Generazione di numeri (pseudo)casuali

Mario Ambrosino

22 Aprile 2018

Indice

Capitolo 1

Numero (pseudo)casuali

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

John von Neumann

Vi è un apparente paradosso nel richiedere ad un calcolatore di fornire risultati casuali in senso stretto, a causa del suo progetto intrisecamente deterministico. La necessità di simulare processi di natura stocastica su calcolatore ha condotto allo studio di metodi per la generazione tramite calcolo numerico di sequenze di numeri tali da soddisfare test di aleatorità.

Questi generatori portano il nome di *generatori di numeri pseudocasuali.*

1.1 Generatori lineari congruenziali (LCG)

Il più semplice e vecchio generatore di numeri pseudocasuali distribuiti uniformemente è il generatore basato sulle congruenze lineari. Essi possono essere utilizzati come *starter* per altri generatori di numeri pseudocasuali, avendo il pregio di essere autonomi e semplici da implementare.

Supponiamo di voler generare una distribuzione di numeri aleatori distribuiti secondo una distribuzione di probabilità $P(x)$ uniforme in $[0, 1]$. Affermiamo allora che

Definizione 1. Un generatore di numeri LCG (basato sullo schema delle congruenze lineari) è una funzione ricorsiva lineare[1, § 11.2]

$$x_{n+1} = (ax_n + b) \mod m \tag{1.1}$$

dove

- a : **moltiplicatore** del LCG;
- b : **incremento** del LCG;
- m : **modulo** del LCG;
- x_0 : **seme** di inizializzazione del LCG.

La qualità di un LCG è lagata alla scelta dei parametri $\{a, b, m\}$. I generatori di numeri LCG sono periodici per definizione, ma non possiamo dire a priori che il periodo T di un dato generatore $\{a, b, m\}$ coincida con il suo modulo, ma solo che esso ne costituisce il massimo: $T \leq m$. Si richiede che un buon generatore abbia un periodo ragionevolmente lungo (rispetto alla destinazione d'uso della sequenza di numeri).

Generatori puramente moltiplicativi Un generatore in cui è assente l'incremento ($b = 0$) è definito come *puramente moltiplicativo.*

$$x_{n+1} = ax_n \mod m$$

Tale categoria di generatori è più performante dal punto di vista computazionale poiché un numero ridotto di operazioni. Il costo di tale scelta sta in periodi generalmente più brevi.

Alcuni teoremi utili Riportiamo testualmente alcuni risultati di teoria dei numeri tratti da [1, §11.2.3] e utili per comprendere la scelta dei numeri magici associati ai generatori.

Teorema 1. (del massimo periodo). *Una sequenza lineare congruenziale ha un periodo di lunghezza $T = m$ se e solo se:*

1) b e m sono coprimi;

2) $\tilde{a} = a - 1$ è un multiplo di p per ogni primo p che divide m;

3) se m è multiplo di 4, $\tilde{a}$ è un multiplo di p.

Definizione 2. Sia a coprimo per m. Allora definiamo **ordine** di a modulo m il minimo$\lambda \in \mathbb{N}$ tale che $\left(a^{\lambda}\right) \mod m = 1$.

Definizione 3. Definiamo **elemento primitivo modulo** m ogni valore di a che dà il massimo possibile per l'ordine a modulo m.

Definizione 4. Definiamo **ordine di un elemento primitivo** la quantità $\lambda(m)$

Teorema 2. ***(di Carmichael).*** *Il massimo periodo che può essere ottenuto per un generatore puramente moltiplicativo è pari all'ordine dell'elemento primitivo modulo m:*

$$\max T_{b=0} = \lambda(m)$$

Il valore di $\lambda(m)$ può essere dedotto usando i seguenti risultati:

1. $\lambda(2) = 1$;
2. $\lambda(4) = 2$;
3. $\lambda(2^z) = 2^{z-2},\ z > 2$;
4. $\lambda(p^z) = p^{z-1}(p-1),\ p > 2$ e primo;
5. per $\{p_i\}_{i=1\ldots k}$, collezione di numeri primi, $\lambda(p_1^{z_1} p_2^{z_2} \ldots p_k^z) = \text{m.c.m.}(\{\lambda(p_i^{z_i})\}_{i=1\ldots k})$.

1.2 Esempi di LCG

1.2.1 Generatore *tipo "Minimal Standard"*

Proposto da Lewis, Goodman e Miller nel 1969 [1, § 11.3], esso è un generatore puramente moltiplicativo:

$$a = 7^5 = 16\,807 \qquad m = 2^{31} - 1 = 2\,147\,483\,647 \qquad b = 0;$$

Esso ha come modulo un numero m primo, dunque (secondo il punto 4 delle conseguenze del teorema di Carmichael) ha ordine $\lambda(m) = m - 1$ che costituisce un ottimo risultato, essendo il massimo periodo teorico proprio m (cfr. Teorema del massimo periodo)

Il periodo del generatore risulta dunque essere

$$T = m - 1 = 2^{31} - 2.$$

Aggirare l'overflow: il metodo di Schrage il prodotto di a e di $x_n \approx m$ nel passo ricorsivo $x_{n+1} = ax_n$ può essere troppo grande per un intero di lunghezza 32 bit su certi calcolatori; l'uscita dalle condizioni di sicurezza garantite dalla ALU comporterebbe effetti imprevedibili dal punto di vista computazionale.

Al fine di evitare l'overflow possiamo seguire il seguente metodo [2, §7 - p.278] dovuto a Schrage (Schrage 1979; Bratley et al. 1983) , basato sulla fattorizzazione approssimata di m.

Proposizione 1. *Consideriamo la decomposizione:*

$$m = aq + r; \qquad q = [m/a] \in \mathbb{N}; \qquad r = m \mod a;$$

dove le parentesi quadre indicano la parte intera della divisione m/a. Allora si ha che:

$$(ax) \mod m = \begin{cases} a\,(x \mod q) - r\,[x/q] & \textit{se positivo;} \\ a\,(x \mod q) - r\,[x/q] + m & \textit{altrimenti.} \end{cases}$$

Se $r \ll q$ e $0 < x < m - 1$.

Dimostrazione. Valgono i seguenti fatti:

(1): *Sia (A) $z \in \{0, 1, \ldots, m-1\}$ e (B) $r < q$. Allora $a\,(z \mod q) \in \{0, 1, \ldots, m-1\}$.* **Dim:**

$a\,(z \mod q) = a\,(z - q\,[z/q]) = az - aq\,[z/q] = aq\,(z/q) - aq\,[z/q] = aq\,(z/q - [z/q])$.

Poiché è sempre vero che $0 \leq (x - [x]) < 1$ e che $[x] < x$ ne consegue che $0 = aq\,(0) \leq aq\,(z \mod q) < aq\,(1) = aq = a\,[m/a] \leq a\,(m/a) = m$.

Ciò dimostra che $a\,(z \mod) \in \{0, 1, \ldots, m-1\}$.

(2): $r\,[z/q] \in \{0, 1, \ldots, m-1\}$. **Dim:**

$r \geq 0 \wedge z/q \implies r\,[z/q] \in \mathbb{N}$; per mostare il limite superiore osserviamo che, per (B) e (A), si ha $r\,[z/q] \leq r\,(z/q) = z\,(r/q) \leq z \leq m - 1$.

(3): $az \mod m = (a\,(z \mod q) - r\,[z/q])$. **Dim:**

per definizione $z \mod q = z - [z/q]$; allora $a\,(z \mod q) - r\,[z/q] = az - (aq + r)\,[z/q] = az - m\,[z/q]$. Ne consegue che l'equazione

$$az - a\,(z \mod q) - r\,[z/q] = m\,[z/q]$$

è un'identità $\mod m$.

Fissando gli opportuni periodi del modulo si ottiene la tesi. □

L'applicazione del metodi di Schrage ai valori del generatore comporta l'uso delle quantità:

$$q = 127773 \quad r = 2836$$

Riescolamento dei risultati iniziali *a là Bays - Durham* Il *Minimal Standard* presenta correlazioni per i valori iniziali della sequenza da essa generata. Per eliminare tale correlazioni si preferisci rimescolare i dati iniziali utilizzando un metodo ricorsivo ideato da DURHAM e BAYS[3] nel 1976 per "riscaldare il generatore".

Inizializzazione:

1. Tramite LCG ottieni la collezione di valori $\{r_i\}_{i=0,\dots,k}$
2. Si alloca una tabella $T[0,\dots,k-1]$ e assegna $\forall\, i = 0,\dots,k-1 \quad r_i \mapsto T[i]$;
3. Si assegna $Z = r_k$;
4. Assegna $i = 0$

Passo ricorsivo: per $i \in \{1, 2, \dots n\}$

- Genera un valore r_i successivo.
- Utilizza il valore Z in input per generare un intero uniforme j: $0 \leq j < k$ che va a selezionare in modo uniforme un elemento della tabella T
- Assegna $Z = T[j]$: rilascia il valore del passo ricorsivo precedente nella posizione j
- Assegna $T[j] = r_i$
- Assegna $i = i + 1$;

1.2.2 Generatore *tipo "RANDU"*

Adottato dall'IBM nel 1968 e tristemente noto per essere un cattivo generatore, anch'esso è un generatore puramente moltiplicativo.

$$a = 2^{16} + 3 = 65\,529 \qquad m = 2^{31} = 2147483648 \qquad b = 0;$$

È noto per avere forti correlazioni dei dati effettuando l'analisi spaziale della distribuzione. Cercherò di riprodurre i risultati in [1, p. 284].

1.2.3 Generatore *tipo "L'Ecuyer"*

Per situazioni in cui sono richieste sequenze di numeri pseudo-casuali con periodo maggiore, L'ECUYER [1, § 12.1] ha suggerito un modo per combinare due sequenze ottenute tramite LCG con diversi periodi per ottenere una nuova sequenza il cui periodo è il minimo comune multiplo dei due periodi.

L'idea di base è semplicemente di sommare le due sequenze, modulo il modulo di entrambi.

Per evitare l'overflow si preferisce sottrarre invece che aggiungere e quindi aggiungere la quantità $m - 1$ per evitare valori negativi.

Le specifiche dei due generatori sono:

$$\text{L'Ecuyer tipo 1: } m_1 = 2147483563,\ a_1 = 40014,\ q_1 = 53668,\ r_1 = 12211;$$
$$\text{L'Ecuyer tipo 2: } m_2 = 2147483399,\ a_2 = 40692,\ q_2 = 52774,\ r_2 = 3791;$$

I periodi $T_1 = m_1 - 1$ e $T_2 = m_2 - 1$ presentano fattorizzazioni:

$$T_1 = m_1 - 1 = 2 \times 3 \times 7 \times 631 \times 81031;$$
$$T_2 = m_2 - 1 = 2 \times 19 \times 31 \times 1019 \times 1789;$$

condividono solo il fattore 2, dunque il periodo dei due generatori combinati risulta essere $T = 2,30 \times 10^{18}$.

1.3 Registri a Scorrimento

Una categoria di generatori di numeri pseudocasuali[1, § 11.5] basata su di un diverso paradigma è quello dei *registri a scorrimento*. Il cuore del funzionamento sta nell'operare direttamente sulla rappresentazione binaria della sequenza di numeri da rendere aleatoria tramite l'operazione di XOR tra bit lontani. Consideriamo la sequenza di bit $\{R_i\}$.

Il $k-$bit aleatorio è generato secondo la regola:

$$R_k = (R_{k-b} \oplus R_{k+c})$$

dove si intende con $\oplus$ l'azione di XOR sui bit.

Valori consigliati sono $b = 24$ e $c = 55$.

Il registro a scorrimento ha bisogno di una fase di *bootstrap* (la produzione della sequenza aleatoria iniziale) che deve necessariamente appoggiarsi su di un generatore di numeri pseudocasuali (p.es. un LCG), ciò lo rende un generatore non autonomo: nel nostro caso è necessario disporre di una sequenza di almeno 55 bit pseudocasuali affinché lo XOR possa essere inizializzato.

Non ho implementato il registro a scorrimento.

1.4 Generatore di distribuzioni non uniformi

1.4.1 Tecnica di inversione

Nella precedente sezione abbiamo osservato come generare numeri pseudocasuali con distribuzione uniforme in $[0, 1]$. Essa corrisponde alla distribuzione di probabilità normalizzata

$$\mathcal{P}\left(x \in (x;\, x+dx)\right) = p\left(x\right)\, dx = \begin{cases} dx & 0 < x < 1 \\ 0 & \text{altrimenti} \end{cases}$$

Immaginiamo [1, § 11.6][2, § 7.2] ora di comporre $x \mapsto y = y\left(x\right)$, con x variabile aleatoria con distribuzione uniforme. La distribuzione di probabilità di y, grazie alla relazione funzionale

$$\mathcal{P}\left(y \in \left(y\left(x\right);\, y\left(x\right) + dy\right)\right) = \mathcal{P}\left(x \in (x;\, x+dx)\right) \implies \left|p\left(y\right)\, dy\right| = \left|p\left(x\right)\, dx\right|$$

ci permette di definire:

$$p\left(y\right) = p\left(x\right)\frac{dx}{dy} \tag{1.2}$$

Sia $P\left(y\right)$la primitiva di $P\left(y\right)$: $\frac{dp}{dy} = P'\left(y\right) = p\left(y\right)$. Allora si ha, considerando che $p\left(x\right) = 1$

$$\frac{dP}{dy}dy = dx \implies P\left(y\right) = x. \tag{1.3}$$

Ne consegue che, assegnata una variabile aleatoria di distribuzione uniforme $\left(x, p\left(x\right)\right)$, è possibile ottenere una variabile aleatoria di distribuzione arbitraria $\left(y, p\left(y\right)\right)$ tramite la mappa

$$\left(x, p\left(x\right)\right) \mapsto \left(y = P^{-1}\left(x\right), p\left(y\right)\right) \tag{1.4}$$

Esempio 1. sia $\{(y, p\left(y\right))\}$ tale che $p\left(\lambda;\, y\right) = \frac{e^{-y/\lambda}}{\lambda}$. Sappiamo che $\int p\left(\lambda;\, y\right)\, dy = \int \frac{e^{-y/\lambda}}{\lambda}\, dy = e^{-y/\lambda} = P\left(y\right)$. Definendo dunque l'equazione $x = e^{-y/\lambda}$ si ottiene che

$$y = -\lambda \cdot \ln\left(x\right). \tag{1.5}$$

1.4.1.1 Caso particolare: trasformata di Box-Müller

Consideriamo il caso specifico della distribuzione gaussiana [1, § 11.6][2, § 7.2].
In tal caso con il metodo precedente si dovrebbe utilizzare la trasformazione:

$$\left(x, p\left(x\right)\right) \mapsto \left(y = P^{-1}\left(x\right), p\left(y\right)\right)$$

dove $p(y) = \frac{1}{\sqrt{2\pi}} e^{-y^2/2}$.
Sappiamo che $p(y)$ non ammette una funzione elementare come primitiva e che la funzione speciale ad essa associata è la $\mathrm{Erf}(x)$. Per aggirare il problema estendiamo al piano cartesiano e sostituiamo alla relazione 1.2 la seguente relazione

$$p(y_1, y_2) = p(x_1, x_2) \left| \frac{\partial x_i}{\partial y_j} \right| \tag{1.6}$$

Si ha l'estensione della distribuzione normale

$$\frac{1}{\sqrt{2\pi}} e^{-y^2/2} \mapsto \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(y_1^2 + y_2^2\right)} \tag{1.7}$$

Effettuiamo il diffeomorfismo che mappa le coordinate cartesiane rettangoli nella rappresentazione polare, in particolare esponenziando poi il raggio (è una funzione monotòna, si conserva l'iniettività):

$$\begin{cases} (y_1, y_2) \mapsto \zeta = y_1^2 + y_2^2 \mapsto e^{-\frac{1}{2}\left(y_1^2 + y_2^2\right)} & \equiv x_1 \in [0, 1] \\ (y_1, y_2) \mapsto \frac{\theta}{2\pi} = \frac{1}{2\pi} \arctan\left(y_2 / y_1\right) & \equiv x_2 \in [0, 1] \end{cases}$$

Lo jacobiano della trasformazione è dato da

$$\left| \frac{\partial x_i}{\partial y_j} \right| = \left[\frac{1}{\sqrt{2\pi}} e^{-y_1^2/2} \right] \left[\frac{1}{\sqrt{2\pi}} e^{-y_2^2/2} \right] \tag{1.8}$$

Ripercorrendo indietro il percorso prima compiuto per ottenere le funzioni inverse da utilizzare nell'algoritmo si ha

$$x_1 = e^{-\frac{1}{2}\left(y_1^2 + y_2^2\right)} \mapsto -2\ln(x_1) = y_1^2 + y_2^2 = \zeta \tag{1.9}$$

$$x_2 = \frac{1}{2\pi} \arctan\left(y_2 / y_1\right) \mapsto \tan(2\pi x_2) = y_2 / y_1 \tag{1.10}$$

identificando $\tan(2\pi x_2) = y_2/y_1 = \frac{\zeta \cos(2\pi x_2)}{\zeta \sin(2\pi x_2)}$ si ottiene:

$$\begin{cases} y_1 = \sqrt{-2\ln(x_1)} \cos(2\pi x_2) & \in [0, 1] \\ y_2 = \sqrt{-2\ln(x_1)} \sin(2\pi x_2) & \in [0, 1] \end{cases} \tag{1.11}$$

che è l'espressione usata per fini computazioni nella generazione di numeri aleatori definiti su popolazione gaussiana. Questo metodo è noto come *trasformata di Box-Müller*. Si scarta uno

dei due numeri per le possibili correlazioni tra i due risultati.

Il metodo polare di Marsaglia. Si può evitare la chiamata a funzioni trigonometriche, onerose dal punto di vista computazionale, utilizzando una piccola modifica introdotta da Marsaglia[4]:

1. Si genera una coppia di numeri casuali (u, v) e se ne fa il quadrato $s^2 = u^2 + v^2$.
2. Fin quando non si verifica $0 < s < 1$ itera tale generazione.
3. È possibile scegliere il parametro s come una nuova variabile aleatoria distribuita uniformemente in $(0, 1)$.
4. Con il valore appena ottenuto si definiscono le funzioni coseno e seno come rapporto tra u (o v) e s.

Possiamo sostituire dunque alle (y_1, y_2) del generatore di variabili distribuite normalmente la seguente coppia di funzioni:

$$\begin{cases} y_1 = \sqrt{-2\ln(s)}\frac{u}{\sqrt{s}} = u \cdot \sqrt{-2\frac{\ln(s)}{s}}; \\ y_2 = \sqrt{-2\ln(s)}\frac{v}{\sqrt{s}} = v \cdot \sqrt{-2\frac{\ln(s)}{s}}; \end{cases} \tag{1.12}$$

1.4.2 Tecnica di rigetto (Hit or Miss)

Questa tecnica [2, § 7.3], più generale e potente, permette di ottenere valori distribuiti secondo una generica distribuzione di probabilità $p(x)$. Essa si basa sul seguente algoritmo:

- Definire un grafico della distribuzione di probabilità dato dall'insieme delle coppie $(x,\ p(x))$ in una certa regione $A \subset \mathbb{R}$.
- Definire un secondo grafico associato alla funzione $f(x)$ che ha un'area finita e che giace sempre al di sopra della distribuzione originaria. Definiremo questa funzione *di riferimento*[1].
- Con il generatore di numeri casuali riempiamo la regione sottostante la funzione di riferimento (utilizzando la sua inversa: si utilizza quindi in una prima fase la tecnica di inversione descritta precedentemente) e rigettiamo i punti che giacciono tra la funzione

[1] La scelta peggiore è quella della funzione costante $f(x) = \varepsilon + \sup_{x \in A}[f]$ dove ε è una quantità positiva arbitraria (piccola ma superiore all'errore macchina).

di riferimento e la distribuzione da utilizzare. Al contrario, i punti che soddisfano tale criterio vengono memorizzati.

- A cause dell'uniformità della distribuzione la densità dei punti x è la distribuzione desiderata.

La scelta della funzione di riferimento è cruciale per l'ottimizzazione delle prestazioni temporali dell'algoritmo.

1.4.2.1 Esempio: Distribuzione Gamma

La distribuzione Gamma (in una forma meno gpuò essere parametrizz

$$p(\alpha;\, x) = \frac{x^{\alpha-1}e^{-x}}{\Gamma(\alpha)}; \quad \alpha, x \in \mathbb{R}^+,\ \Gamma(\alpha) = \int_0^\infty dt\, t^{\alpha-1}e^{-t} \tag{1.13}$$

Una funzione di riferimento utile per grandi α è data dalla distribuzione Lorenziana $f(x)$, generalizzata in modo da avere sufficienti gradi di libertà per effettuare una riscalatura tale da renderla sempre maggiore alla distribuzione.

$$f(x) = \frac{1}{\pi}\left(\frac{1}{1+x^2}\right) \sim \left(\frac{c_0}{1+(x-x_0)^2/a_0^2}\right) \mapsto p^{-1}(y) = x = a_0 \tan(\pi y) + x_0 \tag{1.14}$$

dove y è un elemento di una sequenza generata da un assegnato LCG.

Riporterò nelle applicazioni un algoritmo tratto dalle Numerical Recipes che si basa su questo esempio.

Capitolo 2

Implementazione in C di algoritmi basati su generatori di numeri aleatori.

È stata proposta in aula la seguente esercitazione:

Dopo aver realizzato un insieme di generatori di numeri pseudo-casuali, si richiede di risolvere i seguenti compiti:

Esercizio 1. Valutare il periodo di un generatore.

Esercizio 2. Costruire tramite metodo *hit-or-miss* una distribuzione aleatoria non uniforme.

Esercizio 3. Costruire tramite metodo *di inversione* una distribuzione aleatoria non uniforme.

Esercizio 4. Implementare una variabile aleatoria con *distribuzione gaussiana.*

Questi esercizi sono preliminari alla risoluzioni dei seguenti tre problemi:

Problema 1. Calcolo aree col metodo hit-or-miss e volume della sfera unitaria.

Problema 2. Studio del moto Browniano.

Problema 3. Verifica Teorema del Limite Centrale.

Tali risultati devono essere fatti precedere dalle istruzioni per far funzionare i sorgenti.

Ho caricato la cartella dei sorgenti associata alla tesina su GitHub all'indirizzo `https://github.com/mario-ambrosino/random`, i sorgenti non saranno riportati integralmente nel report per evitare di appesantire ulteriormente (cfr. **Tabella 2.1**).

Aggiungerò estratti di codice laddove significativo.

A causa degli errori di programmazione passati, questa volta ho costruito un unico script di compilazione, `./make.sh`, la cui esecuzione compila tutti i sorgenti necessari per lo svolgimento degli esercizi e che non dipende dalla cartella utente personale.

I risultati delle esecuzioni (immagini incluse) dei programmi sono tutti contenuti nella cartella `./dataset/`, gli eseguibili in `./exe/`.

Suggerirei, dopo l'esecuzione di `make.sh` di eseguire gli script nel seguente ordine:

`make_sequences.sh:` genera le sequenze 1-dim con i metodi descritti nel **Capitolo 1**. Ha come output

- `Uniform.dat:` distribuzione uniforme, ottenuta tramite LCG tipo *L'Ecuyer*;
- `Exp.dat:` distribuzione esponenziale (con parametro $\lambda = 100$, tramite metodo di inversione; **(Esercizio 3)**
- `Gamma.dat:` distribuzione Gamma, tramite metodo *Hit-or-Miss*; **(Esercizio 2)**
- `Normal.dat:` distribuzione Normale tramite metodo di *Box-Müller* (ma è disponibile anche la generazione con metodo di *Marsaglia*, non implementata nella realizzazione di questa sequenza ma funzionante) **(Esercizio 4)**

`make_uniform_vect.sh:` genera le sequenze 2-dim (3-dim) con i generatori uniformi presentati e ne valuta i tempi relativi (rapporto relativo dei cicli macchina rispetto a `rand()`)

- `R21(R31).dat:` Minimal Standard **RAN0** diretto
- `R22(R32).dat:` Minimal Standard **RAN0** con *Schrage*
- `R23(R33).dat:` Minimal Standard **RAN1** con *Schrage* e Shuffling alla *Bays-Durham*
- `R24(R34).dat:` Generatore Tipo **L'Ecuyer** (usato anche per i generatori non uniformi)
- `R25(R35).dat:` Generatore Tipo **RANDU**
- `R26(R36).dat:` Generatore `rand()` interno del compilatore GCC .

make_histograms.sh: Genera gli istogrammi delle distribuzioni generate tramite `make_sequences.sh`; sono i file con il prefisso `H_*`. Plottate con `gnuplot`, queste strutture dati forniscono l'istogramma delle corrispondenti funzioni (cfr. **Figura** 2.1)

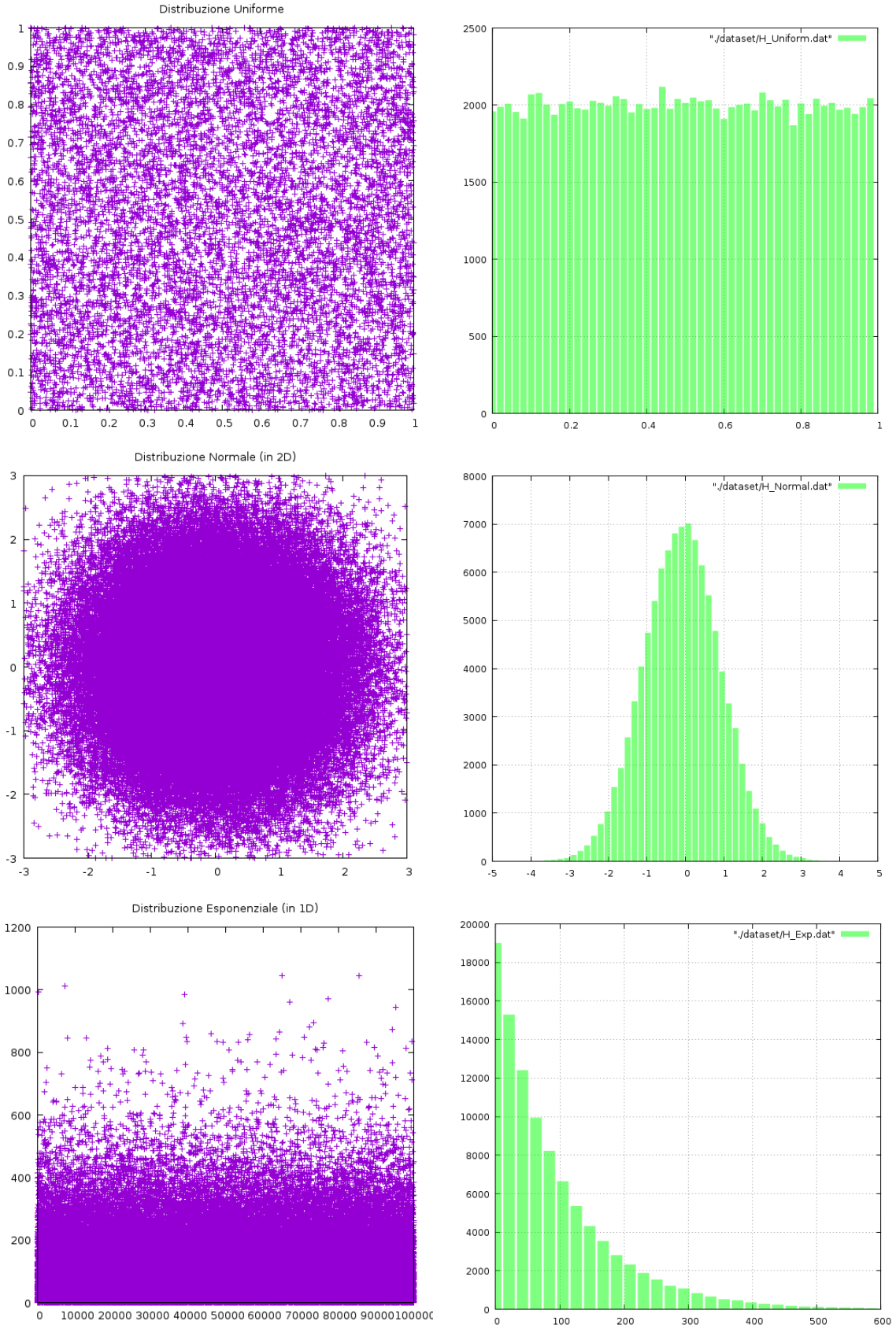

Figura 2.1: Esempi di sequenze ottenute tramite la routine `make_sequences.sh`

Tipo	Nome	Descrizione
Libreria	`./include/lib_data.c`	*Struttura dati e strumenti di controllo sui dati*
Libreria	`./include/lib_random.c`	*Libreria Generatori Random*
Libreria	`./include/lib_analysis.c`	*Libreria analisi di sequenze*
Cartella Listati	`./sources/analysis/`	*Implementazioni strumenti di analisi*
Cartella Listati	`./sources/makers/`	*Implementazioni di generatori pseudo-random.*
Cartella Listati	`./sources/problems/`	*Risoluzione dei problemi proposti*

Tabella 2.1: Tabella delle librerie e delle cartelle rilevanti dal punto di vista numerico.

2.1 Test sui generatori

Abbiamo parlato di RANDU e della sua mancata aleatorietà a causa dell'esistenza di iperpiani su cui si addensano gli elementi della sequenza generata da RANDU stesso.

Per verificare questa proprietà, generiamo triple di numeri casuali tramite RANDU e poi tramite `gnuplot` rappresentiamo in 3 dimensioni (cfr. **Figura 2.2**). Le correlazioni sono evidenti a vista.

Questo è un metodo qualitativo, forniamo di seguito alcuni semplici metodi quantitativi per giudicare sulla bontà di un generatore.

2.1.1 Verifica momenti delle distribuzioni associate ai generatori.

Naturalmente i metodi quantitativi da utilizzare risulteranno essere di tipo statistico. L'analisi più immediata è la verifica della corrispondenza tra stima e valori teorici delle distribuzioni associati ai generatori di numeri casuali. Ho svolto questa verifica sia direttamente che tramite costruzione di istogramma e i risultati risultano essere privi di discrepanze significative (con eccezione della distribuzione Gamma). Ho evitato di riportare qui le stime quantitative. Invito alla lettura della libreria di analisi `./include/lib_analysis.c` alla esecuzione dei programmi (o dello script) sottostanti:

```
usr@dom:~$ ./exe/random_analysis (/path/to/input/file) (int N) (int B) (int T)
usr@dom:~$ ./exe/reduced_analysis (/path/to/input/file)
usr@dom:~$ ./check_period.sh
```

Nel primo c'è sia una routine di calcolo di media e σ della distribuzione letta tramite il parametro `name_input_file` che il calcolo del χ^2 della distribuzione ottenuta dal popolamento di un insieme di bin tramite LCG uniforme. I parametri `N,B,T` sono rispettivamente il numero di valori pseudocasuali uniformi estratti tramite LCG, il numero di bins dell'istogramma e il numero di test effettuati.

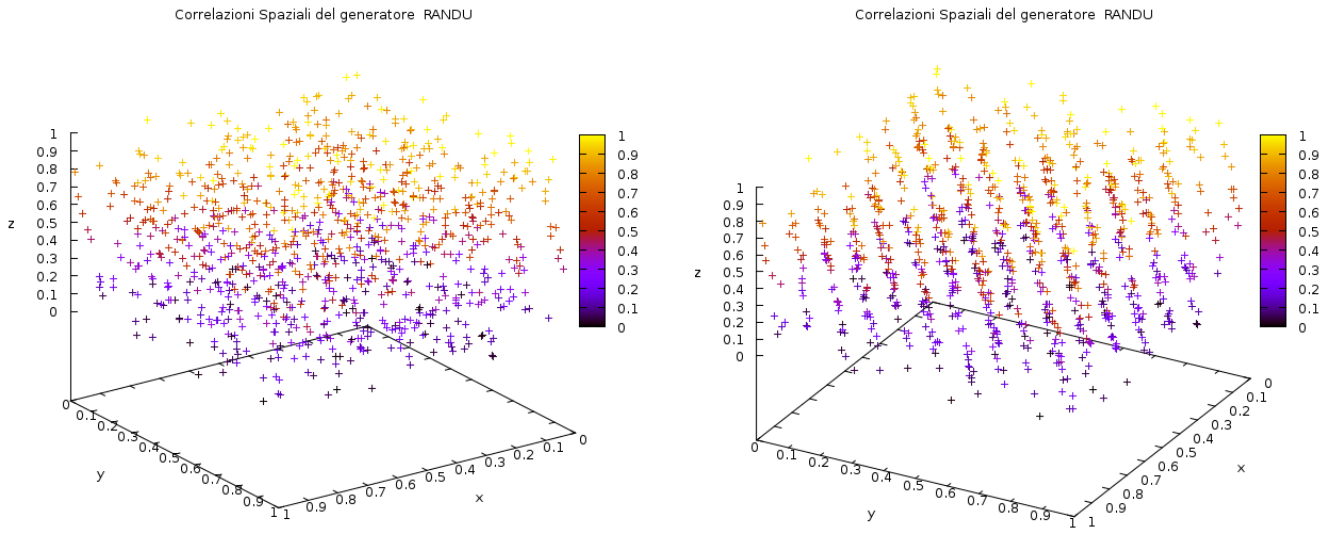

Figura 2.2: Rappresentazione 3D di triple generate tramite RANDU da due punti di vista diversi.

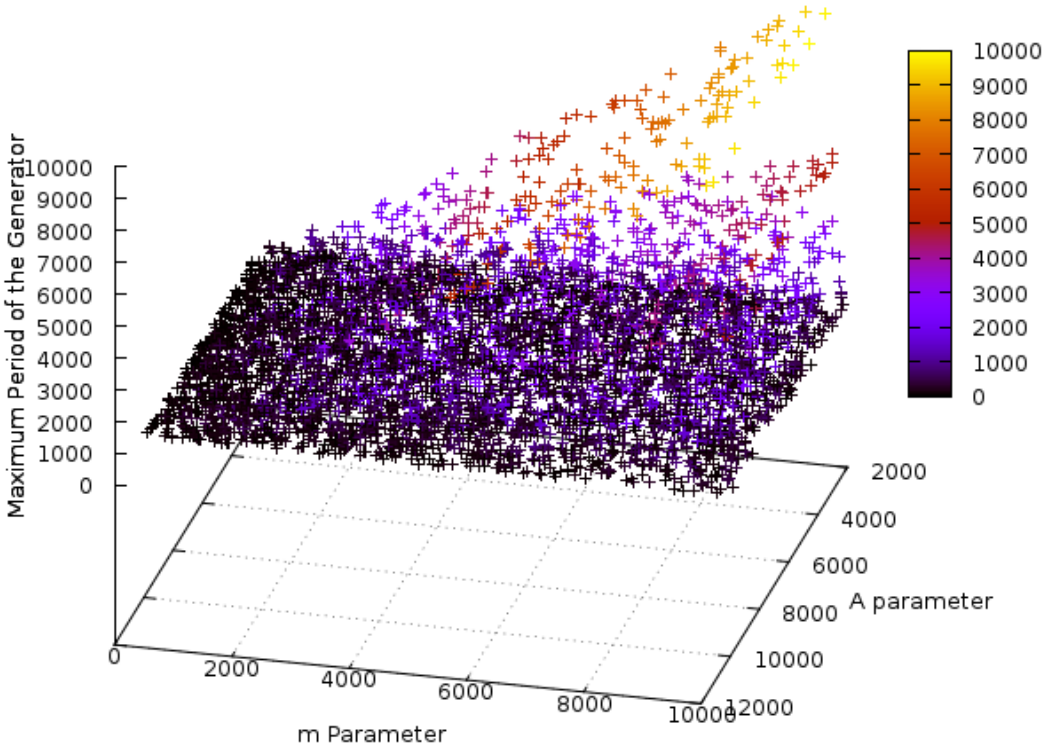

Figura 2.3: Risultato di `./script_period.sh`. Si noti la composizione in piani

Nel secondo listato c'è una versione ridotta, che effettua il solo calcolo di media e σ ed è pensato per lo scripting: il suo output è più semplice del listato precedente e viene utilizzato in una *pipe* all'interno degli script che ne fanno uso.

2.1.2 Verifica Massimo Periodo

Una prima richiesta **(Esercizio 1)** è stata quella di verificare in modo numerico il massimo periodo dei generatori LCG al variare dei parametri (a, b, m).

Per risolvere tale esercizio ho realizzato due listati e uno script associato. Come al solito, suggerisco l'esecuzione del solo script:

`period.c` in `https://github.com/mario-ambrosino/random/blob/master/sources/analysis/`

`period.c`: ricerca il periodo vero di un generatore e ammette come input gli argomenti (A, B, M) separati da uno spazio.

```
usr@dom:~$ ./exe/period.exe (int A) (int B) (int M)
usr@dom:~$ ./check_period.sh
```

sample_period.c in `https://github.com/mario-ambrosino/random/blob/master/sources/analysis/sample_period.c`: estende la funzionalità del primo script, campionando lo spazio degli stati del LCG tramite estrazione casuale uniforme di M triple di parametri (a, b, m) da un reticolo tridimensionale finito: $a, b, m \in \{1, \ldots, N\}$ con $N = 10000$.

```
usr@dom:~$ ./exe/sample_period.exe (int M)
usr@dom:~$ ./script_period.sh
```

Riporto in figura un risultato visualizzato in `gnuplot` dopo aver scelto[1] $M = 6100$. Il risultato è notevole poiché emergono dei piani su cu si assestano i valori ottimali dei generatori: in particolare il piano superiore è quello in cui il periodo è (quasi) massimo.

Non analizzo il periodo dei generatori riportati poiché non riesco a trovare un metodo per poter analizzarne il periodo senza allocare un vettore di dimensioni pari al modulo. La variante proposta in `sample_period.c` non era richiesta, ma estende le funzionalità di `period.c` che intende risolvere l'esercizio proposto.

[1] C'è da dire che in realtà il numero di valori in output è inferiore a quello dell'input: in caso di errore il programma arresta la generazione di numeri; la riesecuzione del programma aggiunge al file dei dati estratti i nuovi file, quindi in realtà 6100 è il numero di dati ottenuto dopo un certo numero di esecuzioni del programma. Naturalmente non ricordo questo numero.

Unif. $\mathbb{E}(X) = 0.5$	$\langle X \rangle$	$\langle \sigma_X^2 \rangle$	$\sigma^2_{\langle X \rangle}$
2000	0.4958	0.003	0.0004
4000	0.4991	0.003	0.0004
6000	0.4848	0.004	0.0005
8000	0.4839	0.004	0.0006

Norm. $\mathbb{E}(X) = 0$	$\langle X \rangle$	$\langle \sigma_X^2 \rangle$	$\sigma^2_{\langle X \rangle}$
2000	0.001	0.12	0.005
4000	0.045	0.13	0.005
6000	0.042	0.15	0.005
8000	0.006	0.12	0.006

Tabella 2.2: Tabella risultati dello script **recurse.sh** per distribuzione uniforme ed esponenziale.

Exp. $\mathbb{E}(X) = 137$	$\langle X \rangle$	$\langle \sigma_X^2 \rangle$	$\sigma^2_{\langle X \rangle}$
2000	145	$12 \cdot 10^3$	$6 \cdot 10^1$
4000	153	$6 \cdot 10^3$	10^2
6000	144	$12 \cdot 10^3$	10^2
8000	144	$12 \cdot 10^3$	10^2

Gamm. $\mathbb{E}(X) = 137$	$\langle X \rangle$	$\langle \sigma_X^2 \rangle$	$\sigma^2_{\langle X \rangle}$
2000	597	$107 \cdot 10^3$	$1.5 \cdot 10^3$
4000	588	$116 \cdot 10^3$	$1.4 \cdot 10^3$
6000	611	$117 \cdot 10^3$	$1.5 \cdot 10^3$
8000	563	$112 \cdot 10^3$	$1.7 \cdot 10^3$

Tabella 2.3: Tabella risultati dello script **recurse.sh** per distribuzione esponenziale e Gamma.

2.1.3 Problema: Verifica teorema del Limite Centrale

Possiamo includere la soluzione di questo problema all'interno degli strumenti di analisi della bontà di un generatore. Il nostro scopo è verificare numericamente la validità del seguente teorema, adattato da [5, § 4.8.8] per i nostri scopi:

Teorema 3. *(del Limite Centrale) Sia $\{x_i\}$ una $N-$sequenza di variabili aleatorie indipendenti. Sia ogni x_i distribuito con valore medio μ e varianza finita σ^2. Allora la funzione $\langle x \rangle = \sum_{i=i}^{N} x_i/N$ è una variabile aleatoria con distribuzione normale $N(\mu; \sigma^2/N)$.*

Traduciamo tale verifica nell'algoritmo seguente: assegnato un generatore di variabili pseudo-aleatorie G ($M = 50$ e $N \in \{2 \cdot 10^3, 4 \cdot 10^3, 6 \cdot 10^3, 8 \cdot 10^3\}$), si applicano le seguenti istruzioni:

1. Genera N volte una $G-$sequenza aleatoria $\{x_i\}_{i=1}^{M}$;
2. Valuta la *media* $\mu_j = \mu\left(\{x_i\}_{i=1}^{M}\right)$; $j = 1 \ldots N$.
3. Genera *l'istogramma della media* μ_j in 50 *bins* tra il valore minimo e il valore massimo della distribuzione.
4. Valuta *media e varianza dell'istogramma* per dedurre le proprietà della distribuzione della variabile aleatoria μ_j.

Non ho scritto un listato specifico in C per la soluzione a questo problema, ho voluto risolverlo tramite i listati già presentati usando più shell script (cfr. **Algoritmi 1 e 2**) per verificare la

```
    #!/bin/bash
    ## $1: N: Num. of tests
    ## $2: M: Num. of samples for each test
# Genera N test con M estrazioni di numeri casuali da un generatore
# (in questo caso è il generatore uniforme)
for i in $‘seq 0 1 $1‘; do
./exe/uniform_1D_maker.exe $2 ./dataset/central/input/$i.dat
done
# calcola media e sigma per i file di input.
for j in $‘seq 0 1 $1‘; do
./exe/reduced_analysis.exe ./dataset/central/input/$j.dat >>
                        [./dataset/central/output.dat
done
# splitta i dati in due dataset 1-dimensionali
./exe/split.exe ./dataset/central/output.dat ./dataset/central/output_media.dat
 [./dataset/central/output_stdev.dat
# calcola gli istogrammi delle due sequenze ottenute.
./exe/histogram2.exe ./dataset/central/output_media.dat 30
                        [./dataset/central/H_media.dat
./exe/histogram2.exe ./dataset/central/output_stdev.dat 30
                        [./dataset/central/H_stdev.dat
```

Algorithm 1: Script per la risoluzione del **Problema 3:** Teorema del Limite Centrale

```
#!/bin/bash
    for i in $‘seq 1000 1000 10000‘; do
    ./central_limit_theorem.sh $i 100
mv ./dataset/central/H_media.dat ./dataset/central/average/m_$i.dat
mv ./dataset/central/H_stdev.dat ./dataset/central/average/s_$i.dat
./exe/split.exe ./dataset/central/average/m_$i.dat ./dataset/central/split/x_$i.dat
                                    [./dataset/central/split/y_$i.dat
./exe/split.exe ./dataset/central/average/s_$i.dat ./dataset/central/split/ex_$i.dat
                                    [./dataset/central/void
./exe/reduced_analysis.exe ./dataset/central/split/x_$i.dat
                                    >> ./dataset/central/average.dat
./exe/reduced_analysis.exe ./dataset/central/split/y_$i.dat
                                    >> ./dataset/central/st_dev.dat
./exe/reduced_analysis.exe ./dataset/central/split/ex_$i.dat
                                    >> ./dataset/central/est_dev.dat
rm ./dataset/central/void
```

Algorithm 2: Script di ricorsione per la risoluzione del **Problema 3:** Teorema del Limite Centrale

versatilità dell'integrazione tra programmi basati su linguaggi interpretati e compilati. L'uso delle script shell è diventato abituale dopo l'esperienza con la relazione riguardante gli algoritmi di interpolazione. Gli script usati sono i seguenti:

central_limit_theorem.sh: in `https://github.com/mario-ambrosino/random/blob/master/central_limit_theorem.sh`: è il codice riportato in **Algoritmo 1**: assegnati i due valori M, N prima descritti, realizza l'algoritmo associato al Teorema del Limite Centrale.

recurse_average.sh: in `https://github.com/mario-ambrosino/random/blob/master/recurse.sh`: è il codice riportato in **Algoritmo 2**, miglioramento del suo predecessore `recurse.sh`. Esso automatizza la verifica del teorema del limite centrale nell'intervallo per N definito nell'insieme già citato $N \in \{2 \cdot 10^3, 4 \cdot 10^3, 6 \cdot 10^3, 8 \cdot 10^3\}$.

Il risultato dell'esecuzione dello script `recurse_average.sh` è dato dalle quattro **Tabelle 2.2 e 2.3,** da confrontarsi con i quattro grafici in **Figura 2.4**. In tutti e quattro i casi si è osservata fin dalla scelta $N = 2000$ una distribuzione associata all'istogramma del valor medio generato dal generatore di tipo gaussiano. Significativa l'asimmetria dell'istogramma associato alla funzione Gamma e la deviazione rispetto al valore atteso.

2.2 Applicazioni

Qui introduciamo alcune semplici soluzioni basate sull'uso dei generatori pseudo-casuali degli altri due problemi proposti in aula.

Il primo, la valutazione di misure di aree e volumi tramite il metodo di rigetto, risulta essere il passo iniziale per quella che poi risulterà essere la tecnica di integrazione numerica tramite metodo Montecarlo.

Il secondo invece, quello dei camminatori aleatori, è lo strumento base delle simulazioni numeriche di fenomeni stocastici.

2.2.1 Metodo del rigetto per la valutazione numerica di misure

Con la stessa tecnica usata per la valutazione della distribuzione Gamma, si vuol valutare l'area associata alla soluzione di una data disequazione algebrica. Per fare ciò si usa l'algoritmo seguente:

1. Si generano N elementi $\{u_i\}_{i=1}^{N}$di una sequenza pseudocasuale a valori in una regione D che comprende il dominio di verità della disequazione algebrica (per esempio, sia X_R :

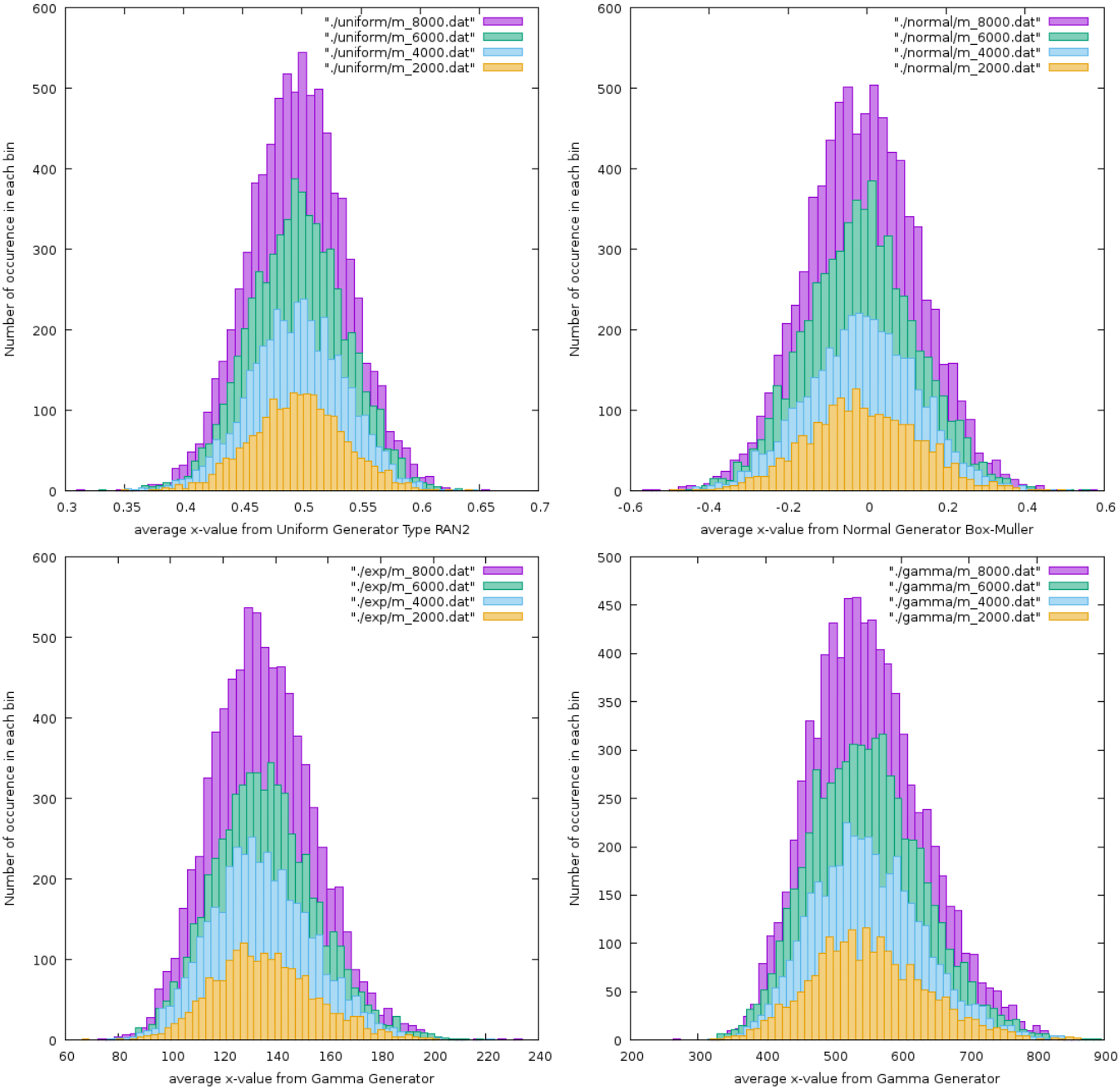

Figura 2.4: Istogrammi risultato dello script usando i quattro generatori. In senso orario a partire da in alto a sinistra: istogramma per distribuzione uniforme, normale, esponenziale, gamma. L'alternarsi di istogrammi è condotto con parametri $N \in \{2 \cdot 10^3, 4 \cdot 10^3, 6 \cdot 10^3, 8 \cdot 10^3\}$ e $M = 50$. Il parametro addizionale $\lambda = 137$ è stato assegnato per entrambe le distribuzioni esponenziale e gamma.

```
#include "./include/lib_random.c"
/*
##  Misure con metodo Hit or Miss
##  Use first argument as Max_Points
##  Use second argument as radius
*/
int main (int argc, char **argv)  {
int i=0;
int Max_Points = atoi(argv[1]);
double radius = atof(argv[2]);
double x, y, z, k, A, V, Atrue, Vtrue;
RANDOM seed = -time(0);

k = 0;
for (i = 1; i < Max_Points; i++) {
x = radius * (2 * RAN2(&seed) - 1);
y = radius * (2 * RAN2(&seed) - 1);
z = radius * (2 * RAN2(&seed) - 1);
if (x*x + y*y + z*z < radius*radius) k++;
}
V = 8 * radius * radius * radius * k / Max_Points;
Vtrue = 4.0 * M_PI * radius * radius * radius / 3;
printf("\nVolume for the sphere (radius = %lf)
vs true volume = %lf vs %lf", radius, V, Vtrue);

k = 0;
for (i = 1; i < Max_Points; i++) {
x = radius * (2 * RAN2(&seed) - 1);
y = radius * (2 * RAN2(&seed) - 1);
if (x*x + y*y < radius*radius) k++;
}
A = 4 * radius * radius * k / Max_Points;
Atrue = M_PI * radius * radius;
printf("\nArea for the circle (radius = %lf)
vs true area = %lf vs %lf", radius, A, Atrue);
return EXIT_SUCCESS;
}
```

Algorithm 3: Listatoin C per la risoluzione del **Problema 2:** Calcolo di aree e volumi tramite Hit or Miss

$\{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| \leq R\}$ la regione da misurare e $D = [-R, R] \times [-R, R]$ la regione in cui giacciono i valori della sequenza).

2. Si valuta se $u_i \in X_R$. Se ciò è vero, si incrementa di un'unità l'intero T.

3. Il valore dell'integrazione può essere valutato come il rapporto T/R.

Bisognerebbe inoltre considerare il trattamento dell'errore commesso con tale stima, ma riprenderò questo algoritmo nella terza tesina, dedicata specificatamente all'integrazione numerica, in cui confronteremo le prestazioni degli algoritmi strettamente numerici con quelli di tipo Montecarlo.

Il listato utilizzato per implementare tale algoritmo è dato dall'**Algoritmo 3**. Un tipico output di tale listato è dato da:

```
Volume for the sphere (radius = 5.000000) vs true volume = 523.594400 vs 523.598776;
Area for the circle (radius = 5.000000) vs true area = 78.530400 vs 78.539816;
```

2.2.2 Moto Browniano

Come argomento finale riporto la realizzazione elementare del moto di una particella con dinamica definita tramite un processo di diffusione in cui la dinamica è definita tramite forze microscopiche impulsive, isotrope, scorrelate e con distribuzione di tipo gaussiano. Questo processo stocastico è stato descritto come "moto browniano" ed è alla base della teoria dei sistemi dinamici stocastici.

Come semplice verifica del funzionamento dello script vogliamo verificare che il cammino medio del random walk abbia un andamento del tipo

$$\langle X \rangle \approx \sqrt{T} \iff \langle X^2 \rangle \approx T$$

Questo ha delle motivazioni teoriche che possono essere dimostrate [1, § 12.1] con semplicità per il caso 1-dimensionale

Dimostrazione. Consideriamo di effettuare M Random Walk (RW) di N passi. Il passo ricorsivo elementare è dato da:

$$X(T+1) = X(T) + G_N \tag{2.1}$$

dove G_N è una variabile aleatoria con distribuzione normale e $X(T)$ è il valore della variabile spaziale al passo $T-$mo. Elevando al quadrato la relazione precedente otteniamo

$$X^2(T+1) = X^2(T) + G_N^2 + 2G_N X(T) \tag{2.2}$$

Osserviamo che i risultati del generatore sono per ipotesi non correlati con il valore della posizione, pertanto si può fattorizzare il valor medio nel doppio prodotto. Effettuando quindi il valor medio della relazione precedente si ottiene

$$\left\langle X^2(T+1) \right\rangle = \left\langle X^2(T) + G_N^2 + 2G_N X(T) \right\rangle =$$

$$= \left\langle X^2(T) \right\rangle + 2\left\langle G_N X(T) \right\rangle + G_N^2 = \left\langle X^2(T) \right\rangle + 2\left\langle G_N \right\rangle \left\langle X(T) \right\rangle + G_N^2 \tag{2.3}$$

Poiché si può verificare che $\langle G_N \rangle = \mathbb{E}\left[X_{N(0,1)}\right] = 0$ e $\langle G_N^2 \rangle = \mathbb{E}\left[X_{N(0,1)}^2\right] = 1$ la precedente relazione conduce al risultato

$$\left\langle X^2(T+1) \right\rangle = \left\langle X^2(T) \right\rangle + 1 \implies \left\langle X^2(T) \right\rangle = T \tag{2.4}$$

□

Poichè nella pratica il calcolatore può eseguire una quantità finita "prove", bisogna supporre che l'identità $\langle X^2(T) \rangle = T$ valga con una certa approssimazione. Ci proponiamo dunque di verificare che

$$\left\langle X^2(T) \right\rangle = T \implies \sum_{j=1}^{M} X^2(T) \approx T$$

Ciò può essere svolto con un generatore LCG, in particolare con l'algoritmo di Box-Müller.

Ho realizzato due algoritmi per la risoluzione del problema con uno script associato.

`usr@dom:~$ ./exe/walker.exe (int Num_Steps) (char /path/to/output/file)`

Realizza un singolo random walk 2-dim con generatore normale Box-Müller di `Num_Steps` passi e esporta nel file in `/path/to/output/file`;
`https://github.com/mario-ambrosino/random/blob/master/sources/problems/walker.c`

`usr@dom:~$ ./exe/walker1D.exe (int Num_Steps) (int Num_Trials) (char /path/to/output) (char /path/to/average/file)`

Realizza `Num_Trials` random walk 1-dim con generatore normale Box-Müller di `Num_Steps` passi, esporta nel file in `/path/to/output/file` l'ultimo cammino generato e in `/path/to/average/f`

la sequenza del quadrato mediata sugli `Num_Trials` random walk generati.
`https://github.com/mario-ambrosino/random/blob/master/sources/problems/walker.1Dc`

`usr@dom:~$ ./make_walk.sh` è uno script che esegue 10 cammini 2-dimensionali con `walker.exe` e una singola volta `walker1D.exe`. Utilizzato per generare i dati riportati nella relazione.

I risultati dell'**Algoritmo 5** sono riportati nelle **Figure 2.5 e 2.6**, essi sembrano confermare quanto prima previsto. Il grafico in **Figura 2.6** è rappresentato in scala log-log ed ha pendenza unitaria: ciò verifica l'asserto proposto.

```
#!/bin/bash
./exe/walker1D.exe 10000 1000 1 ./dataset/walkings/aged1.dat
                                [./dataset/walkings/aged2.dat
for i in $‘seq 1 1 10‘; do
   ./exe/walker.exe 10000 1 ./dataset/walkings/walk_$i.dat
   done
```

Algorithm 4: Script `make_walk.sh` per la risoluzione del problema sul moto Browniano

```
#include "./include/lib_random.c"
#include <sys/time.h>

int main (int argc, char **argv)  {

   int i, j;
   int Dimension = atoi(argv[1]);
   int Trials = atoi(argv[2]);
   double Amplitude = atof(argv[3]);
   data hist1 = new_data(Dimension, hist1);
   data hist2 = new_3D_data(Dimension, hist2);
   RANDOM seed;

   struct timeval tv;       # A causa della velocità di esecuzione
   gettimeofday(&tv, 0);    # è necessario avere un seed diverso
   seed = -tv.tv_usec;      # ad ogni usec (microsecondo).
   (...)
   printf("\nSeed = %Lu\n", seed);
   for (j = 0; j < Trials; j++) {
      hist1.x[0] = hist1.y[0] = 0;
      for (i = 1; i < Dimension; i++) {
         hist1.x[i] = hist2.x[i] = i;
         hist1.y[i] = hist1.y[i-1] + Amplitude*Normal_Random_1(&seed);
         hist2.y[i] += hist1.y[i] * hist1.y[i];
         }
      }
      //make average
      for (i = 0 ; i < Dimension ; i++) {
      hist2.y[i] /= Trials;
      }
      (...)
```

Algorithm 5: Frammenti di interesse del listato in C usato per la risoluzione del **Problema 3:** Simulazione del moto Browniano.

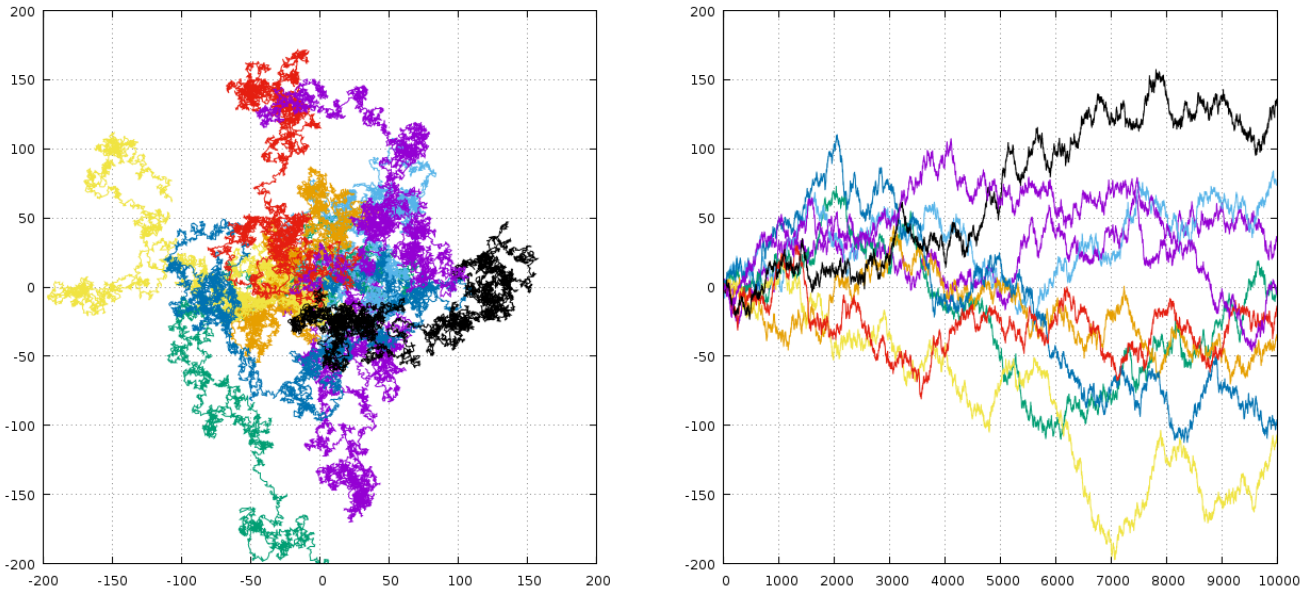

Figura 2.5: A sinistra: rappresentazione sul piano della collezione di 9 moti 2-dimensionale del RW. A destra: proiezione lungo l'asse x dei moti della curva al variare del tempo discretizzato T. Ottenuti con lo script `./make_walk.sh`.

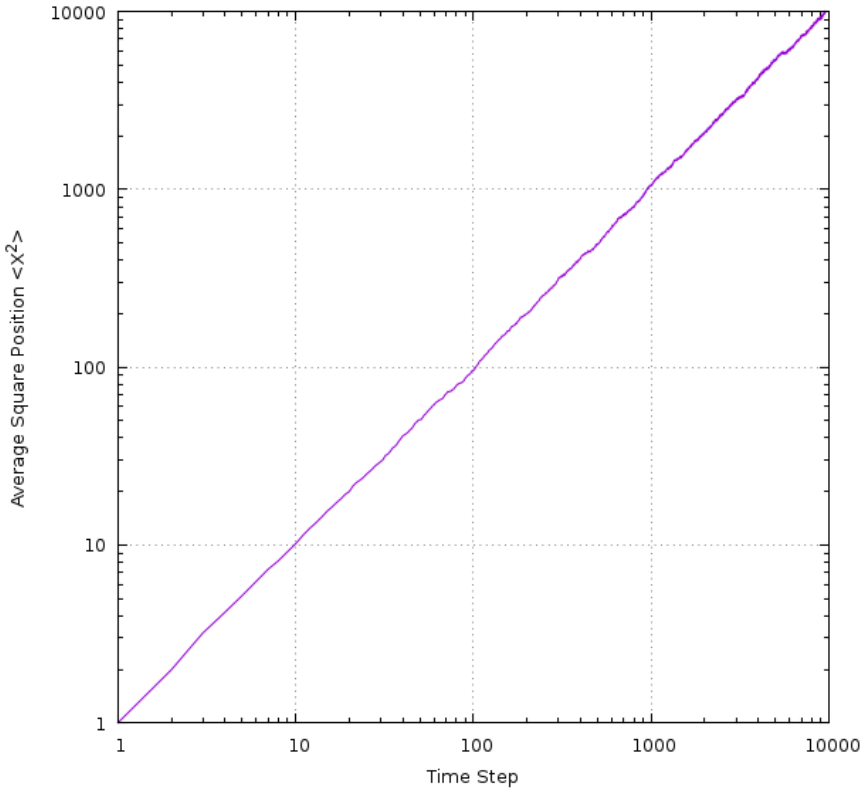

Figura 2.6: Grafico *log-log* della posizione quadratica media di 1000 RW di 10000 passi ottenuta tramite l'esecuzione dello script `./make_walk.sh`

Bibliografia

[1] L. Barone, E. Marinari, G. Organtini, F. Ricci-Tersenghi
Programmazione Scientifica
Pearson Education - (2006) - ISBN: 978-8-8719-2242-3

[2] W. Press, S. Teukolsky, W. Vetterling, B. Flannery
Numerical Recipes in C - The Art of Scientific Computing - Second Edition
Cambridge University Press - (1992)
ISBN: 0–521-43108-5

[3] C. Bays
Improving a random number generator: a comparison between two shuffling methods
Journal of Statistical Computation and Simulation, 36:1, 57-59 (1990)
DOI: 10.1080/00949659008811264

[4] G. Marsaglia, T. A. Bray
A Convenient Method for Generating Normal Variables
SIAM Review - Vol. 6, No. 3 (Jul., 1964), pp. 260-264
JSTOR: URL - `www.jstor.org/stable/2027592`.

[5] A. G. Frodesen, O. Skjeggestad, H. Tøfte
Probability and Statistics in Particle Physics
Columbia University Press - 1979
ISBN: 82-00-01906-3.