Politecnico di Milano

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Matematica

Tesi di laurea magistrale

Calibrazione di modelli e pricing di derivati path-dependent

Sviluppo di un software di pricing per derivati esotici attraverso GUI MATLAB

Candidato:
Riccardo Reale
Matricola 798991

Relatore:
Daniele Marazzina

Anno Accademico 2013–2014

Ringraziamenti

Desidero innanzitutto ringraziare il Professor Marazzina per le numerose ore dedicate alla mia tesi e per essere sempre stato disponibile e presente, nonostante la distanza geografica che separa Francoforte da Milano. Un sentito ringraziamento per il tempo che mi ha dedicato e i consigli che ha saputo rivolgermi. Un enorme grazie ai miei genitori, per l'immancabile supporto e per l'infinita fiducia che avete riposto in me durante tutta la mia vita. Mi hanno permesso di intraprendere questo percorso supportandomi sino a questo momento senza mai farmi mancare nulla, con il loro incrollabile sostegno mi hanno consentito di raggiungere questo traguardo. Grazie perché so che potrò sempre contare su di voi. Grazie a Francesca, la migliore sorella maggiore che si possa desiderare. Grazie a tutti i miei familiari, che si sono sempre dimostrati orgogliosi di me, in particolare un grazie ai miei nonni sempre entusiasti. Grazie ai miei compagni di studio qui al Politecnico, avete saputo essere veri amici piuttosto che semplici compagni. Thank you to all my colleagues in the ECB, we really have a great time together. Grazie a Stefano per avermi sapientemente iniziato al mondo del lavoro. Un grazie molto speciale a Didì per aver camminato al mio fianco dall'inizio alla fine, mi hai insegnato più di quanto tu possa immaginare.

Milano, Luglio 2014 R. R.

The purpose of models is not to fit the data but to sharpen the questions.

Samuel Karlin, April 1983

Per aspera ad astra.

Abstract

L'obiettivo di questa tesi è quello di studiare il problema di calibrazione di modelli e valutazione di derivati, realizzando un software di pricing, ovvero uno strumento per catturare le caratteristiche dei prezzi delle opzioni liquide, quotate sul mercato, ed estrapolare il prezzo di strumenti più esotici. La procedura consiste nel partire dai dati di opzioni reali, eseguire la calibrazione e procedere poi con il pricing. In particolare la calibrazione viene effettuata su opzioni Call europee ed il pricing è di derivati barriera, asiatici e lookback. Nella prima parte dell'elaborato vengono introdotte le nozioni di calcolo stocastico necessarie per l'introduzione dei diversi modelli per l'andamento di un sottostante azionario. In seguito vengono introdotti alcuni metodi di pricing, tra cui l'albero binomiale, il Monte Carlo ed alcune tecniche allo stato dell'arte nell'ambito della calibrazione, alcune delle quali basate sulla FFT. In particolare viene implementato il metodo COS, che permette di valutare opzioni Europee, Barriera ed Asiatiche, con un errore che decade esponenzialmente. Vengono effettuati confronti numerici su dati reali al fine di individuare il metodo più veloce ed affidabile. L'output finale consiste nella creazione di un'interfaccia grafica su `MATLAB` che consente all'utente di selezionare il modello del sottostante tra una gamma di processi di Lévy o a volatilità stocastica e il metodo con cui effettuare la calibrazione. Una volta identificata una delle possibili misure neutrali rispetto al rischio è possibile specificare le caratteristiche dell'opzione esotica che si desidera valutare e lanciare il pricing, scegliendo di volta in volta la tecnica desiderata tra quelle applicabili all'opzione prescelta.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Il problema del pricing di derivati esotici è di fondamentale importanza per gli operatori finanziari che si trovano nella condizione di acquistare o emettere questi tipi di contratti. In particolare la necessità di essere in grado di valutare in modo corretto le opzioni esotiche si scontra con il fatto che sono tipicamente molto meno liquide delle loro controparti Vanilla (anche dette opzioni europee). Infatti per poter eseguire il pricing occorre conoscere i parametri del modello che si desidera utilizzare per descrivere la dinamica del sottostante. La procedura che si occupa di ricavarli prende il nome di calibrazione del modello e necessita di dati relativi ad opzioni liquide per essere effettuata correttamente. Pertanto questa operazione viene svolta grazie all'utilizzo delle informazioni relative alle opzioni europee. Una volta ricavati i parametri è poi possibile utilizzare il modello per prezzare anche i derivati meno liquidi. Di conseguenza il problema del pricing non può prescindere da quello della calibrazione delle opzioni europee. Queste ultime opzioni danno il diritto ma non l'obbligo di acquistare o vendere (in tal caso sono denominate rispettivamente Call e Put), il titolo sottostante ad un prezzo prefissato, detto strike price. Nel caso delle europee esiste una sola possibile data di esercizio, detta maturity dell'opzione.

I lavori empirici sul problema della calibrazione in maniera efficiente si sono sviluppati principalmente nell'ambito delle tecniche basate sulla trasformata di Fourier. Gli esempi principali di questi lavori sono Carr&Madan (1999) [4], Lord et al. [11], Fang et al. [6, 7] (2008-2009), che hanno studiato il modo di rendere la calibrazione sempre meno onerosa dal punto di vista computazionale.

In questo contesto si inserisce il presente lavoro di tesi, che mira a sviluppare un software che sia in grado di occuparsi di entrambi gli aspetti di questo problema. Per sviluppare l'analisi si farà ricorso al linguaggio `Matlab`, con il quale anche la *Graphical User Interface* per l'utente è stata scritta.

Il software prende in ingresso i dati relativi alle option chain di Call europee scritte sul sottostante per il quale si desidera effettuare il pricing. Queste devono essere preventivamente ottenute da un provider di informazioni e copiate in un foglio Excel

opportunamente formattato, che dovrà essere inserito nella cartella di Input. A titolo di esempio verranno riportati i risultati ottenuti a partire da una serie storica di opzioni scritte su Apple Inc, sono in più state incluse anche serie storiche relative a Microsoft nel dataset predefinito a cui il software finale può attingere.

All'utente è lasciata poi la possibilità di selezionare uno dei modelli stocastici per l'andamento del sottostante tra i seguenti: Black&Scholes, Merton, Kou, Variance Gamma, Normal Inverse Gaussian ed Heston. Il modello di Black&Scholes è stato scelto, oltre che per la sua importanza storica, perché le sue caratteristiche di rapidità, a scapito però dell'accuratezza, lo rendono un interessante termine di paragone con gli altri modelli. Sono stati poi scelti quattro modelli appartenenti alla categoria dei processi di Lévy con salti, i primi due ad attività finita e gli ultimi due ad attività infinita. Infine il modello di Heston è stato scelto quale esponente della classe dei modelli a volatilità stocastica, che rappresentano una seconda possibile evoluzione del semplice modello di Black&Scholes.

La calibrazione si può considerare come il problema inverso del pricing, mentre in quest'ultimo sono noti i parametri del modello e da questi si ricava un prezzo, nella calibrazione i prezzi vengono presi dal mercato e lo scopo è risalire ad un modello che descriva la dinamica del sottostante nel modo più fedele possibile. Nella calibrazione è possibile selezionare un metodo di pricing tra i seguenti: Albero Binomiale, Carr & Madan, CONV, COS, mentre per il pricing, oltre ai metodi già citati, è sempre disponibile anche il metodo di Monte Carlo. L'albero binomiale è stato introdotto da Cox-Ross-Rubinstein ed è applicabile al solo caso di sottostante con dinamica data da Black&Scholes, il prezzo evolve in maniera discreta con una distribuzione di probabilità scelta in modo da convergere a Black&Scholes al ridursi del passo temporale di discretizzazione. I metodi di Carr&Madan, CONV, COS consentono un pricing rapido delle opzioni europee per diversi strike contemporaneamente, caratteristica che li rende particolarmente adatti alla calibrazione, inoltre il CONV è generalizzabile al caso di opzioni barriera, mentre il COS a barriera ed asiatiche. Il metodo di Monte Carlo, introdotto da Stanislaw Ulam alla fine degli anni '40, si basa sulla simulazione dell'intera traiettoria dei prezzi del sottostante, e si adatta bene al pricing delle opzioni esotiche che verranno considerate. Si verificherà che la scelta migliore è utilizzare il COS, per la sua velocità ed accuratezza.

Le opzioni esotiche che possono essere prezzate sono le versioni Call o Put delle opzioni Barriera di tipo Out, delle Lookback e delle Asiatiche, sia nelle varianti fixed che floating strike. Tutte queste opzioni sono di tipo path-dependent, ovvero il loro payoff dipende non soltanto dal valore del sottostante a scadenza, ma anche dall'evoluzione del prezzo durante tutta la vita dell'opzione. Nel caso delle opzioni Barriera se il prezzo non si mantiene in un determinato intervallo di valori l'opzione si annulla. Le Lookback sono strutturate in modo che il payoff dipenda dal massimo o dal minimo valore raggiunto dal prezzo del sottostante, mentre nel caso delle asiatiche si considera la media dei prezzi.

L'obiettivo sarà dunque quello di comprendere innanzitutto il problema della calibrazione, confrontando i vari metodi di ottimizzazione offerti da `Matlab` in termini di efficienza e costo computazionale e poi di applicare le conoscenze così ricavate al problema del pricing. Queste osservazioni saranno poi utilizzate per costruire una

GUI che consenta ad un utente, con pochi semplici passi, di ottenere il prezzo di alcuni derivati esotici.

Per raggiungere l'obiettivo prefissato il presente lavoro di tesi si articola nel seguente modo:

Il Capitolo 1 inizia fornendo le nozioni preliminari necessarie alla comprensione del resto della tesi, queste includono molteplici risultati nel campo dei processi stocastici. In seguito vengono presentati alcuni dei modelli più diffusi per la trattazione dei sottostanti azionari, elencandone le specificità e l'applicabilità. Particolare attenzione è dedicata al modello di Black&Scholes, e ne vengono elencati pregi e difetti in comparazione ai modelli più sofisticati. Vengono poi trattati i processi di Lévy sia ad attività finita che infinita, e in particolare vengono introdotti i processi di Merton, Kou, Variance Gamma, Normal Inverse Gaussian. Viene poi presentato il processo di Heston a titolo di esponente della categoria dei processi a volatilità stocastica. Infine viene richiamata la nozione di risk neutral pricing, necessaria per l'applicazione pratica dei modelli presentati.

Il Capitolo 2 presenta tutte le tecniche di pricing che saranno utilizzate nel corso della tesi. In primo luogo viene descritto il metodo dell'albero binomiale di Cox-Ross-Rubinstein. Si passa poi all'introduzione del metodo di Monte Carlo, per il quale viene anche spiegato come si possono simulare le traiettorie dei processi visti in precedenza. Infine vengono introdotti i metodi basati sulle trasformate di Fourier: il metodo di Carr&Madan [4], il metodo CONV [11] e il metodo COS [6, 7]. Per ciascuno di essi viene dapprima effettuata un'introduzione teorica e poi vengono ricavate le formule da utilizzare per il pricing.

Il Capitolo 3 si occupa del problema della calibrazione dei processi stocastici su dati reali di option chain scritte su sottostanti quotati. In primo luogo si formalizza matematicamente il problema e vengono introdotte le grandezze necessarie alla sua risoluzione. Vengono effettuate prove numeriche al fine di identificare, tra le routine messe a disposizione da `Matlab`, quella più adatta allo scopo, la quale sarà poi utilizzata in via esclusiva nel seguito della tesi. Vengono poi confrontate le velocità di convergenza a zero dell'errore per i metodi basati sulla FFT e per il COS, al fine di identificare il metodo più efficiente. Infine tutti i metodi vengono applicati ai dati reali e vengono presentati dei grafici riassuntivi che confrontano la discrepanza tra i prezzi prescritti dal modello e quelli ricavati dal mercato. Un'analisi simile è poi condotta in termini di volatilità implicite.

Il capitolo 4 descrive dettagliatamente il funzionamento della Graphical User Interface che è stata sviluppata, discernendo tra la parte di calibrazione e quella di pricing. Vengono indicati l'ambito delle funzionalità, la formattazione necessaria per gli input e le modalità di generazione dell'output. La GUI è stata sviluppata tenendo in mente le necessità di semplicità di utilizzo e praticità di un possibile utente, i risultati parziali, quali i parametri calcolati dalla calibrazione o i tempi di esecuzioni vengono riportati nella GUI in tempo

reale per fornire la massima informatività possibile. Infine vengono riportati alcuni risultati ottenuti su opzioni specifiche, al fine di verificare il corretto funzionamento del software. Le serie storiche che è possibile utilizzare per il pricing sono le quattro descritte nell'appendice seguente, rimane comunque aperta la possibilità di inserire nella GUI ulteriori serie storiche, con le modalità descritte in questo capitolo.

L'appendice A presenta le serie storiche preimpostate nella GUI, tra cui quella utilizzata per i test nella parte della calibrazione, individuandone le caratteristiche salienti e le specificità. In particolare vengono considerate le serie storiche delle opzioni Call scritte sui colossi informatici Americani Apple e Miscrosoft, con scadenza a Gennaio 2015 e Gennaio 2016.

L'appendice B riporta i codici utilizzati per i calcoli numerici. Vengono indicate le funzioni utilizzate per l'effettivo pricing dei derivati e le funzioni ausiliarie da esse impiegate, quali ad esempio le librerie delle funzioni caratteristiche dei modelli per il sottostante. Infine viene anche riportato il codice completo che consente di generare la GUI.

Capitolo 1

Modelli per il sottostante

In questo capitolo vengono presentati i modelli che saranno utilizzati per la rappresentazione delle traiettorie del titolo finanziario sottostante ai derivati valutati in questo elaborato. Per ciascuna classe di modelli vengono inizialmente introdotti i concetti e le definizioni necessari per la trattazione, in seguito si ricavano alcuni risultati teorici principali e si fornisce una simulazione numerica delle traiettorie di ciascun processo stocastico.

Come punto di partenza viene introdotto il modello di Black&Scholes, il quale viene poi generalizzato mediante le due principali tecniche utilizzate in letteratura: introduzione di salti e volatilità stocastica. Queste generalizzazioni vengono studiate per cercare di rendere le traiettorie simulate più aderenti alle proprietà ("fatti stilizzati") delle serie storiche dei prezzi di mercato, quali smile di volatilità, eteroschedasticità, "code grasse" e non normalità dei log-rendimenti.

Nell'ottica di fare poi pricing utilizzando le tecniche basate sulla trasformata di Fourier vengono richiamati i concetti di funzione caratteristica, esponente caratteristico e viene ricavata la condizione di risk-neutral pricing che tutte le funzioni caratteristiche dovranno incorporare.

1.1 Nozioni preliminari

1.1.1 Processi stocastici

La prima definizione è quella di σ-algebra, utilizzata per codificare l'informazione disponibile relativamente al realizzarsi di determinati eventi.

Definizione 1.1.1 (σ-Algebra)**.** Dato un'insieme Ω e una famiglia $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ di suoi sottoinsiemi si dice che $\mathcal{F}$ è una σ-algebra se $\Omega \in \mathcal{F}$ e soddisfa le proprietà di:

- Stabilità rispetto a complementazione: $A \in \mathcal{F} \implies A^c \in \mathcal{F}$;
- Stabilità rispetto a unione finita: $A_1, \ldots, A_n \in \mathcal{F} \implies A_1 \cup, \ldots, \cup A_n \in \mathcal{F}$;

- Stabilità rispetto a intersezione numerabile: $A_1, A_2, \cdots \in \mathcal{F} \implies \cup_{n=1}^{\infty} A_n \in \mathcal{F}$.

Ω è l'insieme degli stati del mondo, ogni elemento $\omega \in \Omega$ rappresenta uno dei possibili stati del mercato. Ciascun insieme $A \in \mathcal{F}$ è detto *evento* e corrisponde ad un insieme di scenari a cui è possibile associare una probabilità.

Definizione 1.1.2 (Spazio di Probabilità)**.** Si dice spazio di probabilità la terna $(\Omega, \mathcal{F}, \mathbb{P})$ dove

- Ω è uno spazio campionario
- $\mathcal{F}$ è una σ-algebra su Ω
- $\mathbb{P} : \mathcal{F} \mapsto [0,\, 1]$ è una probabilità, $\mathbb{P}(\Omega) = 1$.

A questo punto è possibile introdurre la nozione fondamentale di processo stocastico. Tutta la modellizzazione seguente è basata su questa struttura matematica di famiglia di variabili aleatorie indicizzate nel tempo.

Definizione 1.1.3 (Processo Stocastico)**.** Si dice processo stocastico la collezione $X = (\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in T}, (X_t)_t \in T, \mathbb{P})$ tale che:

- $(\Omega, \mathcal{F}, \mathbb{P})$ è uno spazio di probabilità
- $T \subseteq \mathbb{R}^+$ è l'insieme dei tempi considerati
- $(\mathcal{F}_t)_{t\in T}$ è una filtrazione, ovvero una famiglia crescente di sotto σ-algebre di $\mathcal{F}$. Vale quindi che $\forall s, t \in T \,:\, s \leq t,\ \mathcal{F}_s \subseteq \mathcal{F}_t \subseteq \mathcal{F}$.
- $X_t : (\Omega, \mathcal{F}) \mapsto (E, \mathcal{E})$ famiglia di variabili aleatorie a valori nello spazio misurabile $(E, \mathcal{E})$ è adattata alla filtrazione $\mathcal{F}_t$. Ovvero $(X_t)_t$ è misurabile rispetto a $\mathcal{F}_t\ \forall t \in T$.
- $\mathbb{P}$ è una probabilità su $(\Omega, \mathcal{F})$.

Nel seguito spesso si farà riferimento ad un processo stocastico semplicemente con $(X_t)_t$, sottintendendo gli altri argomenti. L'intepretazione di $\mathcal{F}_t$ è quella di informazione disponibile al tempo t, il fatto che la famiglia sia crescente indica l'accumularsi dell'informazione al trascorrere del tempo. Nel caso in cui la filtrazione non sia specificata si è soliti sottintendere la filtrazione naturale $\mathcal{F}_t = \sigma(X_s,\ 0 \leq s \leq t)$. Nel seguito, con un lieve abuso di notazione, si farà riferimento ai processi stocastici indicando la sola famiglia di variabili aleatorie $(X_t)_t \in T$.

1.1.2 Martingale

Un ruolo importante nella teoria finanziaria è occupato dalle martingale, particolari processi stocastici che possono essere utilizzati per modellizzare il patrimonio di un giocatore nei giochi equi, nei quali in media non si perde né si guadagna. In particolare vale che in un certo istante temporale l'aspettativa sul valore del processo stocastico in un tempo futuro, nota tutta la storia del processo, è uguale al valore presente. Formalmente:

Definizione 1.1.4 (Martingala)**.** Un processo stocastico $M = (M_t)_{t\geq 0}$ a valori in $\mathbb{R}$ è una martingala se:

- $M_t \in \mathbb{L}^1(\Omega, \mathcal{F}, \mathbb{P})\ \forall t \in T$
- $\mathbb{E}[M_t|\mathcal{F}_s] = M_t\ \forall\, 0 \leq s \leq t$

1.1.3 Funzione generatrice dei momenti

Come prima cosa è necessario introdurre la nozione di momento di una variabile aleatoria.

Definizione 1.1.5 (Momenti di ordine k)**.** Sia X una variabile aleatoria con densità f_X, si definisce momento di ordine k la quantità

$$m_k(X) = \int_{\mathbb{R}} x^k f_X(x)dx = \mathbb{E}[X^k]$$

se l'integrale è ben definito.

Per il calcolo dei momenti si può ricorrere alla funzione generatrice dei momenti

$$M_X(z) = \int_{\mathbb{R}} e^{zx} f_X(x)dx$$

che non è altro che la trasformata di Laplace della densità $f_X(x)$. Nota la $M_X(z)$ è possibile conoscere i momenti della distribuzione semplicemente derivando:

$$m_k(X) = \frac{d^k}{dz^k} M_X(z)|_{z=0} \tag{1.1}$$

1.1.4 Funzione caratteristica

Un'altra importante nozione è quella di funzione caratteristica del processo, essa si ottiene se si considera la trasformata di Fourier della densità $\Phi_X(z) : \mathbb{R} \mapsto \mathbb{C}$.

$$\Phi_X(z) = \mathbb{E}[X^{\mathrm{i}\, zX}] = \int_{\mathbb{R}} e^{\mathrm{i}\, zx} f_X(x)dx = M_X(\mathrm{i}\, z)$$

La funzione caratteristica è definita per ogni densità di probabilità e consente di identificare univocamente una data densità.

Se la funzione caratteristica ammette una rappresentazione della forma

$$\Phi_X(z) = e^{\Psi_X(z)}$$

si dice che $\Psi_X(z)$ è l'esponente caratteristico di X.

Teorema 1.1.1. *Date due variabili aleatorie X e Y vale che:*
$\Phi_X(z) \equiv \Phi_Y(z) \iff$ *X e Y hanno la distribuzione di probabilità.*

Grazie all'equivalenza garantita questo teorema, del quale è possibile trovare una dimostrazione ad esempio in [9], è possibile spostare l'analisi dei processi nello spazio di Fourier. Questa possibilità è alla base delle tecniche, mostrate nei capitoli successivi, che si avvalgono della maggiore efficienza computazionale dei calcoli nello spazio di Fourier.

1.2 Il modello di Black&Scholes

1.2.1 Moto Browniano

Il processo di Wiener (o moto Browniano), è sicuramente il più diffuso tra tutti i modelli utilizzati per la rappresentazione delle fluttuazioni del prezzo di mercato di un titolo. Un moto Browniano W_t è caratterizzato da incrementi indipendenti, stazionari e che seguono una distribuzione gaussiana.

Definizione 1.2.1 (Moto Browniano)**.** $W = (\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, (W_t)_t \geq 0, \mathbb{P})$ processo stocastico è moto Browniano se:

1. $W_0 = 0$ q.c.
2. $W_t - W_s \perp \mathcal{F}_s \ \forall\, 0 \leq s \leq t$
3. $W_t - W_s \sim N(0, t-s)$

Si può dimostrare che il moto Browniano è una martingala.

L'esponente caratteristico del moto Browiano è pari a:

$$\Psi_t(u) = -\frac{\sigma^2 u^2}{2} \tag{1.2}$$

A partire dal moto Browniano standard è possibile costruire il moto Browniano con drift

$$X_t = \mu t + \sigma W_t$$

il suo esponente caratteristico è

$$\Psi_t(u) = \mathrm{i}\,\mu u - \frac{\sigma^2 u^2}{2} \tag{1.3}$$

Proprietà principali delle traiettorie del moto browniano:

- le traiettorie del moto Browniano $t \mapsto W_t(\omega)$ sono q.c. non differenziabili.
- il moto Browniano è continuo e γ-holderiano per ogni $0 < \gamma < \frac{1}{2}$.
- le traiettorie del moto Browniano $t \mapsto W_t(\omega)$ sono a variazione infinita q.c. su ogni intervallo di tempo.

1.2.2 Modello lognormale

Utilizzando il moto Browniano è possibile definire il processo lognormale o di moto Browniano geometrico (GBM). Il modello di Black&Scholes assume che il prezzo dei titoli segua un andamento di tipo GBM.

$$\begin{cases} dS_t = \mu S_t dt + \sigma S_t dW_t \\ S_0 = s_0 \end{cases} \tag{1.4}$$

Dove μ è il termine di *drift* e σ è il termine di *diffusione* del processo.

Il processo può anche essere scritto in forma locale come

$$\ln\left(\frac{S_t}{S_0}\right) = X_t = \mu t + \sigma W_t;$$

in forma integrale, invece, si ottiene

$$S_t = S_0 e^{X_t} = S_0 e^{\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t}. \tag{1.5}$$

La scelta del modello lognormale risponde all'esigenza modellistica di concentrarsi sugli incrementi relativi del prezzo piuttosto che su quelli assoluti. Il rendimento di un titolo tra t e $t + dt$ è dato da:

$$\frac{S_{t+dt} - S_t}{S_t} = \frac{dS_t}{S_t}. \tag{1.6}$$

Si suppone che i rendimenti siano guidati da una parte deterministica, il drift, identificato dal coefficiente μ, e da una parte stocastica, la diffusione, misurata dalla deviazione standard σ. Quindi in un intervallo dt si ha un contributo μdt non stocastico e una parte aleatoria σdW_t.

In Figura 1.1 è riportata la simulazione numerica di alcune traiettorie del processo di Wiener, assieme al valore atteso. Si vede come la componente stocastica comporti delle fluttuazioni attorno alla media, tanto più grande è σ tanto più gli scostamenti sarano marcati. In Figura 1.2 sono riportati gli incrementi di una traiettoria del processo X_t. Si sono scelti i parametri $\mu = 1$, $\sigma = 0.2$, l'orizzonte temporale è $T = 1$ mentre la griglia temporale contiene 252 intervalli.

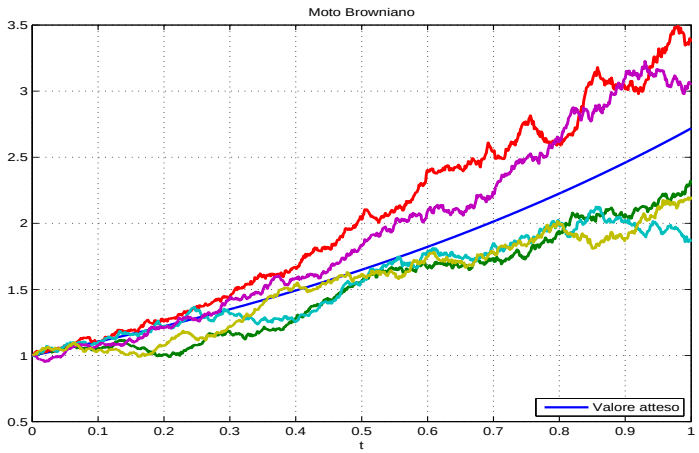


Figura 1.1: Simulazione traiettoria B&S

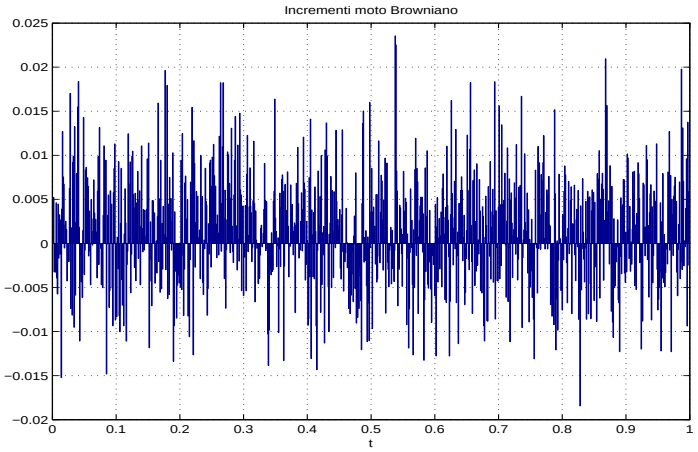


Figura 1.2: Log-incrementi traiettoria B&S

1.2.3 Pricing

Vi è tipicamente un tradeoff tra la semplicità del modello e la trattabilità analitica delle soluzioni che ne scaturiscono. In particolare per il semplice modello di Black&Scholes è possibile ottenere in forma chiusa l'espressione del prezzo di un'opzione europea. La formula nel caso della Call è la seguente

$$C(S_t, K, r, \sigma) = S_t N(d_1) - e^{-r(T-t)} K N(d_2), \tag{1.7}$$

dove

$$d_1 = \frac{\ln(\frac{S_t}{K}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}},$$
$$d_2 = d_1 - \sigma\sqrt{(T-t)}.$$

Per una derivazione si faccia riferimento ad esempio a [3]. Tale espressione semplice da valutare e dal ridotto costo computazionale è alla base del successo di questo modello.

1.2.4 Limiti del modello di Black&Scholes

Se da un lato la semplicità teorica e computazionale del modello di Black&Scholes ne hanno decretato il grande successo, dall'altro il modello non è privo di difetti. Uno dei motivi principali per allontanarsi dai modelli gaussiani in finanza è dato da alcune proprietà empiricamente osservabili che caratterizzano i rendimenti degli asset e che sono in disaccordo con il GBM.

La letteratura accademica ha dedicato molta attenzione allo studio delle proprietà delle serie storiche finanziarie. Analizzando tutti i mercati, anche i più grandi e liquidi, e considerando un insieme ampio di titoli diversi per natura e su diversi periodi temporali si notano delle regolarità empiriche che vengono dette *fatti stilizzati*. Le variazioni apparentemente casuali dei prezzi degli asset condividono alcune proprietà statistiche non triviali. Per la trattazione si fa riferimento a [2, 5].

1. **Continuità** Una delle più importanti proprietà del moto Browniano è la continuità delle sue traiettorie, che non sembra però essere condivisa dai prezzi reali di mercato, i quali presentano dei salti.

2. **Code pesanti** La distribuzione dei rendimenti sembra avere code pesanti con un eccesso positivo di curtosi[1].

3. **Asimmetria** Si osservano grandi salti negativi nei prezzi delle azioni e nei valori degli indici che non sono bilanciati da movimenti verso l'alto egualmente significativi.

4. **Normalità aggregate** All'aumentare dell'orizzonte temporale dt sul quale sono calcolati i rendimenti, la loro distribuzione sembra sempre più simile ad una normale. In particolare la forma non è la stessa su diverse scale temporali: le caratteristiche code pesanti diventano meno pronunciate se l'orizzonte cresce.

5. **Volatility clustering** Variazioni grandi o piccole in valore assoluto tendono ad essere seguite da variazioni di simile ampiezza, di qualsiasi segno. Una manifestazione quantitativa del fenomeno è che sebbene i rendimenti sembrino incorrelati nel tempo, i rendimenti assoluti (o i loro quadrati) mostrano una correlazione positiva. I rendimenti non sono quindi indipendentemente distribuiti nel tempo, cioé la volatilità stessa è variabile e presenta effetti di eteroschedasticità[2].

6. **Effetto leva** La volatilità e il rendimento del sottostante sono negativamente correlati.

7. **Smile di volatilità** Dall'analisi delle serie storiche di mercato si evince che la scelta di σ costante non è giustificabile, in realtà si osserva che $\sigma = \sigma(K, T)$. Invertendo la (1.7) per ricavare σ, inserendo i prezzi di mercato delle opzioni, e plottando rispetto allo strike K si ottiene il c.d. *smile di volatilità*. Questa caratteristica forma è dovuta al fatto che la volatilità tende ad essere maggiore per le opzioni far out of the money o deep into the money.

Almeno qualitativamente i fatti stilizzati sono così vincolanti che non è semplice esibire un processo stocastico che possegga lo stesso insieme di proprietà. La letteratura ha comunque realizzato numerosi passi avanti tramite lo sviluppo di diverse classi di modelli, come i modelli con salti e quelli a volatilità stocastica, i quali riescono a riprodurre in maniera più realistica i rendimenti. Quando nel seguito della tesi verranno mostrati i risultati numerici relativi al fitting di dati reali con diversi modelli è bene tenere conto di queste osservazioni per interpretare i peggiori risultati del modello di Black&Scholes rispetto agli altri modelli più raffinati. La spiegazione del fenomeno non è legata soltanto al maggior numero di gradi di libertà dei modelli più raffinati, bensì al fatto che questi ultimi prevedano la modellizzazione dei fatti stilizzati. Poiché, come osservato precedentemente, questi fenomeni sembrano caratterizzare tutti i sottostanti azionari, tenerne conto porta ad

[1] La curtosi $\kappa = \frac{m_4}{m_2^2} - 3$ è un indice descrittivo della forma di una distribuzione di probabilità che costituisce una misura dello spessore delle code della funzione densità.

[2] In statistica si parla di eteroschedasticità quando la varianza di una variabile casuale varia tra le diverse osservazioni campionarie.

un miglioramento della qualità del pricing. Dall'altro lato il GBM è anche il modello con il minor costo computazionale e possiede un solo parametro da calibrare, di conseguenza la sua appetibilità nelle situazioni pratiche è comunque grande: nella pratica si cerca di utilizzare il GBM ogni volta la situazione lo permette.

1.3 Modelli con salti

Introduzione Una generalizzazione del caso precedente è offerta dai processi di Lévy, diffusamente trattati in [5]. Questi processi prendono il nome dal matematico francesce Paul Lévy (1886-1971) che fu il primo a descriverne la proprietà e a caratterizzarne la struttura. In realtà il processo di Wiener stesso è un caso particolare di processo di Lévy, esso è infatti l'unico Lévy a traiettorie continue. Una delle considerazioni alla base dell'introduzione dei salti nella dinamica è quella che repentine fluttuazioni possano inspessire le code della distribuzione dei log-rendimenti: questi processi sono in grado di generare rendimenti con una distribuzione leptocurtica (cioé hanno code più grasse rispetto alla distribuzione normale assunta da Black&Scholes). Si osserva inoltre che i processi di Lèvy riescono a riprodurre efficacemente lo smile di volatilità soprattutto per le brevi scadenze. Il problema principale di questi modelli sta nel fatto che non riescono a catturare gli effetti di volatility clustering, cosa che invece riscono a fare i modelli a volatilità stocastica. In questa sezione vengono introdotte le nozioni necessarie allo studio di questi processi, mentre nelle successive vengono descritti i processi effettivamente utilizzati nell'elaborato.

1.3.1 Processo di Poisson

Processi di conteggio Data una successione crescente di tempi casuali $T_n, n \geq 1$ si può definite un processo di conteggio come

$$X_t = \sum_n \mathbb{I}_{t \geq T_n} = \#\{n \geq 1,\ T_n \leq t\}.$$

Tale processo conta il numero di tempi casuali (T_n) che cadono tra 0 e t. Un caso particolare è il processo di Poisson, dove i tempi T_n sono costruiti come sommatorie parziali di una successione di variabili aleatorie esponenziali i.i.d.[3].

Si introducono ora le variabili aleatorie con legge Poissoniana ed esponenziale. Queste leggi sono di particolare importanza per la definizione del processo di Poisson.

Definizione 1.3.1 (Variabile Poissoniana)**.** Una variabile aleatoria N a valori in $\mathbb{N}$ ha una distribuzione di Poisson di parametro λ se

$$\mathbb{P}(N = n) = e^{-\lambda}\frac{\lambda^n}{n!} \quad \forall n \in \mathbb{N}$$

Definizione 1.3.2 (Variabile Esponenziale)**.** Una variabile aleatoria τ a valori in $\mathbb{R}^+$ ha una distribuzione di esponenziale di parametro λ se ha densità

$$f_\tau(t) = \lambda e^{-\lambda t}\mathbb{I}_{t \geq 0}$$

[3] Indipendenti Identicamente Distribuite

Definizione 1.3.3 (Processo di Poisson)**.** Sia $(\tau_i)_{i\geq 1}$ una successione di variabili aleatorie indipendenti esponenziali di parametro λ e $T_n = \sum_{i=1}^{n} \tau_i$. Il processo $(N_t)_{t\geq 0}$ definitito da

$$N_t = \sum_{n\geq 1} \mathbb{I}_{t\geq T_n}$$

è detto processo di Poisson.

La sua funzione caratteristica è

$$\Phi_t(u) = e^{\lambda t\left(e^{i\,u}-1\right)}. \tag{1.8}$$

Il processo appena introdotto è anche detto processo di Poisson omogeneo, per differenziarlo dal processo di Poisson composto che verrà introdotto in seguito, il quale, avendo ampiezze dei salti stocastiche, non è più un semplice processo di conteggio. Le variabili aleatorie τ_i sono da interpretarsi come il tempo che intercorre tra un salto e l'altro e di conseguenza le variabili T_n rappesentano gli istanti di salto del processo.

Per ogni ω le traiettorie del processo $t \mapsto N_t(\omega)$ sono costanti a tratti con salti di ampiezza unitaria in corrispondenza di tempi aleatori. In Figura 1.3 è riportata la simulazione numerica di una traiettoria del processo. Si è scelto il parametro $\lambda = 2$, e l'orizzonte temporale è $T = 10$.

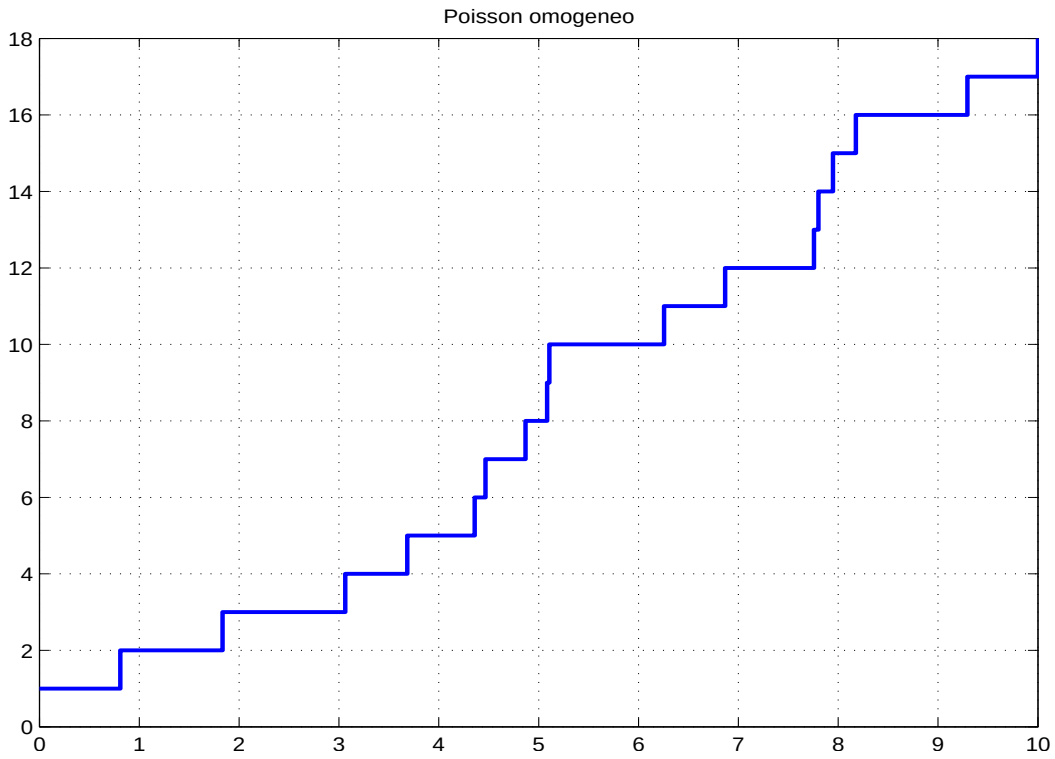


Figura 1.3: Simulazione traiettoria di un Poisson omogeneo

Si può rendere Martingala il processo di Poisson compensandolo con un drift deterministico.

$$\tilde{N}_t = N_t - \lambda t$$

In tal modo si ottiene $\mathbb{E}[\tilde{N}_t|\tilde{N}_s] = \tilde{N}_s \ \forall s \leq t$. Si osservi che $\tilde{N}_t$, a differenza di N_t non è a valori interi. La sua funzione caratteristica è

$$\Phi_t(u) = e^{\lambda t\left(e^{\mathrm{i}\,u}-1-\mathrm{i}\,u\right)} \tag{1.9}$$

1.3.2 Processi di Lévy

I processi di Wiener e di Poisson precedentemente introdotti ricadono in questa classe, anzi ne sono gli esponenti fondamentali, in quanto è possibile pensare ad un processo di Lévy generico come alla sovrapposizione di un moto Browniano e ad una somma (anche infinita) di processi di Poisson indipendenti.

A differenza del moto browniano i processi di Lévy non hanno in generale traiettorie continue, è quindi necessario introdurre una classe particolare di funzioni in cui ambientare lo studio dei processi.

Definizione 1.3.4 (Funzione cadlag)**.** Una funzione $f : [0, T] \mapsto \mathbb{R}$ si dice cadlag se è continua a destra ed ammette limite a sinistra, ovvero $\forall t \in [0, T]$ i limiti

$$f(t^-) = \lim_{s\to t^-} f(s) \qquad \text{e} \qquad f(t^+) = \lim_{s\to t^+} f(s)$$

esistono e $f(t) = f(t^+)$.

Una funzione continua è cadlag ma una funzione cadlag può essere discontinua. Se t è un punto di discontinuità si indica con $\Delta f(t) = f(t) - f(t^-)$ il salto di f in t. Un'osservazione utile nel seguito è che una funzione cadlag può avere al massimo un'infinità numerabile di discontinuità, cioé l'insieme

$$\{t \in [0, T] : f(t) \neq f(t^-)\}$$

è finito o numerabile. Inoltre se, fissato un $\epsilon > 0$, si dividono i salti in base alla loro ampiezza, ovvero piccoli salti se $|\Delta f(t)| < \epsilon$, grandi salti altrimenti, si ha che il numero dei grandi salti deve essere finito.

Definizione 1.3.5 (Processo di Lévy)**.** Un processo stocastico cadlag $(X_t)_{t\geq 0}$ su $(\Omega, \mathcal{F}, \mathbb{P})$ tale che $X_0 = 0$ è detto processo di Lévy se:

1. Ha incrementi indipendenti: per ogni successione crescente di istanti temporali $t_0 \dots t_n$, le variabili aleatorie $X_{t_0}, X_{t_1} - X_{t_0}, \dots, X_{t_n} - X_{t_{n-1}}$ sono tra loro indipendenti.
2. Ha incrementi stazionari: la legge di $X_{t+h} - X_t$ non dipende da t.
3. Soddisfa la continuità stocastica: $\forall \epsilon > 0$, $\lim_{h\to 0} \mathbb{P}(|X_{t+h} - X_t| \geq \epsilon) = 0$.

La terza condizione implica che per un dato istante t la probabilità che si verifichi un salto al tempo t è zero, cioè le discontinuità avvengono a tempi casuali non essendoci masse di probabilità finite in alcun punto.

Definizione 1.3.6 (Infinita divisibilità)**.** Una distribuzione di probabilità G è detta infinitamente divisibile se esistono $n \geq 2$ variabili aleatorie i.i.d. $Z_1, \ldots, Z_n$ tali che $\sum_{i=1}^{n} Z_i \sim G$.

La legge dei processi di Lévy è infinitamente divisibile; per verificarlo basta osservare che posto $t = n\Delta$ si può scrivere

$$X_{n\Delta} = \sum_{i=1}^{n} \underbrace{X_{i\Delta} - X_{(i-1)\Delta}}_{Z_i}$$

dove Z_i sono gli incrementi del processo di Lévy, i quali sono i.i.d. per definizione.

Si può dimostrare che i processi di Lévy ammettono un esponente caratteristico, è possibile, cioé, scrivere la loro funzione caratteristica come

$$\Phi(z) = e^{t\Psi(z)}.$$

Inoltre per i processi che verranno introdotti in seguito $\Psi(z)$ è noto analiticamente, quindi è possibile applicare i metodi che si basano sulla conoscenza di $\Phi(z)$, come quelli basati sulla FFT che verranno introdotti nei capitoli successivi. In generale, invece, non esiste una formula chiusa per la distribuzione di un Lévy, salvo alcuni casi particolari.

Compound Poisson

Aggiungendo ad un processo di Poisson l'informazione relativa all'ampiezza dei salti si ottiene un processo detto compound Poisson.

Definizione 1.3.7 (Compound Poisson)**.** Un compound Poisson composto con intensità $\lambda > 0$ e distribuzione delle ampiezze dei salti f è un processo stocastico X_t della forma

$$X_t = \sum_{i=1}^{N_t} Y_i$$

dove le ampizze dei salti Y_i sono i.i.d. con distribuzione f e N_t è un processo di Poisson con intensità λ, indipendente da $(Y_i)_{i\geq 1}$.

Lo stesso processo di Poisson può essere visto come un compound Poisson tale che $Y_i \equiv 1$.

Vale la seguente

Proposizione 1.3.1. *$(X_t)_{t\geq 0}$ è un processo compound Poisson $\iff$ è un processo di Lévy con traiettorie costanti a tratti.*

In Figura 1.4 è riportata la simulazione numerica di una traiettoria del processo. Si è scelto il parametro $\lambda = 2$, $f \sim N(0, 0.2)$ e l'orizzonte temporale è $T = 10$.

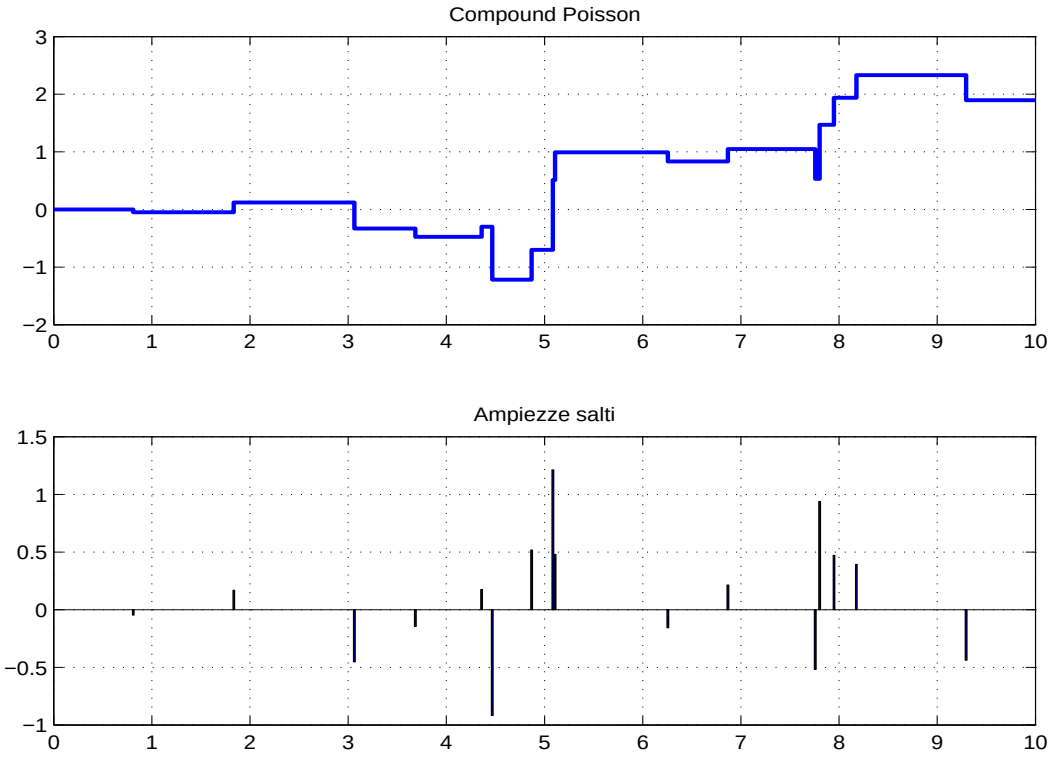


Figura 1.4: Simulazione traiettoria di Poisson non omogeneo

1.3.3 Misure di salto

Considerando un compound Poisson con distribuzione della jump-size f e intensità dei salti λ, è possibile introdurre la misura $\nu(A) = \lambda f(A)$ per ogni $A \subseteq \mathbb{R}$. Tale misura, detta misura di Lévy, non è una misura di probabilità in quanto $\int \nu(dx) = \lambda \int f(dx) = \lambda$ che è diverso da 1 in generale.

Definizione 1.3.8 (Misura aleatoria)**.** Sia $(\Omega, \mathcal{F}, \mathbb{P})$ uno spazio di probabilità. Una misura aleatoria su $(E, \mathcal{E})$ è una mappa $M : \Omega \times E \mapsto \mathbb{R}^+$ tale che:

- $\forall A \in \mathcal{E}$ la misura $\omega \mapsto M(\omega, A)$ è una variabile aleatoria.
- $A \mapsto M(\omega, A)$ è q.c. una misura su $\mathcal{E}$.

Ad ogni processo cadlag può essere associata una misura aleatoria su $\mathbb{R} \times [0, \infty)$ che descrive i salti di X, per ogni insieme misurabile $B \subset \mathbb{R} \times [0, \infty)$

$$J_X(B) = \#\{(t, X_t - X_{t^-}) \in B\}$$

Se $B = [t_1, t_2] \times A$ tale misura conta il numero di salti aventi ampiezze appartenenti ad A che avvengono tra t_1 e t_2.

Nel caso dei compound Poisson con intensità λ e distribuzione delle jump-size f si può dimostare che J_X ha intensità $\mu(dx \times dt) = \nu(dx)dt = \lambda f(dx)dt$. Questo fatto suggerisce di interpretare la misura di Lévy ν di un compound Poisson come il numero medio di salti nell'unità di tempo. Tale interpretazione è molto generale e

consente di definire la misura di Lévy per ogni processo di Lévy, e non solo per i compound Poisson, nel seguente modo

Definizione 1.3.9 (Misura di Lévy)**.** Sia $(X_t)_{t\geq 0}$ un processo di Lévy su $\mathbb{R}$. La misura

$$\nu(A) = \mathbb{E}[\#\{t \in [0,1] : \Delta X_t \neq 0, \Delta X_t \in A\}],\ A \in \mathbb{B}(\mathbb{R})$$

è detta misura di Lévy di X.

Tale definizione consente di scrivere i compound Poisson nel modo seguente

$$X_t = \sum_{s\in[0,t]} \Delta X_s = \int_{[0,t]\times\mathbb{R}} x J_x(ds \times dx) \tag{1.10}$$

Il processo è stato riscritto come somma dei suoi salti, poiché per un compound Poisson il numero di salti è q.c. finito in ogni intervallo limitato l'integrale in (1.10) è ben definito.

1.3.4 Costruzione di processi di Lévy

I modelli finanziari con salti cadono in due categorie distinte. Nella prima, detta "jump-diffusion", la normale evoluzione del prezzo è data dalla componente diffusiva, interrotta da salti che rappresentano eventi rari. La seconda categoria, detta ad attività infinita, consiste in modelli con un numero infinito di salti in ogni intervallo temporale. In questi modelli non è necessario introdurre una componente Browiniana dato che la dinamica dei salti è di per sè sufficientemente ricca.

Definizione 1.3.10 (Misura di Radon)**.** Sia $E \in \mathbb{R}$, una misura di Radon su $(E, \mathcal{B}(E))$ è una misura μ tale che per ogni insieme compatto misurabile $A \in \mathcal{B}(E)$ si ha che $\mu(A) < \infty$.

Teorema 1.3.1 (Decomposizione di Lévy-Ito)**.** *Sia X_t un processo di Lévy e ν la sua misura di Lévy* $\implies$

1. *ν è una misura di Radon su $\mathbb{R}\setminus\{0\}$.*
2. *$\int_{|x|\geq 1} \nu(dx) < +\infty$*
3. *$\int_{|x|\leq 1} |x|^2 \nu(dx) < +\infty$*
4. *$X_t = X_t^c + X_t^l + \lim_{\epsilon\to 0} \tilde{X}_t^\epsilon$*

Dove

$$\begin{aligned} X_t^c &= \gamma t + \sigma W_t \\ X_t^l &= \int_{|x|\geq 1,\, s\in[0,t]} x J_x(ds \times dx) \\ \tilde{X}_t^\epsilon &= \int_{\epsilon\leq|x|\leq 1,\, s\in[0,t]} x\left[J_x(ds \times dx) - \nu(dx)ds\right] \end{aligned}$$

In questa decomposizione X_t^c rappresenta la parte continua del processo, X_t^l rappresenta la componente dei grandi salti e $\tilde{X}_t^\epsilon$ quella dei piccoli salti, convenzionalmente individuati come quelli di ampiezza minore di 1. La misura $\nu(B)$ è finita per ogni insieme compatto tale che $0 \notin B$, se non fosse così il processo avrebbe infiniti salti di ampiezza finita, ma questo contraddirebbe l'ipotesi cadlag. Non è però necessario che la misura sia finita, il processo X_t potrebbe comunque avere infiniti salti di ampiezza infinitesima.

Per definire univocamente un processo di Lévy si può specificare l'esponente caratteristico $\Psi_X(z)$ oppure la tripletta caratteristica (γ, σ, ν). Deve quindi esserci un legame tra le due rappresentazioni, questo è fornito dal seguente

Teorema 1.3.2 (Formula di Lévy-Khintchine)**.** *Sia X_t un processo di Lévy con tripletta caratteristica (γ, σ, ν)* $\implies$

$$\Psi_X(z) = -\frac{1}{2}\sigma^2 z^2 + \mathrm{i}\,\gamma z + \int_{\mathbb{R}} \left(e^{\mathrm{i}\,zx} - 1 - \mathrm{i}\,zx\mathbb{I}_{|x|\leq 1}\right)\nu(dx)$$

Modelli Jump-diffusion Un processo di Lévy del tipo Jump-diffusion ha la forma seguente

$$X_t = \gamma t + \sigma W_t + \sum_{i=1}^{N_t} Y_i \tag{1.11}$$

Dove $(N_t)_{t\geq 0}$ è un processo di conteggio di Poisson e Y_i sono le jump sizes. Il modello è completamente specificato una volta che viene assegnata la distribuzione della jump sizes $f(x)$. In questa categoria i modelli più utilizzati sono quello di Merton e Kou

Subordinazione browniana Una tecnica per la costruzione di processi di Lévy consiste nella subordinazione del modo Browniano con drift $X_t = \theta t + \sigma W_t$ tramite un subordinatore $(Z_t)_{t\geq 0}$. Il processo risultante $Y_t = X_{Z_t} = \theta Z_t + \sigma W_{Z_t}$ è un moto Browniano se viene osservato nella nuova scala temporale stocastica data da Z_t. L'interpretazione finanziaria è quella di *business time*, ovvero il tasso implicito di accumulo dell'informazione. Il subordinatore scelto, $(Z_t)_t$, è un processo α-stabile, con densità

$$\rho(x) = \frac{ce^{-\lambda x}}{x^{1+\alpha}}\mathcal{I}_{x>0}, \tag{1.12}$$

si tratta di un processo di Lévy a tre parametri con salti soltanto positivi, dove $c > 0$, $\lambda > 0$ e $1 > \alpha \geq 0$. Il parametro c controlla la scala del processo, ovvero varia l'intensità dei salti di ogni ampiezza simultaneamente, λ seleziona il tasso di decadimento dei grandi salti e α determina il peso relativo dei piccoli salti nelle traiettorie del processo.

Per comodità il processo viene riparametrizzato come

$$\rho(x) = \frac{1}{\Gamma(1-\alpha)}\left(\frac{1-\alpha}{\kappa}\right)^{1-\alpha}\frac{e^{-(1-\alpha)x/\kappa}}{x^{1+\alpha}} \tag{1.13}$$

Il parametro κ rappresenta la varianza del subordinatore al tempo 1 e di fatto è una misura della casualità del cambio di scala temporale, il caso $\kappa = 0$ corrisponde ad una funzione deterministica.

Effettuando un cambio di scala temporale su di un Moto Browniano (di volatilità σ e drift θ) si ottiene un processo *normal tempered stable*.

In generale il suo esponente caratteristico è del tipo

$$\Psi(u) = \frac{1-\alpha}{\kappa\alpha}\left\{1 - \left(1 + \frac{\kappa(u^2\sigma^2/2 - \mathrm{i}\,\theta u)}{1-\alpha}\right)^2\right\} \tag{1.14}$$

Poiché la densità di probabilità di un subordinatore α-stabile è nota in forma chiusa per $\alpha = \frac{1}{2}$ e $\alpha = 0$, i processi corrispondenti, NIG e VG, sono matematicamente più trattabili e più semplici da simulare, pertanto sono stati ampiamente utilizzati in letteratura.

Adesso che sono state descritte in generale le proprietà dei processi di Lévy, vengono presentati i processi di Lévy più comunemente utilizzati in letteratura.

1.3.5 Merton

Il modello Nel modello ad attività finita di Merton i salti nel log price X_t hanno una distribuzione Gaussiana, $Y_i \sim N(\mu, \delta^2)$. I parametri che entrano in gioco sono dunque 4

1. σ volatilità della parte diffusiva
2. λ intensità dei salti
3. μ media della jump size
4. δ deviazione standard della jump size

Esponente caratteristico L'esponente caratteristico del processo X_t è pari a:

$$\Psi_t(u) = -\frac{\sigma^2 u^2}{2} + \lambda\left(e^{-\frac{\delta^2 u^2}{2} + \mathrm{i}\,\mu u} - 1\right) \tag{1.15}$$

Esempio di traiettorie In Figura 1.5 è riportata la simulazione numerica di una traiettoria del processo. Vengono messi in evidenza gli istanti di salto e le ampiezze dei salti. Si sono scelti i parametri $\mu = 1$, $\sigma = 0.2$, $\lambda = 8$, $\nu = 0.1$, $\delta = 0.1$, l'orizzonte temporale è $T = 1$ mentre la griglia temporale contiene 252 intervalli. Vengono identificati gli istanti di salto con le relative ampiezze, le quali vanno a sommarsi alle oscillazioni dovute alla componente diffusiva.

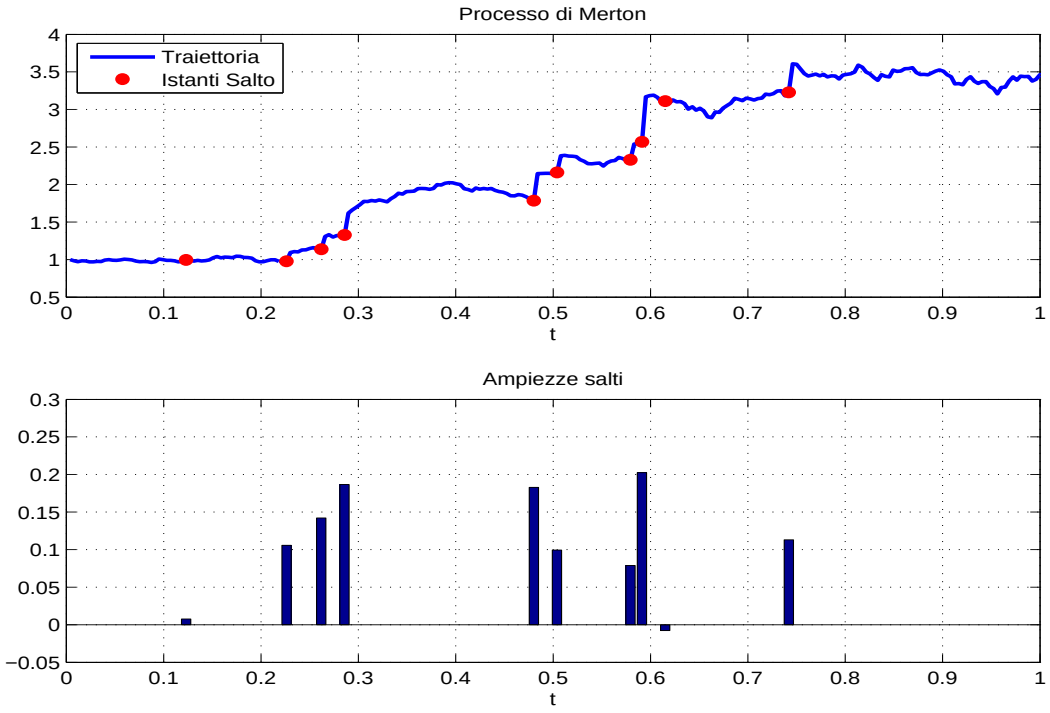


Figura 1.5: Simulazione traiettoria Merton

1.3.6 Kou

Il modello Nel modello ad attività finita di Kou i salti nel log price X_t hanno una distribuzione esponenziale asimmetrica, con una densità della forma:

$$f(dx) = \left[p\lambda_+ e^{-\lambda_+ x}\mathbb{I}_{x>0} + (1-p)\lambda_- e^{-\lambda_- |x|}\mathbb{I}_{x<0}\right] dx,$$

dove $\lambda_+ > 0$ e $\lambda_- > 0$ governano il decadimento delle code nella distribuzione e $p \in [0,1]$ rappresenta la probabilità che il salto sia positivo.

Esponente caratteristico L'esponente caratteristico del processo X_t è pari a:

$$\Psi_t(u) = -\frac{\sigma^2 u^2}{2} + \mathrm{i}\, u\lambda\left(\frac{p}{\lambda_+ - \mathrm{i}\, u} - \frac{1-p}{\lambda_- + \mathrm{i}\, u}\right) \tag{1.16}$$

Esempio di traiettorie In Figura 1.6 è riportata la simulazione numerica di una traiettoria del processo. Vengono messi in evidenza gli istanti di salto e le ampiezze dei salti. Si sono scelti i parametri $\mu = 1$, $\sigma = 0.2$, $\lambda = 8$, $p = 0.5$, $\lambda_+ = 10$, $\lambda_- = 12$, l'orizzonte temporale è $T = 1$ mentre la griglia temporale contiene 252 intervalli.

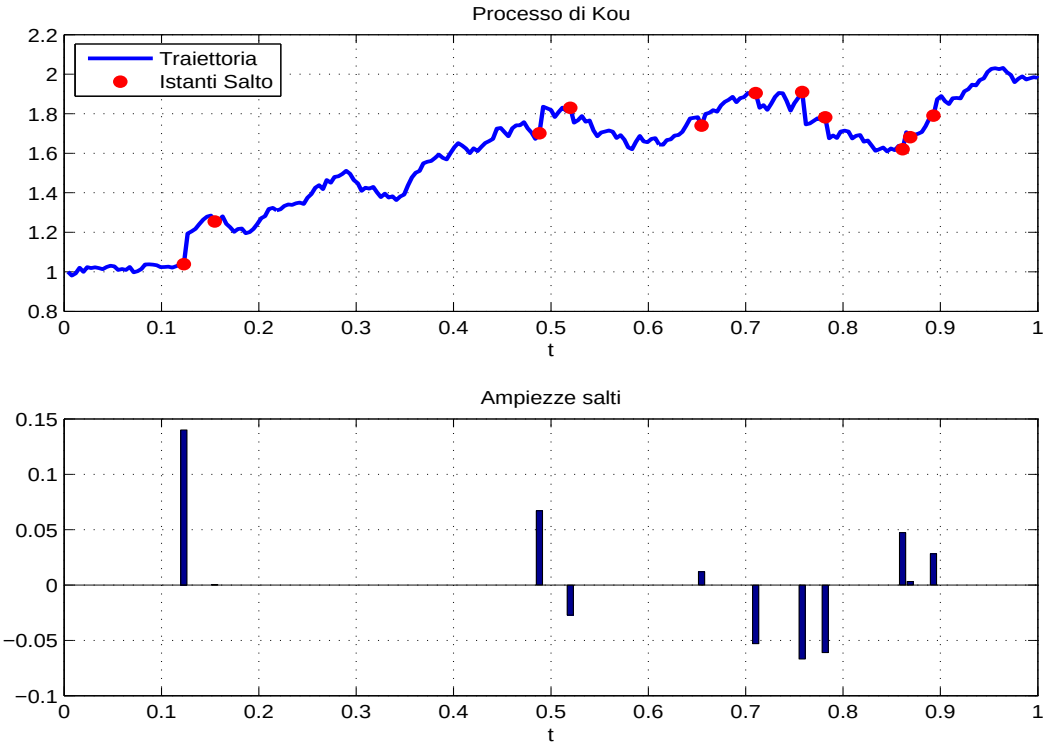


Figura 1.6: Simulazione traiettoria Kou

1.3.7 NIG

Il modello Questo processo ad attività infinita si ottiene subordinando un moto Browniano con un *normal tempered stable process* di parametro $\alpha = 0.5$. Sostituendo nella (1.13) si ottiene

$$\rho(x) = \frac{1}{\kappa}\frac{e^{-x/\kappa}}{x}$$

Esponente caratteristico L'esponente caratteristico del processo X_t è pari a:

$$\Psi_t(u) = \frac{1}{k} - \frac{1}{k}\sqrt{1 + u^2\sigma^2 k - 2\,\mathrm{i}\,\theta u k}. \tag{1.17}$$

Esempio di traiettorie In Figura 1.7 è riportata la simulazione numerica di una traiettoria del processo. Si sono scelti i parametri $\sigma = 0.2$, $k = 0.1$, $\theta = 0.5$, l'orizzonte temporale è $T = 1$ mentre la griglia temporale contiene 252 intervalli.

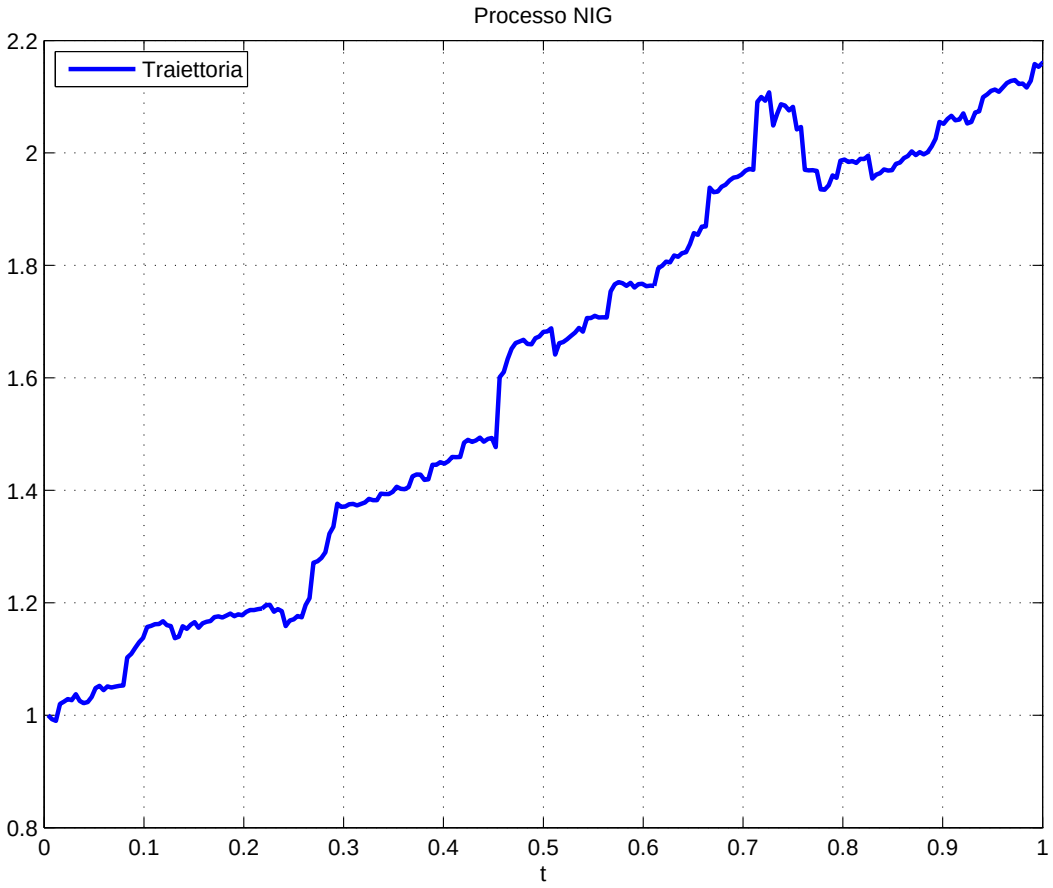


Figura 1.7: Simulazione traiettoria NIG

1.3.8 VG

Il modello Questo processo ad attività infinita si ottiene subordinando un moto Browniano con un *normal tempered stable process* di parametro $\alpha = 0$. Sostituendo nella (1.13) si ottiene

$$\rho(x) = \frac{1}{\sqrt{2\pi\kappa}} \frac{e^{-\frac{x}{2\kappa}}}{x^{3/2}}.$$

Esponente caratteristico L'esponente caratteristico del processo X_t è pari a:

$$\Psi_t(u) = -\frac{1}{k} \log \left\{ 1 + \frac{u^2\sigma^2 k}{2} - \mathrm{i}\,\theta u k \right\} \tag{1.18}$$

Esempio di traiettorie In Figura 1.8 è riportata la simulazione numerica di una traiettoria del processo. Si sono scelti i parametri $\sigma = 0.2$, $k = 0.1$, $\theta = 0.5$, l'orizzonte temporale è $T = 1$ mentre la griglia temporale contiene 252 intervalli.

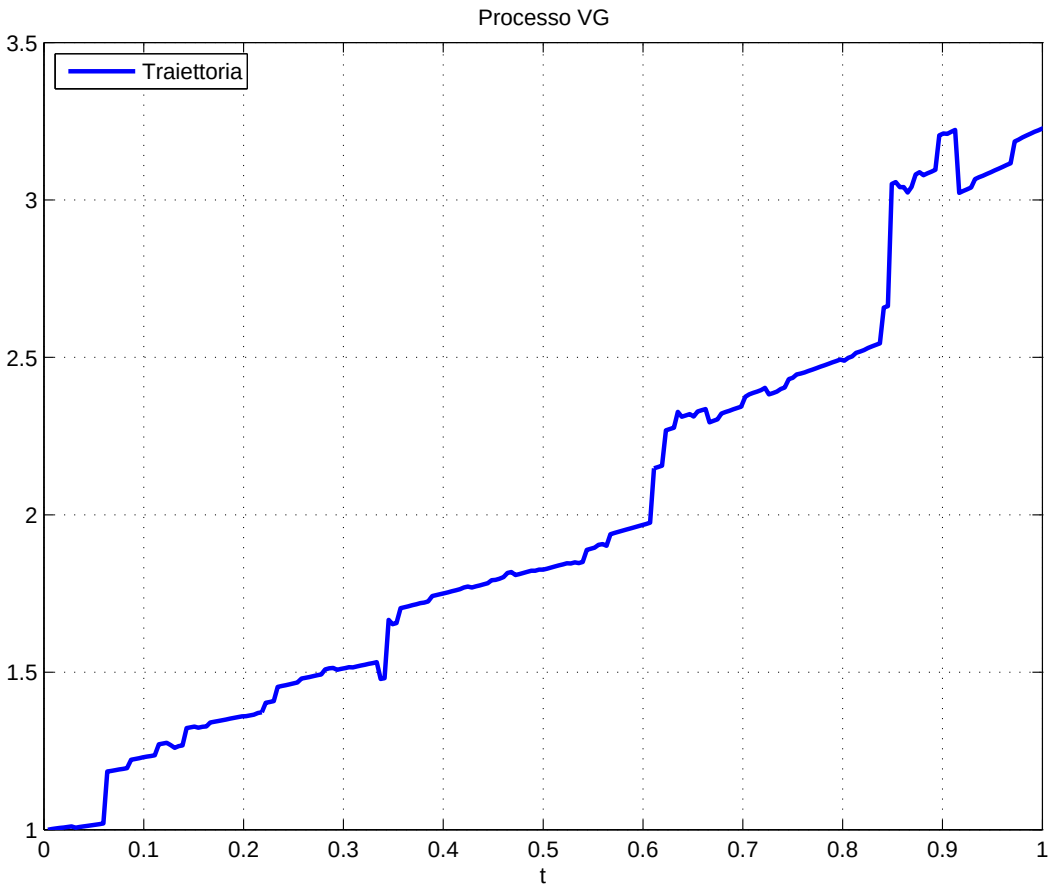


Figura 1.8: Simulazione traiettoria VG

1.4 Modelli a volatilità stocastica

Un'altra classe di modelli introdotti per rispondere alle carenza del modello di Black&Scholes è quella dei modelli a volatilità stocastica.

Questi rappresentano una diversa via nell'evoluzione dei modelli di pricing rispetto all'introduzione di salti. La volatilità del sottostante non è più costante, ma essa stessa ha un andamento stocastico.

La principale differenza rispetto ai modelli con salti sta nella capacità di riprodurre gli smile osservati sul mercato: infatti mentre i processi di Lévy riescono a gestire le brevi scadenze, quelli a volatilità stocastica performano meglio per le lunghe scadenze.

1.4.1 Heston

Il modello Il modello di Heston è dato dal seguente insieme di equazioni differenziali stocastiche

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{V_t} S_t dW_1(t) \\ dV_t &= \kappa(\theta - V_t)dt + \nu\sqrt{V_t} dW_2(t) \\ S(0) &= S_0 \\ V(0) &= V_0 \\ &< dW_1,\, dW_2 >= \rho dt \end{aligned}$$

dove il parametro κ rappresenta la velocità di mean reversion della varianza, θ è la varianza di lungo periodo, ρ è la correlazione tra i moti browniani W_1 e W_2 che guidano i processi, ν è la vol-of-vol[4], S_0 è il prezzo spot e V_0 la varianza spot.

La dinamica neutrale rispetto al rischio del logaritmo del prezzo $X_t = \ln \frac{S_t}{S_0}$ è

$$dX_t = \left(r - \frac{1}{2}V_t\right) dt + \sqrt{v_t} dW_1(t)$$

Funzione caratteristica La funzione caratteristica del processo X_t è nota in forma chiusa ed è pari a:

$$\Psi_t(u, T) = e^{C(T-t,u)+D(T-t,u)v+\mathrm{i}\,ux} \tag{1.19}$$

[4]Volatilità della volatilità

dove $v = V_t$ è il valore al tempo t della volatilità e $x = X_t$ è il valore al tempo t del processo e

$$C(t,u) = \frac{\kappa\theta}{\nu^2}\left\{(\kappa - \rho\nu u\,\mathrm{i} - D)(T-t) - 2\ln\left[\frac{Ge^{-D(T-t)}}{G-1} - 1\right]\right\}$$
$$D(t,u) = \frac{\kappa - \rho\nu u\,\mathrm{i} - D}{\nu^2}\left[\frac{1-e^{-D(T-t)}}{1-Ge^{-D(T-t)}}\right]$$
$$G = \frac{\kappa - \rho\nu u\,\mathrm{i} - D}{\kappa - \rho\nu u\,\mathrm{i} + D}$$
$$D = \sqrt{(\kappa - \rho\nu u\,\mathrm{i})^2 + u(\mathrm{i}+u)\nu^2}$$

Esempio di traiettorie In Figura 1.9 è riportata la simulazione numerica di una traiettoria del processo. Si sono scelti i parametri $\rho = -0.4$, $\kappa = 0.6$, $\theta = 0.4$, $V_0 = 0.4$, $\nu = 0.4$, l'orizzonte temporale è $T = 1$ mentre la griglia temporale contiene 252 intervalli.

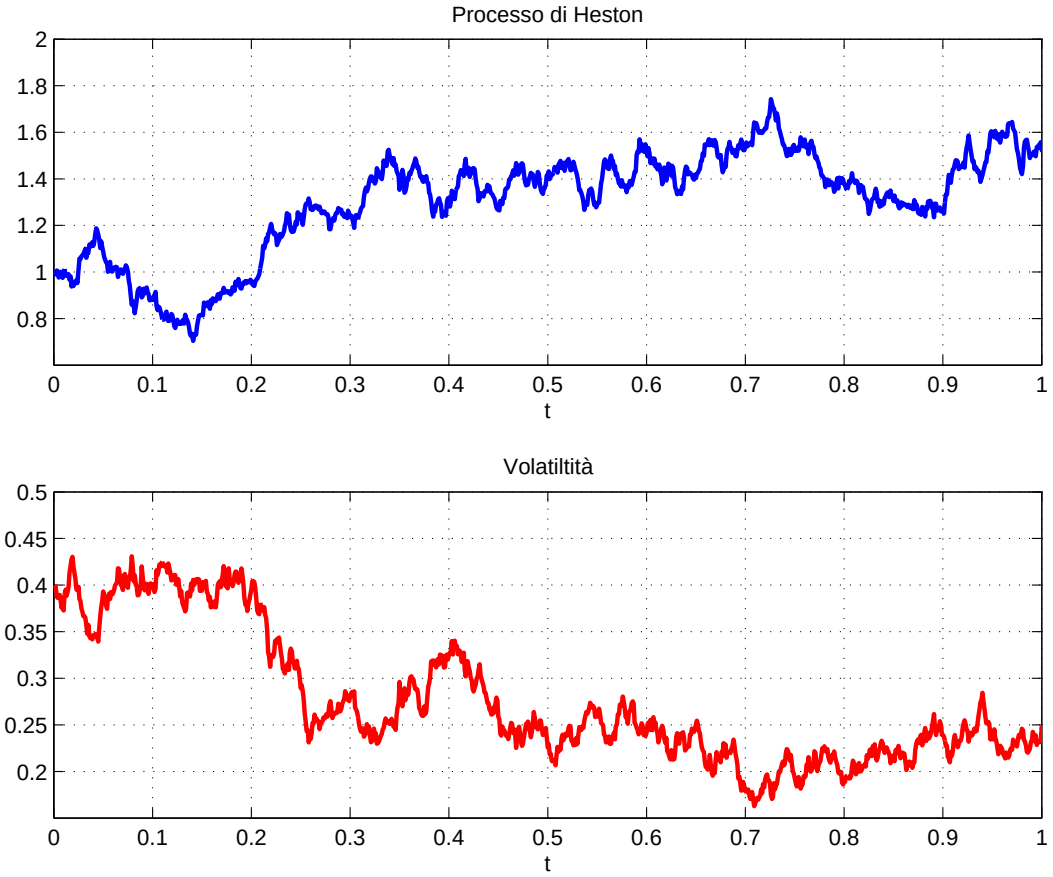


Figura 1.9: Simulazione traiettoria Heston

1.5 Risk Neutral Pricing

Finora si è parlato di modelli per il sottostante, tuttavia per arrivare realmente a fare pricing occorre un ultimo passaggio: si deve introdurre il concetto di assenza di arbitraggio. Verranno quindi presentati i pricipali risultati in questo ambito, quali i due teoremi fondamentali dell'asset pricing e verrà ricavata la condizione di non arbitraggio per i processi di Lévy. Per una trattazione esaustiva si può fare riferimento a [3].

Si consideri un modello di mercato (B, S) nel quale sono presenti due asset:

- Un asset rischioso, il cui prezzo è descritto dal processo stocastico S_t.
- Un asset non rischioso, con drift pari al tasso risk free: B_t t.c. $dB_t = rB_t dt$ e $B_0 = 1$. Tale asset rappresenta l'andamento del risparmio detenuto in un conto bancario che capitalizza al tasso r.

Definizione 1.5.1 (Misura di martingala equivalente)**.** Una misura di probabilità $\mathbb{Q}$ su $\mathcal{F}$ è detta misura di martingala equivalente per il modello (B, S) se è equivalente alla misura $\mathbb{P}$ e il processo $\hat{S}_t = \frac{S_t}{B_t}$ è una martingala sotto la misura $\mathbb{Q}$.

Definizione 1.5.2 (Processo valore di un portafoglio)**.** Dato un portafoglio, ovvero un'allocazione della ricchezza nelle due asset class disponibili data da un processo $\xi = (\xi^B, \xi^S)$, il suo processo di valore si scrive

$$V_t^\xi = \xi_t^B B_t + \xi_t^S S_t$$

Definizione 1.5.3 (Arbitraggio)**.** Un portafoglio $\xi = (\xi^B, \xi^S)$ autofinanziante è detto arbitraggio se il processo di valore V^ξ verifica:

- $V_0^\xi = 0$
- $\mathbb{P}(V_T^\xi \geq 0) = 1$
- $\mathbb{P}(V_T^\xi > 0) > 0$

Un arbitraggio, o *free lunch*, è quindi un portafoglio che non richiede alcun esborso iniziale e che, senza alcun rischio di realizzare delle perdite, offre possibilità di guadagno. Un modello di mercato si dice privo di abitraggio se non esistono portafogli che costituiscono arbitraggi. Sebbene nella realtà dei mercati possano verificarsi situazioni di mispricing tali da portate a dei free lunch, la letteratura accademica è concorde nel considerare modelli di mercato privi di arbitraggio. Questo avviene poiché tali occasioni sono immediatamente sfruttate dagli operatori del mercato e, come conseguenza della legge della domanda e dell'offerta, il prezzo si riporta a valori tali da escludere ogni arbitraggio.

Teorema 1.5.1 (Primo teorema fondamentale dell'asset pricing)**.** *Esiste una misura di martingala $\mathbb{Q}$ per il modello (B, S) $\iff$ tale modello soddisfa la condizione NFLVR (No Free Lunch With Vanishing Risk)*

La condizione NFLVR generalizza l'assenza di arbitraggio, in termini pratici è quasi un arbitraggio nel senso che è possibile individuare una successione di strategie che approssimano quella di arbitraggio e per le quali il rischio tende a 0. L'importanza del teorema risiede nel fatto che l'esistenza di $\mathbb{Q}$ permette di ottenere una formula di pricing neutrale rispetto al rischio.

Teorema 1.5.2 (Formula di pricing neutrale rispetto al rischio)**.** *Dato il modello di mercato (B, S) ed un payoff H_T si ha che il processo di prezzo $V_H(t)$ che soddisfa l'assenza di arbitraggio è dato da*

$$V_H(t) = e^{-r(T-t)}\,\mathbb{E}^{\mathbb{Q}}[H_T|\mathcal{F}_t]. \tag{1.20}$$

Si noti che $\mathbb{E}^{\mathbb{Q}}[\cdot|\mathcal{F}_t]$ è l'operatore di valore atteso rispetto alla misura $\mathbb{Q}$, condizionato alla filtrazione $\mathcal{F}_t$. Questa formula è di fondamentale importanza e sarà utilizzata in molti dei metodi di pricing presentati nel Capitolo 2.

Un'ulteriore nozione è quella di completezza del mercato. Un modello di mercato si dice completo se per ogni payoff H esiste un portafoglio ξ che lo replica.

Teorema 1.5.3 (Secondo teorema fondamentale dell'asset pricing)**.** *Il modello di mercato (B, S) è completo $\iff$ esiste un'unica misura di martingala equivalente $\mathbb{Q}$.*

I modelli precedentemente descritti, a parte il GBM, corrispondono a mercati incompleti: l'hedging perfetto non esiste e i prezzi delle opzioni non possono essere identificati univocamente tramite l'utilizzo di sole argomentazioni di non arbitraggio. Non essendoci un'unica misura di martingala il prezzo di un derivato non è univoco, vi è bensì una forchetta di possibili prezzi, i quali però soddisfano tutti la condizione di non arbitraggio. Tuttavia è noto dalla (1.20) che in ogni mercato privo di arbitraggio è possibile rappresentare i prezzi dei derivati come valori attesi condizionati e scontati rispetto ad una misura di martingala $\mathbb{Q}$. In mercati incompleti $\mathbb{Q}$ ha soltanto una debole relazione con il comportamento delle serie storiche sotto la probabilità reale $\mathbb{P}$. Essa eredita solo alcune proprietà qualitative, come la presenza di salti, l'attività finita o infinita, la variazione finita o infinita, dai prezzi storici del processo.

Si pone quindi il problema di determinare una misura $\mathbb{Q}$ in modo che il processo di prezzo scontato $(e^{-rt}S_t)$ sia una martingala. In particolare, per i modelli di Lévy, vale il seguente:

Teorema 1.5.4. *Sia $(X_t)_{t\geq 0}$ un processo di Lévy con tripletta (γ, σ^2, ν) tale che*

$$\int_{|y|\geq 1} e^y \nu(dy) < +\infty.$$

Il processo e^{X_t} è una martingala $\iff$

$$\Psi(-\mathrm{i}) = \gamma + \frac{\sigma^2}{2} + \int_{-\infty}^{+\infty} \left(e^z - 1 - z\mathbb{I}_{|z|<1}\right) = 0.$$

Per una dimostrazione si faccia riferimento a [5]. Per quanto riguarda Heston, la dinamica neutrale rispetto al rischio è già stata riportata nella sezione 1.4.1.

Per i sottostanti per cui le opzioni sono scambiate su un mercato organizzato, i prezzi quotati forniscono un'ulteriore fonte di informazione nella scelta del modello adeguato. La selezione di una misura di martingala $\mathbb{Q}$ in grado di riprodurre i prezzi delle opzioni è nota come calibrazione del modello. Sorge quindi naturale l'approccio "implicito", ovvero modellizzare direttamente la dinamica risk neutral del prezzo dell'asset scegliendo una misura $\mathbb{Q}$ che rispetti le proprietà qualitative del processo. Nel Capitolo 3 verrà introdotto il problema della calibrazione e verranno confrontate le prestazioni dei diversi modelli per il sottostante e i vari metodi di pricing.

Capitolo 2

Metodi di pricing

In questo capitolo vengono presentati i metodi che saranno utilizzati nel software finale.

I metodi che vengono descritti sono l'albero Binomiale, adatto solo al modello di Black&Scholes, ma particolarmente efficiente, il Monte Carlo e tre metodi basati sulle trasformate di Fourier. Questi ultimi metodi sono veloci e precisi, qualità che li rendono ideali per la calibrazione.

In base a quanto osservato precedentemente, un metodo di pricing adatto per la calibrazione deve essere rapido ed affidabile poiché è necessario valutare i prezzi numerose volte durante il processo di ottimizzazione. Questo esclude il metodo di Monte Carlo, adattabile a svariati tipi di payoff e di semplice implementazione ma dall'elevato costo computazionale. Inoltre produce prezzi aleatori e l'ampiezza dell'intervallo di confidenza tende a zero solo come $O(\sqrt{n})$. Il Monte Carlo sarà quindi utilizzato nella sola sezione di pricing del software finale. Anche gli approcci PIDE vengono esclusi poiché il costo computazionale di un singolo pricing è di per sé molto elevato, moltiplicato per una forchetta di strike e per il numero minimo di valutazioni richieste a trovare il minimo dell'errore produce un tempo inaccettabile.

In particolare, tra i metodi basati sulla trasformata di Fourier vengono esaminati i seguenti:

- Carr&Madan, [4]
- CONV, [11]
- COS, [6, 7]

Il metodo Carr&Madan è il meno efficiente e manca della flessibilità necessaria per adattarsi al princig di strumenti più esotici delle opzioni europee. Verrà quindi utilizzato come confronto per gli altri due metodi, i quali invece hanno costi computazionali minori e contemplano payoff più generali. In particolare si vedrà nella Sezione 3.2 che il metodo COS è il più efficiente e preciso tra i metodi proposti.

2.1 Albero

2.1.1 Il modello

L'albero binomiale di Cox-Ross-Rubinstein è il più semplice modello della finanza matematica. Si tratta di un modello a tempo discreto che è spesso utilizzato nella pratica finanziaria. Nella sua forma base, quella utilizzata in questa tesi, è applicabile al solo modello di Black&Scholes, tuttavia presenta, in virtù della sua forte specializzazione, un'efficienza superiore a quella degli altri metodi più generali. Per la trattazione si fa riferimento a [3].

Nel modello sono presenti due asset, un bond risk-free e un'azione. L'evoluzione temporale del bond è deterministica e segue una legge di capitalizzazione composta:

$$\begin{cases} B_{n+1} = (1+r)B_n \\ B_0 = 1; \end{cases}$$

dove r è il tasso risk-free.

Al contrario l'evoluzione dell'azione è stocastica

$$\begin{cases} S_{n+1} = S_n Z_n \\ S_0 = s; \end{cases}$$

le variabili aleatorie $Z_0, \dots, Z_{T-1}$ sono assunte i.i.d e possono assumere solo due valori

$$\begin{cases} \mathbb{P}(Z_n = u) = p_u \\ \mathbb{P}(Z_n = d) = p_d. \end{cases}$$

In Figura 2.1 è riportata la dinamica del presso dell'azione prescritta dal modello.

2.1.2 Misura di martingala

Poiché la presenza di arbitraggi viene interpretata come un grave caso di mispricing degli asset si cerca una condizione in grado di escluderne la presenza. A tal proposito si può dimostrare la

Proposizione 2.1.1 (Condizione di non arbitraggio)**.** *Il modello appena introdotto è privo di arbitraggio* $\iff$

$$d \leq 1 + r \leq u. \tag{2.1}$$

Si può osservare che questa condizione è equivalente al richiedere che $1 + r$ sia combinazione convessa di u e d, ovvero che $\exists q \geq 0$ t.c.

$$1 + r = qu + (1 - q)d.$$

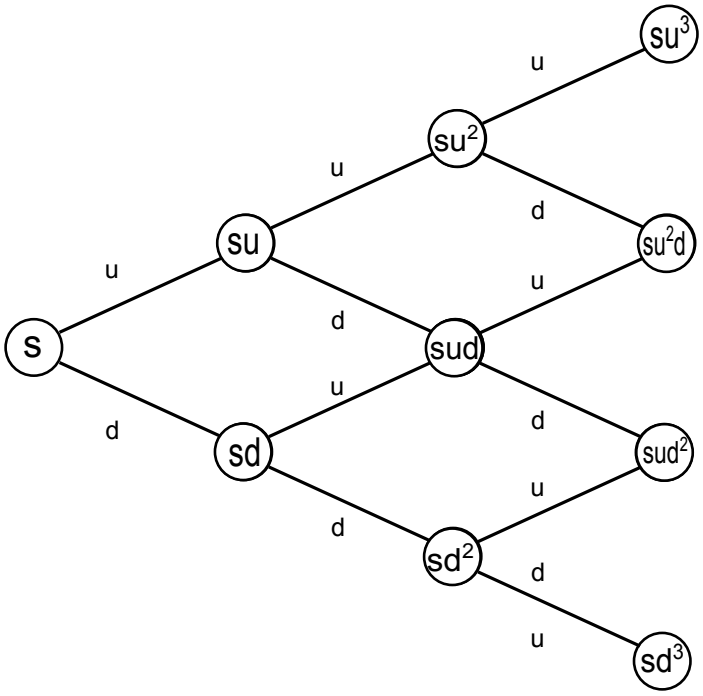


Figura 2.1: Dinamica del prezzo

In particolare i pesi della combinazione q e $1-q$ possono essere interpetati come probabilità sotto una nuova misura di probabilità $\mathbb{Q}$ con la proprietà che

$$\begin{cases} \mathbb{Q}(Z_n = u) = q \\ \mathbb{Q}(Z_n = d) = 1 - q \end{cases}$$

Si può inoltre dimostrare, si veda ad esempio [3], che $\mathbb{Q}$ è una misura di martingala, ovvero vale

$$S_t = \frac{1}{1+r}\,\mathbb{E}^{\mathbb{Q}}[S_{t+1}] \tag{2.2}$$

e che le probabilità di martingala q è pari a

$$q = \frac{(1+r)-d}{u-d}$$

2.1.3 Pricing

Si può dimostrare, [3], che se si sceglie

$$\begin{cases} u = e^{\sigma\Delta t} \\ d = \frac{1}{u} \end{cases}$$

il modello binomiale converge, per $\Delta t \to 0$, al modello di Black&Scholes con volatilità σ. Di conseguenza è possibile fare pricing nell'ambito del processo GBM utilizzando questa struttura.

Si può prezzare un derivato con uno schema ricorsivo: una volta costruito l'albero del sottostante è possibile valutare la funzione payoff $H(S_T, T)$ a scadenza e, procedendo a ritroso, costruire l'albero dei prezzi dell'opzione. Dopo T passi si ottiene il prezzo $V(t, S_t)$ che si sta cercando. Si osserva che è possibile scrivere il prezzo del sottostante al tempo t come

$$S_t = S_0 u^k d^{T-k}$$

dove k è il numero di salti verso l'alto che sono avvenuti. Allora è possibile rappresentare ciascun nodo dell'albero con la coppia (t, k) con $k = 0, \dots, T$.

Proposizione 2.1.2 (Algoritmo di pricing)**.** *Si consideri un'opzione con payoff a scadenza $H = H(S_T, T)$; se $V_t(k)$ denota il prezzo dell'opzione nel nodo (t, k), allora:*

$$\begin{cases} V_t(k) = \frac{1}{1+r}(qV_{t+1}(k+1) + (1-q)V_{t+1}(k)) \\ V_T(k) = H(S_0 u^k d^{T-k}) \end{cases}$$

Si può anche scrivere in forma più compatta direttamente il prezzo al tempo 0 nel seguente modo, che è quello che è stato implementato:

$$V(0, S_0) = \frac{1}{(1+r)^T} \sum_{k=0}^{T} \binom{T}{k} q^k (1-q)^{T-k} H(S_0 u^k d^{T-k}) \tag{2.3}$$

2.2 Monte Carlo

2.2.1 Inquadramento teorico

Il metodo Monte Carlo si basa sulla ripetizione, in maniera casuale, di una simulazione. Nella tesi verrà utilizzato non tanto per la calibrazione, quanto per il pricing, all'interno della GUI sviluppata. Il punto di partenza del metodo è la formula di pricing neutrale rispetto al rischio

$$V(S_0, t_0) = e^{-r\Delta T}\,\mathbb{E}^{\mathbb{Q}}[H(S_T, T)] \tag{2.4}$$

dove si afferma che il valore di un derivato è pari al valore atteso rispetto alla misura di martingala $\mathbb{Q}$ del payoff scontato.

L'idea è che per calcolare l'integrale nella (2.4) si può utilizzare l'integrazione Monte Carlo.

La legge dei grandi numeri suggerisce di simulare delle variabili x_i i.i.d. e di utilizzare

$$\widehat{\theta}_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

come stimatore di $\mathbb{E}(f)$. Per una dimostrazione dei seguenti due teoremi si faccia riferimento a [9].

Teorema 2.2.1 (Legge dei grandi numeri)**.** *Siano $(X_i)_{i\geq 0}$ variabili i.i.d. di media μ e sia*

$$Y_n = \frac{1}{n}\sum_{i=1}^{n} X_i$$

la media dei campionamenti $\implies$

$$\mathbb{P}(|Y_n - \mu| > \epsilon) \underset{n\to+\infty}{\longrightarrow} 0 \quad \forall \epsilon > 0$$

L'approssimazione è tanto migliore quanto più il numero di simulazioni N cresce verso infinito. L'errore commesso è caratterizzato dal teorema del limite centrale.

Teorema 2.2.2 (Teorema del limite centrale)**.** *Siano $(X_i)_{i\geq 0}$ variabili i.i.d. di media μ e varianza σ^2* $\implies$

$$\frac{\sum_{i=1}^{n} X_i - n\mu}{\sigma\sqrt{n}} \underset{n\to+\infty}{\xrightarrow{d}} N(0,1)$$

Di conseguenza si osserva che

$$\frac{n\hat{\theta_n} - n\mu}{\sigma\sqrt{n}} \xrightarrow{d} N(0,1) \iff$$

$$\frac{\hat{\theta_n} - \mu}{\sigma\sqrt{\frac{1}{n}}} \xrightarrow{d} N(0,1) \iff$$

$$\hat{\theta_n} - \mu \xrightarrow{d} N\left(0, \frac{\sigma^2}{n}\right)$$

L'errore ha ovviamente media nulla e la qualità dell'approssimazione è data dalla deviazione standard dell'errore che tende a 0 come $O\left(\sqrt{\frac{1}{N}}\right)$. Questo tipo di convergenza è lento, per ridurre l'errore all'ordine ϵ occorrono $O(\epsilon^{-2})$ simulazioni. Tuttavia per un moderno calcolatore l'effettivo tempo computazionale è accettabile e l'adattabilità del metodo ai payoff più vari, assieme alla relativa semplicita di implementazione, ne hanno decretato il successo.

La risposta che questo metodo fornisce al problema di pricing non è un singolo valore ma un intervallo di confidenza: in tal modo si può essere certi dell'accuratezza del prezzo.

Un modo alternativo per scrivere la convergenza in probabilità è

$$\mathbb{P}\left(\frac{\hat{\theta_n} - \theta}{\frac{\sigma}{\sqrt{n}}} < x\right) \underset{n\to+\infty}{\longrightarrow} \Phi(x)$$

dove $\Phi(x)$ è la funzione di ripartizione della Normale standard.

Dal fatto che se $n \to +\infty$ si ha che

$$\Phi(x) = \mathbb{P}\left(\theta > \hat{\theta_n} - x\frac{\sigma}{\sqrt{n}}\right)$$

discende la seguente osservazione

Osservazione 2.2.1 (Intervallo di confidenza)**.** Dati $a, b \in \mathbb{R}$ con $0 \leq a$ e $b \leq +\infty$

$$\mathbb{P}\left(\theta \in \left\{\hat{\theta_n} - a\frac{\sigma}{\sqrt{n}}, \hat{\theta_n} + b\frac{\sigma}{\sqrt{n}}\right\}\right)$$

vale $\Phi(b) - \Phi(a)$ quando $n \to +\infty$.

Di conseguenza se si vuole un livello di confidenza $1 - \alpha$ basterà porre $a = b = z_{\frac{\alpha}{2}}$, dove $\Phi\left(\frac{\alpha}{2}\right) = 1 - \frac{\alpha}{2}$. Infatti:

$$\begin{aligned}
\Phi(b) - \Phi(a) &= \Phi(\alpha/2) - \Phi(-\alpha/2) \\
&= \Phi(\alpha/2) - 1 + \Phi(\alpha/2) \\
&= 2\Phi(\alpha/2) - 1 \\
&= 2(1 - \alpha/2) - 1 \\
&= 1 - \alpha
\end{aligned}$$

che è quanto desiderato. In `Matlab` la funzione `normfit` si preoccupa di fornire l'intervallo di confidenza cercato. Si osservi che questa teoria è molto generale e vale sia per i processi di Lévy che per quelli ad attività stocastica.

Un vantaggio del Monte Carlo è che fornisce tutte le traiettorie del processo di prezzo, è quindi possibile prezzare opzioni path-dependent con facilità.

2.2.2 Simulazione dei processi continui

Per poter applicare il metodo di Monte Carlo è necessario saper simulare le traiettorie dei processi stocastici utilizzati. Possono essere applicati diversi approcci

1. Probabilità di transizione
2. Dinamica esatta (soluzione dell'EDS[1]) quando disponibile
3. Approccio completamente numerico

Ci si concentrerà sull'approccio numerico, utilizzando lo schema di Eulero. L'idea è quella di discretizzare l'EDS che definisce il processo stocastico in questione su di una griglia numerica.

Per quanto riguarda il modello di Black&Scholes, dove $S_t = S_0 e^{X_t}$, la cosa è molto semplice

$$dX_t = \mu dt + \sigma dW_t$$

diventa

$$X_{i+1} = X_i + \mu dt + \sigma\sqrt{dt}Z$$

dove la variabile aletoria Z è tale che $Z \sim N(0,1)$.

Per i modelli di Lévy ad attività finita, quali Merton e Kou, occorre dapprima simulare i tempi di salti e poi andare ad aggiungere la componente di salto ove necessaria. Per la simulazione dei tempi di salto si può utilizzare la conditional simulation, come suggerito dalla

Proposizione 2.2.1. *Gli istanti di salto del processo di Poisson, condizionatamente a N_t, sono distribuiti come uniformi su $[0,T]$.*

Basterà quindi simulare dapprima il numero di salti $N_t \sim Poi(\lambda T)$ e poi ottenere i tempi di salto $\tau_i \sim U[0,T]$. I tempi di salto così ottenuti vanno spostati sulla griglia di discretizzazione avendo cura di non introdurre salti all'istante iniziale, poiché per ipotesi il processo al tempo t_0 ha un valore noto. Le ampiezze di salto vengono poi campionate dalla distribuzione f, caratteristica del modello preso in considerazione.

Per quanto riguarda i modelli ad attività infinita occorre simulare esattamente gli incrementi, in [5] si trovano gli algoritmi per i processi NIG e VG utilizzati in questa tesi.

Il procedimento è differente per il modello di Heston, in quel caso la volatilità stessa segue un andamento stocastico, ad ogni passo è quindi necessario simularla per poi utilizzarla per calcolare X al tempo successivo. Per tenere conto della correlazione ρ basta simulare $z_1 \sim N(0,1)$ e $z_2 \sim N(0,1)$ con $z_1 \perp z_2$ e porre poi $z_2 = \rho z_1 + \sqrt{1-\rho^2} z_2$.

[1] Equazione Differenziale Stocastica

2.3 Il metodo di Carr&Madan

2.3.1 Inquadramento teorico

Ricerca di formula chiusa Nell'ambito dei modelli basati sui processi di Lévy si è alla ricerca di una formula "chiusa", nel senso di comparabile alla famosa formula di Black&Scholes valida in caso di dinamica Log-normale. Benché questo risultato non sia raggiungibile, negli approcci basati sulla trasformata di Fourier si è individuato un valido sostituto, altamente efficiente in termini di costo computazionale.

Formula di pricing La formula di valutazione neutrale rispetto al rischio afferma che

$$V(t, S_t) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}[V(T, S_T)|\mathcal{F}_t]$$

dove V denota il valore dell'opzione, r è il tasso di interesse risk-free, t è l'istante di tempo attuale, T è la scadenza dell'opzione. Il valore atteso è calcolato rispetto alla misura $\mathbb{Q}$ neutrale rispetto al rischio.

Specificamente al caso delle opzioni Call Europee, definendo il payoff a scadenza

$$H_T = (S_T - K)^+$$

dove K è lo strike price, è possibile scrivere la formula come

$$C(t, S_t) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}[H_T|\mathcal{F}_t] = \int_{\mathbb{R}^+} e^{-r(T-t)}(x - K)^+ f_{S_T}(x)dx$$

dove C è il prezzo dell'opzione Call e $f(x)$ è la densità del sottostante a scadenza rispetto alla misura $\mathbb{Q}$.

Utilizzo della trasformata Il problema da affrontare è che in generale la $f(x)$ non è nota. In [4] gli autori delineano un approccio basato sulla trasformata di Fourier che ben si presta alla calibrazione poiché con un ridotto costo computazionale consente il calcolo simultaneo dei prezzi di opzioni Europee con differenti strike price. Questo metodo necessita della sola conoscenza della funzione caratteristica del modello, la quale, per i modelli considerati è nota in forma chiusa.

Come prima cosa si esegue la trasformazione in log-strike $k = \log(K)$. Nel seguito inoltre, per semplicità, si porrà $t = 0$. Il valore della Call diventa dunque

$$C(t, S_t, k) = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[\left(e^{(r-d)T+X_T} - e^k\right)^+ |\mathcal{F}_t\right]$$

dove X_T è preso con drift nullo, e quindi il processo $(r - d)T + X_T$ che compare all'esponente è proprio quello che determina la dinamica risk-neutral.

Purtroppo $C(k)$ non può essere trasformata poiché

$$\lim_{k\to-\infty} C(t, S_t, k) = e^{-dT} \lim_{k\to-\infty} \mathbb{E}^{\mathbb{Q}}\left[\left(e^{X_T}\right)^+ |\mathcal{F}_t\right] \neq 0$$

Si introduce allora una trasformazione

$$z(k) = C(k) - \left(e^{-dT} - e^{k-rT}\right)^+.$$

Risulta semplice verificare che

$$\lim_{k\to\pm\infty} z(k) = 0.$$

Inoltre poiché la trasformazione introdotta coinvolge soltanto un blocco deterministico (non dipendente da X_t), una volta noto $z(k)$ è immediato ricondursi a $C(k)$.

A questo punto si calcola la trasformata di Fourier $g_T(v)$ di $z_T(k)$.

$$g_T(v) = \mathcal{F}(z_T)(v) = \int_{\mathbb{R}} e^{\mathrm{i}\,vk} z_T(k)dk.$$

Definendo ρ_T densità risk neutral di X_T e Φ_T funzione caratteristica di X_T, è possibile riscrivere $z_T(k)$ nel seguente modo

$$\begin{aligned}
z_T(k) &= C(k) - \left(e^{-dT} - e^{k-rT}\right)^+ \\
&= \underbrace{e^{-rT}\int_{\mathbb{R}} (e^{(r-d)T+x} - e^k)(\mathbb{I}_{k\le x+(r-d)T})\rho_T(dx)}_{C(k)} - \left(e^{-dT}\underbrace{\int_{\mathbb{R}} e^x \rho_T(dx)}_{\Phi_T(-\mathrm{i})=1} - e^{k-rT}\right)\mathbb{I}_{k\le(r-d)T} \\
&= e^{-rT}\int_{\mathbb{R}} (e^{(r-d)T+x} - e^k)(\mathbb{I}_{k\le x+(r-d)T})\rho_T(dx) - \left(\int_{\mathbb{R}} (e^{x-dT} - e^{k-rT})\mathbb{I}_{k\le(r-d)T}\rho_T(dx)\right) \\
&= e^{-rT}\int_{\mathbb{R}} (e^{(r-d)T+x} - e^k)(\mathbb{I}_{k\le x+(r-d)T} - \mathbb{I}_{k\le(r-d)T})\rho_T(dx).
\end{aligned}$$

Si può a questo punto ricavare l'espressione di $g_T(v)$.

$$\begin{aligned}
g_T(v) &= \int_{\mathbb{R}} e^{\mathrm{i}\,vk} z_T(k)dk \\
&= \int_{\mathbb{R}} e^{\mathrm{i}\,vk} e^{-rT}\int_{\mathbb{R}} (e^{(r-d)T+x} - e^k)(\mathbb{I}_{k\le x+(r-d)T} - \mathbb{I}_{k\le(r-d)T})\rho_T(dx)dk \\
&= e^{-rT}\int_{\mathbb{R}}\int_{(r-d)T}^{(r-d)T+x} e^{\mathrm{i}\,vk}(e^{(r-d)T+x} - e^k)dk\rho_T(dx) \\
&= e^{-rT}\int_{\mathbb{R}} \underbrace{\int_{(r-d)T}^{(r-d)T+x} e^{\mathrm{i}\,vk+(r-d)T+x} - e^{(\mathrm{i}\,v+1)k}dk}_{A(x)}\,\rho_T(dx) \\
&= e^{-rT}\int_{\mathbb{R}} A(x)\rho_T(dx).
\end{aligned}$$

Ora è necessario valutare $A(x)$ ed infine sostituirlo nell'espressione appena ricavata.

$$\begin{aligned}
A(x) &= \left[\frac{e^{\mathrm{i}\,vk+(r-d)T+x}}{\mathrm{i}\,v} - \frac{e^{(\mathrm{i}\,v+1)k}}{\mathrm{i}\,v+1}\right]_{(r-d)T}^{(r-d)T+x} \\
&= \frac{\left(e^{(\mathrm{i}\,v+1)((r-d)T+x)} - e^{(\mathrm{i}\,v+1)(r-d)T+x}\right) + \mathrm{i}\,v\left(-e^{(\mathrm{i}\,v+1)(r-d)T+x} + e^{(\mathrm{i}\,v+1)(r-d)T}\right)}{\mathrm{i}\,v(\mathrm{i}\,v+1)} \\
&= \frac{e^{(r-d)T}\left(e^{\mathrm{i}\,v(r-d)T+\mathrm{i}\,vx+x)} - e^{\mathrm{i}\,v(r-d)T+x}\right)}{\mathrm{i}\,v(\mathrm{i}\,v+1)} + \frac{e^{(r-d)T}\left(-e^{\mathrm{i}\,v(r-d)T+x} + e^{\mathrm{i}\,v(r-d)T}\right)}{\mathrm{i}\,v+1} \\
&= e^{(r-d)T}\left[\frac{e^{\mathrm{i}\,v(r-d)T}(1-e^x)}{\mathrm{i}\,v+1} - \frac{e^{x+\mathrm{i}\,v(r-d)T}}{\mathrm{i}\,v(\mathrm{i}\,v+1)} + \frac{e^{(\mathrm{i}\,v+1)x+\mathrm{i}\,v(r-d)T}}{\mathrm{i}\,v(\mathrm{i}\,v+1)}\right].
\end{aligned}$$

Sostituendo $A(x)$ si ottiene

$$\begin{aligned}
g_T(v) = \Bigg(&\frac{e^{\mathrm{i}\,v(r-d)T}}{\mathrm{i}\,v+1}\underbrace{\int_{\mathbb{R}}(1-e^x)\rho_T(dx)}_{0} \\
&- \frac{e^{\mathrm{i}\,v(r-d)T}}{\mathrm{i}\,v(\mathrm{i}\,v+1)}\underbrace{\int_{\mathbb{R}} e^x\rho_T(dx)}_{=1} \\
&+ \frac{e^{\mathrm{i}\,v(r-d)T}}{\mathrm{i}\,v(\mathrm{i}\,v+1)}\underbrace{\int_{\mathbb{R}} e^{(\mathrm{i}\,v+1)x}\rho_T(dx)}_{\Phi_T(v-\mathrm{i})}\Bigg)e^{-dT}.
\end{aligned}$$

In definitiva la formula da utilizzare è la seguente

$$\boxed{g_T(v) = \frac{e^{-dT}e^{\mathrm{i}\,v(r-d)T}}{\mathrm{i}\,v(\mathrm{i}\,v+1)}(\Phi_T(v-\mathrm{i})-1)}. \tag{2.5}$$

2.3.2 Calcolo numerico della trasformata di Fourier

Inversione numerica A questo punto, nota la trasformata di Fourier $g_T(v)$ del prezzo modificato $z_T(k)$, occorre procedere alla sua inversione. Questo passaggio è realizzato numericamente tramite l'algoritmo FFT con complessità $O(n\log(n))$.

$$z_T(k) = \mathcal{F}^{-1}(g_T(v))(k) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-\mathrm{i}\,vk}g_T(v)dv \tag{2.6}$$

Ma l'integranda è pari e il dominio di integrazione è simmetrico, quindi:

$$z_T(k) = \frac{1}{\pi}\int_0^{+\infty} e^{-\mathrm{i}\,vk}g_T(v)dv.$$

Occorre troncare il dominio di integrazione, si sceglie un parametro di troncamento A sufficientemente elevato, in modo da ridurre l'entità della sorgente di errore.

$$z_T(k) \simeq \frac{1}{\pi}\int_0^{A} e^{-\mathrm{i}\,vk}g_T(v)dv.$$

Griglie di discretizzazione Si introduce una griglia nello spazio di Fourier, cioé su v:

$$\begin{cases} \eta = \dfrac{A}{N} \\ v_j = \eta j \\ j = 0 : N-1 \end{cases}$$

dove N, il numero di intervalli, deve essere scelto pari ad una potenza di 2 perché solo in tal caso la complessità di FFT sarà effettivamente pari a quella specificata.

Si discretizza ora l'integrale sulla griglia appena introdotta: si è scelto di utilizzare il metodo dei trapezi:

$$z_T(k) \simeq \frac{1}{\pi}\eta \sum_{j=0}^{N-1} \omega_j e^{-\mathrm{i}\, v_j k} g_T(v_j)$$

dove i pesi di integrazione ω_j sono tutti pari ad 1 ad eccezione del primo e dell'ultimo che sono pari a 0.5.

Si introduce una griglia apposita nello spazio dei log-strike, cioè su k

$$\begin{cases} \lambda = \dfrac{2\pi}{N\eta} = \dfrac{2\pi}{A} \\ k_l = -\lambda \dfrac{N}{2} + \lambda l \\ l = 0 : N-1 \end{cases}$$

Si noti che la griglia è stata scelta in modo da soddisfare la relazione di Nyquist $\eta\lambda = \frac{2\pi}{N}$: i passi delle griglie non possono essere scelti indipendentemente. Si crea dunque un tradeoff tra l'accuratezza delle due griglie. Nei codici `Matlab` si sono utilizzati dei parametri che rendono l'ampiezza delle due griglie dello stesso ordine di grandezza. Al termine del procedimento si otterrà la funzione z_T valutata nei punti k_l, per ottenere gli strike desiderati si procedere all'interpolazione con spline.

Elaborando l'integrale della (2.6) tenendo conto che la funzione FFT è tale che:

$$F_l = \mathsf{fft}(f_j) = \sum_{j=0}^{N-1} e^{-2\pi\,\mathrm{i}\,\frac{lj}{N}} f_j$$

si ottiene

$$\begin{aligned}
z_T(k_l) &= \frac{1}{\pi}\eta \sum_{j=0}^{N-1} \omega_j e^{-\mathrm{i}\, v_j k_l} g_T(v_j) \\
&= \frac{1}{\pi}\eta \sum_{j=0}^{N-1} \omega_j e^{-\mathrm{i}\, \eta j(-\lambda\frac{N}{2}+\lambda l)} g_T(v_j) \\
&= \frac{1}{\pi}\eta \sum_{j=0}^{N-1} \omega_j e^{-\mathrm{i}\, \frac{A}{N} j(-\frac{2\pi}{A}\frac{N}{2}+\frac{2\pi}{A} l)} g_T(v_j) \\
&= \frac{1}{\pi}\eta \sum_{j=0}^{N-1} \omega_j e^{\mathrm{i}\, j\pi - \mathrm{i}\, j\frac{2\pi l}{N}} g_T(v_j) \\
z_T(k_l) &= \frac{1}{\pi} \sum_{j=0}^{N-1} \eta\omega_j e^{\mathrm{i}\, j\pi} g_T(v_j) \cdot e^{-\mathrm{i}\, j\frac{2\pi l}{N}} \\
&= \frac{1}{\pi} FFT\left(\eta\omega_j e^{\mathrm{i}\, j\pi} g_T(v_j)\right)
\end{aligned}$$

La relazione da implementare è quindi

$$\boxed{z_T(k_l) = \frac{1}{\pi} FFT\left(\eta\omega_j e^{\mathrm{i}\, j\pi} g_T(v_j)\right)} \tag{2.7}$$

2.4 Il metodo CONV

2.4.1 Inquadramento teorico

Introduzione Il punto di partenza è la seguente formula di valutazione neutrale rispetto al rischio

$$V(K,T,S_0) = e^{-rT} \int_{\mathbb{R}} H(y) f(y|S_0) dy \tag{2.8}$$

dove V è il valore dell'opzione, $H(\cdot)$ è il suo payoff a scadenza e $f(\cdot|S_0)$ è la densità di probabilità del prezzo del sottostante a scadenza, dato un valore spot pari a S_0. Si può vedere f come una probabilità di transizione,

$$\begin{aligned} f &: \mathbb{R} \times \mathbb{R} \to [0,1] \\ (y,x) &\mapsto f(y,x) \end{aligned}$$

Il metodo CONV (o della convoluzione), introdotto in [11], si basa sull'ipotesi che il valore di f non dipende da x e y ma dalla loro differenza $x - y$.

$$f(y|x) = f(y-x) \tag{2.9}$$

Per comodità si effettua il passaggio in coordinate logaritmiche, l'equazione diventa

$$V(K,T,x) = e^{-rT} \int_{\mathbb{R}} H(y) f(y|x) dy \tag{2.10}$$

nella quale $x = \log(S_t)$ e $f(\cdot|x)$ è la densità della probabilità di transizione del logaritmo del prezzo del sottostante dal tempo 0 al tempo T tenuto conto che inizialmente valeva x.

Convoluzione Da questo momento ci si focalizza su un'opzione Call, se ne indicherà quindi il prezzo con la lettera C. Sostituendo la (2.9) in (2.4.1) e applicando un cambio di variabili $z = y - x$ si ottiene

$$C(K,T) = e^{-rT} \int_{\mathbb{R}} H(x+z) f(z) dz \tag{2.11}$$

L'integrale che vi compare non è altro che una convoluzione.

A partire da questa espressione è possibile utilizzare la funzione caratteristica del sottostante per riscrivere l'equazione in modo adatto al calcolo numerico. L'idea del metodo è quella di applicare la trasformata di Fourier alla (2.11) per esprimere il valore del derivato come l'antitrasformata del prodotto di due funzioni:

- La trasformata di Fourier del payoff del derivato
- La funzione caratteristica del modello che descrive l'andamento del sottostante

questo è possibile poiché la funzione caratteristica di una convoluzione è il prodotto delle funzioni caratteristiche dei fattori.

Gli integrali coinvolti vengono poi approssimati con delle formule di quadratura per il calcolo delle quali è possibile sfruttare l'algoritmo FFT.

Per ottere un risultato sensato è necessario garantire che la funzione a cui viene applicata la trasformata di Fourier sia in L^1. Si considera quindi il prezzo della call temperato tramite un fattore esponenziale $e^{\alpha x}$, dove x è la variabile indipendente della funzione da trasformare e il valore di α verrà scelto a posteriori per garantire il soddisfacimento della proprietà di integrabilità.

Applicando l'operatore $\mathcal{F}$ ad entrambi i membri di (2.10) si ottiene

$$\mathcal{F}\{C(K,T)\} = e^{-rT}\mathcal{F}\left\{\int_{\mathbb{R}} H(x+z)f(z)dz\right\}. \tag{2.12}$$

Definendo:

$$\begin{aligned} c_\alpha(x) &= e^{\alpha x}C(x) \\ h_\alpha(x) &= e^{\alpha(x)}H(x) \end{aligned}$$

si può scrivere

$$\begin{aligned} \mathcal{F}\{c_\alpha(K,T)\} &= e^{-rT+\alpha x}\mathcal{F}\left\{\int_{\mathbb{R}} h_\alpha(x+z)e^{-\alpha(x+z)}f(z)dz\right\} \\ &= e^{-rT+\alpha x}\int_{\mathbb{R}} e^{\mathrm{i}\,ux}\left\{\int_{\mathbb{R}} h_\alpha(x+z)e^{-\alpha(x+z)}f(z)dz\right\}dx \\ &= e^{-rT}\int_{\mathbb{R}} e^{\mathrm{i}\,ux+\alpha x-\alpha x+(\mathrm{i}\,uz-\mathrm{i}\,uz)-\alpha z}\left\{\int_{\mathbb{R}} h_\alpha(x+z)f(z)dz\right\}dx \\ &= e^{-rT}\int_{\mathbb{R}}\int_{\mathbb{R}} e^{\mathrm{i}\,u(x+z)}h_\alpha(x+z)e^{-\mathrm{i}(u-\mathrm{i}\,\alpha)z}f(z)dzdx. \end{aligned}$$

Ritornando alla variabile $y = x + z$ e spezzando l'integrale doppio in due integrali semplici si ottiene

$$\begin{aligned} \mathcal{F}\{c_\alpha(K,T)\} &= e^{-rT}\int_{\mathbb{R}}\int_{\mathbb{R}} e^{\mathrm{i}\,uy}h_\alpha(y)e^{-\mathrm{i}(u-\mathrm{i}\,\alpha)z}f(z)dzdy \\ &= e^{-rT}\int_{\mathbb{R}} e^{\mathrm{i}\,uy}h_\alpha(y)dy\int_{\mathbb{R}} e^{-\mathrm{i}(u-\mathrm{i}\,\alpha)z}f(z)dz \\ &= e^{-rT}\mathcal{F}\{h_\alpha(y)\}(u)\Phi(-(u-\mathrm{i}\,\alpha)). \end{aligned}$$

Applicando l'operatore $\mathcal{F}^{-1}$ si ricava il prezzo della Call.

$$\boxed{C(x,T) = e^{-rT-\alpha x}\mathcal{F}^{-1}\left\{\mathcal{F}\left\{h_\alpha(y)\right\}\Phi(-(u-\mathrm{i}\,\alpha))\right\}(x)} \tag{2.13}$$

dove $\mathcal{F}$ rappresenta la trasformata di Fourier.

2.4.2 Calcolo numerico

Per poter valutare la (2.13) è necessario calcolare le seguenti due trasformate, una interna all'altra

$$\mathcal{G}(u) = \mathcal{F}\{h_\alpha(y)\} = \int_{\mathbb{R}} e^{\mathrm{i}\,ux} h_\alpha(x)dx \tag{2.14}$$

$$c_\alpha(K,T) = e^{-rT} \int_{\mathbb{R}} e^{-\mathrm{i}\,ux} \underbrace{\int_{\mathbb{R}} e^{\mathrm{i}\,uy} h_\alpha(y)dy}_{\mathcal{G}(u)} \Phi(-(u-\mathrm{i}\,\alpha))du \tag{2.15}$$

L'obiettivo è ricondursi al calcolo di una trasformata di Fourier discreta per poter applicare la FFT. Si discretizza la (2.14) con la formula dei trapezi e la (2.15) con la formula dei rettangoli.

Griglia La griglia standard da costruire è tridimensionale e ogni dimensione ha lunghezza N. Questo avviene poiché x rappresenta il log-price al tempo t_0; y il log-price al tempo T e u la variabile delle frequenze nello spazio di Fourier.

La griglia viene scelta nel seguente modo

$$\begin{aligned} u_j &= u_0 + j\Delta_u \\ x_j &= x_0 + j\Delta_x \\ y_j &= y_0 + j\Delta_y \end{aligned}$$

con $j = 1, \dots, N$. Si sceglie $\Delta_x = \Delta_y$ e Δ_u tale da soddisfare la relazione di Nyquist $\Delta_u\Delta_y = \frac{2\pi}{N}$.

Occorre scegliere la griglia in modo da ridurre gli errori di discretizzazione, in [11] gli autori suggeriscono $y_0 = -\frac{L}{2}$ e $\Delta_y = \frac{L}{N}$ dove L viene scelto in modo da coprire quasi tutta la massa della densità del sottostante nell'intervallo $\left[-\frac{L}{2}, \frac{L}{2}\right]$. In letteratura viene suggerita la scelta $L = 10$.

In definitiva la griglia utilizzata è

$$\begin{cases} u_j = \left(j - \dfrac{N}{2}\right)\Delta_u \\ x_j = y_j = x_0 + \left(j - \dfrac{N}{2}\right)\Delta_y \\ y_j = y_0 + \left(j - \dfrac{N}{2}\right)\Delta_y \end{cases} \tag{2.16}$$

Trasformata interna In primo luogo si procede alla discretizzazione della trasformata interna (2.14) sulla griglia y. Per u fissato, utilizzando la regola dei trapezi, si ottiene

$$\mathcal{G}(u) = \mathcal{F}\{h_\alpha(y)\}(u) = \int_{\mathbb{R}} e^{\mathrm{i}\,uy} h_\alpha(y)dy \simeq \sum_{j=0}^{N-1} w_n e^{\mathrm{i}\,uy_j} h_\alpha(y_j)\Delta y \tag{2.17}$$

Dove i pesi w_j sono pari a

$$\begin{cases} w_j = 1 & \text{per} \quad j = 1, \dots, N-2 \\ w_j = \frac{1}{2} & \text{per} \quad j \in \{0, N\} \end{cases}$$

Trasformata esterna Per quanto riguarda la trasformata esterna, si procede alla discretizzazione con formula dei rettangoli, prendendo la funzione calcolata nell'estremo sinistro di ciascun intervallo di discretizzazione.

$$\begin{aligned} \mathcal{F}^{-1}\left[\mathcal{G}(u)\Phi(-(u-\mathrm{i}\,\alpha))\right](x) &= \frac{1}{2\pi}\int_{\mathbb{R}} e^{-\mathrm{i}\,ux}\mathcal{G}(u)\Phi(-(u-\mathrm{i}\,\alpha))du \\ &\simeq \frac{1}{2\pi}\sum_{j=0}^{N-1} e^{-\mathrm{i}\,u_j x}\mathcal{G}(u_j)\Phi(-(u_j-\mathrm{i}\,\alpha))\Delta u. \end{aligned} \tag{2.18}$$

Approssimazione numerica Sostituendo le approssimazioni di entrambe le trasformate (2.17) e (2.18) nella formula (2.13) si giunge alla seguente approssimazione

$$C(x,T) = e^{-rT-\alpha x}\frac{\Delta u \Delta y}{2\pi}\sum_{j=0}^{N-1} e^{-\mathrm{i}\,u_j x}\Phi(-(u_j-\mathrm{i}\,\alpha))\sum_{n=0}^{N-1} w_n e^{\mathrm{i}\,u_j y_n}h_\alpha(y_n). \tag{2.19}$$

Esplicitando la sommatoria interna tenendo conto della struttura delle griglie (2.16) si ottiene

$$\begin{aligned} \sum_{n=0}^{N-1} w_n e^{\mathrm{i}\,u_j y_n}h_\alpha(y_n) &= e^{\mathrm{i}\,x_0 y_0 + \mathrm{i}\,j\Delta u y_0}\sum_{n=0}^{N-1} w_n e^{\mathrm{i}\,jn\Delta u\Delta y}\underbrace{e^{\mathrm{i}\,u_0 n\Delta y}}_{e^{\mathrm{i}\left(-\frac{N}{2}\Delta u\right)n\Delta y}=e^{\mathrm{i}\,n\pi}}h_\alpha(y_n) \\ &= e^{\mathrm{i}\,x_0 y_0 + \mathrm{i}\,j\Delta u y_0}\sum_{n=0}^{N-1} w_n e^{\mathrm{i}\,jn\frac{2\pi}{N}}(-1)^n h_\alpha(y_n) \\ &= e^{\mathrm{i}\,x_0 y_0 + \mathrm{i}\,j\Delta u y_0}\,\mathrm{fft}\left\{w_n(-1)^n h_\alpha(y_n)\right\}. \end{aligned}$$

Sostituendo questa espressione nella (2.19) si ottiene

$$C(x,T) = e^{-rT-\alpha x}\frac{\Delta u \Delta y}{2\pi}\sum_{j=0}^{N-1} e^{-\mathrm{i}\,u_j x}\Phi(-(u_j-\mathrm{i}\,\alpha))e^{\mathrm{i}\,u_0 y_0 + \mathrm{i}\,j\Delta u y_0}\,\mathrm{fft}\left\{w_n(-1)^n h_\alpha(y_n)\right\}.$$

Esplicitando u_j ed applicando calcoli simili ai precedenti si ottiene la formulazione discreta della (2.13).

$$\boxed{C(x_p,T) = e^{-rT-\alpha x_p + \mathrm{i}\,u_0(y_0-x_0)}(-1)^p\,\mathrm{ifft}\left[e^{\mathrm{i}\,j(y_0-x_0)\Delta_u}\Phi(-(u_j-\mathrm{i}\,\alpha))\,\mathrm{fft}\left[(-1)^n w_n h_\alpha(y_n)\right]\right].}$$

2.4.3 Generalizzazione alle opzioni Barriera

Il metodo CONV può essere facilmente generalizzato e reso applicabile al calcolo del prezzo delle opzioni Barriera con un numero M di date di monitoraggio.

L'idea è di dividere semplicemente l'intervallo temporale $[t_0, T]$ in sottointervalli di ampiezza $dt = \frac{T-t_0}{M}$, i cui estremi sono quindi le date di monitoraggio

$$t_0 < t_1 < \cdots < t_m = T.$$

A questo punto si applica il CONV per le europee utilizzando dt come orizzonte temporale ed ottenendo dunque il valore al tempo $t_{m-1} = T - dt$. Si effettua poi una proiezione che annulla tutto quanto cade fuori dalle barriere.

Il valore così ottenuto viene utilizzato come nuovo payoff finale nell'applicare il CONV tra t_{m-1} e $t_{m-2} = T - 2dt$.

Iterando il procedimento M volte si ricava il prezzo dell'opzione Barriera al tempo t_0.

Costo computazionale Il costo computazionale del CONV nella versione base è dell'ordine di $O(N \log_2 N)$, mentre nel caso di opzione barriera, essendo lineare in M, è dell'ordine di $O(MN \log_2 N)$.

2.5 Il metodo COS

2.5.1 Inquadramento teorico

In questo capitolo ci si concentra sull'applicazione dell'espansione in serie di Fourier di coseni nel contesto dell'integrazione numerica come ulteriore metodo in alternativa a quelli basati sulla FFT. Il metodo COS, originariamente introdotto in [6] e generalizzato in [7] può gestire dinamiche molto generali ed è in grado di fornire simultaneamente i prezzi per un vettore di strike prices. Questo metodo è in grado di migliorare la velocità di convergenza nel pricing delle opzioni plain vanilla ed anche nell'ambito di alcune opzioni esotiche, come barriera [7] ed asiatiche [16].

Definizione 2.5.1 (Espansione in serie di coseni)**.** Sia $g : [0, \pi] \to \mathbb{R}$, la sua espansione in serie di coseni è data da

$$g(\theta) = \frac{A_0}{2} + \sum_{k=1}^{\infty} A_k \cos(k\theta)$$
$$A_k = \frac{2}{\pi} \int_0^{\pi} g(\theta) \cos(k\theta)\, d\theta$$

Per una funzione con dominio generico $f : [a, b] \to \mathbb{R}$ è ancora possibile scrivere la serie di coseni, infatti, con il cambio di variabili:

$$\begin{cases} \theta = \frac{x-a}{b-a} \\ x = \frac{b-a}{\pi}\theta + a \end{cases}$$

si ottiene

$$f(x) = \frac{A_0}{2} + \sum_{k=1}^{\infty} A_k \cos\left(k\pi \frac{x-a}{b-a}\right)$$
$$A_k = \frac{2}{b-a} \int_a^b f(x) \cos\left(k\pi \frac{x-a}{b-a}\right) dx$$

Applicazione al pricing Il punto di partenza è la formula di valutazione neutrale rispetto al rischio

$$v(x,t) = e^{-r(T-t)} \mathbb{E}^{\mathbb{Q}}[v(y,T)|x] = e^{-r(T-t)} \int_{\mathbb{R}} v(y,T) f(y|x) dy \tag{2.20}$$

Dove v è il valore dell'opzione, T la maturity, t la data di partenza, r il tasso di interesse risk-free, $\mathbb{Q}$ la misura di probabilità risk neutral.

Poiché il problema di option pricing in (2.20) comporta il calcolo di un integrale con dominio infinito occorre effettuare un troncamento. Scegliendo $[a, b] \subset \mathbb{R}$ tale da rendere piccolo l'errore di approssimazione si ottiene

$$v(x,t) \simeq e^{-r(T-t)} \int_a^b v(y,T) f(y|x) dy \tag{2.21}$$

La densità $f(\cdot)$ in generale non è nota in forma chiusa oppure è difficile da valutare. Di contro però la sua funzione caratteristica è spesso nota. Per questo motivo si sostituisce la $f(\cdot)$ con la sua espansione in serie di coseni (troncata ai primi N termini) rispetto alla variabile y.

$$f(y|x) = \frac{A_0(x)}{2} + \sum_{k=1}^{N-1} A_k(x) \cos\left(k\pi \frac{y-a}{b-a}\right)$$
$$A_k(x) = \frac{2}{b-a} \int_a^b f(y|x) \cos\left(k\pi \frac{y-a}{b-a}\right) dy$$

Sostituendo questa espressione nella (2.21) e scambiando le operazioni di somma ed integrazione si ottiene

$$v(x,T) = \frac{b-a}{2} e^{-rT} \left(\frac{A_0(x)}{2} V_0 + \sum_{k=1}^{N-1} A_k(x) \cos\left(k\pi \frac{y-a}{b-a} V_k\right)\right)$$
$$V_k = \frac{2}{b-a} \int_a^b v(y,T) \cos\left(k\pi \frac{y-a}{b-a}\right) dy$$

Coefficienti A_k Occorre ora determinare i coefficienti A_k che compaiono nell'espressione precedente, l'idea è di approssimare l'integrale finito che compare in A_k con l'integrale infinito, per non abusare della notazione si rinominano questi nuovi coefficienti $\tilde{A}_k$.

$$\begin{aligned}
A_k(x) &= \frac{2}{b-a} \int_a^b f(y|x) \left(k\pi \frac{y-a}{b-a}\right) dy \\
&\simeq \tilde{A}_k(x) \\
&= \frac{2}{b-a} \int_{\mathbb{R}} f(y|x) \cos\left(k\pi \frac{y-a}{b-a}\right) dy \\
&= \frac{2}{b-a} \operatorname{Re}\left\{ \int_{\mathbb{R}} f(y|x) \left(\cos\left(k\pi \frac{y-a}{b-a}\right) + \mathrm{i} \sin\left(k\pi \frac{y-a}{b-a}\right)\right) dy \right\} \\
&= \frac{2}{b-a} \operatorname{Re}\left\{ \int_{\mathbb{R}} f(y|x) e^{\mathrm{i}\, k\pi \frac{y-a}{b-a}} dy \right\} \\
&= \frac{2}{b-a} \operatorname{Re}\left\{ e^{\mathrm{i}\, \frac{-k\pi a}{b-a}} \int_{\mathbb{R}} f(y|x) e^{\mathrm{i}\, y \frac{k\pi}{b-a}} \right\} \\
&= \frac{2}{b-a} \operatorname{Re}\left\{ e^{\mathrm{i}\, \frac{-k\pi a}{b-a}} \Phi\left(\frac{k\pi}{b-a}; x\right)\right\}
\end{aligned}$$

Si ottengono quindi i coefficienti

$$\tilde{A}_k(x) = \frac{2}{b-a} \operatorname{Re}\left(\Phi\left(\frac{k\pi}{b-a}; x\right) e^{-\mathrm{i}\, \frac{ka\pi}{b-a}} \right)$$

dove Φ è la funzione caratteristica del modello sottostante, che si suppone nota in forma chiusa.

Di conseguenza la formula da utilizzare per il pricing è

$$\boxed{v(x,T) = e^{-rT} \left(\frac{1}{2} \operatorname{Re}(\Phi(0) V_0) + \sum_{k=1}^{N-1} \operatorname{Re}\left(\Phi\left(\frac{k\pi}{b-a}; x\right) e^{\mathrm{i}\, k\pi \frac{a}{b-a}} \right) V_k \right)}. \quad (2.22)$$

Errori numerici Una dettagliata analisi degli errori di approssimazione è riportata in [6], vengono analizzati gli errori causati da:

- Troncamento dell'integrale, $\int_{\mathbb{R}} \simeq \int_a^b$
- Troncamento della serie $\sum_{k=1}^{\infty} \simeq \sum_{k=1}^{N-1}$
- Utilizzo della funzione caratteristica $\Phi(u) = \int_{\mathbb{R}} e^{-\mathrm{i}\,ux} f(x)dx$ invece dell'integrale $\int_a^b e^{-\mathrm{i}\,ux} f(x)dx$

Per contenere gli errori di troncamento la scelta suggerita dagli autori è:

$$[a,b] := \left[c_1 - L\sqrt{|c_2| + \sqrt{|c_4|}},\ c_1 + L\sqrt{|c_2| + \sqrt{|c_4|}}\right] \tag{2.23}$$

dove c_n è l'n-esimo cumulant[2] di $\log \frac{S_T}{K}$.

Coefficienti V_k La chiave di volta del pricing di diversi tipi di opzione risiede nei coefficienti V_k, i quali sono determinati dal payoff dell'opzione stessa. Si ricorda che V_k è dato da

$$V_k = \frac{2}{b-a}\int_a^b v(y,T)\cos\left(k\pi\frac{y-a}{b-a}\right)dy \tag{2.24}$$

Dove i payoff a scadenza per le europee sono pari a:

$$v(y,T)^{Call} = K\,(e^y - 1)^+ \tag{2.25}$$
$$v(y,T)^{Put} = K\,(1 - e^y)^+ \tag{2.26}$$

Osservando che

$$e^y - 1 > 0 \iff y > 0$$

e definendo le funzioni $\xi_k(c,d)$ e $\psi_k(c,d)$ come

$$\begin{aligned}
\xi_k(c,d) &= \int_c^d e^y \cos\left(k\pi\frac{y-a}{b-a}\right)dy \\
&= \frac{1}{1+\left(\frac{k\pi}{b-a}\right)^2}\left[\cos\left(k\pi\frac{d-a}{b-a}\right)e^d - \cos\left(k\pi\frac{c-a}{b-a}\right)e^c\right. \\
&\left. + \frac{k\pi}{b-a}\left(\sin\left(k\pi\frac{d-a}{b-a}\right)e^d - \sin\left(k\pi\frac{c-a}{b-a}\right)e^c\right)\right] \\
\psi_k(c,d) &= \int_c^d \cos\left(k\pi\frac{y-a}{b-a}\right)dy \\
&= \begin{cases} \left[\sin\left(k\pi\frac{d-a}{b-a}\right) - \sin\left(k\pi\frac{c-a}{b-a}\right)\right]\frac{b-a}{k\pi} & k \neq 0 \\ d-c & k = 0 \end{cases}
\end{aligned}$$

[2] I cumulants κ_n di una variabile aleatoria X sono definiti tramite la funzione generatrice dei cumulants $g(t) = \ln \mathbb{E}[e^{tX}]$, che è il logaritmo della funzione generatrice dei momenti, attraverso la relazione $\kappa_n = \frac{\partial^n}{\partial t^n} g(t)\big|_{t=0}$

si ottengono i coefficienti cercati:

$$V_k^{Call} = \int_a^b K\left(e^y - 1\right)^+ dy = \int_0^b K\left(e^y - 1\right) dy = \frac{2}{b-a} K\left(\xi_k(0,b) - \psi_k(0,b)\right)$$
$$V_k^{Put} = \int_a^b K\left(1 - e^y\right)^+ dy = \int_a^0 K\left(1 - e^y\right) dy = \frac{2}{b-a} K\left(\psi_k(a,0) - \xi_k(a,0)\right)$$

Semplificazione Per i modelli basati su processi di Lévy o per il modello di Heston l'espansione in serie di coseni può essere semplificata e risulta pari a:

$$v(K,T) = e^{-rT}\,\mathrm{Re}\left(\frac{\Phi(0)V_0}{2} + \sum_{k=1}^{N-1} \Phi\left(\frac{k\pi}{b-a}\right) V_k e^{\mathrm{i}\,k\pi\frac{x-a}{b-a}}\right) \tag{2.27}$$

dove $V_k = U_k K$, con i coefficienti scalari U_k dati da:

$$U_k^{Call} = \frac{2}{b-a}\left(\xi_k(0,b) - \psi_k(0,b)\right) \tag{2.28}$$
$$U_k^{Put} = \frac{2}{b-a}\left(-\xi_k(a,0) + \psi_k(a,0)\right) \tag{2.29}$$

Quindi è possibile separare il calcolo dei coefficienti dalla moltiplicazione ed ottenere

$$v(K,T) = e^{-rT} K\,\mathrm{Re}\left(\frac{\Phi(0)U_0}{2} + \sum_{k=1}^{N-1} \Phi\left(\frac{k\pi}{b-a}\right) U_k e^{\mathrm{i}\,k\pi\frac{x-a}{b-a}}\right) \tag{2.30}$$

Risulta così immediato ottenere il prezzo per diversi strike, qualità molto utile ai fini della calibrazione.

2.5.2 Generalizzazione alle opzioni Barriera

Per il pricing delle opzioni barriera occorre apportare alcune modifiche all'algoritmo. Innanzitutto la procedura si basa, come nel caso del CONV, sulla backward induction. Siano $U, D \in \mathbb{R}^+ : U \geq D$ le barriere up e down dell'opzione considerata.

Posto

$$h_u = \log\frac{U}{K}$$
$$h_d = \log\frac{D}{K}$$

si ha che, con il set di date di monitoraggio $t_1 < \cdots < t_{M-1} < t_M = T$ il prezzo di una knock out monitorata M volte soddisfa la formula ricorsiva per $m = M, M-1, \ldots, 2$.

$$\begin{cases} c(x,t_{m-1}) = e^{-r(t_m - t_{m-1})} \int_{\mathbb{R}} v(x,t_m) f(y|x) dy \\ v(x,t_{m-1}) = \begin{cases} 0 & x < h_d \\ c(x,t_{m-1}) & h_d \leq x \leq h_u \\ 0 & x > h_u \end{cases} \end{cases}$$

Ovvero ad ogni istante temporale si applica la formula del COS per le europee e si proietta a zero ciò che cade fuori dalle barriere per ottenere il prezzo all'istante precedente Per quanto riguarda la procedura di pricing si può applicare, si veda [7], il seguente

Algoritmo 2.5.1. Per $m = M-1, M-2, \ldots, 1$. $V_k(t_m) = C_k(h_d, h_u, t_m)$, dove

$$C_k(x_1, x_2, t_m) = \frac{e^{-r\Delta t}}{\pi} \operatorname{Im}\{(M_c + M_s)\underline{u}\} \tag{2.31}$$

con

$$\begin{aligned} \underline{u} &= \{u_j\}_{j=0,\ldots,N-1} \\ u_j &= \Phi\left(\frac{j\pi}{b-a}\right) V_j(t_{m+1}) \\ u_0 &= \frac{1}{2}\Phi(0) V_0(t_{m+1}) \end{aligned}$$

per le matrici M_c ed M_s si faccia riferimento a [7].

Coefficienti $V_k(t_M)$ Occorre a questo punto inizializzare la ricorsione con i coefficienti al tempo finale $V_k(t_M)$, i quali sono ancora definiti secondo la (2.24).

I payoff a scadenza per le opzioni barriera knock-out sono i seguenti:

$$v(y,T)^{Call} = K\left(e^y - 1\right)^+ \mathbb{I}_{\{y<h_u\}} \mathbb{I}_{\{y>h_d\}} \tag{2.32}$$

$$v(y,T)^{Put} = K\left(1 - e^y\right)^+ \mathbb{I}_{\{y<h_u\}} \mathbb{I}_{\{y>h_d\}} \tag{2.33}$$

Si può quindi scrivere che

$$\begin{aligned} V_k^{Call} &= \int_a^b K\left(e^y - 1\right)^+ \mathbb{I}_{\{y<h_u\}} \mathbb{I}_{\{y>h_d\}} = \int_{h_d}^{h_u} K\left(e^y - 1\right)^+ dy \\ V_k^{Put} &= \int_a^b K\left(1 - e^y\right)^+ \mathbb{I}_{\{y<h_u\}} \mathbb{I}_{\{y>h_d\}} = \int_{h_d}^{h_u} K\left(1 - e^y\right)^+ dy \end{aligned}$$

A questo punto a seconda della posizione reciproca delle barriere e dello strike price si possono verificare diversi casi. Delle $3! = 6$ possibili combinazioni occorre eliminare

le 3 che prevedono $U < D$, che per definizione non può essere. Gli ordinamenti da considerare sono quindi i seguenti

$$\begin{aligned} D < U < K &\iff h_d < h_u < 0 \\ D < K < U &\iff h_d < 0 < h_u \\ K < D < U &\iff 0 < h_d < h_u \end{aligned}$$

Per comodità si pone

$$G_k(x_1, x_2) = \frac{2}{b-a}\alpha K[\xi_k(x_1, x_2) - \psi_k(x_1, x_2)]$$

dove α vale 1 per la Call e -1 per la Put e le funzioni $\xi_k(x_1, x_2)$ e $\psi_k(x_1, x_2)$ sono quelle definite precedentemente nella parte sulle Europee.

Si ricava, analogamente rispetto a quanto fatto per le europee, in maniera immediata che

$$V_k(t_M)^{Call} = \begin{cases} \int_{h_d}^{h_u} K\left(e^y - 1\right) dy = G_k(h_d, h_u) & 0 < h_d < h_u \\ 0 & h_d < h_u < 0 \\ \int_{0}^{h_u} K\left(e^y - 1\right) dy = G_k(0, h_u) & h_d < 0 < h_u \end{cases}$$

$$V_k(t_M)^{Put} = \begin{cases} 0 & 0 < h_d < h_u \\ \int_{h_d}^{h_u} K\left(1 - e^y\right) dy = G_k(h_d, h_u) & h_d < h_u < 0 \\ \int_{h_d}^{0} K\left(1 - e^y\right) dy = G_k(h_d, 0) & h_d < 0 < h_u \end{cases}$$

2.5.3 Generalizzazione alle opzioni Asiatiche

Il metodo COS può essere generalizzato anche al caso delle opzioni Asiatiche, come suggerito in [16].

La procedura consiste nel ricavare la funzione caratteristica della media del valore del sottostante e di utilizzarla poi per calcolare il prezzo dell'opzione tramite l'espansione di Fourier in serie di coseni.

Per quanto riguarda le opzioni asiatiche a media geometrica con M date di monitoraggio, il payoff è della forma:

$$g(S) = \begin{cases} \max\left(\left(\prod_{j=0}^{M} S_j\right)^{\frac{1}{M+1}} - K, 0\right) & \text{per la Call} \\ \max\left(K - \left(\prod_{j=0}^{M} S_j\right)^{\frac{1}{M+1}}, 0\right) & \text{per la Put} \end{cases}$$

La funzione caratteristica della media geometrica è nota in maniera esplicita, trasformando il sottostante applicando il logaritmo:

$$y = \log\left(\left(\prod_{j=0}^{M} S_j\right)^{\frac{1}{M+1}}\right) = \frac{1}{M+1}\sum_{j=0}^{M}\log(S_j) = \frac{1}{M+1}\sum_{j=0}^{M} x_j$$

Dividendo il dominio temporale $[t_0, T]$ in intervalli di ampiezza $dt = \frac{T-t_0}{M}$ e indicando con $\phi(u)$ la funzione caratteristica degli incrementi (indipendenti e stazionari) del processo di Lévy su ciascun intervallino, si può scrivere che la funzione caratteristica $\Phi(u)$ della media è

$$\Phi(u) = e^{\mathrm{i}\, u x_0} \prod_{j=0}^{M} \phi\left(u\frac{M+1-j}{M+1}\right) \tag{2.34}$$

A questo punto si può utilizzare la formula del COS per le europee (2.22) avendo cura di sostituire la (2.34), mentre i coefficienti V_k sono dati da

$$V_k(t_M)^{Call} = \frac{2}{b-a}K\left(\xi_k(0,b) - \psi_k(0,b)\right)$$
$$V_k(t_M)^{Put} = \frac{2}{b-a}K\left(\psi_k(a,0) - \xi_k(a,0)\right)$$

dove le funzioni $\xi_k(x_1, x_2)$ e $\psi_k(x_1, x_2)$ sono quelle definite precedentemente nella parte sulle Europee.

La complessità computazionale del metodo è lineare in N ed M, quindi il costo è $O(NM)$.

2.6 Confronto tra i metodi di pricing

2.6.1 Inquadramento teorico

In questo capitolo vengono confrontati i metodi di pricing applicabili al caso delle opzioni barriera, ovvero CONV, COS e Monte Carlo, in termini di accuratezza e tempo di esecuzione.

2.6.2 Confronto CONV e COS

Ai fini dell'analisi che si vuole effettuare vengono prezzate diverse opzioni barriera, tutte con parametri: $S_0 = 100$, $K = 100$, $r = 0.0367$, $T = 1$, $d = 0$ e barriere Down ed Out variabili. Tutti i diversi modelli introdotti nel Capitolo 1 sono considerati, per quanto riguarda i parametri per Black&Scholes si è scelto $\sigma = 0.17801$; per Merton $\sigma = 0.17801$, $\lambda = 4$, $\mu = 0.05$, $\delta = 0.04$; per Kou $\sigma = 0.17801$, $\lambda = 4$, $\lambda^+ = 20$, $\lambda^- = 4$, $p = 0.80$; per NIG e VG $\sigma = 0.17801$, $\kappa = 0.1$, $\theta = 0.1$; per Heston $V_0 = 0.17801$, $\theta = 0.02$, $\kappa = 0.1$; $\nu = 0.1$, $\rho = 0.5$.

La barriere vengono fatte variare su di una griglia di ampiezza $\Delta = 10$, con la barriera down $D \in [45, 95]$ e la barriera up $U \in [105, 155]$. Vi sono quindi 6 diversi livelli di barriera U e D. Il pricing viene poi riportato sia nel caso di opzione Call che di opzione Put, per un totale di 72 opzioni prezzate.

In Figura 2.2 e Figura 2.3 sono riportati gli andamenti dei prezzi ottenuti con i metodi CONV e COS nel caso di opzioni Call e Put. Il prezzo CONV è indicato in linea continua, mentre quello COS in linea tratteggiata. In Tabella 2.1 vengono invece riportati i tempi computazionali medi, sulle tutte le opzioni considerate, impiegati per il pricing. Tutti i tempi computazionali riportati in questa sezione sono indicati in millisecondi e sono stati ottenuti su un computer con processore Intel(R) Core(R) i7 Quad-Core, il codice è scritto in Matlab R2013b. Si può osservare che i metodi forniscono prezzi molto simili, mentre per quanto riguarda il tempo computazionale medio il COS tende a performare in maniera più efficiente di un fattore circa pari a 2.

Metodo	BS	Merton	Kou	NIG	VG	Heston
CONV	0.0794	0.1340	0.1314	0.1324	0.1401	0.1801
COS	0.0428	0.0778	0.0776	0.0769	0.0794	0.0735

Tabella 2.1: Confronto tempi computazionali CONV e COS

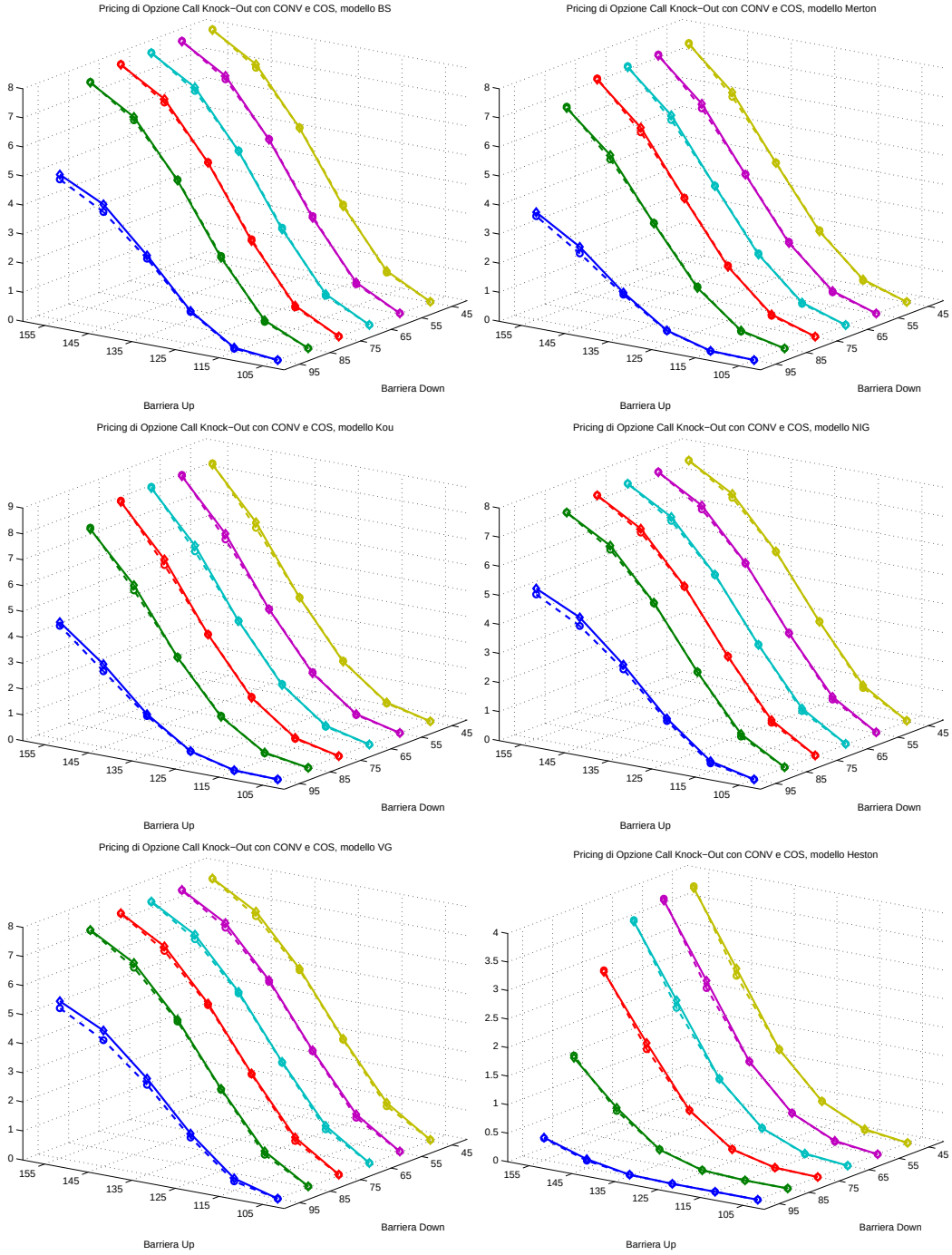


Figura 2.2: Confronto tra i prezzi per le opzioni Call.

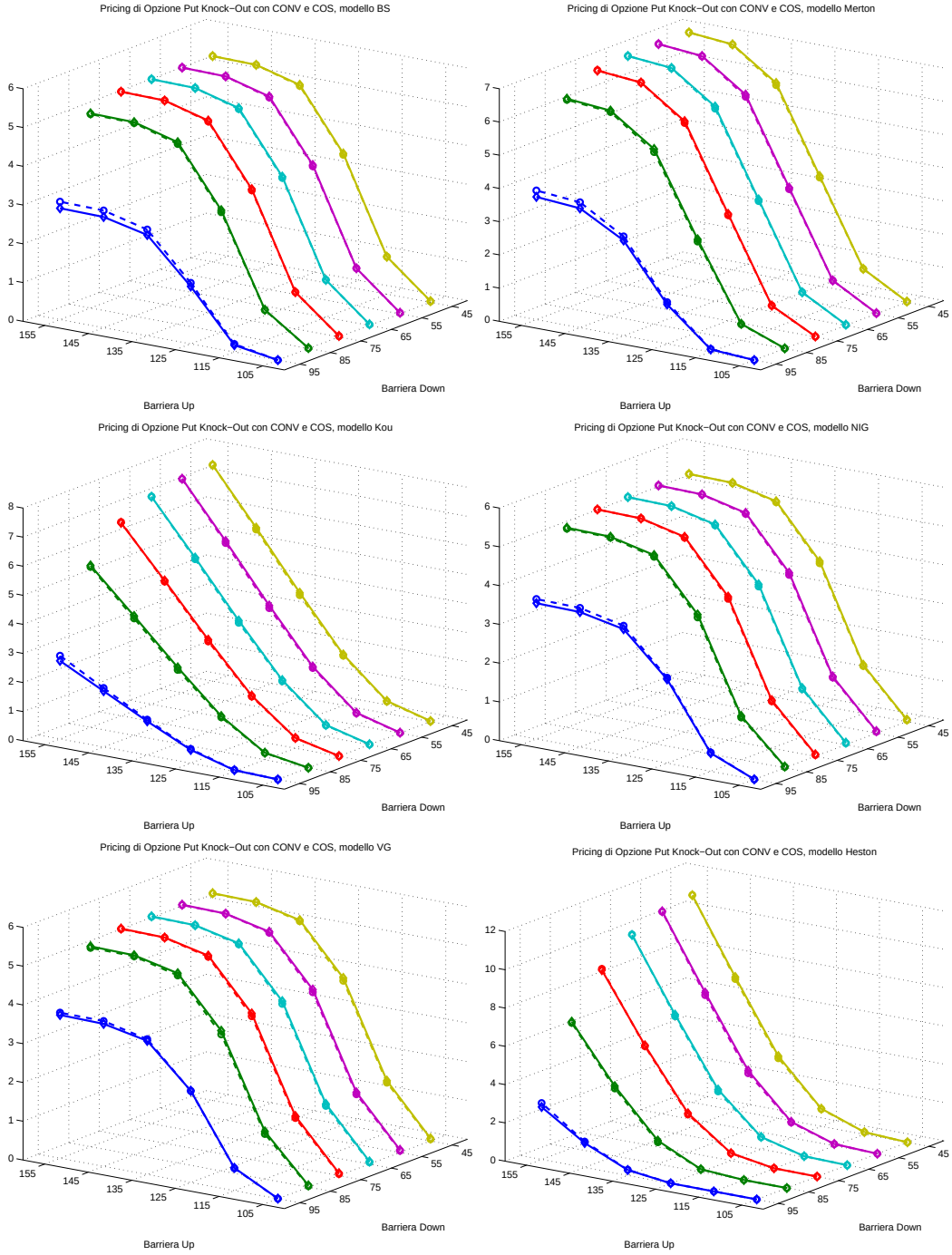


Figura 2.3: Confronto tra i prezzi per le opzioni Put.

2.6.3 Confronto COS e MC

Nella sezione precedente si è visto che il COS è in grado di performare meglio del metodo CONV, in questa sezione si confronterà il COS con il metodo di Monte Carlo. Vengono prezzate le stesse opzioni del caso precedente, con l'unica differenza che stavolta la barriera Down è fissata $D = 80$, e viene fatta variare la barriera up $U \in [105, 155]$. Vengono considerate soltanto opzioni Call.

In Figura 2.4 e Figura 2.5 sono riportati gli andamenti dei prezzi ottenuti con i metodi COS e Monte Carlo, oltre che all'ampiezza dell'intervallo di confidenza al 95% ottenuto per il Monte Carlo utilizzando rispettivamente 1000 e 10000 simulazioni. In Tabella 2.2 vengono invece riportati i tempi computazionali medi sulle 6 opzioni considerate, indicati in millisecondi, impiegati per il pricing.

Nel caso con 1000 simulazioni si può osservare che per il modello di Black&Scholes e Heston il Monte Carlo è addirittura più veloce del COS, mentre nei modelli più complessi il COS nettamente più performante. In ogni caso questo numero di simulazioni ha comportato un intervallo di confidenza inaccettabile. Passando a 10000 simulazioni si riesce a determinare con precisione la prima cifra decimale del prezzo anche con il Monte Carlo, tuttavia l'accuratezza non è comunque sufficiente ed il costo computazionale è significativamente superiore a quello del COS.

Metodo	BS	Merton	Kou	NIG	VG	Heston
COS	0.0476	0.0810	0.0793	0.0850	0.0818	0.0779
MC_{1000}	0.0190	1.9506	8.4619	0.1456	1.9362	0.0266
MC_{10000}	0.1040	19.6264	85.9670	1.4302	20.2226	0.2375

Tabella 2.2: Confronto tempi computazionali COS e MC

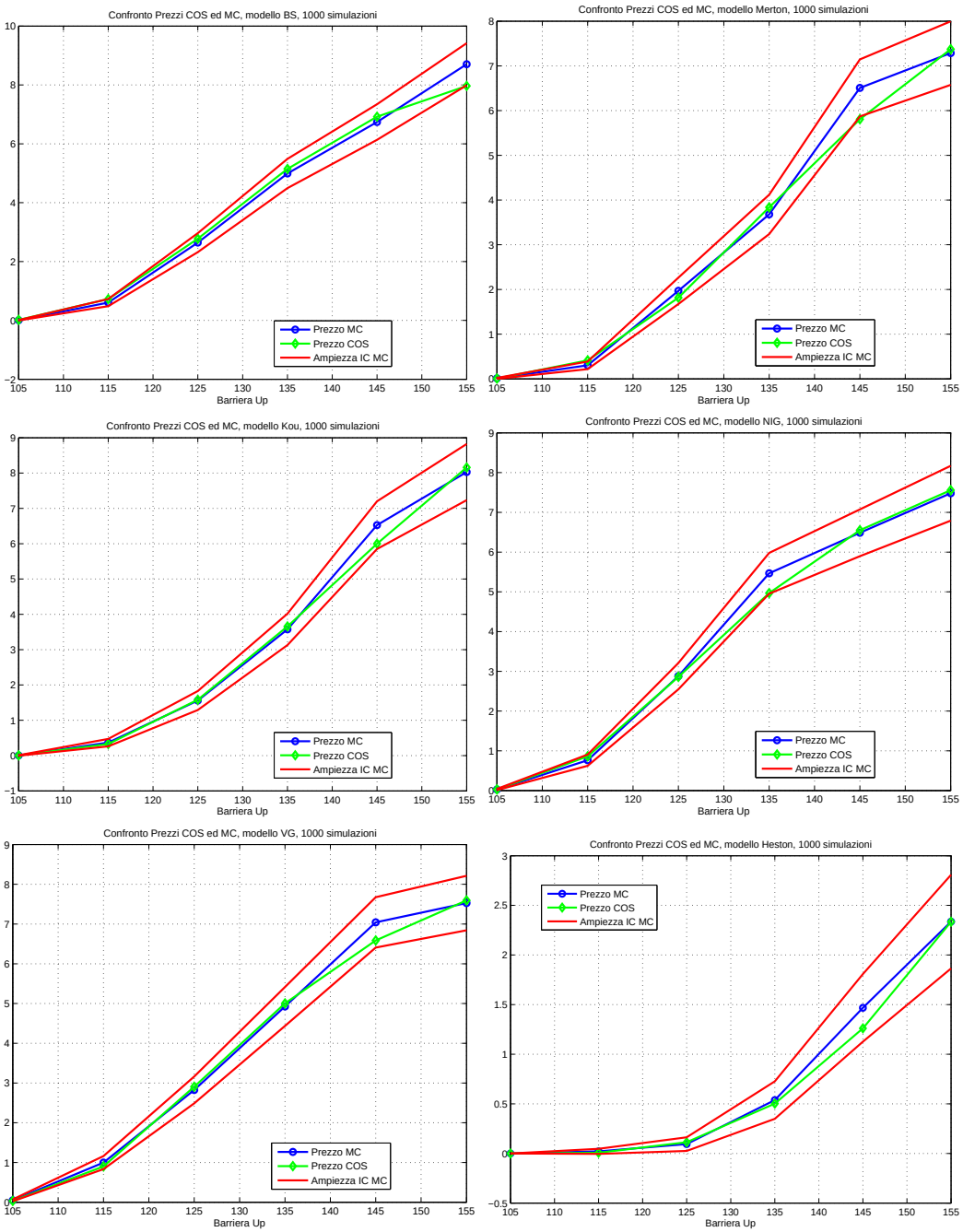


Figura 2.4: Confronto tra i prezzi per le opzioni, 1000 simulazioni.

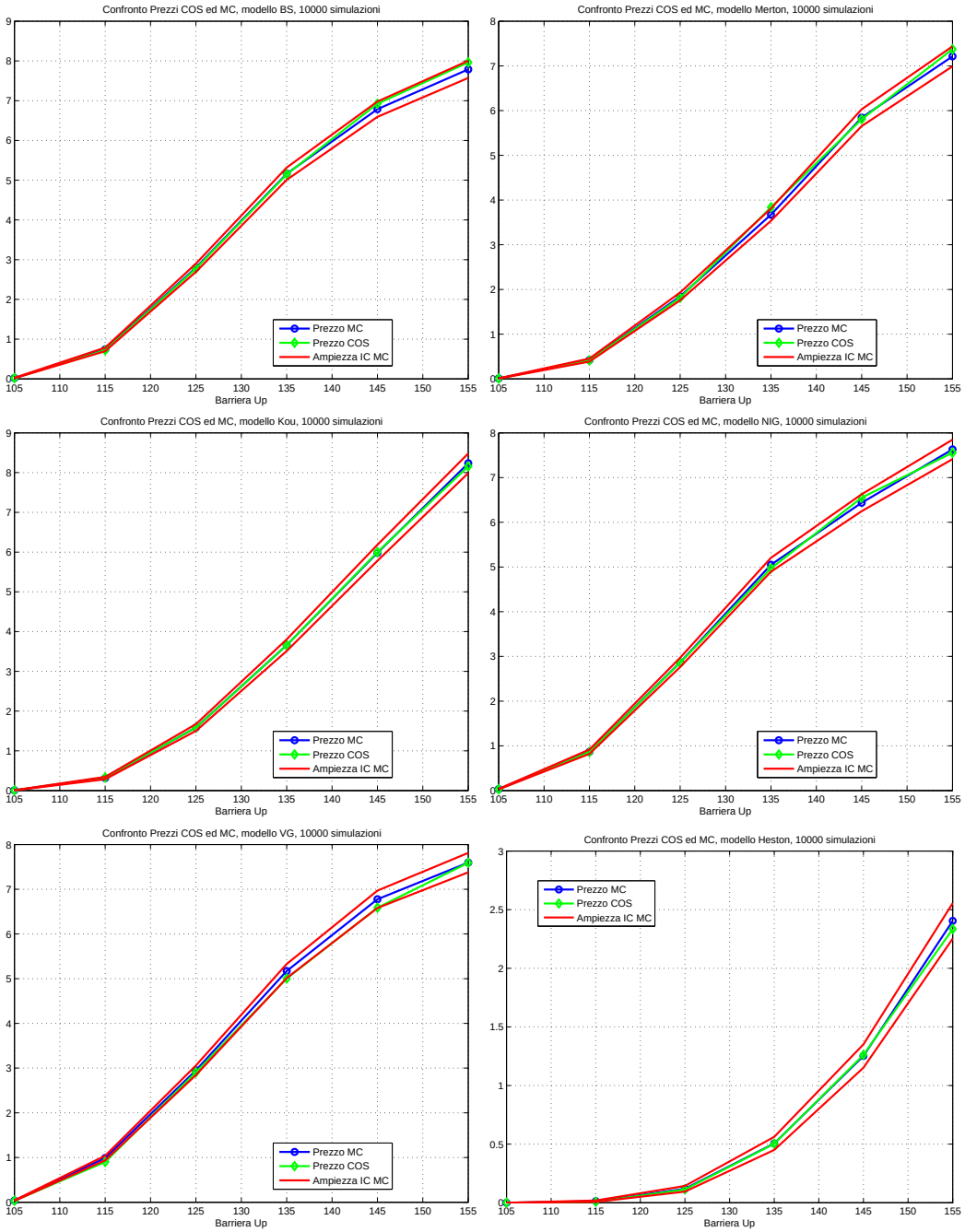


Figura 2.5: Confronto tra i prezzi per le opzioni, 10000 simulazioni.

2.6.4 Convergenza del COS per le Barriera

Nelle sezioni precedenti si è visto come il COS sia in grado di fornire prezzi accurati con un ridotto costo computazionale, in questa sezione viene analizzata la velocità di convergenza dell'errore di approssimazione numerica al variare del numero di termini della serie di coseni che vengono mantenuti. Una analisi dettagliata del comportamento del COS e degli altri metodi relativamente alle opzioni Europee verrà condotta nella sezione 3.2.1.

Viene calcolato l'errore di pricing prendendo come riferimento il prezzo che il metodo fornisce utilizzando un N molto elevato, pari a 2^{12}. Poiché tale numero di punti comporta un elevato costo computazionale, si cerca di dimostrare che è possibile ottenere risultati accettabili anche con N minori.

Dall'osservazione dei grafici in scala logaritmica di Figura 2.6 è possibile verificare che la velocità del COS è esponenziale in N. In generale con un $N = 8$ si ottengono prezzi sufficientemente accurati.

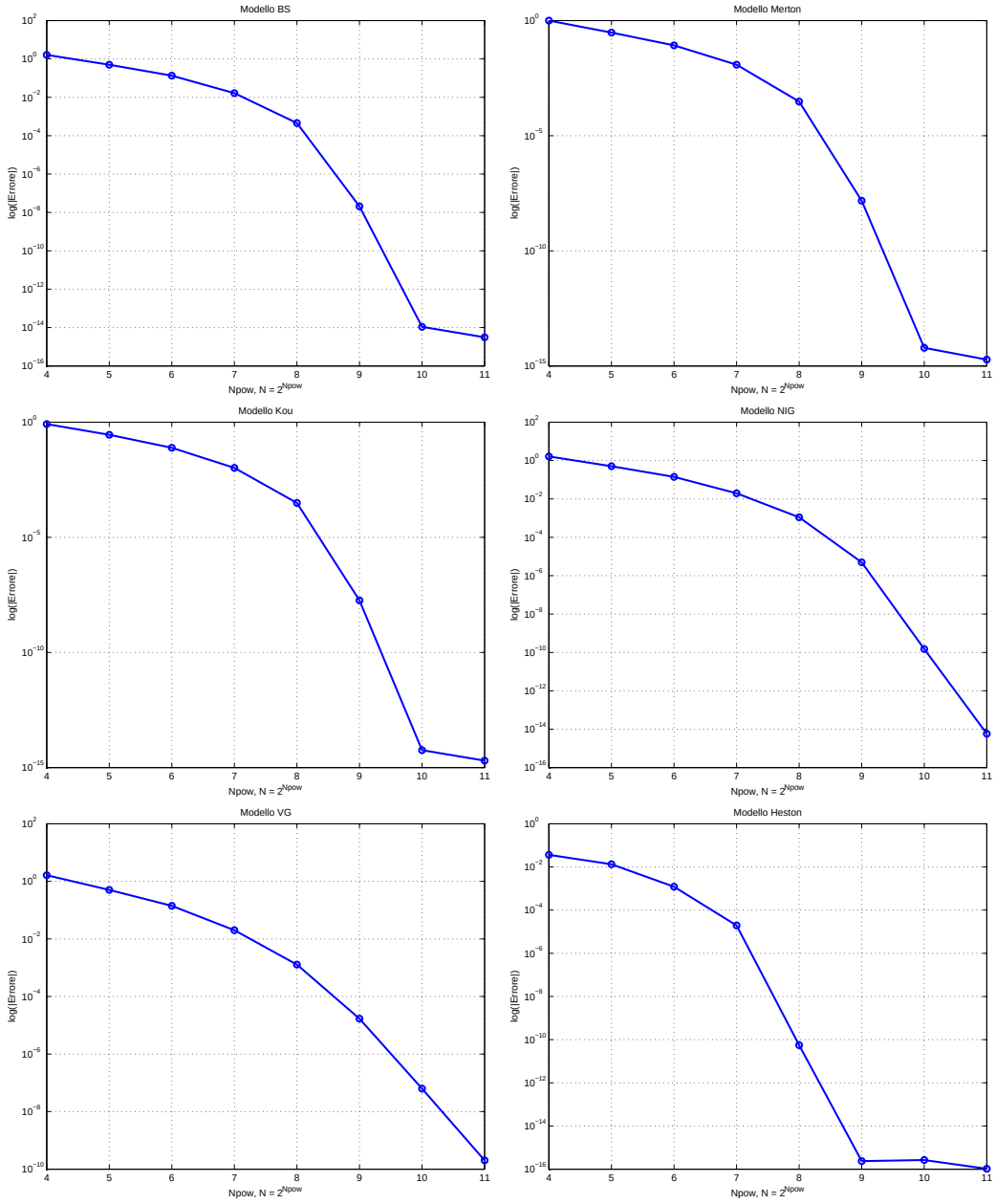


Figura 2.6: Convergenza dell'errore numerico per le opzioni barriera con metodo COS

2.6.5 Convergenza del COS per le Asiatiche

Il lavoro della sezione precedente, volto ad identificare l'ordine di convergenza del metodo COS nel caso di opzioni barriera, viene qui ripetuto applicato all'algoritmo del COS per opzioni Asiatiche. Anche in questo caso viene calcolato l'errore di pricing prendendo come riferimento il prezzo che il metodo fornisce utilizzando un N molto elevato, pari a 2^{10} e si cerca di verificare che è possibile ottenere risultati accettabili anche con N minori.

Dall'osservazione dei grafici in scala logaritmica di Figura 2.7 è si verifica che anche in questo caso la velocità del di convergenza del COS è esponenziale in N. In generale con un $N = 6$ il metodo ha già raggiunto la convergenza, come si vede dal fatto che la linea si interrompe, segno che l'errore è nullo e non è quindi possibile calcolarne il logaritmo.

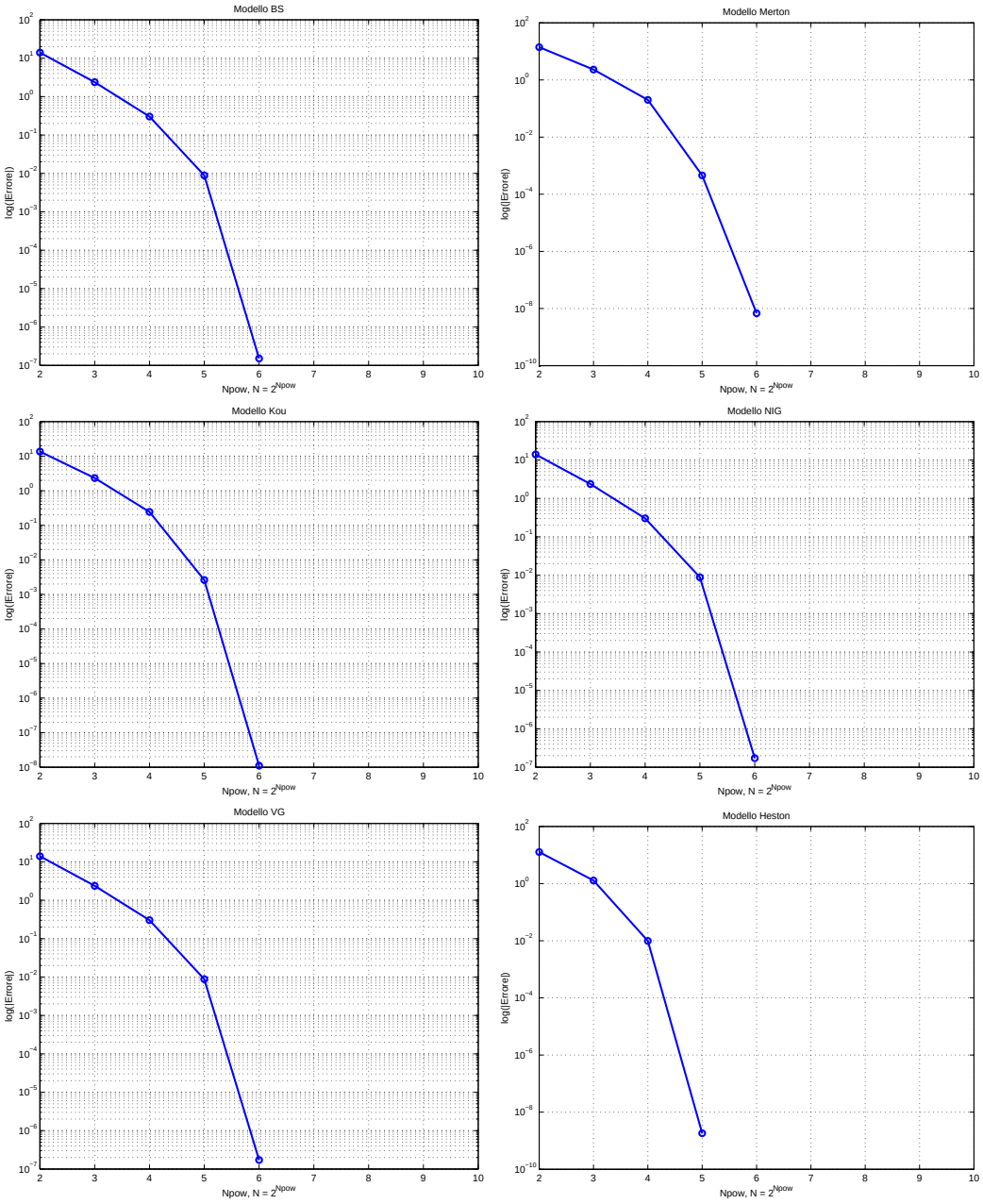


Figura 2.7: Convergenza dell'errore numerico per le opzioni barriera con metodo COS

Capitolo 3

La calibrazione dei modelli di pricing

3.1 Il problema della calibrazione

In questa sezione ci si occuperà della calibrazione dei modelli di pricing considerati. Prima di introdurre i risultati numerici verranno ricordate le principali grandezze in gioco.

3.1.1 Formalizzazione del problema

Modello di pricing Un modello di pricing delle opzioni è uno strumento per catturare le caratteristiche dei prezzi delle opzioni quotate sul mercato[1], relazionare tali prezzi in modo risk-neutral ed estrapolare il prezzo di strumenti non quotati sul mercato o comunque meno liquidi (opzioni esotiche).

Una prima richiesta da fare ad un modello di pricing delle opzioni è catturare lo stato delle opzioni nel mercato ad un dato istante: per riuscirvi i parametri del modello sono scelti in modo da riprodurre le principali caratteristiche dei prezzi, tramite la procedura della calibrazione. Di conseguenza è necessario sviluppare modelli e tecniche che in primo luogo consentano e in secondo luogo rendano efficace la calibrazione.

Calibrazione La calibrazione è la procedura con la quale si ricavano i parametri di un modello facendo il matching con i prezzi delle opzioni quotate. L'assunzione su cui si fonda è la presenza di un numero sufficiente di opzioni liquide trattate sul mercato, per le quali è possibile valutare il prezzo in modo rapido ed efficiente.

Mentre nel pricing si è interessati a calcolare i prezzi di un'opzione dati i parametri del modello, nella calibrazione si vuole ricavare i parametri che descrivono una dinamica risk-neutral per i prezzi osservati sul mercato. La calibrazione è quindi il

[1] Vengono utilizzate le opzioni vanilla, non sono le uniche quotate, sono però le più diffuse e semplici da prezzare, il che le rende adatte per la calibrazione

problema inverso associato a quello del pricing. In generale la soluzione del problema di calibrazione non è unica, in ogni caso si cercano i parametri che consentono la migliore approssimazione dei prezzi di mercato all'interno di una data classe di modelli.

Misura della distanza Lo scopo della calibrazione è minimizzare una misura dell'errore, cioé della distanza dei prezzi ottenuti con il modello di pricing e quelli di mercato. In questa tesi si è scelta la RMSE (root-mean-square-error) data da

$$RMSE(x) = \frac{1}{\sqrt{N}} \sqrt{\sum_{i=1}^{N} (C_i^* - C_i(x))^2} \tag{3.1}$$

dove N è il numero di opzioni quotate considerate, x è il vettore n-dimensionale dei parametri del modello, $C_i(x)$ sono i prezzi calcolati ipotizzando un modello per il sottostante, quindi fissando i parametri, e C_i^* sono i prezzi quotati sul mercato.

Il problema di calibrazione può quindi essere interpretato nel seguente modo

$$\min_{x \in A} f(x) \tag{3.2}$$

dove $A \subseteq \mathbb{R}^n$ è il dominio di ammissibilità dei parametri del modello.

3.1.2 Risoluzione del problema

Una volta impostato il problema occorre procedere alla sua risoluzione, il software `Matlab` mette a disposizione diverse routine finalizzate a questo obiettivo. Le tecniche possono essenzialmente dividersi in due gruppi, quelle che cercano un ottimo locale e quelle che cercano un ottimo globale. Ovviamente queste ultime sono computazionalmente molto più onerose. Nei paragrafi successivi verranno passate in rassegna alcune di queste tecniche applicandole al problema della calibrazione.

Vengono riportati i risultati relativi alla calibrazione con il dataset della Apple, dettagliato nell'Appendice A. Tra i vari metodi di pricing che verranno proposti nei capitoli successivi si è scelto di utilizzare il COS per quest'analisi poiché è il metodo computazionalmente più efficiente. Questa scelta verrà poi giustificata nella Sezione 3.2.

3.1.3 Prove numeriche

Come appena detto, scopo di questo capitolo è determinare il miglior metodo di calibrazione, tale obiettivo viene raggiunto tramite un'analisi sperimentale. Vengono quindi confrontati diversi metodi di ottimizzazione, tutti quanti già implementati in funzioni di `Matlab`, in particolare:

1. **Algoritmo per i minimi quadrati** Ha il minore costo computazionale tra tutti i metodi proposti ed è appositamente sviluppato per la funzione obiettivo RMSE. Ricerca un ottimo locale ed è implementata in `lsqcurvefit`.

2. **Algoritmo genetico** Un algoritmo genetico è un algoritmo euristico basato sul principio della selezione naturale. Il nome deriva dal fatto che gli algoritmi genetici attuano dei meccanismi concettualmente simili a quelli dei processi evolutivi. Gli algoritmi genetici introducono elementi di disordine per creare nuove soluzione a partire da una soluzione data nel tentativo di convergere a soluzioni ottime. Propone un esito aleatorio, dipendente dalla popolazione che si è fatta evolvere. Viene implementato nella funzione `ga`.

3. **Risolutore globale** Si tratta dell'algoritmo con di gran lunga il più elevato costo computazionale, ma è quello che fornisce i risultati migliori. Qui viene utilizzato per scopi di verifica della bontà delle soluzioni ottenute dagli altri metodi. Viene implementato in `gs`.

4. **Simulated Annealing** Il concetto di annealing (ricottura) deriva dalla scienza dei metalli, dov'è usato per descrivere il processo di eliminazione di difetti reticolari dai cristalli tramite una procedura di riscaldamento seguita da un lento raffreddamento. La tecnica mira a trovare un minimo globale quando si è in presenza di più minimi locali, con un costo computazionale minore della ricerca globale. Viene implementato nella funzione `sa`.

5. **Ricerca locale** L'algoritmo base di `Matlab` per la ricerca di un minimo locale. Richiede un ridotto costo computazionale ma produce risultati molto variabili a seconda del punto iniziale. Inoltre non è possibile inserire vincoli all'ottimizzazione. Viene implementato nella funzione `fminsearch`.

6. **Ricerca locale vincolata** Il comportamento è simile a quello della ricerca locale non vincolata, tuttavia è prevista l'aggiunta di bounds alle variabili indipendenti della funzione da ottimizzare. Il costo computazione è leggermente maggiore rispetto al caso semplice. Viene implementato nella funzione `fmincon`.

Punti iniziali e bounds Tutti i metodi appena citati vengono testati utilizzando i diversi modelli presentati nel Capitolo 1, vengono ora elencati i punti di partenza scelti per gli algoritmi che li richiedono e i bound imposti per la significatività dei parametri dei modelli.

Black&Scholes:

$$\sigma \in [0, 1]$$

$$\left[\sigma_0\right] = \left[0.2\right]$$

Merton:

$$\begin{aligned}\sigma &\in [0,1]\\ \lambda &\in [0,20]\\ \nu &\in [-5.5,5.5]\\ \delta &\in [0,0.5]\end{aligned}$$

$$\begin{bmatrix}\sigma_0\\ \lambda_0\\ \nu_0\\ \delta_0\end{bmatrix} = \begin{bmatrix}0.2\\ 2\\ 0\\ 0.2\end{bmatrix}$$

Kou:

$$\begin{aligned}\sigma &\in [0,1]\\ \lambda &\in [0,20]\\ \lambda^+ &\in [0,20]\\ \lambda^- &\in [0,20]\\ p &\in [0,1]\end{aligned}$$

$$\begin{bmatrix}\sigma_0\\ \lambda_0\\ \lambda_0^+\\ \lambda_0^-\\ p_0\end{bmatrix} = \begin{bmatrix}0.1\\ 1\\ 4\\ 4\\ 0.5\end{bmatrix}$$

NIG:

$$\begin{aligned}\sigma &\in [0,1]\\ \theta &\in [-2,2]\\ \kappa &\in [0,1]\end{aligned}$$

$$\begin{bmatrix}\sigma_0\\ \kappa_0\\ \theta_0\end{bmatrix} = \begin{bmatrix}0.1\\ 0.1\\ 0.1\end{bmatrix}$$

VG:

$$\begin{aligned}\sigma &\in [0,1]\\ \theta &\in [-2,2]\\ \kappa &\in [0,1]\end{aligned}$$

$$\begin{bmatrix}\sigma_0\\ \kappa_0\\ \theta_0\end{bmatrix} = \begin{bmatrix}0.1\\ 0.1\\ 0.1\end{bmatrix}$$

Heston:

$$V \in [0,1]$$
$$\theta \in [0,1]$$
$$\kappa \in [0,5]$$
$$\nu \in [0,0.5]$$
$$\rho \in [-1,1]$$

$$\begin{bmatrix} V_0 \\ \theta_0 \\ \kappa_0 \\ \nu_0 \\ \rho_0 \end{bmatrix} = \begin{bmatrix} 0.02 \\ 0.02 \\ 0.1 \\ 0.1 \\ 0 \end{bmatrix}$$

In Tabella 3.1 sono riportati i tempi macchina relativi ai diversi metodi usati e i valori dell'RMSE. Tutti tempi della CPU sono riportati in millisecondi e il computer utilizzato per tutti i test di questo capitolo ha un Intel(R) Core(R) i7 Quad-Core mentre il codice è scritto in Matlab R2013b.

	Black&Scholes		Merton		Kou	
Metodo	Tempo	RMSE	Tempo	RMSE	Tempo	RMSE
lsq	1.1458	1.9082	1.0735	0.12517	1.3686	0.11110
ga	1.7788	1.9082	2.8672	0.18466	4.7730	0.95151
gs	6.1613	1.9082	7.6554	0.23888	70.830	0.11212
sa	1.7351	1.9082	7.3918	0.50196	12.798	0.44716
fmins	0.078409	1.9082	0.53411	1.9082	2.4574	0.32269
fminc	0.50945	1.9082	2.8916	0.12517	8.6543	0.11033

	NIG		VG		Heston	
Metodo	Tempo	RMSE	Tempo	RMSE	Tempo	RMSE
lsq	0.18576	0.26188	0.14884	0.34418	1.9117	0.29265
ga	4.7498	0.30874	2.6590	0.36127	6.9356	0.32402
gs	51.602	0.26188	46.416	0.34418	178.95	0.29307
sa	9.0751	0.51098	5.8538	0.60572	20.231	0.69709
fmins	1.4492	0.90045	0.24065	1.6770	0.42905	1.9081
fminc	0.47143	0.26188	0.50261	0.34418	11.873	0.29264

Tabella 3.1: Confronto tempi computazionali ed RMSE diversi solutori

Il metodo `gs` ha costi computazionali talmente elevati da non renderlo un metodo realisticamente applicabile. Il metodo del simulated annealing `sa` ha tempi più ridotti e funzioni obiettivo sufficientemente accurate e rappresenta quindi una valida alternativa al `gs`. Il metodo genetico `ga` produce risultati stocastici ed in generale valori peggiori della funzione obiettivo, di conseguenza viene scartato. Il metodo `fminsearch` è molto rapido ma non ottiene risultati soddisfacenti in termini di minimizzazione dell'errore. Dall'altro lato `fmincon` riesce quasi sempre a ottenere una buona minimizzazione con un costo computazionale solo lievemente superiore. Spicca sugli altri metodi l'algoritmo `lsqcurvefit`, appositamente scritto per la minimizzazione ai minimi quadrati, il quale fornisce quasi sempre il miglior fitting ma con costi computazionali irrisori. Alla luce di queste considerazioni, nel seguito,

ogni volta che si effetturà una calibrazione, la parte di ottimizzazione sarà affidata alla routine `lsqcurvefit`.

Per i parametri ottenuti nella calibrazione si faccia riferimento alla Tabella 3.3. Per quanto riguarda questa particolare serie storica si possono trarre alcune conclusioni, tuttavia la loro validità rimane circoscritta al set di prezzi utilizzati. Si intravede che il modello Kou sembra essere il migliore per spiegare i prezzi di mercato, seguito dal modello di Merton. Sempre nell'ambito dei modelli di Lévy si vede che anche VG e NIG performano bene, però non come le loro controparti Jump-Diffusion. Il costo computazionale è minore, ma il fitting lievemente peggiore. Il peggior modello è, come ci si poteva aspettare, quello di Black&Scholes, a cui manca la flessibilità necessaria per adattarsi ai prezzi di mercato osservati.

3.2 Confronto tra i metodi utilizzati

Nella sezione precedente si è mostrato come la scelta della routine `lsqcurvefit` sia la migliore in termini di accuratezza e costo computazionale. Dato che ogni passo degli algoritmi di minimizzazione richiede la valutazione di un'opzione europea, in questa sezione viene ora realizzato un confronto tra alcuni dei metodi numerici per il pricing introdotti nel Capitolo 2. Per ciascun metodo viene analizzata la performance nella risoluzione del problema di valutazione di un'opzione europea in termini di costo computazionale ed accuratezza.

3.2.1 Analisi di convergenza

In questa sezione vengono effettuati una varietà di test numerici per valutare l'efficienza e l'accuratezza dei metodi proposti. L'attenzione è focalizzata sulle opzioni Call Europee; per la dinamica del sottostante vengono utilizzati i diversi processi introdotti nel Capitolo 1.

I metodi utilizzati sono accomunati dall'introduzione di errori numerici dovuti all'approssimazione su di una griglia, ci si propone di capire quanto questa griglia debba essere fitta per rendere trascurabili gli errori. Nel seguito ci si soffermerà su tre metodi (Carr&Madan, CONV e COS) poiché risultano i tre più veloci ed accurati per la valutazione di opzioni europee. Sebbene dalle analisi emerga che il metodo COS è migliore degli altri dal punto di vista della calibrazione, nell'interfaccia grafica descritta nel Capitolo 4 è comunque possibile scegliere ciascuno dei metodi implementati.

Lo scopo di questa analisi è individuare, per ciascuno dei metodi proposti, il valore di N, il numero di punti della griglia per Carr&Madan e CONV e il numero di termini nell'espansione di coseni per COS, tale da assicurare adeguata accuratezza con il minor costo computazionale possibile. L'analisi si basa sull'osservazione empirica, utilizzando dei grafici che riportano in ordinata il logaritmo dell'errore di pricing al variare del numero di punti della griglia di discretizzazione. Come ci si aspetta l'errore è decrescente all'aumentare del numero di punti, tuttavia i metodi hanno rapporti di convergenza molto diversi.

In Figura 3.1 è riportato l'andamento degli errori di pricing al variare del numero di punti della griglia di discretizzazione per ciascuno dei modelli presentati nel Capitolo 1. L'opzione che viene prezzata è una Call Europea con parametri: $S_0 = 100$, $K = 100$, $r = 0.0367$, $T = 1$, $d = 0$. Per il modello di Black&Scholes si è scelto $\sigma = 0.17801$; per Merton $\sigma = 0.17801$, $\lambda = 4$, $\mu = 0.05$, $\delta = 0.04$; per Kou $\sigma = 0.17801$, $\lambda = 4$, $\lambda^+ = 20$, $\lambda^- = 4$, $p = 0.80$; per NIG e VG $\sigma = 0.17801$, $\kappa = 0.1$, $\theta = 0.1$; per Heston $V_0 = 0.17801$, $\theta = 0.02$, $\kappa = 0.1$; $\nu = 0.1$, $\rho = 0.5$.

L'errore di pricing viene calcolato prendendo come riferimento il prezzo che il metodo fornisce utilizzando un N molto elevato, pari a 2^{16}. Tale numero di punti comporta un elevato costo computazionale, ci si aspetta quindi di ottenere risultati accettabili anche con N minori. In effetti, almeno per il COS, si osserva che a partire da 2^5 termini, il prezzo prodotto non cambia più, questo spiega l'interruzione

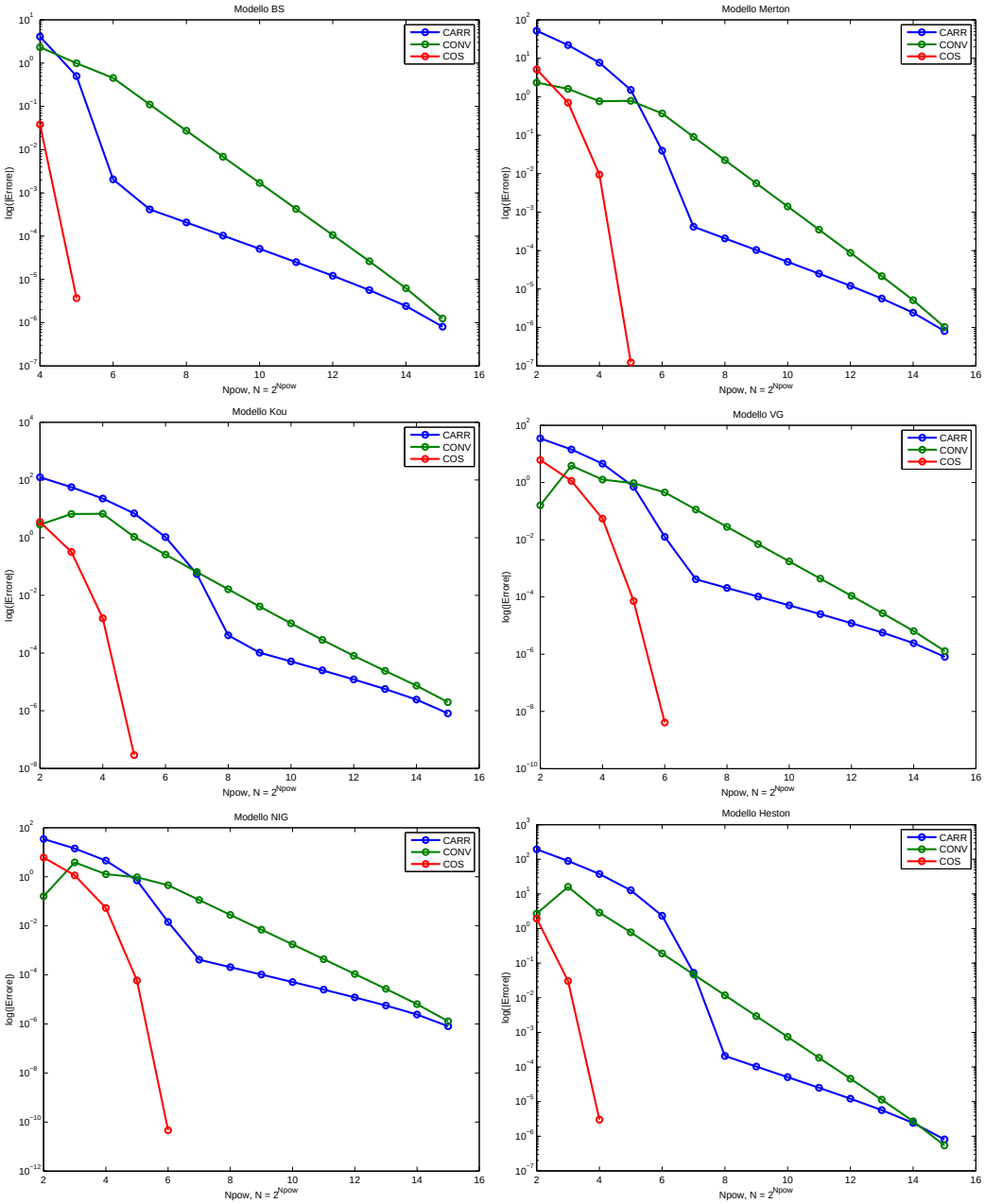


Figura 3.1: Confronto tra gli errori nei diversi metodi.

della linea nei grafici, essendo l'errore pari a 0 non è possibile rappresentarne il logaritmo. Dall'osservazione dei grafici si può osservare che in ogni caso il COS è il metodo che converge prima, nel peggiore dei casi richiede $N = 2^6$ per arrivare a convergenza completa. In effetti si può verificare, si veda [6], che il COS è un metodo a convergenza esponenziale. Negli altri casi per avere un errore sufficientemente ridotto, e.g. dell'ordine di 10^{-6}, è necessario utilizzare $N = 2^{14}$.

3.2.2 Tempi di esecuzione

Nella Tabella 3.2 vengono riportati i tempi macchina impiegati per l'esecuzione degli algoritmi di pricing al variare della precisione dell'approssimazione. Tutti i tempi computazionali indicati, in millisecondi, sono determinati mediando su 10 esperimenti e sono stati ottenuti su un computer con processore Intel(R) Core(R) i7 Quad-Core, il codice è scritto in Matlab R2013b. L'opzione prezzata è una Call europea e i parametri sono gli stessi di quelli utilizzati per ricavare la Figura 3.1.

$\log_2(N)$	*Black&Scholes*			*Merton*			*Kou*		
	CarrM	Cos	Conv	CarrM	Cos	Conv	CarrM	Cos	Conv
5	1.5635	0.52220	0.53508	1.6968	0.70437	0.76567	1.6316	0.65987	0.70251
6	1.5620	0.53639	0.52626	1.7382	0.70288	0.76773	1.6861	0.68054	0.71529
7	1.6198	0.56050	0.56820	1.8739	0.78121	0.82655	1.7639	0.71473	0.75826
8	1.9465	0.68306	0.66225	2.2011	0.86509	0.91080	2.0418	0.83318	0.85799
9	2.3621	0.80985	0.78499	2.5600	1.0815	1.0744	2.4703	1.0227	1.0330
10	3.2526	1.1365	1.0744	3.4491	1.4498	1.4299	3.4539	1.3666	1.3403
11	5.0292	1.7309	1.5810	5.3918	2.1851	2.1435	5.2175	2.0699	2.0025
12	8.1748	2.8767	2.5787	9.1089	3.6748	3.2563	8.4819	3.3906	2.9644
13	15.226	5.1305	4.4428	16.294	6.4554	5.4596	15.612	6.1593	5.2818
14	26.377	6.3266	6.7813	30.100	7.9474	8.4506	27.500	7.3151	8.0240
15	55.246	12.050	13.644	56.452	15.152	16.527	55.095	13.956	15.975
16	115.81	25.189	29.314	125.44	32.265	41.084	118.29	28.509	34.880

$\log_2(N)$	*NIG*			*VG*			*Heston*		
	CarrM	Cos	Conv	CarrM	Cos	Conv	CarrM	Cos	Conv
5	1.6376	0.64746	0.68478	1.6537	0.66757	0.68217	1.6397	0.56634	0.58206
6	1.7384	0.70960	0.75107	1.6970	0.68623	0.70284	1.6140	0.58752	0.60086
7	1.8969	0.77057	0.75830	2.2445	0.73297	0.73493	1.7585	0.66178	0.67018
8	2.0907	0.84083	0.82646	2.0785	0.85958	0.84615	2.0770	0.83593	0.84106
9	2.5902	1.0584	1.0440	2.6324	1.1095	1.0661	2.7300	1.1500	1.1368
10	3.6880	1.5004	1.3518	3.6700	1.5446	1.4298	3.9519	1.7453	1.7222
11	5.5642	2.3217	2.0803	5.5331	2.4811	2.2509	6.4411	2.9061	2.9336
12	8.8587	3.5275	3.3328	9.5264	4.0764	3.3523	10.915	5.1735	5.1290
13	16.205	6.1236	5.6249	17.173	7.5109	5.9645	20.593	9.7681	9.3411
14	29.913	8.4564	8.8545	29.016	8.9117	9.8807	32.869	12.053	12.821
15	57.198	15.683	16.646	58.745	17.910	18.935	65.266	22.722	25.983
16	123.06	30.803	35.287	125.66	35.819	40.204	140.71	47.711	54.387

Tabella 3.2: Confronto tempi computazionali diversi metodi in msec.

Dall'osservazione della tabella si evince che a parità di complessità della griglia il metodo di Carr&Madan è il più lento, di un fattore che sembra appartenere all'intervallo $[2, 3]$, mentre COS e CONV appaiono confrontabili. Se però si tiene

anche conto dell'effettivo numero di punti che occorrono per ottenere lo stesso livello di accuratezza si verifica che il COS è migliore di Carr&Madan e CONV di un fattore rispettivamente circa pari a 40 e 10. Si suggerisce quindi l'applicazione del metodo COS durante la procedura di calibrazione.

3.2.3 Accuratezza della calibrazione

Mentre nelle sottosezioni precedenti si è valutato un singolo derivato, ora si utilizzeranno i tre metodi di pricing, insieme all'algoritmo `lsqcurvefit` per effettuare la calibrazione. La Tabella 3.3 riporta i parametri ottimali ottenuti dalla calibrazione. Con tali parametri è stato riapplicato il metodo di pricing per fornire dei prezzi che, confrontati con quelli di mercato, consentono di individuare l'RMSE ottimale, riportato nelle tabelle.

L'algoritmo di ottimizzazione, come annunciato nel Capitolo 3, e già ricordato, è quello implementato nella routine `lsqcurvefit`.

Il confronto seguente ha lo scopo di accertare la qualità della calibrazione fornita dai diversi metodi al variare dei modelli che descrivono la dinamica del sottostante. Vengono dapprima confrontati i parametri ottimali prodotti, ci si aspetta che siano molto simili per uno stesso modello al variare del metodo di calibrazione. In effetti è così, con piccole fluttuazioni dei parametri che però producono valori dell'RMSE molto simili; questo fenomeno è dovuto alla forma della funzione obiettivo, che presenta molti ottimi locali ravvicinati: piccole differenze nell'implementazione nei metodi possono portare l'algoritmo di ottimizzazione a protendere verso un minimo piuttosto che un altro.

Si noti che, per il caso Black&Scholes è stato anche preso in considerazione l'algoritmo dell'Albero binomiale.

(a) Black&Scholes

Modello	Albero	Carr&Madan	CONV	COS
σ	0.22875	0.22895	0.22983	0.22897
$RMSE$	1.9266	1.9134	1.9082	1.9082

(b) Merton

Modello	Carr&Madan	CONV	COS
σ	0.18860	0.18981	0.18872
λ	0.12179	0.12133	0.12174
θ	-0.40260	-0.40315	-0.40219
δ	0.50000	0.50000	0.50000
$RMSE$	0.12387	0.12509	0.12517

(c) Kou

Modello	Carr&Madan	CONV	COS
σ	0.19550	0.17478	0.18722
λ	0.075332	1.3971	0.17146
λ^+	5.7364	19.383	4.8721
λ^-	0.16036	2.3432	0.82153
p	0.58743	0.89111	0.56891
$RMSE$	0.10509	0.12082	0.11110

(d) NIG

Modello	Carr&Madan	CONV	COS
σ	0.24046	0.24113	0.24041
κ	-0.12161	-0.12271	-0.12172
θ	0.57199	0.55468	0.56853
$RMSE$	0.25843	0.26392	0.26188

(e) VG

Modello	Carr&Madan	CONV	COS
σ	0.23760	0.23829	0.23754
κ	-0.12690	-0.12827	-0.12712
θ	0.38793	0.37673	0.38560
$RMSE$	0.34087	0.34762	0.34418

(f) Heston

Modello	Carr&Madan	CONV	COS
V_0	0.035644	0.031867	0.034580
θ	0.51441	0.99999	0.59187
κ	0.16954	0.096079	0.15109
ν	0.50000	0.49986	0.50000
ρ	-0.30803	-0.30770	-0.30815
$RMSE$	0.28935	0.29473	0.29265

Tabella 3.3: Parametri ottimali ed RMSE corrispondente.

3.2.4 Confronto grafico dei prezzi

Nelle sottosezioni seguenti si forniscono dei grafici dai quali è possibile dedurre la bontà di ciascun metodo ai fini della calibrazione. Tutti i grafici riportati sono relativi alla serie storica di Apple[2] con maturity Gennaio 2015.

Al variare del metodo di pricing le differenze sono minime, come è lecito aspettarsi. Sono invece più marcate le differenze al variare del modello utilizzato, con Black&Scholes visivamente in difficoltà nel riprodurre i prezzi di mercato: produce infatti il massimo scostamento dei prezzi quanto più ci si allontana dallo strike corrispondente all'opzione in the money. I modelli di Levy ad attività infinita sono dei buoni candidati poiché sono veloci da calibrare e producono errori ridotti. Il miglior modello è quello di Kou, dai grafici si vedono confermati i dati numerici delle tabelle che lo piazzano al primo posto come capacità di spiegare i prezzi osservati sul mercato. Anche il modello a volatilità stocastica di Heston è molto efficace ed è comparabile al modello di Merton, intermedio tra Kou e i Lévy ad attività infinita.

Segue l'elenco delle figure riportate

Albero In Figura 3.2 sono confrontati i prezzi di mercato ed i prezzi prodotti dal modello e gli errori assoluti e relativi dei prezzi del modello rispetto a quelli quotati relativamente al metodo dell'Albero binomiale.

Carr&Madan In Figura 3.3 e Figura 3.4 sono confrontati i prezzi di mercato ed i prezzi prodotti dal modello e gli errori assoluti e relativi dei prezzi del modello rispetto a quelli quotati relativamente al metodo di Carr&Madan.

CONV In Figura 3.5 e Figura 3.6 sono confrontati i prezzi di mercato ed i prezzi prodotti dal modello e gli errori assoluti e relativi dei prezzi del modello rispetto a quelli quotati relativamente al metodo CONV.

COS In Figura 3.7 e Figura 3.8 sono confrontati i prezzi di mercato ed i prezzi prodotti dal modello e gli errori assoluti e relativi dei prezzi del modello rispetto a quelli quotati relativamente al metodo COS.

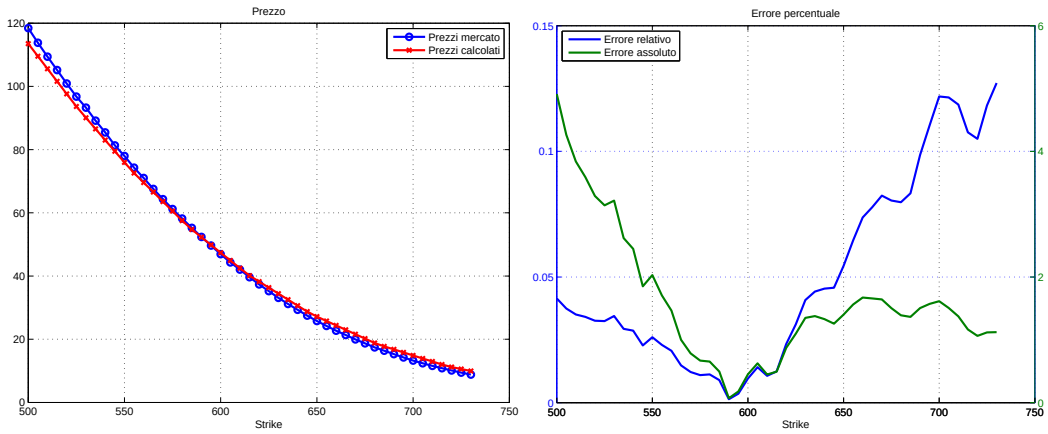
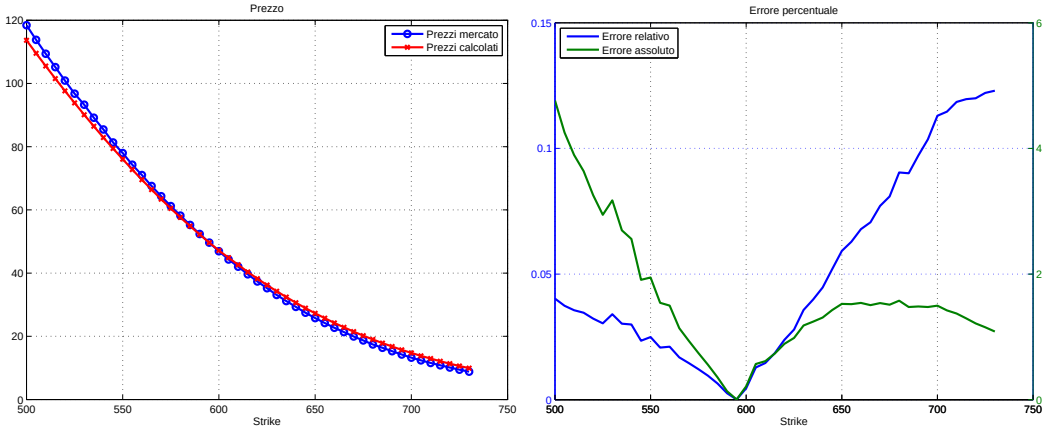
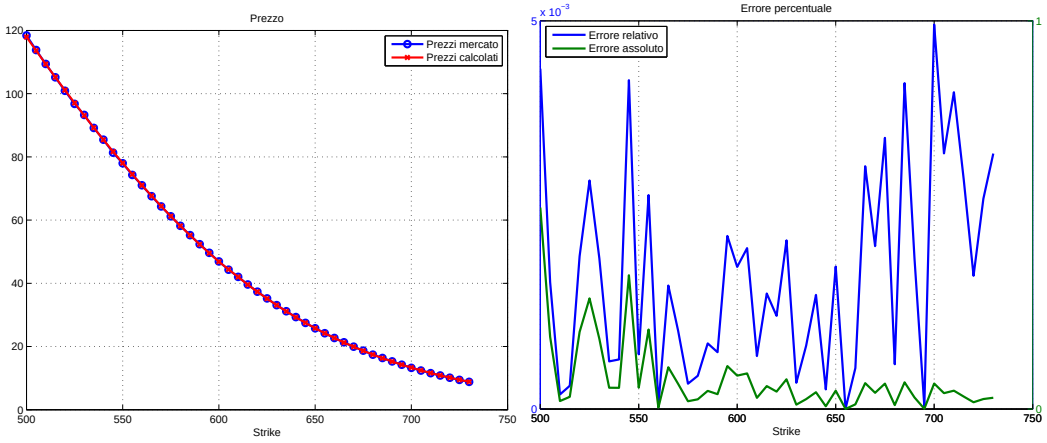


Figura 3.2: Confronto prezzi per il metodo dell'Albero binomiale

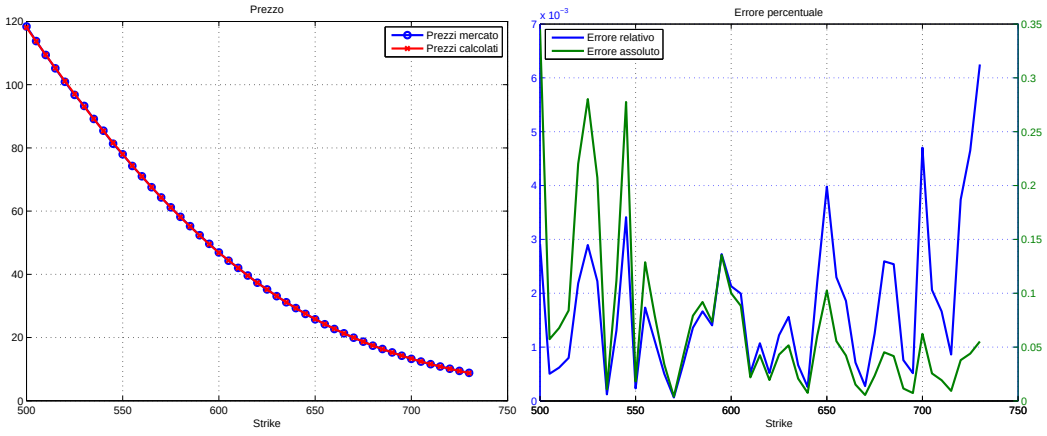
[2]Ticker AAPL



(a) Black&Scholes

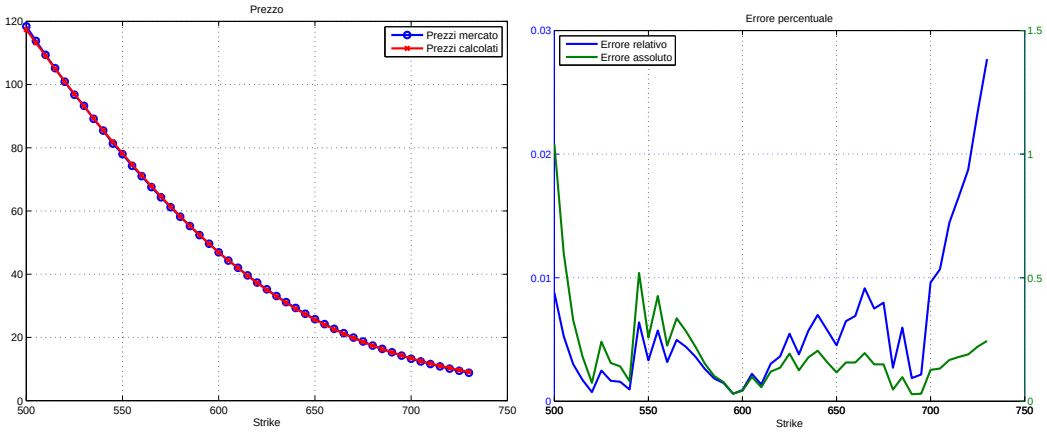


(b) Merton

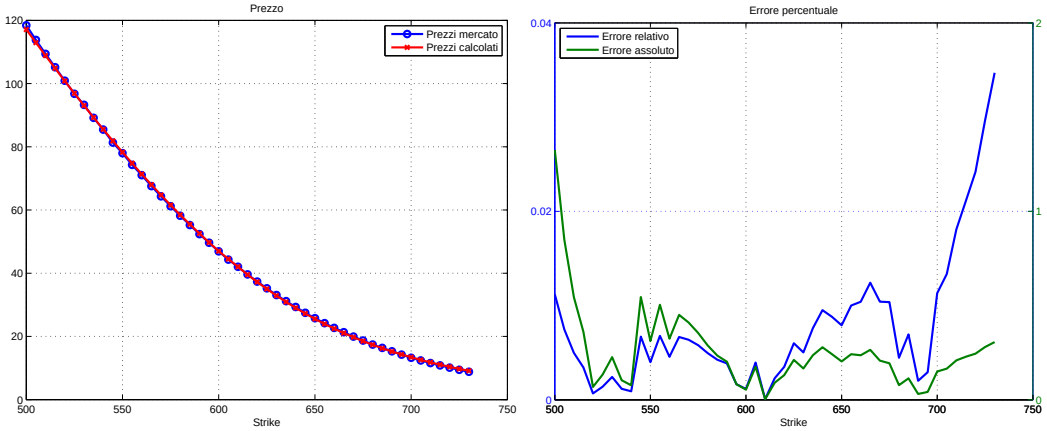


(c) Kou

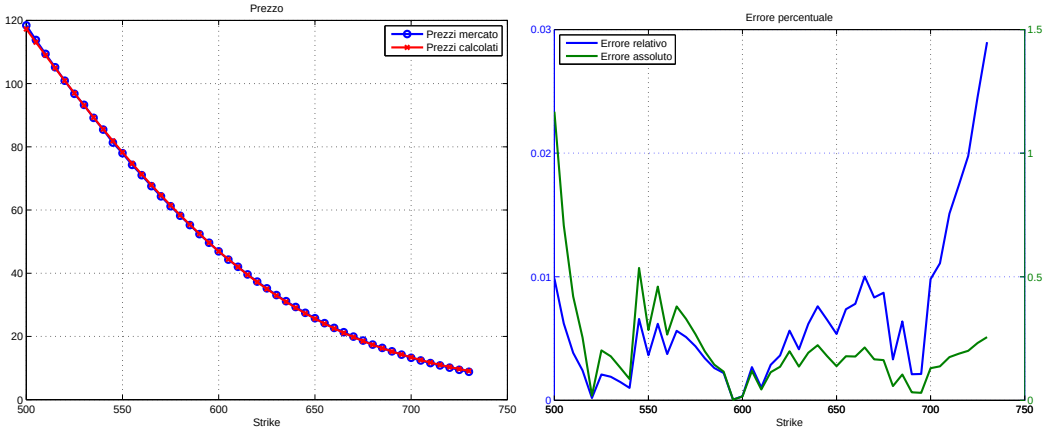
Figura 3.3: Confronto prezzi per il metodo Carr&Madan, parte 1



(a) NIG

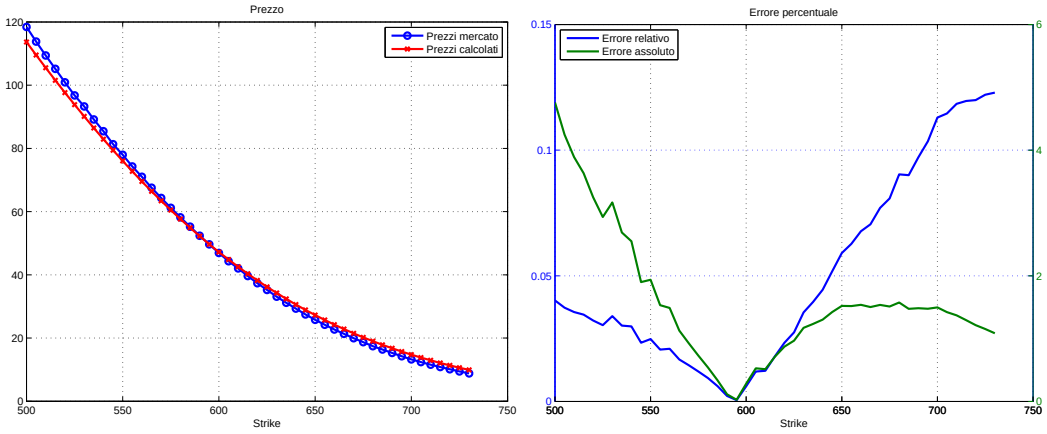


(b) VG

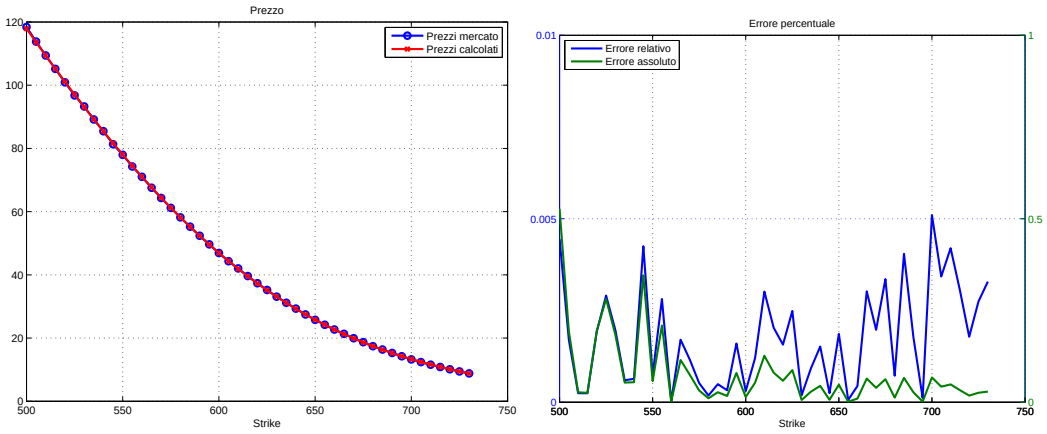


(c) Heston

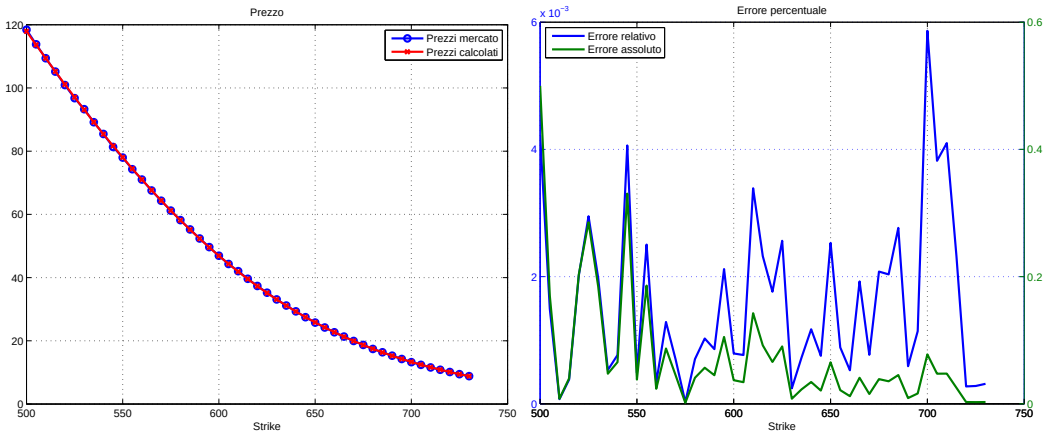
Figura 3.4: Confronto prezzi per il metodo Carr&Madan, parte 2



(a) Black&Scholes

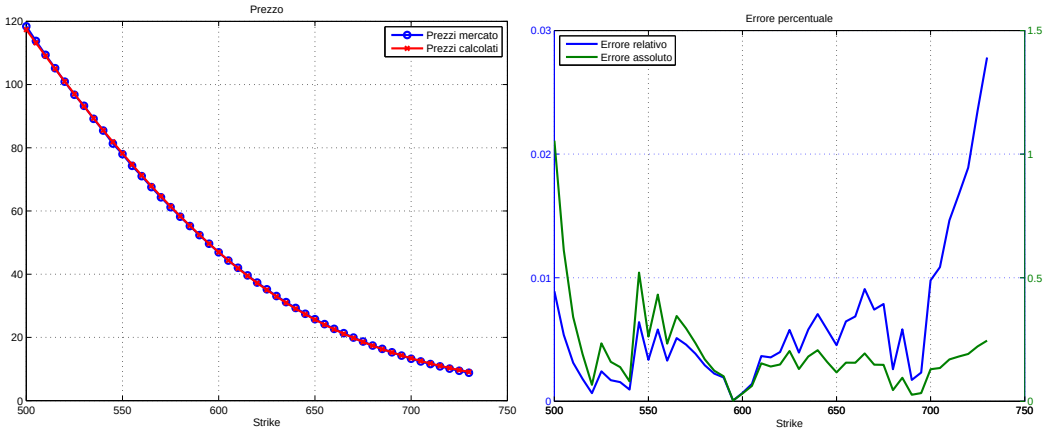


(b) Merton

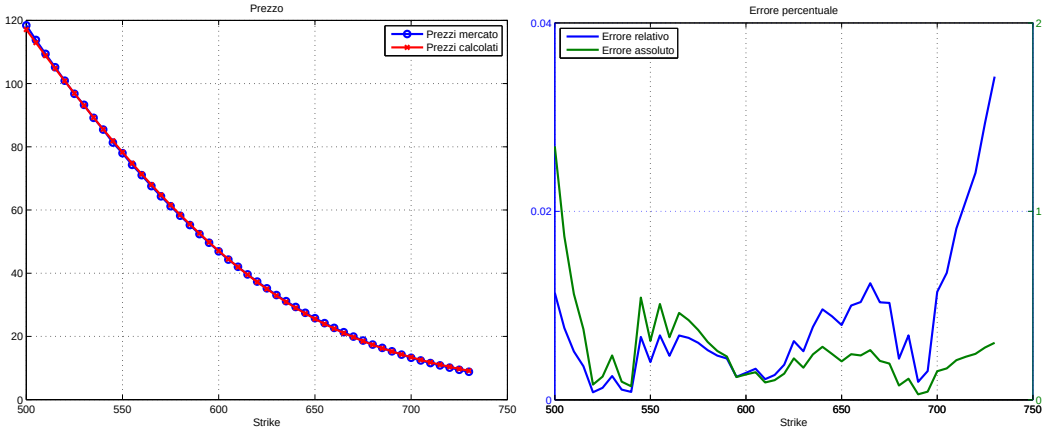


(c) Kou

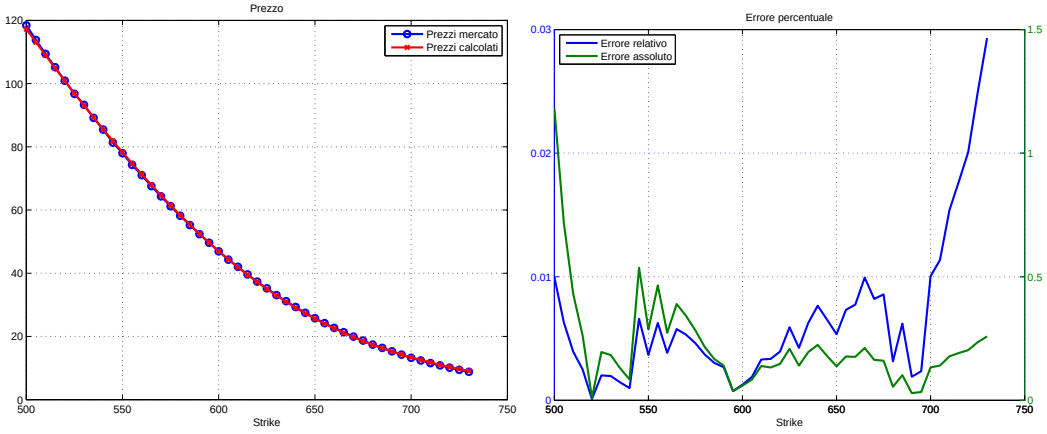
Figura 3.5: Confronto prezzi per il metodo CONV, parte 1



(a) NIG

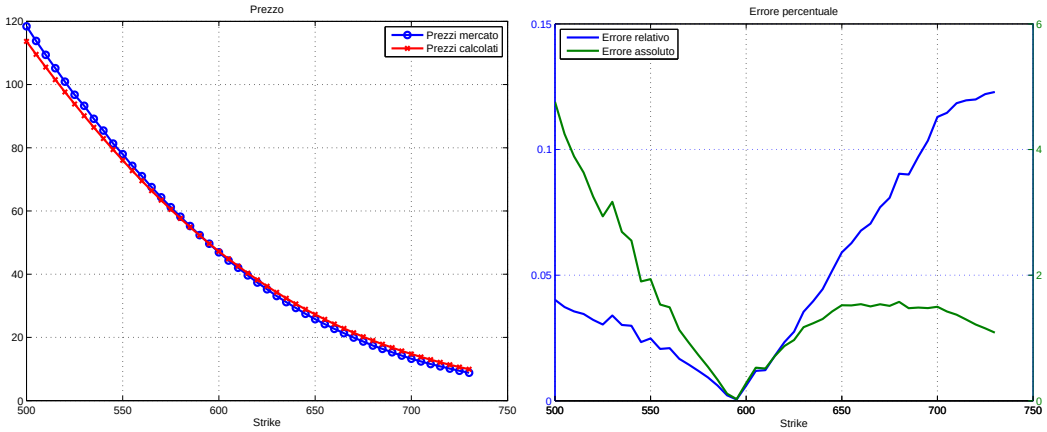


(b) VG

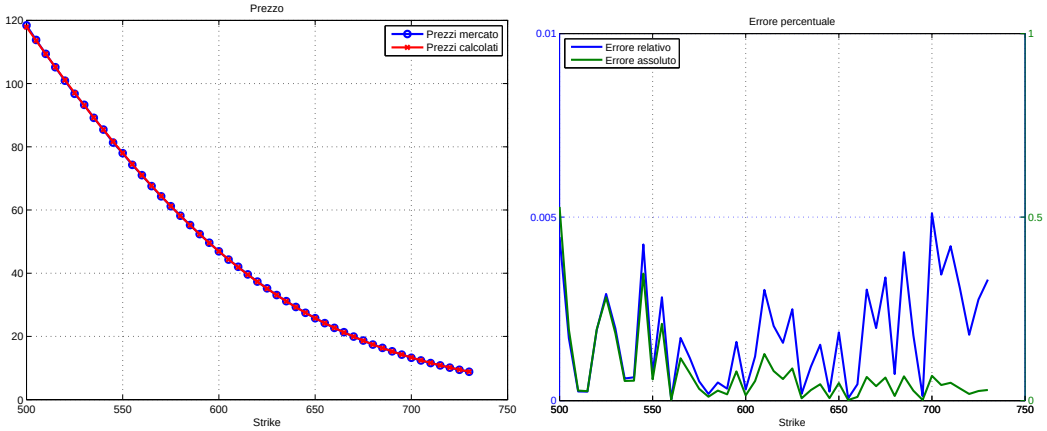


(c) Heston

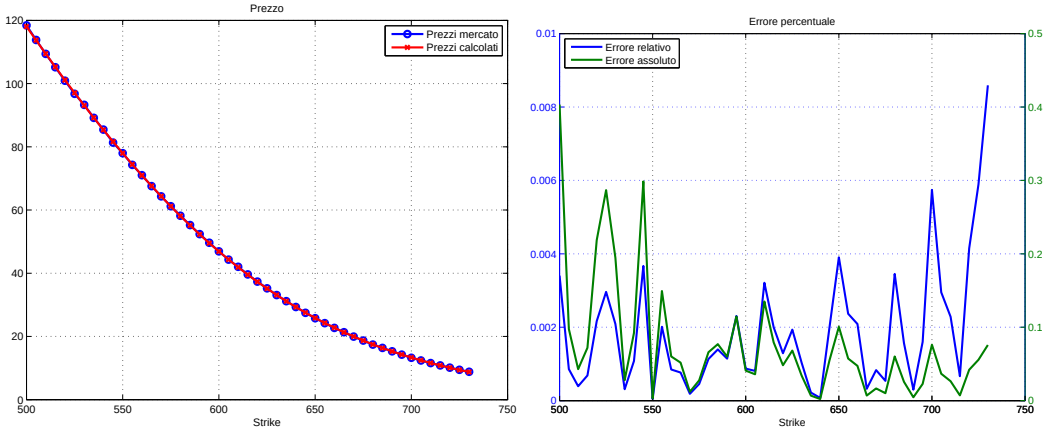
Figura 3.6: Confronto prezzi per il metodo CONV, parte 2



(a) Black&Scholes

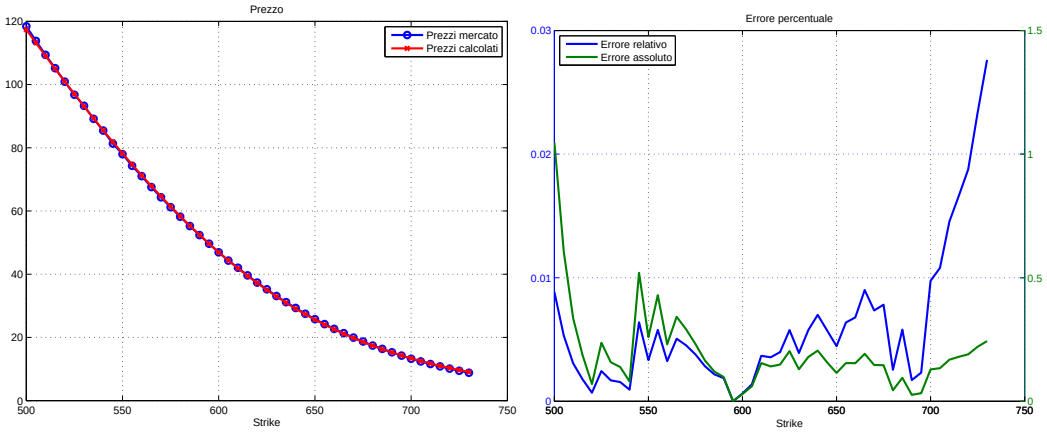


(b) Merton

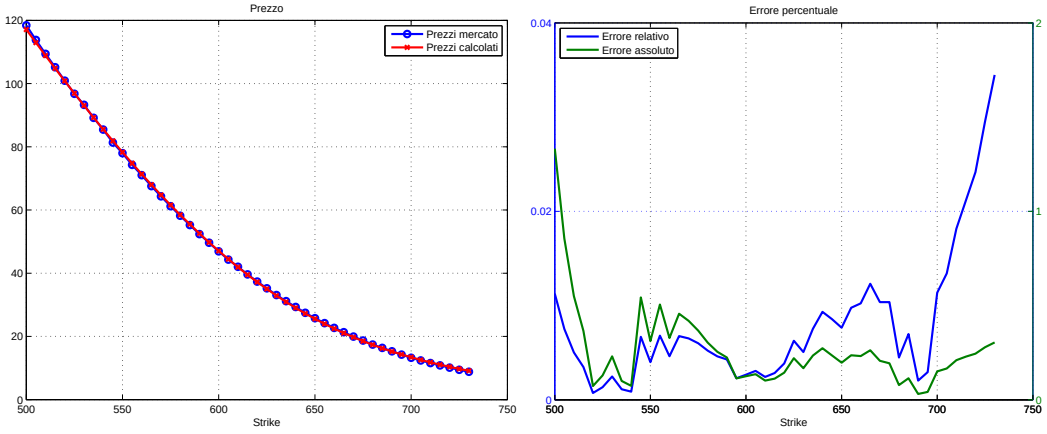


(c) Kou

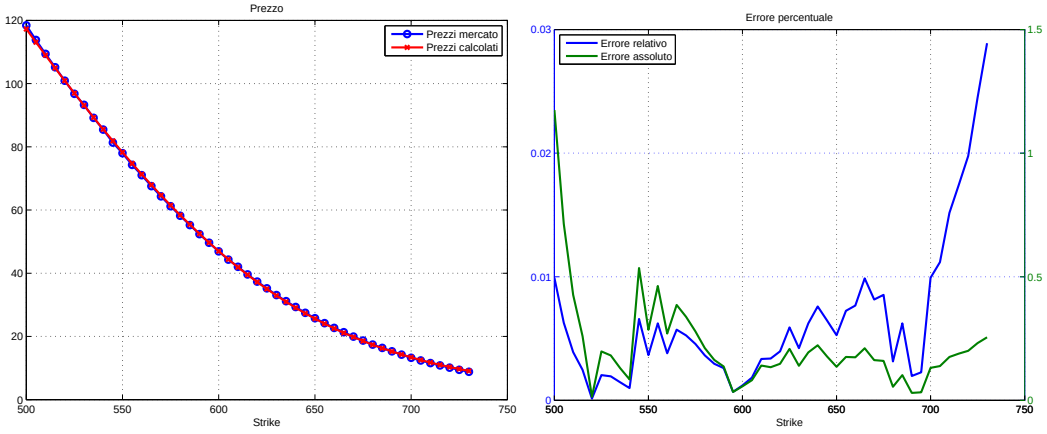
Figura 3.7: Confronto prezzi per il metodo COS, parte 1



(a) NIG



(b) VG



(c) Heston

Figura 3.8: Confronto prezzi per il metodo COS, parte 2

3.2.5 Confronto volatilità implicite

Volatilità implicita Spesso i prezzi delle opzioni sono rappresentati in termini della loro volatilità implicita, definita come il numero σ^* che soddisfa la relazione:

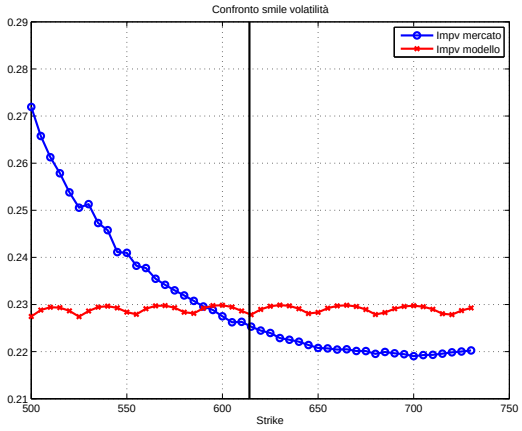
$$C(S_t, K, r, \sigma^*) = C^*(T, K) \tag{3.3}$$

dove $C = C(S_t, K, r, \sigma)$ è ottenuto dalla formula di Black&Scholes e $C^*(T, K)$ sono i prezzi quotati sul mercato. In questo senso la formula di Black&Scholes viene utilizzata per tradurre i prezzi di mercato in una rappresentazione in termini della volatilità implicita.

Un confronto interessante, sebbene non del tutto rilevante ai fini della calibrazione dei prezzi, è quello sulla capacità del modello di riprodurre gli smile di volatilità del mercato. Tale caratteristica sarebbe di fondamentale importanza se si volessero ad esempio prezzare derivati scritti sulla volatilità.

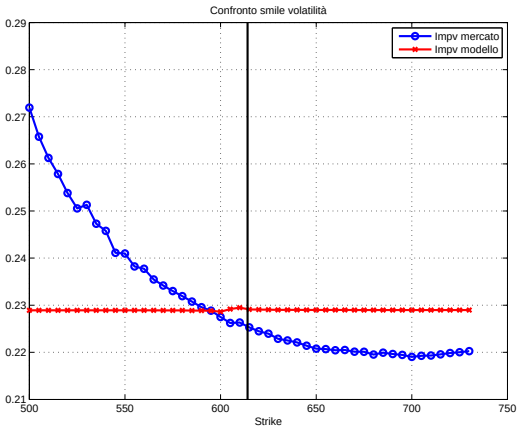
Nel seguito vengono riportati dei grafici che confrontano la volatilità implicita dei prezzi ottenuti applicando i vari modelli utilizzando i parametri ottimali forniti dalla calibrazione con quella dei prezzi di mercato.

Performance dei modelli Come ci si può aspettare la volatilità prodotta dal modello di Black&Scholes è costante e pari a σ, in effetti, come detto precedentemente, questa è una delle caratteristiche del modello. Gli altri processi, invece, riescono tutti a produrre smile realistici, con Kou che produce lo smile più simile a quello del mercato, grazie alla sua flessibilità che consente di tarare separatamente i parametri che controllano salti positivi e negativi.

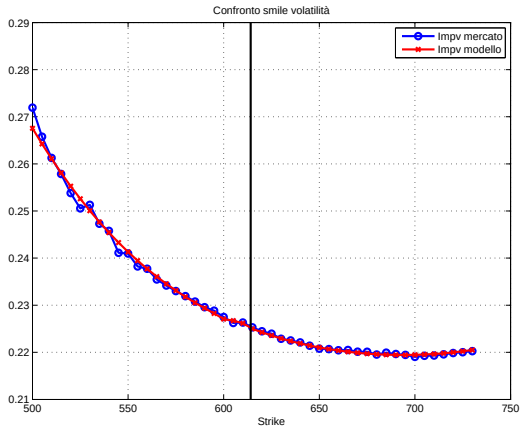


(a) Black&Scholes

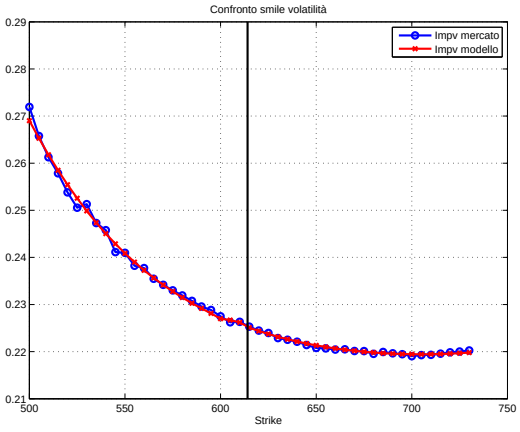
Figura 3.9: Confronto volatilità implicite per il metodo dell'albero binomiale



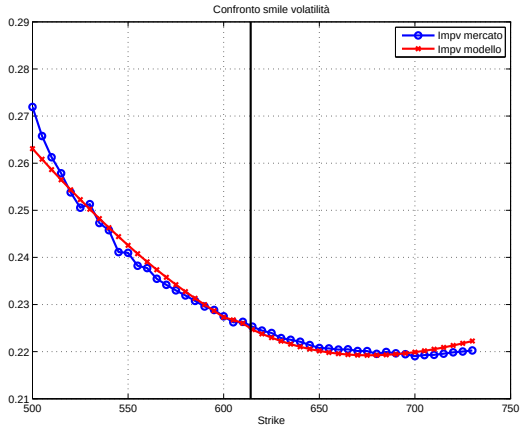
(a) Black&Scholes



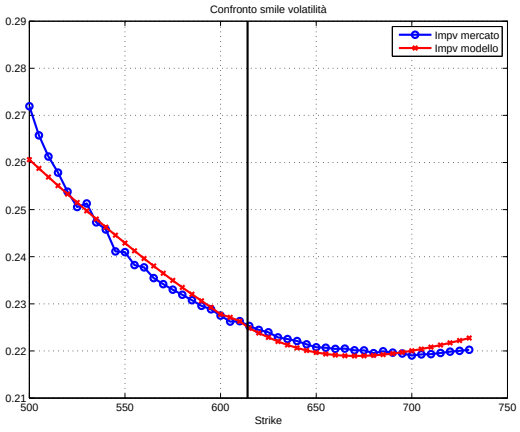
(b) Merton



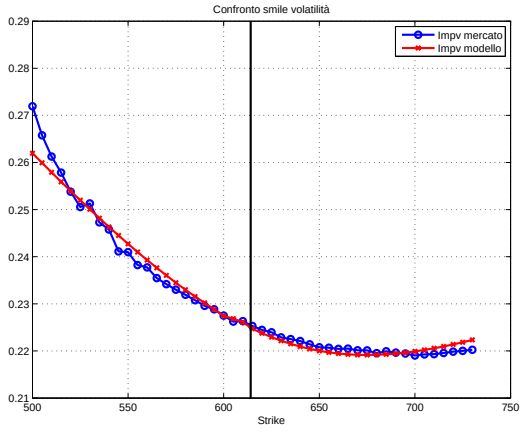
(c) Kou



(d) NIG

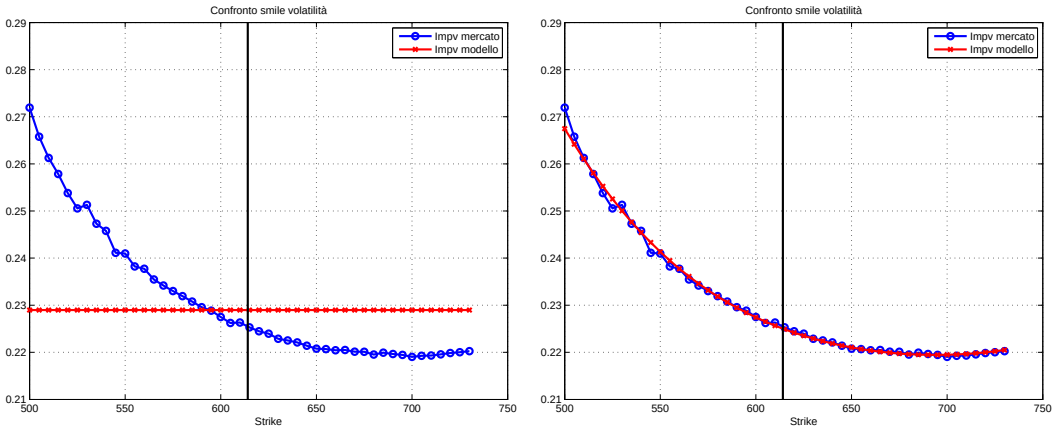


(e) VG



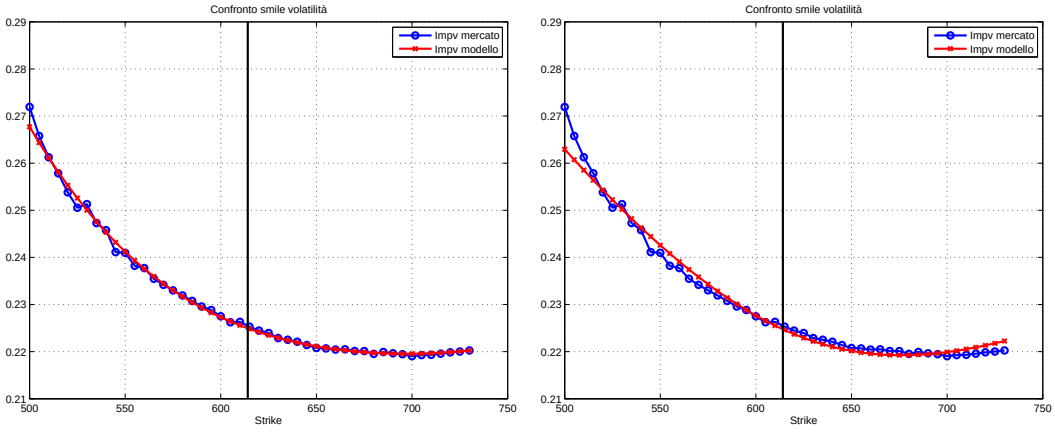
(f) Heston

Figura 3.10: Confronto volatilità implicite per il metodo di Carr&Madan



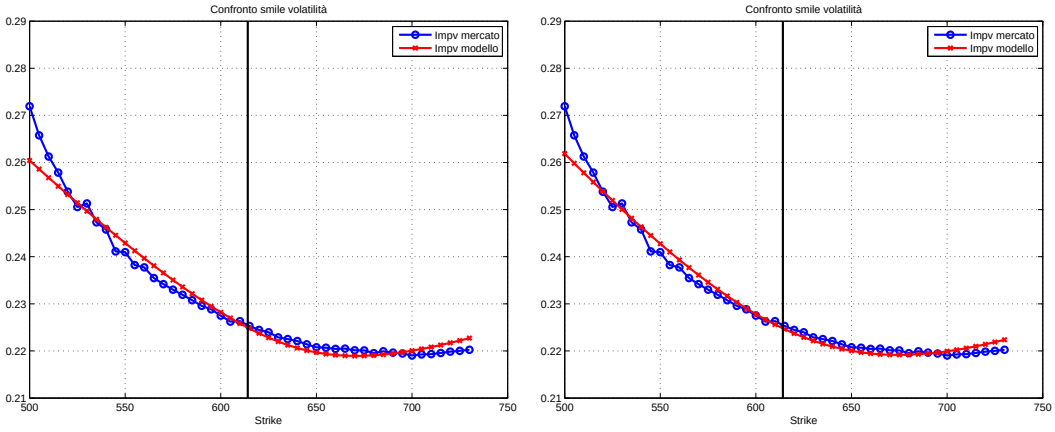
(a) Black&Scholes

(b) Merton



(c) Kou

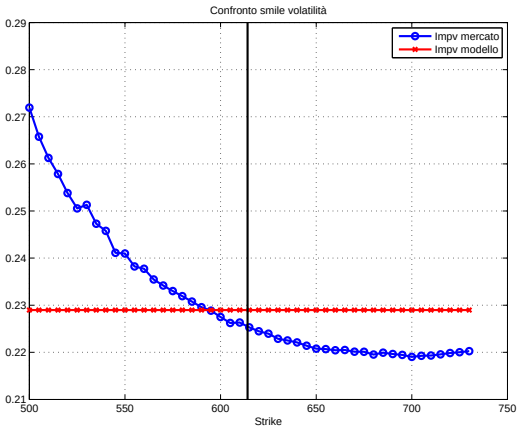
(d) NIG



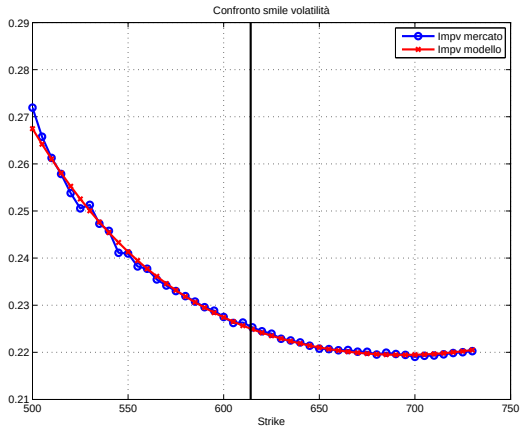
(e) VG

(f) Heston

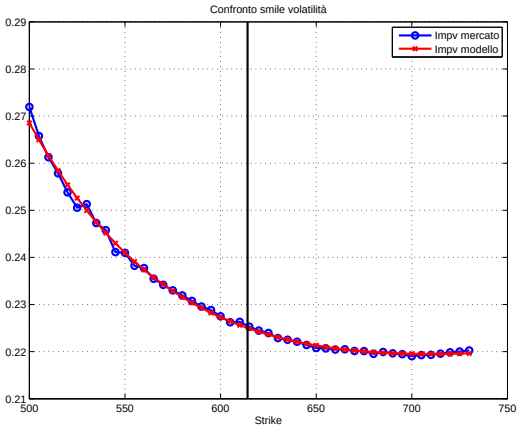
Figura 3.11: Confronto volatilità implicite per il metodo CONV



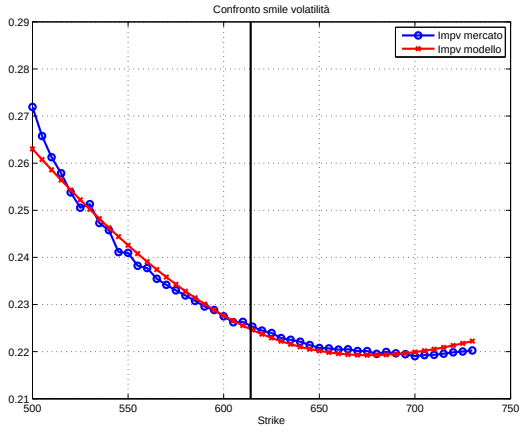
(a) Black&Scholes



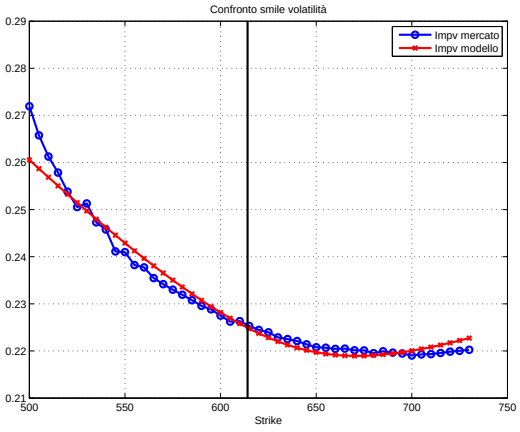
(b) Merton



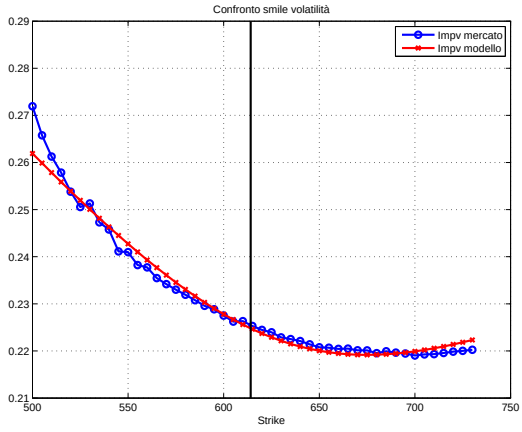
(c) Kou



(d) NIG



(e) VG



(f) Heston

Figura 3.12: Confronto volatilità implicite per il metodo COS

Capitolo 4

Interfaccia Grafica

Come si è detto all'inizio lo scopo finale era quello di creare una "scatola nera" a disposizione dell'utente per calibrare sui dati di mercato di opzioni liquide e prezzare opzioni esotiche.

L'idea di sollevare l'utente da ogni responsabilità implementativa e di fornirgli quindi uno strumento prontamente fruibile si concretizza con la costruzione di un'interfaccia grafica in grado di accettare gli input del problema di pricing e di visualizzare gli output richiesti.

In `Matlab` vengono messe a disposizione librerie apposite per la creazione di queste GUI, in particolare il comando `guide` consente di accedere all'editor della GUI. Ogni funzione della GUI va poi programmata attraverso apposite funzioni callback, eseguite non appena l'utente compie un'azione, come selezionare un valore da un menù a tendina o premere un pulsante. Tramite queste funzioni é stato anche possibile modificare dinamicamente la GUI, ad esempio per escludere scelte tra loro incompatibili, o output irrilevanti (come ad esempio l'intervallo di confidenza se il metodo scelto non è il Monte Carlo).

Tutto lo studio dei capitoli precedenti ha consentito di inviduare metodi di pricing più adatti di altri, tuttavia si è cercato di rendere l'intefaccia più completa possibile, lasciando comunque all'utente la possibilità di selezionare il metodo di calibrazione e pricing desiderato. Sono stati invece pre-settati i parametri di discretizzazione: è comunque possibile modificare la GUI in modo che sia l'utente a sceglierli.

In Figura 4.1 è riportata l'interfaccia grafica creata. Si nota subito la divisione tra la parte di calibrazione e quella di pricing. Nelle sezioni successive viene spiegata ciascuna delle due parti e la GUI viene testata prezzando un derivato di tipo barriera.

4.1 Calibrazione

I comandi a disposizione dell'utente consentono di selezionare quello che desidera fare. Nel box selezione sono presenti due menù a tendina.

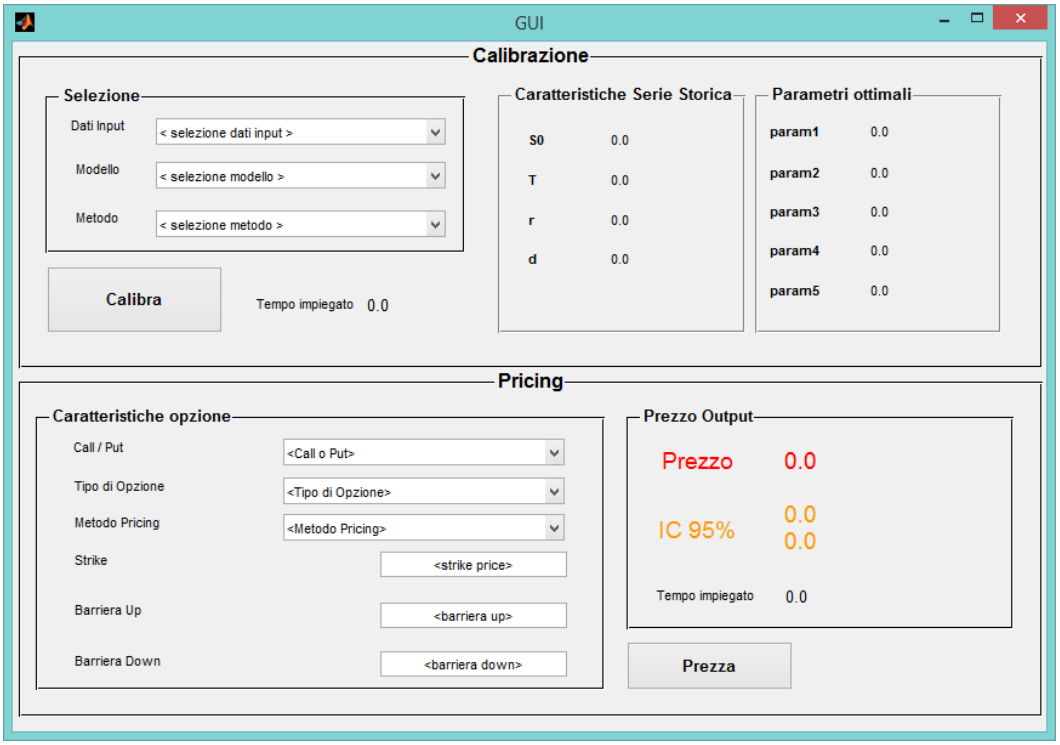


Figura 4.1: GUI appena aperta

Il primo menù a tendina, *Dati Input*, consente di scegliere il dataset da utilizzare per calibrazione e pricing, le scelte possibili sono:

1. AAPL-JAN15
2. AAPL-JAN16
3. MSFT-JAN15
4. MSFT-JAN16

Il secondo menù a tendina, *Modello*, consente di scegliere il modello da utilizzare per la calibrazione, nella scelta si dovrebbe tenere conto delle considerazioni fatte nei capitoli precedenti, le scelte possibili sono:

1. Black&Scholes
2. Merton
3. Kou
4. NIG
5. VG
6. Heston

Il terzo menù a tendina, *Metodo*, consente di scegliere il metodo per la calibrazione: i metodi disponibili sono

1. Albero binomiale
2. Carr&Madan
3. CONV
4. COS

Il pulsante *Calibra* inizia la procedura di calibrazione.

I dati su cui calibrare devono essere in formato Excel, in particolare in un workbook composto da due fogli, con una specifica (ma semplice) formattazione. Nel primo foglio Excel occorre inserire nella colonna `A` gli strike delle opzioni e nella colonna `B` i prezzi di mercato corrispondenti. Nel secondo foglio occorre inserire i parametri S_0, r, T e d disposti in colonna, nell'ordine indicato. In Figura 4.2 viene riportata la formattazione richiesta appena descritta.

Questo da un lato può essere visto come l'unico vero limite all'usabilità della GUI e dall'altro può fornire uno spunto di ulteriore sviluppo all'ampliamento del lavoro portato avanti in questa tesi. Infatti si potrebbero utilizzare delle interfacce apposite con il web per ricavare i dati in tempo reale su richiesta dell'utente da uno specifico data provider.

Le caratteristiche della serie storica sono riportate nell'apposito box non appena la calibrazione viene lanciata. Vengono riportati il prezzo spot, il time to maturity espresso in anni, un tasso risk-free rappresentativo del mercato di cui fa parte il sottostante ed il dividend yield.

I parametri ottimali ottenuti al termine della calibrazione sono riportati nel box apposito, la configurazione è dinamica e varia ogni volta che si sceglie un modello differente.

Infine viene riportato il tempo impiegato dalla procedura di calibrazione.

In Figura 4.3 viene riportata la GUI appena dopo il lancio della procedura di calibrazione.

4.2 Pricing

I parametri ottimali ricavati dalla procedura di calibrazione vengono salvati nella memoria del calcolatore pronti per essere utilizzati per la procedura di pricing.

Nell'apposito box, *Caratteristiche opzione*, l'utente può specificare le caratteristiche dell'opzione che desidera prezzare.

Il primo menù a tendina *Call / Put* consente di selezionare se l'opzione è una Call o una Put.

Il secondo menù a tendina *Tipo di Opzione* consente di specificare la tipologia di opzione tra le seguenti alternative:

1. Europea

	A	B	C	D	E	F	G	H
1	Strike	Price						
2	500,00	118,43						
3	505,00	113,80						
4	510,00	109,40						
5	515,00	105,18						
6	520,00	100,90						
7	525,00	96,78						
8	530,00	93,28						
9	535,00	89,15						
10	540,00	85,45						
11	545,00	81,33						
12	550,00	77,98						
13	555,00	74,28						
14	560,00	71,03						
15	565,00	67,55						
16	570,00	64,33						
17	575,00	61,20						
18	580,00	58,18						
19	585,00	55,23						
20	590,00	52,35						
21	595,00	49,65						
22	600,00	46,93						
23	605,00	44,30						
24	610,00	42,03						

	A	B	C	D	E	F	G
1	S0	614,13					
2	r	0,03%					
3	T	0,64					
4	d	2,10%					
5							
6							

Figura 4.2: Formattazione dei dati in Excel

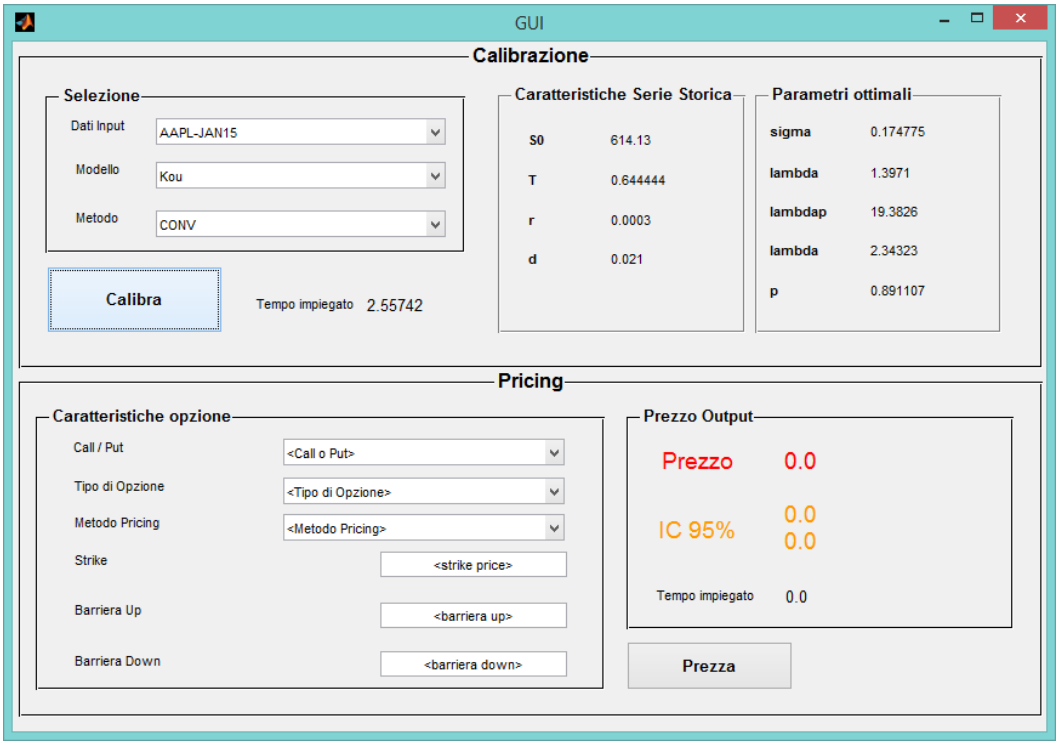


Figura 4.3: GUI dopo la calibrazione

2. Barriera
3. LookBack Fixed Strike
4. LookBack Floating Strike
5. Asiatica Fixed Strike a media geometrica o aritmetica
6. Asiatica Floating Strike a media geometrica o aritmetica

Il terzo menù a tendina *Metodo Pricing* consente di specificare il metodo da utilizzare tra le seguenti alternative:

1. Monte Carlo
2. Albero binomiale
3. Carr&Madan
4. CONV
5. COS

Si noti che i campi *Strike*, *Barriera Up*, *Barriera Down* scompaiono se non sono applicabili all'opzione selezionata.

Il pulsante *Prezza* consente di avviare il pricing richiesto. Se il metodo selezionato non é applicabile all'opzione scelta il prezzo verrà semplicemente impostato a NaN. In particolare, il metodo di Monte Carlo, affidabile e capace di fornire anche un intervallo

di confidenza per il prezzo è applicabile a tutte le opzioni esotiche considerate. Per le Barriera sono in più disponibili CONV e COS, alle Asiatiche Fixed Strike a media geometrica è anche applicabile il COS, mentre le Europee possono essere prezzate con ciascuno dei metodi visti.

Infine i risultati vengono presentati nel box *Prezzo Output*. Il prezzo calcolato viene scritto nella stringa messa in evidenza dal colore rosso e dal carattere di dimensioni maggiori, mentre l'intervallo di confidenza ottenuto col metodo di Monte Carlo appare in una apposita stringa di colore arancione, la quale scompare se vengono selezionati altri metodi di pricing.

In Figura 4.4 viene riportata la GUI appena dopo il lancio della procedura di calibrazione per due diversi metodi di pricing, si noti che il tempo impiegato dal metodo CONV è molto minore di quello impiegato dal Monte Carlo.

Questo esempio ha messo in luce la semplicità e l'usabilità della GUI. Si verifica che vi è una consistenza al variare del metodo scelto per la calibrazione, quindi la scelta si riduce ad un problema di costo computazionale: come si è visto nel Capitolo 3.2 il metodo COS è quello suggerito.

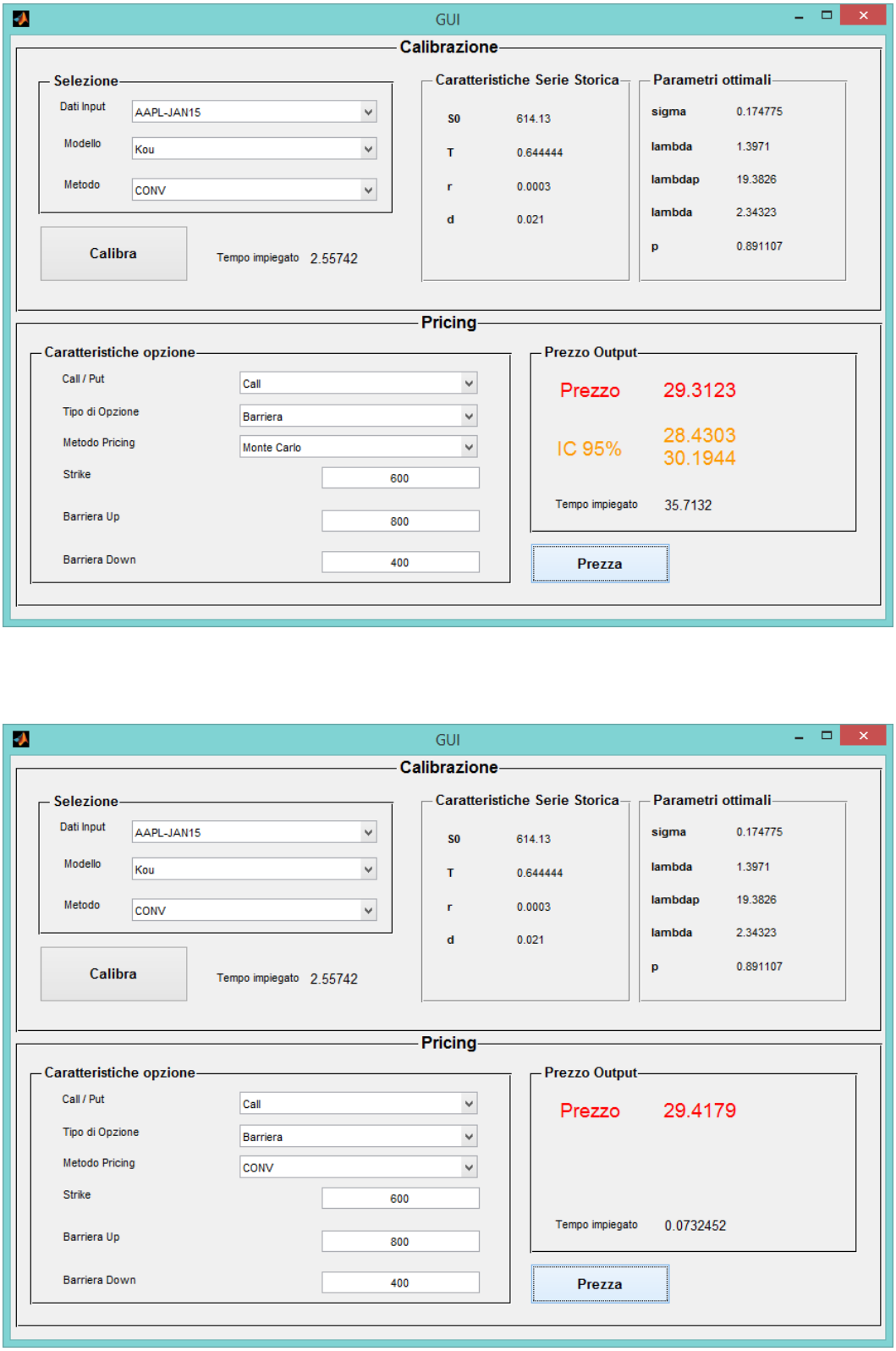


Figura 4.4: GUI dopo il pricing, procedura ultimata

Capitolo 5

Conclusioni

In questo lavoro di tesi ci si è posto l'obiettivo di implementare un modello di pricing in un software per cui è stata sviluppata un'opportuna interfaccia grafica. Per raggiungere questo scopo è necessario affrontare due problemi che sorgono spontanemente: quello della calibrazione e quello del pricing. Entrambi questi aspetti sono stati integrati nel software, il quale consente dapprima di calibrare ed in seguito di utilizzare i parametri ottimali per procedere al pricing.

Il focus della tesi è prevalentemente sulla risoluzione efficiente del problema della calibrazione, in quanto esso richiede numerose valutazioni di funzione. Infatti occorre valutare un funzione non lineare per ogni punto del dataset a ciascuna iterazione del solutore di minimizzazione.

Molteplici sono gli aspetti da considerare a questo riguardo. In primo luogo si è determinato, tra gli algoritmi di ottimizzazione presenti in `Matlab`, quello che è in grado di fornire i migliori risultati senza però richiedere un costo computazionale inaccettabile. L'analisi empirica effettuata ha spinto a scegliere il metodo `lsqcurvefit` che ha fornito risultati soprendentemente buoni, con tempi molto ridotti si avvicina alla precisione del risolutore globale di `gs` il quale però ha tempi di esecuzione insostenibili.

In seconda istanza si sono confrontati numericamente i diversi metodi di pricing di opzioni europee descritti nel Capitolo 2, l'albero (solo per Black&Scholes), e gli approcci basati sulla trasformata di Fourier (Carr&Madan, CONV e COS) in termini di costo e precisione. Dall'analisi è emerso che il metodo COS è il più performante sotto questi punti di vista. I risultati empirici contenuti negli articoli in cui questi metodi sono stati introdotti hanno costituito una significativa base di partenza per il lavoro svolto.

Infine si deve tenere conto che sono disponibili vari modelli per il sottostante e che l'utente ha la possibilità di selezionare quello che ritiene più adatto. Si noti comunque che modelli più semplici, quale Black&Scholes forniscono in generale performance peggiori. Per quantificare questo aspetto sono stati prodotti grafici e tabelle che riportano gli errori tra i prezzi di mercato e quelli previsti dal modello, il valore ottimale della funzione RMSE e gli smile di volatilità implicita ottenuti con

il modello. Nel complesso si osserva che, a parte Black&Scholes, i diversi modelli ottengono performance confrontabili.

Per quanto riguarda il pricing, il metodo di Monte Carlo, affidabile e capace di fornire anche un intervallo di confidenza per il prezzo è applicabile a tutte le opzioni esotiche considerate. Per le Barriera sono in più disponibili CONV e COS, mentre le Europee possono essere prezzate con ciascuno dei metodi visti.

In conclusione si osserva che la presente tesi offre alcune opportunità di ampliamento delle funzionalità del software. In particolare si potrebbe implementare una routine che consenta di reperire direttamente i dati delle option chain da un provider online. Potrebbero inoltre essere introdotti altri metodi di pricing o di calibrazione, oppure consentire l'applicazione ad altre categorie di opzioni in aggiunta a quelle già considerate.

Appendice A

Serie Storiche

In questa appendice vengono riportati i dataset utilizzati per le analisi numeriche.

A.1 AAPL

Il primo set di dati riguarda le opzioni Call Europee con sottostante il colosso informatico Apple. Sono state considerate due diverse scadenze, il 17/01/2015 e 15/01/2016. I prezzi sono presi il 25/05/2014, il valore spot del sottostante è 614.13 \$, il dividend yield è del 2.1%, come tasso risk-free si è preso 0.03% ovvero il rendimento annuale di un treasury bill americano a 3 mesi.

In Figura A.1 e Figura A.2 sono riportati i grafici dei prezzi quotati e delle volatilità implicite. In quest'ultimo grafico è possibile riscontrare la presenza di un volatility skew, condizione tipica del caso equity.

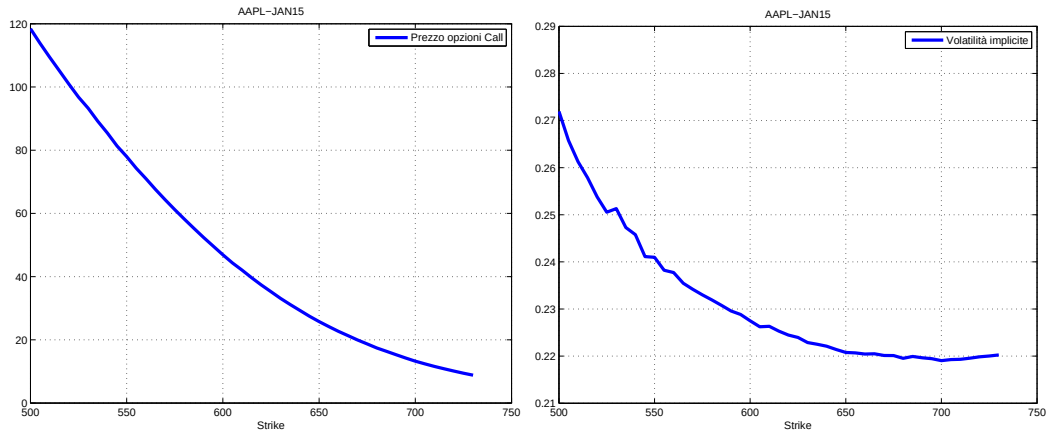


Figura A.1: Prezzi e smile di volatilità impliciti AAPL-JAN15.

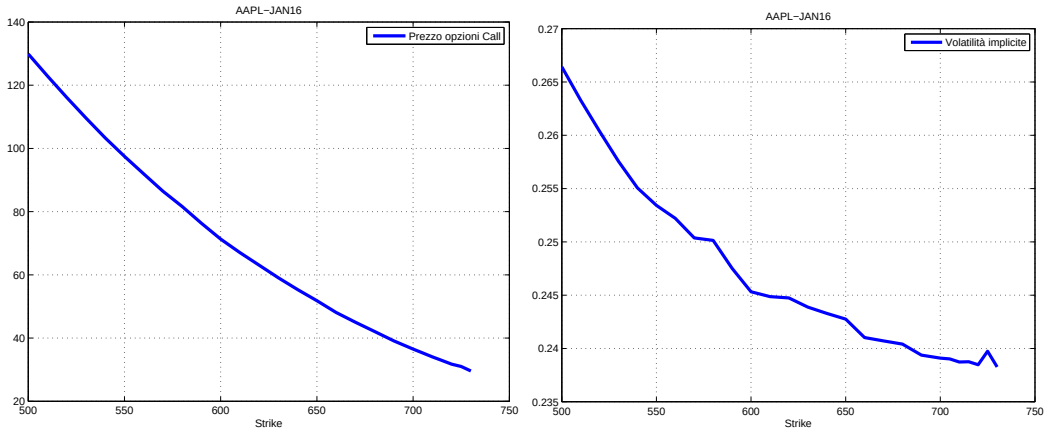


Figura A.2: Prezzi e smile di volatilità impliciti AAPL-JAN16.

A.2 MSFT

Il secondo set di dati riguarda le opzioni Call Europee con sottostante Microsoft. Sono state considerate due diverse scadenze, il 17/01/2015 e 15/01/2016. I prezzi sono presi il 25/05/2014, il valore spot del sottostante è 40.12$, il dividend yield è del 2.8%, come tasso risk-free si è preso 0.03% ovvero il rendimento annuale di un treasury bill americano a 3 mesi.

In Figura A.3 e Figura A.4 sono riportati i grafici dei prezzi quotati e delle volatilità implicite. In quest'ultimo grafico è possibile riscontrare la presenza di un volatility skew, condizione tipica del caso equity.

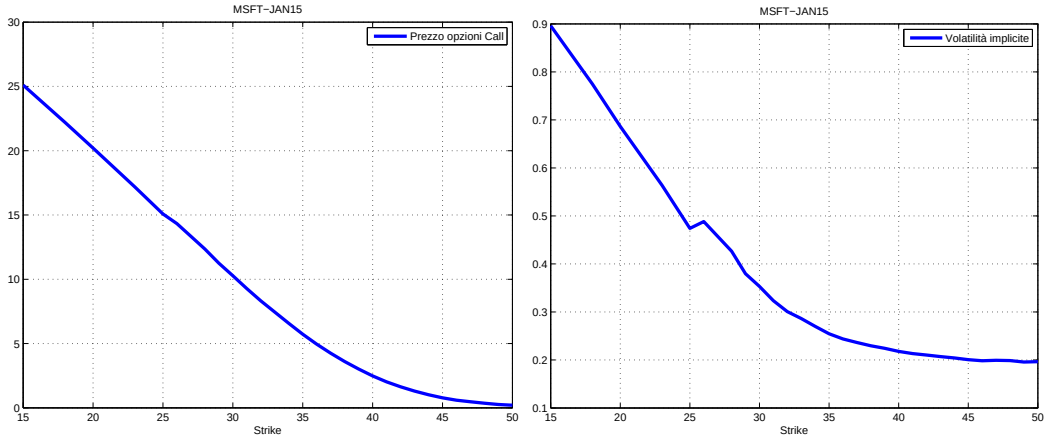


Figura A.3: Prezzi e smile di volatilità impliciti MSFT-JAN15.

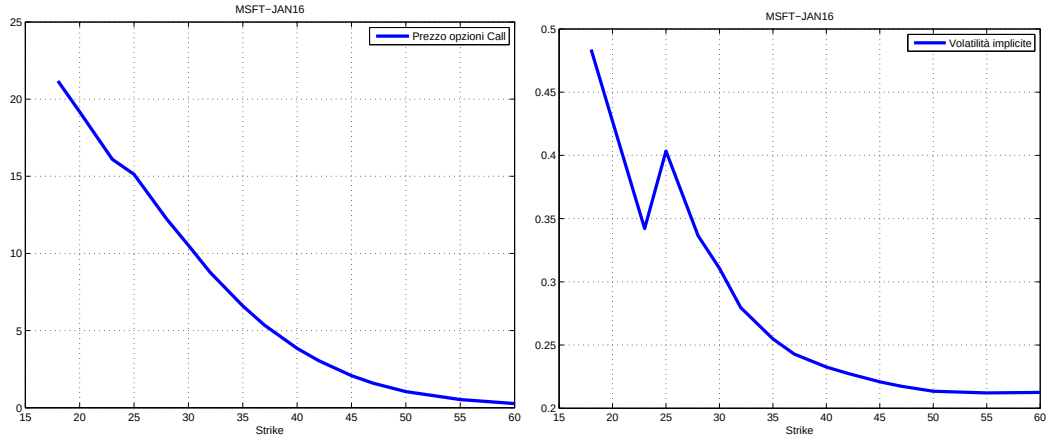


Figura A.4: Prezzi e smile di volatilità impliciti MSFT-JAN16.

Appendice B

Codici

In questa appendice vengono riportati i principali codici utilizzati per la realizzazione della tesi.

B.1 Metodi di Pricing per opzioni Europee

B.1.1 Vanilla_Pricing

Questo script consente di prezzare un'opzione Europea, occorre passargli tutti i parametri dell'opzione, quelli del modello, il modello scelto ed il metodo di pricing. Lo script si preoccupa di chiamare la funzione adeguata.

```
% Function that returns the price of an european option using the
    selected
% method and the selected model for underlying
%   [PrezzoCall, PrezzoPut] = Vanilla_Pricing(S0, r, d, T,
    Strike_mercato, param_modello, modello, metodo)
%
%   Inputs:
%   S0              - spot price
%   r               - risk free rate
%   div             - dividend yield
%   T               - maturity
%   Strike          - stike prices (can be a vector)
%   param_modello   - parameters of underlying model
%   modello         - string specifying the model that has to be used
%   metodo          - method that is to be used
%
%   Outputs:
%   PrezzoCall      - price of european Call option
%   PrezzoPut       - price of european Put option

function [PrezzoCall, PrezzoPut] = Vanilla_Pricing(S0, r, d, T,
    Strike_mercato, param_modello, modello, metodo)

if strcmp(metodo, 'CARRM');
```

```
    [PrezzoCall, PrezzoPut] = Vanilla_CARRM(S0, r, d, T,
    Strike_mercato, param_modello, modello);
elseif strcmp(metodo, 'Albero');
    [PrezzoCall, PrezzoPut] = Vanilla_Albero_BS(S0, r, d, T,
    Strike_mercato, param_modello);
elseif strcmp(metodo, 'CONV');
    [PrezzoCall, PrezzoPut] = Vanilla_CONV(S0, r, d, T,
    Strike_mercato, param_modello, modello);
elseif strcmp(metodo, 'COS')
    [PrezzoCall, PrezzoPut] = Vanilla_COS(S0, r, d, T, Strike_mercato,
    param_modello, modello);
end

end
```

B.1.2 Vanilla_Albero_BS

Metodo dell'albero binomiale per il modello di Black&Scholes.

```
% Function that returns the price of an european option using binomial
    tree
% algorithm
%   [PrezzoCall, PrezzoPut] = Vanilla_Albero_BS(S0, r, div, T, Strike,
    param_modello)
%
%   Inputs:
%   S0              - spot price
%   r               - risk free rate
%   div             - dividend yield
%   T               - maturity
%   Strike          - stike prices (can be a vector)
%   param_modello   - parameters of underlying model
%
%   Outputs:
%   PrezzoCall      - price of european Call option
%   PrezzoPut       - price of european Put option

function [PrezzoCall, PrezzoPut] = Vanilla_Albero_BS(S0, r, div, T,
    Strike, param_modello)

N = 50;          % Number of Time Steps
dt = T/N;        % Time Step

% >> BS Parameters
sigma_BS = param_modello(1);

% >> CRR Parameters
u = exp(sigma_BS*sqrt(dt));
d = 1/u;
q = (exp((r-div)*dt) - d)/(u-d);

% >> Explicit multistep formula
somma = zeros(size(Strike));
for k = 0:N
    somma = somma ...
            + nchoosek(N, k) ...
```

```
            * q^k*(1-q)^(N-k)  ...
            * max(S0*u^k*d^(N-k) - Strike, 0);
end

% Discounting
PrezzoCall = somma / (1+r)^T;

% Put-Call parity
PrezzoPut =  PrezzoCall - S0*exp(-div*T) + Strike*exp(-r*T);

end
```

B.1.3 Vanilla_CeM

Metodo di Carr&Madan per il pricing di opzioni Europee.

```
% Function that returns the price of an european option using
    Carr&Madan's
% FFT algorithm
%   [PrezzoCall, PrezzoPut] = Vanilla_CARRM(S0, r, d, T, Strike,
    param_modello, modello)
%
%   Inputs:
%   S0              - spot price
%   r               - risk free rate
%   div             - dividend yield
%   T               - maturity
%   Strike          - stike prices (can be a vector)
%   param_modello   - parameters of underlying model
%   modello         - string specifying the model that has to be used
%
%   Outputs:
%   PrezzoCall      - price of european Call option
%   PrezzoPut       - price of european Put option

function [PrezzoCall, PrezzoPut] = Vanilla_CARRM(S0, r, d, T, Strike,
    param_modello, modello)

% Characteristic function of the model
CharFunc = @(u)  exp(-T*1i*u*(r-d)).*CharFuncLib(u, r, d, T,
    param_modello, modello);

% >> Grids
% v grid (Fourier)
Npow = 14;
N = 2^Npow;                     % number of grid points
A = 600;                        % truncation of integration domain
eta = A/N;                      % grid step
j = 0:N-1;                      % v grid index
v = eta*j;                      % Grid
v(1) = 1e-22;                   % doesn't work if grid contains 0

% k grid (logstrike)
lambda = 2*pi/A;                % grid step
l = (0:N-1);                    % k grid index
k = -lambda*N/2 + lambda*l;     % Griglia logstrike
```

```
% >> Fourier transform of modified price z
% g is known from Carr&Madan's formula
g = exp(-d*T)*exp(-1i*d*v*T).*exp(1i*r*v*T).*(CharFunc(v-1i)-1)./
    (1i*v.*(1i*v+1));

% >> Inversion of the transform
w = ones(1, N); w(1) = 0.5; w(end) = 0.5;   % integration weights
    (trap.)
x = w.*eta.*g.*exp(1i*pi*j);          % * exp(-i*j*2*pi*l/N) comes from
    fft
z = fft(x)/pi;                                % corrected price
z = real(z);                                  % elimination of imaginary
    part
C = S0*(z + max(exp(-d*T)-exp(k-r*T), 0));  % going back to call price
K = S0*exp(k);                                % vector of strikes

% Interpolation
PrezzoCall = interp1(K, C, Strike, 'spline');

% Put-Call parity
PrezzoPut =  PrezzoCall - S0*exp(-d*T) + Strike*exp(-r*T);

end
```

B.1.4 Vanilla_CONV

Metodo CONV per il pricing di opzioni Europee.

```
% Function that returns the price of an european option using CONV FFT
% algorithm
%   [PrezzoCall, PrezzoPut] = Vanilla_CONV(S0, r, d, T, Strike,
    param_modello, modello)
%
%   Inputs:
%   S0              - spot price
%   r               - risk free rate
%   div             - dividend yield
%   T               - maturity
%   Strike          - stike prices (can be a vector)
%   param_modello   - parameters of underlying model
%   modello         - string specifying the model that has to be used
%
%   Outputs:
%   PrezzoCall      - price of european Call option
%   PrezzoPut       - price of european Put option

function [PrezzoCall, PrezzoPut] = Vanilla_CONV(S0, r, d, T, Strike,
    param_modello, modello)

alpha = 0;     % Dampening factor

% Characteristic function of the model
CharFunc = @(u) CharFuncLib(u, r, d, T, param_modello, modello);

% >> Grids
```

```
Nstrike = length(Strike);    % Number of strikes
Npow = 8;
Ngrid = 2^Npow;              % number of grid points
L = 10;                      % truncation parameter

% grid steps (satisfying Nyquist relation)
dy = L/Ngrid;                % log-price (t = T)
dx = dy;                     % log price (t = t0)
du = 2*pi/L;                 % fourier frequency grid

j = repmat((0:Ngrid-1)', 1, Nstrike);   % grid index
sgn = (-1).^j;                          % (-1)^j

x = - dx*Ngrid/2 + dx*j;                    % adjusted x grid
y = -dy*Ngrid/2 + dy*j ...
    + repmat(log(Strike / S0)', Ngrid, 1);  % adjusted y grid
u = - du*Ngrid/2 + du*j;                    % adjusted u grid

% >> Pricing
V = max( S0*exp(y) - repmat(Strike', Ngrid, 1), 0); % Payoff
v = V .* exp(alpha .* y);                           % Dampening

% integration weights (trap.)
w = ones(Ngrid, Nstrike); w(1,:) = 0.5; w(Ngrid, :) = 0.5;

% internal transform
FT_Vec = ifft( (sgn) .* w .* v);

% vector to be transformed
FT_Vec_tbt = exp(1i .* j * (diag(y(1,:) - x(1,:))) .* du) ...
             .* CharFunc(-(u-(1i*alpha))) ...
             .* FT_Vec;

% final formula
C = exp(-r*T - (alpha.*x) + (1i .* u*(diag(y(1,:) - x(1,:))))) ...
    .* (sgn) .* fft(FT_Vec_tbt);
C = abs(C);

% Interpolation
PrezzoCall = C(Ngrid/2 + 1, :)';

% Put-Call parity
PrezzoPut =  PrezzoCall - S0*exp(-d*T) + Strike*exp(-r*T);

end
```

B.1.5 Vanilla_COS

Metodo COS per il pricing di opzioni Europee.

```
% Function that returns the price of an european option using COS
    algorithm
%   [PrezzoCall, PrezzoPut] = Vanilla_COS(S0, r, d, T, Strike,
    param_modello, modello)
%
%   Inputs:
```

```
%   S0              - spot price
%   r               - risk free rate
%   div             - dividend yield
%   T               - maturity
%   Strike          - stike prices (can be a vector)
%   param_modello   - parameters of underlying model
%   modello         - string specifying the model that has to be used
%
%   Outputs:
%   PrezzoCall      - price of european Call option
%   PrezzoPut       - price of european Put option

function [PrezzoCall, PrezzoPut] = Vanilla_COS(S0, r, d, T, Strike,
    param_modello, modello)

CharFunc = @(u) CharFuncLib(u, r, d, T, param_modello, modello);

Npow = 6;
N = 2^Npow;                 % Numero di punti della griglia
Nstrike = length(Strike);   % Numero di Strikes

% >> Griglia spazio degli strikes
x = repmat(double(log(S0 ./ Strike))', N, 1);   % Centri griglia

% >> truncation of integration domain
a = -2*ones(size(x)) + x;
b = 2*ones(size(x)) + x;

% >> Griglia spazio fourier
k = repmat((0:N-1)', 1, Nstrike);               % Indici della griglia

% >> Coefficienti per la Put
vk_p = @(x) calcvkp(x, b, a, Strike);
Vk = vk_p(k);

% >> Integration
fk_i = CharFunc(k.*pi./(b-a)) ...
       .* exp(1i .* k .* pi .* (x - a) ./ (b - a));
fk_i(1,:) = 0.5*fk_i(1,:);      % Il primo termine va *0.5

% Put price
PrezzoPut = exp(-r .* T) .* real(sum(fk_i .* Vk)');

% Put-Call Parity
PrezzoCall = PrezzoPut + S0*exp(-d*T) - Strike*exp(-r*T);

end

%% function that returns the V_k coefficients for a european put option
function y = calcvkp(k, b, a, strike)
[chi, psi] = coeff(k,a,0,a,b);
y = 2./(b - a).*(psi - chi)*diag(strike);
end

%% function that returns the values of chi and psi
function [chi, psi] = coeff(k, c, d, a, b)

% evaluation of Psi
```

```
psi = (d-c);

temp = double((b - a) ./ (k .* pi) ...
        .* ( sin(k .* pi .* (d - a) ./ (b - a)) ...
        - sin(k .* pi .* (c - a) ./ (b - a)) ));

psi(2:end,:) = temp(2:end,:);

% evaluation of Chi
chi = (exp(d)-exp(c));
chi1 = 1 ./ ( 1 + ( k .* pi ./ (b - a) ) .^ 2 );
chi2 = exp(d) .* cos(k .* pi .* (d - a) ./ (b - a)) ...
        - exp(c) .* cos(k .* pi .* (c - a) ./ (b - a));
chi3 = k .* pi ./ (b - a) .* ...
        ( exp(d) .* sin(k .* pi .* (d - a) ./ (b - a)) ...
        - exp(c) .* sin(k .* pi .* (c - a) ./ (b - a)) );

temp = chi1 .* (chi2 + chi3);

chi(2:end,:) = temp(2:end,:);

end
```

B.2 Metodi di Pricing per opzioni Barriera

B.2.1 Barrier_Pricing

Questo script consente di prezzare un'opzione Barriera, occorre passargli tutti i parametri dell'opzione, quelli del modello, il modello scelto ed il metodo di pricing. Lo script si preoccupa di chiamare la funzione adeguata.

```
% Function that returns the price of a knock-out barrier option using
    the
% selected method and the selected model for underlying
%   Prezzo = Barrier_Pricing(S0, r, d, T, U, D, Strike, flagCP,
    param_modello, modello, metodo)
%
%   Inputs:
%   S0              - spot price
%   r               - risk free rate
%   d               - dividend yield
%   T               - maturity
%   U               - Up Barrier
%   D               - Down Barrier
%   Strike          - stike price
%   flagCP          - 1: Call, -1:Put
%   param_modello   - parameters of underlying model
%   modello         - string specifying the model that has to be used
%   metodo          - method that is to be used
%
%   Outputs:
%   Prezzo          - price of the option
```

```
function Prezzo = Barrier_Pricing(S0, r, d, T, U, D, Strike, flagCP,
    param_modello, modello, metodo)

if strcmp(metodo, 'CONV');
    Prezzo = Barrier_CONV(S0, r, d, T, U, D, Strike, flagCP,
    param_modello, modello);
elseif strcmp(metodo, 'COS')
    Prezzo = Barrier_COS(S0, r, d, T, U, D, Strike, flagCP,
    param_modello, modello);
end

end
```

B.2.2 Barrier_CONV

Metodo CONV per il pricing di opzioni Barriera.

```
% Function that returns the price of a knock-out barrier option using
    CONV
% FFT algorithm
%   Prezzo = Barrier_CONV(S0, r, d, T, U, D, Strike, flagCP,
    param_modello, modello)
%
%   Inputs:
%   S0              - spot price
%   r               - risk free rate
%   d               - dividend yield
%   T               - maturity
%   U               - Up Barrier
%   D               - Down Barrier
%   Strike          - stike price
%   flagCP          - 1: Call, -1:Put
%   param_modello   - parameters of underlying model
%   modello         - string specifying the model that has to be used
%
%   Outputs:
%   Prezzo          - price of the option

function Prezzo = Barrier_CONV(S0, r, d, T, U, D, Strike, flagCP,
    param_modello, modello)

M = 252;    % monitoraggio
dt = T/M;   % delta temporale

alpha = 0.0;    % Dampening factor

% Characteristic function of the model
CharFunc = @(u) CharFuncLib(u, r, d, dt, param_modello, modello);

% >> Grids
Nstrike = length(Strike);   % Number of strikes
Npow = 10;
Ngrid = 2^Npow;             % number of grid points
L = 10;                     % truncation parameter

% grid steps (satisfying Nyquist relation)
```

```
dy = L/Ngrid;                    % log-price (t = T)
dx = dy;                         % log price (t = t0)
du = 2*pi/L;                     % fourier frequency grid

j = repmat((0:Ngrid-1)', 1, Nstrike);   % grid index
sgn = (-1).^j;                          % (-1)^j

x = - dx*Ngrid/2 + dx*j;                    % adjusted x grid
y = -dy*Ngrid/2 + dy*j;                     % adjusted x grid
u = - du*Ngrid/2 + du*j;                    % adjusted u grid

% >> Pricing
V = max( flagCP*(S0*exp(y) - repmat(Strike', Ngrid, 1)), 0); % Payoff
v = V .* exp(alpha .* y);                                    % Dampening

% integration weights (trap.)
w = ones(Ngrid, Nstrike); w(1,:) = 0.5; w(Ngrid, :) = 0.5;

% ------ Main Loop
for i = 1:M

%%%%% QUESTO E' IL CODICE DELL'EUROPEA, al posto di T c'e dt
% >> convoluzione
FT_Vec = ifft( (sgn) .* w .* v);

% >> trasformata inversa
FT_Vec_tbt = exp(1i .* j * (diag(y(1,:) - x(1,:))) .* du) ...
             .* CharFunc(-(u-(1i*alpha))) ...
             .* FT_Vec;

C = exp(-r*dt - (alpha.*x) + (1i .* u*(diag(y(1,:) - x(1,:))))) ...
    .* (sgn) .* fft(FT_Vec_tbt);
C = abs(C);
%%%%%

%%%%% VIENE AGGIUNTA LA PROIEZIONE
% >> Proiezione che annulla fuori dalla barriera
C = C .* (S0*exp(y) < U) .* (S0*exp(y) > D);
v = C;
end

% >> Interpolation
Prezzo = C(Ngrid/2 + 1, :)';

end
```

B.2.3 Barrier_COS

Metodo COS per il pricing di opzioni Barriera.

```
% Function that returns the price of a knock-out barrier option using
    COS
% algorithm
%   Prezzo = Barrier_COS(S0, r, div, T, U, D, K, flagCP,
    param_modello, modello)
%
```

```
%   Inputs:
%   S0              - spot price
%   r               - risk free rate
%   div             - dividend yield
%   T               - maturity
%   U               - Up Barrier
%   D               - Down Barrier
%   Strike          - stike price
%   flagCP          - 1: Call, -1:Put
%   param_modello   - parameters of underlying model
%   modello         - string specifying the model that has to be used
%
%   Outputs:
%   Prezzo          - price of the option

function Prezzo = Barrier_COS(S0, r, div, T, U, D, K, flagCP,
    param_modello, modello)

Nex = 252;      % number of examination points
dt = T / Nex;   % time interval

CharFunc = @(u) CharFuncLib(u, r, div, dt, param_modello, modello);

Npow = 8;

Ngrid = 2 ^ Npow;           % Grid points
Nstrike = length(K);        % number of strikes

x = double(log(S0 ./ K));   % center
hu = double(log(U ./ K));   % log up barrier
hd = double(log(D ./ K));   % log down barrier

% >> Tronco il dominio dell'integrale a occhio
L = 2;
a = -L + x;
b = L + x;

% >> Griglia spazio fourier
k = repmat((0:Ngrid-1)', 1, Nstrike);  % Grid index


% Controllo coerenza sulla barriera
if hu < hd
   Prezzo = 0;
   return
end

% ------ Coefficienti V(x1, x2), a scadenza. Sono i G dell'articolo.
% Call
    if flagCP == 1
        if (hd >= 0 && hu >= 0)
            V = calcG(k, hd, hu, a, b, flagCP, K);
        elseif (hd < 0 && hu >= 0)
            V = calcG(k, 0, hu, a, b, flagCP, K);
        else
            V = 0;
        end
    % Put
```

```
    else
        if (hd < 0 && hu >= 0)
            V = calcG(k, hd, 0, a, b, flagCP, K);
        elseif (hd < 0 && hu < 0)
            V = calcG(k, hd, hu, a, b, flagCP, K);
        else
            V = 0;
        end
    end

% ------ Calcolo matrici Mc ed Ms
x2 = hu;
x1 = hd;
% >> I due esponenziali che compaiono nella matrice M
exp2 = exp( 1i .* (1:Ngrid)' .* (x2 - a) ./ (b - a) .* pi );    % init
exp1 = exp( 1i .* (1:Ngrid)' .* (x1 - a) ./ (b - a) .* pi );    % init

% >> Calcolo tutti NxN gli m_i che uso per costruire le matrici M_c ed
    M_s
m = zeros(3*Ngrid-1, 1);                                        % init
    base

m(Ngrid, 1) = 1i * pi * (x2 - x1) / (b - a);
m( (Ngrid+1):(2*Ngrid), 1) = 1 ./ (1:Ngrid)' .* ( exp2 - exp1 );
m(1:Ngrid-1, 1) = - conj(flipud(m( (Ngrid+1):(2*Ngrid-1), 1)));
m(2*Ngrid+1:3*Ngrid-1, 1) = ( exp2(1:Ngrid-1, 1) .* exp2(Ngrid,1) -
    exp1(1:Ngrid-1, 1) ...
    .* exp1(Ngrid, 1) ) ./ ( (Ngrid+1:2*Ngrid-1)' );

% >> Scrivo i tre vettori che occorrono per il calcolo efficiente della
% fomula, usando la trasformata di Fourier. Non occorrono le matrici.
% m_s = [m_0, m_-1, ..., m_1-N, 0, m_N-1, ..., m_1]
% u_s = [u_0, u_1, ..., u_N-1, 0, ... 0]
% m_c = [m_2N-1, m_2N-2, ..., m_1, m_0]
m_s = [m(Ngrid:-1:1, 1); 0; m(2*Ngrid-1:-1:Ngrid+1, 1)];
m_c = m(3*Ngrid-1:-1:Ngrid, 1);

% Funzione che definisce il valore di C(x1, x2, t_m)
cv = @(y) cvalue(m_s, m_c, a, b, Ngrid, y, modello, param_modello, dt,
    r, div);

% ------ Induzione Backward
for m = Nex-1:-1:1              % backward induction
    C = cv(V);
    V = C;
end

% ------ Calcolo del prezzo finale
fk_i = CharFunc(k.*pi./(b-a)) ...
       .* exp(1i .* k .* pi .* (x - a) ./ (b - a));
fk_i(1,:) = 0.5*fk_i(1,:);      % Il primo termine va *0.5

Prezzo = exp(-r * dt) * real(sum(fk_i.*V));  % Option value at t_0

end

%% function that returns the V_k coefficients for a european put
   option
```

```
function y = calcG(k, c, d, a, b, FlagCP, Strike)
[chi, psi] = coeff_b(k, c, d, a, b);
y = 2 * FlagCP ./(b - a).*(chi - psi)*Strike;
end

%% function that returns the values of chi and psi
function [chi, psi] = coeff_b(k, c, d, a, b)

% evaluation of Psi
psi = ones(size(k))*(d-c);                % se k=0 vale d-c

temp = double((b - a) ./ (k .* pi) ...
        .* ( sin(k .* pi .* (d - a) ./ (b - a)) ...
        - sin(k .* pi .* (c - a) ./ (b - a)) ));

psi(2:end,:) = temp(2:end,:);

% evaluation of Chi
chi = ones(size(k))*(exp(d)-exp(c));      % se k = 0 viene e^d-e^c
chi1 = 1 ./ ( 1 + ( k .* pi ./ (b - a) ) .^ 2 );
chi2 = exp(d) .* cos(k .* pi .* (d - a) ./ (b - a)) ...
        - exp(c) .* cos(k .* pi .* (c - a) ./ (b - a));
chi3 = k .* pi ./ (b - a) .* ...
        ( exp(d) .* sin(k .* pi .* (d - a) ./ (b - a)) ...
        - exp(c) .* sin(k .* pi .* (c - a) ./ (b - a)) );

temp = chi1 .* (chi2 + chi3);

chi(2:end,:) = temp(2:end,:);

end

%%
function C = cvalue(m_s, m_c, a, b, N, V, ...
    modello, param_modello, T, r, q)

% C = e^-rt / pi * Im((Mc + Ms)*u) <- formula finale

Grid_j = (0:N-1)';              % fix grid

% >> Calcolo i valori di u
temporanea = Grid_j.*pi./(b-a);
u = CharFuncLib(temporanea, r, q, T, param_modello, modello) .* V;
u(1) = 0.5*u(1);

% >> Calcolo i prodotti matrice vettore Mc*u e Ms*u
% uso che x (conv) y = D^-1 ( D(x) * D(y) )
u_s = [u; zeros(N, 1)];
Mcs = ifft( fft(m_s) .* fft(u_s));
Mcs = Mcs(1:N);      % Il prodotto sono i primi N elementi

segno = -ones(2*N, 1);
segno(2 .* (1:N)' - 1) = 1;
Mcu = ifft( segno .* fft(m_c) .* fft(u_s) );
Mcu = flipud(Mcu(1:N)); % Il prodotto sono i primi N elementi in
    ordine inverso

C = exp(-r * T) / pi .* imag( Mcs + Mcu);
```

```
end
```

B.3 Metodi di Pricing per opzioni Asiatiche

B.3.1 AsianGeom_COS

Metodo COS per il pricing di opzioni Asiatiche fixed-strike a media geometrica.

```
% Function that returns the price of an fixed strike geometric mean
    asian
% option using COS algorithm
%   Prezzo = AsianGeom_COS(S0, r, d, T, Strike, flagCP, param_modello,
    modello)
%
%   Inputs:
%   S0              - spot price
%   r               - risk free rate
%   d               - dividend yield
%   T               - maturity
%   Strike          - stike prices (can be a vector)
%   flagCP          - 1 for Call, -1 for Put
%   param_modello   - parameters of underlying model
%   modello         - string specifying the model that has to be used
%
%   Outputs:
%   Prezzo          - price of asian option

function Prezzo = AsianGeom_COS(S0, r, d, T, Strike, flagCP,
    param_modello, modello)

M = 252;     % Date di monitoraggio
dt = T/M;    % Passo temporale

CharFunc = @(u) CharFuncLib(u, r, d, dt, param_modello, modello);

Npow = 6;
N = 2^Npow;                 % Numero di termini
Nstrike = length(Strike);   % Numero di Strikes

%>> Griglia spazio degli strikes
x = repmat(double(log(S0 ./ Strike))', N, 1);   % Centri griglia

%>> Troncamento del dominio di integrazione
a = -2*ones(size(x)) + x;
b = 2*ones(size(x)) + x;

%>> Griglia spazio fourier
k = repmat((0:N-1)', 1, Nstrike);               % Indici della griglia

%>> Coefficienti Vk
if flagCP == 1
    Vk = calcG(k, 0, b, a, b, flagCP, Strike);
elseif flagCP == -1
    Vk = calcG(k, a, 0, a, b, flagCP, Strike);
```

```
end

% >> Main loop to recover charachteristic function
chF = ones(size(a));
for j = 1:M
    chF = chF.*CharFunc(k.*pi./(b-a).* (M+1 - j)./(M+1));
end

% >> COS formula for pricing
fk_i = chF .* exp(1i .* k .* pi .* (x - a) ./ (b - a));
fk_i(1,:) = 0.5*fk_i(1,:);      % Il primo termine va *0.5
Prezzo = exp(-r .* T) .* real(sum(fk_i .* Vk)');

end

 %% function that returns the V_k coefficients
function y = calcG(k, c, d, a, b, FlagCP, Strike)
[chi, psi] = coeff(k, c, d, a, b);
y = 2 * FlagCP ./(b - a).*(chi - psi)*Strike;
end

%% function that returns the values of chi(c,d) and psi(c,d)
function [chi, psi] = coeff(k, c, d, a, b)

% evaluation of Psi
psi = (d-c);

temp = double((b - a) ./ (k .* pi) ...
        .* ( sin(k .* pi .* (d - a) ./ (b - a)) ...
        - sin(k .* pi .* (c - a) ./ (b - a)) ));

psi(2:end,:) = temp(2:end,:);

% evaluation of Chi
chi = (exp(d)-exp(c));
chi1 = 1 ./ ( 1 + ( k .* pi ./ (b - a) ) .^ 2 );
chi2 = exp(d) .* cos(k .* pi .* (d - a) ./ (b - a)) ...
        - exp(c) .* cos(k .* pi .* (c - a) ./ (b - a));
chi3 = k .* pi ./ (b - a) .* ...
        ( exp(d) .* sin(k .* pi .* (d - a) ./ (b - a)) ...
        - exp(c) .* sin(k .* pi .* (c - a) ./ (b - a)) );

temp = chi1 .* (chi2 + chi3);

chi(2:end,:) = temp(2:end,:);

end
```

B.4 GUI

B.4.1 GUI.m

Codice che implementa la GUI sviluppata.

```
function varargout = GUI(varargin)
```

```
% GUI MATLAB code for GUI.fig
%      GUI, by itself, creates a new GUI or raises the existing
%      singleton*.
%
%      H = GUI returns the handle to a new GUI or the handle to
%      the existing singleton*.
%
%      GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI.M with the given input arguments.
%
%      GUI('Property','Value',...) creates a new GUI or raises the
%      existing singleton*.  Starting from the left, property value
    pairs are
%      applied to the GUI before GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
    application
%      stop.  All inputs are passed to GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
    only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI

% Last Modified by GUIDE v2.5 03-Jun-2014 21:36:18

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before GUI is made visible.
function GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI (see VARARGIN)

% Choose default command line output for GUI
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

addpath ../FUNZIONI_AUSILIARIE/CALIBRAZIONE
addpath ../FUNZIONI_AUSILIARIE/INPUT
addpath ../FUNZIONI_AUSILIARIE/PRICING_MC
addpath ../FUNZIONI_AUSILIARIE/VANILLA_PRICING
addpath ../FUNZIONI_AUSILIARIE/BARRIER_PRICING
addpath ../FUNZIONI_AUSILIARIE/ASIAN_PRICING

% UIWAIT makes GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox1
    contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in Selezione_modello.
function Selezione_modello_Callback(hObject, eventdata, handles)
% hObject    handle to Selezione_modello (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = cellstr(get(hObject,'String')) returns
    Selezione_modello contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    Selezione_modello
contents = cellstr(get(hObject,'String'));
modello = contents{get(hObject,'Value')}; %returns popupmenu1 contents
    as cell array
handles.calibrazione.modello = modello;
guidata(hObject,handles);

if strcmp(modello, 'BS')
    set(handles.param1, 'String', 'sigma');
    set(handles.param2, 'String', '');
    set(handles.param3, 'String', '');
    set(handles.param4, 'String', '');
    set(handles.param5, 'String', '');
    set(handles.param1ott, 'String', '0.0');
    set(handles.param2ott, 'String', '');
    set(handles.param3ott, 'String', '');
    set(handles.param4ott, 'String', '');
    set(handles.param5ott, 'String', '');
elseif strcmp(modello, 'Merton')
    set(handles.param1, 'String', 'sigma');
    set(handles.param2, 'String', 'lambda');
    set(handles.param3, 'String', 'mu');
    set(handles.param4, 'String', 'delta');
    set(handles.param5, 'String', '');
    set(handles.param1ott, 'String', '0.0');
    set(handles.param2ott, 'String', '0.0');
    set(handles.param3ott, 'String', '0.0');
    set(handles.param4ott, 'String', '0.0');
    set(handles.param5ott, 'String', '');
elseif strcmp(modello, 'Kou')
    set(handles.param1, 'String', 'sigma');
    set(handles.param2, 'String', 'lambda');
    set(handles.param3, 'String', 'lambdap');
    set(handles.param4, 'String', 'lambdam');
    set(handles.param5, 'String', 'p');
    set(handles.param1ott, 'String', '0.0');
    set(handles.param2ott, 'String', '0.0');
    set(handles.param3ott, 'String', '0.0');
    set(handles.param4ott, 'String', '0.0');
    set(handles.param5ott, 'String', '0.0');
elseif strcmp(modello, 'VG')
    set(handles.param1, 'String', 'sigma');
    set(handles.param2, 'String', 'k');
    set(handles.param3, 'String', 'theta');
    set(handles.param4, 'String', '');
    set(handles.param5, 'String', '');
    set(handles.param1ott, 'String', '0.0');
    set(handles.param2ott, 'String', '0.0');
    set(handles.param3ott, 'String', '0.0');
    set(handles.param4ott, 'String', '');
    set(handles.param5ott, 'String', '');
elseif strcmp(modello, 'NIG')
    set(handles.param1, 'String', 'sigma');
    set(handles.param2, 'String', 'k');
    set(handles.param3, 'String', 'theta');
```

```
    set(handles.param4, 'String', '');
    set(handles.param5, 'String', '');
    set(handles.param1ott, 'String', '0.0');
    set(handles.param2ott, 'String', '0.0');
    set(handles.param3ott, 'String', '0.0');
    set(handles.param4ott, 'String', '');
    set(handles.param5ott, 'String', '');
elseif strcmp(modello, 'Heston')
    set(handles.param1, 'String', 'V0');
    set(handles.param2, 'String', 'theta');
    set(handles.param3, 'String', 'k');
    set(handles.param4, 'String', 'nu');
    set(handles.param5, 'String', 'rho');
    set(handles.param1ott, 'String', '0.0');
    set(handles.param2ott, 'String', '0.0');
    set(handles.param3ott, 'String', '0.0');
    set(handles.param4ott, 'String', '0.0');
    set(handles.param5ott, 'String', '0.0');
end

% --- Executes during object creation, after setting all properties.
function Selezione_modello_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Selezione_modello (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in Selezione_metodo.
function Selezione_metodo_Callback(hObject, eventdata, handles)
% hObject    handle to Selezione_metodo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
    Selezione_metodo contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    Selezione_metodo
contents = cellstr(get(hObject,'String'));
metodo = contents{get(hObject,'Value')}; %returns popupmenu1 contents
    as cell array
handles.calibrazione.metodo = metodo;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function Selezione_metodo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Selezione_metodo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called
```

```
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Avvio_calibrazione.
function Avvio_calibrazione_Callback(hObject, eventdata, handles)
% hObject    handle to Avvio_calibrazione (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
metodo = handles.calibrazione.metodo;
modello = handles.calibrazione.modello;

%%%%%%%
% CALIBRAZIONE VERA E PROPRIA
dati = handles.DatiInput;
Lettura_Dati;
t_inizio = tic;
Calibrazione;
t_elapsed = toc(t_inizio)
param_ott
dist_ott = RMSE(param_ott)
%%%%%%%

handles.calibrazione.param_ott = param_ott;
handles.calibrazione.S0 = S0;
handles.calibrazione.T = T;
handles.calibrazione.r = r;
handles.calibrazione.d = d;

guidata(hObject,handles);

set(handles.S0_scrivi, 'String', S0);
set(handles.T_scrivi, 'String', T);
set(handles.r_scrivi, 'String', r);
set(handles.d_scrivi, 'String', d);

set(handles.tcalibrazione, 'String', t_elapsed);
if strcmp(modello, 'BS')
    set(handles.param1ott, 'String', param_ott(1));
    set(handles.param2ott, 'String', '');
    set(handles.param3ott, 'String', '');
    set(handles.param4ott, 'String', '');
    set(handles.param5ott, 'String', '');
elseif strcmp(modello, 'Merton')
    set(handles.param1ott, 'String', param_ott(1));
    set(handles.param2ott, 'String', param_ott(2));
    set(handles.param3ott, 'String', param_ott(3));
    set(handles.param4ott, 'String', param_ott(4));
    set(handles.param5ott, 'String', '');
elseif strcmp(modello, 'Kou')
    set(handles.param1ott, 'String', param_ott(1));
    set(handles.param2ott, 'String', param_ott(2));
    set(handles.param3ott, 'String', param_ott(3));
    set(handles.param4ott, 'String', param_ott(4));
```

```
    set(handles.param5ott, 'String', param_ott(5));
elseif strcmp(modello, 'VG')
    set(handles.param1ott, 'String', param_ott(1));
    set(handles.param2ott, 'String', param_ott(2));
    set(handles.param3ott, 'String', param_ott(3));
    set(handles.param4ott, 'String', '');
    set(handles.param5ott, 'String', '');
elseif strcmp(modello, 'NIG')
    set(handles.param1ott, 'String', param_ott(1));
    set(handles.param2ott, 'String', param_ott(2));
    set(handles.param3ott, 'String', param_ott(3));
    set(handles.param4ott, 'String', '');
    set(handles.param5ott, 'String', '');
elseif strcmp(modello, 'Heston')
    set(handles.param1ott, 'String', param_ott(1));
    set(handles.param2ott, 'String', param_ott(2));
    set(handles.param3ott, 'String', param_ott(3));
    set(handles.param4ott, 'String', param_ott(4));
    set(handles.param5ott, 'String', param_ott(5));
end

%%%%%%%%%%% PRICING
% --- Executes on button press in Pricing.
function Pricing_Callback(hObject, eventdata, handles)
% hObject    handle to Pricing (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% RECUPERO PARAMETRI SERIE STORICA CALIBRAZIONE
T = handles.calibrazione.T;
d = handles.calibrazione.d;
r = handles.calibrazione.r;
S0 = handles.calibrazione.S0;

% RECUPERO PARAMETRI UTENTE IN PRICING
flagCP = handles.pricing.flagCP;
param_modello = handles.calibrazione.param_ott;
modello = handles.calibrazione.modello;
Nsim = 1e4;
Nsteps = 252;

tipo_opzione = handles.pricing.TipoOpzione;
metodo_pricing = handles.pricing.MetodoPricing;
t_inizio_pricing = tic;

% Inizializzo a NaN Prezzo e IC, se vengono calcolati saranno
    sovrascritti
Prezzo = NaN;
IC = NaN;

% PRICING
if strcmp(tipo_opzione, 'Europea')
    % Lettura dati
    K = str2num(handles.pricing.K);
    % Pricing
    if strcmp(metodo_pricing, 'Monte Carlo')
        [Prezzo, IC, ampiezza] = MC_Pricing_Europea(S0, K, r, d, T,
    flagCP, param_modello, modello, Nsim, Nsteps);
```

```
        else
            [PrezzoCall, PrezzoPut] = Vanilla_Pricing(S0, r, d, T, K,
        param_modello, modello, metodo_pricing);
            if flagCP == 1
                Prezzo = PrezzoCall;
            else
                Prezzo = PrezzoPut;
            end
        end
    elseif strcmp(tipo_opzione, 'Barriera')
        % Lettura dati
        K = str2num(handles.pricing.K);
        D = str2num(handles.pricing.D);
        U = str2num(handles.pricing.U);
        % Pricing
        if strcmp(metodo_pricing, 'Monte Carlo')
            [Prezzo, IC, ampiezza] = MC_Pricing_Barriera(S0, K, r, d, T,
        D, U, flagCP, param_modello, modello, Nsim, Nsteps)
        elseif strcmp(metodo_pricing, 'CONV') || strcmp(metodo_pricing,
        'COS')
            Prezzo = Barrier_Pricing(S0, r, d, T, U, D, K, flagCP,
        param_modello, modello, metodo_pricing);
        end
    elseif strcmp(tipo_opzione, 'LookBack Fixed Strike')
        % Lettura dati
        K = str2num(handles.pricing.K);
        % Pricing, funziona solo col MC
        if strcmp(metodo_pricing, 'Monte Carlo')
            [Prezzo, IC, ampiezza] = MC_Pricing_LookBack_FixedStrike(S0,
        K, r, d, T, flagCP, param_modello, modello, Nsim, Nsteps);
        end
    elseif strcmp(tipo_opzione, 'LookBack Floating Strike')
        % Non occorrono dati
        % Pricing, funziona solo col MC
        if strcmp(metodo_pricing, 'Monte Carlo')
            [Prezzo, IC, ampiezza] =
        MC_Pricing_LookBack_FloatingStrike(S0, r, d, T, flagCP,
        param_modello, modello, Nsim, Nsteps);
        end
    elseif strcmp(tipo_opzione, 'Asian Fixed Strike Aritm')
        % Lettura dati
        K = str2num(handles.pricing.K);
        % Pricing, funziona solo col MC
        if strcmp(metodo_pricing, 'Monte Carlo')
            [Prezzo, IC, ampiezza] = MC_Pricing_AsianAritm_FixedStrike(S0,
        K, r, d, T, flagCP, param_modello, modello, Nsim, Nsteps);
        end
    elseif strcmp(tipo_opzione, 'Asian Floating Strike Aritm')
        % Non occorrono dati
        % Pricing, funziona solo col MC
        if strcmp(metodo_pricing, 'Monte Carlo')
            [Prezzo, IC, ampiezza] =
        MC_Pricing_AsianAritm_FloatingStrike(S0, r, d, T, flagCP,
        param_modello, modello, Nsim, Nsteps);
        end
    elseif strcmp(tipo_opzione, 'Asian Fixed Strike Geom')
        % Lettura dati
        K = str2num(handles.pricing.K);
```

```
    % Pricing, funziona solo col MC
    if strcmp(metodo_pricing, 'Monte Carlo')
        [Prezzo, IC, ampiezza] = MC_Pricing_AsianGeom_FixedStrike(S0,
    K, r, d, T, flagCP, param_modello, modello, Nsim, Nsteps);
    elseif strcmp(metodo_pricing, 'COS')
        Prezzo = AsianGeom_COS(S0, r, d, T, K, flagCP, param_modello,
    modello);
    end
elseif strcmp(tipo_opzione, 'Asian Floating Strike Geom')
    % Non occorrono dati
    % Pricing, funziona solo col MC
    if strcmp(metodo_pricing, 'Monte Carlo')
        [Prezzo, IC, ampiezza] =
    MC_Pricing_AsianGeom_FloatingStrike(S0, r, d, T, flagCP,
    param_modello, modello, Nsim, Nsteps);
    end
end

t_elapsed_pricing = toc(t_inizio_pricing);

set(handles.tempo_pricing, 'String', t_elapsed_pricing);
set(handles.Prezzo, 'String', Prezzo);
set(handles.IC, 'String', IC);

% --- Executes on selection change in CallPut.
function CallPut_Callback(hObject, eventdata, handles)
% hObject    handle to CallPut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns CallPut
    contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    CallPut
contents = cellstr(get(hObject,'String'));
flagCP = contents{get(hObject,'Value')}; %returns popupmenu1 contents
    as cell array
if strcmp(flagCP, 'Call')
    handles.pricing.flagCP = 1;
else
    handles.pricing.flagCP = -1;
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function CallPut_CreateFcn(hObject, eventdata, handles)
% hObject    handle to CallPut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function barrieraUp_Callback(hObject, eventdata, handles)
% hObject    handle to barrieraUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of barrieraUp as text
%        str2double(get(hObject,'String')) returns contents of
    barrieraUp as a double
contents = cellstr(get(hObject,'String'));
U = cell2mat(contents);
handles.pricing.U = U;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function barrieraUp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to barrieraUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function barrieraDown_Callback(hObject, eventdata, handles)
% hObject    handle to barrieraDown (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of barrieraDown as text
%        str2double(get(hObject,'String')) returns contents of
    barrieraDown as a double
contents = cellstr(get(hObject,'String'));
D = cell2mat(contents);
handles.pricing.D = D;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function barrieraDown_CreateFcn(hObject, eventdata, handles)
% hObject    handle to barrieraDown (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function Strike_Callback(hObject, eventdata, handles)
% hObject    handle to Strike (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Strike as text
%        str2double(get(hObject,'String')) returns contents of Strike
    as a double
contents = cellstr(get(hObject,'String'));
K = cell2mat(contents);
handles.pricing.K = K;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function Strike_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Strike (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called


% --- Executes on selection change in TipoOpzione.
function TipoOpzione_Callback(hObject, eventdata, handles)
% hObject    handle to TipoOpzione (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns TipoOpzione
    contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    TipoOpzione
contents = cellstr(get(hObject,'String'));
tipo_opzione = contents{get(hObject,'Value')}; %returns popupmenu1
    contents as cell array
handles.pricing.TipoOpzione = tipo_opzione;
guidata(hObject,handles);

% RENDO VISIBILI SOLO I PARAMETRI NECESSARI PER UNA DATA OPZIONE
if strcmp(tipo_opzione, 'Europea')
    set(handles.Strike, 'visible', 'on')
    set(handles.Striketext, 'visible', 'on')
    set(handles.BarrieraUpText, 'visible', 'off')
    set(handles.barrieraUp, 'visible', 'off')
```

```
    set(handles.BarrieraDownText, 'visible', 'off')
    set(handles.barrieraDown, 'visible', 'off')
elseif strcmp(tipo_opzione, 'Barriera')
    set(handles.Strike, 'visible', 'on')
    set(handles.Striketext, 'visible', 'on')
    set(handles.BarrieraUpText, 'visible', 'on')
    set(handles.barrieraUp, 'visible', 'on')
    set(handles.BarrieraDownText, 'visible', 'on')
    set(handles.barrieraDown, 'visible', 'on')
elseif strcmp(tipo_opzione, 'LookBack Fixed Strike')
    set(handles.Strike, 'visible', 'on')
    set(handles.Striketext, 'visible', 'on')
    set(handles.BarrieraUpText, 'visible', 'off')
    set(handles.barrieraUp, 'visible', 'off')
    set(handles.BarrieraDownText, 'visible', 'off')
    set(handles.barrieraDown, 'visible', 'off')
elseif strcmp(tipo_opzione, 'LookBack Floating Strike')
    set(handles.Strike, 'visible', 'off')
    set(handles.Striketext, 'visible', 'off')
    set(handles.BarrieraUpText, 'visible', 'off')
    set(handles.barrieraUp, 'visible', 'off')
    set(handles.BarrieraDownText, 'visible', 'off')
    set(handles.barrieraDown, 'visible', 'off')
elseif strcmp(tipo_opzione, 'Asian Fixed Strike Aritm')
    set(handles.Strike, 'visible', 'on')
    set(handles.Striketext, 'visible', 'on')
    set(handles.BarrieraUpText, 'visible', 'off')
    set(handles.barrieraUp, 'visible', 'off')
    set(handles.BarrieraDownText, 'visible', 'off')
    set(handles.barrieraDown, 'visible', 'off')
elseif strcmp(tipo_opzione, 'Asian Floating Strike Aritm')
    set(handles.Strike, 'visible', 'off')
    set(handles.Striketext, 'visible', 'off')
    set(handles.BarrieraUpText, 'visible', 'off')
    set(handles.barrieraUp, 'visible', 'off')
    set(handles.BarrieraDownText, 'visible', 'off')
    set(handles.barrieraDown, 'visible', 'off')
elseif strcmp(tipo_opzione, 'Asian Fixed Strike Geom')
    set(handles.Strike, 'visible', 'on')
    set(handles.Striketext, 'visible', 'on')
    set(handles.BarrieraUpText, 'visible', 'off')
    set(handles.barrieraUp, 'visible', 'off')
    set(handles.BarrieraDownText, 'visible', 'off')
    set(handles.barrieraDown, 'visible', 'off')
elseif strcmp(tipo_opzione, 'Asian Floating Strike Geom')
    set(handles.Strike, 'visible', 'off')
    set(handles.Striketext, 'visible', 'off')
    set(handles.BarrieraUpText, 'visible', 'off')
    set(handles.barrieraUp, 'visible', 'off')
    set(handles.BarrieraDownText, 'visible', 'off')
    set(handles.barrieraDown, 'visible', 'off')
end


% --- Executes during object creation, after setting all properties.
function TipoOpzione_CreateFcn(hObject, eventdata, handles)
% hObject    handle to TipoOpzione (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in DatiInput.
function DatiInput_Callback(hObject, eventdata, handles)
% hObject    handle to DatiInput (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns DatiInput
    contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    DatiInput
contents = cellstr(get(hObject,'String'));
dati_input = contents{get(hObject,'Value')}; %returns popupmenu1
    contents as cell array
handles.DatiInput = dati_input;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function DatiInput_CreateFcn(hObject, eventdata, handles)
% hObject    handle to DatiInput (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in MetodoPricing.
function MetodoPricing_Callback(hObject, eventdata, handles)
% hObject    handle to MetodoPricing (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
    MetodoPricing contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    MetodoPricing
contents = cellstr(get(hObject,'String'));
metodo_pricing = contents{get(hObject,'Value')}; %returns popupmenu1
    contents as cell array
handles.pricing.MetodoPricing = metodo_pricing;
guidata(hObject,handles);
```

```
% RENDO VISIBILE IC SOLO PER MC
if strcmp(metodo_pricing, 'Monte Carlo')
    set(handles.IC, 'visible', 'on')
    set(handles.ICtext, 'visible', 'on')
else
    set(handles.IC, 'visible', 'off')
    set(handles.ICtext, 'visible', 'off')
end

% --- Executes during object creation, after setting all properties.
function MetodoPricing_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MetodoPricing (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

B.5 Funzioni ausiliarie

B.5.1 CharFuncLib.m

Libreria delle funzioni caratteristiche risk-neutral dei modelli utilizzati nella tesi.

```
function V = CharFuncLib(u, r, d, T, param_modello, modello)

if strcmp(modello, 'BS')
    % Parametri modello
    sigma_BS = param_modello(1);
    V = CF_BS(u, T, r, d, sigma_BS);
elseif strcmp(modello, 'Merton')
    % Parametri modello
    sigma_MERTON = param_modello(1);
    lambda_MERTON = param_modello(2);
    mu_MERTON = param_modello(3);
    delta_MERTON = param_modello(4);
    V = CF_Merton(u, T, r, d, sigma_MERTON, lambda_MERTON, mu_MERTON,
    delta_MERTON);
elseif strcmp(modello, 'Kou')
    % Parametri modello
    sigma_KOU = param_modello(1);
    lambda_KOU = param_modello(2);
    lambdap_KOU = param_modello(3);
    lambdam_KOU = param_modello(4);
    p_KOU = param_modello(5);
    V = CF_KOU(u, T, r, d, sigma_KOU, lambda_KOU, lambdap_KOU,
    lambdam_KOU, p_KOU);
elseif strcmp(modello, 'NIG')
    % Parametri modello
    sigma_NIG = param_modello(1);
```

```
    theta_NIG = param_modello(2);
    k_NIG = param_modello(3);
    V = CF_NIG(u, T, r, d, sigma_NIG, theta_NIG, k_NIG);
elseif strcmp(modello, 'VG')
    % Parametri modello
    sigma_VG = param_modello(1);
    theta_VG = param_modello(2);
    k_VG = param_modello(3);
    V = CF_VG(u, T, r, d, sigma_VG, k_VG, theta_VG);
elseif strcmp(modello, 'Heston')
    % Parametri modello
    V0_Heston = param_modello(1);
    theta_Heston = param_modello(2);
    k_Heston = param_modello(3);
    omega_Heston = param_modello(4);
    rho_Heston = param_modello(5);
    V = CF_Heston(u, T, r, d, V0_Heston, theta_Heston, k_Heston,
    omega_Heston, rho_Heston);
end

end

%%
function y = CF_BS(u, T, r, d, sigma)
    y = 1i*u*((r-d-0.5*sigma*sigma)*T) - 0.5*sigma*sigma*u.*u*T;
    y = exp(y);
end

%%
function y = CF_Merton(u, T, r, d, sigma_MERTON, lambda_MERTON,
    mu_MERTON, delta_MERTON)
    e = @(u) - sigma_MERTON^2/2*u.^2 +
    lambda_MERTON*(exp(-delta_MERTON^2 * u.^2 /2 + 1i*mu_MERTON*u) -
    1);
    omega = e(-1i);
    y = 1i * u * ( (r - d - omega) * T ) + T*e(u);
    y = exp(y);
end

%%
function y = CF_KOU(u, T, r, d, sigma_KOU, lambda_KOU, lambdap_KOU,
    lambdam_KOU, p_KOU)
    e = @(u) - sigma_KOU^2/2.*u.^2 + 1i*u*lambda_KOU .* (
    p_KOU./(lambdap_KOU - 1i*u) - (1 - p_KOU)./(lambdam_KOU + 1i*u));
    omega = e(-1i);
    y = 1i * u * ( (r - d - omega) * T ) + T*e(u);
    y = exp(y);
end

%%
function y = CF_VG(u, T, r, d, sigma, nu, theta)
% Variance Gamma
    e = @(u) -1/nu*log(1 - 1i * theta * nu * u + 0.5 * sigma * sigma *
    u .* u * nu);
    omega = e(-1i);
    y = 1i * u * ( (r - d - omega) * T ) + T*e(u);
    y = exp(y);
end
```

```
%%
function y = CF_NIG(u, T, r, d, sigma, theta, k)
    e = @(u) 1/k - 1/k * sqrt(1 + u.*u*sigma^2*k - 2*1i*theta*u*k);
    omega = e(-1i);
    y = 1i * u * ( (r - d - omega) * T ) + T*e(u);
    y = exp(y);
end

%%
function y = CF_Heston(u, T, r, d, V0, theta, kappa, omega, rho)

alfa = -.5*(u.*u + u*1i);
beta = kappa - rho*omega*u*1i;
omega2 = omega * omega;
gamma = .5 * omega2;

D = sqrt(beta .* beta - 4.0 * alfa .* gamma);

bD = beta - D;
eDt = exp(- D * T);

G = bD ./ (beta + D);
B = (bD ./ omega2) .* ((1.0 - eDt) ./ (1.0 - G .* eDt));
psi = (G .* eDt - 1.0) ./(G - 1.0);
A = ((kappa * theta) / (omega2)) * (bD * T - 2.0 * log(psi));

y = A + B*V0 + 1i*u*((r-d)*T);

y = exp(y);

end
```

B.5.2 impvol.m

Funzione che restituisce la volatilità implicita di un prezzo, performa meglio della `blsimpv` già presente in `Matlab`, utilizza l'algoritmo di bisezione per la ricerca degli zeri.

```
% Function that returns implied volatility using Bisection algorithm
    it is
%   faster than blsimpv and never returns NaN.
%   sigma = impvol(C, S, K, r, T)
%
%   Inputs:
%   C       - market Call prices (can be a vector)
%   S       - value of underlying
%   K       - stike prices (can be a vector)
%   r       - risk free rate
%   T       - maturity
%
%   Outputs:
%   sigma   - implied volatility

function sigma = impvol(C, S, K, r, T, d)
n = length(C);
```

```
f = @(x) blsprice(S, K, r, T, x, d) - C;
sigma = bisezione(f, n);
end

% Algoritmo di bisezione
function mm = bisezione(f, n)
    aa = zeros(n, 1);           % Estremi inferiori
    bb = 2*ones(n, 1);          % Estremi superiori
    for count = 1:1e2
        fa = f(aa);
        mm = (bb - aa)/2 + aa;  % Punti medi
        fm = f(mm);
        idx = (fa.*fm) > 0;
        aa = aa.*(idx == 0) + mm.*(idx == 1);
        bb = mm.*(idx == 0) + bb.*(idx == 1);
    end
end
```

B.5.3 AssetLib.m

Libreria delle simulazioni del sottostante con i vari processi stocastici utilizzati nella tesi. Viene utilizzata nel pricing Monte Carlo.

```
function S = AssetLib(S0, r, d, T, param_modello, modello, Nsim,
    Nsteps)

if strcmp(modello, 'BS')
    % Parametri modello
    sigma_BS = param_modello(1);
    S = AssetBS(S0, r, d, sigma_BS, T, Nsim, Nsteps);
elseif strcmp(modello, 'Merton')
    % Parametri modello
    sigma_MERTON = param_modello(1);
    lambda_MERTON = param_modello(2);
    mu_MERTON = param_modello(3);
    delta_MERTON = param_modello(4);
    S = AssetMerton(S0, r, d, sigma_MERTON, T, lambda_MERTON,
    mu_MERTON, delta_MERTON, Nsim, Nsteps);
elseif strcmp(modello, 'Kou')
    % Parametri modello
    sigma_KOU = param_modello(1);
    lambda_KOU = param_modello(2);
    lambdap_KOU = param_modello(3);
    lambdam_KOU = param_modello(4);
    p_KOU = param_modello(5);
    [S] = AssetKou(S0, r, d, sigma_KOU, T, lambda_KOU, p_KOU,
    lambdap_KOU, lambdam_KOU, Nsim, Nsteps);
elseif strcmp(modello, 'NIG')
    % Parametri modello
    sigma_NIG = param_modello(1);
    theta_NIG = param_modello(2);
    k_NIG = param_modello(3);
    S = AssetNIG(S0, r, d, sigma_NIG, T, k_NIG, theta_NIG, Nsim,
    Nsteps);
elseif strcmp(modello, 'VG')
    % Parametri modello
```

```
    sigma_VG = param_modello(1);
    theta_VG = param_modello(2);
    k_VG = param_modello(3);
    [S] = AssetVG(S0, r, d, sigma_VG, T, k_VG, theta_VG, Nsim, Nsteps);
elseif strcmp(modello, 'Heston')
    % Parametri modello
    V0_Heston = param_modello(1);
    theta_Heston = param_modello(2);
    k_Heston = param_modello(3);
    omega_Heston = param_modello(4);
    rho_Heston = param_modello(5);
    [S] = AssetHeston(S0, r, d, V0_Heston, rho_Heston, k_Heston,
    theta_Heston, omega_Heston, T, Nsim, Nsteps);
end

end

%%
function S = AssetBS(S0, r, d, sigma, T, Nsim, Nsteps)

t = linspace(0, T, Nsteps+1);
dt = t(2) - t(1);

temp = randn(Nsim, Nsteps);

X = zeros(Nsim, Nsteps + 1);

for i = 1:Nsteps
    X(:, i+1) = X(:, i) + (r-d-sigma^2/2)*dt +
    sigma*sqrt(dt)*temp(:,i);
end

S = S0 * exp(X);

end

%%
function [S] = AssetMerton(S0, r, d, sigma, T, lambda, nu, delta,
    Nsim, Nsteps)

t = linspace(0, T, Nsteps+1);
dt = T/Nsteps;

S = zeros(Nsim, Nsteps+1);

for iter = 1:Nsim
    NT = icdf('Poisson', rand, lambda*T);
    istante_salto = sort(rand(NT, 1)*T);

    if NT ~= 0
        for i = 1:NT
            [minimo, index] = min(abs(istante_salto(i) - t(2:end)));
            indice(i) = index(1);
        end
        istante_salto_new = t(indice+1)';
    end

    temp = randn(Nsteps, 1);
```

```
    X = zeros(Nsteps+1, 1);
    for i = 1:Nsteps
        X(i+1) = X(i) + sigma*sqrt(dt)*temp(i);
        for j = 1:NT
            if istante_salto_new(j) == t(i+1);
                X(i+1) = X(i+1) + nu + delta*randn(1);
            end
        end
    end

    psi = @(u) ExpCarattMerton(u, sigma, lambda, nu, delta);
    S(iter, :) = S0 * exp( (r-d-psi(-1i))*t' + X );
end

end

%%
function [S] = AssetKou(S0, r, d, sigma, T, lambda, p, lambdap,
    lambdam, Nsim, Nsteps)

t = linspace(0, T, Nsteps+1);
dt = T/Nsteps;

S = zeros(Nsim, Nsteps+1);

for iter = 1:Nsim
    NT = icdf('Poisson', rand, lambda*T);
    istante_salto = sort(rand(NT, 1)*T);

    if NT ~= 0
        for i = 1:NT
            [minimo, index] = min(abs(istante_salto(i) - t(2:end)));
            indice(i) = index(1);
        end
        istante_salto_new = t(indice+1)';    %ho messo +1 perche sopra
    avevo t(2:end)
    end

    X = zeros(Nsteps + 1, 1);
    temp = randn(Nsteps, 1);
    for i = 1:Nsteps
        X(i+1) = X(i) + sigma*sqrt(dt)*temp(i);
        for j = 1:NT
            if istante_salto_new(j) == t(i+1)
                if rand < p    % Salto positivo
                    Y = icdf('Exp', rand, 1/lambdap);
                else           % Salto negativo
                    Y = -icdf('Exp', rand, 1/lambdam);
                end
                X(i+1) = X(i+1) + Y;
            end
        end
    end

    psi = @(u) ExpCarattKOU(u, sigma, lambda, lambdap, lambdam, p);
    S(iter, :) = S0 * exp( (r-d-psi(-1i))*t' + X );
end
```

```
end

%%
function [S] = AssetNIG(S0, r, d, sigma, T, k_NIG, theta_NIG, Nsim,
    Nsteps)

t = linspace(0, T, Nsteps+1);
dt = T/Nsteps;

S = zeros(Nsim, Nsteps + 1);

for iter = 1:Nsim
    mu = dt;
    lambda = dt^2/k_NIG;
    dS = zeros(Nsteps, 1);
    for i = 1:Nsteps
        N_IG = randn;
        Y_IG = N_IG^2;
        X_IG = mu + mu^2*Y_IG/(2*lambda) - ...
            mu/(2*lambda)*sqrt(4*mu*lambda*Y_IG + mu^2*Y_IG^2);
        U_IG = rand;
        P_IG = mu/(X_IG + mu);
        dS(i) = X_IG.*(U_IG <= P_IG) + mu^2./X_IG *(U_IG > P_IG);
    end

    X = zeros(Nsteps + 1, 1);   % Giusto che parta da 0
    N = randn(Nsteps, 1);       % Simulo le Normali standard

    for i = 1:Nsteps
        X(i+1) = X(i) + theta_NIG*dS(i) + sigma*N(i)*sqrt(dS(i));
    end

    psi = @(u) ExpCarattNIG(u, sigma, theta_NIG, k_NIG);
    S(iter, :) = S0 * exp( (r-d-psi(-1i))*t' + X );
end

end

%%
function [S] = AssetVG(S0, r, d, sigma, T, k_VG, theta_VG, Nsim,
    Nsteps)

t = linspace(0, T, Nsteps+1);
dt = T/Nsteps;

S = zeros(Nsim, Nsteps + 1);

for iter = 1:Nsim
    u = rand(size(t));
    dS = icdf('Gamma', u, dt/k_VG, 1);
    dS = k_VG*dS;

    X = zeros(Nsteps + 1, 1);  % Giusto che parta da 0
    N = randn(Nsteps, 1);      % Simulo le Normali standard

    for i = 1:Nsteps
        X(i+1) = X(i) + theta_VG*dS(i) + sigma*N(i)*sqrt(dS(i));
    end
```

```
    psi = @(u) ExpCarattVG(u, sigma, theta_VG, k_VG);
    S(iter, :) = S0 * exp( (r-d-psi(-1i))*t' + X );
end

end

%%
function [S] = AssetHeston(S0, r, d, V0, rho, k, theta, epsilon, T,
    Nsim, Nsteps)

t = linspace(0, T, Nsteps+1);
dt = t(2) - t(1);

Zx = randn(Nsim, Nsteps);
Zv = randn(Nsim, Nsteps);
Zv = rho * Zx + sqrt(1 - rho^2) * Zv;   % sovrascrivo per introdurre
    la correlazione

S = zeros(Nsim, Nsteps + 1);
V = zeros(Nsim, Nsteps + 1);
S(:, 1) = S0;
V(:, 1) = V0;

X = zeros(Nsim, Nsteps + 1);
for i = 1:Nsteps
    % Calcolare S
    X(:, i+1) = X(:, i) + (r - d - 0.5*V(:, i))*dt + sqrt(V(:, i)*dt)
    .* Zx(:, i);
    % Calcolare V
    V(:, i+1) = V(:, i) + k*(theta - V(:, i))*dt + sqrt(V(:,
    i)*dt)*epsilon.*Zv(:, i);
    V(:, i+1) = max(V(:, i+1), 0);
end

S = S0*exp(X);

end

%%
function V = ExpCarattMerton(u, sigma, lambda, mu, delta)
V = - sigma^2/2*u.^2 + lambda*(exp(-delta^2 * u.^2 /2 + 1i*mu*u) - 1);
end

%%
function V = ExpCarattKOU(u, sigma, lambda, lambdap, lambdam, p)
V = - sigma^2/2.*u.^2 + 1i*u*lambda .* ...
    ( p./(lambdap - 1i*u) - (1 - p)./(lambdam + 1i*u));
end

%%
function V = ExpCarattNIG(u, sigma, theta, k)
V = 1/k - 1/k * sqrt(1 + u.^2*sigma^2*k - 2*1i*theta*u*k);
end

%%
function V = ExpCarattVG(u, sigma, theta, k)
V = -1/k * log(1 + (u.^2*sigma^2*k)/2 - 1i*theta*k*u);
```

```
end
```

Bibliografia

[1] P. Baldi. *Equazioni differenziali stocastiche e applicazioni*. Quaderni dell'Unione Matematica Italiana. Pitagora, 2000. ISBN: 9788837112110. URL: http://books.google.it/books?id=qkHMPQAACAAJ.

[2] E. Barucci. *Ingegneria finanziaria. Un'introduzione quantitativa*. I Manuali. EGEA, 2009. ISBN: 9788823820951. URL: http://books.google.it/books?id=fWeEPgAACAAJ (cit. a p. 6).

[3] T. Björk. *Arbitrage Theory in Continuous Time*. Oxford Finance Series. Oxford University Press, Incorporated, 2009. ISBN: 9780199574742. URL: http://books.google.it/books?id=Z9rHMgEACAAJ (cit. alle pp. 6, 23, 28, 29).

[4] Peter Carr e Dilip B. Madan. «Option Valuation Using the Fast Fourier Transform». In: *JOURNAL OF COMPUTATIONAL FINANCE* 2 (1999), pp. 61–73 (cit. alle pp. xi, xiii, 27, 34).

[5] R. Cont e P. Tankov. *Financial Modelling with Jump Processes*. Chapman & Hall/Crc Financial Mathematics Series. Chapman & Hall/CRC, 2012. ISBN: 9781420082197. URL: http://books.google.it/books?id=-fZtKgAACAAJ (cit. alle pp. 6, 9, 24, 33).

[6] F. Fang e C. W. Oosterlee. «A Novel Pricing Method for European Options Based on Fourier-Cosine Series Expansions». In: *SIAM J. Sci. Comput.* 31.2 (nov. 2008), pp. 826–848. ISSN: 1064-8275. DOI: 10.1137/080718061. URL: http://dx.doi.org/10.1137/080718061 (cit. alle pp. xi, xiii, 27, 44, 46, 69).

[7] F. Fang e C. W. Oosterlee. «Pricing Early-exercise and Discrete Barrier Options by Fourier-cosine Series Expansions». In: *Numer. Math.* 114.1 (ott. 2009), pp. 27–62. ISSN: 0029-599X. DOI: 10.1007/s00211-009-0252-4. URL: http://dx.doi.org/10.1007/s00211-009-0252-4 (cit. alle pp. xi, xiii, 27, 44, 48).

[8] E.R. Gianin e C. Sgarra. *Esercizi di finanza matematica*. Collana Unitext. Springer, 2007. ISBN: 9788847006102. URL: http://books.google.it/books?id=Z2O9o006jZcC.

[9] J. Jacod e P.E. Protter. *Probability Essentials*. Universitext (1979). Springer Berlin Heidelberg, 2003. ISBN: 9783540438717. URL: http://books.google.de/books?id=OK%5C_d-w18EVgC (cit. alle pp. 3, 31).

[10] J. Kienitz e D. Wetterau. *Financial Modelling: Theory, Implementation and Practice with MATLAB Source*. The Wiley Finance Series. Wiley, 2012. ISBN: 9781118413319. URL: http://books.google.it/books?id=fZpu8-sNPo8C.

[11] R. Lord et al. «A Fast and Accurate FFT-Based Method for Pricing Early-Exercise Options Under Levy Processes». In: *SIAM J. Sci. Comput.* 30.4 (apr. 2008), pp. 1678–1705. ISSN: 1064-8275. DOI: 10.1137/070683878. URL: http://dx.doi.org/10.1137/070683878 (cit. alle pp. xi, xiii, 27, 39, 41).

[12] A.J. McNeil, R. Frey e P. Embrechts. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton Series in Finance. Princeton University Press, 2010. ISBN: 9781400837571. URL: http://books.google.it/books?id=vgy98mM9zQUC.

[13] P.J. Schönbucher. *Credit Derivatives Pricing Models: Models, Pricing and Implementation*. The Wiley Finance Series. Wiley, 2003. ISBN: 9780470868171. URL: http://books.google.it/books?id=YDeJ%5C_Kv8QZwC.

[14] R. Seydel. *Tools for Computational Finance*. Universitext. Springer, 2009. ISBN: 9783540929291. URL: http://books.google.it/books?id=1r8uixovrtQC.

[15] P. Wilmott, J.N. Dewynne e S. Howison. *Option Pricing: Mathematical Models and Computation*. Oxford Financial, 1998. URL: http://books.google.it/books?id=cGuGcgAACAAJ.

[16] Bowen Zhang e Cornelis W. Oosterlee. «Efficient Pricing of European-Style Asian Options under Exponential Lévy Processes Based on Fourier Cosine Expansions.» In: *SIAM J. Financial Math.* 4.1 (2013), pp. 399–426. URL: http://dblp.uni-trier.de/db/journals/siamfm/siamfm4.html#ZhangO13 (cit. alle pp. 44, 49).