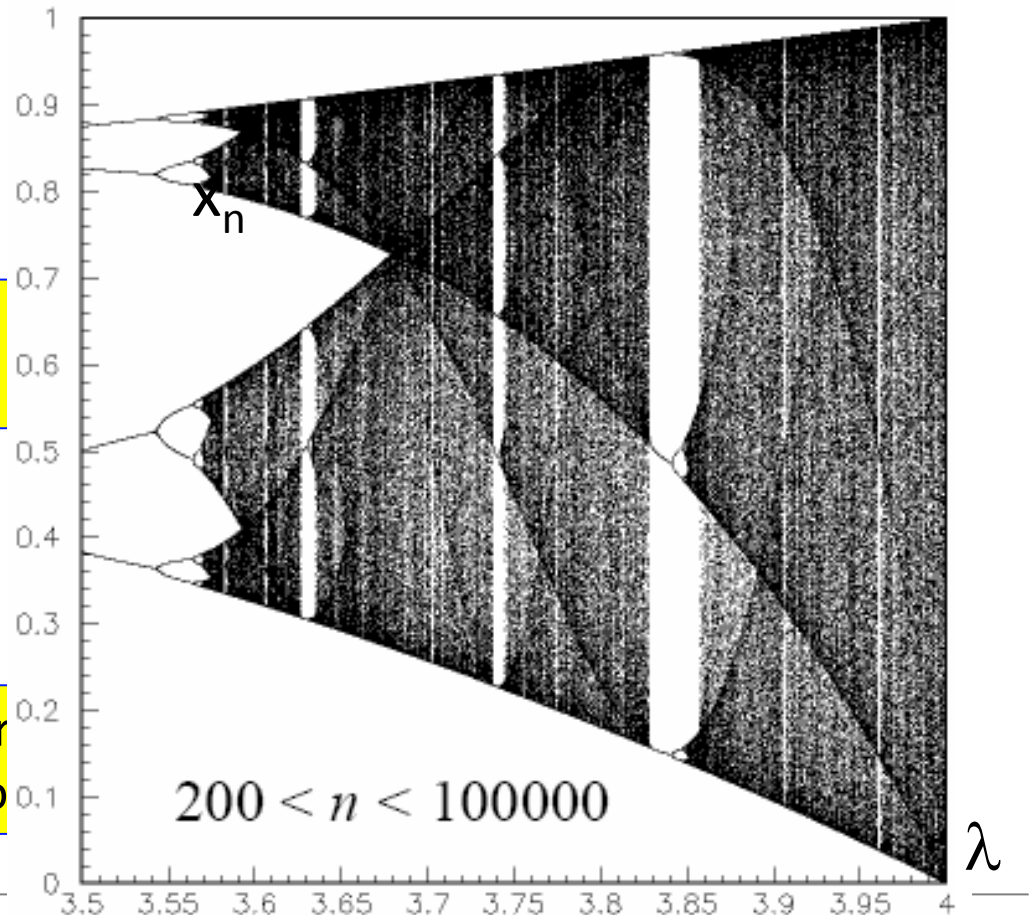
Gli algoritmi utilizzati hanno tuttavia delle proprietà tali per cui le sequenze da loro prodotte sono caratterizzate da proprietà molto simili a quelle di numeri veramente casuali. Esempio ([mappa logistica](#)):

$$x_{n+1} = \lambda \cdot x_n (1 - x_n)$$

Comportamento asintotico
per **piccoli** valori del parametro:

$$n \rightarrow \infty \quad x_{\infty} = \frac{\lambda - 1}{\lambda}$$

Per **λ maggiori** si assiste ad una transizione verso un comportamento “caotico”



Generazione di numeri casuali

- esempio di transizione al comportamento caotico della sequenza:

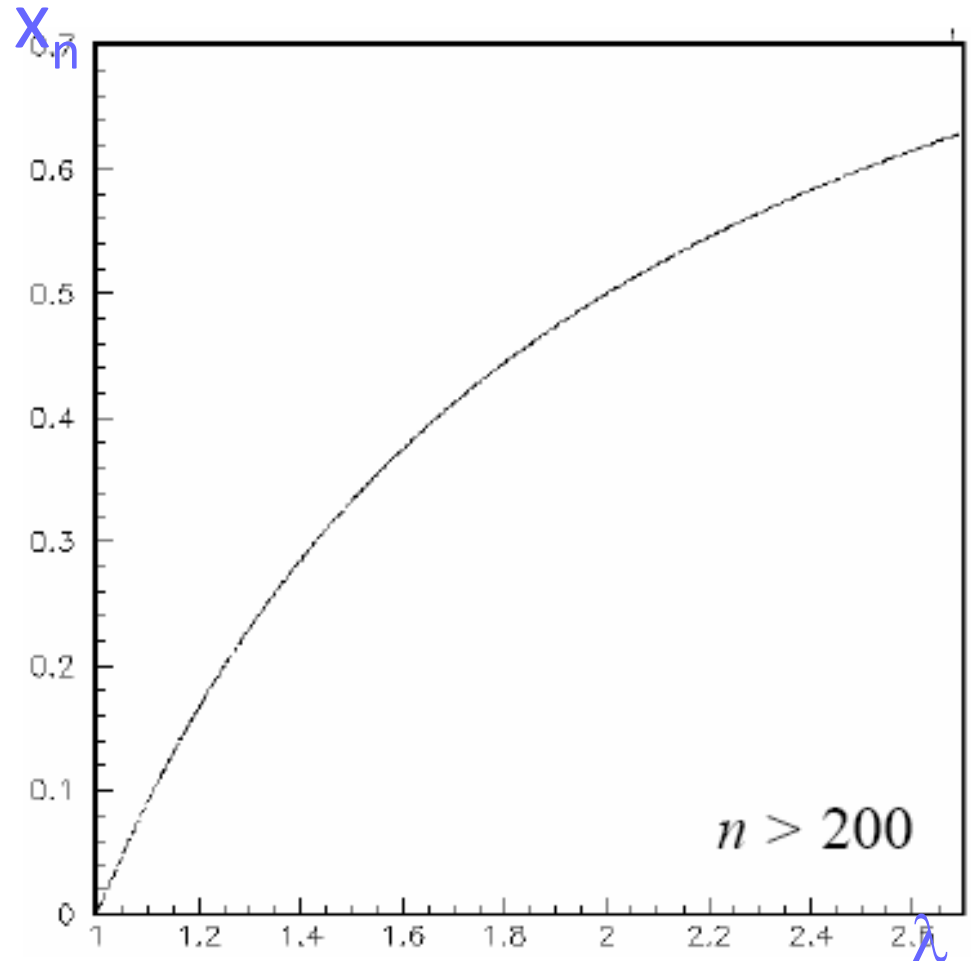
$$x_{n+1} = \lambda x_n (1 - x_n)$$

- La successione assume comportamenti diversi a seconda del parametro λ ; per $\lambda < 1$ converge a 0, per l'insieme di valori $(1 < \lambda < 3)$ la sequenza, per qualunque x iniziale in $[0,1]$, converge ad un valore stabile per $n \rightarrow \infty$ (n = numero di iterazioni):

$$x_{\infty} = \lambda x_{\infty} (1 - x_{\infty}) \Rightarrow x_{\infty} = \frac{\lambda - 1}{\lambda}$$

Generazione di numeri casuali

- Per piccoli valori di λ , la successione converge a un valore univoco.
- Nell'esempio è posto $x_0=0.5$ e sono mostrate le “storie” delle sequenze per $n>200$

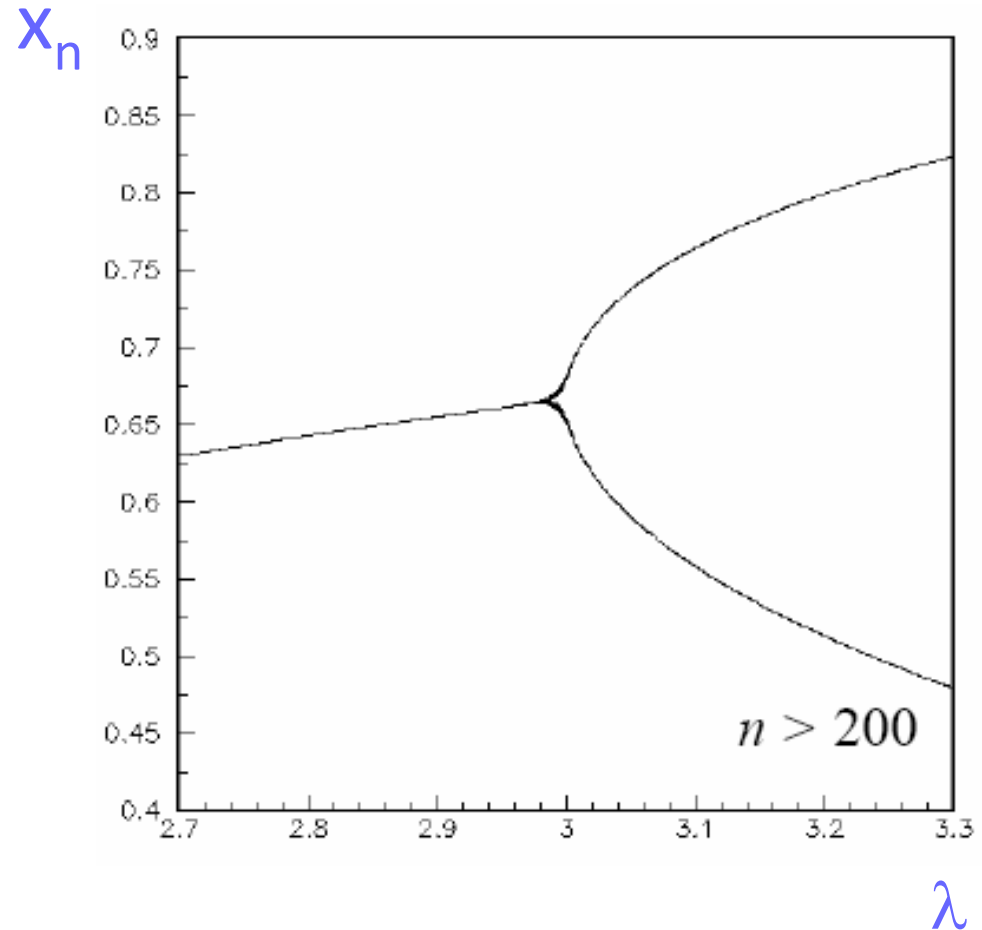


Generazione di numeri casuali

- Per $\lambda > 3$, la successione non converge, ma oscilla tra due valori:

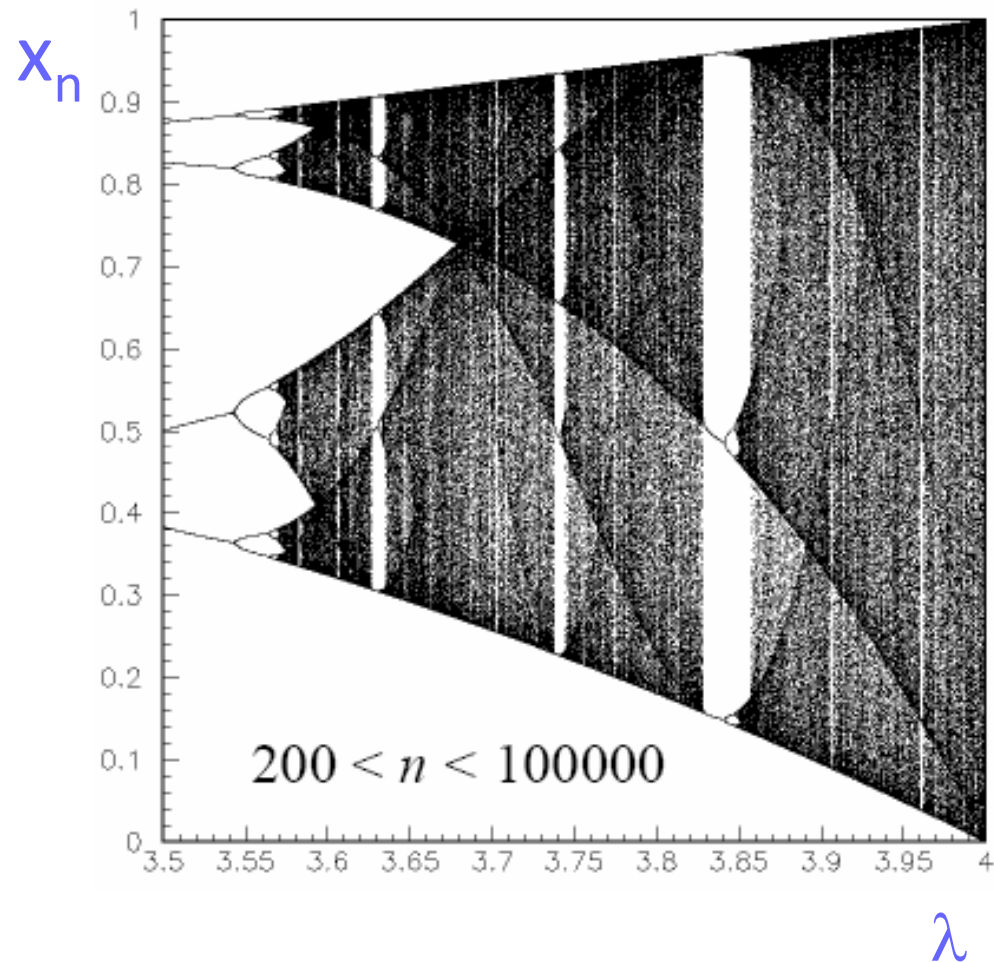
$$x_a = \lambda x_b (1 - x_b)$$

$$x_b = \lambda x_a (1 - x_a)$$



Generazione di numeri casuali

- Per λ ancora maggiori il comportamento diviene **completamente caotico**. E.g., per $\lambda = 4$ i valori riempiono densamente l'intervallo $[0,1]$



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
 - 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 (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):

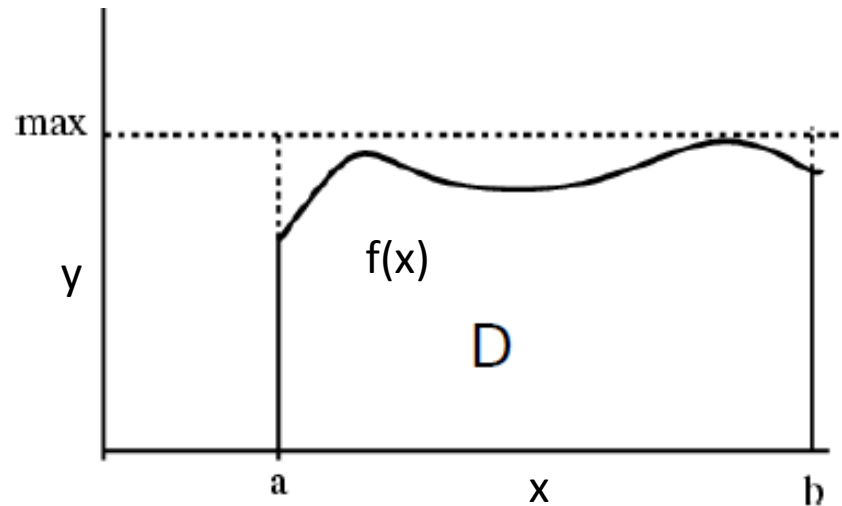
Supponete di voler generare random una variabile x distribuita secondo una distribuzione di probabilità $f(x)$. Si può usare un metodo del tipo “*hit or miss*” (applicabile a ogni $f(x)$ su un dominio finito, anche se può essere molto inefficiente a seconda del tipo di funzione).

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

$$x = U(a, b),$$

$$y = U(0, \max)$$

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

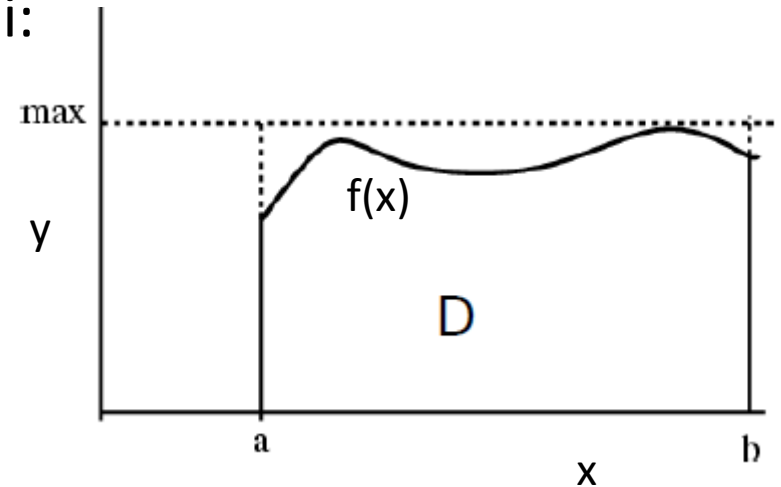


Metodo del Rigetto

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

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

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



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 (trasf. inversa), e tale che:

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

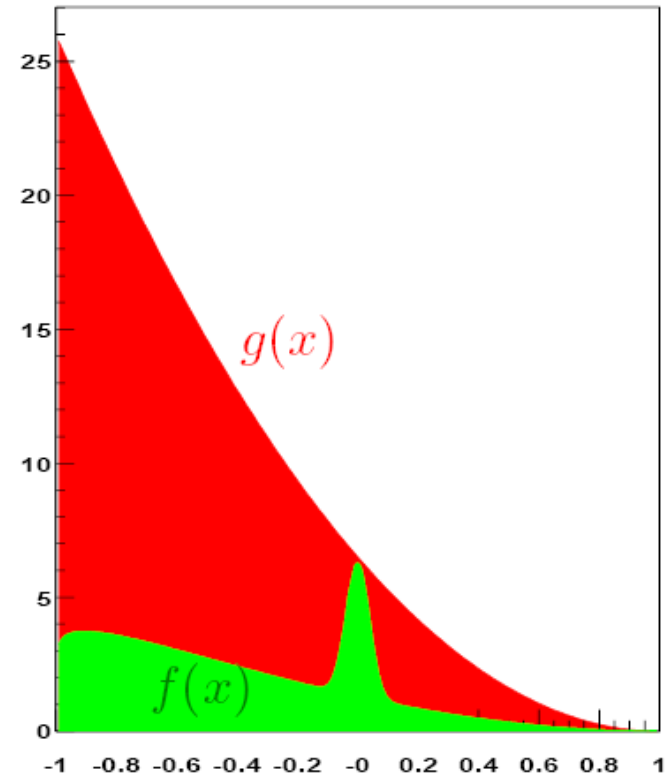
$g(x)$ "copre" $f(x)$

- b) Generate punti in x secondo la $g(x)$ ed estraete un numero casuale $y \in U(0, g(x))$
- c) Se $y < f(x)$ 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 distribuita secondo $f(X)$

Metodo della Trasformazione Inversa, dimostrazione:

vogliamo dimostrare che y è distribuita secondo $f(x)$, per cui la probabilità di avere $y < X$ deve essere uguale alla cumulativa $F(X)$:

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

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 $F(X)$, e automaticamente y risulta distribuita secondo la pdf $f(x)$

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 F 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:

Potenza:

$$f(x) = x^\alpha \qquad y = F(x) = \int_0^x x^\alpha dy = \frac{1}{\alpha + 1} x^{\alpha+1}$$

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

$$x = F^{-1}(y) \propto (y)^{1/\alpha+1}$$

x sarà distribuito secondo una
Funzione di potenza

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:

$$\iint_{x,y} \frac{1}{2\pi} \cdot e^{-\frac{x^2+y^2}{2}} dx dy = \iint_{r,\varphi} \frac{1}{2\pi} \cdot e^{-\frac{r^2}{2}} r dr d\varphi = 1$$

E la cumulativa ha la forma:

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

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 n_1 e n_2 in $[0,1]$ e determino x e y come:

(Metodo diBox-Muller)

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

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 $N(\mu, \sigma)$, è sufficiente fare una trasformazione del tipo: $x' = \sigma \cdot x + \mu$
-

Metodo della Trasformazione Inversa:

Alternativa per la generazione della gaussiana: si può sfruttare il teorema del limite centrale, che afferma che la somma di n variabili casuali per $N \rightarrow \infty$ tende a una gaussiana:

$$f\left(\sum_{i=1..N} \sqrt{n/12} \cdot (U_i - \frac{1}{2})\right) \rightarrow G(0,1)$$

La validità dell'approssimazione è molto buona già per $N=12$. occorre però fare almeno 12 estrazioni random uniformi per avere un valore di x distribuito come una gaussiana.

[testGaussian.C](#)

Trasformazione Inversa per distribuzioni discrete

Per una distribuzione di probabilità di una **variabile discreta**:

$$P(X = x_i) = p_i, \quad i = 1, \dots, n, \dots \quad \sum_i p_i = 1.$$

La procedura è estrarre un numero random $U(0,1)$, e confrontarlo con la probabilità cumulativa $F(i)$:

$$X = \begin{cases} x_1 & \text{se } U \leq p_1; \\ x_2 & \text{se } p_1 < U \leq p_1 + p_2; \\ \vdots & \\ x_i & \text{se } \sum_{j=1}^{i-1} p_j < U \leq \sum_{j=1}^i p_j; \\ \vdots & \end{cases}$$

Implementazione algoritmica Traf. Inversa.

Quindi, in generale, per una qualunque distribuzione di probabilità di una variabile discreta, l'implementazione algoritmica con il metodo della trasformazione inversa è del tipo:

```
int discreteProb(double *p){  
    int k=0;  
    double sum=p[0];  
    double r=rand();  
    while (sum<r)sum+=p[k++];  
    return k-1;  
}
```

generazione di una distribuzione discreta

■ Distribuzione Binomiale:

■ p costante

■ Estrazioni indipendenti

$$p(k) = \binom{n}{k} p^k q^{n-k}$$

Probabilità di k successi su n lanci $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad p = (1-q)$

■ Algoritmo alternativo ([binomial.C](#)) alla trasformata inversa (somma di variabili di Bernoulli), molte estrazioni random, ma si risparmia nel calcolo dei p[i]:

```
int binomial(int n, double p) {  
    Int k = 0;  
    for (int i=0; i<n; i++) { if (rand() > p) continue; k++; }  
    return k;  
}
```

Limitazioni del metodo di trasformazione inversa:

- Solitamente richiede che la funzione di distribuzione cumulativa sia conosciuta analiticamente, sia integrabile e che sia invertibile. 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 può però essere 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 ad es. generate esplicitamente:
 - `Exp(tau)`
 - `Integer(imax)`
 - `Uniform(a,b)`
 - `Gaus(media,sigma)`
 - `Landau(mod,sigma)`
 - `Poisson(media)`
 - `Binomial(ntot,prob)`

PRNG in ROOT: distribuzioni generiche

- In generale si può generare secondo qualunque funzione:

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);
- In entrambi i casi si usa la tecnica di integrare numericamente e tabulare la cumulativa

Come far girare un vostro programma con root

- Se avete un vostro programma e al suo interno volete utilizzare le classi di ROOT (ad es. fare un istogramma o un grafico), come si procede? Supponiamo la seguente struttura:

Un main

- `main.cxx`

Una serie di classi associate, utilizzate nel main program, contenute nei rispettivi file di intestazione ed implementazione (qui ad es. una madre e una derivata);

- `MyClass_Mother.cxx, .h`
- `MyClass_Daugther.cxx, .h`

Come far girare un vostro programma con root

- Da linea di comando di root (modalità **interpretata**):

```
>.L MyClass_Mother.cxx
```

```
>.L MyClass_Daughter.cxx
```

```
>.L main.cxx
```

- Oppure (equivalente):

```
>gROOT->LoadMacro("MyClass_Mother.cxx");
```

```
>gROOT->LoadMacro("MyClass_Daughter.cxx");
```

```
>gROOT->LoadMacro("main.cxx");
```

} Anche dentro main.cxx

- Per eseguire il programma:

```
>main()
```

Come far girare un vostro programma con root

- Da linea di comando di root (modalità **compilata!**):

```
>gROOT->LoadMacro("MyClass_Mother.cxx++");  
>gROOT->LoadMacro("MyClass_Daughter.cxx++");  
>gROOT->LoadMacro("main.cxx++");
```

} non dentro main.cxx

Crea tre librerie dinamiche con estensione **.so**

Per eseguire il programma:

```
>mymain()
```

N.B. in modalità compilata non potete più utilizzare il nome main per la vostra funzione di steering, qualunque altro nome va bene

Come far girare un vostro programma con root

N.B. in modalità compilata, occorre mettere tutti i necessari `#include` files esattamente come in un programma c++, compresi quelli relativi alle classi di Root (questo non è necessario in modalità interpretata, ma ricordate che il codice non è ottimizzato e l'esecuzione è più lenta).

- Ad es. librerie standard;
`#include <iostream>`
`#include <stdlib>`
- Ad es. se si usa un istogramma e un file root,
`#include "TH1D.h", #include "TFile.h", Etc.`