Esercitazione Numeri Pseudo-Random

Matteo Duranti

matteo.duranti@pg.infn.it

Esercitazione

- Realizzare una *classe* C++ che faccia da *intefaccia* a diversi algoritmi di generazione di numeri pseudo-random:
    - si possa settare la *seed*;
    - si possa scegliere l'algoritmo;
    - si possa ottenere un numero random generato con l'algoritmo scelto;

- Implementare (ed inserire nell'interfaccia) i seguenti algoritmi:
    - LCG;
    - Generatore a 16 bit semplice;
    - TRandom (quale/i volete) di ROOT (solo "accesso")

- Implementare la generazione di numeri distribuiti secondo un'esponenziale (τ arbitrario) e secondo una gaussiana (μ e σ arbitrarie)

- Realizzare un programma che utilizzi l'interfaccia realizzata per:
    - profilare in tempo i vari generatori;
    - verificare la periodicità di LCG e generatore 16 bit;
    - verificare l'effetto Marsaglia per un LCG con modulo 2

- [facoltativo] Verificare l'effetto Marsaglia per un LCG con modulo "grande", facendo il plot (TGraph di ROOT, gnuplot, carta millimetrata, …)

Numeri pseudo-random

Generatore di una sequenza pseudo-random a 16 bit:

```
unsigned int random;  // Variabile globale in cui è memorizzato il numero casuale
(16bit)

void randomNext(void) {
   // Aggiorna sequenza random
   // Algoritmo Polinomiale:
   // +> b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15
   // |     |  |     |                                     |
   // ------+--+-----+-------------------------------------+
   // carry = b1^b2^b4^b15
   // Pn+1=(Pn<<1)|carry
   unsigned short int randomtmp;  // Accumulo le operazioni ex-OR
   if (random==0) random++;  // N.B. : il seed dovrebbe essere != 0
   randomtmp=0;
   if ((unsigned short int)random&0x02) randomtmp=1;
   if ((unsigned short int)random&0x04) randomtmp^=1;
   if ((unsigned short int)random&0x10) randomtmp^=1;
   if ((unsigned short int)random&0x8000) randomtmp^=1;
   random = (unsigned short int)((random<<1)|randomtmp);
}
```

Generatore Lineare Congruente (Linear Congruent Generators, LCG)

La sequenza è definita da: $x_{n+1} = (\lambda x_n + \mu) \mod m$

cioè x_{n+1} è il resto della divisione per *m* di $(\lambda x_n + \mu)$

x_0 ($0 \leq x_0 < m$) è il valore di partenza o ***seed***
λ ($0 < \lambda < m$) è il **moltiplicatore**
μ ($0 \leq \mu < m$) è l'**incremento**
m ($m > 0$) è il **modulo**

→ la lunghezza della sequenza (i.e. la periodicita) è *m*, per tutte le seed, solo se le λ e la μ sono scelti accuratamente:

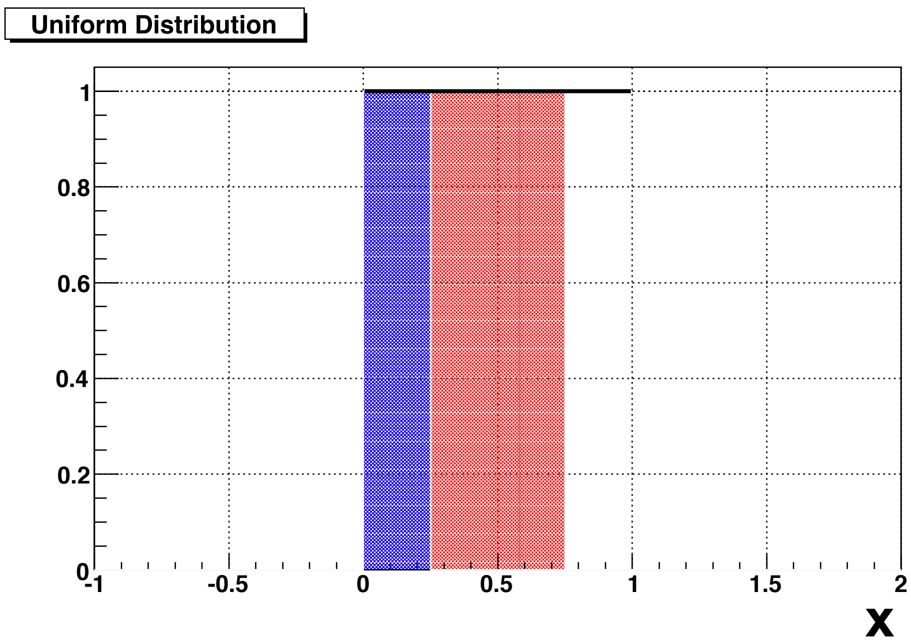
- μ e m sono co-primi (massimo comun divisore è *1*)
- λ-1 è divisibile per tutti i fattori primi di *m*
- λ-1 è divisibile per *4* se *m* è divisibile per *4*

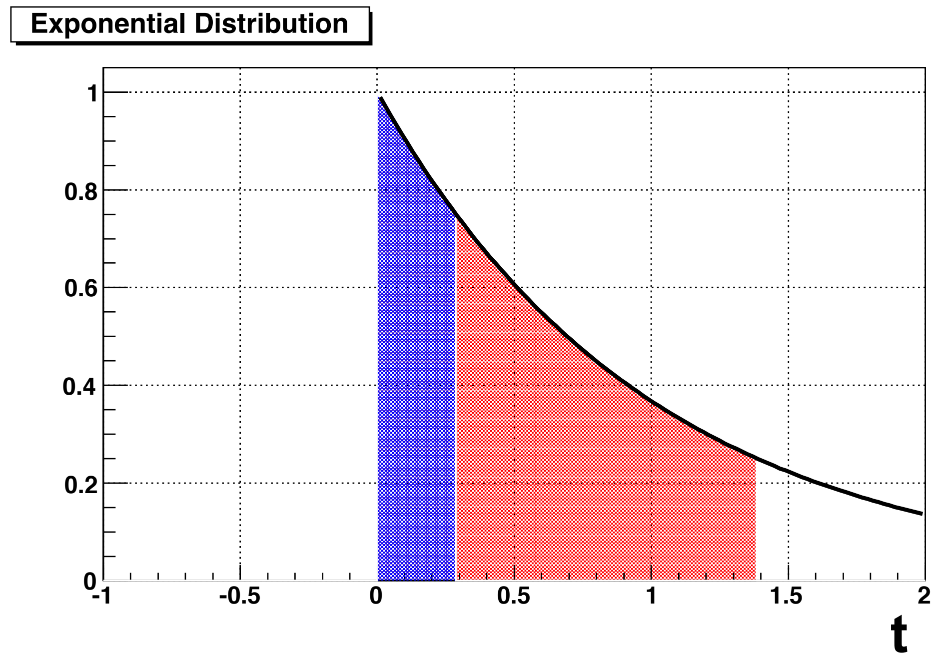
$$x_0 = 1\,; \lambda = 9; \mu = 3; m = 32$$

Utilizzare come modulo una potenza del 2 produce un LCG computazionalmente molto efficiente: i bit più significativi non vengono nemmeno calcolati (scrivendo il codice nel giusto modo…)

Metodo di inversione

L'idea è quella di "rimappare" la distribuzione uniforme generata in quella desiderata:

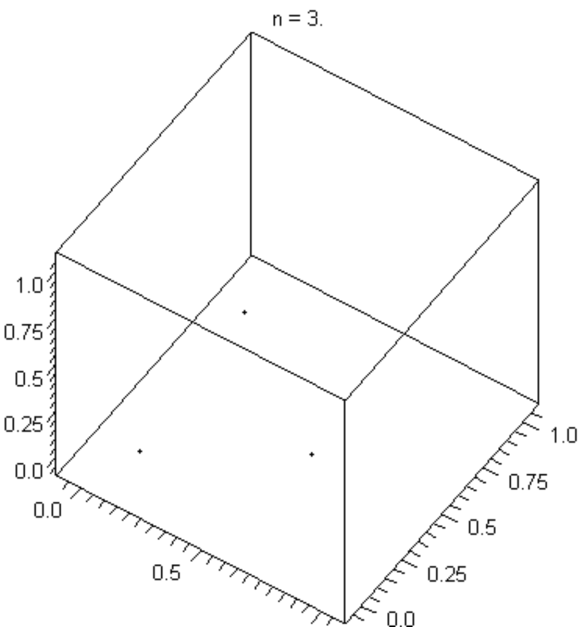


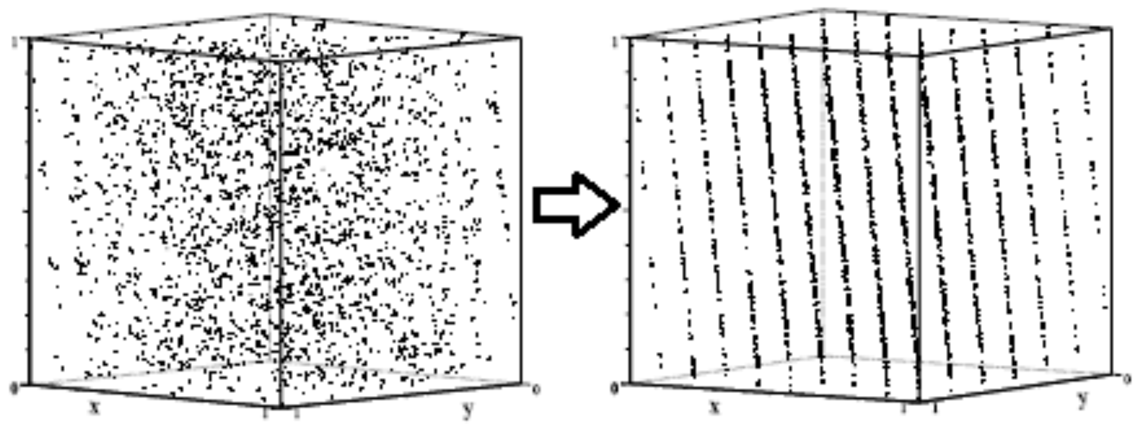


- la zona blu contiene il 25% dell'integrale delle due funzioni;
- la zona rossa il 75%

Problemi dei generatori congruenti (Marsaglia effect)

Se un generatore è utilizzato per produrre numeri pseudo-random in uno spazio a *d* dimensioni, questi giaceranno su, al massimo, $(d!\ m)^{1/d}$ iperpiani (teorema di Marsaglia)





- di fatto i numeri sono generati con dei pattern specifici;
- l’effetto può essere mitigato usando un modulo, *m*, molto grande;