Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Triennale in Matematica

Metodo Monte Carlo e tecniche di riduzione della varianza

Relatore:
Prof. Markus Fischer

Laureando: Alessandro Poletto
Matricola: 1216925

Anno Accademico 2021/2022
16 Dicembre 2022

Indice

Introduzione

Per "metodo Monte Carlo"si intende una vasta gamma di algoritmi computazionali che si basano sulla generazione di numeri casuali per condurre simulazioni statistiche ed ottenere risultati numerici.

La nascita del metodo viene fatta risalire alla seconda metà degli anni '40, con la costruzione dei primi ordigni nucleari e dei primi computer: il problema della diffusione dei neutroni in una sfera di materiale fissile poteva essere affrontato utilizzando i metodi di campionamento statistico, grazie alle capacità di calcolo dei computer.

Successivamente, il metodo Monte Carlo ha trovato applicazione in molti altri ambiti al di fuori della fisica, come la matematica, la finanza e l'ingegneria [6] (pg 103-104).

Nel primo capitolo di questa tesi vengono presentati i risultati teorici, tra cui la legge dei grandi numeri, che sono alla base del funzionamento del metodo Monte Carlo. Quindi si passa al secondo capitolo dedicato alla trattazione dell'errore e alla costruzione di intervalli di confidenza grazie al teorema del limite centrale, concludendo con l'integrazione Monte Carlo come esempio applicativo. Con il terzo capitolo vengono introdotte le tecniche di riduzione della varianza che, proseguendo il tema iniziato con il secondo capitolo, permettono di diminuire l'errore del metodo. Infine, il quarto capitolo presenta dei risultati numerici, ottenuti utilizzando l'integrazione Monte Carlo, sulle funzioni test del Genz Package e alcuni esempi di applicazione delle tecniche di riduzione della varianza nel caso del calcolo di Pi greco.

Capitolo 1

Introduzione al metodo Monte Carlo

In questo capitolo sono riportati alcuni risultati teorici fondamentali alla giustificazione del principio del metodo MC.

1.1 Nozioni preliminari

Definizione 1.1.1. $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ si dice una *σ-algebra* sull'insieme Ω se:

- $\varnothing \in \mathcal{F}$;
- $E \in \mathcal{F} \implies E^c \in \mathcal{F}$;
- $\{E_n\}_{n\in\mathbb{N}} \subseteq \mathcal{F} \implies \bigcup_{n\in\mathbb{N}} E_n \in \mathcal{F}$.

La coppia $(\Omega, \mathcal{F})$ si dice *spazio misurabile*.

Definizione 1.1.2. Sia (X, τ) uno spazio topologico. Si definisce $\mathcal{B}(X)$ la σ-algebra generata dagli aperti di X e viene detta *σ-algebra dei boreliani*.

Definizione 1.1.3. Sia $(\Omega, \mathcal{F})$ uno spazio misurabile. Una funzione $\mu : \mathcal{F} \mapsto [0, +\infty]$ si dice una *misura* su $(\Omega, \mathcal{F})$ se:

- $\mu(\varnothing) = 0$;
- $\mu\left(\bigcup_{n\in\mathbb{N}} E_n\right) = \sum_{n\in\mathbb{N}} \mu(E_n)$, con $\{E_n\}_{n\in\mathbb{N}} \subseteq \mathcal{F}$ disgiunti a due a due.

La terna $(\Omega, \mathcal{F}, \mu)$ si dice *spazio con misura*. Nel caso in cui una misura $\mathbb{P}$ sia tale che $\mathbb{P}(\Omega) = 1$, $\mathbb{P}$ viene detta *misura di probabilità* e $(\Omega, \mathcal{F}, \mathbb{P})$ *spazio di probabilità*.

Definizione 1.1.4. Una funzione $f : (\Omega, \mathcal{F}) \mapsto (E, \mathcal{E})$ si dice $(\mathcal{F}, \mathcal{E})$-misurabile se $\forall Y \in \mathcal{E}$ si ha che $f^{-1}(Y) \in \mathcal{F}$, dove $f^{-1}(Y)$ è la controimmagine di Y tramite f. In contesto probabilistico, una funzione $X : (\Omega, \mathcal{F}) \mapsto (E, \mathcal{E})$ misurabile si dice *variabile aleatoria* (v.a.).

Definizione 1.1.5. La funzione di ripartizione di una variabile aleatoria reale X è definita da

$$F_X(x) \coloneqq \mathbb{P}(X \leq x),\ x \in \mathbb{R}.$$

Definizione 1.1.6. Sia $(\Omega, \mathcal{F}, \mathbb{P})$ uno spazio di probabilità ed $(E, \mathcal{E})$ uno spazio misurabile. Sia $X : \Omega \mapsto E$ una variabile aleatoria reale non negativa o in $L^1(\Omega)$, allora il valor medio (o valore atteso) è definito come

$$\mathbb{E}[X] = \int_\Omega X d\mathbb{P},$$

cioè l'integrale di Lebesgue rispetto alla misura $\mathbb{P}$.

Definizione 1.1.7. Sia $X \in L^2(\Omega)$. Allora per "varianza di X" si intende

$$Var(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right].$$

Definizione 1.1.8. Sia $X : \Omega \mapsto \mathbb{R}^d$ una variabile aleatoria. La funzione $\varphi : \mathbb{R}^d \mapsto \mathbb{C}$ definita da

$$\varphi_X(t) = \mathbb{E}[e^{i\langle t, X\rangle}], \;\; t \in \mathbb{R}^d,$$

è detta funzione caratteristica di X.

Definizione 1.1.9. Siano $(X_n)_{n\in\mathbb{N}}$, X variabili aleatorie reali. Allora si dice che:

- $X_n \overset{\mathbb{P}-q.c.}{\longrightarrow} X$ se $\mathbb{P}(X_n \nrightarrow X) = 0$;
- $X_n \overset{L^p}{\longrightarrow} X$, con $1 \leq p < +\infty$, se $\mathbb{E}[|X_n - X|^p] \to 0$ per $n \to \infty$;
- $X_n \to X$ in probabilità se $\forall \varepsilon > 0, \; \lim_{n\to\infty} \mathbb{P}(|X_n - X| > \varepsilon) = 0$;
- $X_n \to X$ in distribuzione se, $\forall f$ continua e limitata, $\lim_{n\to\infty} \mathbb{E}[f(X_n)] = \mathbb{E}[f(X)]$.

Definizione 1.1.10. Sia $\theta \in \mathbb{R}$ un valore che si desidera conoscere. Una variabile aleatoria $\hat{\theta}$ utilizzata per stimare θ viene detta stimatore di θ.

Definizione 1.1.11. Uno stimatore $\hat{\theta}$ di un numero θ si dice corretto se

$$\mathbb{E}[\hat{\theta}] = \theta.$$

Definizione 1.1.12. Si dice root-mean-square error (RMSE) associato ad uno stimatore $\hat{\theta} \in L^2$ di un numero θ la quantità

$$RMSE(\hat{\theta}) = \sqrt{\mathbb{E}[(\hat{\theta} - \theta)^2]}.$$

In particolare, se $\hat{\theta}$ è corretto, il RMSE è la sua deviazione standard.

1.2 Legge dei grandi numeri (LLN)

Esistono più versioni della legge dei grandi numeri. Qui riporteremo quelle in L^2: variabili aleatorie in L^2 permettono di costruire una stima dell'errore commesso dal metodo (in qualche senso da specificare che sarà chiaro più avanti). Per le dimostrazioni di questa sezione si fa riferimento a [3] (pg. 119, 127).

Teorema 1.2.1 (Legge debole dei grandi numeri (LLN), versione L^2)**.** *Siano $X_1, X_2, \dots$ v.a. reali in $L^2(\Omega, \mathcal{F}, \mathbb{P})$ identicamente distribuite e scorrelate a due a due. Sia, inoltre, $m = \mathbb{E}[X_1] < \infty$ il valore atteso comune delle X_i. Allora:*

$$M_n \coloneqq \frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{L^2} m, \text{ per } n \to \infty$$

In particolare, dalla convergenza in L^2, discende anche la convergenza in probabilità.

Dimostrazione. La convergenza in L^2 implica quella in probabilità, per cui basta provare la prima.

$M_n \xrightarrow{L^2} m$ se e solo se:

$$\mathbb{E}[(M_n - m)^2] \to 0, \text{ per } n \to \infty.$$

Per $n \in \mathbb{N}$, si ha

$$\mathbb{E}[(M_n - m)^2] = \mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(X_i - m)\right)^2\right] = \frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n} X_i - nm\right)^2\right] = \frac{1}{n^2} Var\left(\sum_{i=1}^{n} X_i\right).$$

Ora:

$$Var\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} Var(X_i) + \sum_{i,j=1,\, i\neq j}^{n} Cov(X_i, X_j) = nVar(X_1) < +\infty,$$

poiché le X_i sono i.d., scorrelate e in L^2. Allora:

$$\mathbb{E}[(M_n - m)^2] = \frac{nVar(X_1)}{n^2} = \frac{Var(X_1)}{n} \to 0, \text{ per } n \to \infty.$$

□

Si può avere un risultato di convergenza più forte:

Teorema 1.2.2 (Legge forte dei grandi numeri (LLN), versione L^2)**.** *Siano $X_1, X_2, \dots$ v.a. reali in $L^2(\Omega, \mathcal{F}, \mathbb{P})$ i.i.d. Siano, inoltre, $m = \mathbb{E}[X_1] < \infty$ il valore atteso comune e $v = Var(X_1)$ la varianza comune delle X_i. Allora:*

$$M_n \coloneqq \frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{\mathbb{P}-q.c.} m, \text{ per } n \to \infty$$

Dimostrazione. Senza perdita di generalità possiamo assumere $m = 0$, altrimenti si possono considerare $X_i' \coloneqq X_i - m$.

Primo passo: si vuole dimostrare che $M_{n^2} \to 0$ $\mathbb{P}$-q.c.. Per ogni $\epsilon > 0$ si ha, per la disuguaglianza di Chebyshev, che $\mathbb{P}(|M_{n^2}| > \epsilon) \leq \frac{v}{n^2\epsilon^2}$. Quindi, siccome queste probabilità formano una serie convergente, si può applicare il lemma di Borel-Cantelli:

$$\mathbb{P}\left(\limsup_{n\to\infty} |M_{n^2}| > \epsilon\right) = \mathbb{P}\left(\bigcap_{k\geq 1}\bigcup_{n\geq k}\{|M_{n^2}| > \epsilon\}\right) = 0.$$

Facendo il passaggio al limite per $\epsilon \to 0^+$ e usando la continuità dal basso della misura, si ottiene che

$$\mathbb{P}\left(M_{n^2} \nrightarrow 0\right) = \mathbb{P}\left(\limsup_{n\to\infty} |M_{n^2}| > 0\right) = \lim_{\epsilon\to 0^+} \mathbb{P}\left(\limsup_{n\to\infty} |M_{n^2}| > \epsilon\right) = 0.$$

Secondo passo: per $m \in \mathbb{N}$, sia $n = n(m)$ tale che $n^2 \leq m < (n+1)^2$. L'obiettivo è confrontare M_m con M_{n^2}. Si ponga $S_k = kM_k = \sum_{i=1}^k X_i$. Dalla disuguaglianza di Chebyshev, unito al fatto che le X_i sono i.i.d. con media zero, si ha che

$$\mathbb{P}\left(|S_m - S_{n^2}| > \epsilon n^2\right) \leq \epsilon^{-2} n^{-4} Var\left(\sum_{i=n^2+1}^{m} X_i\right) = \frac{(m-n^2)v}{\epsilon^2 n^4},$$

e, inoltre,

$$\begin{aligned}\sum_{m\geq 1} \mathbb{P}\left(|S_m - S_{n(m)^2}| > \epsilon n(m)^2\right) &\leq \frac{v}{\epsilon^2} \sum_{n\geq 1} \sum_{m=n^2}^{(n+1)^2-1} \frac{m-n^2}{n^4} = \frac{v}{\epsilon^2} \sum_{n\geq 1} \sum_{k=1}^{2n} \frac{k}{n^4} \\ &= \frac{v}{\epsilon^2} \sum_{n\geq 1} \frac{(2n)(2n+1)}{2n^4} < +\infty,\end{aligned}$$

perciò, applicando il lemma di Borel-Cantelli come fatto in precedenza, si ha

$$\mathbb{P}\left(|\frac{S_m}{n(m)^2} - M_{n(m)^2}| \overset{m\to\infty}{\longrightarrow} 0\right) = 1.$$

Combinato con il risultato del primo passo:

$$\mathbb{P}\left(\frac{S_m}{n(m)^2} \overset{m\to\infty}{\longrightarrow} 0\right) = 1.$$

Dato che $|M_m| = \frac{|S_m|}{m} \leq \frac{|S_m|}{n(m)^2}$, segue che $\mathbb{P}\left(M_m \to 0\right) = 1$. □

1.3 Il metodo Monte Carlo

Utilizzando la legge dei grandi numeri, si può descrivere a grandi linee la struttura del metodo MC: sia $Z \in L^2(\Omega, \mathcal{F}, \mathbb{P})$ una variabile aleatoria, con valore atteso $z = \mathbb{E}[Z]$, e $Z_1, \ldots, Z_N$ copie i.i.d. di Z. Allora, per la LLN, vale:

$$\mathbb{E}[Z] \approx \hat{z}_{CMC} = \frac{1}{N} \sum_{k=1}^{N} Z_k, \text{ per } N \gg 1.$$

Dunque il metodo MC permette di approssimare numericamente il valore atteso di una variabile aleatoria di cui si sia in grado di generare delle prove ripetute e indipendenti. In questa sua forma generale, il metodo viene spesso chiamato **Crude Monte Carlo (CMC)**, per distinguerlo da altre sue varianti che implementano tecniche particolari, e $\hat{z}_{CMC}$ è lo stimatore (corretto) Crude Monte Carlo di z. Di seguito è riportato l'algoritmo in pseudocodice.

Algoritmo 1 Algoritmo CMC

Input: N
Output: approssimazione di $\mathbb{E}[Z]$
1: $S \leftarrow 0$
2: **for** $k \leftarrow 1$ to N **do**
3: Genera $Z_k \sim Z$
4: $S \leftarrow S + Z_k$
5: **end for**
6: $M \leftarrow \frac{S}{N}$
7: **return** M

Quindi il metodo risulta molto semplice e facilmente implementabile. Il passo successivo è capire come controllare l'errore e che tipo di errore sarà, considerata la natura probabilistica del metodo.

Capitolo 2

Gestione dell'errore nel metodo Monte Carlo

2.1 Teorema del limite centrale (CLT)

Per dimostrare il teorema del limite centrale sono utili i seguenti risultati sulle funzioni caratteristiche:

Lemma 2.1.1 (Proprietà delle funzioni caratteristiche)**.** *Sia X una v.a. a valori in $\mathbb{R}^d$ e $\varphi_X(t) = \mathbb{E}[e^{i<t,X>}]$ la sua funzione caratteristica. Allora:*

1. *$|\varphi_X(t)| \leq 1$ per ogni $t \in \mathbb{R}^d$ e $\varphi_X(0) = 1$.*
2. *$\varphi_{aX+b}(t) = \varphi_X(at)e^{i<t,b>}$ per ogni $a \in \mathbb{R}$ e $b \in \mathbb{R}^d$.*
3. *$\mathbb{P}_X = \mathbb{P}_{-X}$ se e solo se φ è a valori reali.*
4. *Se X e Y sono indipendenti, allora $\varphi_{X+Y} = \varphi_X \cdot \varphi_Y$.*

Queste sono proprietà note delle funzioni caratteristiche e per il lettore che fosse interessato ad una dimostrazione si rimanda al testo di Klenke [4] (pg.303). Le prossime dimostrazioni seguono sempre la fonte appena citata.

Lemma 2.1.2. *Per ogni $t \in \mathbb{R}$ e $n \in \mathbb{N}$ si ha*

$$\left| e^{it} - \sum_{l=0}^{n-1} \frac{(it)^l}{l!} \right| \leq \frac{|t|^n}{n!}$$

Dimostrazione. Conoscendo lo sviluppo di Taylor dell'esponenziale (che è C^∞) e utilizzando il resto di Lagrange, si ottiene che

$$\left| e^{it} - \sum_{l=0}^{n-1} \frac{(it)^l}{l!} \right| = \left| \frac{i^n e^{i\xi}}{n!} t^n \right| = \frac{|t|^n}{n!} \text{ per ogni } t \in \mathbb{R}$$

dove $\xi \in (0, t)$ □

Lemma 2.1.3 (Momenti e differenziabilità)**.** *Sia X una variabile aleatoria reale con funzione caratteristica φ. Se $\mathbb{E}[|X|^n] < \infty$, allora φ è n volte continuamente differenziabile con derivate*

$$\varphi^{(k)}(t) = \mathbb{E}[(iX)^k e^{itX}] \text{ per } k = 0, \ldots, n.$$

In particolare, se $\mathbb{E}[X^2] < \infty$*, allora*

$$\varphi(t) = 1 + it\mathbb{E}[X] - \frac{1}{2}t^2\mathbb{E}[X^2] + \varepsilon(t)t^2$$

con $\varepsilon(t) \to 0$ *per* $t \to 0$*.*

Dimostrazione. Per $t \in \mathbb{R}, h \in \mathbb{R}^*$ e $k \in 1, \dots, n$, si definisca

$$Y_k(t,h,x) = k!h^{-k}e^{itx}\left(e^{ihx} - \sum_{l=0}^{k-1}\frac{(ihx)^l}{l!}\right),$$

allora

$$\mathbb{E}[Y_k(t,h,X)] = k!h^{-k}\left(\varphi(t+h) - \varphi(t) - \sum_{l=1}^{k-1}\mathbb{E}[e^{itX}(iX)^l]\frac{h^l}{l!}\right).$$

Se il limite $\varphi_k(t) \coloneqq \lim_{h\to 0}\mathbb{E}[Y_k(t,h,X)]$ esiste, allora φ è differenziabile k volte in t con $\varphi^{(k)}(t) = \varphi_k(t)$. Applicando il lemma precedente (o utilizzando il teorema di Taylor), si ottiene che

$$Y_k(t,h,x) = k!h^{-k}e^{itx}\left(e^{ihx} - \sum_{l=0}^{k}\frac{(ihx)^l}{l!} + \frac{(ihx)^k}{k!}\right) = (ix)^k e^{itx} + o(1) \overset{h\to 0}{\longrightarrow} (ix)^k e^{itx}.$$

Sempre per il lemma precedente

$$|Y_k(t,h,x)| \le k!h^{-k}\frac{|hx|^k}{k!} = |x|^k.$$

Considerando che $\mathbb{E}[|X|^k] < \infty$ per ogni $k \in \mathbb{N}$ per ipotesi, il teorema della convergenza dominata implica che

$$\mathbb{E}[Y_k(t,h,X)] \overset{h\to 0}{\longrightarrow} \mathbb{E}[(iX)^k e^{itX}] = \varphi^{(k)}(t).$$

□

Teorema 2.1.1 (Teorema di continuità di Lévy)**.** *Siano* $(X_n)_{n\in\mathbb{N}}$ *v.a. in* $\mathbb{R}^d$ *e* $(\varphi_n)_{n\in\mathbb{N}}$ *la successione delle corrispondenti funzioni caratteristiche.*

1. *Se* $X_n \overset{n\to\infty}{\longrightarrow} X$ *in distribuzione, allora* $\varphi_n \to \varphi$ *uniformemente sui compatti;*
2. *Se esiste* $f : \mathbb{R}^d \mapsto \mathbb{C}$ *parzialmente continua in 0 tale che* $\varphi_n(y) \overset{n\to\infty}{\longrightarrow} f(y)$ *per ogni* $y \in \mathbb{R}^d$*, allora* $(X_n)_{n\in\mathbb{N}}$ *converge in distribuzione ad una v.a.* Y *con funzione caratteristica* $\varphi_Y = f$*.*

Per una dimostrazione completa del Teorema 2.1.1 si rimanda a [4] (pg. 311-312).

Teorema 2.1.2 (Teorema del limite centrale (CLT))**.** *Siano* $(X_n)_{n\in\mathbb{N}}$ *v.a. reali i.i.d. in* L^2*, con* $\mathbb{E}[X_1] = m$ *e* $Var(X_1) = \sigma^2$*. Si ponga*

$$S_n \coloneqq \frac{1}{\sqrt{n\sigma^2}}\sum_{k=1}^{n}(X_k - m),\ n \in \mathbb{N}.$$

Allora $(S_n)_{n\in\mathbb{N}}$ *converge in distribuzione ad una normale standard.*

Dimostrazione. Sia $\varphi = \varphi_{X_k - m}$. Allora, per il lemma 2.1.3,

$$\varphi(t) = 1 - \frac{\sigma^2}{2}t^2 + \varepsilon(t)t^2,$$

dove $\varepsilon(t) \to 0$ per $t \to 0$. Per il lemma 2.1.1,

$$\varphi_{S_n}(t) = \left(\varphi\left(\frac{t}{\sqrt{n\sigma^2}}\right)\right)^n.$$

Sapendo che $|u^n - v^n| \leq |u - v| \cdot n \cdot \max(|u|, |v|)^{n-1}$ per ogni $u, v \in \mathbb{C}$,

$$\left|\left(1 - \frac{t^2}{2n}\right)^n - \left(\varphi\left(\frac{t}{\sqrt{n\sigma^2}}\right)\right)^n\right| \leq n\left|1 - \frac{t^2}{2n} - \varphi\left(\frac{t}{\sqrt{n\sigma^2}}\right)\right| \leq n\frac{t}{n\sigma^2}\left|\varepsilon\left(\frac{t}{\sqrt{n\sigma^2}}\right)\right| \overset{n\to\infty}{\longrightarrow} 0.$$

Mettendo assieme ciò che è stato scritto sopra e che $\left(1 - \frac{t^2}{2n}\right)^n \overset{n\to\infty}{\longrightarrow} e^{-\frac{t^2}{2}}$, utilizzando la disuguaglianza triangolare, si ottiene che

$$\lim_{n\to\infty} \varphi_{S_n}(t) = e^{-\frac{t^2}{2}}.$$

Essendo $e^{-\frac{t^2}{2}}$ la funzione caratteristica di una normale standard, per il Teorema 2.1.1, si ha la tesi. □

2.2 RMSE e intervalli di confidenza

Si considerino $Z_1, \ldots, Z_N$ i.i.d. in L^2 e si supponga di calcolare il valore atteso comune $z = \mathbb{E}[Z]$ utilizzando il suo stimatore CMC $\hat{z}_{CMC}$. Nel caso in esame, il RMSE di $\hat{z}_{CMC}$ si può riscrivere usando il fatto che è uno stimatore corretto di z:

$$RMSE(\hat{z}_{CMC}) = \sqrt{\mathbb{E}[(\hat{z}_{CMC} - z)^2]} = \sqrt{Var(\hat{z}_{CMC})}.$$

Però le $Z_1, \ldots, Z_N$ sono i.i.d., con $\sigma^2 = Var(Z_1)$ la varianza comune, e quindi, per le proprietà della varianza,

$$Var(\hat{z}_{CMC}) = \frac{1}{N^2}\sum_{k=1}^{N} Var(Z_k) = \frac{N\sigma^2}{N^2} = \frac{\sigma^2}{N}.$$

Dunque

$$RMSE(\hat{z}_{CMC}) = \frac{\sigma}{\sqrt{N}},$$

che altro non è che la deviazione standard di $\hat{z}_{CMC}$.

Si osservi che esistono due modi per ridurre il RMSE:

- aumentare N;
- diminuire σ scegliendo stimatori appropriati.

Il primo punto è banale e si tratta solo di "forza bruta", mentre il secondo punto verrà sviluppato più avanti.

Utilizzando il CLT e il RMSE, è possibile costruire degli intervalli di confidenza asintotici per z. Consideriamo $t \in (0,1)$, allora si dice t-quantile un numero q_t tale per cui $F(q_t) = t$, dove F è una funzione di ripartizione. Se $\alpha \in (0,1)$ e $q_{\frac{\alpha}{2}}, q_{1-\frac{\alpha}{2}}$ sono quantili della distribuzione normale standard, allora

$$\mathbb{P}\left(\frac{\sqrt{N}}{\sigma}(\hat{z}_{CMC} - z) \in (q_{\frac{\alpha}{2}}, q_{1-\frac{\alpha}{2}})\right) \approx \Phi(q_{1-\frac{\alpha}{2}}) - \Phi(q_{\frac{\alpha}{2}}) = 1 - \alpha, \text{ per } N \gg 1,$$

dove Φ è la funzione di ripartizione della distribuzione normale standard e l'approssimazione è data dall'applicazione del CLT, cioè $\frac{\sqrt{N}}{\sigma}(\hat{z}_{CMC} - z) \xrightarrow{dist} V \sim N(0,1)$ per $N \to \infty$. Quindi l'intervallo

$$I_\alpha = (\hat{z}_{CMC} - q_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{N}},\ \hat{z}_{CMC} - q_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{N}})$$

è un intervallo di confidenza asintotico di livello $1-\alpha$ per z. In molti casi, però, è raro conoscere il valore di σ e questo va stimato. Per far ciò si considera lo stimatore (corretto) della varianza σ^2 noto come varianza campionaria:

$$s^2 = \frac{1}{N-1}\sum_{k=1}^{N}(Z_k - \hat{z}_{CMC})^2.$$

Quindi, rimpiazzando σ con s in I_α e sapendo che $s^2 \xrightarrow{\mathbb{P}} \sigma^2$, si ottiene l'intervallo di confidenza asintotico di livello $1-\alpha$ seguente:

$$(\hat{z}_{CMC} - q_{1-\frac{\alpha}{2}}\frac{s}{\sqrt{N}},\ \hat{z}_{CMC} - q_{\frac{\alpha}{2}}\frac{s}{\sqrt{N}}).$$

Quindi, vista la necessità di calcolare anche s^2, ecco una versione "aggiornata" dell'algoritmo di CMC che utilizza l'algoritmo di Welford[7] per calcolare media e varianza campionaria:

Algoritmo 2 Algoritmo CMC (Welford)

Input: N
Output: approssimazione di $\mathbb{E}[Z]$ e s^2

$M \leftarrow 0$
$S \leftarrow 0$
for $k \leftarrow 1$ to N **do**
    Genera $Z_k \sim Z$
    $\delta \leftarrow Z_k - M$
    $M \leftarrow M + \delta/k$
    $S \leftarrow S + (k-1)/k\delta^2$
end for
$s^2 = S/(N-1)$
return M, s^2

2.3 Integrazione Monte Carlo

Un'applicazione del metodo Monte Carlo si trova nel campo dell'integrazione numerica. Si supponga, per esempio, di avere un boreliano limitato $B \in \mathcal{B}(\mathbb{R}^d)$ e una funzione $f \in L^2(\mathbb{R}^d, \mathcal{B}, m_d)$, dove m_d è la misura di Lebesgue d-dimensionale. L'obiettivo è quello di calcolare

$$\int_B f dm_d.$$

Poiché B è limitato, si può prendere un plurirettangolo R tale che $B \subseteq R$ e una successione $(X_n)_{n\in\mathbb{N}}$ di variabili aleatorie i.i.d. tali che $X_n \sim Unif_d(R)$. Ora si ponga

$$Y_n = f(X_n) \cdot \mathbb{1}_B(X_n),\ n \in \mathbb{N}.$$

Allora $(Y_n)_{n\in\mathbb{N}}$ è una successione di v.a. i.i.d. in $L^2(\Omega, \mathcal{F}, \mathbb{P})$. Quindi, per la legge forte dei grandi numeri,

$$\frac{1}{N}\sum_{n=1}^{N} Y_n \overset{N\to\infty}{\longrightarrow} \mathbb{E}[Y_1]\ \mathbb{P}\text{-q.c.}$$

Ma

$$\mathbb{E}[Y_1] = \int_\Omega f(X_1)\mathbb{1}_B(X_1)d\mathbb{P}_{X_1} = \frac{1}{m_d(R)}\int_{\mathbb{R}^d} f(x)\mathbb{1}_B(x)\mathbb{1}_R(x)dx = \frac{1}{m_d(R)}\int_B f(x)dx.$$

Quindi le medie empiriche approssimano l'integrale che si voleva calcolare a meno di un fattore moltiplicativo. Se $f \in L^\infty(\mathbb{R}^d, \mathcal{B}, m_d)$, si può anche stimare dall'alto il RMSE, usando la norma infinito di f per maggiorare la deviazione standard.

Questo metodo presenta sia vantaggi che svantaggi. Partendo dagli ultimi, l'ordine di convergenza $O(N^{-1/2})$ è molto lento rispetto alle controparti deterministiche (formule di cubatura) e l'errore è di tipo probabilistico. Per quanto riguarda i vantaggi, l'algoritmo è di facile implementazione, le ipotesi sulla regolarità di f sono molto deboli e l'errore non dipende dalla dimensione d, ma solo da N. Infatti, l'integrazione Monte Carlo è una buona scelta in situazioni in cui la dimensione è molto alta, cioè casi in cui altri metodi numerici potrebbero perdere prestazione (anche data una certa regolarità su f) ed essere difficili da implementare.

Capitolo 3

Tecniche di riduzione della varianza

Le tecniche di riduzione della varianza consistono nel trovare uno stimatore alternativo $\hat{z}_{VR}$ di un numero $z = \mathbb{E}[Z]$, avente una varianza più piccola dello stimatore $\hat{z}_{CMC}$. Per far ciò, di solito, si utilizzano informazioni che si hanno a priori sul modello/problema, oppure si conduce una simulazione pilota CMC per ottenere più informazioni e utilizzarle per aumentare l'accuratezza dello stimatore, tramite la formulazione di una tecnica appropriata. Per alcuni esempi applicativi, si rimanda il lettore al capitolo successivo. Nei paragrafi che seguono, si considereranno sempre $(\Omega, \mathcal{F}, \mathbb{P})$ uno spazio di probabilità, $Z \in L^2(\Omega)$ e $z = \mathbb{E}[Z]$. Le fonti di riferimento per questo capitolo sono [1], [6] e [5].

3.1 Importance Sampling

L'Importance Sampling ha come principio di fondo l'ottenere una riduzione della varianza cambiando la misura di probabilità $\mathbb{P}$ rispetto alla quale si campiona con una che si concentra nella parte di spazio che contribuisce di più al valore di z. Si può, quindi, scegliere una misura di probabilità $\tilde{\mathbb{P}}$ per cui esiste una funzione L tale che

$$\mathbb{1}_{\{Z(\omega)\neq 0\}}\mathbb{P}(d\omega) = \mathbb{1}_{\{Z(\omega)\neq 0\}}L(\omega)\tilde{\mathbb{P}}(d\omega).$$

Dunque,

$$\mathbb{E}[Z] = \int_\Omega Z d\mathbb{P} = \int_\Omega ZL d\tilde{\mathbb{P}} = \tilde{\mathbb{E}}[ZL],$$

dove $\tilde{\mathbb{E}}$ è il valore atteso rispetto a $\tilde{\mathbb{P}}$. Da qui si prosegue come con CMC, ovvero si generano N copie i.i.d. di ZL rispetto $\tilde{\mathbb{P}}$ e si stima z tramite

$$\hat{z}_{IS} = \frac{1}{N}\sum_{i=1}^{N} Z_i L_i.$$

Tutto ciò, comunque, non garantisce una riduzione della varianza, poiché questo dipende da $\tilde{\mathbb{P}}$ e, dunque, il problema diventa scegliere una $\tilde{\mathbb{P}}$ appropriata. In realtà, si può dimostrare che esiste una scelta ottimale:

Teorema 3.1.1. *Sia $\mathbb{P}^*$ tale che*

$$\mathbb{P}^*(d\omega) = \frac{|Z|}{\mathbb{E}[|Z|]}\mathbb{P}(d\omega), \;\; i.e., \;\; L^*(\omega) = \frac{\mathbb{E}[|Z|]}{|Z|(\omega)} \quad se\ Z(\omega) \neq 0.$$

Allora la varianza di ZL^* rispetto a $\mathbb{P}^*$ è minore di quella di ZL rispetto a qualsiasi altra $\tilde{\mathbb{P}}$. Inoltre, se $Z \geq 0$ $\mathbb{P}-q.c.$, allora $Var^*(ZL^*) = 0$.

Dimostrazione. Poiché sia ZL^* che ZL sono stimatori corretti (rispetto le misure $\mathbb{P}*$ e $\tilde{\mathbb{P}}$) di z, basterà dimostrare che il secondo momento del primo stimatore è minore di quello del secondo. Si ha che, per come è stata definita L*,

$$\mathbb{E}^*[(ZL^*)^2] = \mathbb{E}[Z^2 L^*] = \mathbb{E}[Z^2 \frac{\mathbb{E}[|Z|]}{|Z|}] = (\mathbb{E}[|Z|])^2,$$

da cui,

$$(\mathbb{E}[|Z|])^2 = (\tilde{\mathbb{E}}[|Z|L])^2 \leq \tilde{\mathbb{E}}[(ZL)^2],$$

per definizione di L e per la disuguaglianza di Jensen. Se $Z \geq 0 \ \mathbb{P} - q.c.$, allora

$$Var^*(ZL^*) = \mathbb{E}^*[(ZL^*)^2] - z^2 = \mathbb{E}[Z]^2 - z^2 = 0.$$

□

In generale, però, la misura $\mathbb{P}^*$ potrebbe essere problematica da utilizzare, poiché richiede la conoscenza di $\mathbb{E}[|Z|]$ che, nel caso di $Z \geq 0 \ \mathbb{P} - q.c.$, sarebbe proprio $z = \mathbb{E}[Z]$, cioè la quantità che si vuole stimare. Inoltre, in molti casi, non si conosce nemmeno l'espressione analitica di Z a priori. Un primo approccio potrebbe essere quello di condurre una simulazione pilota per ottenere una stima di $\mathbb{E}[|Z|]$ e utilizzare quest'ultima per costruire una misura $\tilde{\mathbb{P}}$ che si avvicina il più possibile a $\mathbb{P}^*$, con il rischio che, per alte dimensioni, sia poco efficiente campionare da quest'ultima ([6], cap. 5, sez. 5.7, pg. 161).

Prima di passare alla sezione successiva, si vuole mostrare un semplice caso in cui si può utilizzare l'importance sampling riducendo la varianza.

Esempio 3.1.1. Sia $z = \int_a^b e^{-\frac{x^2}{2}} dx$ con $a, b \in \mathbb{R}$ tali che l'intervallo (a, b) sia spostato verso le code della gaussiana. A meno di una costante moltiplicativa, calcolare z significa trovare la probabilità che una v.a. normale standard stia in (a, b). Quindi un primo approccio è generare copie indipendenti di tale variabile aleatoria e farne la media campionaria. Però, essendo l'intervallo spostato verso le code, la probabilità che escano osservazioni in (a, b) è molto bassa e, dunque, la convergenza molto lenta. Invece, l'idea è che se si campionasse in modo uniformemente distribuito su (a, b) la convergenza del metodo MC dovrebbe essere più veloce.

Perciò, siano

$$Z = \sqrt{2\pi} \mathbb{1}_{(a,b)}(X), \text{ con } X \sim N(0, 1)$$
$$Z' = (b - a) e^{-\frac{(X')^2}{2}}, \text{ con } X' \sim Unif(a, b)$$

Si nota facilmente che entrambi sono stimatori corretti di z e, di conseguenza, la varianza di Z è maggiore di quella di Z' se e solo se il momento secondo di Z è maggiore di quello di Z'. Allora

$$\mathbb{E}[Z^2] = \mathbb{E}[2\pi(\mathbb{1}_{(a,b)}(X))^2] = \sqrt{2\pi}\mathbb{E}[\sqrt{2\pi}\mathbb{1}_{(a,b)}(X)] = \sqrt{2\pi}z$$

e

$$\mathbb{E}[(Z')^2] = (b - a)^2 \mathbb{E}[e^{-(X')^2}] = (b - a)^2 \frac{1}{b - a} \int_a^b e^{-x^2} dx = (b - a) \int_a^b e^{-x^2} dx.$$

Si vuole trovare una condizione sufficiente su a e b tale che

$$\sqrt{2\pi}z \geq (b - a) \int_a^b e^{-x^2} dx.$$

Osservando che $e^{-\frac{x^2}{2}} \geq e^{-x^2}$, per monotonia dell'integrale, una condizione sufficiente affinché valga la disuguaglianza (stretta) è

$$\sqrt{2\pi} \geq b - a.$$

Si noti, infine, che

$$z = \mathbb{E}[Z] = \int_\Omega Z d\mathbb{P}_X$$
$$z = \mathbb{E}[Z'] = \int_\Omega Z' d\mathbb{P}_{X'}$$

e

$$\mathbb{P}_X(dx) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$
$$\mathbb{P}_{X'}(dx) = \frac{1}{b-a} \mathbb{1}_{(a,b)}(x),$$

da cui

$$\begin{aligned}\mathbb{1}_{\{Z(x)\neq 0\}} \mathbb{P}_X(dx) = \mathbb{1}_{(a,b)}(x) \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} = \\ = \mathbb{1}_{(a,b)}(x) \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \frac{\frac{1}{b-a}\mathbb{1}_{(a,b)}(x)}{\frac{1}{b-a}} = \\ = \mathbb{1}_{(a,b)}(x) \frac{b-a}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \frac{1}{b-a} \mathbb{1}_{(a,b)}(x) = \mathbb{1}_{\{Z(x)\neq 0\}} L(x) \mathbb{P}_{X'}(dx),\end{aligned}$$

dove $L(x) = \frac{b-a}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

3.2 Antithetic Sampling

Sia $M = 2N$, e siano $Z_1 \dots Z_M$ copie identicamente distribuite di Z tali che le coppie (Z_{2i-1}, Z_{2i}), per $i = 1, \dots, N$, siano fra loro indipendenti e che le componenti di una coppia siano "il più negativamente correlate possibile". Allora si introduce lo stimatore $\hat{z}_{Anth} = \frac{1}{M} \sum_{i=1}^{N} (Z_{2i-1} + Z_{2i})$ che ha varianza:

$$\begin{aligned}Var(\hat{z}_{Anth}) = \frac{1}{N} Var\left(\frac{Z_1 + Z_2}{2}\right) = \frac{1}{4N}(Var(Z_1) + Var(Z_2) + 2Cov(Z_1, Z_2)) = \\ = \frac{1}{4N}(\sigma^2_{CMC} + \sigma^2_{CMC} + 2\rho\sigma^2_{CMC}) = \frac{\sigma^2_{CMC}}{2N}(1+\rho), \quad (3.1)\end{aligned}$$

dove $\sigma^2_{CMC} = Var(Z)$.
Quindi, rispetto allo stimatore $\hat{z}_{CMC}$, il MSE diminuisce di un fattore $\frac{1+\rho}{2}$ solo se $\rho \leq 0$, da cui la necessità di prendere Z_1 e Z_2 "il più negativamente correlate possibile".

Sotto opportune ipotesi, può tornare utile il seguente:

Teorema 3.2.1 (Co-monotonia)**.** *Sia $Z : (\Omega, \mathcal{F}, \mathbb{P}) \to (\mathbb{R}, \mathcal{B}_\mathbb{R})$ una variabile aleatoria reale e siano $\phi, \psi : \mathbb{R} \to \mathbb{R}$ due funzioni monotone (quindi Borel-misurabili) con monotonia concorde. Se $\phi(Z), \psi(Z) \in L^2(\Omega)$, allora si ha che*

$$Cov\left(\phi(Z), \psi(Z)\right) \geq 0. \quad (3.2)$$

Viceversa, se ϕ e ψ hanno monotonia discorde, allora

$$Cov\left(\phi(Z), \psi(Z)\right) \leq 0. \tag{3.3}$$

Inoltre, vale l'uguaglianza se e solo se $\phi(Z) = \mathbb{E}[\phi(Z)]$ $\mathbb{P}$-q.c. o $\psi(Z) = \mathbb{E}[\psi(Z)]$ $\mathbb{P}$-q.c.

Dimostrazione. Si prenda Z come sopra e Z' copia indipendente di Z. Supponiamo che ϕ e ψ abbiano monotonia concorde, allora, se $a, b \in \mathbb{R}$, si ha che

$$\phi(a) - \phi(b), \psi(a) - \psi(b) \geq 0 \tag{3.4}$$

o

$$\phi(a) - \phi(b), \psi(a) - \psi(b) \leq 0 \tag{3.5}$$

a seconda della monotonia delle due funzioni.
Quindi,

$$(\phi(a) - \phi(b))(\psi(a) - \psi(b)) \geq 0 \tag{3.6}$$

e sostituendo a e b con Z e Z',

$$(\phi(Z) - \phi(Z'))(\psi(Z) - \psi(Z')) \geq 0. \tag{3.7}$$

Applicando il valore atteso ad entrambi i membri, le proprietà del valore atteso e l'indipendenza delle due v.a., si ottiene:

$$\mathbb{E}[\phi(Z)\psi(Z)] + \mathbb{E}[\phi(Z')\psi(Z')] - \mathbb{E}[\phi(Z)]\mathbb{E}[\psi(Z')] - \mathbb{E}[\phi(Z')]\mathbb{E}[\psi(Z)] \geq 0, \tag{3.8}$$

ciò, ricordando che Z e Z' hanno la stessa distribuzione, diventa:

$$2Cov\left(\phi(Z), \psi(Z)\right) = 2\mathbb{E}[\phi(Z)\psi(Z)] - 2\mathbb{E}[\phi(Z)]\mathbb{E}[\psi(Z)] \geq 0, \tag{3.9}$$

da cui la (3.2). Per la (3.3), basta assumere ϕ e ψ con monotonia discordi e, facendo un ragionamento analogo, si arriva ad invertire il segno nella (3.7).

Uguaglianza Un verso dell'equivalenza è banale: se una tra $\phi(Z)$ e $\psi(Z)$ è costante $\mathbb{P} - q.c.$, allora l'uguaglianza vale.

Ora si consideri l'altro verso: senza perdita di generalità assumiamo ϕ e ψ non decrescenti e le supponiamo continue a destra sui reali che non sono atomi della distribuzione di Z.

Percorrendo la dimostrazione di sopra al contrario, se vale l'uguaglianza, segue che $\mathbb{E}[(\phi(Z) - \phi(Z'))(\psi(Z) - \psi(Z'))] = 0$ e quindi $(\phi(Z) - \phi(Z'))(\psi(Z) - \psi(Z')) = 0$ $\mathbb{P} - q.c.$ Sia ora I l'inviluppo convesso del supporto della distribuzione di Z sulla retta reale. Si può assumere $I = [a, b] \subset \mathbb{R}$, a, $b \in \mathbb{R}$, $a < b$ (gli altri casi possono essere adattati da questo). Per costruzione si ha che a e b sono nel supporto della distribuzione, quindi per ogni $\varepsilon \in (0, b - a)$, le probabilità $\mathbb{P}(a \leq Z \leq a + \varepsilon)$ e $\mathbb{P}(b - \varepsilon \leq Z' \leq b)$ sono strettamente positive. Se a è un atomo della distribuzione di Z, si può considerare $\varepsilon_a = 0$, lo stesso vale per b. Quindi l'evento $C_\varepsilon = \{a \leq Z \leq a + \varepsilon\} \cap \{b - \varepsilon \leq Z' \leq b\}$ ha probabilità positiva siccome Z e Z' sono indipendenti. Si assuma ora che $\phi(Z)$ non sia costante quasi certamente. Allora ϕ non è costante su I e $\phi(a) < \phi(b)$. Di conseguenza su C_ε, per ε sufficientemente piccolo, $\phi(Z) - \phi(Z') < 0$ quasi certamente, da cui $\psi(Z) - \psi(Z') = 0$ quasi certamente; quindi $\psi(a + \varepsilon) = \psi(b - \varepsilon)$. Facendo andare ε a 0, si ottiene che $\psi(a) = \psi(b)$ (tenendo a mento la convenzione fissata inizialmente sugli atomi). Ciò mostra che $\psi(Z)$ è costante $\mathbb{P}$-q.c. □

Corollario 3.2.1.1. *Sia $T : \mathbb{R} \to \mathbb{R}$ una funzione con monotonia opposta a ϕ, dove ϕ è come nel teorema precedente, tale che $T(Z) \stackrel{d}{=} Z$. Allora:*

$$Cov(\phi(Z), \phi(T(Z))) \leq 0. \tag{3.10}$$

Dimostrazione. Poiché T è monotona (e quindi boreliana) nel verso opposto a ϕ, si ha che ϕ e $\phi \circ T$ hanno monotonie discordi. Quindi, sostituendo opportunamente nella (3.3), si ottiene la tesi. □

Da questo corollario si hanno due casi importanti dell'Antithetic Sampling:

- una v.a. simmetrica, cioè $Z \stackrel{d}{=} -Z$ e $T(x) = -x$;
- una v.a. a valori in $[0, l]$ tale che $Z \stackrel{d}{=} l - Z$ e $T(x) = l - x$.

Una v.a. aleatoria che soddisfa il secondo punto è $Z \stackrel{d}{=} Unif(0, l)$.

Il teorema 3.2.1 può essere esteso a più dimensioni:

Teorema 3.2.2. *Sia $d \in \mathbb{N}^*$ e siano ϕ, $\psi : \mathbb{R}^d \to \mathbb{R}$ due funzioni tali che $\forall i \in \{1, \dots, d\}$ e $\forall\, (z_{i+1}, \dots, z_d) \in \mathbb{R}^{d-i}$ le funzioni*

$$z_i \mapsto \phi\,(z_1, \dots, z_i, \dots, z_d)$$
$$z_i \mapsto \psi\,(z_1, \dots, z_i, \dots, z_d)$$

hanno la stessa monotonia, con il senso di monotonia indipendente da $(z_1, \dots, z_{i-1}) \in \mathbb{R}^{i-1}$.
Siano $Z_1, \dots, Z_d$ variabili aleatorie reali e indipendenti. Se $\phi\,(Z_1, \dots, Z_d)\,, \psi\,(Z_1, \dots, Z_d) \in L^2\,(\Omega, \mathcal{A}, \mathbb{P})$, allora

$$Cov\,(\phi\,(Z_1, \dots, Z_d)\,, \psi\,(Z_1, \dots, Z_d)) \geq 0.$$

Dimostrazione. Si procede per induzione su d. Se $d = 1$, allora il risultato è vero per il teorema 3.2.1. Si supponga vero il risultato per $d - 1$, allora per il teorema di Fubini-Tonelli:

$$\begin{aligned}\mathbb{E}\left[\phi\,(Z_1, \dots, Z_d)\,\psi\,(Z_1, \dots, Z_d)\right] &= \int_{\mathbb{R}} \mathbb{E}\left[\phi\,(Z_1, \dots, Z_{d-1}, z_d)\,\psi\,(Z_1, \dots, Z_{d-1}, z_d)\right] \mathbb{P}\,(dz_d) \\ &\geq \int_{\mathbb{R}} \mathbb{E}\left[\phi\,(Z_1, \dots, Z_{d-1}, z_d)\right] \mathbb{E}\left[\psi\,(Z_1, \dots, Z_{d-1}, z_d)\right] \mathbb{P}\,(dz_d)\end{aligned}$$

dove la disuguaglianza si ottiene applicando l'ipotesi induttiva per $d-1$, considerato che, per z_d fissato, $(z_1, \dots, z_{d-1}) \mapsto \phi(z_1, \dots, z_{d-1}, z_d)$ e $(z_1, \dots, z_{d-1}) \mapsto \psi(z_1, \dots, z_{d-1}, z_d)$ rispettano le condizioni di monotonia dell'enunciato. Invece, poiché $z_d \mapsto \phi(z_1, \dots, z_{d-1}, z_d)$ e $z_d \mapsto \psi(z_1, \dots, z_{d-1}, z_d)$ hanno senso di monotonia indipendente da $(z_1, \dots, z_{d-1})$, le funzioni $z_d \mapsto \Phi(z_d) = \mathbb{E}[\phi\,(Z_1, \dots, z_d)]$ e $z_d \mapsto \Psi(z_d) = \mathbb{E}[\psi\,(Z_1, \dots, z_d)]$ hanno la stessa monotonia. Perciò,

$$\begin{aligned}\int_{\mathbb{R}} \mathbb{E}\left[\phi\,(Z_1, \dots, Z_{d-1}, z_d)\right] \mathbb{E}\left[\psi\,(Z_1, \dots, Z_{d-1}, z_d)\right] \mathbb{P}\,(dz_d) &= \mathbb{E}[\Phi(Z_d)\Psi(Z_d)] \\ &\geq \mathbb{E}[\Phi(Z_d)]\mathbb{E}[\Psi(Z_d)] \\ &= \mathbb{E}[\phi(Z_1, \dots, Z_d)]\mathbb{E}[\psi(Z_1, \dots, Z_d)],\end{aligned}$$

dove la disuguaglianza viene applicando la base induttiva a Φ e Ψ e si conclude usando Fubini-Tonelli nell'ultimo passaggio. □

Dall'ultimo teorema discende il seguente:

Corollario 3.2.2.1. *Siano $T_i : \mathbb{R} \to \mathbb{R}$, $i = 1, \ldots, d$ funzioni non crescenti tali che $T_i(Z_i) \overset{d}{=} Z_i$. Allora, se ϕ e ψ sono come nella 3.2.2, si ha che*

$$Cov\left(\phi\left(Z_1, \ldots, Z_d\right), \psi\left(T_1(Z_1), \ldots, T_d(Z_d)\right)\right) \leq 0.$$

Dimostrazione. Il risultato si ottiene facendo un ragionamento analogo a quello del teorema precedente, cambiando il verso delle disuguaglianze a causa delle monotonie discordi. □

3.3 Control Variates

Lo scopo del Control Variates è quello di trovare una v.a. W con media $w = \mathbb{E}[W]$ nota e tale da essere il più correlata possibile (negativamente o positivamente) a Z, per poi combinarle e ottenere uno stimatore con varianza minore di $\hat{z}_{CMC}$. Più nello specifico, l'idea è quella di prendere

$$\tilde{Z} = Z + \alpha(W - w)$$

dove α viene scelto in modo tale che minimizzi la varianza di $\tilde{Z}$. Si osservi che, indipendentemente da α, $\tilde{Z}$ è uno stimatore corretto di z.

Usando le proprietà della varianza si vede che

$$Var(\tilde{Z}) = Var(Z) + \alpha^2 Var(W) + 2\alpha Cov(Z, W) = \sigma_Z^2 + \alpha^2\sigma_W^2 + 2\alpha\sigma_{Z,W}^2.$$

Usando il calcolo differenziale, si mostra facilmente che la varianza viene minimizzata per $\alpha = -\frac{\sigma_{Z,W}^2}{\sigma_W^2}$ e che il valore minimo è $\sigma_{CMC}^2(1 - \rho^2)$, dove ρ è il coefficiente di correlazione di Z e W. L'ideale, perciò, sarebbe quello di utilizzare

$$\hat{z} + \alpha(\hat{w} - w)$$

come stimatore, dove $\hat{z}$ e $\hat{w}$ sono gli stimatori CMC. Ma questo richiede la conoscenza di α. Quindi, ad α si preferisce sostituire la stima

$$\hat{\alpha} = -\frac{s_{Z,W}^2}{s_W^2},$$

dove $s_{Z,W}^2$ e s_W^2 indicano la covarianza e varianza campionarie. Allora si ottiene lo stimatore

$$\hat{z}_{CV} = \hat{z} + \hat{\alpha}(\hat{w} - w)$$

che ha le stesse proprietà asintotiche del precedente e varianza asintotica $\sigma_{CMC}^2(1-\rho^2)/N$. Perciò, l'obiettivo sarebbe quello di riuscire ad avere $|\rho|$ il più vicino ad 1 possibile. In generale, è difficile conoscere il valore di ρ a priori e si cerca di rendere Z e W "il più dipendenti possibili" seguendo l'intuito. In ogni caso, anche non trovando una W con una buona correlazione, la varianza non aumenta mai e quindi non peggiora "la bontà" dello stimatore.

3.4 Conditional Monte Carlo

L'obiettivo di questa tecnica è quello di sostituire la v.a. $Z = Z_{CMC}$ con $Z_{Cond} = \mathbb{E}[Z|W]$, dove W è una variabile aleatoria arbitraria. Chiaramente, Z_{Cond} è uno stimatore corretto di z visto che, per le proprietà della media condizionata,

$$\mathbb{E}[Z_{Cond}] = \mathbb{E}[\mathbb{E}[Z|W]] = \mathbb{E}[Z] = z.$$

Perciò si definisce $\hat{z}_{Cond}$ come lo stimatore CMC ottenuto utilizzando copie i.i.d. di Z_{Cond}. Questa tecnica fornisce sempre riduzione della varianza:

$$\begin{aligned}
\sigma^2_{Z_{cond}} &= Var(Z_{Cond}) = Var(\mathbb{E}[Z_{CMC}|W]) \\
&\leq Var(\mathbb{E}[Z_{CMC}|W]) + \mathbb{E}[Var[Z_{CMC}|W]] \\
&= \mathbb{E}[(\mathbb{E}[Z|W])^2] - (\mathbb{E}[\mathbb{E}[Z|W]])^2 + \mathbb{E}[\mathbb{E}[(Z - \mathbb{E}[Z|W])^2|W]] \\
&= \mathbb{E}[(\mathbb{E}[Z|W])^2] - (\mathbb{E}[Z])^2 + \mathbb{E}[Z^2] - \mathbb{E}[(\mathbb{E}[Z|W])^2] \\
&= Var(Z_{CMC}) = \sigma^2_{CMC}
\end{aligned}$$

per le proprietà della media condizionata ($Var[Z_{CMC}|W]$ è la varianza condizionata).

La difficoltà del metodo sta nel trovare W tale che il valore atteso condizionato sia calcolabile. Infine, per avere un effettivo vantaggio pratico:

- W dovrebbe essere facilmente generabile;
- il valore atteso condizionato dovrebbe essere facilmente calcolabile;
- $\mathbb{E}[Var[Z_{CMC}|W]]$ dovrebbe essere relativamente più grande di $Var(\mathbb{E}[Z_{CMC}|W])$ per avere un guadagno considerevole in termini di precisione.

Capitolo 4

Esempi e risultati numerici

Questo capitolo è diviso in due sezioni: nella prima vengono riportati i risultati del metodo CMC nel caso dell'integrazione delle funzioni test di Genz, mentre nella seconda sono presentati degli esempi di implementazione delle tecniche di riduzione della varianza nel contesto del calcolo di Pi Greco. Tutti i risultati sono stati ottenuti tramite simulazione in C++. I codici utilizzati possono essere trovati in appendice.

4.1 Funzioni test

Come detto sopra, lo scopo di questa sezione è quello di riportare i risultati del metodo Monte Carlo sul pacchetto delle funzioni test di Genz[2]. Il pacchetto è stato costruito per il problema dell'integrazione numerica ed è costituito da 6 famiglie di funzioni definite su $[0,\,1]^d$. Ogni famiglia è caratterizzata da un suo attributo:

- **Oscillatory**: $f_1(x) = \cos(2\pi w_1 + \sum_{i=1}^{d} c_i x_i)$;
- **Product peak**: $f_2(x) = \prod_{i=1}^{d} (c_i^{-2} + (x_i - w_i)^2)^{-1}$;
- **Corner peak**: $f_3(x) = (1 + \sum_{i=1}^{d} c_i x_i)^{-(d+1)}$;
- **Gaussian**: $f_4(x) = \prod_{i=1}^{d} \exp(-\sum_{i=1}^{d} c_i^2 (x_i - w_i)^2)$;
- **Continuous**: $f_5(x) = \prod_{i=1}^{d} \exp(-\sum_{i=1}^{d} c_i |x_i - w_i|)$;
- **Discontinuous**: $f_6(x) = \begin{cases} 0 & \text{if } x_1 \geq w_1 \text{ or } x_2 \geq w_2, \\ \exp(\sum_{i=1}^{d} c_i x_i) & \text{otherwise.} \end{cases}$

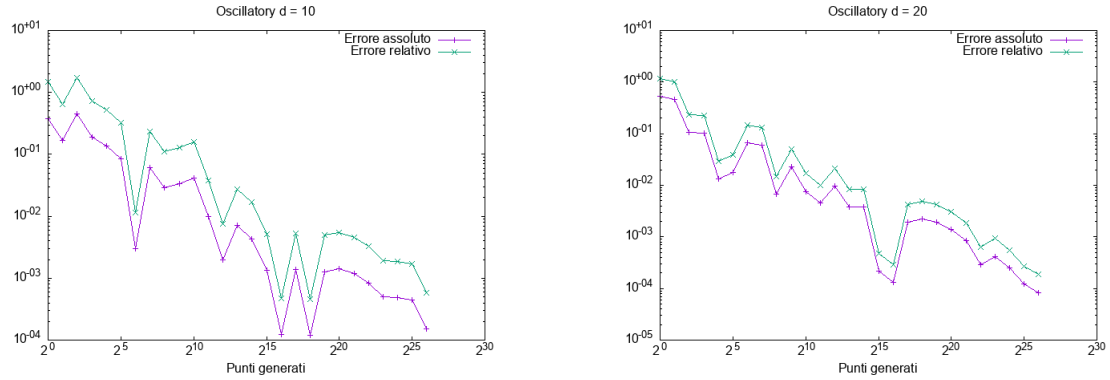
I vettori $w = (w_1, \ldots, w_d), c = (c_1, \ldots, c_d)$ sono parametri che possono essere variati per modificare la funzione test all'interno di una stessa famiglia. Il vettore w agisce come parametro di traslazione e non dovrebbe influenzare la difficoltà del problema finché $w \in [0,\,1]^d$. Invece, il vettore c, che si suppone avere componenti tutte positive, influenza la "difficoltà" della funzione integranda al crescere di $||c||_2$. Un modo per scegliere l'integranda all'interno di una famiglia è quello di generare casualmente w e c. Inoltre, per fissare il livello di difficoltà, è conveniente generare c in modo che

$$\sum_{i=1}^{d} c_i = b,$$

dove b è una costante arbitraria che "misura" qualitativamente la difficoltà.

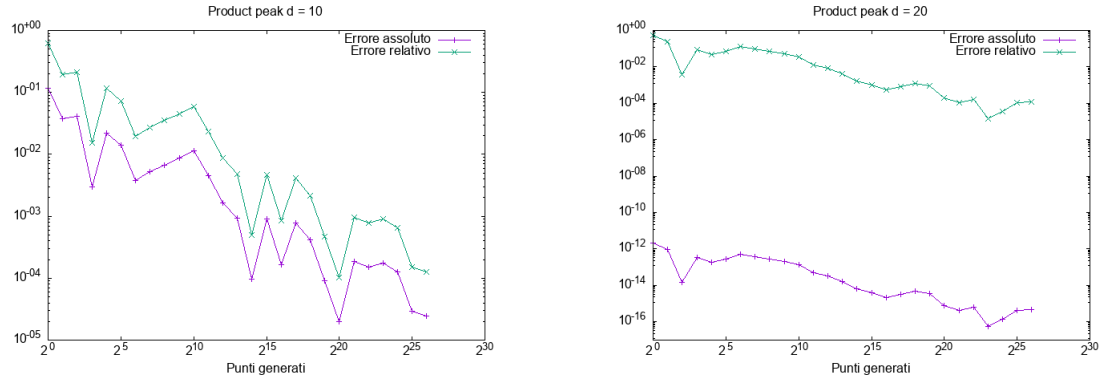
Per testare il metodo su ciascuna delle famiglie, si è semplicemente implementata l'integrazione Monte Carlo, adattando opportunamente l'algoritmo CMC (Welford) e ponendo il valore di $b = 14$. Inoltre, è stato calcolato l'errore di approssimazione della soluzione per $N \leq 10^8$ che sono potenze di 2, dove N rappresenta il numero di punti generati.

Nei grafici che seguono, si possono osservare i risultati ottenuti sulle funzioni rappresentanti di ciascuna famiglia. Il seme usato per i risultati presentati è 684519747630582831, ma sono stati condotti esperimenti anche con altri semi, ottenendo esiti simili che non si riportano per evitare ridondanza.



(a) Dimensione d=10 (b) Dimensione d=20

Figura 4.1: Funzioni oscillatory

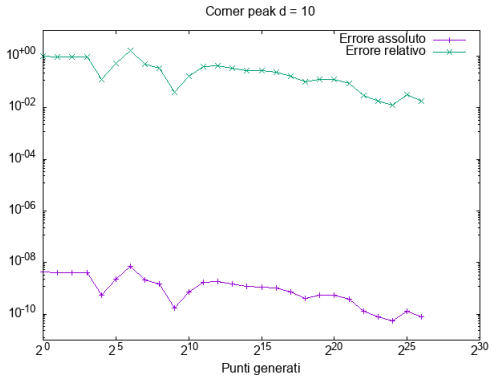


(a) Dimensione d=10 (b) Dimensione d=20

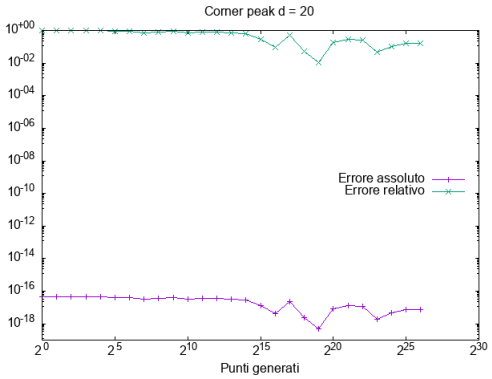
Figura 4.2: Funzioni product peak

Come si nota dai vari grafici, il metodo converge per ogni rappresentante. Questo era un risultato atteso, considerato che le condizioni poste sull'integranda dal metodo Monte Carlo sono molto deboli.

Un altro fatto da notare è che la performance del metodo non sembra influenzata dalla dimensione del problema, infatti l'errore relativo non sembra essere influenzato da d in modo sensibile. Anche questo era un risultato atteso, visto che il RMSE non dipende da d.

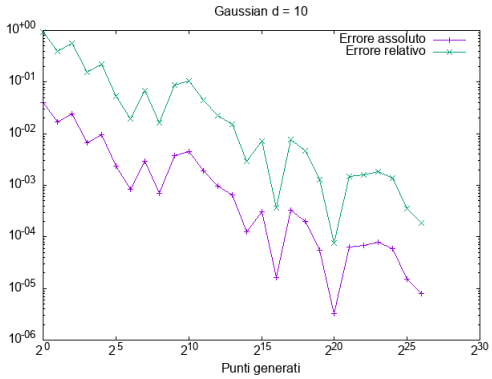


(a) Dimensione d=10

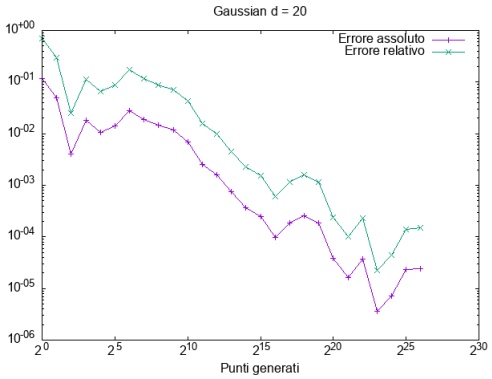


(b) Dimensione d=20

Figura 4.3: Funzioni corner peak

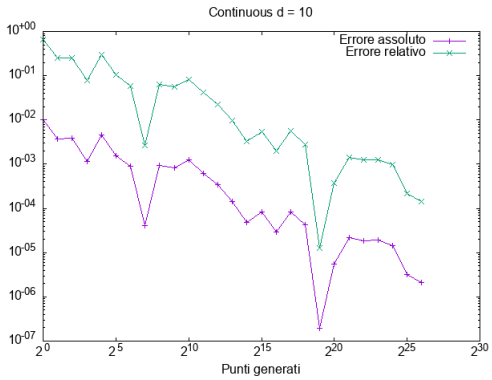


(a) Dimensione d=10

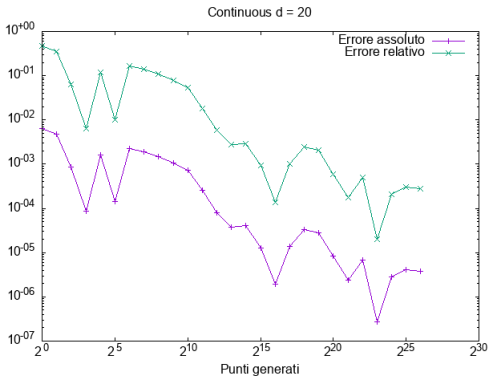


(b) Dimensione d=20

Figura 4.4: Funzioni gaussian



(a) Dimensione d=10

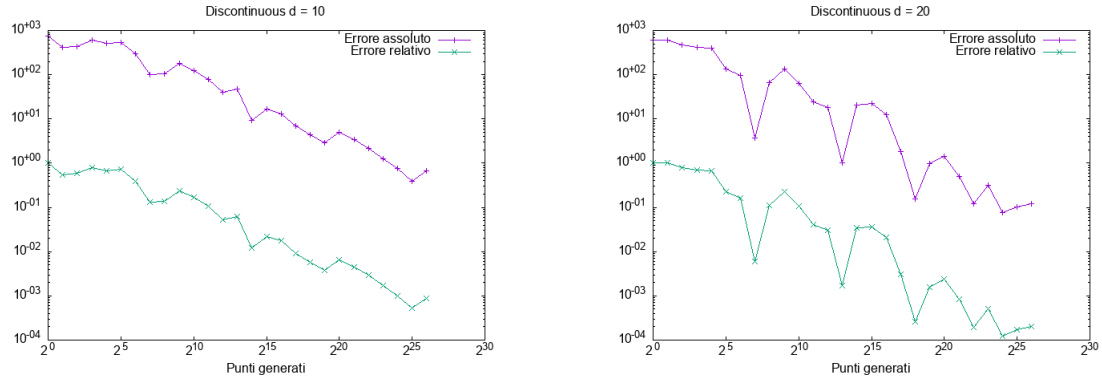


(b) Dimensione d=20

Figura 4.5: Funzioni continuous

Nel caso delle funzioni corner peak si nota che l'errore relativo decade più lentamente: a parità di punti campionati, l'errore relativo non scende sotto 10^{-2}, mentre l'ordine di grandezza è circa 10^{-4} negli altri casi. Questo può essere spiegato da due motivi principali:

- la famiglia di funzioni in questione ha il suo massimo nell'origine e la "massa" dell'integrale tende a concentrarsi vicino ad essa al crescere del parametro b;



(a) Dimensione d=10 (b) Dimensione d=20

Figura 4.6: Funzioni discountinuous

- il metodo è stato implementato considerando un vettore uniformemente distribuito e, perciò, che campiona abbastanza facilmente in zone del dominio di integrazione che contribuiscono di meno al valore dell'integrale.

Infatti, prendendo una b più piccola, si riescono ad ottenere risultati paragonabili agli altri casi, ma ciò significa diminuire la difficoltà dell'integrale.

Anche il caso delle funzioni discontinuous sembra anomalo, ma in realtà, come mostrato dall'errore relativo, è in linea con gli altri risultati. L'errore assoluto così grande è spiegato dal fatto che le rappresentanti di questa famiglia, a seconda del valore di $c = (c_1, \ldots, c_d)$, sono le uniche che possono assumere valori più grandi di 1.

4.2 Riduzione della varianza applicata al calcolo di π

In questa sezione sono presenti degli esempi di tecniche di riduzione della varianza applicate al calcolo del Pi greco. I risultati sono confrontati a parità di numero di punti generati $N = 10^6$ e il seme utilizzato per generarli è 684519747630582831. Tutti gli algoritmi sono facilmente adattabili da quello presentato nel capitolo 2.

4.2.1 Pi greco CMC

Si considerino $X, Y \sim U(0,1)$ indipendenti. Sia $Z = 4\mathbb{1}_{\{X^2+Y^2\leq 1\}}$, allora Z è uno stimatore corretto di Pi greco. Quindi si definisce lo stimatore CMC di Pi greco:

$$\hat{\pi}_{CMC} = \frac{1}{N}\sum_{i=1}^{N} Z_i,$$

dove Z_1, $Z_2, \ldots,$ Z_N sono copie indipendenti di Z. Quindi si ha:

$$\sigma^2_{CMC} = Var(Z) = \pi(4-\pi) \approx 2.6967662$$

I risultati ottenuti dalla simulazione per $N = 10^6$ sono:

- $\hat{\pi}_{CMC} = 3.14348$,
- $s^2_Z = 2.6924562$,

dove s^2_Z è la varianza campionaria di Z. L'errore relativo di approssimazione di Pi è $e_r \approx 6.0076102 \cdot 10^{-4}$, che è stato stimato utilizzando il valore M_PI in C++. Si può, infine, costruire un intervallo di confidenza asintotico di livello 0.995 utilizzando il 0.9975-quantile della distribuzione normale standard $q = q_{0.9975} = 2.81$:

$$\pi \in \left(\hat{\pi}_{CMC} - q\sqrt{\frac{s^2_Z}{N}},\, \hat{\pi}_{CMC} + q\sqrt{\frac{s^2_Z}{N}}\right) = (3.1388692,\, 3.1480908)\,, \text{ con probabilità } 0.995.$$

4.2.2 Pi e Antithetic Sampling

Si considerino sempre $X, Y \sim U([0,1])$ indipendenti e $Z = 4\mathbb{1}_{\{X^2+Y^2\leq 1\}}$ stimatore corretto di Pi. Si osservi che $f(x,y) = 4\mathbb{1}_{B(0,1)}(x,y)$ soddisfa le ipotesi del corollario 3.2.2.1, prendendo $\phi = \psi = f$ e scegliendo $T_1(U) = T_2(U) = 1 - U$. Si possono, quindi, prendere Z_{2i-1}, per $i = 1, \ldots, N$, copie indipendenti di Z e $Z_{2i} = 4\mathbb{1}_{B(0,1)}(1-X, 1-Y) = 4\mathbb{1}_{\{(1-X)^2+(1-Y)^2\leq 1\}}$. Ora, si definisce lo stimatore AS di Pi come

$$\hat{\pi}_{Anth} = \frac{1}{N}\sum_{i=1}^{N} \frac{Z_{2i-1} + Z_{2i}}{2},$$

che ha varianza

$$\sigma^2_{Anth} = \frac{\sigma^2_{CMC}}{2N}(1+\rho),$$

dove ρ è il coefficiente di correlazione di Z_1 e Z_2, e che si sa essere non positivo per il corollario 3.2.2.1.

L'implementazione dell'algoritmo è analoga a quella della sezione precedente.

I risultati ottenuti dalla simulazione per $N = 10^6$ sono:

- $\hat{\pi}_{Anth} = 3.142302$,
- $s_W^2 = 0.97975112$,

dove $W = \frac{Z_1+Z_2}{2}$.

L'errore relativo di approssimazione di Pi è $e_r \approx 2.2579198 \cdot 10^{-4}$ e l'intervallo di confidenza asintotico di livello 0.995 che si ottiene utilizzando $q = q_{0.9975} = 2.81$ è:

$$\pi \in \left(\hat{\pi}_{Anth} - q\sqrt{\frac{s_W^2}{N}},\ \hat{\pi}_{Anth} + q\sqrt{\frac{s_W^2}{N}} \right) = (3.1395206,\ 3.1450834)\,, \text{ con probabilità } 0.995.$$

Infine, si osserva che

$$\frac{\sigma_{Anth}^2}{\sigma_{CMC}^2/N} = \frac{1+\rho}{2} \approx \frac{s_W^2}{s_Z^2} \approx 0.36388749,$$

verificando una riduzione non indifferente della varianza (e quindi la negatività di ρ).

4.2.3 Pi e Control Variates

Si considerino X, Ye Z come sopra. Ora, si definisca $W := 4\mathbb{1}_{\{X+Y>\sqrt{2}\}}$, che ha valore atteso $w = \mathbb{E}[W] = 2(2-\sqrt{2})^2$. L'idea geometrica dietro la scelta di W è ben resa dalla figura 4.7. Quindi, Z e W, intuitivamente, dovrebbero essere "molto correlate". A questo

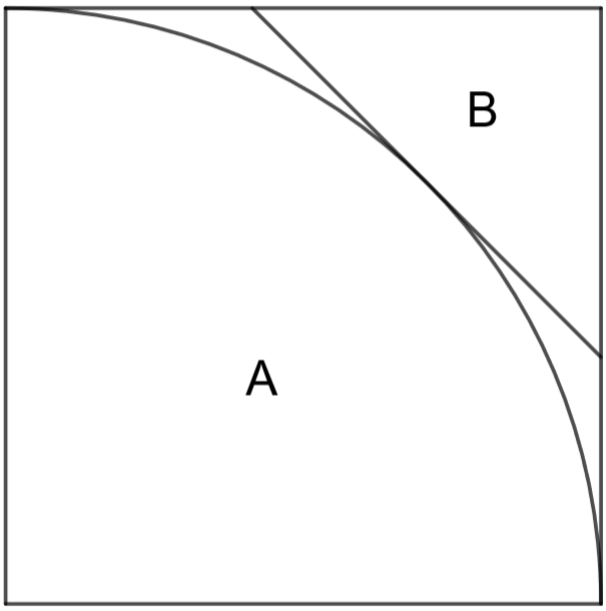


Figura 4.7: A è la regione in cui Z è non nulla, mentre B è la regione dove W è non nulla

punto, si considera lo stimatore

$$\hat{\pi}_{CV} = \hat{z} + \hat{\alpha}(\hat{w} - w),$$

dove $\hat{z}$, $\hat{w}$ sono gli stimatori CMC (cioè le medie empiriche) di Z e W rispettivamente. Invece, $\hat{\alpha} = -\frac{s^2_{Z,W}}{s^2_W}$, dove $s^2_{Z,W}$ è la covarianza campionaria di Z e W, mentre s^2_W è la varianza campionaria di W. Adattando l'algoritmo visto precedentemente per calcolare contemporaneamente $\hat{z}, \hat{w}, s^2_{Z,W}$e s^2_W, per $N = 10^6$ si ottiene:

- $\hat{\pi}_{CV} = 3.1423956$,
- $\hat{\alpha} = 0.94830176$.

Da questi risultati si può vedere che:

- $e_r \approx 2.5559086 \cdot 10^{-4}$,
- il fattore di riduzione della varianza vale $1 - \rho^2 \approx 1 - \frac{(s^2_{Z,W})^2}{s^2_W \cdot s^2_Z} = 0.24143388$.

Infine, si può costruire un intervallo di confidenza asintotico:

$$\pi \in \left(\hat{\pi}_{CV} - q\sqrt{\frac{s^2_Z(1-\rho^2)}{N}},\ \hat{\pi}_{CV} + q\sqrt{\frac{s^2_Z(1-\rho^2)}{N}} \right) = (3.14013,\ 3.1446612)\text{, con probabilità } 0.995.$$

4.2.4 Pi e Conditional MC

Siano X, Y e Z come in precedenza. Si consideri

$$Z_{Cond} \coloneqq \mathbb{E}[Z|Y] = 0 \cdot \mathbb{P}(Z = 0|Y) + 4 \cdot \mathbb{P}(Z = 4|Y) = 4\mathbb{P}(Z = 4|Y)$$

Ma $\mathbb{P}(Z = 4|Y = y) = \int_{[0,1]} \mathbb{1}_{\{X^2+Y^2\leq 1\}} \frac{f_{X,Y}(x,y)}{f_Y(y)} dx = \int_0^{\sqrt{1-y^2}} dx = \sqrt{1-y^2}$. Quindi, si ottiene che:

$$Z_{Cond} = 4\sqrt{1-Y^2}.$$

Ora si definisce lo stimatore corretto di Pi $\hat{\pi}_{Cond}$, che non è altro che lo stimatore CMC riferito a Z_{Cond}.

Dalla simulazione per $N = 10^6$ si ha che:

- $\hat{\pi}_{Cond} = 3.1423369$,
- la varianza campionaria è 0.79530753.

Quindi, $e_r \approx 2.3689868 \cdot 10^{-4}$ e l'intervallo di confidenza è:

$$\pi \in \left(\hat{\pi}_{Cond} - q\sqrt{\frac{s^2_{Z_{Cond}}}{N}},\ \hat{\pi}_{Cond} + q\sqrt{\frac{s^2_{Z_{Cond}}}{N}} \right) = (3.1398309,\ 3.1448429)\text{, con probabilità } 0.995.$$

Infine, ecco una tabella riassuntiva dove è presente anche lo speed up (in media) di convergenza, dato dalla radice del rapporto tra la varianza della tecnica usata e la varianza del CMC, e dei grafici che mostrano l'andamento dell'errore relativo delle diverse tecniche su tre semi differenti.

Tecnica	Varianza campionaria	Speed up medio (rispetto CMC)
CMC	2.6924562	1
AS	0.97975112	∼1.658
CV	0.65005013	∼2.035
CondMC	0.79530753	∼1.840

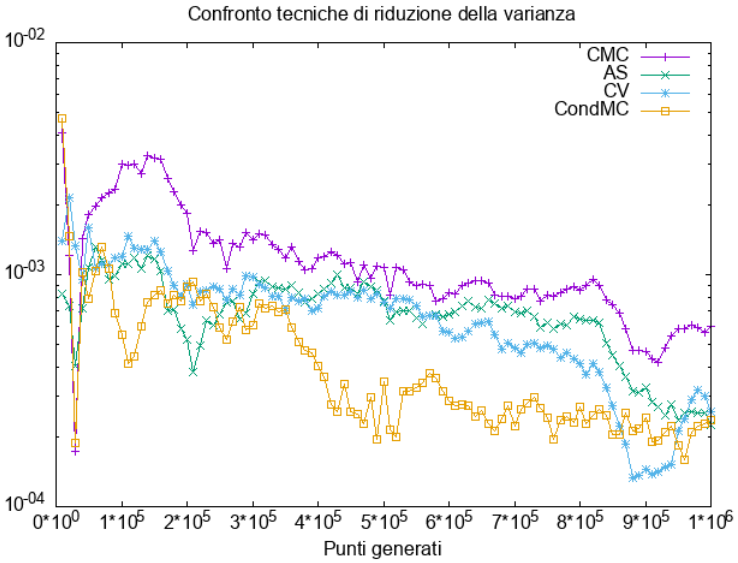


Figura 4.8: Errore relativo delle varie tecniche con seme 684519747630582831

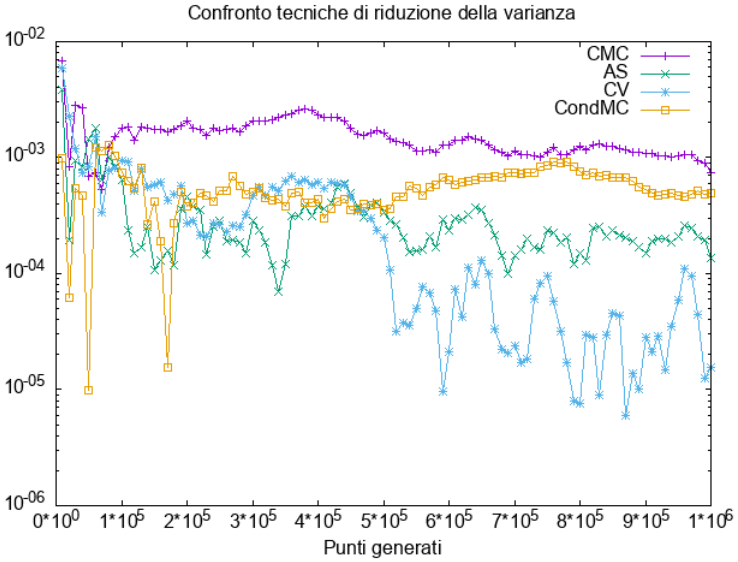


Figura 4.9: Errore relativo delle varie tecniche con seme 171857478208432824

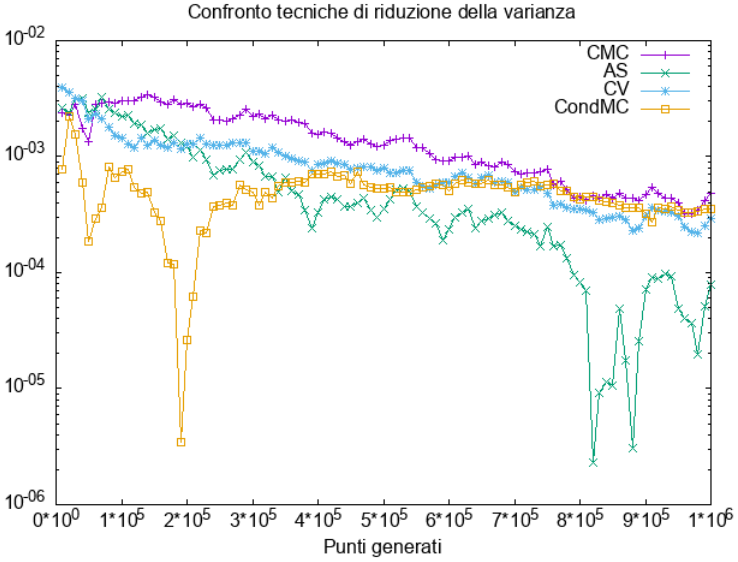


Figura 4.10: Errore relativo delle varie tecniche con seme 8454141231276219 76

Si sottolinea che il RMSE è, appunto, un errore in media che è differente dall'errore effettivamente commesso: gli errori relativi riportati dai grafici sono gli errori su un solo ω nello spazio campionario Ω, quindi alcune tecniche potrebbero performare meglio di altre con varianza più piccola. In questo contesto, però, si sta analizzando un esempio giocattolo in cui si conosce già un'approssimazione molto precisa di Pi greco, che è dovuta ad altri metodi. In casi più concreti, di solito, non si conosce il valore che si vuole calcolare e, quindi, ridurre la varianza diventa particolarmente importante per restringere l'intervallo di confidenza.

Appendix A

Script funzioni test

Qui sono presenti gli script in C++ che sono stati utilizzati per le simulazioni sulle funzioni test. Poiché i codici sono fra loro quasi identici, solo il primo viene riportato nella sua completezza. Per i rimanenti vengono solo riportate parti che li distinguono dal primo: sono tutte funzioni che vanno sostituite a quelle analoghe nel primo codice.

A.1 Oscillatory

```
/*
    File: oscillatory.cpp
    Author: Alessandro Poletto

    Description: Test of Monte Carlo method
    with an oscillatory function in [0,1]^d

*/

// Libraries
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include <random>
#include <cmath>
#include <numeric>
#include <valarray>

using namespace std;

const int d = 20; //dimension

//Prototypes
bool is_integer(double k);
long double func(long double* , long double* , long double* );
long double analytical_eval(long double* , long double* );
```

```
/********* MAIN BEGINNING **********/
int main()
{
    //Variables and constants
    long long mtseed = 6845197476305828 31;   //seed for the rng
                       //17185747820843 2824;
                       //78676165725123 9849;
    long int iter_max = pow(10, 8);          //number of points
    long double b = 14, k;        //diff. coeff., auxiliary constant
    long double w[d], c[d], x[d];  //shift, dilation, point arrays
    long double delta, variate;    //auxiliary variable, variate
    long double sample_mean = 0;   //sample mean

    //sets output format
    cout.setf(ios::scientific);
    cout.precision(8);

    ofstream file;
    //sets output format
    file.setf(ios::scientific);
    file.precision(8);

    //Builds random number generator
        mt19937_64 rng(mtseed);          //Mersenne twister
        rng.discard(700000);             //discards the first 700000 numbers

        //Builds uniform distribution on [0,1]
        uniform_real_distribution<double> dist(0.0,1.0);

        //opens the file and starts writing
        file.open("oscillatory.txt", ios::trunc);
        //file << "Dimension: " << d << endl << "Difficulty parameter: ";
        //file << b << endl << endl;
        //file << setw(8) << "w" << setw(14) << " c" << endl;

        //randomly generates w and c
        for(int i = 0; i < d; i ++)
    {
        w[i] = dist(rng);
        c[i] = dist(rng);
    }

    k = b / accumulate(c, c + d, 0.0);

    //adjusts c to the "complexity condition"
    for(int i = 0; i < d; i ++)
    {
        c[i] = k * c[i];
```

```
        //saves the values inside the file
        //file << w[i] << " " << c[i] << endl;
    }

    //calculates the analytical solution
    long double sol = analytical_eval(w, c);
    //cout << sol << endl;

    //file << endl;
    //file << setw(8) << "ea" << setw(14) << " er" << endl;

    //Crude Monte Carlo
    for(int N = 1; N <= iter_max; N ++)
    {
        //Point coordinates
                for(int i = 0; i < d; i ++)
            x[i] = dist(rng);
        //Welford's Algorithm
                variate = func(c, w, x);
                delta = variate - sample_mean;
                sample_mean += (long double) delta / N;

                //saves some values inside the file
                if(is_integer(log2(N)))
        {
             file << log2(N) << " " <<abs(sample_mean - sol);
             file << " " << abs(sample_mean - sol) / abs(sol) << endl;
        }
    }

    //closes file
    file.close();
    return 0;
}
/********* MAIN ENDING **********/

bool is_integer(double k)
{
    return (k == floor(k));
}

//function that gives the value of the integrand at a point
long double func(long double* c, long double* w, long double* x)
{
    return cos(2 * M_PI * *w + inner_product(c, c + d, x, 0.0));
}
```

```
//evaluates the analytical solution
long double analytical_eval(long double* w, long double* c)
{
    long double v[d];                    //auxiliary array
    long double sum = 0;

    //sets v equal to the null array
    for(int i = 0; i < d; i ++)
    {
        v[i] = 0;
    }

    //calculates the answer, iterating on the possible tuples of bounds
    for(int i = 0; i < pow(2, d); i ++)
    {
        sum += pow(-1, d - accumulate(v, v + d, 0.0)) * (((d + 1) % 2) *
        cos(2 * M_PI * *w+ inner_product(v, v + d, c, 0.0)) +
        (d % 2 )* sin(2 * M_PI * *w + inner_product(v, v + d, c, 0.0)));

        //updates v by interpreting it as in binary form and adding 1
        for(int j = 0; j < d; j ++)
        {
            if(v[j] == 0)
            {
                v[j] = 1;
                break;
            }
            v[j] = 0;
        }
    }

    //returns the value
    return pow(-1, (d % 4) / 2) /
    accumulate(c, c + d, 1.0, multiplies<long double>()) * sum;
}
```

A.2 Corner Peak

```
//function that gives the value of the integrand at a point
long double func(long double* c, long double* w, long double* x)
{
    return pow(1 + inner_product(c, c + d, x, 0.0), (-1)*(d + 1));
}

//calculates factorial
long double factorial(long int n)
{
    long double fact = 1;
```

```
    for(long int i = 2; i <= n; i++)
        fact *= (long double) i;

    return fact;
}

//evaluates the analytical solution
long double analytical_eval(long double* w, long double* c)
{
    long double v[d];                          //auxiliary array
    long double sum = 0;

    //sets v equal to the null array
    for(int i = 0; i < d; i ++)
    {
        v[i] = 0;
    }

    //calculates the answer, iterating on the possible tuples of bounds
    for(int i = 0; i < pow(2, d); i ++)
    {
        sum += pow(-1.0, d - accumulate(v, v + d, 0.0))*
         1.0 / (1 + inner_product(c, c + d, v, 0.0));

        //updates v by interpreting it as in binary form and adding 1
        for(int j = 0; j < d; j ++)
        {
            if(v[j] == 0)
            {
                v[j] = 1;
                break;
            }
            v[j] = 0;
        }
    }

    //returns the value
    return (long long int) pow(-1.0, d) /
    (accumulate(c, c + d, 1.0, multiplies<long double>())
     * factorial(d)) * sum;
}
```

A.3 Product Peak

```
//function that gives the value of the integrand at a point
long double func(long double* c, long double* w, long double* x)
{
```

```
    long double prod = 1;

    for(int i = 0; i < d; i ++)
        prod *= 1.0/(pow(*(c + i), -2) + pow(*(x + i) - *(w + i), 2));

    return prod;
}

//evaluates the analytical solution
long double analytical_eval(long double* w, long double* c)
{
    long double prod = 1;

    for(int i = 0; i < d; i ++)
    {
        prod *= *(c + i) *
        (atan(*(c + i) * (1 - *(w + i))) + atan(*(c + i) * *(w + i)));
    }

    //returns the value
    return prod;
}
```

A.4 Gaussian

```
//function that gives the value of the integrand at a point
long double func(long double* c, long double* w, long double* x)
{
    long double prod = 1;

    for(int i = 0; i < d; i ++)
        prod *= exp(-c[i] * c[i] * (x[i] - w[i]) * (x[i] - w[i]));

    return prod;
}

//evaluates the analytical solution
long double analytical_eval(long double* w, long double* c)
{
    long double prod = 1;

    for(int i = 0; i < d; i ++)
    {
        prod *= 1.0 / c[i] * 1.0 / M_2_SQRTPI *
        (erf(c[i] * (1 - w[i])) + erf(c[i] * w[i]));
    }

    //returns the value
```

```
    return prod;
}
```

A.5 Continuous

```
//function that gives the value of the integrand at a point
long double func(long double* c, long double* w, long double* x)
{
    long double prod = 1;

    for(int i = 0; i < d; i ++)
        prod *= exp(-c[i] * abs(x[i] - w[i]));

    return prod;
}

//evaluates the analytical solution
long double analytical_eval(long double* w, long double* c)
{
    long double prod = 1;

    for(int i = 0; i < d; i ++)
    {
        prod *= 1.0 / c[i] *
        (2 - exp(c[i] * (w[i] - 1)) - exp(-c[i] * w[i]));
    }

    //returns the value
    return prod;
}
```

A.6 Discontinuous

```
//function that gives the value of the integrand at a point
long double func(long double* c, long double* w, long double* x)
{
    if(x[0] > w[0] || x[1] > w[1])
        return 0;

    return exp(inner_product(c, c + d, x, 0.0));
}

//evaluates the analytical solution
long double analytical_eval(long double* w, long double* c)
{
    long double prod = 1;

    for(int i = 2; i < d; i ++)
```

```
    {
        prod *= 1.0 / c[i] * (exp(c[i]) - 1);
    }

    //returns the value
    return 1.0 / (c[0] * c[1]) * (exp(c[0] * w[0]) - 1)
    * (exp(c[1] * w[1]) - 1) * prod;
}
```

Appendix B

Script sul calcolo di Pi greco

B.1 Pi greco CMC

```
/*
    File: CMCpi.cpp
    Author: Alessandro Poletto

    Description: The code estimates pi
    using the Crude Monte Carlo method

*/

// Libraries
#include <iostream>
#include <iomanip>
#include <random>
#include <cstdlib>
#include <cmath>

using namespace std;

//Prototypes
int indicator_function(long double x, long double y);
void welford_step(long double* , long double variate, int i);
bool is_integer(double k);

/********************* MAIN BEGINNING **********************/
int main()
{
    // Variables and constants
        int N;                          //Number of iterations
        long double x, y;               //Point coordinates
        long double variate;            //value of the random variable
        long double sample_mean;        //sample mean
        long double sample_var_unb;     //sample variance
        long double half_width;         //half width of the confidence inter
```

```
const double q = 2.81;          //0.9975-quantile of the std norm distr
long long mtseed = 6845197476305828 31;    //seed for the rng
long double v[2] = {0, 0};                 //auxiliary array

//sets output precision
cout.precision(8);

// Builds random number generators
    mt19937_64 rng(mtseed);          // Mersenne twister
    rng.discard(700000);             // discards the first 700000 numbers

    // Builds uniform distribution in [0, 1]
    uniform_real_distribution<double> dist(0.0,1.0);

    //Input of the number of iterations
    cout << "Number of points to be generated: ";
    cin >> N;

    //Loop initialization
    for (int i = 1; i <= N; i++)
    {
        //Point coordinates
            x = dist(rng);
            y = dist(rng);

            //Welford's algorithm for mean and variance
            variate = indicator_function(x, y);
    welford_step(v, variate, i);
    }
//Loop ending

    //Calculates the sample mean and the sample variance
    sample_mean = 4 * v[0];
sample_var_unb = 16 * v[1] / (N - 1);
half_width = q * sqrt(sample_var_unb / N);

//Output on screen of the results
    cout << "Pi approximation: " << sample_mean << endl;

    cout << "Relative error: ";
    cout << abs(sample_mean - M_PI) / M_PI << endl;

    cout << "Sample variance: " << sample_var_unb << endl;

    cout << "Asymptotic confidence interval of level 0.995: (";
    cout << sample_mean - half_width << ", ";
    cout << sample_mean + half_width << ")" << endl;
```

```
        cout << "Interval width: " << 2 * half_width << endl;

        return 0;
}
/********************* MAIN ENDING **********************/

//indicator function of the unit circle centered in the origin
int indicator_function(long double x, long double y)
{
    if( x * x + y * y <= 1.0)
        return 1;

    return 0;
}

//Welford's step
void welford_step(long double* v, long double variate, int i)
{
    long double delta = variate - *v;
    *v += delta / i;
    *(v + 1) += (long double) (i - 1) / i * delta * delta;
}

bool is_integer(double k)
{
    return (k == floor(k));
}
```

B.2 Pi greco AS

```
/*
    File: CMCpi.cpp
    Author: Alessandro Poletto

    Description: The code estimates pi using the Monte Carlo method
    using Antithetic Sampling to reduce variance.
    So if Z_1 is the indicator function of {X^2 + Y^2 <= 1},
    then Z_2 is the indicator function of {(1-X)^2 + (1-Y)^2 <= 1}.

*/

// Libraries
#include <iostream>
#include <iomanip>
#include <random>
#include <cstdlib>
#include <cmath>
```

```
using namespace std;

//Prototypes
int indicator_function(long double x, long double y);
void welford_step(long double* , long double variate, int i);
bool is_integer(double k);

/********************* MAIN BEGINNING **********************/
int main()
{
    //Variables
        int N;                          //number of iterations
        int z_1, z_2;                   //random variates
        long double x, y;               //point coordinates
        long double variate;            //value of the random variable
        long double mean = 0, var;      //sample mean and sample variance
        long double half_width;         //half width of the confidence interval
    long long mtseed = 6845197476305828 31;   //seed for the rng
    const double q = 2.81;          //0.9975-quantile of the std norm distr
    long double v[2] = {0, 0};      //auxiliary array

    //sets output precision
    cout.precision(8);

    // Builds random number generators
        mt19937_64 rng(mtseed);             // Mersenne twister
        rng.discard(700000);                // discards the first 700000 numbers

        // Builds uniform distribution in [0, 1]
        uniform_real_distribution<double> dist(0.0,1.0);

        //Input of the number of iterations
        cout << "Number of points to be generated: ";
        cin >> N;

        //Loop initialization
        for (int i = 1; i <= N; i++)
        {
            //generation of the point
                x = dist(rng);
                y = dist(rng);

        //generates variates
                z_1 = indicator_function(x, y);
                z_2 = indicator_function(1 - x, 1 - y);
                variate = (long double) (z_1 + z_2) / 2;

                //Welford's algorithm for mean and variance
```

```
        welford_step(v, variate, i);
        }
    //Loop ending

        mean = 4 * v[0];
    var = 16 * v[1] / (N - 1);
        half_width = q * sqrt(var / N);

    //Output on screen of the results
        cout << "Pi approximation: " << mean << endl;
        cout << "Relative error: " << abs(mean - M_PI) / M_PI << endl;
        cout << "Sample variance: " << var << endl;
        cout << "Asymptotic confidence interval of level 0.995: (";
        cout << mean - half_width << ", ";
        cout << mean + half_width << ")" << endl;
        cout << "Interval width: " << 2 * half_width << endl;

        return 0;
}
/********************* MAIN ENDING **********************/

//indicator function of the unit circle centered in the origin
int indicator_function(long double x, long double y)
{
    if( x * x + y * y <= 1.0)
        return 1;

    return 0;
}

//Welford's step
void welford_step(long double* v, long double variate, int i)
{
    long double delta = variate - *v;
    *v += delta / i;
    *(v + 1) += (long double) (i - 1) / i * delta * delta;
}

bool is_integer(double k)
{
    return (k == floor(k));
}
```

B.3 Pi greco CV

```
/*
    File: CVpi.cpp
    Author: Alessandro Poletto
```

```
    Description: The code estimates pi using the Monte Carlo method and
    the control variates method to reduce variance.
    In this case the control variate is the indicator function of
    the event R = {X + Y > sqrt(2)} with expected value 2*(2-sqrt(2))^2.

*/

// Libraries
#include <iostream>
#include <iomanip>
#include <random>
#include <cstdlib>
#include <cmath>

using namespace std;

//Prototypes
int indicator_function_circle(long double x, long double y);
int indicator_function_R(long double x, long double y);
void welford_step(long double* , long double variate_z ,
                  long double variate_w, int i);
bool is_integer(double k);

/********************* MAIN BEGINNING ***********************/
int main()
{
    // Variables and constants
        int N;                                      //number of iterations
        long double x, y;                           //point coordinates
        long double z, w; //variates: original estimator, control variate
        long double z_mean, w_mean;                 //sample means
        long double pi_approximation;               //pi approximation
        long double sample_var_unb_z, sample_var_unb_w; //variances
        long double sample_covar_zw, alpha;         //covariance, alpha coeff
        long double factor;                       //factor of variance reduction
        long double sample_variance;                //sample variance
    long double half_width;       //half width of the confidence interval
    long long mtseed = 6845197476305828 31;     //seed for the rng
    const double q = 2.81;          //0.975-quantile of the std norm distr
    long double v[5] = {0, 0, 0, 0, 0};         //auxiliary array

    //sets output precision
    cout.precision(8);

         // Builds random number generators
        mt19937_64 rng(mtseed);             // Mersenne twister
        rng.discard(700000);                // discards the first 700000 numbers
```

```
    // Builds uniform distribution on [0,1]
    uniform_real_distribution<double> dist(0.0,1.0);

    //Input of the number of iterations
    cout << "Number of points to be generated: ";
    cin >> N;

    //Loop initialization
    for (int i = 1; i <= N; i++)
    {
        //generates point coordinates
            x = dist(rng);
            y = dist(rng);

            //generates variates
    z = indicator_function_circle(x, y);
    w = indicator_function_R(x, y);

            //Welford's algorithm
            welford_step(v, z, w, i);
    }
//Loop ending

//calculates means, variances, alpha coefficient, ecc.
    z_mean = 4 * v[0];
    w_mean = 4 * v[1];
sample_var_unb_w = 16 * v[3] / (N - 1);
sample_var_unb_z = 16 * v[2] / (N - 1);;
sample_covar_zw = 16 * v[4] / (N - 1);;
alpha = -sample_covar_zw / sample_var_unb_w;

factor = 1 - sample_covar_zw * sample_covar_zw /
(sample_var_unb_w * sample_var_unb_z);

sample_variance = sample_var_unb_z * factor;
half_width = q * sqrt(sample_variance / N);

pi_approximation = z_mean + alpha *
(w_mean - 2 * (2.0-M_SQRT2) * (2.0-M_SQRT2));

//Output on screen of the results
    cout << "Pi approximation: " << pi_approximation << endl;
    cout << "Relative error: " << abs(pi_approximation - M_PI) / M_PI
    cout << endl;
    cout << "Alpha: " << alpha << endl;
    cout << "Factor of variance reduction: " << factor << endl;
    cout << "Sample variance: " << sample_variance << endl;
```

```
        cout << "Asymptotic confidence interval of level 0.995: (";
        cout << pi_approximation - half_width << ", ";
        cout << pi_approximation + half_width << ")" << endl;

        cout << "Interval width: " << 2 * half_width << endl;

        return 0;
}
/********************* MAIN ENDING **********************/

//indicator function of the unit circle centered in the origin
int indicator_function_circle(long double x, long double y)
{
    if( x * x + y * y <= 1.0)
        return 1;

    return 0;
}

//indicator function of the event R
int indicator_function_R(long double x, long double y)
{
    if( x + y > M_SQRT2)
        return 1;

    return 0;
}

//Welford's step
void welford_step(long double* v, long double variate_z,
                  long double variate_w, int i)
{
    long double delta_0 = variate_z - *v;
    long double delta_1 = variate_w - *(v + 1);
    *v += delta_0 / i;
    *(v + 1) += delta_1 / i;
    *(v + 2) += (long double) (i - 1) / i * delta_0 * delta_0;
    *(v + 3) += (long double) (i - 1) / i * delta_1 * delta_1;
    *(v + 4) += (long double) (i - 1) / i * delta_0 * delta_1;
}

bool is_integer(double k)
{
    return (k == floor(k));
}
```

B.4 Pi greco Conditional MC

```
/*
    File: ConditionalMCpi.cpp
    Author: Alessandro Poletto

    Description: The code estimates pi using the conditional Monte Carlo
     method (in order to have reduced variance).
    It's considered the indicator function of {X^2 + Y^2 <= 1}
    conditioned with respect to X
*/

// Libraries
#include <iostream>
#include <iomanip>
#include <random>
#include <cstdlib>
#include <cmath>

using namespace std;

//Prototypes
long double func(long double x);
void welford_step(long double* , long double variate, int i);
bool is_integer(double k);

/********************* MAIN BEGINNING **********************/
int main()
{
    // Variables and constants
        int N;                                  //number of points
        long double x;                          //generated point
        long double variate;                    //value of the r.v.
        long double sample_mean;                //Sample mean
        long double sample_var_unb;             //unbiased sample varian
    long double half_width;       //half width of the confidence interval
    long long mtseed = 6845197476305828 31;   //seed for the rng
    const double q = 2.81;          //0.975-quantile of the std norm distr
    long double v[2] = {0, 0};                  //auxiliary array

    //sets output precision
    cout.precision(8);

    // Builds random number generators
        mt19937_64 rng(mtseed);     //Mersenne twister
        rng.discard(700000);        //discards the first 700000 numbers

        // Builds uniform distribution on [0,1]
```

```
        uniform_real_distribution<double> dist_x(0.0,1.0);

    //Input of the number of iterations
        cout << "Number of points to be generated: ";
        cin >> N;

    //Loop initialization
        for (int i = 1; i <= N; i++)
        {
            //generates the point
            x = dist_x(rng);

        //Welford's algorithm for mean and variance
                variate = func(x);
        welford_step(v, variate, i);
        }
    //Loop ending

        //Calculates the sample mean and the sample variance
        sample_mean = 4 * v[0];
    sample_var_unb = 16 * v[1] / (N - 1);
    half_width = q * sqrt(sample_var_unb / N);

    //Output on screen of the results
        cout << "Pi approximation: " << sample_mean << endl;
        cout << "Relative error: " << abs(sample_mean - M_PI) / M_PI;
        cout << endl;
        cout << "Sample variance: " << sample_var_unb << endl;

        cout << "Asymptotic confidence interval of level 0.995: (";
        cout << sample_mean - half_width << ", ";
        cout << sample_mean + half_width << ")" << endl;

        cout << "Interval width: " << 2 * half_width << endl;

        return 0;
}
/********************** MAIN ENDING **********************/

long double func(long double x)
{
    return sqrt(1 - pow(x, 2));
}

//Welford's step
void welford_step(long double* v, long double variate, int i)
{
    long double delta = variate - *v;
```

```
    *v += delta / i;
    *(v + 1) += (long double) (i - 1) / i * delta * delta;
}

bool is_integer(double k)
{
    return (k == floor(k));
}
```

Bibliografia

[1] Søren Asmussen and Peter W. Glynn. *Stochastic Simulation: Algorithms and Analysis*. Stochastic Modelling and Applied Probability. Springer, 2007.

[2] Alan Genz. A package for testing multiple integration subroutines. In Patrick Keast and Graeme Fairweather, editors, *Numerical Integration: Recent Developments, Software and Applications*. Reidel, 1987.

[3] Hans-Otto Georgii. *Stochastics: Introduction to Probability and Statistics*. de Gruyter, 2007.

[4] Achim Klenke. *Probability Theory: A Comprehensive Course*. Universitext. Springer, 2014.

[5] Gilles Pages. *Numerical Probability: An Introduction with Applications to Finance*. Universitext. Springer, 2018.

[6] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo method*. Wiley Series in Probability and Statistics. Wiley, 2016.

[7] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.