Corso di Fisica Computazionale

- Elaborato finale -

METODI MONTE CARLO PER IL PRICING DI OPZIONI EUROPEE ED ASIATICHE

Garbi Luca

Issue: 1

30 ottobre 2020

Università degli Studi di Trento
Dipartimento di Fisica
Via Sommarive 14, 38123
Povo (TN), Italia

Indice

1 Metodi Monte Carlo per il prezzo di una *Call Europea* 1

1.1 Richiamo teorico 1

1.1.1 L'equazione di Black-Scholes 1

1.1.2 Misura neutrale al rischio 3

1.1.3 *Antithetic Variates* 4

1.2 Descrizione del codice 5

1.3 Discussione dei risultati e confronto 6

2 Metodi Quasi-Monte Carlo per il *pricing* di Opzioni Asiatiche. 8

2.1 Richiamo teorico 8

2.1.1 Opzioni Asiatiche 8

2.1.2 *Quasi-Monte Carlo* - La sequenza di Halton 9

2.2 Descrizione del codice 12

2.3 Discussione dei risultati e confronto 13

A Risultati teorici 18

A.1 L'equazione di Black-Scholes 18

A.2 Risoluzione analitica dell'equazione di Black-Scholes 19

B Codice 22

B.1 Codice in linguaggio C - Opzione Europea 22

B.2 Codice in linguaggio C - Opzione Asiatica 28

B.3 Script MATLAB per grafici 33

Abstract

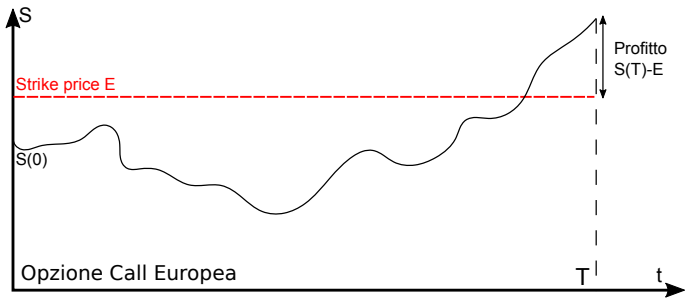
In questo elaborato finale vengono implementati metodi Monte Carlo e Quasi-Monte Carlo per il *pricing* di strumenti derivati, la cui dinamica è data dal modello finanziario di Black-Scholes-Merton. In particolare, nella prima breve sezione viene proposta un'alternativa, che implementa il metodo MC, alla risoluzione dell'equazione alle derivate parziali di Black-Scholes, la quale viene confrontata con metodi risolutivi alle differenze finite. Nella seconda parte invece viene fatto un confronto tra approccio QMC e MC riguardo al problema di *pricing* di opzioni Asiatiche.

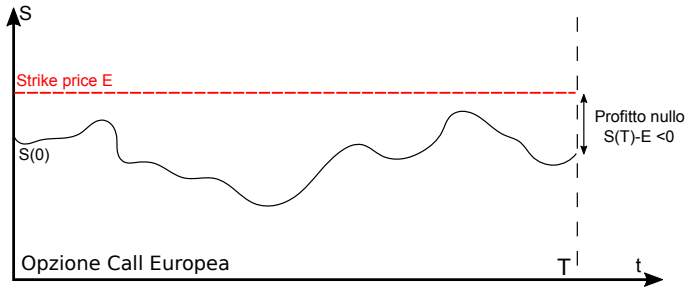
1 Metodi Monte Carlo per il prezzo di una *Call Europea*

1.1 Richiamo teorico

1.1.1 L'equazione di Black-Scholes

Un'opzione, strumento finanziario appartenente alla famiglia dei contratti derivati, fornisce al suo compratore il diritto (ma non l'obbligo) di comprare ad un prezzo prefissato e per un premio prefissato una particolare attività finanziaria, chiamata sottostante. In questa trattazione ci si concentra solamente sulle Opzioni *Call* (la soluzione è comunque analoga per Opzioni *Put*) di tipo Europeo, ovvero con la possibilità di esercitare il contratto di acquisto o meno (se il prezzo d'esercizio sarà vantaggioso rispetto al prezzo di mercato corrente del sottostante) solo alla data di scadenza. In particolare, il tipo di opzione analizzato è il *Plain Vanilla*, senza variazioni di tipo esotico (una schematizzazione è presente nella coppia di figure seguenti).





Per quanto riguarda la notazione: verrà utilizzato $S(t)$ per indicare il prezzo dell'*asset* sottostante al tempo t, con $V(S, t)$ si indicherà il prezzo dell'opzione come funzione di S al tempo t, mentre E sarà lo *strike price* dell'opzione, ovvero il prezzo di esercizio alla scadenza del contratto, che assumiamo avvenire al tempo $t = T$.
È necessario specificare le assunzioni che vengono fatte per lo sviluppo del modello:

- possibilità di acquisto o vendita di una qualsiasi quantità, anche frazionaria, di sottostante S e di un *asset* senza rischio R (tipicamente obbligazioni) con un tasso di interesse composto costante r;
- l'*asset* non paga alcun dividendo, se non il premio di vendita, e nessuna transazione è soggetta a commissione di sorta;
- l'andamento dell'*asset* in funzione del tempo è un processo markoviano e segue un moto browniano geometrico con *drift* e volatilità costanti;
- sul mercato non c'è opportunità di arbitraggio, ovvero non è possibile ottenere un profitto (con ritorno superiore a quello dato da r) senza rischi.

L'intuizione finanziaria dietro al modello di Black-Scholes-Merton è che sia possibile eliminare completamente il rischio dell'acquisto o vendita di un'opzione attraverso, rispettivamente, la vendita o l'acquisto dell'*asset* sottostante (fenomeno chiamato *hedging*). Di conseguenza, a causa dell'ultima assunzione elencata, ci dovrà essere un *fair price*, ossia un prezzo corretto per il derivato V tale da non permettere di avere un profitto (superiore a quello che si avrebbe investendo nell'asset R) senza rischi. Inoltre, dato il tipo di PDE e di condizioni al contorno che si trovano con questa trattazione, emerge che il *fair price*, per ogni istante di tempo t, è unico e determinato interamente da S, r, E e T.

Secondo l'assunzione che il prezzo del sottostante segua un moto geometrico browniano, possiamo descrivere la variazione di S in un intervallo infinitesimo $\mathrm{d}t$ come

$$\mathrm{d}S = \mu S \mathrm{d}t + \sigma S \mathrm{d}z\,. \tag{1}$$

Qui il parametro assunto noto μ è il *rate* di ritorno atteso per l'*asset* e identifica un *trend* del prezzo, σ invece è un parametro che rappresenta la volatilità del prezzo, ovvero la deviazione standard intorno al prezzo medio dato dal termine di *trend*, $\mathrm{d}z$ infine è un incremento infinitesimo di una variabile stocastica $z(t)$, ben modellizzata da una *random walk*. In definitiva, l'equazione (15) ci dice che, localmente, il tasso di ritorno dell'*asset* ha un valore d'aspettazione $\mu \mathrm{d}t$ ed una varianza $\sigma^2 \mathrm{d}t$.
A partire dalle assunzioni precedenti si ricava (in appendice A.1) **l'equazione di Black-Scholes**:

$$\boxed{\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0} \tag{2}$$

La condizione finale data dall'opzione di esecuzione o meno del contratto al *payoff time* T è

$$V(S, T) = \max\{S - E, 0\}\,, \quad \text{per } S \geq 0\,. \tag{3}$$

Le due condizioni al contorno della equazione differenziale alle derivate parziali sono

$$\begin{cases} V(0,\,t) = 0 & \text{per ogni } t \\ V(S,\,t) \to S & \text{per } S \to \infty \end{cases} \tag{4}$$

Questa particolare PDE per la dinamica di un'Opzione Europea di tipo *Call* ammette una soluzione analitica (la cui risoluzione si trova in appendice A.2):

$$\boxed{V(S,\,t) = S\Phi(d_+) - Ee^{-r(T-t)}\Phi(d_-)} \tag{5}$$

posto $\Phi(d)$ la funzione di ripartizione di una distribuzione gaussiana e posti

$$d_+ \equiv \frac{\log\left(\frac{S}{E}\right) + (T-t)(r+\sigma^2/2)}{\sigma\sqrt{T-t}}, \tag{6}$$

$$d_- \equiv \frac{\log\left(\frac{S}{E}\right) + (T-t)(r-\sigma^2/2)}{\sigma\sqrt{T-t}}. \tag{7}$$

1.1.2 Misura neutrale al rischio

La soluzione dell'equazione di Black-Scholes può essere effettuata con vari metodi alle differenze finite, in seguito se ne utilizzano uno esplicito e uno implicito con decomposizione LU. Alternativamente metodi Monte Carlo si possono utilizzare per simulare andamenti di mercato dell'*asset* sottostante. Ciò può essere fatto in quanto è possibile scrivere il prezzo corrente dell'opzione V come il valore atteso di una grandezza.
Per arrivare a fare ciò si introduce la *misura neutrale al rischio*. Questo strumento emerge dalla forma dell'equazione differenziale di BS, la quale non contiene alcuna variabile che sia influenzata dalle inclinazioni al rischio degli investitori. In altre parole non abbiamo informazioni su quali siano le preferenze riguardo al rischio degli acquirenti di opzioni. Ciò non si potrebbe dire se invece nell'equazione comparisse il trend μ, che convenientemente viene eliso nella derivazione.
Questa proprietà viene utilizzata allora decidendo a priori quali siano le tendenze degli investitori, poiché ciò non cambierebbe la soluzione. In un mondo in cui gli investitori sono neutrali al rischio, il ritorno aspettato su tutti i titoli è quello che si otterrebbe investendo in un *asset* senza rischio, ovvero dato dal tasso di interesse precedentemente chiamato r. In questo mondo quindi si può identificare il trend del prezzo μ con r. Con la misura di neutralità al rischio quindi il valore corrente (al tempo $t = 0$) di un'opzione è uguale a quello che avrà alla scadenza del contratto (al tempo T), però scontato del tasso di interesse che viene maturato in questo lasso di tempo, ovvero e^{-rT}.
Chiamato $\hat{E}$ l'operatore che fornisce il valore aspettato di una funzione in un mondo neutrale al rischio, allora sappiamo che il *payoff* di un'opzione *Call Europea* alla scadenza sarà

$$\hat{E}[\max\{S-E,\,0\}].$$

Di conseguenza per la misura di neutralità al rischio possiamo dire che il valore attuale dell'opzione è

$$V(S,t) = e^{-r(T-t)}\hat{E}[\max\{S-E,\,0\}].$$

Per valutare il valore atteso precedente dobbiamo simulare, con metodi MC, andamenti di prezzo $S(t)$ in accordo con le assunzioni fatte in precedenza.
Dalla relazione sul prezzo (1) e utilizzando la misura di neutralità al rischio ($\mu = r$) possiamo scrivere

$$\frac{\mathrm{d}S(t)}{S(t)} = r\mathrm{d}t + \sigma\mathrm{d}z(t)\,.$$

che integrata sull'intervallo da 0 a T e sfruttando il fatto che $\mathrm{d}z(t) = \frac{\mathrm{d}z}{\mathrm{d}t}\mathrm{d}t$ fornisce

$$S(T) = S(0)e^{(r-\frac{1}{2}\sigma^2)T+\sigma z(T)}\,.$$

dove ricordiamo $z(t)$ essere distribuita normalmente. Chiamando Z una variabile gaussiana standardizzata (media nulla e varianza unitaria), allora riscriviamo un prezzo casuale come

$$S(T) = S(0)\exp\left((r-\frac{1}{2}\sigma^2)T + \sigma\sqrt{T}Z\right).$$

A tal proposito nell'implementazione del programma per campionare due variabili aleatorie η_1 e η_2 distribuite in modo gaussiano, con media nulla e varianza σ_v^2, a partire da due variabili stocastiche ξ_1 e ξ_2 distribuite uniformemente in $[0,\,1]$, sono state utilizzate le cosiddette formule di Box-Muller:

$$\begin{cases} \eta_1 &= \sigma_v\sqrt{-2\ln(1-\xi_1)}\cos(2\pi\xi_2) \\ \eta_2 &= \sigma_v\sqrt{-2\ln(1-\xi_1)}\sin(2\pi\xi_2) \end{cases} \tag{8}$$

1.1.3 *Antithetic Variates*

Un utile metodo per la riduzione della varianza semplicemente implementabile in simulazioni Monte Carlo è quello dell'utilizzo di *Antithetic Variates*. Volendo stimare un'osservabile $\theta = \hat{E}(Y)$ avendo due campioni Y_1 e Y_2, uno stimatore senza bias di θ è dato da

$$\widetilde{\theta} = \frac{Y_1 + Y_2}{2}$$

la cui varianza sarà

$$\mathrm{Var}(\widetilde{\theta}) = \frac{\mathrm{Var}(Y_1) + \mathrm{Var}(Y_2) + 2\mathrm{Cov}(Y_1,\,Y_2)}{4}.$$

Dall'ultima relazione risulta chiara la possibilità di ridurre la varianza se i due campioni sono correlati negativamente.
Nel nostro caso sappiamo che Y è funzione di una variabile X distribuita normalmente, quindi

$$\theta = E(Y) = E(g(X)), \qquad X \sim N(0,1),$$

allora la stima Monte Carlo con variabili aleatorie antitetiche è data da

$$\widetilde{\theta}_a = \frac{1}{N}\sum_{i=1}^{N}\frac{g(X_i)+g(-X_i)}{2} \tag{9}$$

con le $X_i \sim N(0,1)$ i.i.d. Le X_i e $-X_i$ sono dette variabili aleatorie antitetiche e sono negativamente correlate, inoltre la nostra g è monotona, quindi abbiamo una riduzione della varianza usando questo metodo.

1.2 Descrizione del codice

Per la risoluzione della PDE vengono utilizzati due metodi alle differenze finite, uno esplicito e l'altro implicito, in cui il problema è ricondotto all'inversione di una matrice tridiagonale. Per l'applicazione di tali metodi si rende necessaria la suddivisione dell'intervallo temporale $[0,\, T]$ in una *mesh* di $M \in \mathbb{N}$ sottointervalli, ciascuno di ampiezza $\Delta T \equiv T/M$. L'intervallo $[0,\, \infty)$ su cui è definito il prezzo S dell'*asset*, invece, viene limitato superiormente introducendo il limite artificiale $S_{max} \equiv 3E$. Tale assunzione, nonostante permetta di ottenere risultati che, almeno per bassi valori di S, presentano un buon accordo con la soluzione analitica dell'equazione BS, rappresenta comunque un'approssimazione forte, i cui limiti risulteranno evidenti nel confronto tra soluzione esatta e numerica presentato nella sezione seguente. L'intervallo $[0,\, S_{max}]$ su cui si limita la soluzione viene diviso in $N \in \mathbb{N}$ sottointervalli, ciascuno di ampiezza $\Delta S \equiv S_{max}/N$.

$$(n\Delta S,\, m\Delta t) \in [0,\, N\Delta S] \times [0,\, M\Delta t\,],$$

con $n = 0,\, 1, ...,\, N$ e $m = 0,\, 1, ...,\, M$.

Si introduce inoltre il cambio di variabile

$$\mathscr{T} \equiv T - t\,,$$

con cui l'equazione (18) diventa

$$\frac{\partial \tilde{V}}{\partial \mathscr{T}} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 \tilde{V}}{\partial S^2} - rS\frac{\partial \tilde{V}}{\partial S} + r\tilde{V} = 0\,, \tag{10}$$

avendo definito $\tilde{V}(S,\, \mathscr{T}) \equiv V(S,\, T - \mathscr{T})$. Di conseguenza, la condizione finale (19) diventa

$$\tilde{V}(S,\, 0) = \max\{S - E,\, 0\}\,, \tag{11}$$

mentre le condizioni al bordo (20), tenendo conto del limite superiore S_{max} al valore di S, assumono la forma

$$\begin{cases} \tilde{V}(0,\, \mathscr{T}) = 0 \\ \tilde{V}(S_{max},\, \mathscr{T}) = S_{max} - E \end{cases} \tag{12}$$

per ogni $\mathscr{T} \in [0,\, T]$.

Nella risoluzione numerica dell'equazione BS, per entrambi i metodi utilizzati, si fissano per i parametri i valori $T = 0.25$, $E = 20$, $r = 0.1$, $\sigma = 0.4$, e si considerano $N = 1000$ e $M = 50000$.

Per questi valori di M e N l'algoritmo esplicito utilizzato risulta stabile, per quello implicito invece la stabilità è sempre garantita.

I due metodi sopracitati congiuntamente alla stima con metodo Monte Carlo (e MC con antithetic variates) sono stati implementati in un programma in linguaggio C con lo scopo di individuare il *fair price* dell'opzione al momento iniziale. La struttura del codice è molto semplice: nella prima parte la funzione principale *main*, data la coppia di valori (N, M) scelta e ciascuno dei due algoritmi alle differenze finite utilizzati, richiama ciclicamente in modo alternativo le funzioni *passo_esplicito* e *passo_implicito*, incaricate rispettivamente di eseguire un passo dei metodi esplicito e implicito. Inoltre nella seconda parte un ciclo sul numero *N_MC* di passi Monte Carlo richiama le funzioni *price_paths* e *anti_price_paths* che generano prezzi casuali per l'asset sottostante come spiegato nella sezione precedente. La variabile distribuita normalmente viene calcolata utilizzando le trasformate di Box-Muller.

I risultati ottenuti vengono salvati in un file di testo per la successiva elaborazione. Il codice completo è riportato in Appendice B.1.

1.3 Discussione dei risultati e confronto

Al fine di vedere la validità di un approccio Monte Carlo per il pricing di opzioni Europee osserviamo la tabella seguente che riporta i valori dello strumento derivato calcolato al momento iniziale. Sono stati utilizzati 10^6 step Monte Carlo (e Antithetic Monte Carlo), $T = 0.25$, $E = 20$, $r = 0.1$ e $\sigma = 0.4$, ma osservazioni del tutto analoghe a quelle riportate si riscontrano utilizzando altri valori. La prima colonna rappresenta il valore del sottostante S al tempo $t = 0$.

Tabella 1: Sono riportate le soluzioni numeriche e la soluzione analitica (25) dell'equazione BS (18), al tempo $t = 0$ fissato, per alcuni valori del prezzo S dell'*asset*.

S	**S. analitica**	**M. esplicito**	**M. Implicito**	**MC**	σ_{MC}	**AMC**	σ_{AMC}
12	0.00759	0.00760	0.00760	0.0077	0.0001	0.0077	0.0009
18	0.85920	0.85920	0.85920	0.8596	0.0019	0.8583	0.0009
24	4.82881	4.82883	4.82882	4.8318	0.0044	4.8292	0.0009
30	10.52074	10.52072	10.52072	10.523	0.006	10.520	0.001
36	16.49540	16.49437	16.49437	16.505	0.007	16.495	0.001
42	22.49388	22.48290	22.48290	22.498	0.008	22.493	0.001
48	28.49380	28.43517	28.43517	28.490	0.009	28.494	0.001
54	34.49380	34.29503	34.29502	34.49	0.01	34.495	0.002

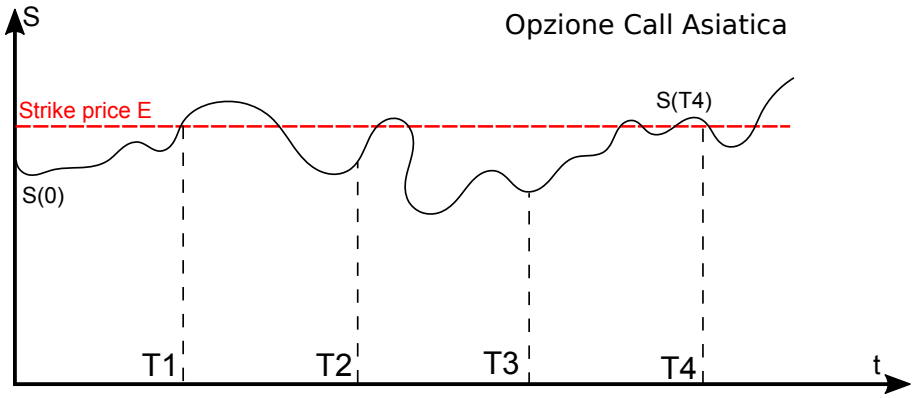
Notiamo innanzitutto che la differenza tra metodo esplicito e metodo implicito è minima pressoché per tutti i valori di S considerati.
Sempre a riguardo di questi due metodi è presente un ottimo accordo (fino alla quinta cifra significativa) con la soluzione analitica, fintanto che il valore iniziale del sottostante è basso. Come anticipato infatti la finitezza di S_{max}, contrapposta alla condizione al bordo (4) crea problemi per valori di S che si avvicinano a $S_{max} = 3E = 60$.
Dall'altro lato invece si nota che la precisione dei metodi Monte Carlo, sia con che senza riduzione della varianza, migliora sempre più quando il valore dell'asset è maggiore. Il metodo con le variabili antitetiche (AMC) inizia ad essere in accordo con la soluzione analitica entro una deviazione standard già a partire da $S = 24$, per il Monte Carlo standard invece ciò avviene a partire da $S = 30$.
Quest'ultimo metodo risulta avere inizialmente una varianza paragonabile a quella dell'AMC, mentre per S maggiori il rapporto tra le varianze arriva ad essere quasi di un fattore 10. Il motivo di questa differenza sta nel fatto che uno *spot price* di 12 è ben sotto il prezzo di esecuzione del contratto considerato in questo caso, quindi i prezzi simulati per il sottostante finiranno molte volte al di sotto di esso al momento della scadenza.
In generale si può dire che l'approccio Monte Carlo per il pricing di opzioni di tipo Europeo è buono solamente entro certi limiti. I metodi alle differenze finite infatti sono ottimi modi per approcciare questo tipo di problema e forniscono soluzioni più precise con minor sforzo computazionale. Per questi ultimi però è opportuno utilizzare cautela nell'individuare fino a che punto reggono le approssimazioni sulle condizioni al contorno.
L'uso congiunto di metodi MC insieme a quello delle differenze finite nel regime ad $S(0)$ alti può essere una valida (ed utilizzata) strategia per definire il prezzo di uno strumento derivato, basandosi su quello attuale del sottostante.
Quando però i metodi MC (e ancora di più Quasi-Monte Carlo) si rivelano veramente efficienti è nel pricing di Opzioni Asiatiche. In questo caso infatti la dimensionalità del problema non è 1, ma tipicamente maggiore di 3, e la convergenza $\sim N^{-1/2}$ (per MC) degli integrali risulta molto efficace.

2 Metodi Quasi-Monte Carlo per il *pricing* di Opzioni Asiatiche.

2.1 Richiamo teorico

2.1.1 Opzioni Asiatiche

Come anticipato le Opzioni Asiatiche sono un tipo di opzioni rispetto al quale i metodi Monte Carlo risultano strumenti computazionali competitivi.
Un'opzione Asiatica, è un titolo derivato il cui valore dipende da una media nel tempo dei valori assunti dal titolo sottostante.



Il *payoff* finale infatti, non dipende più solamente da $S(T)$ come nel caso dell'opzione europea, ma da i valori assunti dal sottostante in determinati tempi $(T_1,\ T_2, \dots)$ decisi al momento dell'acquisto dell'opzione.
Il valore dell'opzione V al tempo di scadenza finale T del contratto sarà (per una *Call*)

$$V(S,\ T) = \max\{S_A(T) - E,\ 0\}\,, \quad \text{per } S_A \geq 0\,. \tag{13}$$

In questo caso $S_A(T)$ però non corrisponde a $S(T)$, ma alla media degli $S(T_i)$. Viene fatta quindi un'ulteriore divisione per queste opzioni oltre a quella *Call/Put*, ovvero quella data dalle opzioni *aritmetiche* e *geometriche* con riferimento alla media. Nel caso in cui ci siano $T_1, \dots, T_M = T$ tempi di controllo (oltre a T_0 tempo iniziale), allora per le opzioni aritmetiche

$$S_A(T) = \frac{1}{M+1} \sum_{i=0}^{M} S(T_i),$$

mentre per quelle geometriche

$$S_A(T) = \exp\left(\frac{1}{M+1} \sum_{i=0}^{M} \log S(T_i) \right).$$

Per il prezzo $V(S,t)$ di entrambi i tipi di opzione non esiste una soluzione analitica, ma soltanto stime numeriche.

In maniera simile a quanto fatto nel caso delle opzioni Europee è possibile simulare percorsi di prezzo per implementare metodi Monte Carlo e dare un fair value a V utilizzando il *payoff* aspettato, scontato al tempo attuale, grazie alla *risk-neutral valuation*:

$$V(S,t) = e^{-r(T-t)}\hat{E}[\max\{S_A - E, 0\}].$$

Per le opzioni Asiatiche serve chiaramente l'accortezza di simulare il prezzo fino ad un tempo di controllo al successivo, ovvero

$$S(T_{i+1}) = S(T_i)\exp\left((r - \frac{1}{2}\sigma^2)(T_{i+1} - T_i) + \sigma\sqrt{T}Z_{i+1}\right),$$

con variabili gaussiane standardizzate $Z_i \sim N(0,1)$.
Risulta evidente che per questo tipo di opzioni, a parità del numero di step Monte Carlo, il costo computazionale è maggiore rispetto a quelle Europee, in quanto è necessaria la generazione di $N \times M$ punti casuali.

2.1.2 *Quasi-Monte Carlo* - La sequenza di Halton

Con l'approccio Monte Carlo standard, data una sequenza di d variabili $U_1, \ldots, U_d$ indipendenti e uniformemente distribuite e una loro funzione f, la stima di

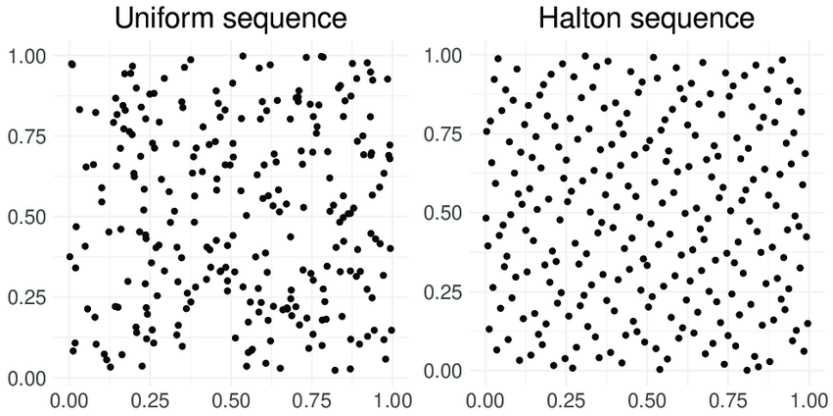
$$\hat{E}[f(U_1, \ldots, U_d] = \int_{[0,1]^d} f(x)dx$$

è data da

$$\int_{[0,1]^d} f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{14}$$

per punti $x_1, \ldots, x_n$ i.i.d. casuali nell'ipercubo $[0,1]^d$.
I metodi Quasi-Monte Carlo invece prevedono l'utilizzo di punti $x_1, \ldots, x_n$ deterministicamente e opportunamente scelti nell'ipercubo $[0,1]^d$.
Nel caso delle opzioni Asiatiche chiaramente la dimensionalità d del problema coincide con la variabile precedentemente chiamata M, ovvero il numero di punti in cui è suddiviso l'intervallo $[0,\ T]$.
La dipendenza dei metodi QMC dalla dimensionalità del problema è una della caratteristiche che più li differenziano dal MC standard. In quest'ultimo infatti l'ordine in cui sono campionati gli $N \times d$ punti U_i non importa, per $N \to \infty$ infatti è indifferente come vengano raggruppati in vettori $(U_1, \ldots, U_d)$, $(U_{d+1}, \ldots, U_{2d}), \ldots$. In QMC al contrario la costruzione dei punti x_i dipende esplicitamente dalla dimensione del problema d e i vettori non possono più essere costruiti prendendo acriticamente dei set di d elementi consecutivi di una sequenza scalare. Per questo motivo per il QMC, a differenza del Monte Carlo standard dove raramente ci si interessa della dimensionalità del problema, è necessario conoscere a priori la dimensione del reticolo di punti da generare.
Le sequenze cosiddette a *bassa discrepanza* sono quelle che forniscono valori per gli x_i tali che l'errore nella stima dell'integrale (14) sia il più piccolo possibile per una gran-

de classe di integrandi f. Si dimostra, ed è intuitivamente chiaro, che queste sequenze sono quelle forniscono punti x_i che riempiano uniformemente l'ipercubo (nella figura seguente se ne riporta un esempio per $d = 2$).



Per arrivare alla sequenza a bassa discrepanza utilizzata nel codice, ovvero quella di *Halton*, è necessario prima introdurre quella di *Van der Corput*. Quest'ultima è una sequenza unidimensionale definita per una base intera $b \geq 2$.
Un qualsiasi intero k può essere riscritto come un'unica combinazione lineare di potenze di b, con coefficienti $a_j \in \{0, 1, \ldots, b-1\}$:

$$k = \sum_{j=0}^{\infty} a_j(k) b^j$$

La funzione inversa radicale ψ_b mappa ogni k in un punto in $[0, 1]$ riscrivendo k come $0.a_0 a_1 a_2 \ldots$ e trasformando successivamente questo numero dalla base b-aria a quella decimale. Formalmente

$$\psi_b(k) = \sum_{j=0}^{\infty} \frac{a_j(k)}{b^{j+1}}$$

Definiamo **sequenza in base b di Van der Corput** la sequenza di valori

$$0 = \psi_b(0), \psi_b(1), \psi_b(2), \ldots$$

La proprietà peculiare di questa sequenza è che ogni punto successivo è il più distante possibile dai precedenti nella sequenza, nella tabella seguente ne è riportato un esempio.

Tabella 2: Sono riportati i primi 8 valori della sequenza di VdC con base binaria.

k	**k Binario**	ψ_2**(k) Binario**	ψ_2**(k)**
0	0	0	0
1	1	0.1	1/2
2	10	0.01	1/4
3	11	0.11	3/4
4	100	0.001	1/8
5	101	0.101	5/8
6	110	0.011	3/8
7	111	0.111	7/8

Una volta definita la sequenza di Van der Corput, introdurre quella di Halton e Hammersley risulta come una semplice generalizzazione.
Data un'arbitraria dimensione d infatti, le coordinate della sequenza di Halton d-dimensionale sono semplicemente sequenze di Van der Corput in basi differenti. Una condizione ncessaria per riempire l'ipercubo inoltre è che le d basi $b_1, \dots, b_d$ siano tutte composte da numeri coprimi tra loro e maggiori di 1. In altre parole

$$x_k = (\psi_{b_1}(k), \psi_{b_2}(k), \dots, \psi_{b_d}(k)), \qquad k = 0, 1, \dots$$

In generale per una maggiore uniformità e comodità si prediligono le basi più piccole, pertanto nel codice come basi sono stati utilizzati i primi d numeri primi.
Il problema principale dei metodi Quasi-Monte Carlo è la mancanza di una precisa stima dell'errore, l'*upper bound* classicamente usato per l'integrazione QMC è la disuguaglianza di Koksma-Hlawka

$$\left| \int_{[0,1]^d} f(x)dx - \frac{1}{N}\sum_{i=1}^{N} f(x_i) \right| \leq V(f)D^*(x_1, \dots, x_N)$$

dove $V(f)$ è la *variazione* di $f(x)$ nell'ipercubo nel senso di Hardy-Krause, mentre $D^*(x_1, \dots, x_N)$ è la cosiddetta *star discrepancy*, ovvero la discrepanza massima tra i punti $x_1, \dots, x_N$. Si dimostra che l'andamento asintotico dell'upper bound sull'errore risulta dell'ordine $\sim O((\log N)^d/N)$, il che ci porta a considerare di utilizzare i metodi QMC quando la dimensionalità non supera valori dell'ordine di 50 (anche se varia da problema a problema).
La disuguaglianza precedente è un risultato teorico che risulta molto difficilmente implementabile in una stima immediata dell'errore, per questo motivo si introducono metodi **Quasi-Monte Carlo randomizzati**.
L'intuizione dietro questi metodi è quella di introdurre una componente statistica all'interno delle sequenze a bassa discrepanza, in modo poi da utilizzare un errore che permetta di assegnare un intervallo di confidenza ai valori trovati.
Si fa quindi un *tradeoff* tra le caratteristiche migliori del MC standard e del QMC: viene sacrificata un po' di precisione per ottenere una miglior misura dell'errore.
Il metodo di randomizzazione implementato nel codice è quello di *Random Shift*. Es-

so comporta la generazione di un vettore U_s distribuito uniformemente nell'ipercubo unitario d-dimensionale e nello shift di ogni x_i di (U_s modulo 1).
In altre parole la stima dell'integrale risulta

$$I_s = \frac{1}{N}\sum_{i=1}^{N} f(x_i + U_s mod 1)$$

Ripetere la randomizzazione con repliche indipendenti di U_s produce dei set indipendenti, ognuno con N punti. Nel codice la variabile N_{rand} rappresenta il numero di questi set: per ognuno degli N_{rand} set generati, si calcolano gli I_s che saranno indipendenti e identicamente distribuiti. Inoltre ogni I_s è una stima senza bias dell'integrale, quindi è possibile applicare il teorema del limite centrale per la stima dell'errore ed assegnare un intervallo di confidenza.

2.2 Descrizione del codice

Il *main* si divide in due cicli principali, il primo dedicato alla stima Monte Carlo dell'opzione, mentre il secondo per il Quasi-Monte Carlo randomizzato. Prima dei cicli viene richiamata la funzione *first_d_primes* la quale riempie l'array, variabile globale, *primes[d]* con i primi d numeri primi, per successivo utilizzo.

Il primo ciclo, ripetuto per un numero di N passi, non fa altro che richiamare la funzione *price_paths_MC* che genera gli andamenti di prezzo $\bar{S}$ ed accumulare le grandezze C_{MC} e $C2_{MC}$, ovvero la stima

$$e^{-r(T-t)}\max\{\bar{S} - E,\, 0\}$$

ed il suo quadrato rispettivamente.
La funzione *price_paths_MC* differisce chiaramente da quella implementata con il codice della sezione precedente in quanto è necessario fissare il valore di S ad ogni tempo di controllo, ovvero ogni $\mathrm{d}t = T/d$. A tal proposito si nota, en passant, che nonostante la maggior parte delle opzioni Asiatiche abbiano tempi di controllo equidistanti, è possibile che esistano opzioni esotiche con $\mathrm{d}t$ differenti per ogni intervallo.
Di nuovo il calcolo delle N variabili gaussiane standardizzate viene fatto a partire da altrettante variabili stocastiche uniformemente distribuite in $[0,\ 1]$ attraverso le trasformazioni di Box-Muller.

Il secondo ciclo principale viene effettuato fino ad N_{rand} ed implementa la randomizzazione per arrivare ad una corretta stima dell'errore. Inizialmente infatti viene creato il vettore d-dimensionale $U_s = shift[d]$ che serve nel ciclo successivo fatto sugli N step della simulazione. Questo secondo ciclo funziona analogamente al primo, con la differenza che i cammini di prezzo sono generati tramite la funzione *price_paths_QMC*.
Quest'ultima risulta più complicata della precedente. Inizialmente grazie alla funzione *Van_dC* con d basi diverse viene generato un array d-dimensionale. Le basi sono date dall'array *primes[d]* precedentemente calcolato, portando così alla creazione di una sequenza di Halton d-dimensionale.

Successivamente il vettore di shift viene sommato componente per componente ai vettori x_i della sequenza e se ne viene fatto il modulo in base 1.
A questo punto l'accumulo delle grandezze avviene in maniera analoga al caso precedente, tenendo presente che per quanto riguarda la deviazione standard del QMC è necessario calcolare la varianza sul valore medio del ciclo su N_{rand}.

Nel codice sono commentate alcune linee che modificano il calcolo dell'opzione da aritmetica a geometrica.

I risultati ottenuti vengono salvati in un file di testo per la successiva elaborazione in ambiente MATLAB (Appendice B.3). Nel codice il linguaggio C riportato in Appendice B.2 sono stati presenti anche i due cicli che variano il numero di step Monte Carlo N in modo da disegnare alcuni dei grafici nella sezione seguente.

2.3 Discussione dei risultati e confronto

Per il confronto tra il metodo Monte Carlo e QMC sono stati utilizzati parametri iniziali differenti dal caso precedente. Si è supposto il caso reale di un'opzione Asiatica con valore del sottostante iniziale (*spot price*) pari a $S(0) = 50€$. La scadenza del contratto è fissata a $T = 1$ anno, con tempi di controllo T_i ogni mese, quindi dimensionalità $d = 12$. Il prezzo di esercizio, come spesso accade in realtà, viene fissato uguale a quello attuale del sottostante $E = 50€$. Per quanto riguarda l'interesse composto annuo prendiamo l'1%, ovvero $r = 0.01$, e come volatilità il 40%, cioè $\sigma = 0.4$.
Nel grafico seguente (Fig.1) è riportato il prezzo dell'opzione all'istante iniziale in funzione del numero di step utilizzati, fino ad un massimo di 50000 cammini di prezzo simulati. In questo grafico la simulazione è stata fatta ripartire, con diverso *seed*, per ogni valore assegnato di N: ciò risulta utile a capire quanto possa cambiare il prezzo cambiando N per un diverso set di variabili casuali.

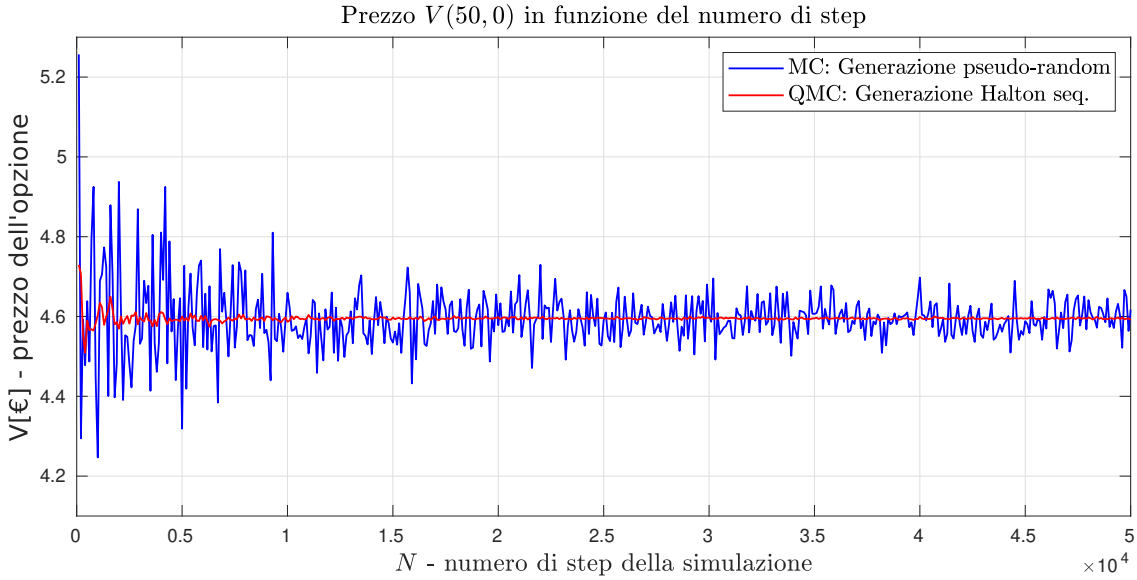


Figura 1: Grafico per punti, ogni *N_print* $= 100$ step nell'intervallo $[100, 5000]$ è calcolato il prezzo di un'opzione asiatica con i parametri $T = 1$ $d = 12$, $S(0) = 50$, $E = 50$, $r = 0.01$, $\sigma = 0.4$.

Dal grafico risulta evidente la ben più rapida convergenza della sequenza di Halton rispetto a quella pseudo-random del Monte Carlo standard. Il valore di $V(50,0)$ finale trovato con quest'ultimo è (4.61 ± 0.04)€, mentre quello risultante dal metodo Quasi-Monte Carlo è (4.595 ± 0.001)€. I due risultati sono in accordo tra loro e si vede la maggior precisione del QMC che arriva a dare una stima del prezzo dell'opzione più precisa di un fattore 10 rispetto a quella ottenuta con le sequenze pseudo-random.
Nel grafico che segue il valore dell'opzione con gli stessi parametri precedenti è stato plottato progressivamente una volta fissato il *seed*, quindi è rappresentativo di come dato un singolo set di variabili casuali avvenga la convergenza al valore finale.

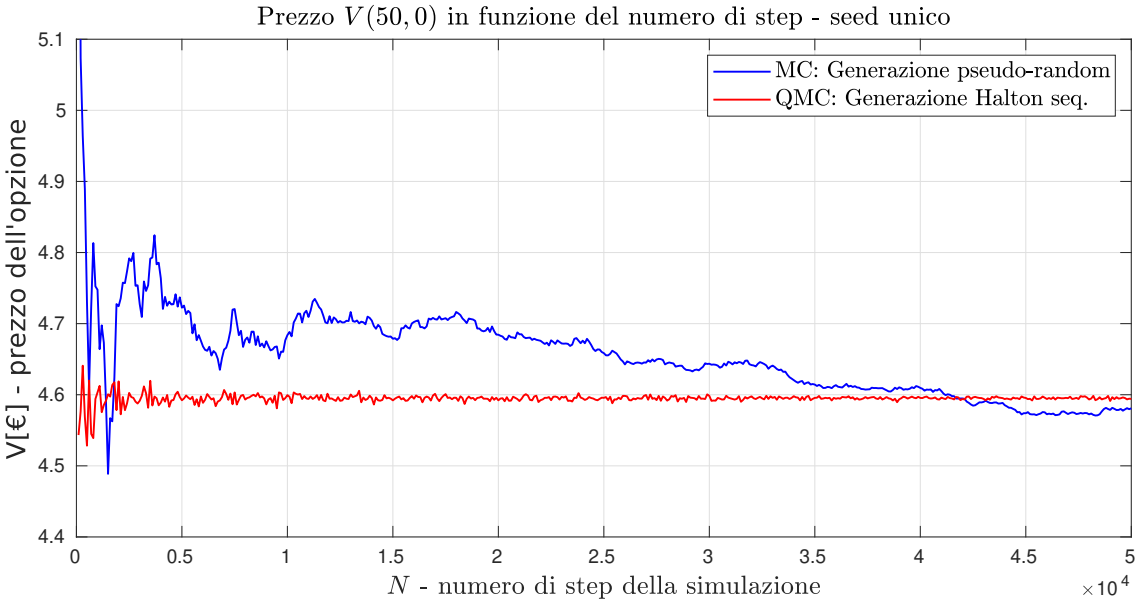


Figura 2: Grafico per punti, ogni $N\_print = 100$ step nell'intervallo $[100, 5000]$ è calcolato il prezzo di un'opzione asiatica con i parametri $T = 1$ $d = 12$, $S(0) = 50$, $E = 50$, $r = 0.01$, $\sigma = 0.4$.

A sostegno di quanto visto in precedenza si nota un rapido assestamento del prezzo per quanto riguarda le sequenze di Halton, mentre un andamento asintotico più lento per quanto riguarda la convergenza del Monte Carlo normale.
Questo fatto si osserva ancora meglio in Fig.3, dove è rappresentata direttamente la deviazione standard sul prezzo dell'opzione asiatica nei due casi.

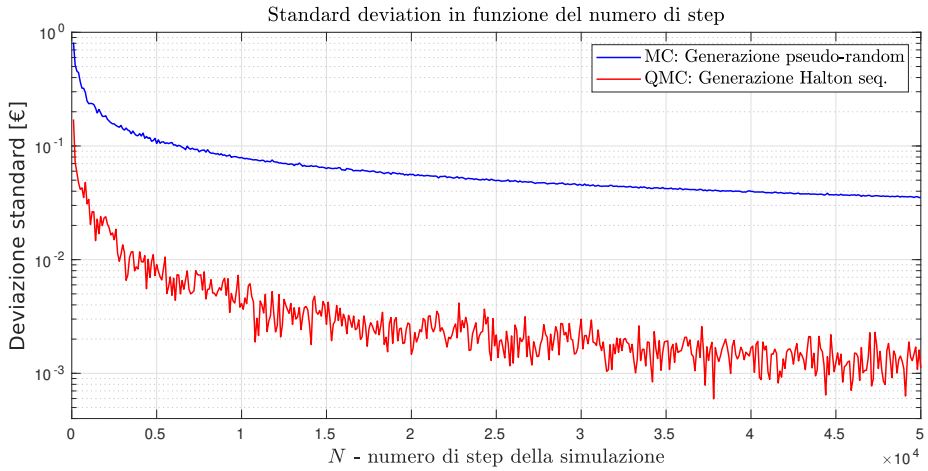


Figura 3: Grafico per punti, ogni $N\_print = 100$ step nell'intervallo $[100, 5000]$ è calcolata la deviazione standard complessiva un'opzione asiatica con i parametri $T = 1$ $d = 12$, $S(0) = 50$, $E = 50$, $r = 0.01$, $\sigma = 0.4$.

Il QMC risulta con deviazione standard molto più piccola (di un fattore 6 circa) rispetto a quella del MC già a partire da bassi valori di N. Questa discrepanza cresce progressivamente sempre più all'aumentare del numero di step di simulazione. D'altronde è cio che ci si aspetta ricordando che l'andamento dell'errore sul QMC è dell'ordine $\sim O((\log N)^d/N)$, in contrasto con quello del MC $\sim O(N^{-1/2})$.
Un'altra osservazione che si può fare osservando il grafico precedente è che sebbene il valore medio della deviazione standard trovato per le sequenze di Halton sia minore di quello del MC, la varianza su di essa è nettamente maggiore per quanto riguarda il metodo Quasi-Monte Carlo.
La causa di ciò risiede nella definizione dell'errore attraverso la randomizzazione delle sequenze di Halton. La varianza sullo scarto quadratico medio infatti dipende fortemente dal valore scelto per N_{rand}, come testimonia la Fig.4, dove la randomizzazione è fatta solamente su 2 campioni.

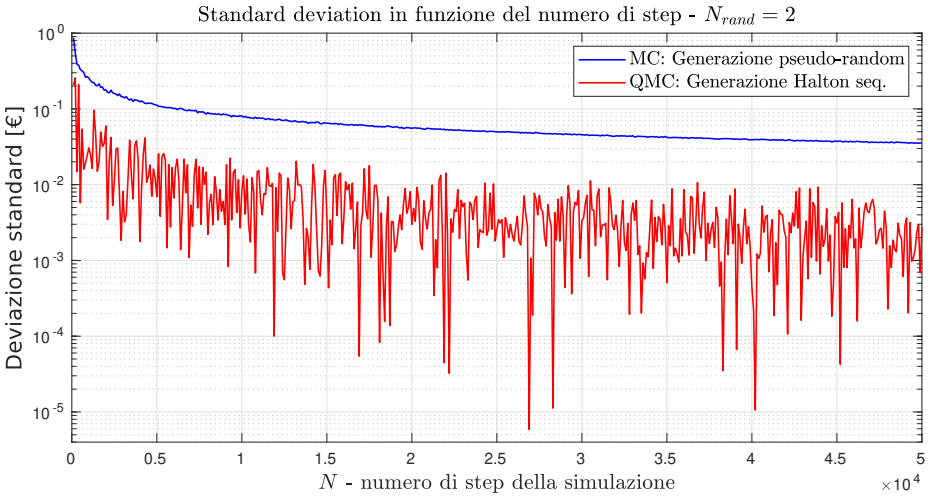


Figura 4: Grafico per punti con $N_{rand} = 2$, ogni $N\_print = 100$ step nell'intervallo $[100, 5000]$ è calcolata la deviazione standard complessiva un'opzione asiatica con i parametri $T = 1$ $d = 12$, $S(0) = 50$, $E = 50$, $r = 0.01$, $\sigma = 0.4$.

Qui infatti la varianza della deviazione standard è molto più grande del caso precedente e N_{rand} è troppo piccolo per darne una buona stima.

Per concludere si vede come la dimensionalità del problema influisca sull'efficacia dei metodi QMC.
Per fare ciò ci avvaliamo della coppia di grafici che seguono (Fig.5 e 6). L'unica differenza nei parametri iniziali rispetto al caso precedente è il fatto che i tempi di controllo per il calcolo della media finale sono settati una volta ogni trimestre, quindi la dimensionalità risulta $d = 4$.

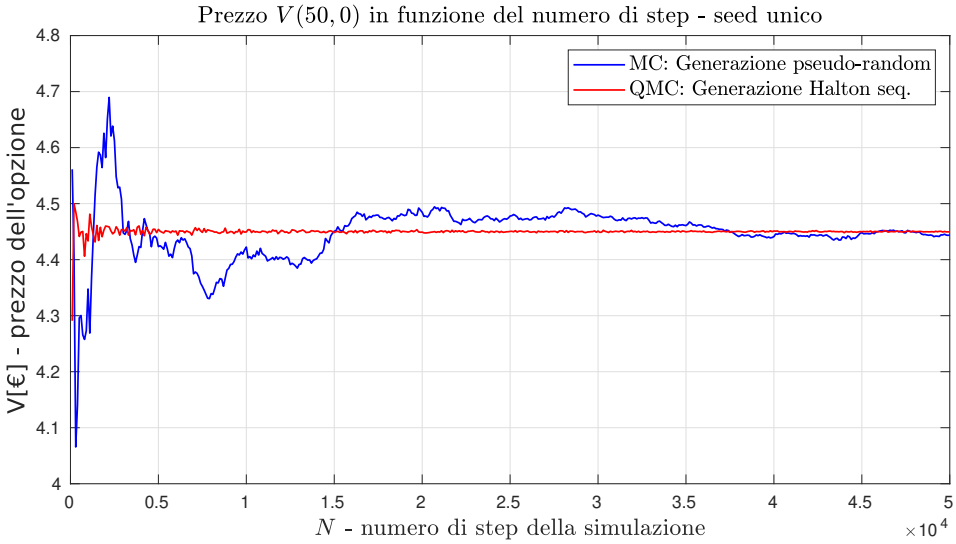


Figura 5: Grafico per punti, ogni $N\_print = 100$ step nell'intervallo $[100, 5000]$ è calcolato il prezzo di un'opzione asiatica con i parametri $T = 1$ $d = 4$, $S(0) = 50$, $E = 50$, $r = 0.01$, $\sigma = 0.4$.

Nonostante l'andamento del prezzo possa sembrare simile a quello per $d = 12$, oltre ad uno shift del valor medio di circa 15 centesimi, il secondo grafico rivela chiaramente che per il Quasi-Monte Carlo la deviazione standard è diminuita. Ciò può essere testimoniato dal valore di scarto quadratico medio finale, che nel caso $d = 12$ valeva 0.001€, mentre per $d = 4$ è 0.0003€.

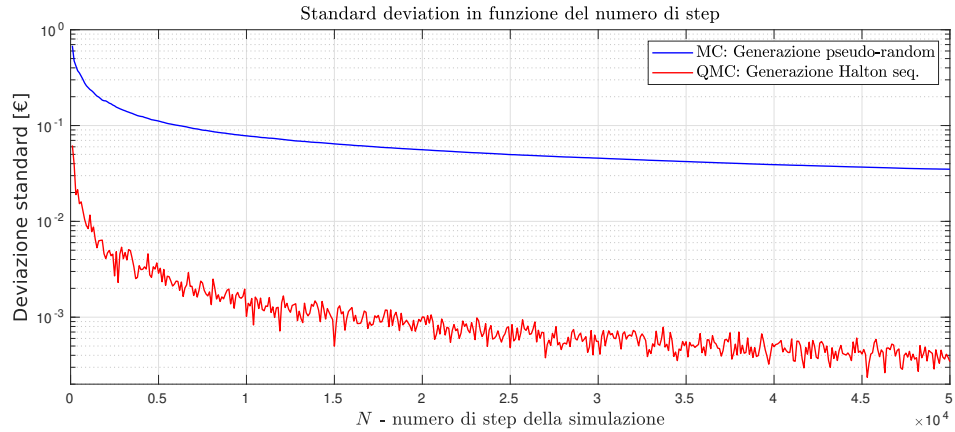


Figura 6: Grafico per punti con, ogni $N\_print = 100$ step nell'intervallo $[100, 5000]$ è calcolata la deviazione standard complessiva un'opzione asiatica con i parametri $T = 1$ $d = 4$, $S(0) = 50$, $E = 50$, $r = 0.01$, $\sigma = 0.4$.

Per quanto riguarda la deviazione standard del metodo Monte Carlo invece non ci sono variazioni eclatanti: in entrambi i casi è circa 0.04€. Di nuovo, è proprio quello che ci aspettiamo, come già illustrato nella sezione di teoria la dimensionalità dell'ipercubo considerato non influenza la generazione mediante sequenze pseudo-random di punti per l'algoritmo Monte Carlo. Al contrario essa è fondamentale per l'implementazione di approcci con sequenze a bassa discrepanza, tanto che l'andamento asintotico dell'errore ne ha una dipendenza esplicita, ovvero $O((\log N)^d/N)$.

A Risultati teorici

A.1 L'equazione di Black-Scholes

Secondo l'assunzione che il prezzo del sottostante segua un moto geometrico browniano, possiamo descrivere la variazione di S in un intervallo infinitesimo $\mathrm{d}t$ come

$$\mathrm{d}S = \mu S \mathrm{d}t + \sigma S \mathrm{d}z\,. \tag{15}$$

Qui il parametro assunto noto μ è il *rate* di ritorno atteso per l'*asset* e identifica un *trend* del prezzo, σ invece è un parametro che rappresenta la volatilità del prezzo, ovvero la deviazione standard intorno al prezzo medio dato dal termine di *trend*, $\mathrm{d}z$ infine è un incremento infinitesimo di una variabile stocastica $z(t)$, ben modellizzata da una *random walk*. In definitiva, l'equazione (15) ci dice che, localmente, il tasso di ritorno dell'*asset* ha un valore d'aspettazione $\mu \mathrm{d}t$ ed una varianza $\sigma^2 \mathrm{d}t$.
A questo punto, sapendo che il derivato è una funzione deterministica di S e di t, possiamo utilizzare il lemma di Itô (un'applicazione dello sviluppo in serie di Taylor per funzioni di più variabili) per scrivere il differenziale di una funzione dipendente dal tempo per un processo stocastico:

$$\mathrm{d}V = \left(\frac{\partial V}{\partial S}\mu S + \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right)\mathrm{d}t + \frac{\partial V}{\partial S}\sigma S \mathrm{d}z\,. \tag{16}$$

Con Π definiamo ora il nostro portafoglio di investimento misto, strutturato così:

$$\Pi = \alpha(t)R + \beta(t)S + V\,, \tag{17}$$

dove $\alpha(t)$ e $\beta(t)$ rappresentano, rispettivamente, le quantità di *asset* senza rischio a prezzo R e di sottostante S. Notiamo, en passant, che, per come è definito l'*asset* senza rischio, vale $\mathrm{d}R = rR\mathrm{d}t$.
Calcoliamo la variazione del portafoglio $\mathrm{d}\Pi$ dopo un tempo $\mathrm{d}t$:

$$\begin{aligned}
\mathrm{d}\Pi &= \alpha \mathrm{d}R + \beta \mathrm{d}S + \mathrm{d}V \\
&= \alpha r R \mathrm{d}t + \beta(\mu S \mathrm{d}t + \sigma S \mathrm{d}z) + \left(\frac{\partial V}{\partial S}\mu S + \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right)\mathrm{d}t + \frac{\partial V}{\partial S}\sigma S \mathrm{d}z \\
&= \left(\alpha r R + \beta \mu S + \frac{\partial V}{\partial S}\mu S + \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right)\mathrm{d}t + \left(\beta \sigma S + \frac{\partial V}{\partial S}\sigma S\right)\mathrm{d}z
\end{aligned}$$

Dall'ultima espressione risulta evidente come si possano eliminare gli effetti stocastici della volatilità, presenti nel differenziale $\mathrm{d}z$, scegliendo $\beta = -\frac{\partial V}{\partial S}$. Fatta questa sostituzione, il differenziale del portafoglio diventa

$$\mathrm{d}\Pi = \mathrm{d}(\alpha R + \beta S + V) = \left(\alpha r R + \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right)\mathrm{d}t\,.$$

A questo punto, bisogna imporre la condizione di *fair value* che non permetta l'arbitraggio. Quindi imponiamo che il ritorno massimo che si possa avere investendo con il

portafoglio misto (17) sia pari a quello che si avrebbe se si investisse in un portafoglio formato solamente da *asset* senza rischio (quindi con $\Pi = R$). Otteniamo la condizione $\mathrm{d}\Pi = r\Pi \mathrm{d}t$, quindi

$$r\Pi \mathrm{d}t = r(\alpha R + \beta S + V)\mathrm{d}t = \left(\alpha r R + \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right)\mathrm{d}t\,,$$

che, elidendo i termini dipendenti da α, porta all'**equazione di Black-Scholes**:

$$\boxed{\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0} \tag{18}$$

La condizione finale data dall'opzione di esecuzione o meno del contratto al *payoff time* T è

$$V(S, T) = \max\{S - E, 0\}\,, \quad \text{per } S \geq 0\,. \tag{19}$$

Ora, essendo presente una derivata parziale al secondo ordine rispetto ad S, sono necessarie due condizioni al contorno per questa variabile. Scegliamo le seguenti:

$$\begin{cases} V(0, t) = 0 & \text{per ogni } t \\ V(S, t) \to S & \text{per } S \to \infty \end{cases} \tag{20}$$

dove la prima rispecchia il fatto che il prezzo dello strumento derivato sia nullo quando lo è quello del sottostante; inoltre è chiaro come per la (19) il contratto non abbia alcun valore se al tempo finale vale $S = 0$. La seconda condizione, invece, deriva dal fatto che per $S \to \infty$ è sempre più probabile che $S(T)$ sia maggiore del prezzo di esercizio E e quindi che il valore del contratto, in questo limite, sia $V \sim S - E \sim S$. È opportuno notare come la seconda condizione, seppur scelta con cognizione di causa rispetto alla realtà, non sempre rispecchi la natura degli strumenti derivati.

A.2 Risoluzione analitica dell'equazione di Black-Scholes

Per un'Opzione Europea di tipo *Call* si può mostrare come l'equazione (18) risulti sostanzialmente un'equazione di diffusione.
Volendo trasformare l'equazione parabolica di tipo *backward* in una di tipo *forward*, più comoda nella risoluzione, procediamo effettuando le sostituzioni

$$\tau \equiv \frac{\sigma^2}{2}(T - t)\,, \qquad x \equiv \ln\left(\frac{S}{E}\right), \qquad v \equiv \frac{V}{E}\,,$$

con le quali otteniamo per le derivate

$$\frac{\partial V}{\partial t} = -\frac{E\sigma^2}{2}\frac{\partial v}{\partial \tau}\,,$$

$$\frac{\partial V}{\partial S} = \frac{E}{S}\frac{\partial v}{\partial x},$$
$$\frac{\partial^2 V}{\partial S^2} = \frac{\partial}{\partial S}\left(\frac{\partial V}{\partial S}\right) = -\frac{E}{S^2}\frac{\partial v}{\partial x} + \frac{E}{S^2}\frac{\partial^2 v}{\partial x^2},$$

e per la condizione finale (19)

$$V(S, T) = \max\{S - E, 0\} = \max\{Ee^x - E, 0\} \implies v(x, 0) = \max\{e^x - 1, 0\}.$$

Sostituendo le derivate all'interno dell'equazione BS si ottiene

$$-\frac{E\sigma^2}{2}\frac{\partial v}{\partial \tau} + \frac{1}{2}\sigma^2 S\left(-\frac{E}{S^2}\frac{\partial v}{\partial x} + \frac{E}{S^2}\frac{\partial^2 v}{\partial x^2}\right) + rs\left(\frac{E}{S}\frac{\partial v}{\partial x}\right) - rEv = 0,$$

che, semplificando e sostituendo $a \equiv (2r/\sigma^2) - 1$ e $b \equiv -(a+1)$, diventa

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + a\frac{\partial v}{\partial x} + bv, \tag{21}$$

ovvero un'equazione di diffusione. Notiamo che, grazie al cambiamento di variabile, ora la variabile x può assumere tutti i valori della retta reale e l'equazione è a coefficienti costanti, determinati dal coefficiente di volatilità e dal tasso di interesse. Possiamo semplificare ulteriormente l'equazione introducendo un altro cambio di variabile:

$$u(x, \tau) \equiv \frac{v(x, \tau)}{e^{\lambda x + \nu\tau}},$$

con λ e ν coefficienti da determinare. Le derivate temporali e spaziali di v ora risultano

$$\frac{\partial v}{\partial t} = \nu e^{\lambda x + \nu\tau} u + e^{\lambda x + \nu\tau}\frac{\partial u}{\partial \tau},$$
$$\frac{\partial v}{\partial x} = \lambda e^{\lambda x + \nu\tau} u + e^{\lambda x + \nu\tau}\frac{\partial u}{\partial x},$$
$$\frac{\partial^2 v}{\partial x^2} = \lambda^2 e^{\lambda x + \nu\tau} u + 2\lambda e^{\lambda x + \nu\tau}\frac{\partial u}{\partial x} + e^{\lambda x + \nu\tau}\frac{\partial^2 u}{\partial x^2}.$$

A questo punto, possiamo scegliere opportunamente i coefficienti. Prendendo $\lambda = -a/2$ e $\nu = \lambda^2 + a\lambda + b$, l'equazione (21) si riduce semplicemente a

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}, \tag{22}$$

ovvero un'equazione del calore con condizione iniziale

$$u(x, 0) \equiv u_0(x) = e^{\lambda x + \nu 0} v(x, 0) = e^{\lambda x}\max\{e^x - 1, 0\} = \max\left\{2e^{x(a+1)/2}\sinh\left(\frac{x}{2}\right), 0\right\}.$$

La soluzione generale per l'equazione del calore si ricava semplicemente sfruttando le trasformate di Fourier e risulta

$$u(x,\,\tau) = \frac{1}{2\sqrt{\pi\tau}} \int_{-\infty}^{+\infty} u_0(s) e^{-\frac{(x-s)^2}{4\tau}} \mathrm{d}s\,.$$

Riscriviamo l'integrale con il cambiamento di variabile $z \equiv (s-x)/\sqrt{2\tau}$:

$$u(x,\,\tau) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} u_0\left(z\sqrt{2\tau} + x\right) e^{-\frac{z^2}{2}} \mathrm{d}z\,.$$

Poiché la funzione u_0 si annulla per $x \leq 0$, possiamo utilizzare come estremo inferiore dell'integrale $z = -x/\sqrt{2\tau}$. La funzione precedente allora diventa la somma di due integrali del tipo

$$\frac{1}{\sqrt{2\pi}} \int_{-x/\sqrt{2\tau}}^{+\infty} e^{\frac{q_{1,2}}{2}(x+z\sqrt{2\tau})} e^{-\frac{z^2}{2}} \mathrm{d}z\,,$$

dove $q_1 \equiv a+2$ e $q_2 \equiv a+1$. Questo tipo di integrale si trova tabulato e si riscrive come

$$\frac{e^{\frac{qx}{2}+\frac{\tau q^2}{4}}}{\sqrt{2\pi}} \int_{-x/\sqrt{2\tau}-q\sqrt{\tau/2}}^{+\infty} e^{-\frac{z^2}{2}} \mathrm{d}z\,.$$

Quindi l'integrale rimasto risulta essere la funzione di ripartizione di una distribuzione gaussiana; allora, definendo

$$\Phi(d) \equiv \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{d} e^{-\frac{z^2}{2}} \mathrm{d}z\,,$$

la soluzione per l'equazione del calore (22) risulta

$$u(x,\,\tau) = e^{\frac{(a+2)x}{2}+\frac{\tau(a+2)^2}{4}} \Phi(d_1) - e^{\frac{(a+1)x}{2}+\frac{\tau(a+1)^2}{4}} \Phi(d_2)\,,$$

con $d_{1,2} \equiv \frac{x}{\sqrt{2\tau}} + q_{1,2}\sqrt{\tau/2}$. Per concludere, torniamo alle variabili di partenza: ponendo

$$d_+ \equiv \frac{\log\left(\frac{S}{E}\right) + (T-t)(r+\sigma^2/2)}{\sigma\sqrt{T-t}}\,, \tag{23}$$

$$d_- \equiv \frac{\log\left(\frac{S}{E}\right) + (T-t)(r-\sigma^2/2)}{\sigma\sqrt{T-t}}\,, \tag{24}$$

la soluzione dell'equazione BS per la dinamica di un'Opzione Europea di tipo *Call* risulta

$$\boxed{V(S,\,t) = S\Phi(d_+) - Ee^{-r(T-t)}\Phi(d_-)} \tag{25}$$

B Codice

Di seguito sono riportati codice e script utilizzati rispettivamente per la parte di risoluzione numerica e per la parte grafica.

B.1 Codice in linguaggio C - Opzione Europea

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define MAX(a, b) ((a) > (b) ? (a) : (b))

// Costanti globali
#define T 0.25 // Payoff time
#define E 20.0 // Strike price
#define r 0.1 // Tasso di interesse
#define sigma 0.4 // Volatilita' del prezzo
#define S_max 3*E // Prezzo massimo per metodi FD

// Salvataggio e stampa
#define NUM_PRINT_n 9 // Numero di valori di S per cui e' stampato a video V (a tempo fissato)
#define INT_PRINT_m 5000 // Numero di passi tra una stampa a video e l'altra
#define NUM_SAVE_S 200 // (Numero-1) di valori di S per cui viene salvato V su file
#define NUM_SAVE_t 200 // (Numero-1) di valori di t per cui viene salvato V su file

const int N = 1000; // Numero di intervalli in cui e' diviso [0, S_max]
const int M = 50000; // Numero di intervalli in cui e' diviso [0, T]
const int N_MC = 1000000; // Numero step simulazione MC

// File per salvataggio dati
const char nome_file[] = "dati_esame_"; // Prima parte nome file
const char estens[] = ".txt"; // Estensione file

// Prototipi funzioni
void passo_esplicito(); // Esegue un passo del metodo esplicito
```

```
31 void passo_implicito(); // Esegue un passo del metodo
      implicito
32 void price_paths(); // Calcola un andamento di prezzo
      casuale per MC
33 void anti_price_paths(); // Calcola un andamento di prezzo
       casuale per anti MC
34 // Variabili globali FD
35 double DeltaS; // Ampiezza intervalli in cui e' diviso [0,
       S_max]
36 double DeltaT; // Ampiezza intervalli in cui e' diviso [0,
       T]
37 double alpha; // sigma^2 * DeltaT
38 double beta; // r * DeltaT
39 double *V; // Puntatore a vettore che conterra' i valori V
      (n * DeltaS, m * DeltaT) (con m \in {0, 1, ..., M}
      fissato) per n = 0, 1, ..., N
40 double t; // Tempo (trasformato secondo t' = T - t)
41
42 // Variabili globali MC
43 double S[2], S_anti[2]; // Price paths casuali e per
      antithetic MC
44 double Z[2]; // Numeri casuali distribuiti normalmente
45 double S_now[NUM_PRINT_n] = {6, 12, 18, 24, 30, 36, 42,
      48, 54}; // Valori iniziali di S
46 int mc_index; // Indice che corre sui valori di S_now
47
48
49 int main(){
50     FILE *file;
51
52     srand(time(0)); // Aggiorno seed
53
54     short i, k;
55     int j, m;
56     int n_print[NUM_PRINT_n], n_save[NUM_SAVE_S+1],
          int_save_m;
57     char str[56];
58
59     printf("RISOLUZIONE DELL'EQUAZIONE DI BLACK-SCHOLES
          CON FD METHODS\n\n");
60
61
62     for(k = 0; k < 2; k++){ // k = 0: metodo esplicito, k
          = 1: metodo implicito
63         /* File per salvataggio dati
```

```
64              '1': metodo esplicito, '2': metodo implicito;
                  la lettera identifica la coppia (N, M) */
65          sprintf(str, "%s%d%c%s", nome_file, k+1, 'f',
               estens);
66          file = fopen(str, "w");
67
68          // Calcolo parametri
69          DeltaS = S_max / N;
70          DeltaT = T / M;
71          alpha = sigma * sigma * DeltaT;
72          beta = r * DeltaT;
73
74          // Istanzio vettore V
75          if(k == 0){
76              V = malloc(sizeof(double) * (N + 1));
77          }
78
79          // Condizioni al bordo e condizioni iniziali
80          V[0] = 0;
81          V[N] = S_max - E;
82          for(j = 1; j < N; j++)
83              V[j] = MAX(j * DeltaS - E, 0);
84          t = 0;
85
86          // Stampo a video info e condizioni iniziali
87          if(k == 0){
88              printf("METODO ESPLICITO\n");
89              for(j = 0; j < NUM_PRINT_n; j++)
90                  n_print[j] = N / (NUM_PRINT_n + 1) * (j +
                       1);
91
92              int_save_m = M / NUM_SAVE_t;
93              for(j = 0; j <= NUM_SAVE_S; j++)
94                  n_save[j] = N / NUM_SAVE_S * j;
95          }
96          else{
97              printf("METODO IMPLICITO\n");
98          }
99          printf("N = %d, M = %d\n\n", N, M);
100         printf("Passo\tt");
101         for(j = 0; j < NUM_PRINT_n; j++)
102             printf("\t\tV(%1.2f,t)", n_print[j] * DeltaS);
103         printf("\n0\t%.4f", t);
104         for(j = 0; j < NUM_PRINT_n; j++)
105             printf("\t\t%.5f", V[n_print[j]]);
106         printf("\n");
```

```
107
108         // Salvo su file condizioni iniziali
109         fprintf(file, "0");
110         for(j = 0; j <= NUM_SAVE_S; j++)
111             fprintf(file, " %f", n_save[j] * DeltaS);
112         fprintf(file, "\n");
113         fprintf(file, "%f", t);
114         for(j = 0; j <= NUM_SAVE_S; j++)
115             fprintf(file, " %f", V[n_save[j]]);
116         fprintf(file, "\n");
117
118         // Evoluzione temporale
119         for(m = 1; m <= M; m++){
120             if(k == 0)
121                 passo_esplicito(); // Passo algoritmo
                       esplicito
122             else
123                 passo_implicito(); // Passo algoritmo
                       implicito
124
125             // Stampo a video passo
126             if(m % INT_PRINT_m == 0){
127                 printf("%d\t%.5f", m, t);
128                 for(j = 0; j < NUM_PRINT_n; j++)
129                     printf("\t\t%.5f", V[n_print[j]]);
130                 printf("\n");
131             }
132
133             // Salvo su file passo
134             if(m % int_save_m == 0){
135                 fprintf(file, "%f", t);
136                 for(j = 0; j <= NUM_SAVE_S; j++)
137                     fprintf(file, " %f", V[n_save[j]]);
138                 fprintf(file, "\n");
139             }
140         }
141
142         printf("\n----------------------------\n\n");
143
144         fclose(file);
145     }
146
147
148     printf("RISOLUZIONE DELL'EQUAZIONE DI BLACK-SCHOLES
           CON METODO MC\n\n");
149     double C_MC = 0, C2_MC = 0, C_anti = 0, C2_anti = 0;
```

```
150     double std_MC, std_anti, acc_anti, acc_mc;
151
152     sprintf(str, "%s%d%c%s", nome_file, 1, 'M', estens);
153     file = fopen(str, "w");
154     fprintf(file, "S MC MC_STD AMC AMC_STD \n");
155
156     for(mc_index = 0; mc_index < NUM_PRINT_n; mc_index++){
157         for(j = 0; j < N_MC /2; j++){
158             price_paths();
159             anti_price_paths();
160             // Ciclo accumula passi MC
161             for(i = 0; i < 2; i++){
162                 acc_mc = MAX((S[i] - E), 0);
163                 C_MC += acc_mc;
164                 C2_MC += acc_mc * acc_mc;
165             }
166             // Ciclo accumula passi Antithetic MC
167             for(i = 0; i < 2; i++){
168                 acc_anti = (MAX((S_anti[i] - E), 0) + MAX
                        ((S[i] - E), 0)) * 0.5;
169                 C_anti += acc_anti;
170                 C2_anti += acc_anti * acc_anti;
171             }
172         }
173         // Calcolo grandezze
174         C_MC = exp(-r * T) * C_MC / N_MC;
175         C2_MC = exp(-r * T) * exp(-r * T) * C2_MC / N_MC;
176         C_anti = exp(-r * T) * C_anti / N_MC;
177         C2_anti = exp(-r * T) * exp(-r * T) * C2_anti /
                N_MC;
178         std_MC = sqrt((C2_MC - C_MC * C_MC) / (N_MC-1));
179         std_anti = sqrt((C2_anti - C_anti * C_anti) / (
                N_MC-1));
180         // Stampa a video V(0)
181         printf("MONTE CARLO \t \t V(%.0f, 0) = %.15lf +-
                %.10lf\n", S_now[mc_index],C_MC, std_MC);
182         printf("ANTITHETIC MONTE CARLO   V(%.0f, 0) = %.10
                lf +- %.10lf\n", S_now[mc_index], C_anti,
                std_anti);
183         // Salva su file
184         fprintf(file, "%f %f %f %f %f", S_now[mc_index],
                C_MC, std_MC, C_anti, std_anti);
185         fprintf(file, "\n");
186     }
187     fclose(file);
188     return 0;
```

```
189 }
190
191
192 // Passo del metodo esplicito
193 void passo_esplicito(){
194     int n;
195     double V_prec = V[0], temp;
196
197     for(n = 1; n < N; n++){
198         temp = V[n];
199         V[n] = 0.5 * (alpha * n * n - beta * n) * V_prec +
                (1 - alpha * n * n - beta) * V[n] + 0.5 * (
               alpha * n * n + beta * n) * V[n+1];
200         V_prec = temp;
201     }
202
203     t += DeltaT;
204 }
205
206
207 /* Passo del metodo implicito:
208    risoluzione dell'equazione matriciale A * v = h (v
         incognito) */
209 void passo_implicito(){
210     int n;
211     double a[N], b[N], c[N], x[N];
212
213     // A = LU decomposition
214     a[1] = 1 + beta + alpha;
215     c[1] = -0.5 * (beta + alpha);
216     for(n = 2; n < N; n++){
217         a[n] = 1 + beta + alpha * n * n + 0.25 * (beta * n
                - alpha * n * n) * (beta * (n - 1) + alpha * (
               n - 1) * (n - 1)) / a[n-1];
218         b[n] = 0.5 * (beta * n - alpha * n * n) / a[n-1];
219         c[n] = -0.5 * (beta * n + alpha * n * n);
220     }
221
222     // Soluzione di L * x = h (x = U * v incognito)
223     x[1] = V[1] - 0.5 * (beta - alpha) * V[0];
224     for(n = 2; n < N-1; n++)
225         x[n] = V[n] - b[n] * x[n-1];
226     x[N-1] = V[N-1] + 0.5 * (N - 1) * (beta + alpha * (N -
         1)) * V[N] - b[N-1] * x[N-2];
227
228     // Soluzione di U * v = x
```

```
229     V[N-1] = x[N-1] / a[N-1];
230     for(n = N-2; n > 0; n--)
231         V[n] = (x[n] - c[n] * V[n+1]) / a[n];
232 
233     // Incremento del tempo
234     t += DeltaT;
235 }
236 
237 // Generazione price paths
238 void price_paths(){
239     double xi[2];
240     xi[0] = (double)rand() / RAND_MAX;
241     xi[1] = (double)rand() / RAND_MAX;
242     Z[0] = sqrt(-2 * log(1 - xi[0])) * cos(M_PI* 2 * xi
          [1]);
243     Z[1] = sqrt(-2 * log(1 - xi[0])) * sin(M_PI* 2 * xi
          [1]);
244     S[0] = S_now[mc_index] * exp((r - 0.5 * sigma * sigma)
          * T + sigma * sqrt(T) * Z[0]);
245     S[1] = S_now[mc_index] * exp((r - 0.5 * sigma * sigma)
          * T + sigma * sqrt(T) * Z[1]);
246 }
247 
248 // Generazione antithetic price paths
249 void anti_price_paths(){
250     S_anti[0] =  S_now[mc_index] * exp((r - 0.5 * sigma *
          sigma) * T + sigma * sqrt(T) * (-Z[0]));
251     S_anti[1] =  S_now[mc_index] * exp((r - 0.5 * sigma *
          sigma) * T + sigma * sqrt(T) * (-Z[1]));
252 }
```

B.2 Codice in linguaggio C - Opzione Asiatica

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <time.h>
5  
6  #define MAX(a, b) ((a) > (b) ? (a) : (b))
7  
8  // Costanti
9  #define N_max 10000 // Numero massimo step MC
10 #define T 1.0 // Payoff time
11 #define E 50.0 // Strike price
12 #define r 0.01 // Tasso di interesse
13 #define sigma 0.4 // Volatilita' del prezzo
```

```c
#define S0 50.0 // Valore del sottostante all'istante iniziale
#define d 4 // Dimensionalita' del problema
#define N_rand 2 // Numero di step per la randomizzazione
#define N_print 1000 // Numero di step tra un salvataggio e il successivo

// Variabili globali
int primes[d];
double xi[d], Z[d];
double S[d+1];
double S_eff, S_eff_mc;
int index_qmc;
double dt = T / (double)d;
double S_new, S_old;
double shift[d];
int N; // Numero step MC, varia per grafici da N_print a N_max

// Prototipi funzioni
double Van_dC(int, double); // Funzione genera sequenza di Van der Corput
void first_d_primes(); // Funzione genera primi d numeri primi
void price_paths_MC(); // Funzione genera andamenti di prezzo con metodo MC
void price_paths_QMC(); // Funzione genera andamenti di prezzo con metodo QMC

// File per salvataggio dati
const char nome_file[] = "asia_esame_"; // Prima parte nome file
const char estens[] = ".txt"; // Estensione file

int main(){
    FILE *file;

    // Variabili locali MC
    double C_MC_fin = 0, C2_MC_fin = 0, C_MC = 0, C2_MC = 0, std_MC, acc_MC;
    // Variabili locali QMC
    double C_QMC = 0, C2_QMC = 0, C_QMC_TOT = 0, C2_QMC_TOT = 0, std_QMC_TOT, acc_QMC;
    // Indici
    int s, i;
    // Scelta seed
```

```
50      srand(time(0));
51
52      S[0] = S0;
53      char str[56];
54
55      first_d_primes();
56      ////////// MONTE CARLO
57      sprintf(str, "%s%d%c%s", nome_file, 1, 'm', estens);
58      file = fopen(str, "w");
59      // Ciclo sui valori di N per grafici
60      for(i = 1; i <= (N_max / (N_print)); i++){
61          for(s = 0; s < N_print; s++){
62              S_eff_mc = S0;//log(S[0]); //Opz. geometriche
63
64              // Genero gli andamenti di prezzo
65              price_paths_MC();
66
67              acc_MC = MAX((S_eff_mc / (d + 1) - E), 0); //
                    MAX((exp(S_eff_mc / (d + 1)) - E), 0); //
                    Opz. geometriche
68              C_MC +=  acc_MC;
69              C2_MC += acc_MC * acc_MC;
70          }
71
72          C_MC_fin = exp(-r * T) * C_MC / (N_print * i);
73          C2_MC_fin = exp(-r * T) * exp(-r * T) * C2_MC / (
                N_print * i);
74          std_MC = sqrt((C2_MC_fin - C_MC_fin * C_MC_fin) /
                ((N_print * i)-1));
75
76          fprintf(file, "%d %.10lf %.10lf \n", i * N_print,
                C_MC_fin, std_MC);
77      }
78      printf("MONTE CARLO   \t  %d %.20lf +- %.20lf\n", (i -
            1) * N_print, C_MC_fin, std_MC);
79      fclose(file);
80
81      ////////// QUASI MONTE CARLO
82      sprintf(str, "%s%d%c%s", nome_file, 1, 'q', estens);
83      file = fopen(str, "w");
84
85      // Ciclo sui valori di N per grafici
86      for(N = N_print; N <= N_max; ){
87          C_QMC_TOT = 0; C2_QMC_TOT = 0;
88          // Ciclo per calcolare errore
89          for(s = 0; s < N_rand; s++){
```

```
90              // Creazione vettore shift
91              for(i = 0; i < d; i++){
92                  shift[i] = (double)rand() / RAND_MAX;
93              }
94              C_QMC = 0;
95              // Ciclo di generazione prezzi casuali e
                   accumulo
96              for(index_qmc = 0; index_qmc < N; index_qmc++)
                   {
97                  S_eff = S0;//log(S[0]); //Opz. geometriche
98                  // Genero gli andamenti di prezzo
99                  price_paths_QMC();
100
101                 acc_QMC = MAX((S_eff / (d + 1) - E), 0);
                       // MAX((exp(S_eff / (d + 1)) - E), 0);
                       //Opz. geometriche
102                 C_QMC +=  acc_QMC;
103             }
104
105             C_QMC = exp(-r * T) * C_QMC / N;
106             C_QMC_TOT += C_QMC;
107             C2_QMC_TOT += C_QMC * C_QMC;
108         }
109
110         C_QMC_TOT = C_QMC_TOT / N_rand;
111         C2_QMC_TOT = C2_QMC_TOT / N_rand;
112         std_QMC_TOT = sqrt((C2_QMC_TOT - C_QMC_TOT *
               C_QMC_TOT) / (N_rand-1));
113
114         fprintf(file, "%d %.10lf %.10lf \n", N, C_QMC_TOT,
                std_QMC_TOT);
115         N += N_print;
116     }
117     printf("QUASI MONTE CARLO %d %.20lf +- %.20lf\n", N -
           N_print, C_QMC_TOT, std_QMC_TOT);
118     fclose(file);
119 }
120
121 void price_paths_MC(){
122     int j;
123     double xi_mc[2], Z_mc[2];
124     S_old = S0;
125     for(j = 1; j <= d / 2; j++){
126         xi_mc[0] = (double)rand() / RAND_MAX;
127         xi_mc[1] = (double)rand() / RAND_MAX;
128         Z_mc[0] = sqrt(-2 * log(1 - xi_mc[0])) * cos(M_PI*
```

```
            2 * xi_mc[1]);
129         Z_mc[1] = sqrt(-2 * log(1 - xi_mc[0])) * sin(M_PI*
            2 * xi_mc[1]);
130
131         S_new = S_old * exp((r - 0.5 * sigma * sigma) * dt
            + sigma * sqrt(dt) * Z_mc[0]);
132         S_eff_mc += S_new; // log(S_new); //Opz.
            geometriche
133         S_old = S_new * exp((r - 0.5 * sigma * sigma) * dt
            + sigma * sqrt(dt) * Z_mc[1]);
134         S_eff_mc += S_old; // log(S_old); //Opz.
            geometriche
135     }
136 }
137
138 void price_paths_QMC(){
139     int i;
140     for(i = 0; i < d; i++){
141         xi[i] = Van_dC(index_qmc, primes[i]);
142         xi[i] = fmod((xi[i] + shift[i]), 1);
143     }
144     for(i = 0; i < d; i++){
145         if((i+2) % 2 == 0)
146             Z[i] = sqrt(-2 * log(1 - xi[i])) * cos(M_PI* 2
                * xi[i+1]);
147         else
148             Z[i] = sqrt(-2 * log(1 - xi[i-1])) * sin(M_PI*
                2 * xi[i]);
149
150         S[i+1] = S[i] * exp((r - 0.5 * sigma * sigma) * dt
            + sigma * sqrt(dt) * Z[i]);
151         S_eff += S[i+1]; //log(S[i+1]); //S[i+1]; //Opz.
            geometriche
152     }
153 }
154
155 /* Calcolo componente index della sequenza di Van der
    Corput con
156    base assegnata, numero restituito in base decimale */
157 double Van_dC(int index, double base){
158     double f = 1, rem = 0;
159     while(index > 0){
160         f = f / base;
161         rem = rem + f * (fmod(index, base));
162         index = index / base;
163     }
```

```
164     return rem;
165 }
166
167 // Riempio vettore con i primi d numeri primi
168 void first_d_primes(){
169     int i = 3, count, c;
170     if (d >= 1){
171         primes[0] = 2;
172     }
173     for(count = 2; count <= d; ){
174         for(c = 2; c <= i - 1; c++){
175             if(i % c == 0)
176                 break;
177         }
178         if(c == i){
179             primes[count - 1] = i;
180             count++;
181         }
182         i++;
183     }
184 }
```

B.3 Script MATLAB per grafici

```
1  clear all; close all;
2  datim = importdata(['asia_esame_1m.txt']);
3  datiq = importdata(['asia_esame_1q.txt']);
4  N = datim(:,1);
5  mc = datim(:,2);
6  smc = datim(:,3);
7  qmc = datiq(:,2);
8  sqmc = datiq(:,3);
9
10 figure();
11 grid on;
12 hold on;
13 box on;
14 pl = plot(N, mc, 'b', 'LineWidth', 1);
15 pl = plot(N, qmc, 'r', 'LineWidth', 1);
16
17 tt = title('Prezzo $V(50,0)$ in funzione del numero di
       step - seed unico', 'Interpreter', 'latex');
18 xx = xlabel('$N$ - numero di step della simulazione', '
       Interpreter', 'latex');
19 yy = ylabel("V[] - prezzo dell'opzione", 'Interpreter', '
       latex');
```

```
20 ll = legend('MC: Generazione pseudo-random', 'QMC:
      Generazione Halton seq.','Interpreter', 'latex');
21 set(xx, 'FontSize', 14);
22 set(yy, 'FontSize', 14);
23 set(tt, 'FontSize', 14);
24 set(ll, 'FontSize', 12);
25 %xlim([]);
26 ylim([4.00 4.8]);
27
28 figure();
29 grid on;
30 hold on;
31 box on;
32 pl = semilogy(N, smc, 'b', 'LineWidth', 1);
33 pl = plot(N, sqmc, 'r', 'LineWidth', 1);
34 set(gca, 'Yscale', 'log');
35
36 tt = title('Standard deviation in funzione del numero di
      step', 'Interpreter', 'latex');
37 xx = xlabel('$N$ - numero di step della simulazione', '
      Interpreter', 'latex');
38 yy = ylabel("Deviazione standard []", 'Interpreter', '
      latex');
39 ll = legend('MC: Generazione pseudo-random', 'QMC:
      Generazione Halton seq.','Interpreter', 'latex');
40 set(xx, 'FontSize', 14);
41 set(yy, 'FontSize', 14);
42 set(tt, 'FontSize', 14);
43 set(ll, 'FontSize', 12);
44 %xlim([]);
45 ylim([2e-4 1]);
```