Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione - Milano Leonardo

Laurea Magistrale in Ingegneria Matematica

Valutazione di derivati tramite il metodo FST

Relatore:

Prof. Daniele Marazzina

Tesi di Laurea di:

Marco Nanni

Matr. n. 783055

Anno Accademico 2012-2013

I Wish You Were Here

Sommario

Negli anni tra il 1989 ed il 1992 il mercato degli strumenti derivati ha subito una brusca espansione; nel 2010 il valore complessivo delle attività sottostanti i derivati ammontava a circa 670000 miliardi di dollari, superando di gran lunga il mercato azionario. Di conseguenza, la modellistica finalizzata al pricing di tali strumenti è diventata sempre più importante, portando allo sviluppo di quell'area che oggi chiamiamo ingegneria finanziaria.

Lo scopo di questa tesi è presentare un metodo di pricing che sia versatile, preciso e veloce. Per questo motivo è stato scelto il metodo FST (Fourier Space Time-stepping) ideato da Jackson, Jaimungal e Surkov nel 2007. Tale metodo si basa sull'utilizzo dell'algoritmo FFT (Fast Fourier Transform) e sulla conoscenza della funzione caratteristica del processo sottostante. Verrà presentata la derivazione di tale metodo e la sua applicazione a vari tipi di opzioni e modelli; verranno inoltre studiate le proprietà di convergenza, stabilità e precisione (Capitoli 2 e 3). Successivamente saranno presentate alcune estensioni del metodo (Capitolo 4) ai modelli a volatilità stocastica e all'equazione forward di Dupire, generalizzata ai processi con salto. I risultati ottenuti sono risultati in linea con le caratteristiche sopra elencate.

Infine viene dato ampio spazio all'implementazione del metodo su scheda grafica (Capitolo 5), tramite la tecnologia CUDA. I risultati ottenuti consentono di affermare che tale metodo, grazie alla natura altamente parallelizzabile della FFT, è in grado di sfruttare a pieno i nuovi hardware multicore, caratteristica che lo rende quindi altamente performante.

Abstract

The financial derivatives market had a huge expansion between 1989 and 1992. During the 2010, the total amount of the derivatives' underlying activities was about 670000 bilions of dollars, overtaking the equity market. Consequently, different pricing models have been developed and this has led to the growth of the financial engineering.
The aim of this Thesis is to present an high performing pricing method which should be versatile and accurate. For this reason, it has been chosen the FST (Fourier Space Time-stepping) method, created by Jackson, Jaimungal and Surkov in 2007. This is based on the FFT (Fast Fourier Transform) algorithm and on the knowledge of the characteristic function of the process. The method is derived analytically and it is applied to different types of options and models, studying the properties of convergence, stability and precision (Chapter 2 and 3). Moreover, the FST method is extended to stochastic volatility models and to Dupire's equation for jump processes (Chapter 4). The obtained results are alligned with the previous expected characteristics.
Finally, the method is implemented on Graphic Processing Unit with CUDA technology (Chapter 5). Thanks to the FFT, which can be easily parallelized, the FST method is able to use the new multi-core hardware efficiently.

Indice

Introduzione

Era il 24 Aprile 1973 quando al Chicago Board Options Exchange, per la prima volta nella storia, fu possibile scambiare opzioni su un mercato regolamentato. A quel tempo erano solamente 16 le azioni sulle quali era possibile negoziare, ma nel corso degli anni, il mercato dei contratti derivati ha largamente superato, sia per volume che per controvalore, il mercato azionario.
Con l'aumentare dell'utilizzo di tali titoli, soprattutto a scopo speculativo, non sono mancati però casi di gravi perdite. Questo ha generato molte perplessità da parte dell'opinione pubblica nei confronti degli strumenti derivati, attribuendo ad essi un effetto destabilizzante nei confronti dei mercati finanziari. La crescente complessità di tali strumenti rende estremamente complessa la funzione di vigilanza degli organismi incaricati, ed anche la corretta valutazione dei derivati risulta una questione sempre più difficile. Per questo motivo la teoria matematica dei derivati finanziari si è sviluppata in parallelo allo sviluppo dei mercati.

Lo scopo di questa tesi è quello di presentare un metodo di pricing per gli strumenti derivati che sia:

- Applicabile ad un'ampia classe di modelli.
- Efficiente, nel senso che deve essere sia preciso che computazionalmente non troppo costoso.
- Versatile, che consenta di prezzare strumenti differenti con lievi modificazioni.
- Parallelizzabile, dato il crescente sviluppo e utilizzo di hardware multi-core.

Per questo motivo abbiamo scelto il metodo FST (Fourier Space Time-stepping) ideato da V. Surkov. Tale metodo era già applicabile ad un'ampia classe di modelli, gli exponential-Lévy, ne abbiamo esteso l'utilizzo ad alcuni modelli a volatilità stocastica, quelli di Heston e Bates. È un metodo preciso (confrontandolo con i metodi maggiormente utilizzati in letteratura) ed applicabile a molte tipologie di contratti e payoff, in generale è utilizzabile per tutti i payoff non path-dependent. Inoltre è facilmente estendibile a strumenti con payoff di tipo Americano o ad opzioni Barriera.
È computazionalmente efficiente e parallelizzabile, grazie all'utilizzo dell'algoritmo FFT.

Presentiamo più in dettaglio il contenuto della tesi:
Nel Capitolo 1 forniamo la definizione di derivato finanziario e presentiamo i modelli maggiormente utilizzati in letteratura per il sottostante. Infine enunciamo alcune definizioni e proprietà relative ai processi di Lévy, fondamentali per la comprensione dei capitoli successivi.
Il Capitolo 2 è dedicato allo sviluppo del metodo FST per opzioni plain-vanilla, con particolare riguardo sia per la derivazione analitica che per l'implementazione numerica. Sono presenti esempi e test sulle proprietà che caratterizzano il metodo. Viene infine presentata la tecnica dell'estrapolazione di Richardson, per aumentare l'ordine di convergenza del metodo.
Nella prima parte del Capitolo 3 presentiamo l'utilizzo del metodo FST per opzioni path-dependent, in particolare per opzioni Americane e Barriera, indicando per entrambe le tipologie i risultati ottenuti e fornendo alcuni esempi. Nella seconda parte vengono trattati i modelli con Regime-Switching, con particolare attenzione per le superfici di volatilità implicita da essi generate. I modelli con Regime-Switching sono sostanzialmente l'unione di più modelli , legati tra loro da una catena di Markov. Questo tipo di modellistica viene solitamente utilizzata in alternativa alla volatilità stocastica, e come vedremo produce delle superfici di volatilità implicita molto simili a quelle di mercato.
Proponiamo inoltre un'estensione del modello di Black e Cox per titoli defaultable, basato sull'utilizzo del Regime-Switching, al fine di introdurre stocasticità nella barriera e nel recovery-rate.

Il Capitolo 4 contiene la parte totalmente inedita della tesi (insieme all'estensione per titoli defaultable del capitolo precedente). Viene trattata l'estensione del metodo FST ai modelli a volatilità stocastica di Heston, Bates ed Heston-Hull-White, quest'ultimo con anche il tasso d'interesse stocastico; sono inoltre presenti alcuni test che ne verificano la correttezza. Successivamente viene derivato il metodo FST per l'equazione di Dupire generalizzata ai modelli con salto (equazione Forward). Tale equazione è scritta in funzione dello strike e non dello spot, permette quindi di trovare il prezzo di molte opzioni, aventi tutte la stessa maturity, utilizzando un solo algoritmo FST. Tale approccio si rivela particolarmente utile per la calibrazione dei modelli, presenteremo a fine capitolo un esercizio di questo tipo, ottenendo dei risultati ottimi dal punto di vista del costo computazionale.
Infine presentiamo nel Capitolo 5 alcune implementazioni parallele su scheda grafica, con l'utilizzo della tecnologia CUDA. Confronteremo tale implementazione con una seriale in C, ottenendo anche in questo caso degli ottimi risultati: con performance del codice parallelo sino ad 11 volte maggiori rispetto al codice seriale.

Capitolo 1

Nozioni preliminari

1.1 I derivati finanziari

Con il termine *strumento derivato*, in ambito finanziario, si definisce un contratto il cui prezzo dipende dal valore di mercato di un altro strumento finanziario, quest'ultimo detto sottostante dello strumento derivato. Tali strumenti possono essere utilizzati principalmente per due finalità: hedging e speculazione. Con la prima si intende coprirsi da alcuni rischi naturalmente presenti nel mondo finanziario, mentre la seconda è sostanzialmente una scommessa sull'andamento del mercato.
Facciamo un esempio di hedging: ipotizziamo che un individuo debba acquistare una certa quantità di petrolio tra un anno, vuole però fissare il prezzo oggi, coprendosi dal rischio di rialzo e rinunciando anche ad eventuali ribassi. Comprerà allora un derivato che gli conceda tale diritto, per esempio un Forward. Il Forward è un titolo derivato che obbliga il detentore ad acquistare, in un istante fissato T, un determinato sottostante ad un prezzo K, tale prezzo viene fissato nel momento della stipula del contratto.
Per quanto riguarda la speculazione invece, ipotizziamo che un investitore sia convinto che il prezzo di una certo titolo salirà nel corso del prossimo anno. Vorrà quindi acquistare uno strumento derivato che gli dia la possibilità, ma non l'obbligo, di comprare tra un anno tale titolo ad un prezzo fissato oggi, più basso di quanto si aspetta che sarà il prezzo futuro. Tale strumento si chiama opzione Call.

Esistono contratti derivati con diversi gradi di sofisticazione, quelli più semplici sono detti plain-vanilla, mentre quelli più complessi sono detti esotici. I contratti sono inoltre divisi per tipologie: opzioni, swap, futures, forward rate agreement e molti altri, vedi [5].
Noi ci occuperemo principalmente di opzioni, cioè di contratti standardizzati che danno al possessore il diritto (non l'obbligo) di comprare/vendere un determinato sottostante ad un prezzo fissato, detto strike price. Tale diritto può essere esercitato entro oppure in un istante prefissato, detto scadenza dell'opzione o maturity. Le opzioni che consentono di vendere sono dette Put, mentre quelle che consentono di comprare sono dette Call.
Sia S_T il valore del sottostante alla maturity T dell'opzione e K lo strike price, un'opzione Call ha un payoff del tipo $(S_T - K)^+$, in quanto il possessore ha il diritto, non l'obbligo, di comprare al prezzo K. Eserciterà quindi il diritto di acquisto solo se gli consentirà di ottenere un guadagno. Per un'opzione di tipo Put vale un ragionamento analogo, il payoff è quindi $(K - S_T)^+$.
Sottolineiamo che l'acquisizione un qualsiasi contratto derivato comporta una spesa proporzionale al diritto acquisito con esso.
Riportiamo un esempio. Se pensassimo che un certo titolo, che oggi vale 100, domani varrà 105, abbiamo a disposizione due strategie per beneficiare di tale rialzo: comprare il titolo o un opzione Call su di esso. Nel primo caso è necessario un esborso di denaro pari a 100, sobbarcandosi il rischio che il titolo al posto che apprezzarsi si deprezzi. Nel secondo caso invece, investendo una cifra di denaro molto minore, possiamo aggiudicarci tale differenza, non perdendo nulla in caso di deprezzamento. I derivati danno quindi la possibilità di ottenere grandi guadagni a fronte di una piccola spesa, tale effetto viene chiamato *leva finanziaria*.
Le opzioni il cui prezzo dipende non solo dal valore del sottostante alla scadenza, ma anche dai valori assunti durante tutta la vita dell'opzione (per esempio dal massimo o dal minimo raggiunto), vengono dette path-dependent. Avremmo modo di trattare alcune opzioni di questo tipo nel Capitolo 3.

1.2 Modelli per il sottostante

Il prezzo di un contratto derivato dipende, come già detto in precedenza, dal valore di un altro strumento; modellizzare l'andamento del sottostante è quindi un passo fondamentale per poter prezzare uno strumento derivato. Noi ci limiteremo ad una breve presentazione dei modelli che sono stati introdotti nel corso degli anni, per una trattazione completa rimandiamo a [5] e [13].
Focalizziamoci su sottostanti di tipo equity, cioè titoli azionari. Questi sono soggetti a fluttuazioni imprevedibili, che spesso presentano un certo trend, per esempio il prezzo può essere mediamente crescente o decrescente. Per questo motivo, un primo modello proposto per catturare tali caratteristiche è stato un moto Browniano con drift:

$$dS_t = \mu dt + \sigma dW_t\,, \tag{1.1}$$

dove con S_t indichiamo il valore di un titolo al tempo t, con μ il drift (o tasso di crescita), con W_t un moto Browniano (o processo di Wiener) e con σ la volatilità del titolo. Per la definizione formale di moto Browniano rimandiamo a [2]. Tale modello possiede lo svantaggio di consentire valori negativi per S_t, ma come sappiamo il prezzo di un titolo non può essere negativo.
Nel 1973 Black e Scholes proposero il modello che sarebbe diventato il riferimento per il mondo finanziario, cioè il modello lognormale:

$$dS_t = \mu S_t dt + \sigma S_t dW_t\,. \tag{1.2}$$

Tale modello impone che il processo stocastico S_t sia positivo. Il modello (1.2) è stato, ed è ancora, largamente utilizzato dagli operatori finanziari. Purtroppo però, a seguito di numerosi studi sui dati di mercato, si è giunti alla conclusione che la distribuzione lognormale non riproduce in maniera realistica la dinamica dei titoli. Le analisi statistiche evidenziano alcune caratteristiche che il modello di Black and Scholes non è in grado di riprodurre:

- I parametri non sono costanti nel tempo.

- Le code della distribuzione (valori di S_t molto piccoli e molto grandi) sono più spesse di quelle della lognormale e sono asimmetriche; in particolare la coda sinistra è più spessa di quella destra.

- Il prezzo non è un processo continuo, ma possono essere presenti dei salti.

Per ovviare a questi problemi, nel corso degli anni, sono state proposte molte alternative. La prima è stata porre i parametri time-dependent, ossia di vedere il tasso di crescita e la volatilità come funzioni del tempo, $\mu = \mu(t)$ e $\sigma = \sigma(t)$. Per ovviare agli ultimi due problemi invece, sono state sviluppate due diverse tipologie di modelli: quelli a volatilità locale e/o stocastica, e quelli con processi di salto. Per una trattazione dettagliata sui primi rimandiamo a [24], per i secondi a [13].
I modelli a volatilità locale prevedono che σ sia funzione sia del tempo che del sottostante, $\sigma = \sigma(t, S_t)$, ipotizzando una forma parametrica oppure ricavandola direttamente dai dati di mercato.
I modelli a volatilità stocastica prevedono invece che σ sia a sua volta un processo stocastico, per esempio:

$$dS_t = \mu S_t dt + \sqrt{V_t} S_t dW_t^1 , \tag{1.3}$$

$$dV_t = (b - aV_t)dt + \nu dW_t^2 , \tag{1.4}$$

dove i moti Browniani W_t^1 e W_t^2 possono essere correlati. Il modello (1.4) è detto modello di Vasiček, ma ne esistono molti altri adatti a descrivere la volatilità.
Nei modelli con salto invece, viene aggiunto al processo lognormale un processo non continuo J_t:

$$dS_t = \mu S_t dt + \sigma S_t dW_t + dJ_t . \tag{1.5}$$

Infine è possibile unire volatilità locale, volatilità stocastica, modelli con salto e parametri time-dependent, ottenendo quindi modelli sempre più sofisticati.

1.3 Processi di Lévy

Durante lo svolgimento di questa tesi ricorreremo frequentemente all'utilizzo di un particolare insieme di processi con salto, detti processi di Lévy. Per questo motivo ne presentiamo la definizione ed alcune proprietà che ci saranno utili nel corso dei capitoli successivi.

Definizione 1.1 (Processo di Lévy)**.** Sia $(X_t)_{t\geq 0}$ un processo stocastico a valori in $\mathbb{R}^d$, definito sullo spazio probabilizzato $(\Omega, \mathcal{F}, \mathbb{P})$, dove con Ω indichiamo l'insieme degli raggiungibili da X_t, con $\mathcal{F}$ la sua filtrazione e con $\mathbb{P}$ una misura di probabilità.
X_t è detto processo di Lévy se soddisfa le seguenti proprietà:

1. X_0=0.
2. È un processo *cadlag* (continuo a destra con limite a sinistra).
3. Ha incrementi indipendenti: per ogni sequenza crescente di tempi $t_0, \ldots, t_n$ le variabili $X_{t_0}, X_{t_1} - X_{t_0}, \cdots, X_{t_n} - X_{t_{n-1}}$ sono indipendenti.
4. Ha incrementi stazionari: $X_{t+h} - X_t$ non dipende da t.
5. Continuità stocastica: $\forall \epsilon > 0, \lim_{h\to 0} \mathbb{P}\left(|X_{t+h} - X_t| \geq \epsilon\right) = 0$.

Proposizione 1.3.1 (Funzione caratteristica per un processo di Lévy)**.** *Sia $(X_t)_{t\geq 0}$ un processo di Lévy a valori in $\mathbb{R}^d$. Allora esiste una funzione $\Psi : \mathbb{R}^d \to \mathbb{R}$ chiamata esponente caratteristico di X_t, tale che:*

$$\mathbb{E}\left[e^{i\omega \cdot X_t}\right] = e^{t\,\Psi(\omega)}, \quad \omega \in \mathbb{R}^d\,. \tag{1.6}$$

Mostreremo nel prossimo capitolo la decomposizione di Lévy-Ito (2.8), da questa deriva il seguente lemma:

Lemma 1.3.2. *Sia $(X_t)_{t\geq 0}$ un processo di Lévy d-dimensionale, allora la sua distribuzione è definita in maniera univoca da una tripla (A, ν, γ), dove $A \in \mathbb{R}^{d\times d}$ indica una matrice definita positiva, ν una misura positiva e $\gamma \in \mathbb{R}^d$ un generico vettore.*

Grazie le informazioni appena fornite, possiamo enunciare un teorema fondamentale per lo svolgimento di questa tesi.

Teorema 1.3.3 (Rappresentazione di Lévy-Khinchin)**.** *Sia $(X_t)_{t\geq 0}$ un processo di Lévy d-dimensionale con tripla caratteristica (A, ν, γ), il suo esponente caratteristico è della forma:*

$$\Psi(\omega) = -\frac{1}{2}\omega \cdot A\omega + i\gamma \cdot \omega + \int_{\mathbb{R}^d} \left(e^{i\omega\cdot x} - 1 - i\omega \cdot x\mathbf{1}_{|x|\leq 1}\right) \nu(dx)\,. \tag{1.7}$$

I processi di Lévy si dividono in due categorie, vedi [13]: ad attività finita e ad attività infinita. I primi sono sostanzialmente dei processi diffusivi, con l'aggiunta di sporadici salti che avvengono in maniera imprevedibile. L'ampiezza di questi salti ha una distribuzione ben determinata, sono quindi formati da un moto Browniano più un Compound Poisson.
I processi ad attività infinita non contengono necessariamente una componente diffusiva, inoltre la misura di Lévy ν è tale da consentire infiniti piccoli salti. La distribuzione dell'ampiezza dei salti non esiste e i salti avvengono a distanza infinitesima l'uno dall'altro, rendendo non necessaria la presenza della componente continua.
Tra i processi ad attività finita rientrano i modelli di Merton e Kou, tra quelli ad attività infinita il Variance-Gamma e il CGMY (ottenuti tramite subordinazione di un moto Browniano).

	Merton	Kou
Parametri	σ volatilità della parte diffusiva, λ intensità dei salti, μ media dei salti, δ deviazione standard del'ampiezza dei salti	σ volatilità della parte diffusiva, λ intensità dei salti, $(\lambda_-, \lambda_+, p)$ parametri per la distribuzione dell'ampiezza dei salti
Densità di Levy	$\nu(x) = \frac{\lambda}{\delta\sqrt{2\pi}} exp\left\{-\frac{(x-\mu)^2}{2\delta^2}\right\}$	$\nu(x) = (1-p)\lambda\lambda_- e^{-\lambda_-\lvert x\rvert}\mathbf{1}_{x<0} + p\lambda\lambda_+ e^{-\lambda_+ x}\mathbf{1}_{x>0}$
Esponente caratteristico	$\Psi(\omega) = i\omega\gamma - \frac{\sigma^2\omega^2}{2} + \lambda\left\{e^{-\delta^2\omega^2/2+i\mu\omega} - 1\right\}$	$\Psi(\omega) = i\omega\gamma - \frac{\sigma^2\omega^2}{2} + \lambda\left\{\frac{p}{\lambda_+ - i\omega} - \frac{1-p}{\lambda_- + i\omega}\right\}$
	Variance Gamma	CGMY
Parametri	σ volatilità del moto Browniano, μ drift del moto Browniano, k varianza del subordinatore	σ volatilità del moto Browniano, μ drift del moto Browniano, k varianza del subordinatore
Densità di Levy	$\nu(x) = \frac{1}{k\lvert x\rvert} e^{Ax - B\lvert x\rvert}$ con $A = \frac{\theta}{\sigma^2}$ e $B = \frac{\sqrt{\theta^2 + 2\sigma^2}/k}{\sigma^2}$	$\nu(x) = \frac{C}{\lvert x\rvert} e^{Ax} K_1(B\lvert x\rvert)$ con $A = \frac{\theta}{\sigma^2}$ $B = \frac{\sqrt{\theta^2+\sigma^2}/k}{\sigma^2}$ e $C = \frac{\sqrt{\theta^2+\sigma^2}/k}{2\pi\sigma\sqrt{k}}$
Esponente caratteristico	$\Psi(\omega) = i\omega\gamma - \frac{1}{k}\log\left(1 + \frac{\omega^2\sigma^2 k}{2} - i\theta k\omega\right)$	$\Psi(\omega) = i\omega\gamma + \frac{1}{k} - \frac{1}{k}\sqrt{1 + \omega^2\sigma^2 k - 2i\theta k\omega}$

1.4 Il problema di pricing

Un titolo derivato consente di ottenere un payoff che dipende dal valore del sottostante alla scadenza T, il prezzo deve quindi dipendere dall'aspettativa sul guadagno futuro. A questo si aggiunge che il valore del denaro cambia nel tempo; per questo motivo serve scontare il valore atteso, prendendo l'equivalente oggi del guadagno atteso alla maturity dell'opzione. Questo sconto viene effettuato utilizzando un tasso d'interesse che definiremo privo di rischio (ipotizzando che esista). Sia $V(t, S_t)$ il valore di un contratto derivato in t con scadenza T, allora possiamo scrivere:

$$V(t, S_t) = e^{-r(T-t)}\mathbb{E}\left[\varphi(S_T)\right] , \tag{1.8}$$

dove $\varphi(S_T)$ indica il payoff del contartto derivato.
Si pone ora un altro problema, cioè sotto quale probabilità calcolare il valore atteso. Prendiamo un sottostante che evolve secondo il modello di Black&Scholes (1.2), diversi individui potrebbero aver una percezione diversa del drift μ del processo, e quindi del valore atteso del contartto derivato a scadenza. Il prezzo di un derivato deve essere unico, per questo motivo si sceglie, tra tutte le misure di probabilità, quella che viene definita risk-neutral, cioè quella che un'investitore neutrale al rischio assegnerebbe al processo stocastico S_t. L'imposizione di tale misura di probabilità, nei modelli diffusivi, coincide con la sostituzione di r, il tasso risk-free, al posto del drift μ. Per i processi con salto tale sostituzione non è sufficiente, enunceremo nel prossimo capitolo la condizione risk-neutral per processi di Lévy.
Nel corso di questa tesi cercheremo di risolvere il problema di pricing (1.8) per vari modelli e contratti, utilizzando un metodo introdotto da Vladimir Surkov nel 2007, chiamato Fourier-Time-Stepping (FST).
In letteratura si trovano moltissimi metodi per risolvere tale problema in maniera efficiente, tra i principali troviamo metodi Montecarlo [32], i metodi basati sulla trasformata di Fourier [16] [48] e metodi variazionali, che sfruttano la possibilità di riscrivere il valore atteso in (1.8) sotto forma di PDE (o PIDE nel caso di processi con salto), utilizzando

tecniche come gli elementi finiti, vedi [52].
L'approccio del metodo FST, come vedremo, sfrutta sia la scrittura del problema sotto forma di PDE/PIDE, sia l'utilizzo della trasformata di Fourier.

Capitolo 2

Metodo FST

2.1 Introduzione

In questo capitolo presenteremo la derivazione del metodo FST (Fourier Space Time-stepping): richiameremo la definizione ed alcune proprietà della trasformata di Fourier [Sezione 2.2], presenteremo il modello scelto per il sottostante [Sezione 2.3], deriveremo quindi il metodo analiticamente [Sezione 2.4] ed infine tratteremo la sua implementazione numerica [Sezione 2.5], con una breve trattazione sulle proprietà di tale metodo [Sezione 2.6].

Il metodo FST è basato sulla trasformazione di una PIDE (Partial Integro-Differential Equation) nello spazio di Fourier (rispetto al sottostante), come mostreremo successivamente. La PIDE relativa ad un problema di pricing si trasforma in una ODE (Ordinary Differential Equation) nello spazio di Fourier, nella sola variabile temporale; quest'ultima è facilmente risolvibile analiticamente. Dopo aver ricavato la soluzione dell'ODE nello spazio delle frequenze, è sufficiente antitrasformarla per ottenere la soluzione nello spazio di partenza. L'idea è molto semplice, illustriamo in maniera schematica il procedimento:

$$
\begin{array}{ccc}
 & \overset{\mathcal{F}}{} & \\
f(S_{t_2}) & \longrightarrow & \widehat{f}(S_{t_2}) \\
 & & \downarrow \quad \text{Time-step} \\
f(S_{t_1}) & \longleftarrow & \widehat{f}(S_{t_1}) \\
 & \underset{\mathcal{F}^{-1}}{} &
\end{array}
$$

dove con $\mathcal{F}$ e $\mathcal{F}^{-1}$ indichiamo rispettivamente la trasformata di di Fourier e la sua inversa, con f e $\widehat{f}$ una generica funzione del sottostante S in un istante t_i e la sua trasformata di Fourier. Gli istanti t_1 e t_2 sono tali che $t_1 < t_2$.

Prima di derivare analiticamente il metodo, illustriamo brevemente i principali svantaggi dei metodi più utilizzati in letteratura. Successivamente spiegheremo perchè il metodo FST può essere considerato un'ottima alternativa.

Come accennato nel Capitolo 1, i metodi maggiormente utilizzati per il pricing di opzioni sono tre: metodi variazionali per la risoluzione di PIDE (differenze o elementi finiti), metodi semi-espliciti basati sulla trasformata di Fourier (Carr-Madan e Lewis) e metodi Monte Carlo (MC). Per una trattazione completa rimandiamo a [49].

La risoluzione con metodi variazionali ha principalmente due tipi di problemi: uno dovuto alla difficoltà di trattazione del termine integrale, l'altro dovuto alla discretizzazione temporale. Per quanto riguarda il primo, il termine integrale non può essere trattato numericamente in maniera implicita, a causa della sua natura non locale. Tale trattazione, infatti, richiede l'inversione di una matrice piena e spesso mal condizionata, ma questo risulta numericamente difficile e poco efficiente. Si utilizza quindi la trattazione esplicita che introduce però possibili instabilità nell'algoritmo numerico; è quindi richiesta grande accortezza sulla definizione della griglia spaziale per ovviare al problema della stabilità. Per processi di Lévy ad attività infinita è necessario troncare i piccoli salti in un intorno dello zero, questo perchè la misura di Lévy diverge in questo punto e l'integrale numerico non risulta quindi ben definito. Infine, sempre relativamente al termine integrale, esiste

un'ulteriore difficoltà nell'implementazione numerica: o si fa coincidere la griglia d'integrazione con quella utilizzata per il calcolo della soluzione della PIDE, o si definiscono due griglie distinte e si interpolano i valori per passare da una all'altra. Nel primo caso la griglia d'integrazione non può essere scelta fitta a piacere, nel secondo l'esecuzione di continue interpolazioni rallenta in maniera sostanziale l'esecuzione del codice.
Il secondo fattore problematico dei metodi variazionali è la discretizzazione temporale. Questi metodi hanno bisogno di griglie temporali sufficientemente fitte per ottenere una soluzione precisa e per garantire la stabilità del metodo; tale discretizzazione deve essere effettuata anche nel caso in cui la natura del derivato tratto richieda in teoria uno solo (come le opzioni plain-vanilla) o pochi step temporali (come le opzioni a monitoraggio discreto).
Il metodo FST consente di ovviare ad entrambi i problemi. Infatti da un lato tratta la parte diffusiva e integrale della PIDE nello stesso modo, dall'altro permette la risoluzione analitica nella variabile temporale. In questo modo, consente di utilizzare un solo step per prezzare opzioni plain-vanilla e un numero di step pari al numero di date di monitoraggio per opzioni a monitoraggio discreto.

Per quanto riguarda i principali metodi basati sulla trasformata di Fourier, è richiesta la formula analitica della trasformata del payoff. Tali metodi sono quindi utilizzati principalmente per opzioni Europee o Barriera. Le opzioni con payoff più complessi sono invece difficilmente trattabili, in quanto l'espressione analitica del valore atteso condizionato nello spazio di Fourier è nota solo in casi semplici.
Il metodo FST permette invece di utilizzare, in linea teorica, anche payoff più complessi (per esempio quello delle opzioni Americane), in quanto non richiede in alcun modo la conoscenza analitica della soluzione nello spazio delle frequenze.

I metodi Monte Carlo (MC) si prestano in linea teorica al pricing di qualunque tipo di opzione. Nella pratica però non sono esenti da alcune criticità. Innanzitutto i metodi MC sono molto costosi dal punto di vista computazionale, in quanto è necessario svolgere un gran numero di simulazioni per avere risultati abbastanza accurati. Inoltre le simulazioni

devono essere svolte con una discretizzazione dell'intervallo temporale $[0, T]$ abbastanza fitta, al fine di ridurre al minimo l'errore dato dalla generazione numerica del processo stocastico. Questo perchè, a parte in rari casi, la densità di transizione del processo stocastico non è nota analiticamente, viene quindi approssimata con un errore $o(\Delta t)$.
Per le opzioni path-dependent è spesso necessario tenere in memoria sia le singole simulazioni, sia lo sviluppo temporale delle stesse, così come avviene nei metodi regressivi applicati alle opzioni Americane. Di conseguenza, ciò rende necessaria la memorizzazione di una grande quantità di dati.
Un altro problema consiste nella simulazione delle variabili aleatorie. Infatti gli algoritmi per campionare da distribuzioni più complesse della normale e della uniforme non sono sempre disponibili già implementati. Per ottenere un'implementazione accurata sono a volte necessari algoritmi molto sofisticati. Infine è noto che i metodi MC non sono adatti per calcolare in modo accurato le greche [32].
Contrariamente ai metodi MC, il metodo FST richiede di tenere in memoria un singolo step temporale anche per opzioni path-dependent come quelle americane. Le attuali implementazioni della FFT (Fast Fourier Transform) sono molto accurate e computazionalmente molto efficienti, inoltre il calcolo delle greche è svolto in maniera semplice ed accurata tramite il metodo FST. Non tratteremo all'interno della tesi l'estensione del metodo FST per il calcolo delle greche, rimandando per ulteriori dettagli a [50].
Concludendo, da un punto di vista computazionale, grazie alla natura degli algoritmi FFT per il calcolo della trasformata di Fourier discreta (DFT), il codice numerico è facilmente parallelizzabile: questo rende utilizzabili pienamente i nuovi hardware multi-core, come ad esempio le moderne schede video. Tratteremo successivamente lo sviluppo di alcuni codici CUDA per accelerare il calcolo di diversi tipi di opzioni (Capitolo 5).

2.2 Trasformata di Fourier

Richiamiamo brevemente la definizione di trasformata di Fourier ed alcune proprietà che ci saranno utili nella derivazione del metodo. Per una trattazione completa rimandiamo a [25].

Teorema 2.2.1 (Trasformata di Fourier)**.** *Sia $f(x)$ una funzione integrabile nel senso di Lebesgue, con $x \in \mathbb{R}$ (diremo $f(x) \in \mathcal{L}^1(\mathbb{R})$), la sua trasformata di Fourier è definita come segue:*

$$\mathcal{F}[f](\omega) = \widehat{f}(\omega) = \int_{\mathbb{R}} e^{-i\omega x} f(x)\, dx\,. \tag{2.1}$$

Inoltre valgono le seguenti proprietà:

$$\begin{aligned} &\text{se } g(x) = f(x-y) \quad \text{allora} \quad \widehat{g}(\omega) = e^{-i\omega y}\, \widehat{f}(\omega)\,, \\ &\text{se } g(x) = \partial_x^n f(x) \quad \text{allora} \quad \widehat{g}(\omega) = (i\,\omega)^n\, \widehat{f}(\omega)\,, \end{aligned} \tag{2.2}$$

dove con il simbolo ∂_x^n denotiamo la derivata n-esima rispetto alla variabile x.
Un'altra proprietà, che ci tornerà utile nel Capitolo 4, è la seguente:

$$\mathcal{F}\left[\int f(y-x)g(x)\, dx\right] = \mathcal{F}\,[f]\,\mathcal{F}\,[g]\ , \tag{2.3}$$

detto anche Teorema di Convoluzione.

Teorema 2.2.2 (Anti-Trasformata di Fourier)**.** *Sia $\widehat{f}(\omega)$ la trasformata di Fourier di $f(x)$; se sia $f(x)$ che la sua trasformata $\widehat{f}(\omega)$ sono integrabili (nel senso di Lebesgue) allora é possibile invertire la formula (2.1) :*

$$\mathcal{F}^{-1}\left[\widehat{f}\right](x) = f(x) = \frac{1}{2\pi}\int_{\mathbb{R}} e^{i\omega x}\, \widehat{f}(\omega)\, d\omega\,. \tag{2.4}$$

Riportiamo inoltre la definizione di trasformata e antitrasformata discreta di Fourier (DFT), per un vettore **x** di lunghezza N:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-ik2\pi n/N} \qquad k = 0, \ldots, N-1\,, \tag{2.5}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{ik2\pi n/N} \qquad n = 0, \ldots, N-1\,, \tag{2.6}$$

dove x_n e X_k sono le componenti rispettivamente del vettore di partenza $\mathbf{x}$ e del vettore trasformato $\mathbf{X}$.

Prendiamo, per esempio, un opzione europea di tipo Call, scritta su un solo sottostante S_t. Il prezzo di questa opzione è una funzione crescente del valore di S_t. Come notiamo in Figura 2.1 questo tipo di funzione non appartiene ad $\mathcal{L}^1(\mathbb{R})$. Nell'utilizzo del metodo FST ci servirà poter eseguire la trasformata di Fourier di questa funzione e delle sue derivate, operazione che in linea teorica non sarebbe possibile. Per ovviare al problema, si trovano in letteratura alcune trasformazioni della funzione tali da renderla integrabile (un esempio è l'approccio adottato nel metodo di Carr-Madan [13]). Quello che adotteremo noi è un approccio differente: sfrutteremo infatti le proprietà della trasformata di Fourier discreta. La DFT permette di trattare i segnali periodici come una somma finita di componenti armoniche [42]. Un segnale periodico di periodo T viene campionato in N+1 punti, $p_0 = 0, p_1 = \Delta t, \ldots, p_N = T$, con $\Delta t = \frac{T}{N}$; la DFT riceve in ingresso solo i primi N punti, in quanto p_N è superfluo essendo il segnale periodico ed essendo quindi $p_0 = p_N$. Quello che faremo nella costruzione del metodo è l'operazione seguente: passeremo il campionamento di una funzione non periodica su un dominio limitato alla DFT; l'algoritmo tratterà l'input come se fosse invece periodico. Questo, come vedremo più avanti, crea delle distorsioni ai bordi del dominio, vedi Figura 2.2, distorsioni che riusciremo ad escludere con qualche accortezza, vedi Figura 2.1. Tale approccio ci consente di utilizzare la trasformata di Fourier continua per derivare analiticamente il metodo, senza preoccuparci se la funzione che andremo a trasformare appartiene o no ad $\mathcal{L}^1(\mathbb{R}^n)$. L'implementazione della DFT tramite l'algoritmo FFT (Fast Fourier Transform) ci consente di ottenere ottime performance rispetto ad un metodo di

quadratura generico per il calcolo di integrali complessi (come la trasformata di Fourier continua). Mostreremo nelle Sezioni 2.4 e 2.5 come le due versioni della trasformata di Fourier (continua e discreta) impatteranno nella derivazione e nell'utilizzo dell'algoritmo FST.

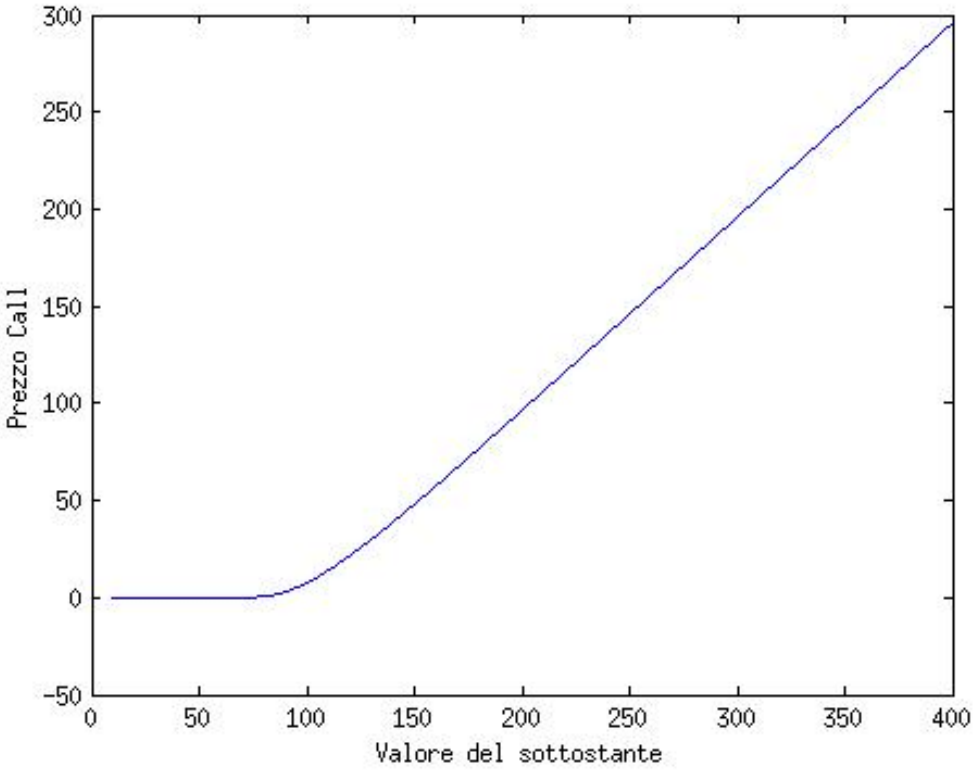

Figura 2.1: Prezzo di una opzione Call ricavato con il metodo FST nell'area di nostro interesse.

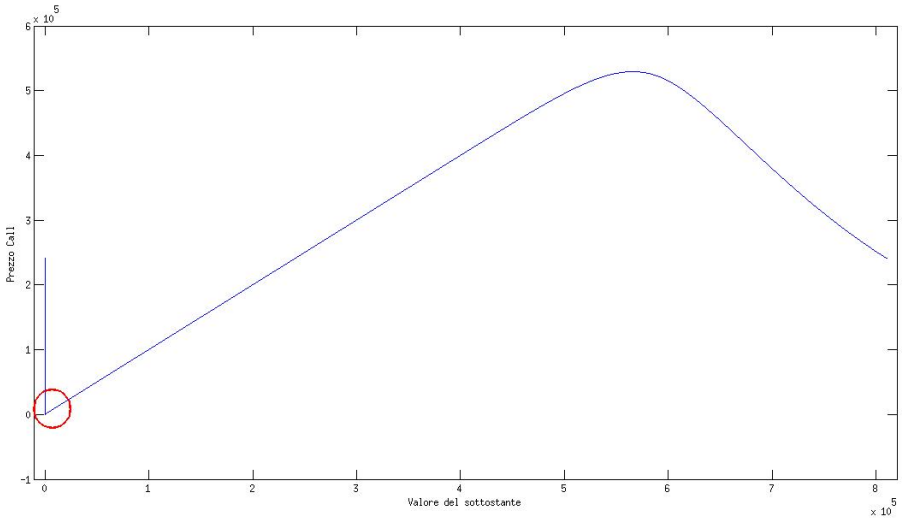

Figura 2.2: Prezzo di una opzione Call ricavato con il metodo FST su tutto il domino $\mathbf{\Omega}$, l'area di nostro interesse è una piccola area sulla sinistra escludendo un intorno dello zero, all'interno dell'area evidenziata di rosso. In questa zona i prezzi sono esenti da distorsioni e il pricing risulta quindi molto accurato. Ci focalizziamo sostanzialmente su un'area compresa tra 0,1 e 10 volte lo spot.

2.3 Modellizzazione del prezzo di un derivato

Sia $X(t)$ un processo di Lévy con tripletta caratteristica (γ, σ, ν), dove γ è il drift, σ è la volatilità della parte diffusiva e ν è la densità di Lévy. Utilizziamo come modello per il sottostante un processo di tipo exponential-Lévy, per cui indichiamo il prezzo spot come:

$$S(t) = S(0)e^{X(t)}\,. \tag{2.7}$$

$X(t)$, grazie alla decomposizione di Lévy-Ito, è definito come segue:

$$
\begin{aligned}
X(t) &= \gamma t + \sigma W(t) + X^l(t) + \lim_{\epsilon \to 0} \tilde{X}^\epsilon(t)\,, \\
X^l(t) &= \int_0^t \int_{|y|\geq 1} y\, J_X(dy \times ds)\,, \\
\tilde{X}^\epsilon(t) &= \int_0^t \int_{\epsilon \leq |y| < 1} y\, [J_X(dy \times ds) - \nu(dy \times ds)]\,,
\end{aligned}
\tag{2.8}
$$

dove $W(t)$ è un moto browniano, $J_X(dy \times ds)$ è la misura di Poisson del processo e $\nu(dy \times ds)$ il suo compensatore.
Il drift γ deve essere scelto in modo da soddisfare la condizione risk-neutral, necessaria nella trattazione del pricing di opzioni. Tale condizione può essere scritta in vari modi, tutti equivalenti, riportiamo quella che nel nostro caso si mostrerà più utile:

$$
\gamma : \mathbb{E}[e^{X_j(1)} = e^{r-q}] \Leftrightarrow \Psi(-i) = r - q\,, \tag{2.9}
$$

dove con Ψ indichiamo l'esponente caratteristico del processo di Lévy. I parametri r e q sono rispettivamente il tasso d'interesse risk-neutral e il tasso di dividendo continuo associato al sottostante.
Sia $V = V(t, S_t)$ il prezzo in t di una generica opzione con sottostante S_t e maturity T. È noto che il prezzo scontato $e^{r(T-t)}V(t, S_t)$ deve essere una martingala per la condizione di non arbitraggio. Utilizzando la martingale-drift decomposition per la scrittura del processo $e^{r(T-t)}V(t, S_t)$ come in [13], e ponendo il drift uguale a zero, otteniamo la PIDE associata al problema di pricing.
Riportiamo l'equazione monodimensionale per semplicità, l'equivalente multidimensionale si ricava in maniera analoga:

$$
\frac{\partial V}{\partial t} + (r-q)S\frac{\partial V}{\partial S} + \frac{\sigma^2}{2}S^2\frac{\partial^2 V}{\partial S^2} - rV + \int_{\mathbb{R}\setminus\{0\}} \left(V(S\,e^y) - V(S) - (e^y - 1)S\frac{\partial V}{\partial S} \right) dy = 0\,, \tag{2.10}
$$

che può essere scritta equivalentemente come:

$$\frac{\partial V}{\partial t} + \gamma S \frac{\partial V}{\partial S} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} - rV + \int_{\mathbb{R}\setminus\{0\}} \left(V(S\,e^y) - V(S) - \mathbf{1}_{|y|<1} y S \frac{\partial V}{\partial S} \right) dy = 0\,. \quad (2.11)$$

Abbiamo omesso la dipendenza di V da t ed S, tranne dove necessario (all'interno dell'integrale).
Il passaggio dalla (2.10) alla (2.11) si ottiene utilizzando la formula di Lévy-Khintchine (1.7) e applicando la condizione (2.9). Infatti, unendo entrambe le cose, deve risultare che:

$$r - q = \gamma + \frac{\sigma^2}{2} + \int_{\mathbb{R}} e^{-i\omega y} - 1 - i\omega y \mathbf{1}_{|y|<1}\, \nu(dx)\,, \quad (2.12)$$

cioè che:

$$\gamma = r - q - \frac{\sigma^2}{2} - \int_{\mathbb{R}} e^{-i\omega y} - 1 - i\omega y \mathbf{1}_{|y|<1}\, \nu(dx)\,. \quad (2.13)$$

L'equazione (2.11) può essere riscritta in maniera leggermente differente applicando il cambio di variabile $x = \ln\left(\frac{S}{S_0}\right)$:

$$\frac{\partial V}{\partial t} + \gamma \frac{\partial V}{\partial x} + \frac{\sigma^2}{2} \frac{\partial^2 V}{\partial x^2} - rV + \int_{\mathbb{R}\setminus\{0\}} \left(V(x+y) - V(x) - \mathbf{1}_{|y|<1} y \frac{\partial V}{\partial x} \right) dy = 0\,. \quad (2.14)$$

dove $V(t, S)$ è stata sostituita con $V(t, x)$. Partendo da quest'ultima equazione, detta di Kolmogorov all'indietro (poichè necessita di una condizione finale per essere risolta), deriviamo in maniera analitica il metodo FST.

2.4 Derivazione del metodo FST

Riscriviamo la PIDE associata al problema di pricing (2.14) con una notazione più compatta e aggiungiamo la condizione finale:

$$\begin{cases} (\partial_t + \mathcal{L} - r)V(t,x) = 0\,, \\ V(T,x) = \varphi(x)\,, \end{cases} \tag{2.15}$$

dove l'operatore $\mathcal{L}$, per una funzione differenziabile due volte $f(x)$, è definito come segue:

$$\mathcal{L}f(x) = \gamma\frac{\partial f}{\partial x} + \frac{\sigma^2}{2}\frac{\partial^2 f}{\partial x^2} + \int_{\mathbb{R}\setminus\{0\}} \left(f(x+y) - f(x) - \mathbf{1}_{|y|<1} y \frac{\partial f}{\partial x} \right) \nu(dy)\,.$$

L'operatore $\mathcal{L}$ è detto generatore infinitesimale del processo di Lévy.
Vogliamo trovare la soluzione del sistema (2.15) per ottenere il prezzo dell'opzione. Applichiamo all'operatore $\mathcal{L}$ la trasformata di Fourier rispetto alla variabile x; grazie alle proprietà (2.2) osserviamo che:

$$\begin{aligned} \mathcal{F}[\mathcal{L}V](T,\omega) &= \left\{ i\omega\gamma - \omega^2\frac{\sigma^2}{2} + \int_{\mathbb{R}\setminus\{0\}} \left(e^{i\omega y} - 1 - i\omega y \mathbf{1}_{|y|<1} \right) \nu(dy) \right\} \mathcal{F}[V](t,\omega) \\ &= \Psi(\omega)\,\mathcal{F}[V](t,\omega)\,, \end{aligned} \tag{2.16}$$

dove la formula (2.16) è stata ottenuta notando che il termine tra parentesi graffe nella riga sopra, è esattamente l'esponente caratteristico dato dalla formula di Lévy-Khintchine (1.7).
Applicando la trasformati di Fourier ad entrambi i membri delle equazioni (2.15)

otteniamo:

$$\begin{cases} \partial_t \mathcal{F}[V](t,\omega) + (\Psi(\omega) - r)\mathcal{F}[V](t,\omega) = 0\,, \\ \mathcal{F}[V](T,\omega) = \mathcal{F}[\varphi](\omega)\,. \end{cases} \tag{2.17}$$

Il sistema di equazioni (2.17) è una ODE nella variabile tempo, con condizione finale. Il sistema è risolvibile analiticamente. Conoscendo il valore $\mathcal{F}[V](t_2,\omega)$ la soluzione in $t_1 < t_2$ risulta infatti essere:

$$\mathcal{F}[V](t_1,\omega) = \mathcal{F}[V](t_2,\omega)\, e^{(\Psi(\omega)-r)\,(t_2-t_1)}\,. \tag{2.18}$$

Ricapitoliamo le operazioni svolte sino ad ora: partendo dalla condizione finale $V(T,x)$ abbiamo trasformato il sistema (2.15) nello spazio delle frequenze, qui abbiamo risolto analiticamente l'ODE. La soluzione trovata non è altro che il prezzo della nostra opzione nello spazio di Fourier. Dobbiamo ora antitrasformare il risultato ottenuto per ricavare la soluzione nello spazio di partenza. L'applicazione di quest'ultima operazione ci porta alla seguente formula:

$$V(t,x) = \mathcal{F}^{-1}\left[\mathcal{F}[V](T,\omega)\, e^{(\Psi(\omega)-r)\,(T-t)}\right](x)\,. \tag{2.19}$$

Abbiamo così ottenuto il prezzo di una opzione plain-vanilla in un solo step, passando direttamente da T a t senza effettuare alcuna discretizzazione temporale. Le variazioni da apportare al metodo per prezzare opzioni path-dependent verranno trattate nel capitolo successivo.

2.5 Implementazione numerica

Passiamo ora all'implementazione numerica della formula (2.19). La trasformata di Fourier continua, come si nota dalla formula (2.1), è un integrale su un intervallo

illimitato. Dobbiamo quindi troncare il dominio e scegliere una griglia su cui risolvere numericamente l'integrale. Il dominio viene quindi definito come $\Omega = [x_{min}, x_{max}]$, e partizionato come segue: $x_n = x_{min} + i\,\Delta x$, con $i = 0, 1, ..., N-1$ e $\Delta x = \frac{x_{max}-x_{min}}{N-1}$. La griglia così definita è equispaziata rispetto alla variabile x; rispetto alla variabile S invece, è molto fitta nell'intorno di $S = S_0$ e sempre meno al crescere di S. Si pone quindi il problema di come scegliere correttamente i valori di x_{min} e x_{max}. La natura della FFT prevede di ricevere in ingresso un segnale periodico, il nostro dato però non lo è; per questo motivo il dominio deve essere abbastanza grande da poter trascurare gli effetti di bordo, negli intorni di x_{min} e x_{max}.

Inoltre l'ampiezza del dominio deve essere tale da riuscire ad incorporare la natura del sottostante e dell'opzione che andiamo a considerare (per esempio volatilità totale e maturity dell'opzione); non dobbiamo dimenticarci che la nostra variabile $x(t)$ è un processo stocastico con una propria distribuzione di probabilità. Se per esempio il sottostante ha una volatilità molto elevata, utilizzare un dominio troppo piccolo può creare distorsioni nel prezzo, anche nella zona d'interesse (quella distante da x_{min} e x_{max}), vedi Figura (2.3); questo perchè escluderemmo dalla nostra analisi alcuni valori raggiungibili dal nostro processo X_t con probabilità non trascurabile. Un altro fattore da tenere in conto è la maturity dell'opzione. Al crescere di T cresce la probabilità che S_t raggiunga valori più distanti da S_0, è quindi necessario un intervallo di discretizzazione più ampio al crescere della maturity. La scelta di un intervallo troppo ampio invece, a parità di punti di discretizzazione, da luogo ad una maggiore inaccuratezza dell'algoritmo FFT, x_{min} e x_{max} vanno quindi scelti con cura. Dato l'interesse per il pricing nell'intorno di $x = 0$ riteniamo abbastanza naturale scegliere $x_{min} = -x_{max}$. Per quanto riguarda il numero di punti di discretizzazione, la scelta di una potenza di due, o un numero dato dal prodotto di potenze di piccoli numeri primi ($2^a \cdot 3^b \cdot 5^c \cdot 7^d$), consente di ottimizzare gli algoritmi FFT, che raggiungono una complessità di $O(N\,logN)$ invece di $O(N^2)$, complessità di una FFT con un numero generico di punti[1]. Nei nostri test useremo solamente potenze di due per sfruttare questa diminuzione di complessità computazionale.

[1] Quando diciamo che un algoritmo ha una complessità pari ad $O(n)$, intendiamo dire che sono necessarie Cn operazioni per eseguire l'algoritmo, con C costante che varia a seconda del caso.

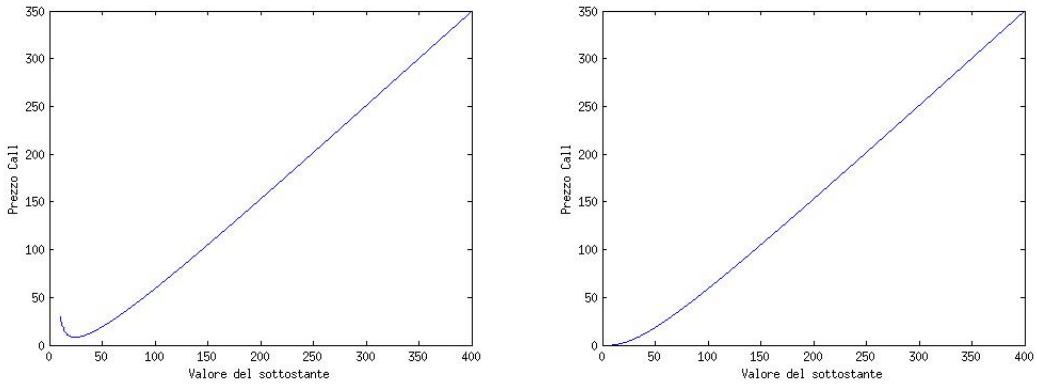

Figura 2.3: A sinistra: prezzo di una opzione Call con maturity di 15 anni e $\mathbf{\Omega} = [-6, 6]$, notiamo che è presente una distorsione abbastanza visibile, dato che ci aspettiamo una soluzione monotona crescente. A destra: prezzo di una opzione Call con maturity di 15 anni e $\mathbf{\Omega} = [-16, 16]$. Notiamo che aumentando l'intervallo di discretizzazione abbiamo eliminato la distorsione nel prezzo.

Non avendo una regola teorica per scegliere gli estremi del dominio, abbiamo indagato numericamente su come sceglierli in maniera corretta. È stata eseguita la seguente procedura: abbiamo scelto un numero di punti di discretizzazione abbastanza elevato, al fine di mantenere una buona accuratezza anche con un intervallo ampio. Partendo da un dominio abbastanza piccolo, abbiamo aumentato l'ampiezza dell'intervallo fino a quando il prezzo dell'opzione mostrava cambiamenti abbastanza piccoli da poter essere trascurati. In questo modo siamo giunti alla conclusione che, per opzioni a breve scadenza e sottostante poco volatile, $x_{max} \in [5, 8]$ è sufficiente, per scadenze più lunghe e sottostanti più volatili $x_{max} \in [8, 12]$ produce dei risultati più accurati.
Dobbiamo ora definire la griglia nello spazio di Fourier. Notiamo innanzitutto che, essendo i dati di partenza reali, è soddisfatta la seguente uguaglianza: $\mathcal{F}[V](t, -\omega) = \overline{\mathcal{F}[V](t, \omega)}$[2]. È quindi necessario calcolare la FFT solo per gli ω_n positivi, mentre per quelli negativi è possibile usare la relazione precedente, in questo modo si dimezzano le operazioni da svolgere. Gli algoritmi FFT per dati reali sono ottimizzati in modo da ridurre quindi il numero di calcoli da eseguire, rendendo il codice nettamente più veloce di una generica FFT con dati complessi. A causa di questa simmetria il dominio nello

[2]Con $\overline{g}$ indichiamo il coniugato di g

spazio di Fourier viene definito come: $\widehat{\Omega} = [0, \omega_{max}]$, dominio partizionato in $\frac{N}{2} + 1$ punti $\omega_n = i\,\Delta\omega$ con $i = 0, 1, \dots, N/2$. La scelta di $\Delta\omega$ è tale da rendere ω_{max} pari alla frequenza critica di Nyquist, cioè $\omega_{max} = \frac{1}{2\Delta x}$. Come per lo spazio di partenza, ω_{max} deve essere abbastanza grande da riuscire a tenere conto delle caratteristiche del processo, più precisamente non deve perdere le alte frequenze nella funzione caratteristica. Allo stesso tempo $\Delta\omega$ deve rimanere abbastanza piccolo da non creare inaccuratezze; per fare ciò è ancora una volta necessario scegliere correttamente il dominio Ω nello spazio di partenza, essendo gli estremi dei domini legati dalla relazione: $\omega_{max} \cdot x_{max} = (N-1)/4$. Per maggiori dettagli rimandiamo al teorema di campionamento di Nyquist-Shannon [42].

Rimane infine da definire la griglia temporale, operazione non necessaria per opzioni plain-vanilla, ma che ci sarà utile in seguito, quando applicheremo il metodo FST ad opzioni path-dependent. Dividiamo quindi il dominio $t \in [0, T]$ in $M + 1$ punti $t_0, \dots, t_M$, con $\Delta t_m = t_m - t_{m-1}$.

Vediamo ora come applicare la discretizzazione all'equazione continua (2.19). Sia $v_{mn} = v(t_m, x_n)$ e $\hat{v}_{m\,n} = \hat{v}(t_m, \omega_n)$ il corrispettivo punto trasformato nello spazio di Fourier. Utilizzando la discretizzazione appena presentata per risolvere la prima trasformata di Fourier otteniamo:

$$\hat{v}_{mn} \approx \sum_{k=0}^{N-1} v(t_m, x_k) e^{i\omega_n x_k}\,\Delta x = \alpha_n \sum_{k=0}^{N-1} v_{mk} e^{\frac{ink}{N}} = \alpha_n \mathrm{FFT}\,[v_m]_n\ , \tag{2.20}$$

dove $\alpha_n = e^{i\omega_n x_{min}\Delta x}$ e $\mathrm{FFT}\,[v_m]_n$ indica la componente n-esima della trasformata di Fourier discreta del vettore v_m. Allo stesso modo, la discretizzazione della trasformata inversa si ottiene come:

$$v_{mn} = \alpha_n^{-1} \mathrm{IFFT}\,[\hat{v}_m]_n\ . \tag{2.21}$$

Applichiamo ora la (2.21) e la (2.20) alla soluzione continua (2.19), dato il vettore v_m otteniamo:

$$
\begin{aligned}
v_{m-1} &= \text{IFFT}[\,\alpha^{-1}\hat{v}_{m-1}\,], \\
&= \text{IFFT}[\,\alpha^{-1}\hat{v}_m \cdot e^{\Psi(\cdot)\Delta t_m}\,], \\
&= \text{IFFT}[\,\alpha^{-1}\alpha\text{FFT}[v_m] \cdot e^{\Psi(\cdot)\Delta t_m}\,], \\
&= \text{IFFT}[\,\text{FFT}[v_m] \cdot e^{\Psi(\cdot)\Delta t_m}\,], \qquad (2.22)
\end{aligned}
$$

dove $\alpha = (\alpha_0, \ldots, \alpha_{N-1})$.
Ponendo $M = 1$ abbiamo $t_1 = T$ e $t_0 = 0$: otteniamo quindi l'algoritmo FST single-step per opzioni eEuropee tramite la formula (2.22).

In Tabella 2.1 presentiamo alcuni risultati numerici ottenuti per il pricing di opzioni Europee con metodo FST e con metodi variazionali, indicando il numero di punti temporali (M) e spaziali (N) utilizzati.

Metodo	N	M	Prezzo	Tempo esecuzione (sec)
FDM	500	50	6.7379	6.5752
FEM	500	50	6.7381	6.6823
FDM	1000	100	6.7404	28.5304
FEM	1000	100	6.7404	28.4635
FDM	1500	150	6.7414	70.1246
FDM	2500	250	6.7421	227.8418
FST	2^{12}	1	6.7431	0.0022
FST	2^{16}	1	6.7432	0.0380
FST	2^{20}	1	6.7432	0.3609

Tabella 2.1: Risultati ottenuti per una Call europea con modello di Merton jump-diffusion. FDM=Finite Differences Method, FDM=Finite Elements Method, FST=Fourier Time Stepping Method. Oltre i 1000 punti spaziali e 100 temporali riportiamo solo i risultati ottenuti con FDM, in quanto i valori e i tempi ottenuti con FEM sono gli stessi.

Notiamo come il metodo FST sia altamente più performante (confrontando i tempi d'esecuzione), rispetto alla controparte a differenze ed elementi finiti. Questo risultato è

in gran parte dovuto alla possibilità di utilizzare un solo step temporale per le opzioni plain-vanilla. I risultati riportati fino a questo punto sono stati calcolati utilizzando MATLAB; un aumento di prestazioni può essere ottenuto tramite l'implementazione C/C++ del codice. Uno speed-up ancora maggiore si ottiene utilizzando implementazioni parallele della FFT. Per quanto riguarda le piattaforme multi-core, considereremo nel Capitolo 5 l'implementazione del metodo FST su GPU, tramite l'utilizzo della tecnologia CUDA.

2.6 Proprietà del metodo numerico

2.6.1 Stabilità

La trasformata e l'anti-trasformata di Fourier vengono eseguite integrando sull'asse reale. Alcuni payoff potrebbero però avere delle singolarità su quest'asse; in tal caso un semplice shifting $\omega \rightarrow \omega + i\epsilon$ è sufficiente ad ovviare il problema. È quindi possibile eseguire l'antitrasformata su un asse parallelo a quello reale, ma non contenente singolarità. L'algoritmo con uno shifting pari ad ϵ viene quindi implementato in questo modo:

$$v_{m-1} = FFT^{-1}[FFT[v_m e^{\mathbf{x}\epsilon}] \cdot e^{\Psi(\omega+i\epsilon)\Delta t_m}]\,. \tag{2.23}$$

Per esempio un'opzione di tipo Put ha un payoff che tende ad un valore costante quando $x \rightarrow -\infty$: questo introduce una singolarità nella trasformata di Fourier. L'utilizzo della DFT, grazie al troncamento del dominio (escludendo S=0 ed S=+∞), ci permette però di eliminare tale singolarità. Per quanto ci riguarda abbiamo quindi utilizzato sempre $\epsilon = 0$, in quanto è stato dimostrato che tale valore appartiene al range per cui l'algoritmo risulta stabile [39].
Uno dei fattori che in moltissimi metodi crea possibili instabilità è la discretizzazione temporale. Nel nostro caso, la risoluzione analitica della ODE (2.17) evita instabilità dovute alla scelta del passo di discretizzazione Δt, infatti la soluzione (2.18) è valida per ogni $t_2 > t_1$.

La conoscenza dell'esponente caratteristico in forma chiusa ci garantisce un'ulteriore proprietà: essendo la funzione caratteristica di ogni variabile aleatoria limitata in modulo da 1, il termine $e^{\Psi \Delta t}$ rimane limitato per ogni Δt limitato, vedi [2]. È quindi garantita la stabilità della soluzione continua ricavata in (2.19). Per quanto riguarda la soluzione numerica (2.22), non è altro che il campionamento in un dominio limitato della soluzione continua, sono così garantite le stesse proprietà di stabilità della soluzione continua.

2.6.2 Convergenza

Trattiamo ora le proprietà di convergenza dell'algoritmo FST. Il metodo che utilizziamo per derivare l'ordine di convergenza segue un approccio puramente numerico, ed è stato largamente utilizzato in letteratura, per esempio da Halluin, Forsyth, and Vetzal [14] e successivamente da V.Surkov [50] negli articoli da cui siamo partiti per la nostra trattazione.
Sia $\mathbf{e}^{[N,M]}$ l'errore tra il valore approssimato $\mathbf{v}^{[N,M]}$, ricavato con N punti spaziali e M punti temporali, ed il valore esatto v. Ipotizziamo che l'errore sia della forma:

$$\mathbf{e}^{[N,M]} = |\mathbf{v}^{[N,M]} - v| = C_n N^{-p_n} + C_m M^{-p_m}\,, \tag{2.24}$$

dove p_n e p_m sono gli ordini di convergenza in spazio e tempo rispettivamente. Per quanto riguarda le opzioni europee, come detto in precedenza, non è richiesto l'utilizzo di time-stepping; per questo motivo l'equazione (2.24) si riduce a:

$$\mathbf{e}^{[N,\cdot]} = |\mathbf{v}^{[N,\cdot]} - v| = C_n N^{-p_n}\,. \tag{2.25}$$

Per ricavare quindi l'ordine di convergenza p_n ricaviamo la soluzione approssimata $\mathbf{v}^{[N,\cdot]} = \mathbf{v}(h)$ con diverse griglie spaziali, raddoppiando ogni volta il numero di punti. In questo modo il passo di discretizzazione $h = \Delta x$ si dimezza rispetto al passo precedente, ricaviamo quindi numericamente p_n come segue:

$$p_n = log_2 \frac{|\mathbf{v}(h) - \mathbf{v}(h/2)|}{|\mathbf{v}(h/2) - \mathbf{v}(h/4)|} \,. \tag{2.26}$$

I risultati numerici in Tabella 2.2 ci indicano che per le opzioni Europee l'ordine di convergenza è all'incirca quadratico. Ordine che diventa più oscillante e meno attendibile nei casi in cui le opzioni abbiano parametri irrealistici, maturity molto lontane o nel caso in cui non siano stati scelti in maniera corretta l'intervallo $\Omega = [x_{min}, x_{max}]$ e il passo di discretizzazione Δx.

Numero di punti	Prezzo	Variazione	log_2 ratio
2^{11}	17.48613083362		
2^{12}	17.48788693547	0.00175610184	
2^{13}	17.48832589387	0.00043895840	2.00022036906
2^{14}	17.48843568563	0.00010979175	1.99931451787
2^{15}	17.48846310377	0.00002741813	2.00156729514
2^{16}	17.48846992850	0.00000682473	2.00628626284
2^{17}	17.48847165703	0.00000172853	1.98122279296
2^{18}	17.48847210407	0.00000044703	1.95109039951

Tabella 2.2: Risultati di convergenza in spazio ottenuti per una Call Europea con modello di Merton jump-diffusion. Parametri $(r, q, \sigma, \lambda, \mu, \delta) = (0.05, 0.0, 0.25, 0.31, 0.32, 0.4)$, spot=100, strike=100 e maturity un anno.

L'ordine di convergenza quadratico è raggiunto non solo in corrispondenza dello spot-price, ma anche nei restanti punti della grigli spaziale; rimanendo naturalmente lontani dalle zone dove l'FFT crea una distorsione nei prezzi (cioè nell'intorno di x_{min} e x_{max}). Per quanto riguarda le opzioni che richiedono l'utilizzo di vari step temporali, possiamo innanzitutto calcolare gli ordini di convergenza spaziali e temporali in maniera distinta, fissando rispettivamente il passo Δt, come nell'equazione (2.26), e successivamente il passo Δx allo stesso modo:

$$p_m = log_2 \frac{|\mathbf{v}(\Delta t) - \mathbf{v}(\Delta t/2)|}{|\mathbf{v}(\Delta t/2) - \mathbf{v}(\Delta t/4)|} \,. \tag{2.27}$$

Per opzioni path-dependent, che richiedono l'utilizzo dell'algoritmo FST con time-step, parleremo dei relativi ordini di convergenza nelle sezioni a loro dedicate, in quanto differiscono in base al tipo di opzione considerata.
Per aumentare l'ordine di convergenza, in entrambe le variabili, è possibile utilizzare il metodo di estrapolazione di Richardson.

Estrapolazione di Richardson

L'estrapolazione di Richardson si basa sulla seguente approssimazione:

$$A = A(h) + \sum_{k=1}^{\infty} \alpha_k h^{\beta_k} , \tag{2.28}$$

cioè la quantità A è approssimabile dalla quantità $A(h)$, ottenuta con una discretizzazione di passo h, più altri termini. I coefficienti β devono soddisfare la proprietà $0 < \beta_i < \beta_{i+1}$ e vengono considerati come noti. Nel corso della nostra analisi utilizzeremo $\beta_i = i$ come suggerito in [35] e in [37]. Possiamo quindi approssimare A come:

$$A = A(h) + \alpha_1 h + O(h^2) . \tag{2.29}$$

Allo stesso modo potremmo utilizzare un'approssimazione di passo $\frac{h}{2}$:

$$A = A(h/2) + \alpha_1 \frac{h}{2} + O(h^2) . \tag{2.30}$$

Le equazioni (2.29) e (2.30) costituiscono un sistema di due equazioni in due incognite. Risolviamo in modo da ottenere α_1 e ricaviamo infine A:

$$A = 2A(h/2) - A(h) + O(h^2) . \tag{2.31}$$

Abbiamo quindi ricavato un'approssimazione di A di ordine maggiore, cioè di ordine h^2 invece che h.
La trattazione appena svolta è di carattere generale: nel caso di un algoritmo con ordine di convergenza k, l'estrapolazione di Richardson ci consente di ottenere ordine $k + 1$. Nel nostro caso l'ordine di convergenza è quadratico, la trattazione può quindi essere svolta ponendo:

$$A = A(h) + \alpha_2 h^2 + O(h^3)\,, \tag{2.32}$$

ed ottenere un'errore di ordine h^3.
È possibile ricavare in generale approssimazioni sempre più precise, prendendo in esame altri termini α_k della sommatoria presente in (2.28).
Riportiamo in Tabella 2.3 i valori ottenuti utilizzando l'estrapolazione di Richardson per la stessa opzione della Tabella 2.2. Notiamo che l'ordine di convergenza è passato da quadratico a cubico, come ci aspettavamo.

Numero di punti	Prezzo	Variazione	log_2 ratio
2^{11}			
2^{12}	17.48964303731		
2^{13}	17.48876485228	0.00087818503	
2^{14}	17.48854547739	0.00021937489	2.99975522234
2^{15}	17.48849052190	0.00005495548	3.00019600503
2^{16}	17.48847675323	0.00001376867	3.00941915233
2^{17}	17.48847338557	0.00000336766	2.99376804628
2^{18}	17.48847255110	0.00000083446	2.91328836680

Tabella 2.3: Risultati di convergenza in spazio ottenuti per una Call europea con modello di Merton jump-diffusion ed utilizzo dell'estrapolazione di Richardson.

2.6.3 Precisione

Come abbiamo visto in Tabella 2.1, i valori dell'opzione calcolati con il metodo FST sono ugualmente precisi a quelli calcolati con altri metodi. Se si confrontano i valori dell'opzione ottenuti anche lontano dallo spot, si può notare che il metodo FST rimane ugualmente preciso e l'ordine di convergenza rimane comunque quadratico. Non riportiamo qui i risultati ottenuti sulla precisione del metodo per punti lontani dallo spotprice (cioè lontani dal centro della griglia $x = 0$), avremmo modo di testare tale precisione nella Sezione 4.4, quando tratteremo l'equazione Forward.
I prezzi ottenuti con il metodo FST sono stati confrontati con quelli dei principali metodi di pricing, come differenze finite e metodi Monte Carlo; in tutti i casi trattati sono risultati in linea con gli altri metodi.

Capitolo 3

Opzioni Path-dependent e Regime-Switching

3.1 Introduzione

In questo capitolo analizzeremo in dettaglio come applicare il metodo FST ad opzioni di tipo path-dependent. In particolare tratteremo le opzioni Americane [Sezione 3.2] e le opzioni Barriera [Sezione 3.3]. Successivamente modificheremo lievemente l'algoritmo di base per renderlo applicabile a modelli con Regime-Switching [Sezione 3.4]. I modelli con salti sono in grado di riprodurre smile di volatilità realistici per scadenze molto brevi, sulle scadenze lontane invece, i modelli a volatilità stocastica riproducono meglio le dinamiche di mercato. Vedremo nel capitolo successivo come applicare il metodo FST al più noto modello a volatilità stocastica, il modello di Heston. L'utilizzo dei modelli con Regime-Switching ci permette di riprodurre smile consistenti con quelli di mercato [Sezione 3.4.1], senza dover per forza ricorrere alla volatilità stocastica. Infine, sempre tramite l'utilizzo del Regime-Switching, presenteremo un modello per titoli defaultable [Sezione 3.5]. La possibilità che un derivato sia legato ad un sottostante defaultable, è un fattore che spesso non viene tenuto in considerazione, ma è di grande importanza nel mondo reale.

3.2 Opzioni Americane

Il fattore che rende differente un'opzione Americana da un'opzione Europea è la possibilità di esercizio anticipato. Infatti un'opzione Americana può essere esercitata in ogni istante della sua vita, ossia $\forall t \in [0, T]$. Oltre alle opzioni Americane continue esiste la versione discreta, chiamata opzione Bermuda, che consente l'esercizio della stessa in alcune date prefissate $t_1, t_2, \ldots, t_M \in [0, T]$. La differenza tra i due tipi di esercizio anticipato non è per nulla banale, infatti il prezzo può essere anche molto diverso nei due casi. Facciamo un esempio: prendiamo un'opzione con maturity T pari ad un anno. La possibilità di esercitare l'opzione solo a sei mesi e a scadenza (opzione Bermuda con due date di esercizio), oppure in ogni momento durante il corso dell'anno (opzione Americana), cambia in maniera sostanziale i diritti che ne derivano per il possessore.
Sia $W(t, S_t)$ il prezzo in t di una generica opzione Bermuda con scadenza T, sottostante $S_t = e^{x_t}$ e strike price K. Il prezzo attuale $W(0, S_0)$ è la soluzione del seguente problema:

$$W(0, S_0) = \mathbb{E}\left[\max_{k=t_1 \ldots t_n} f(S_k, K)\right] . \tag{3.1}$$

Il prezzo di un'opzione Americana $V(0, S_0)$ si determina risolvendo un problema leggermente differente:

$$V(0, S_0) = \mathbb{E}\left[\max_{t\in[0,T]} f(S_t, K)\right] . \tag{3.2}$$

In particolare per un'opzione Bermuda o Americana di tipo Call $f(S, K)$ sarà pari a $(S - K)^+$, mentre per un'opzione di tipo Put avremo $(K - S)^+$.

Partiamo dall'analisi della versione continua (Americana). La possibilità di esercitare il diritto annesso all'opzione in ogni momento, impone la condizione che il prezzo della stessa si maggiore o uguale al payoff che è possibile ottenere esercitandola. Questa condizione deve valere onde evitare che si possano generare arbitraggi nel mercato. Facciamo un esempio: ipotizziamo che un'opzione Americana di tipo Call, in un determinato

istante $\bar{t}$, abbia valore $C(\bar{t}) < (S_{\bar{t}} - K)$. Potrei quindi comprare l'opzione al prezzo $C(\bar{t})$ ed esercitarla subito, guadagnando istantaneamente $(S_{\bar{t}} - K) - C(\bar{t})$. Questo genera un arbitraggio.

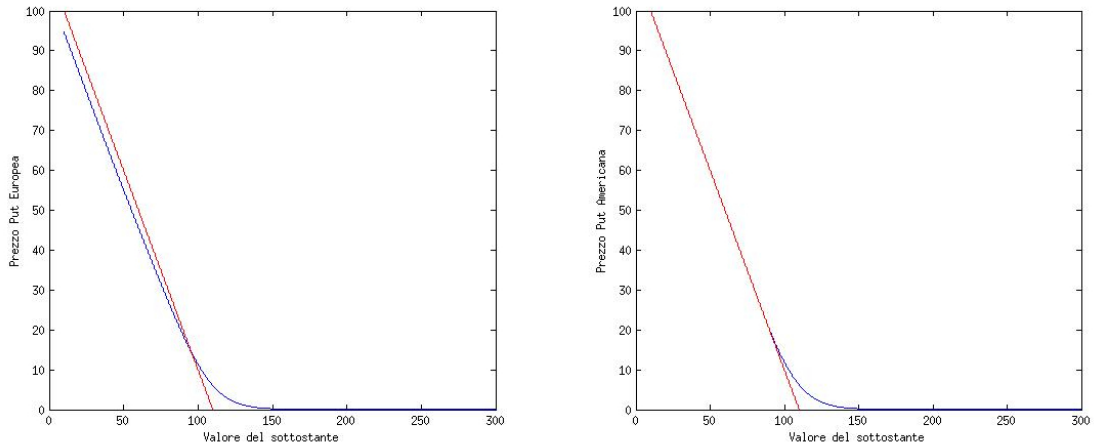

Figura 3.1: L'immagine a sinistra mostra il prezzo di un'opzione Put Europea (in blu), questo può trovarsi sotto la linea del payoff (in rosso). Non essendo immediatamente esercitabile questo non genera arbitraggi. A destra vediamo invece il prezzo di un'opzione Americana (in blu). Notiamo che è vincolato a rimanere sopra la linea del payoff (in rosso).

Il problema di pricing (3.2) si può riscrivere come la risoluzione di una PIDE [13], la stessa dell'equivalente Europeo, con l'aggiunta di un vincolo f, tale da garantire la condizione di non arbitraggio:

$$\begin{cases} \frac{\partial V}{\partial t} + \gamma \frac{\partial V}{\partial x} + \frac{\sigma^2}{2} \frac{\partial^2 V}{\partial x^2} - rV + \int_{\mathbb{R}\setminus\{0\}} \left(V(x+y) - V(x) - \mathbf{1}_{|y|<1} y \frac{\partial V}{\partial x} \right) dy = 0 \,, \\ V(t,x) \geq f(t,x) \quad \forall\, t \in [0,T] \,, \end{cases} \tag{3.3}$$

dove il vincolo per opzioni di tipo Call è pari a: $f(t,x) = (S - K)^+$, con $S = e^x$.
Per quanto riguarda le opzioni di tipo Put avremo invece: $f(t,x) = (K - S)^+$.

Il problema non è di facile risoluzione. I metodi semi-espliciti basati sulla trasfor-

mata di Fourier, come quelli di Carr-Madan o Lewis, non sono applicabili. Infatti non è possibile trovare una soluzione analitica del prezzo nello spazio delle frequenze.
Alcuni metodi Monte Carlo permettono di eseguire il pricing di questo tipo di opzioni, purtroppo però risultano difficili da implementare e poco efficienti. I metodi MC di tipo parametrico forniscono un intervallo per il prezzo, che spesso non è abbastanza accurato per essere utilizzato a fini pratici. I metodi MC regressivi, come quello di Longstaff e Schwarz (vedi [49]) sono molto più accurati, ma computazionalmente molto costosi; infatti richiedono la memorizzazione dell'intera matrice dei path. Inoltre l'algoritmo regressivo richiede di risolvere un problema di ottimizzazione ad ogni passo temporale, fattore che li rende generalmente molto lenti.
Per quanto riguarda i metodi variazionali (per esempio l'utilizzo degli elementi finiti), viene risolto un problema a frontiera libera equivalente al problema (3.3). Ad ogni step temporale è necessario calcolare la soluzione con un algoritmo iterativo, come ad esempio il SOR Proiettato [49]. Anche in questo caso il costo computazionale aumenta in maniera considerevole rispetto al pricing delle opzioni Europee. Sono stati recentemente sviluppati molti altri metodi per la risoluzione di questo tipo di problema, per esempio basati sull'utilizzo della densità di transizione o tramite l'utilizzo del metodo delle linee, vedi [10] [11] [41].
Il metodo FST permette di valutare questo derivato in maniera semplice. La modifica da apportare all'algoritmo FST è la seguente: ad ogni step temporale, conoscendo la soluzione in t_m, si calcola il prezzo in t_{m-1} come per le opzioni Europee, si impone la condizione al bordo (cioè che il prezzo dell'opzione sia maggiore o uguale al payoff che sarebbe possibile ottenere esercitandola) e si itera il procedimento $\forall t_i$ fino a t_0.
Vediamo come si modifica l'algoritmo per un'opzione di tipo Call:

$$v^*_{m-1} = \text{IFFT}[\,\text{FFT}[v_m] \cdot e^{\Psi(\cdot)\Delta t_m}\,], \tag{3.4}$$

$$v_{m-1} = \max(\,v^*_{m-1}\,,\,(S-K)^+\,), \tag{3.5}$$

dove v_m è il vettore del valore dell'opzione, calcolato in t_m.
Per opzioni di tipo Put è sufficiente sostituire $(S - K)^+$ con $(K - S)^+$.
Per prezzare opzioni Bermuda, il metodo richiede di utilizzare come Δt_m esattamente il tempo tra gli istanti di possibile esercizio; il prezzo si ottiene quindi con un numero di step temporali pari al numero di date di monitoraggio. Per quanto riguarda le opzioni Americane invece, esistono due tipi di approcci.
Il primo è quello di approssimare l'esercizio continuo con un'opzione Bermuda con un numero molto elevato di date di esercizio, in linea teorica facendo quindi tendere Δt_m a zero. Questo risulta però molto costoso dal punto di vista computazionale.
Il secondo approccio si basa sull'utilizzo dell'estrapolazione di Richardson. L'idea è di approssimare l'opzione americana con una serie di opzioni Bermuda, con numero di date di monitoraggio crescente. Utilizzeremo quindi quattro opzioni Bermuda come suggerito in [37] e [38]. Il risultato è il seguente:

$$V(t, S_t) \approx \frac{64V_{\Delta_{n+3}}(t, S_t) - 56V_{\Delta_{n+2}}(t, S_t) + 14V_{\Delta_{n+1}}(t, S_t) - V_{\Delta_n}(t, S_t)}{21}, \tag{3.6}$$

dove V indica il prezzo dell'opzione Americana, $V_{\Delta_{n+k}}$ il prezzo delle opzioni Bermuda con stessa maturity e 2^{n+k} date di esercizio. Utilizzando questo tipo di procedimento, non è necessario che le date di monitoraggio delle singole opzioni Bermuda siano molto ravvicinate.
Da notare che per prezzare le opzioni Bermuda, non imporremmo la condizione (3.5) nell'ultimo step temporale. Prendiamo per esempio un'opzione Bermuda di tipo Call, con la prima data di monitoraggio tra un mese. Il valore dell'opzione in t_0 può trovarsi sotto la linea del payoff $(S - K)^+$, in quanto in questo momento non è esercitabile; non potrà invece rimanere al di sotto nelle date di possibile esercizio. La condizione (3.5) viene invece imposta, anche all'ultimo step, per approssimare le opzioni Americane continue, sia nel caso di utilizzo dell'estrapolazione di Richardson, sia nel caso contrario. Infatti il diritto annesso all'opzione deve poter essere esercitato in ogni istante, senza

poter generare arbitraggi.

Occupiamoci ora dell'ordine di convergenza del metodo, procederemo quindi in questo modo: inizialmente fisseremo il numero di step temporali e andremo a valutare la convergenza spaziale, poi fisseremo il numero di punti spaziali e studieremo la convergenza temporale.
Riportiamo i risultati ottenuti per la convergenza spaziale in Tabella 3.1 e per quella temporale in Tabella 3.2.

Numero di punti spaziali	Prezzo	Variazione	log_2 ratio
2^{10}	5.64647091120		
2^{11}	5.64943398159	0.00296307039	
2^{12}	5.65024195779	0.00080797619	1.8747
2^{13}	5.65044057743	0.00019861963	2.0243
2^{14}	5.65048799198	0.00004741454	2.0666
2^{15}	5.65050042943	0.00001243745	1.9306
2^{16}	5.65050363474	0.00000320530	1.9561
2^{17}	5.65050437910	0.00000074436	2.1063
2^{18}	5.65050456697	0.00000018786	1.9862

Tabella 3.1: Risultati di convergenza spaziale ottenuti per un'opzione Bermuda di tipo Put con modello di Merton jump-diffusion. Parametri $(r, q, \sigma, \lambda, \mu, \delta) = (0.05, 0, 0.15, 0.1, -0.38, 0.4)$, spot=100, strike=100 e maturity un anno. Utilizzate 256 valutazioni temporali, corrispondenti ad una possibilità di esercizio all'incirca giornaliera.

L'analisi dei risultati ottenuti, ci suggerisce che l'ordine di convergenza del metodo sia due in spazio ed uno in tempo per le opzioni Americane. Da notare che le opzioni Bermuda sono valutate esattamente negli istanti di esercizio, la griglia temporale è fissata: non ha quindi senso parlare di ordine di convergenza temporale.

Confrontiamo infine i prezzi ed i tempi di calcolo ottenuti per il pricing di un'opzione Americana. Presentiamo in Tabella 3.3 sia l'approssimazione con opzioni Bermuda con molte date di monitoraggio, sia il pricing ottenuto utilizzando l'estrapolazione di Richardson in tempo.

Numero di punti temporali	Prezzo	Variazione	log_2 ratio
2^5	5.63535744076		
2^6	5.64381545451	0.00845801375	
2^7	5.64808919675	0.00427374224	0.9848
2^8	5.65024195779	0.00215276104	0.9893
2^9	5.65131951590	0.00107755810	0.9984
2^{10}	5.65186232358	0.00054280768	0.9892
2^{11}	5.65212708193	0.00026475834	1.0357
2^{12}	5.65231019175	0.00018310981	0.5319
2^{13}	5.65238633759	0.00007614584	1.2658

Tabella 3.2: Risultati di convergenza in tempo ottenuti per una Put Americana con modello di Merton jump-diffusion. Stessi parametri della Tabella 3.1. Numero di punti spaziali fissati a 2^{12}.

Numero di punti temporali	Prezzo	Tempo (sec.)
2^4	9.3183518797	1.162620
2^5	9.3269550084	2.332468
2^6	9.3311524295	3.561804
2^7	9.3331995095	5.837837
Richardson	9.3352024187	12.894729
2^9	9.3342003636	26.332056
2^{12}	9.3351119764	199.444000

Tabella 3.3: Risultati ottenuti per una Put Americana con modello di Merton jump-diffusion. Parametri $(r, q, \sigma, \lambda, \mu, \delta) = (0.05, 0, 0.25, 0.1, 0.32, 0.4)$, spot=100, strike=100 e maturity un anno. Le prime quattro opzioni bermuda sono state utilizzate per calcolare il valore approssimato con l'estrapolazione di Richardson (3.6).

Notiamo innanzitutto che i prezzi sono crescenti all'aumentare delle di date di esercizio. L'approssimazione ottenuta tramite l'utilizzo dell'estrapolazione di Richardson è in linea con l'approssimazione di esercizio continuo, fatta con 2^{12} punti, ma dal punto di vista computazionale è estremamente più rapida.

3.3 Opzioni Barriera

Trattiamo ora il pricing di opzioni Barriera con il metodo FST. Un'opzione Barriera ha un payoff che dipende dal percorso seguito dal sottostante, se durante la vita dell'opzione questo raggiunge un determinato livello il valore dell'opzione cambia. Possiamo dividere le opzioni barriera in due tipi: tipo In e tipo Out. Le prime si attivano quando il sottostante entra in una determinata regione di prezzi (delimitata dalla barriera). Le seconde invece, sono attive da subito e smettono di esserlo se il sottostante esce da una determinata regione (delimitata da una o due barriere).
Analizziamo più in dettaglio le opzioni di tipo Out. Quelle che smettono di essere attive quando il sottostante supera un certo valore vengono chiamate Up-and-Out, quelle che smettono di essere attive se il sottostante scende sotto una soglia prestabilita sono dette Down-and-Out. È inoltre possibile avere un opzione che rimane in vita se il sottostante non esce da una regione delimitata da entrambe le barriere (Up e Down), in questo caso si parla di opzione Knock-and-Out.
Allo stesso modo si definiscono le opzioni di tipo In, con la semplice differenza che il raggiungimento della barriera serve per rendere attiva l'opzione.
Come nel caso di opzioni Americane e Bermuda, anche per le opzioni Barriera esistono quelle di tipo discreto e quelle di tipo continuo, che differiscono per come viene effettuato il monitoraggio del sottostante. Le opzioni a monitoraggio continuo sono soggette ai movimenti del sottostante S_t $\forall t \in [0, T]$, dove T è la maturity dell'opzione. Per quanto riguarda la versione discreta, il controllo del valore del sottostante, rispetto alle barriere, viene effettuato solo ad alcuni istanti prestabiliti $t_1, \ldots, t_m = T$. Per esempio, un'opzione di tipo Out a monitoraggio giornaliero (ipotizziamo che venga considerato il prezzo di chiusura della borsa) resta valida se durante la giornata il sottostante ha superato la barriera ma è rientrato prima della chiusura.

Per quanto riguarda il payoff $f(T, S)$, dove come sempre utilizziamo $S_t = e^{x_t}$, questo è definito nella maniera seguente:

$$f(T, S) = \max(S_T - K, 0)\, \mathbf{1}_{g(S_t)} \quad \text{per opzioni di tipo Call}, \tag{3.7}$$

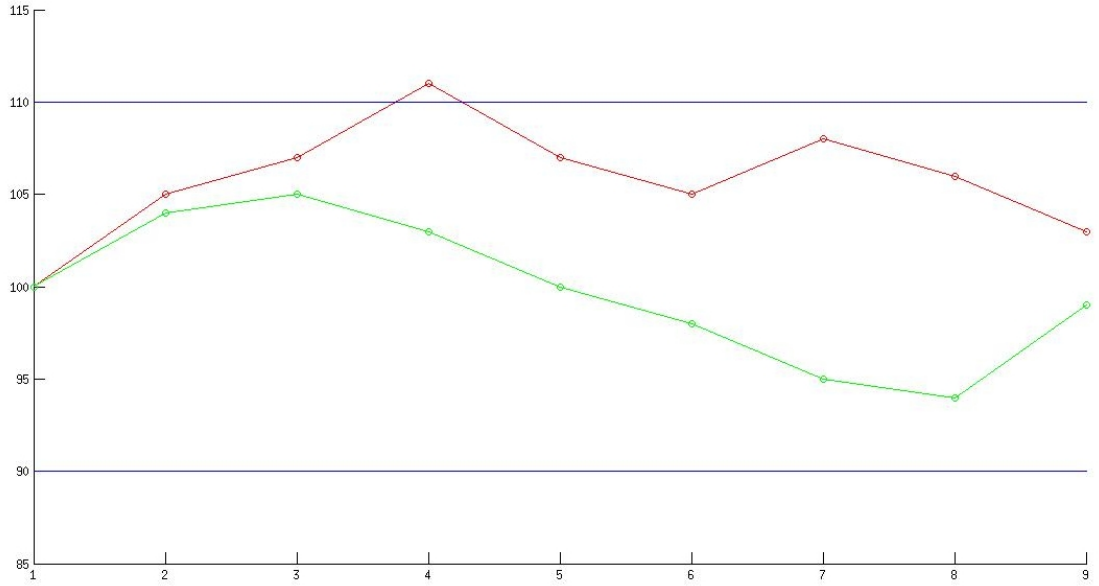

Figura 3.2: Mostriamo due possibili percorsi per un sottostante che oggi vale 100. Ipotiziamo un'opzione di tipo Knock-and-Out a monitoragio giornaliero. In blu sono segnate le barriere rispettivamente a 90 e 110. Il percorso verde si mantiene sempre nell'area delimitata tra le due barriere, l'opzione rimane quindi attiva fino a scadenza. Il percorso rosso invece supera la barriera, toccando quota 111 il quarto giorno. Alla scadenza (giorno 9), il sottostante è rientrato nella zona delimitata dalle due barriere ma l'opzione non ha comunque alcun valore, in quanto ha smesso di essere attiva il giorno 4, quando il sottostante ha attraversato la barriera superiore.

$$f(T,S) = \max(K - S_T, 0)\,\mathbf{1}_{g(S_t)} \quad \text{per opzioni di tipo Put}\,, \tag{3.8}$$

dove la funzione $g(S_t)$ indica la condizione per cui l'opzione risulta attiva, condizione che è differente se l'opzione è di tipo In o Out, se il monitoraggio è continuo o discreto e se si hanno una o due barriere. Per esempio, un'opzione Down-and-Out a monitoraggio continuo con barriera D vale:

$$g(S_t) = S_t > D \quad \forall t \in [0,T]\,. \tag{3.9}$$

È inoltre possibile aggiungere un rebate costante nel caso in cui venga toccata la barriera,

cioè il pagamento di una somma di denaro prestabilita qualora l'opzione smettesse di essere attiva. La modifica è banale, in quanto richiede di modificare il payoff aggiungendo la costante R moltiplicata per una funzione indicatrice, che vale uno se la condizione $g(S_t)$ non è stata rispettata.
La risoluzione del problema di pricing per opzioni di tipo Out è legata ad una PIDE, la stessa del caso Europeo, con l'aggiunta di un vincolo che indica se l'opzione si trova nella regione di attività o no. Il valore per le opzioni di tipo In viene ottenuto tramite la In-Out Parity: sia C il prezzo di un'opzione Europea, C_{out} e C_{in} quelle di un'opzione Barriera di tipo Out e una di tipo In con la stessa barriera. In tal caso vale la relazione:

$$C = C_{in} + C_{out} \,. \tag{3.10}$$

Il metodo FST consente di prezzare facilmente le opzioni Barriera di tipo Out, imponendo una condizione tra i vari step temporali come nel caso delle opzioni Americane.
Vediamo in maggiore dettaglio l'implementazione per un'opzione Up-and-Out, con barriera B e rebate R. Dato che nell'algoritmo di pricing preferiamo utilizzare il log-price x, scriviamo la log-barriera $b = \ln(B)$. La risoluzione di un singolo step temporale sarà quindi svolta nella maniera seguente:

$$v_{m-1} = \text{IFFT}[\,\text{FFT}[v_m] \cdot e^{\Psi(\cdot)\Delta t_m}\,] \cdot \mathbf{1}_{x>b} + R \cdot \mathbf{1}_{x\leq b} \,. \tag{3.11}$$

Da notare che per le opzioni a monitoraggio discreto l'algoritmo prevede un numero di step temporali pari al numero di date di monitoraggio. Inoltre, come nel caso di opzioni Bermuda, nell'ultimo step temporale non bisogna imporre la condizione sulla barriera (cioè l'indicatrice davanti al rebate R viene messa a zero e l'altra ad uno). Prendiamo un'opzione Down-and-Out con barriera L e primo monitoraggio tra un mese, se il sottostante oggi vale $S_t < L$ opzione risulta comunque attiva, vedi Figura 3.3.
Per quanto riguarda le opzioni a monitoraggio continuo, il prezzo si può ottenere in due modi: approssimandole con monitoraggi discreti tali che $\Delta t_m \to 0$, oppure utilizzando

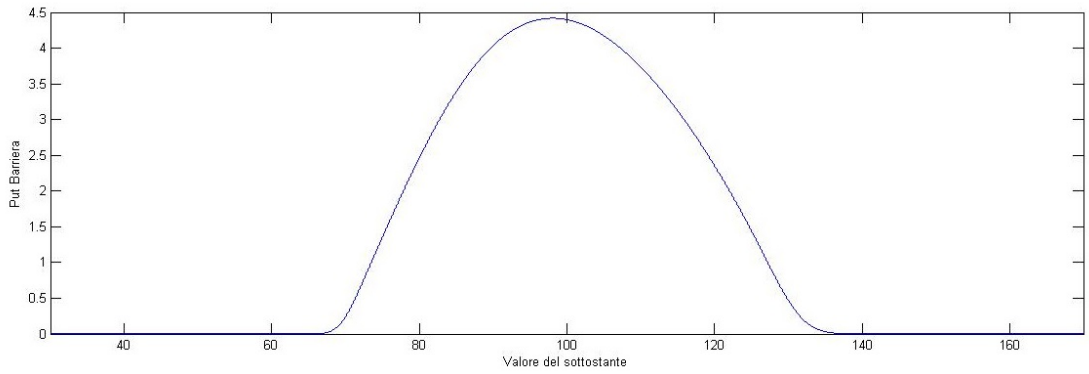

Figura 3.3: Prezzo di una Put Barriera Knock-and-Out con maturity un anno, 128 date di monitoraggio e barriere a 70 e 130.

l'estrapolazione di Richardson come nel caso di opzioni Americane. Notiamo che se il numero di step temporali non è abbastanza elevato, otteniamo una distorsione del prezzo nell'intorno della barriera, problema che può essere risolto aumentando il numero di step temporali (Figura 3.4).
Notiamo invece che, non dovendo imporre il vincolo sulla barriera nell'ultimo step temporale, le opzioni a monitoraggio discreto sono prive di questo inconveniente in quanto il problema di pricing è risolto in maniera esatta e non approssimata (Figura 3.3).
Analizziamo ora l'ordine di convergenza del metodo, prima in spazio mantenendo fisso il numero di passi temporali, poi in tempo, mantenendo fisso il numero di punti sulla griglia spaziale. I risultati ottenuti sono riportati in Tabella 3.4 ed in Tabella 3.5.
Per le opzioni barriera a monitoraggio continuo abbiamo trovato un ordine di convergenza pari ad uno in spazio ed uno in tempo. Per le opzioni a monitoraggio discreto, non ha senso parlare di convergenza temporale, dato che gli istanti di valutazione sono fissati.

Torniamo per un momento ad occuparci di opzioni a monitoraggio continuo, in particolare del problema del prezzo nell'intorno della barriera. Come evidenziato in precedenza, l'imposizione della condizione al bordo crea una distorsione del prezzo in quell'intorno. L'aumento degli step temporali per ridurre questo effetto è un metodo corretto, ma risulta abbastanza costoso, soprattutto poichè l'ordine di convergenza in tempo è pari

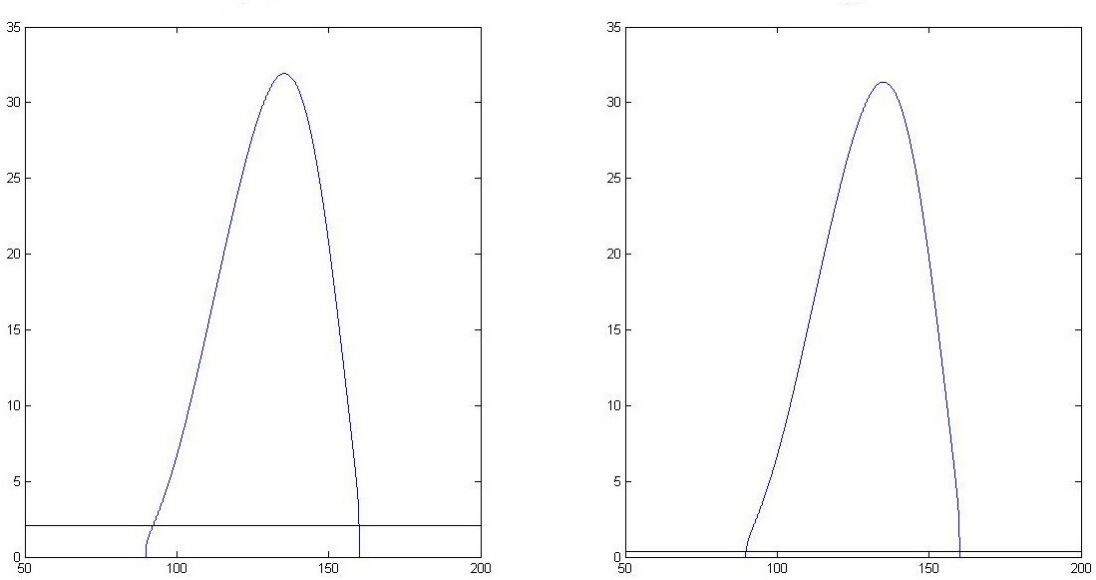

Figura 3.4: A sinistra, prezzo di un'opzione Call Barriera continua approssimata da una discreta con 32 date di monitoraggio, a destra con 256 date di monitoraggio. La linea nera indica l'errore generato dall'imposizione del vincolo (valore del derivato uguale a zero all'esterno della barriera), notiamo che questo diminuisce considerevolmente all'aumentare delle date di monitoraggio.

ad uno. Per risolvere il problema due sono le strade possibili: la prima è utilizzare l'estrapolazione di Richardson: per aumentare l'ordine di convergenza temporale o per approssimare l'esercizio continuo come fatto per le opzioni Americane (3.6). Il secondo è utilizzare il metodo delle immagini proposto da Buchen nel 1996 [7] [8]. Purtroppo quest'ultimo non introduce nessun tipo di distorsione nei prezzi solo per i modelli diffusivi, è invece provato che introduce un bias se il sottostante segue un modello con salti, come spiegato da V.Surkov nella sua tesi [50]. Per questo motivo ne abbiamo tralasciato l'implementazione, data la scarsa applicabilità nel nostro contesto, che si basa principalmente sull'utilizzo di modelli non puramente diffusivi.

Numero di punti spaziali	Prezzo	Variazione	log_2 ratio
2^{11}	4.65553796039		
2^{12}	4.55904408294	0.09649387744	
2^{13}	4.60232096731	0.04327688437	1.1568
2^{14}	4.62436165181	0.02204068449	0.9734
2^{15}	4.61303150141	0.01133015039	0.9600
2^{16}	4.61774142915	0.00470992774	1.2663
2^{17}	4.62010239657	0.00236096741	0.9963
2^{18}	4.62150493017	0.00140253360	0.7513
2^{19}	4.62220658634	0.00070165616	0.9991

Tabella 3.4: Risultati di convergenza spaziale ottenuti per una Put Barriera con modello di Merton jump-diffusion. Parametri $(r, q, \sigma, \lambda, \mu, \delta) = (0.05, 0, 0.25, 0.31, 0.32, 0.4)$, spot=100, strike=100, maturity un anno, barriere a 70 e 130. Utilizzate 64 date di monitoraggio.

Numero di punti temporali	Prezzo	Variazione	log_2 ratio
2^3	6.13403618		
2^4	5.98867263	0.14536355	
2^5	5.90500580	0.08366682	0.7969
2^6	5.86015851	0.04484728	0.8996
2^7	5.83698180	0.02317671	0.9523
2^8	5.82523152	0.01175027	0.9799
2^9	5.81932624	0.00590527	0.9926
2^{10}	5.81636642	0.00295982	0.9964
2^{11}	5.81488430	0.00148212	0.9978

Tabella 3.5: Risultati di convergenza temporale ottenuti per una Call Barriera Knock-and-Out, modello Variance Gamma. Parametri $(r, q, \sigma, \theta, k) = (0.05, 0, 0.10, 0.03, 0.2)$, spot=100, strike=100, maturity un anno, barriere a 70 e 130. Utilizzati 2^{16} punti sulla griglia spaziale.

3.4 Regime-Switching

Trattiamo ora l'estensione del metodo FST a modelli con Regime-Switching, largamente studiati ed utilizzati in letteratura, vedi [12] [21] e [22].

L'idea alla base di questi modelli è la seguente: il mondo finanziario si trova in condizioni differenti al passare del tempo, per esempio periodi di alta e bassa volatilità si alternano

in maniera impredicibile. La stessa cosa può essere pensata per un sottostante. Sia $S(t)$ il prezzo di un generico asset, la dinamica risk-neutral di questo dipende da diversi parametri: il tasso d'interesse risk-free r, la volatilità σ e se utilizziamo modelli con salti, anche i parametri che descrivono il processo di puro salto. Ipotizziamo ora che ci siano d diversi stati del mondo, ognuno caratterizzato da diversi parametri $(r_i, \sigma_i, \ldots)$, con $i = 1, \ldots, d$; possiamo associare ad ogni stato del mondo un processo $S_i(t) = e^{X_i(t)}$, con $X_i(t)$ dipendente dai parametri dello stato i-esimo:

$$dX_i(t) = \log(S_0) + \gamma_i t + \sigma_i dW_i(t) + dJ_i(t)\,, \tag{3.12}$$

dove con la lettera J indichiamo la parte di processo relativa ai salti.
Non si tratta semplicemente di calcolare il valore di d opzioni, una per ogni stato del mondo (caratterizzato da un processo differente), ma di legare questi processi in maniera stocastica. Associamo ad ogni stato del mondo i, una probabilità di saltare in un altro stato del mondo j; questo può essere fatto legandoli tramite una catena di Markov.
Sia $\mathbf{Q}$ la matrice dei tassi di transizione $d \times d$ associata al processo di Markov. La probabilità che il processo in un tempo Δt salti dallo stato i allo stato j è pari a: $exp(\mathbf{Q}\Delta t)_{ij}$, dove ij indica riga i e colonna j della matrice esponenziale. La modellistica più utilizzata in letteratura consente al processo di saltare solo negli stati adiacenti, sempre con la stessa probabilità. In tal caso sia λ il tasso di transizione (costante per ogni stato), la matrice $\mathbf{Q}$ viene definita come segue:

$$\mathbf{Q} = \begin{pmatrix} -\lambda & \lambda & 0 & 0 & \cdots & 0 \\ \lambda & -2\lambda & \lambda & 0 & \cdots & 0 \\ 0 & \lambda & -2\lambda & \lambda & \cdots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & \lambda & -2\lambda & \lambda \\ 0 & \cdots & 0 & 0 & -\lambda & \lambda \end{pmatrix}.$$

Negli esempi sviluppati in questa tesi, utilizzeremo solo matrici di transizione di questo

tipo; la trattazione verrà invece svolta per una generica matrice di transizione $\mathbf{Q}$.
Sia $V_k(t, x)$ il valore di un'opzione Europea al tempo t, con valore del log-spot pari a x, condizionato al fatto di trovarsi al tempo t nello stato k-esimo. Il prezzo soddisfa il seguente sistema di PIDE: $\forall k = 1, \dots, d$

$$\begin{cases} (\partial_t + \mathbf{Q}_{kk} + \mathcal{L}_k - r)V_k(t, x) + \sum_{l \neq k} \mathbf{Q}_{kl} V_l(t, x) = 0\,, \\ V_k(T, x) = \varphi(x)\,, \end{cases} \tag{3.13}$$

dove $\mathcal{L}_k$ indica il generatore infinitesimale del processo di Lévy k-esimo e $\mathbf{Q}_{ij}$ l'elemento ij della matrice $\mathbf{Q}$.
Applicando la trasformata di Fourier alla (3.13) rispetto al sottostante x, il sistema di PIDE si riduce al seguente sistema di ODE:

$$\begin{cases} (\partial_t + \mathbf{Q}_{kk} + \Psi_k(\omega) - r)\mathcal{F}[V_k](t, \omega) + \sum_{l \neq k} \mathbf{Q}_{kl} \mathcal{F}[V_l](t, x) = 0\,, \\ \mathcal{F}[V_k](T, \omega) = \mathcal{F}[\varphi](\omega)\,. \end{cases} \tag{3.14}$$

Sia $\mathbf{V}$ il vettore d-dimensionale composto dalle V_k, con $k = 1, \dots, d$. Possiamo riscrivere il sistema in una forma più compatta:

$$\begin{cases} (\partial_t + \mathbf{\Psi}(\omega) - r\, \mathbb{1}_{d\times d})\mathcal{F}[\mathbf{V}](t, \omega) = 0\,, \\ \mathcal{F}[\mathbf{V}](T, \omega) = \mathcal{F}[\varphi](\omega)\, \mathbb{1}_d\,, \end{cases} \tag{3.15}$$

dove $\mathbb{1}_{d\times d}$ indica la matrice identità di dimensione $d \times d$, mentre $\mathbf{\Psi}(\omega)$ è una matrice della stessa dimensione definita come segue:

$$(\mathbf{\Psi}(\omega))_{kl} = \begin{cases} \mathbf{Q}_{kk} + \Psi_k(\omega) & se\ k = l\,, \\ \mathbf{Q}_{kl} & se\ k \neq l\,. \end{cases} \tag{3.16}$$

Conoscendo la condizione finale $\mathbf{f}$, la soluzione in un tempo $t < T$ può essere quindi

ricavata come segue:

$$\mathcal{F}[\mathbf{V}](t,\omega) = exp\{(\mathbf{\Psi}(\omega) - r\mathbb{1}_{d\times d})(T-t)\}\mathcal{F}[\mathbf{f}](\omega)\,. \tag{3.17}$$

Applicando la discretizzazione numerica come abbiamo fatto in precedenza, l'algoritmo rsFST per sottostanti con Regime-switching, nel caso di opzioni Europee, è quindi il seguente:

$$\mathbf{v}_t = \text{IFFT}[\,\text{FFT}[\mathbf{v}_T] \cdot e^{(\mathbf{\Psi}-r)(T-t)}\,]\,. \tag{3.18}$$

È utile notare che, nell'equazione (3.17), $\mathbf{V}(t,x)$ è un vettore contenente i d valori delle opzioni nei d stati del mondo, mentre $\mathbf{\Psi}(\omega)$ è una matrice $d \times d$. La discretizzazione aumenta la dimensionalità del problema, infatti il vettore d-dimensionale $\mathbf{V}(t,x)$, ha come corrispettivo discreto la matrice $\mathbf{v}_t$, di dimensione $N \times d$, contenente la discretizzazione in N punti delle d componenti di $\mathbf{V}(t,x)$. Allo stesso modo, la matrice $\mathbf{\Psi}(\omega)$ viene discretizzata in un tensore tridimensionale; la gestione numerica del problema diventa quindi leggermente più complicata.
Come dicevamo in precedenza, i modelli Regime-Switching trasformano il pricing di un derivato in un problema multidimensionale. La soluzione $\mathbf{V}(t,x)$ è un vettore contenente le soluzioni $V_k(t,x)$ di ogni stato del mondo. Quando si effettua il pricing si possono fare due ipotesi: che lo stato del mondo in cui ci troviamo sia noto oppure no. Nel primo caso il prezzo dell'opzione è il $V_k(t,x)$ corrispondente allo stato del mondo attuale k. Nel secondo caso invece, il prezzo dell'opzione è una media pesata dei $V_k(t,x)$, dove i pesi saranno le probabilità associate ad ogni stato del mondo. Queste probabilità possono essere quelle che un investitore neutrale al rischio assegna ai diversi stati del mondo, oppure possono essere ricavate dai dati storici tramite l'utilizzo del filtro di Kalman.

Per quanto riguarda le opzioni di tipo path-dependent, l'algoritmo rsFST si comporta in maniera molto simile all'algoritmo FST. Ad ogni step temporale si utilizza l'equazione

(3.18) e si impongono i vincoli sul prezzo; l'imposizione del vincolo è però leggermente differente.
Prendiamo per esempio un'opzione Bermuda. È necessario distinguere la trattazione in due casi: stato del mondo in cui ci si trova noto oppure no.
Nel primo caso è possibile applicare la condizione ad ogni stato del mondo k indipendentemente. Infatti, se ci è noto lo stato del mondo k in cui ci troviamo, siamo in grado di sapere qual'è il valore $V_k(t, x)$ corretto della nostra opzione. Basta quindi imporre il vincolo che il valore dell'opzione si trovi sopra il payoff, come abbiamo fatto con il metodo FST:

$$v^*_{m-1} = \text{IFFT}[\,\text{FFT}[v_m] \cdot e^{\boldsymbol{\Psi}(\cdot)\Delta t_m}\,], \tag{3.19}$$

$$v_{m-1} = \max(\, v^*_{m-1}\,, (S-K)^+\,), \tag{3.20}$$

Nel caso in cui lo stato del mondo non sia noto, l'algoritmo si modifica leggermente. Infatti, non essendo a conoscenza dello stato in cui ci troviamo, non siamo in grado di scegliere il giusto $V_k(t, x)$. Ad ogni step è quindi necessario calcolare (3.19), successivamente bisogna trovare le probabilità di trovarsi in un determinato stato del mondo k. Data la distribuzione iniziale di probabilità $\vec{p}$, la distribuzione in t_{m-1} è data da: $exp\{(m-1)\,\Delta t\,\mathbf{Q}\}\vec{p}$. Otteniamo quindi l'algoritmo :

$$v^*_{m-1} = \text{IFFT}[\,\text{FFT}[v_m] \cdot e^{\boldsymbol{\Psi}(\cdot)\Delta t_m}\,], \tag{3.21}$$

$$\hat{v}_{m-1} = exp\{(m-1)\,\Delta t\,\mathbf{Q}\}\vec{p} \cdot v^*_{m-1}\,, \tag{3.22}$$

$$v_{m-1} = \max(\, \hat{v}_{m-1}\mathbb{1}_d\,, (S-K)^+\,). \tag{3.23}$$

Il valore di continuazione (3.23) è quindi lo stesso per ogni stato del mondo. Non conoscendo lo stato k dove ci troviamo, non possiamo sapere se l'esercizio anticipato è

ottimale oppure no; per questo motivo il valore dell'opzione è dato dalla media pesata dei valori nei diversi stati del mondo (3.22).

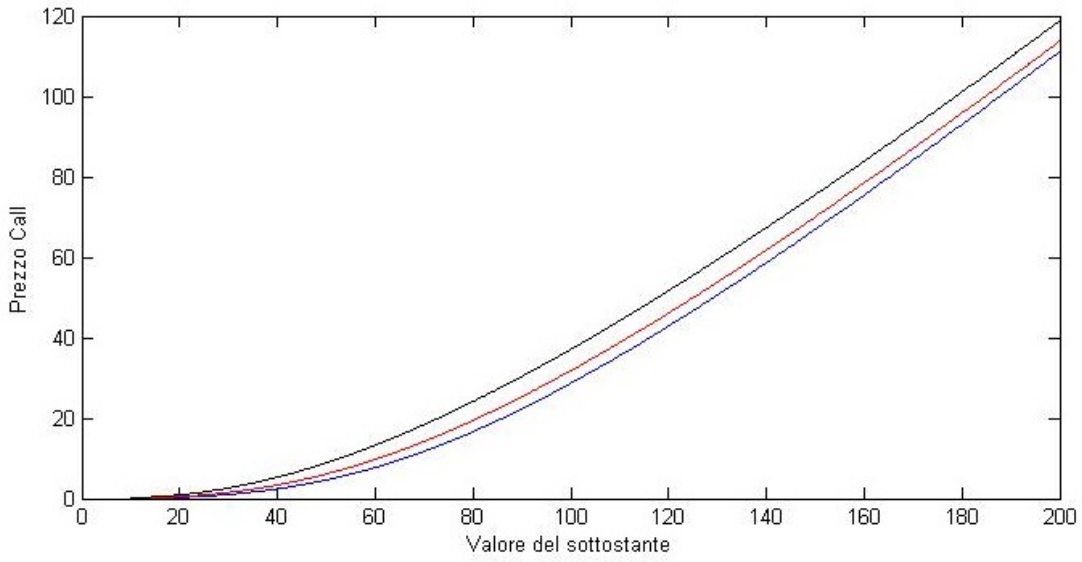

Figura 3.5: Prezzi di una Call Europea con modello di Merton e Regime Switching in 3 diversi stati del mondo, blu=1, rosso=2 e nero=3. Ipotizziamo gli stati del mondo noti. Parametri $(r, q, \sigma_1, \sigma_2, \sigma_3, \lambda, \mu, \delta) = (0.05, 0, 0.10, 0.4, 0.9, 0.8, -0.22, 0.3)$, spot=100, strike=110, maturity 2 anni, probabilità dei diversi stati (0.2,0.3,0.5).

Riportiamo in Figura 3.5 il prezzo che un'opzione Call con Regime-Switching avrebbe nei diversi stati del mondo, se questi fossero noti. In Tabella 3.6 riportiamo il prezzo ottenuto ipotizzando gli stati del mondo non noti. Riportiamo inoltre i prezzi ottenuti con i tre modelli sottostanti al Regime-Switching e la media di questi pesata sulle probabilità degli stati del mondo. Notiamo che il prezzo con Regime-Switching e quello ottenuto con la semplice media sono differenti, come ci aspettavamo.

Sfrutteremo all'interno della sezione successiva l'algoritmo con Regime-Switching, per fare vedere che è in grado di generare superfici di volatilità implicita consistenti con quelle di mercato.

	Prezzo
Regime Switching	33.77799
σ_1	17.18644
σ_2	27.49053
σ_3	49.80825
media	36.58857

Tabella 3.6: Prezzi ottenuti con e senza Regime-Switching.

3.4.1 Volatilità implicita

Sul mercato borsistico le opzioni sono spesso quotate in termini di volatilità implicita e non in termini di prezzo. La volatilià implicita non è altro che quel numero che, messo nella formula di Black&Scholes al posto della volatilità, rende il prezzo di mercato pari al prezzo ricavato dal modello. Sia $C_{MKT}(K,T)$ il prezzo di mercato di una Call Europea con strike K e maturity T. Nel modello di Black&Scholes l'unico ulteriore parametro che serve per effettuare il pricing è la volatilità σ. Sia quindi $C_{BS}(\sigma,K,T)$ il prezzo ricavato con il modello. La volatilità implicita σ_{imp} è quel numero che risolve il seguente problema:

$$C_{BS}(\sigma_{imp}) = C_{MKT}\,, \tag{3.24}$$

dove abbiamo tralasciato la dipendenza da K e T, essendo uguali per entrambi i termini della (3.24).
Dato quindi un set di opzioni scambiate sul mercato $C_{MKT}(K,T)$, per vari strike K e varie maturity T, è possibile ricavare quei valori di volatilità che rendono i prezzi di Black&Scholes pari ai prezzi di mercato. In questo modo si ricava la superficie di volatilità implicita $\sigma_{imp}(K,T)$. Fissando una maturity, possiamo vedere la volatilità implicita come funzione del solo strike, $\sigma_{imp} = \sigma_{imp}(K)$, in questo caso si parla di *smile* di volatilità.

Presentiamo ora due superfici di volatilità, ottenute con il modello di Merton con Regime-Switching a 5 stati del mondo e senza in Figura 3.6. Per fare questo, una volta

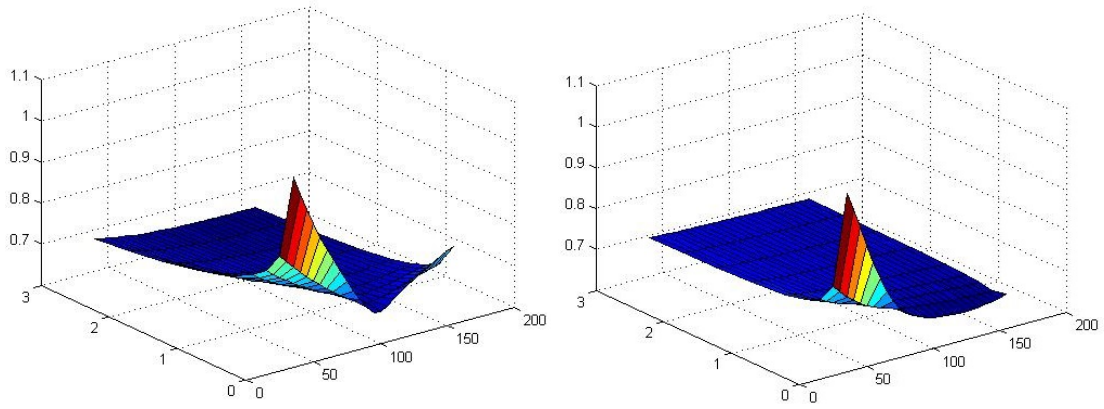

Figura 3.6: L'immagine a sinistra mostra la superficie di volatilità implicita ottenuta con un modello di Merton con Regime-Switching, parametri $(r, q, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \lambda, \mu, \delta) = (0.05, 0, 0.10, 0.3, 0.5, 0.7, 0.9, 0.8, -0.22, 0.3)$ e probabilita degli stati del mondo pari a (0.1,0.1,0.2,0.3,0.3). A destra senza Regime-Switching, stessi parametri del modello precedente ma volatilità fissata $\sigma = 0.62$. Abbiamo considerato maturity comprese tra 0.1 e 3 anni, strike tra 40 e 160, con un prezzo spot pari a 100.

fissati i parametri dei due modelli, abbiamo calcolato i prezzi delle opzioni per vari strike e maturity, successivamente li abbiamo invertiti per ricavare la volatilità implicita. Nel secondo caso il valore di volatilità è stato fissato pari alla media delle volatilità nei diversi stati del mondo del Regime-Switching, media ottenuta pesando i valori con le probabilità relative ad ogni stato del mondo. Avremmo potuto scegliere un valore di volatilità qualsiasi ed avremmo ottenuto la stessa forma di superficie. La scelta del valore medio è stata fatta per rimarcare che i risultati ottenuti con il modello con Regime-Switching non sono, in alcun modo, la media dei risultati ottenibili con i modelli dei vari stati del mondo.

Notiamo che l'utilizzo del Regime-Switching ci permette di ottenere una superficie non piatta anche per scadenze lontane, diversamente dal modello con i soli salti, che sia appiattisce quasi subito.

3.5 Modello per titoli defaultable

Il default di un titolo azionario è un evento che non condiziona solo il valore del titolo stesso, ma anche quello dei derivati ad esso legati. Prendiamo per esempio un'opzione Europea di tipo Put. In linea teorica, il possessore riceve il massimo payoff $(K - S)^+$ se il titolo sottostante va a zero, ma il fatto che valga zero vuol dire che è avvenuto l'evento di default. In questo caso il valore dell'opzione può non essere più quello che ci si aspetta. Tali eventi vengono considerati nel momento della stipula del contratto derivato, in maniera differente per ogni contratto. Presentiamo alcune possibilità:

1. In caso di default il valore del sottostante viene posto pari a zero, l'opzione paga quindi K.

2. In caso di default il valore del sottostante viene posto pari al recovery-rate Π, l'opzione paga quindi $(K - \Pi)^+$ (supponendo di conoscere Π).

3. In caso di default il contratto derivato perde validità.

Le diverse opzioni modificano evidentemente il valore del derivato.
Come presentato in [40], utilizzando l'approccio di Black e Cox, l'evento di default può essere modellizzato come l'attraversamento di una barriera inferiore. Sia S_t il valore di un generico sottostante, possiamo descrivere l'evento di default come il momento in cui accade che $S_t < L$, dove L è la barriera inferiore.
La Put Europea su questo sottostante viene quindi prezzata in maniera differente a seconda dei casi. Relativamente alle possibilità presentate prima, l'opzione è sostanzialmente:

1. Un'opzione Put Down-and-Out con Barriera L e rebate $R = K$, in tal caso per il possessore dell'opzione il default è lo scenario più conveniente.

2. Un'opzione Put Down-and-Out con Barriera L e rebate $R = (K - \Pi)^+$.

3. Un'opzione Put Down-and-Out con Barriera L e rebate $R = 0$.

Fino a questo punto il pricing potrebbe essere effettuato con il metodo FST standard, presentato nel capitolo precedente.

Tuttavia presentiamo questo tipo di modellistica solo dopo aver introdotto il Regime-Switching, perchè pensiamo che possa essere utilizzato per modellizzare il default in maniera dinamica. L'idea è la seguente: in ogni stato del mondo k il modello si comporta sempre nello stesso modo, con gli stessi parametri; la barriera L_k ed il recovery rate Π_k sono invece differenti per ogni stato del mondo. In questo modo riusciamo a riprodurre due caratteristiche che con un'opzione barriera classica non riusciremmo a riprodurre:

- Il recovery rate è stocastico. Infatti, in caso di default, non sappiamo esattamente quale sarà il valore di Π. Con l'utilizzo del Regime-Switching non siamo obbligati a fissarlo al suo valore atteso, possiamo invece dargli una certa varianza, facendolo oscillare tra diversi valori con probabilità differenti.

- La barriera è anch'essa stocastica. Non possiamo sapere se il default avverà nel caso in cui il titolo scenda sotto un valore oppure un altro. Con l'utilizzo del Regime-Switching siamo in grado di concedere maggiore libertà anche al livello della barriera.

Infine possiamo legare entrambe le cose: se il default avviene al superamento della barriera L_k, il recovery rate Π_k può essere fissato ad una certa percentuale della barriera. Non presentiamo l'implementazione di tale modello, in quanto è la (3.11) con B e R differenti per ogni stato del mondo.

Presentiamo due esempi in cui confrontiamo il prezzo di un'opzione Put standard con quello ottenuto tenendo conto della possibilità di default. Abbiamo considerato le tre diverse casistiche elencate precedentemente per il payoff dell'opzione (casi 1-2-3). In Figura 3.7 riportiamo i risultati ottenuti senza Regime-Switching, in Figura 3.8 quelli ottenuti con l'utilizzo del Regime-Switching, per avere Barriera e Recovery differenti nei vari stati del mondo.
Notiamo che il prezzo (presente sotto il grafico di ogni opzione) è differente se si considera la possibilità di default, inoltre può essere sia più basso che più alto del caso classico, in base alla definizione delle clausole contrattuali.

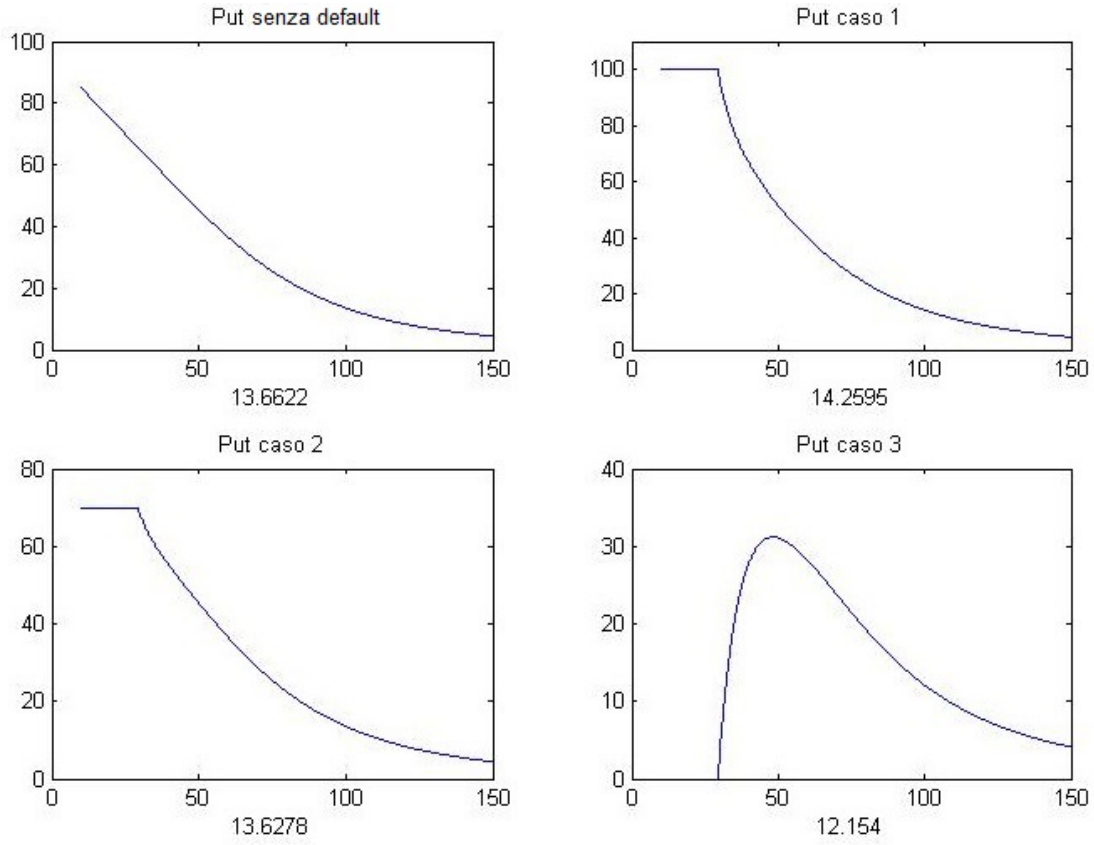

Figura 3.7: Prezzi ottenuti per un'opzione Put con modello di Merton Jump-Diffusion al variare del sottostante, parametri $(r, q, \sigma, \lambda, \mu, \delta) = (0.05, 0, 0.3, 0.8, -0.22, 0.3)$, maturity un anno, strike=100, spot=100. In tutti i casi L=30 e Π=30.

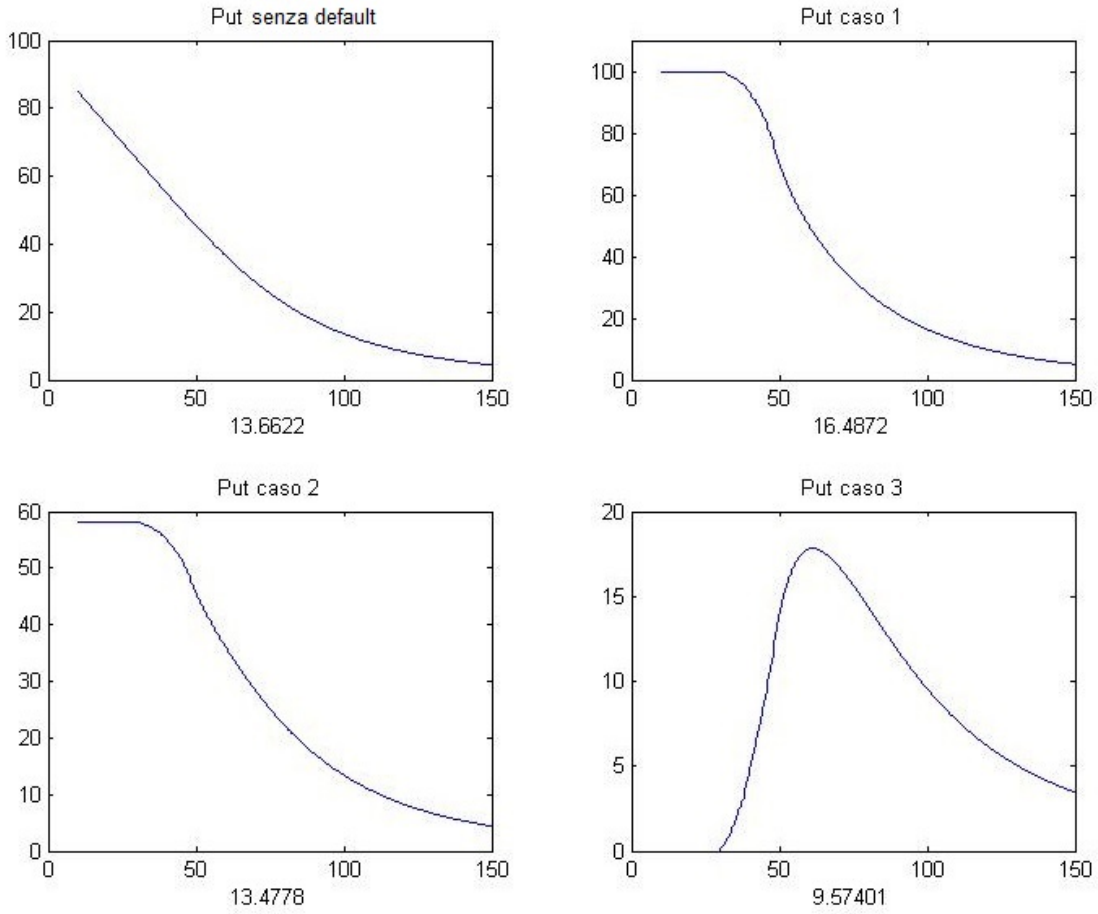

Figura 3.8: Prezzi ottenuti per un'opzione Put con modello di Merton Jump-Diffusion e Regime-Switching (media dei vari stati del mondo pesati per le rispettive probabilità) al variare del sottostante, parametri $(r, q, \sigma, \lambda, \mu, \delta) = (0.05, 0, 0.3, 0.8, -0.22, 0.3)$, maturity un anno, strike=100, spot=100. Dieci stati del mondo con barriere L=30,32,...,48 e Π=L . Probabilità P=(0.05,0.05,0.05,0.05,0.1,0.1,0.1,0.1,0.2,0.2).

Capitolo 4

Estensioni del metodo FST

4.1 Introduzione

In questo capitolo presentiamo due estensioni del metodo FST. Nella prima parte [Sezione 4.2] tratteremo l'estensione del metodo FST a tre modelli: quello Heston [Sezione 4.2.1], quello di Bates [Sezione 4.2.4] e quello di Heston-Hull-White [Sezione 4.2.5]. Lo stesso V.Surkov, nella sua tesi [50], afferma che non era stato possibile applicare il metodo ai modelli a volatilità stocastica, in particolare fa riferimento al modello di Heston. Dimostreremo che invece è possibile applicare il metodo FST anche a tale modello senza modificare l'algoritmo.
Successivamente presenteremo brevemente il metodo CONV, che non è stato fino ad ora trattato, al fine di derivare anche per quest'ultimo l'estensione al modello di Heston [Sezione 4.3]. Tale scelta è dovuta alla stretta parentela tra il metodo FST ed il metodo CONV.
Infine tratteremo l'estensione che denomineremo fFST [Sezione 4.4], cioè l'applicazione del metodo FST all'equazione Forward. Tale equazione è scritta in funzione dello strike e non dello spot, ci consente quindi di trovare il prezzo di moltissime opzioni risolvendo una sola PIDE.

4.2 FST e volatilità stocastica

Trattiamo ora il problema di pricing di opzioni Europee per due modelli a volatilità stocastica, Heston e Bates. L'estensione ad altri tipi di opzioni, come le Barriera e le Americane, segue direttamente da quanto presentato nei capitoli precedenti.

4.2.1 Il modello di Heston

Tra i modelli a volatilità stocastica il più famoso ed usato è sicuramente il modello di Heston. Questo prevede un modello lognormale per il sottostante S_t dove la volatilità non è però costante, come nel caso del modello di Black&Scholes, ma è a sua volta un processo stocastico. Sia S_t il sottostante e v_t la sua varianza, il modello di Heston è infatti rappresentato dal seguente sistema di equazioni differenziali stocastiche (SDE):

$$dS_t = rS_t dt + \sqrt{v_t} S_t dW_t^{(1)}, \tag{4.1}$$

$$dv_t = k(\theta - v_t)dt + \sigma \sqrt{v_t} dW_t^{(2)}, \tag{4.2}$$

dove $W_t^{(1)}$ e $W_t^{(2)}$ sono due moti Browniani correlati tali che: $dW_t^{(1)} dW_t^{(2)} = \rho dt$.
Il processo v_t è un processo di tipo square-root, denominato modello CIR. Come noto, questo tipo di processo non garantisce che v_t sia maggiore di zero, può avere infatti una probabilità non nulla di annullarsi. Esiste però una condizione, sufficiente ma non necessaria, per cui v_t risulti sempre positivo . Questa è conosciuta come condizione di Feller, che riportiamo qui di seguito:

$$2k\theta \geq \sigma^2 . \tag{4.3}$$

4.2.2 Funzione caratteristica nel modello di Heston

Introduciamo brevemente la derivazione della funzione caratteristica per il modello di Heston, in quanto ci sarà utile successivamente. Per la trattazione completa e dettagliata rimandiamo all'articolo originale di Heston [30].
Sia x_t il processo del logprice[1], la funzione caratteristica è definita come segue:

$$\varphi(t, \omega | x_0, v_0) = \mathbb{E}\left[e^{i\omega x_t}\right], \tag{4.4}$$

dove x_0 e v_0 sono i valori iniziali per logprice e volatilità.
Grazie al teorema di Feyman-Kac [2], il problema (4.4) può essere riscritto come una PDE bidimensionale (non dimentichiamoci infatti che x_t dipende da v_t):

$$\frac{\partial \varphi}{\partial t} + \left(r - q - \frac{v}{2}\right)\frac{\partial \varphi}{\partial x} + \frac{v}{2}\frac{\partial^2 \varphi}{\partial x^2} + \rho\sigma v \frac{\partial^2 \varphi}{\partial x \partial v} + [k(\theta - v) + \lambda]\frac{\partial \varphi}{\partial v} + \frac{v}{2}\sigma^2 \frac{\partial^2 \varphi}{\partial v^2} = 0, \tag{4.5}$$

dove λ è il premio per il rischio della volatilità. In generale $\lambda = \lambda(S, v, t)$: noi non ci occuperemo di come viene ricavato, dato che è un problema più econometrico che modellistico [6], lo tratteremo quindi come un parametro noto.
Si cerca una soluzione per l'equazione (4.5) della forma:

$$\varphi(T - t, \omega | x_t, v_t) = e^{C(T-t,\omega) + v_t D(T-t,\omega)} \cdot e^{i\omega x_t}. \tag{4.6}$$

A questo punto è sufficiente inserire la (4.6) nella (4.5) e svolgere i conti per ottenere l'espressione di C e D. Riportiamo di seguito solo il risultato, rimandando ancora una volta a [30] per una trattazione dettagliata.
Si ottiene quindi:

[1] In questo capitolo definiamo il logprice come $x_t = \ln S_t / S_0$, diversamente da quanto fatto in precedenza dove usavamo $x_t = \ln S_t$.

$$C(\tau,\omega) = i\omega(r-q)\tau + \frac{k\theta+\lambda}{\sigma^2}\left\{(k-i\omega\rho\sigma+d)\tau - 2\ln\left[\frac{1-ge^{d\tau}}{1-g}\right]\right\}, \quad (4.7)$$

$$D(\tau,\omega) = \frac{k-i\omega\rho\sigma+d}{\sigma^2}\left[\frac{1-e^{d\tau}}{1-ge^{d\tau}}\right], \quad (4.8)$$

avendo effettuato il cambio di variabile $\tau = T - t$ e posto:

$$g = \frac{k-i\omega\rho\sigma+d}{k-i\omega\rho\sigma-d}, \quad (4.9)$$

$$d = \sqrt{(i\omega\rho\sigma)^2 - \sigma^2(-i\omega-\omega^2)}. \quad (4.10)$$

È utile notare che il termine $e^{i\omega x_t}$ nella (4.6), una volta inserito nella (4.5), ha il seguente effetto sulle derivate rispetto ad x:

$$\begin{aligned}\frac{\partial\varphi}{\partial x} &= i\omega\varphi,\\ \frac{\partial^2\varphi}{\partial x^2} &= -\omega^2\varphi,\\ \frac{\partial\varphi}{\partial x\partial v} &= i\omega\frac{\partial\varphi}{\partial v}.\end{aligned}$$

Inoltre, per come abbiamo definito la trasformazione in logprice $x_t = \ln S_t/S_0$, abbiamo $x_0 = 0$. Più precisamente, se cerchiamo la funzione caratteristica del modello di Heston, che da questo punto in avanti indicheremo con $\varphi^H(T-t,\omega|x_t,v_t)$, riferendoci a t come istante iniziale, otteniamo $x_t = 0$ e quindi:

$$\varphi^H(T-t,\omega|x_t,v_t) = e^{C(T-t,\omega)+v_tD(T-t,\omega)}. \quad (4.11)$$

4.2.3 Il metodo FST applicato al modello di Heston

Passiamo ora al problema di pricing. Il prezzo di un'opzione Europea, che indichiamo con $V(t, S, v)$, soddisfa la seguente PIDE:

$$\frac{\partial V}{\partial t}+(r-q)S\frac{\partial V}{\partial S}+\frac{v}{2}S^2\frac{\partial^2 V}{\partial S^2}+\rho\sigma v S\frac{\partial^2 V}{\partial S\,\partial v}+[k(\theta-v)+\lambda]\frac{\partial V}{\partial v}+\frac{v}{2}\sigma^2\frac{\partial^2 V}{\partial v^2}-rV=0\,. \quad (4.12)$$

Come prima, applichiamo alla (4.12) il cambio di variabile in logprice $x = \ln(S/S_0)$:

$$\frac{\partial V}{\partial t}+\left(r-q-\frac{v}{2}\right)\frac{\partial V}{\partial x}+\frac{v}{2}\frac{\partial^2 V}{\partial x^2}+\rho\sigma v\frac{\partial^2 V}{\partial x\partial v}+[k(\theta-v)+\lambda]\frac{\partial V}{\partial v}+\frac{\sigma^2}{2}v\frac{\partial^2 V}{\partial v^2}-rV=0\,. \quad (4.13)$$

L'equazione (4.13) è la stessa utilizzata per ricavare la funzione caratteristica nel modello di Heston (4.5), a meno del termine $-rV$.
Possiamo ora applicare la trasformata di Fourier rispetto ad x, ottenendo quindi:

$$\frac{\partial \widehat{V}}{\partial t}+i\omega\left(r-q-\frac{v}{2}\right)\widehat{V}-\omega^2\frac{v}{2}\widehat{V}+i\omega\rho\sigma v\frac{\partial \widehat{V}}{\partial v}+[k(\theta-v)+\lambda]\frac{\partial \widehat{V}}{\partial v}+\frac{\sigma^2}{2}v\frac{\partial^2 \widehat{V}}{\partial v^2}-r\widehat{V}=0\,, \quad (4.14)$$

dove $\widehat{V}$ indica la trasformata di V.
L'applicazione della trasformata di Fourier alla (4.13), ha lo stesso effetto che il termine $e^{i\omega x}$ della funzione caratteristica (4.4) ha sulla (4.5), come fatto notare in precedenza. Infatti al posto del termine derivata prima compare $i\omega$, mentre al posto della derivata seconda compare $-\omega^2$. Partendo quindi da questa idea e seguendo la trattazione di Ingersoll [31], ipotizziamo che la soluzione della (4.14) sia del tipo $e^{\widetilde{C}(T-t)+v\widetilde{D}(T-t)}$. Inserendola nella PDE otteniamo:

$$\hat{V}\left[-\frac{\partial\widetilde{C}}{\partial\tau} - v\frac{\partial\widetilde{D}}{\partial\tau} + i\omega\left(r - q - \frac{v}{2}\right) - \omega^2\frac{v}{2} + i\omega\rho\sigma v\widetilde{D} + (a - kv)\widetilde{D} + \frac{\sigma^2}{2}v\widetilde{D}^2 - r\right] = 0\,, \tag{4.15}$$

dove $a = k\theta + \lambda$.
La (4.15) deve essere verificata per ogni valore di v, separando quindi i termini contenenti v dagli altri e ponendoli entrambi uguali a zero, otteniamo un sistema di due ODE:

$$\frac{\partial\widetilde{D}}{\partial\tau} = -i\frac{\omega}{2} - \frac{\omega^2}{2} + i\omega\rho\sigma\widetilde{D} - k\widetilde{D} + \frac{\sigma^2}{2}\widetilde{D}^2\,, \tag{4.16}$$

$$\frac{\partial\widetilde{C}}{\partial\tau} = i\omega(r - q) + a\widetilde{D} - r\,. \tag{4.17}$$

La (4.17), prima di essere risolta, necessita della soluzione della (4.16). La (4.16) ammette una soluzione esplicita, in quanto riconosciamo che si tratta di un'equazione di Riccati. Risolvendo entrambe troviamo $\widetilde{D} = D$, come definita in (4.8), e $\widetilde{C} = C - r\tau$, dove C è definita come in (4.7). Il termine extra in $\widetilde{C}$ rispetto a C è dovuto alla presenza di $-r\widehat{V}$ nella PDE (4.14).
Possiamo quindi ricavare in maniera esplicita $\widehat{V}$ in t, conoscendone il valore in T:

$$\widehat{V}(t, \omega, v) = \widehat{V}(T, \omega, v)e^{C(T-t)+vD(T-t)-r(T-t)}\,. \tag{4.18}$$

Per ottenere la soluzione nello spazio di partenza non ci resta che antitrasformare. Troviamo quindi l'equazione del metodo FST applicato al modello di Heston:

$$V(t, x, v) = \mathcal{F}^{-1}_{\omega\to x}\left[\mathcal{F}_{x\to\omega}[V](T, \omega)\varphi^H e^{-r(T-t)}\right](t, x, v)\,. \tag{4.19}$$

Infine, per validare la bontà della nostra soluzione, riportiamo alcuni risultati ottenuti con il metodo FST e il software Premia (www.rocq.inria.fr/mathfi/Premia/) con i metodi di Carr-Madan [9] (Tabella 4.1) e Attari [1] (Tabella 4.2).

	Prezzo FST	Prezzo Premia (Carr-Madan)
Put	29.661138	29.661146
Call	19.661148	19.661146

Tabella 4.1: Risultati ottenuti per il modello di Heston con parametri: $(v_0, \theta, k, \sigma, \rho) = (0.2, 0.2, 0.3, 0.4, -0.2)$, $r = q = 0$, Maturity 2 anni, Spot=100, Strike= 110, 2^{14} punti per l'FST.

	Prezzo FST	Prezzo Premia (Attari)
Put	7.437280	7.437022
Call	8.432297	8.432583

Tabella 4.2: Risultati ottenuti per il modello di Heston con parametri: $(v_0, \theta, k, \sigma, \rho) = (0.2, 0.2, 3, 0.04, -0.2)$, $r = 0.05$, $q = 0$, Maturity 0.2 anni, Spot=100, Strike= 100, 2^{14} punti per l'FST.

4.2.4 Il metodo FST applicato al modello di Bates

Dopo aver esteso l'utilizzo del metodo FST al modello di Heston, risulta quasi naturale l'estensione al modello di Bates. Questo tipo di modello non è altro che il modello a volatilità stocastica di Heston con l'aggiunta di un processo di Poisson con salti lognormali. L'equazione differenziale stocastica (SDE) bidimensionale che definisce il processo è la seguente:

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^{(1)} + dJ_t\,, \tag{4.20}$$

$$dv_t = k(\theta - v_t)dt + \sigma \sqrt{v_t} dW_t^{(2)}\,, \tag{4.21}$$

dove notiamo che la (4.20), rispetto alla (4.1), ha in più la parte dJ_t relativa ai salti. La scrittura del problema di pricing in logspot x, sotto forma di una PIDE, risulta quindi:

$$\begin{aligned}\frac{\partial V}{\partial t} + \left(r - q - \frac{v}{2}\right)\frac{\partial V}{\partial x} &+ \frac{v}{2}\frac{\partial^2 V}{\partial x^2} + \rho\sigma v\frac{\partial^2 V}{\partial x \partial v} + [k(\theta - v) + \lambda]\frac{\partial V}{\partial v} + \frac{\sigma^2}{2}v\frac{\partial^2 V}{\partial v^2} - rV + \\ &+ \int_{\mathbb{R}\setminus\{0\}} \left(V(x+y) - V(x) - (e^y - 1)\frac{\partial V}{\partial x}\right) dy = 0\,, \qquad (4.22)\end{aligned}$$

dove rispetto alla (4.13) notiamo la presenza del termine integrale.
Prima di procedere con la trasformata di Fourier analizziamo in maggiore dettaglio la modellistica utilizzata. Il modello di Bates non è il modello di Heston più il modello di Merton, si tratta di aggiungere al modello di Heston la sola parte con salti del modello di Merton, che vuol dire il termine integrale della PIDE (4.22). Per questo motivo l'esponente caratteristico della parte Jump è della forma:

$$\Psi^J(\omega) = i\omega\gamma + \lambda\left(e^{i\omega(\mu - \delta^2/2) - \delta^2\omega^2/2} - 1\right), \qquad (4.23)$$

con λ, μ e δ rispettivamente l'intensità, la media e la volatilità dei salti. È dimostrato in [13] che la funzione caratteristica del modello di Bates è data dal prodotto di quella di Heston con quella della parte Jump. Può essere quindi scritta nel seguente modo:

$$\begin{aligned}\varphi^B(T-t,\omega) &= exp\left\{C(T-t,\omega) + v_t D(T-t,\omega) + \left[i\omega\gamma + \lambda\left(e^{i\omega(\mu-\delta^2/2)-\delta^2\omega^2/2} - 1\right)\right](T-t)\right\}, \\ &= \varphi^H(T-t,\omega)e^{\Psi^J(\omega)(T-t)}\,. \qquad (4.24)\end{aligned}$$

La condizione risk-neutral (2.9) deve essere ancora valida, e lo è utilizzando il modello di Heston descritto dalle (4.1) e (4.2). Dato che la funzione caratteristica (4.24) è data da due componenti, quella di Heston che già verifica la (2.9) e quella relativa alla parte con

salti, per non modificare la condizione di risk-neutral occorre imporre che:

$$\Psi^J(-i) = 0 \iff \gamma = -\lambda\left(e^{i\omega(\mu-\delta^2/2)-\delta^2\omega^2/2} - 1\right). \tag{4.25}$$

Trasformando nello spazio di Fourier la (4.22), ritroviamo ancora una volta una PDE simile a quella relativa alla funzione caratteristica di Heston (4.14):

$$\frac{\partial\widehat{V}}{\partial t}+i\omega\left(r-q-\frac{v}{2}\right)\widehat{V}-\omega^2\frac{v}{2}\widehat{V}+i\omega\rho\sigma v\frac{\partial\widehat{V}}{\partial v}+[k(\theta-v)+\lambda]\frac{\partial\widehat{V}}{\partial v}+\frac{\sigma^2}{2}v\frac{\partial^2\widehat{V}}{\partial v^2}+\left[-r+\Psi^J(\omega)\right]\widehat{V}=0\,, \tag{4.26}$$

dove il termine $\Psi^J(\omega)$ si ottiene, come sempre, utilizzando la formula di Lévy-Khintchine. Eseguendo le stesse operazioni effettuate per il modello di Heston, cioè ipotizzando una soluzione di tipo esponenziale e risolvendo il sistema di ODE ottenuto, giungiamo a:

$$\hat{V}(t,\omega,v) = \hat{V}(T,\omega,v)\varphi^H(T-t,\omega)e^{(\Psi^J(\omega)-r)(T-t)}\,. \tag{4.27}$$

Infine, antitrasformando, ricaviamo il metodo FST per il modello di Bates:

$$V(t,x,v) \;=\; \mathcal{F}^{-1}\left[\mathcal{F}[V]\varphi^H e^{(\Psi^J-r)(T-t)}\right](t,x,v)\,. \tag{4.28}$$

Il modello di Heston, come abbiamo appena visto, è un modello bidimensionale. La risoluzione del problema di pricing tramite metodi MC o metodi variazionali risulta molto complessa, a causa della seconda dimensione. Abbiamo invece visto che il metodo FST, grazie all'utilizzo della funzione caratteristica in logprice, permette di risolvere anche in questo caso un problema monodimensionale. Tale metodo risulta quindi estremamente vantaggioso dal punto di vista computazionale.

Esempi

Riportiamo due esempi ricavati utilizzando il metodo FST con il modello di Heston (Figura 4.1) e di Bates (Figura 4.2).

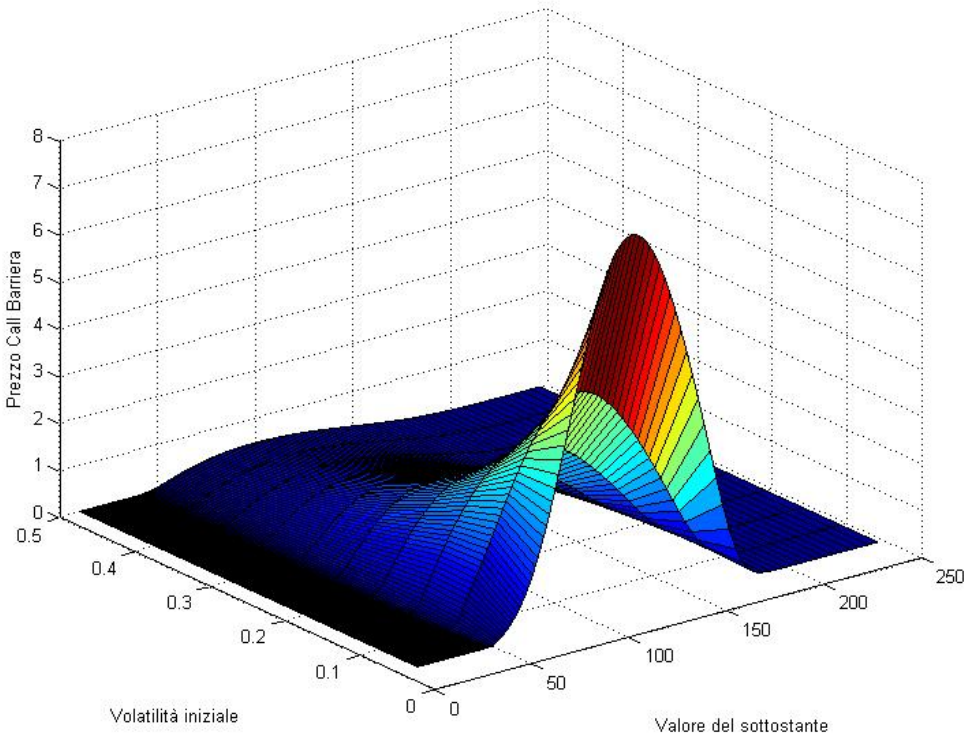

Figura 4.1: Prezzo di una opzione Call Barriera Knock-and-Out, modello di Heston. Parametri $(r, q, \theta, k, \rho, \sigma) = (0.05, 0, 0.2, 0.3, -0.2, 0.4)$, maturity 2 anni, spot=100, strike=110, barriera inferiore 50 e barriera superiore 170. Discretizzazione con 2^{10} punti spaziali.

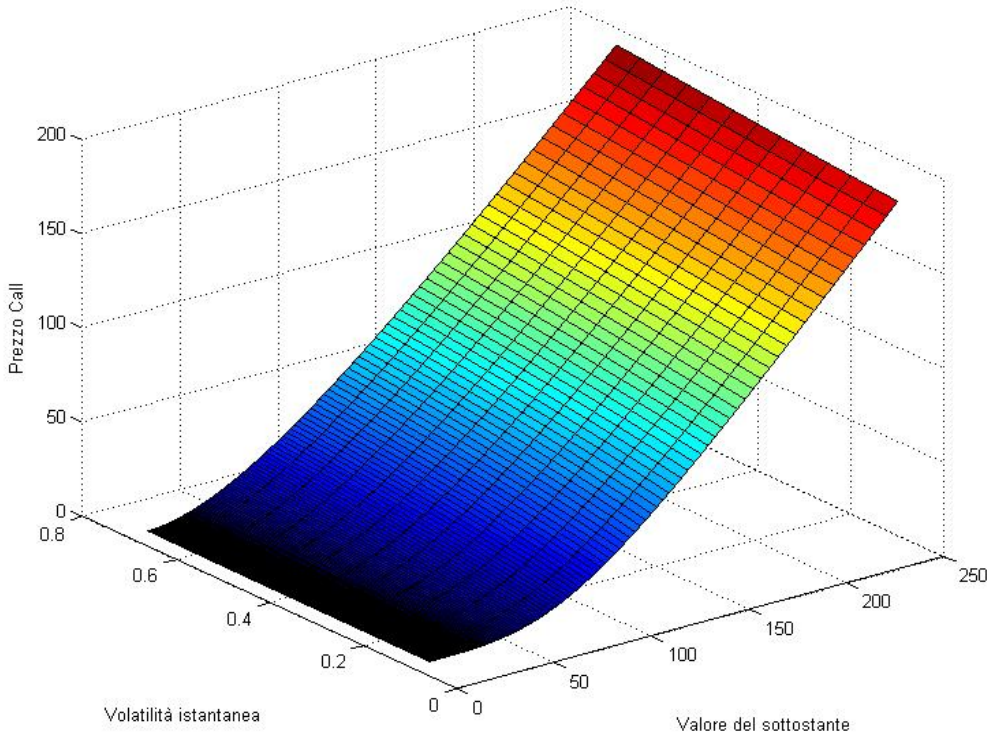

Figura 4.2: Prezzo di una opzione Call, modello di Bates. Parametri $(r, q, \theta, k, \rho, \sigma, \lambda, \mu, \delta) = (0.05, 0, 0.2, 0.3, -0.2, 0.4, 0.1, 0.2, 0.16)$, maturity 2 anni, spot=100, strike=110. Discretizzazione con 2^{10} punti spaziali.

Presentiamo nella prossima sezione un modello a volatilità e tasso d'interesse stocastici, mostrando come il metodo FST renda semplice la risoluzione del problema tridimensionale, riducendolo ad uno monodimensionale.

4.2.5 Il metodo FST applicato al modello di Heston-Hull-White

Per alcuni tipi di derivati, per esempio nel mondo FX (Foreign exchange market), l'utilizzo della volatilità stocastica, con l'approssimazione di tasso d'interesse deterministico, non riesce a riprodurre fedelmente le dinamiche di mercato. Per questo motivo, introduciamo un modello che tenga conto di entrambe le caratteristiche, fondendo il più famoso modello a volatilità stocastica (Heston) con il più famoso per la dinamica dei tassi d'interesse (Hull-White), giungendo a quello che viene chiamato modello di Heston-Hull-White. La trattazione di tale modello è abbastanza complessa, alcuni risultati che utilizzeremo richiedono dimostrazioni lunghe e complicate. Dato che lo scopo di questa tesi non è quello di trattare nel dettaglio questo modello, ma di farne vedere l'applicazione con il metodo FST, rimandiamo a [26] [27] [29] per una trattazione dettagliata. Il risultato che siamo interessati a raggiungere è il seguente: mostrare come il metodo FST riduca un complesso problema tridimensionale in un più trattabile problema monodimensionale.

Sia S_t il sottostante, v_t la sua varianza e r_t il tasso d'interesse. Il modello di Heston-Hull-White è rappresentato dal seguente sistema di equazioni differenziali stocastiche:

$$dS_t = r_t S_t dt + \sqrt{v_t} S_t dW_t^{(1)}, \tag{4.29}$$

$$dv_t = k(\theta - v_t)dt + \sigma \sqrt{v_t} dW_t^{(2)}, \tag{4.30}$$

$$dr_t = \lambda(\theta_r - r_t)dt + \eta dW_t^{(3)}, \tag{4.31}$$

dove θ_r può eventualmente essere una funzione di t, per esempio si può considerare $\theta_r(t)$ l'attuale struttura a termine dei tassi d'interesse. $W_t^{(1)}$, $W_t^{(2)}$, $W_t^{(3)}$ sono tre moti Browniani correlati tali che: $dW_t^{(1)} dW_t^{(2)} = \rho_{12} dt$, $dW_t^{(1)} dW_t^{(3)} = \rho_{13} dt$, mentre $dW_t^{(2)} dW_t^{(3)} = 0$, è infatti difficile pensare che la dinamica dei tassi d'interesse e quella della volatilità di un sottostante possano essere correlati, tale ipotesi è largamente utilizzata in letteratura, come mostrato negli articoli citati in precedenza.

Il problema di pricing deve essere riscritto in maniera leggermente differente rispetto a quanto fatto sino ad ora, infatti il fattore di sconto è stocastico, non può quindi essere estratto dal valore atteso. Sia $f(S_T)$ un generico payoff, allora il prezzo di un derivato $V(t, S_t)$ in t con scadenza T verifica la seguente uguaglianza:

$$V(t, S_t) = \mathbb{E}\left[e^{-\int_t^T r_s\, ds} f(S_T)\right]. \tag{4.32}$$

Grazie al teorema di Feynman-Kac il problema (4.32) può essere riscritto sotto forma di una PDE. Utilizzando come in precedenza il cambio di variabile in logspot, si ottiene:

$$\begin{aligned} \frac{\partial V}{\partial t} &+ \left(r - \frac{v}{2}\right)\frac{\partial V}{\partial x} + \frac{v}{2}\frac{\partial^2 V}{\partial x^2} + \rho_{12}\sigma v \frac{\partial^2 V}{\partial x \partial v} + [k(\theta - v)]\frac{\partial V}{\partial v} + \frac{\sigma^2}{2} v \frac{\partial^2 V}{\partial v^2} + \\ &+ [\lambda(\theta_r - r)]\frac{\partial V}{\partial r} + \frac{1}{2}\eta^2 r^2 \frac{\partial^2 V}{\partial r^2} + \rho_{13}\sqrt{v}\frac{\partial^2 V}{\partial x \partial r} - rV = 0. \end{aligned} \tag{4.33}$$

Allo stesso modo, è possibile ricavare la funzione caratteristica scontata φ del logspot x:

$$\varphi(T - t, \omega) = \mathbb{E}\left[e^{-\int_t^T r_s\, ds} e^{i\omega x}\right]. \tag{4.34}$$

Passando ancora una volta dal valore atteso alla PDE otteniamo:

$$\begin{aligned} \frac{\partial \varphi}{\partial t} &+ \left(r - \frac{v}{2}\right)\frac{\partial \varphi}{\partial x} + \frac{v}{2}\frac{\partial^2 \varphi}{\partial x^2} + \rho_{12}\sigma v \frac{\partial^2 \varphi}{\partial x \partial v} + [k(\theta - v)]\frac{\partial \varphi}{\partial v} + \frac{\sigma^2}{2} v \frac{\partial^2 \varphi}{\partial v^2} + \\ &+ [\lambda(\theta_r - r)]\frac{\partial \varphi}{\partial r} + \frac{1}{2}\eta^2 r^2 \frac{\partial^2 \varphi}{\partial r^2} + \rho_{13}\sqrt{v}\frac{\partial^2 \varphi}{\partial x \partial r} - r\varphi = 0. \end{aligned} \tag{4.35}$$

Purtroppo la PDE (4.35) non rientra nella classe di quelle affini, in quanto troviamo il termine $\sqrt{v}$ davanti alla derivata mista rispetto ad x ed r. Non è quindi possibile ricavare

analiticamente la funzione caratteristica di tale modello, se non nel caso particolare in cui $\rho_{13} = 0$. Per risolvere tale problema si trovano in letteratura due soluzioni: la prima è sostituire il termine $\sqrt{v(t)}$ con il suo valore atteso (giungendo al modello H1HW), la seconda prevede invece di sostituirlo con un processo Gaussiano (giungendo al modello H2HW), aumentando la dimensionalità del problema a quattro. In entrambi i casi è possibile ricavare analiticamente la funzione caratteristica scontata. Abbiamo deciso di presentare il modello H1HW, il caso con l'approssimazione gaussiana presenta una forma della funzione caratteristica ancora più complessa di quella che mostreremo, ma il metodo FST tratta il pricing in maniera analoga, riducendo il modello quadridimensionale ad uno monodimensionale.
Torniamo quindi ad occuparci della derivazione della funzione caratteristica, come mostrato in [35] vale la seguente approssimazione:

$$\begin{aligned}
\sqrt{v(t)} &\approx \mathbb{E}\left[\sqrt{v(t)}\right], \\
&\approx F(t) := \sqrt{\theta(1-e^{kt}) + v(0)e^{-kt} - \frac{\sigma^2}{8k}\frac{(-e^{-kt})(\theta e^{kt} - \theta + 2v(0))}{\theta e^{kt} - \theta + v(0)}}, \\
&\approx a + bexp(-ct),
\end{aligned}$$

dove i coefficienti a, b e c sono definiti come segue:

$$a = \sqrt{\theta - \frac{\sigma^2}{8k}}, \quad b = \sqrt{v(0)} - a, \quad c = -\log\left(\frac{F(1)-a}{b}\right).$$

Utilizzando tale approssimazione, la PDE (4.35) si riduce ad una affine, ammette quindi una soluzione analitica del tipo:

$$\varphi = exp\left\{A(u,\tau) + iux(0) + B_v(u,\tau)v(0) + B_r(u,\tau)r(0)\right\}, \tag{4.36}$$

dove abbiamo sostituito $\tau = T - t$.
Le funzioni A, B_v e B_r sono definite come segue:

$$\begin{aligned} A(u,\tau) &= \sum_{k=1}^{4} I_k\,, \\ B_v(u,\tau) &= \frac{1-e^{-d\tau}}{\sigma^2(1-ge^{-d\tau})}(k-\sigma\rho_{12}iu-d)\,, \\ B_r(u,\tau) &= \frac{iu-1}{\lambda}(1-e^{-\lambda\tau})\,, \end{aligned}$$

con $d = \sqrt{(\sigma\rho_{12}iu-k)^2-\sigma^2 iu(iu-1)}$ e $g = \frac{k-\rho_{12}iu-d}{k-\rho_{12}iu+d}$. Infine i termini I_i sono degli integrali di cui riportiamo la soluzione analitica:

$$\begin{aligned} I_1 &= \theta_r(iu-1)\left(\tau+\frac{e^{-\lambda\tau}-1}{\lambda}\right), \\ I_2 &= \frac{k\theta\tau}{\sigma^2}(k-\sigma\rho_{12}iu-d)-\frac{2}{\sigma^2}\log\left(\frac{1-ge^{-d\tau}}{1-g}\right), \\ I_3 &= \frac{\eta^2(i+u)^2}{4\lambda^3}(3+e^{-2\lambda\tau}-4e^{-\lambda\tau}-2\lambda\tau)\,, \\ I_4 &= -\frac{\eta\rho_{13}}{\lambda}(iu+u^2)\left[\frac{b}{c}(e^{-ct}-e^{-cT})+a\tau+\frac{a}{\lambda}(e^{-\lambda\tau}-1)+\frac{b}{c-\lambda}e^{-cT}\left(1-e^{-\tau(\lambda-c)}\right)\right]. \end{aligned}$$

Abbiamo quindi una forma analitica per la funzione caratteristica del modello di Heston-Hull-White approssimato (H1HW). Come per il modello di Heston, ci accorgiamo che il termine $e^{iux(t)}$ ha lo stesso effetto sulle derivate rispetto ad x della trasformata di Fourier, e grazie alla scelta del logprice $x = \log(S/S_0)$ tale termine scompare dalla funzione caratteristica.
Appplicando la trasformata di Fourier rispetto ad x all'equazione (4.33) e risolvendo la PDE risultante, come abbiamo fatto nel caso di Heston e Bates, troviamo:

$$\mathcal{F}[V](t,\omega,v,r) = \mathcal{F}[V](T,\omega,v,r)\varphi(T-t,\omega,v,r)\,. \tag{4.37}$$

Ancora una volta, antitrasformando, ricaviamo la formula risolutiva tramite il metodo FST:

$$V(t,x,v,r) = \mathcal{F}^{-1}\left[\mathcal{F}[V](T,\omega,v,r)\varphi(T-t,\omega,v,r)\right](t,x,v,r)\,. \tag{4.38}$$

Notiamo che nella formula (4.38) manca il termine $e^{-r(T-t)}$, in quanto il tasso d'interesse è stocastico, il discount-factor compare quindi all'interno del valore atteso e viene inglobato all'interno di φ, che come detto in precedenza è la funzione caratteristica **scontata**.

Il metodo FST applicato al modello di Heston-Hull-White (modello approssimato H1HW), o ad un banalissimo Black and Scholes, risulta altrettanto semplice e veloce. Produrre un pricer MC per tale modello risulterebbe invece estremamente costoso, sarebbe infatti necessario simulare 3 diversi processi stocastici. Per la risoluzione della PDE con metodi variazionali invece, la scrittura di un codice a differenze o elementi finiti 3D risulta non impossibile, ma quantomeno complicata e poco intuitiva.

Riportiamo in Tabella 4.3 un esempio di pricing per alcune opzioni Europee, utilizzando il modello H1HW.
In Tabella 4.4 riportiamo un confronto tra il modello di Heston e l'H1HW, con volatilità del tasso d'interesse nulla. Come ci aspettiamo ritroviamo lo stesso prezzo, a meno di errori numerici.
Nelle Tabelle 4.5 e 4.6 presentiamo il confronto tra Heston e H1HW rispettivamente per opzioni Call e Put. Notiamo che l'aleatorietà sul tasso d'interesse impatta poco per scadenze brevi, ma non è trascurabile sulle lunghe scadenze.

Strike	Call	Put
80	23.4282	1.8454
90	15.1927	3.4120
100	8.3279	6.3494
110	4.0590	11.8826
120	2.1464	19.7722

Tabella 4.3: Risultati ottenuti per il pricing di opzioni Europee al variare dello strike, modello H1HW. Parametri parte Heston $(k, \theta, \sigma, v_0, \rho_{12}) = (0.3, 0.05, 0.6, 0.05, -0.3)$, parametri parte Hull-White $(\lambda, \theta_r, \eta, r_0, \rho_{13}) = (0.01, 0.02, 0.01, 0.02, 0.2)$, spotprice=100, maturity un anno. Discretizzazione con 2^{18} punti spaziali.

	H1HW	Heston
Call	6.824	6.820
Put	12.507	12.507

Tabella 4.4: Confronto tra i valori ottenuti con il modello H1HW ed Heston, ponendo a zero la parte stocastica sul tasso d'interesse. Come notiamo, a meno di errori numerici, i valori coincidono. Parametri parte Heston $(k, \theta, \sigma, v_0, \rho_{12}) = (0.3, 0.05, 0.6, 0.05, -0.3)$, maturity 2 anni, spot=100, strike=110. Discretizzazione con 2^{18} punti spaziali.

T	Heston	H1HW
0.1	10.5960	10.6024
0.5	13.5261	13.6654
1	15.9289	16.4872
1.5	17.8127	19.1097
2	19.4662	21.7286
2.5	21.0568	24.4327

Tabella 4.5: Confronto tra i valori ottenuti con il modello H1HW ed Heston, Call Europea al variare della maturity. Parametri parte Heston $(k, \theta, \sigma, v_0, \rho_{12}) = (0.3, 0.05, 0.6, 0.05, -0.3)$, parametri parte Hull-White $(\lambda, \theta_r, \eta, r_0, \rho_{13}) = (0.01, 0.02, 0.15, 0.02, 0.2)$, spot=110, strike=100. Discretizzazione con 2^{18} punti spaziali.

T	Heston	H1HW
0.1	0.3962	0.4030
0.5	2.5311	2.7166
1	3.9488	4.8726
1.5	4.8572	7.3764
2	5.5404	10.6830
2.5	6.0973	15.0632

Tabella 4.6: Confronto tra i valori ottenuti con il modello H1HW ed Heston, Put Europea al variare della maturity. Parametri parte Heston $(k, \theta, \sigma, v_0, \rho_{12}) = (0.3, 0.05, 0.6, 0.05, -0.3)$, parametri parte Hull-White $(\lambda, \theta_r, \eta, r_0, \rho_{13}) = (0.01, 0.02, 0.15, 0.02, 0.2)$, spot=110, strike=100. Discretizzazione con 2^{18} punti spaziali.

4.3 Il metodo CONV e la sua estensione al modello di Heston

Nello stesso periodo in cui V.Surkov sviluppava il metodo FST [33][34][50][51], in modo indipendente R.Lord sviluppava il metodo CONV [37][38][39] con l'aiuto di Fang e Oosterlee. Questi svilupperanno successivamente un ulteriore metodo basato sull'utilizzo della FFT, il metodo COS [18][20], che può essere visto come un'evoluzione del CONV. Il metodo COS può essere facilmente utilizzato per prezzare opzioni che utilizzano come sottostante il modello di Heston [19], cosa che non è invece così ovvia per il metodo CONV.

L'idea di proporre l'estensione al modello di Heston anche per questo metodo, oltre che per l'FST, è dovuta al fatto che CONV e FST sono molto simili. Entrambi giungono infatti ad una formula di pricing che è sostanzialmente la stessa, partendo da un approccio di tipo probabilistico per il CONV, rispetto a quello PIDE dell'FST. Essendo la stessa formula di pricing, derivata in due modi differenti, deve quindi essere applicabile agli stessi modelli.

4.3.1 Metodo CONV

Il metodo della Convoluzione, detto CONV, si basa su un approccio probabilistico. Infatti, al posto che vedere il problema di pricing sotto forma di una PIDE, parte dalla definizione di prezzo di un'opzione come valore atteso condizionato del payoff. Tratteremo rapidamente la derivazione del metodo, per maggiori dettagli rimandiamo agli articoli citati in precedenza, dato che il nostro scopo è principalmente spiegare perchè può essere esteso al modello a volatilità stocastica di Heston (di conseguenza anche a Bates) e soprattutto la relazione con il metodo FST.
Sia $V(t, x)$ il prezzo di un'opzione Europea in t con logprice x_t, scadenza in T e payoff $\varphi(x_T)$. Allora vale la seguente uguaglianza:

$$\begin{aligned} V(t,x) &= e^{-r(T-t)}\mathbb{E}\left[\varphi(x_T)|x_t\right], && (4.39)\\ &= e^{-r(T-t)}\int_{\mathbb{R}}\varphi(y)f(y|x)\,dy, && (4.40)\end{aligned}$$

dove $f(y|x)$ indica la densità di probabilità del sottostante a scadenza $y = x_T$, condizionata al valore $x_t = x$ all'istante iniziale t.
Il metodo si basa sull'ipotesi che la densità sia riscrivibile in questa forma:

$$f(y|x) = f(y - x), \tag{4.41}$$

cioè che f non dipenda dai valori assoluti di x e y, ma solo dalla differenza $y - x$. Questo è certamente vero per processi ad incrementi indipendenti come gli exponential-Lévy. Riscrivendo sotto l'ipotesi (4.41) l'integrale (4.40), ed applicando il cambio di variabile $z = y - x$ otteniamo:

$$V(t,x) = e^{-r(T-t)}\int_{\mathbb{R}}\varphi(z+x)f(z)\,dz. \tag{4.42}$$

dove notiamo che l'integrale (4.42) non è altro che una convoluzione. Possiamo moltipilicare V per un fattore $e^{\alpha x}$, al fine di ottenere una funzione che appartenga ad $\mathcal{L}^1(\mathbb{R})$, $v = Ve^{\alpha x}$. Applicando quindi a v l'anti-trasformata di Fourier rispetto ad x, ed utilizzando la proprietà (2.3) otteniamo:

$$\begin{aligned}
\mathcal{F}^{-1}\left[v(t,x)\right](\omega) &= \frac{1}{2\pi}e^{-r(T-t)}\int_{\mathbb{R}} e^{-i\omega x}e^{\alpha x}\left\{\int_{\mathbb{R}}\varphi(z+x)f(z)\,dz\right\}dx\,, \\
&= \frac{1}{2\pi}e^{-r(T-t)}\mathcal{F}^{-1}\left[e^{\alpha(x+z)}\varphi(x+z)\right](\omega)\int_{\mathbb{R}} e^{-i(\omega-i\alpha)z}f(z)\,dz\,, \quad (4.43) \\
&= \frac{1}{2\pi}e^{-r(T-t)}\mathcal{F}^{-1}\left[e^{\alpha y}\varphi(y)\right](\omega)\,\Phi(-(\omega-i\alpha))\,, \quad (4.44)
\end{aligned}$$

dove per passare dalla (4.43) alla (4.44) abbiamo utilizzato la definizione di funzione caratteristica (4.4), che qui definiamo con la lettera Φ.
Possiamo ora applicare la trasformata di Fourier e moltiplicare v per il termine $e^{-\alpha x}$, al fine di ottenere il valore dell'opzione V nello spazio di partenza. Giungiamo quindi alla formulazione del metodo CONV:

$$V(t,x) = e^{-r(T-t)}e^{-\alpha x}\mathcal{F}\left[\mathcal{F}^{-1}\left[e^{\alpha y}\varphi(y)\right](\omega)\,\Phi(i\alpha-\omega)\right]\,, \quad (4.45)$$

che come possiamo notare, a meno del parametro di damping α e dell'ordine della trasformata/antitrasformata di Fourier, è sostanzialmente il metodo FST. Tralasciamo l'implementazione numerica, maggiori dettagli sono disponibili in [35], oltre che negli articoli originali.

4.3.2 Applicazione al modello di Heston

Come fatto in (4.39), anche per il modello di Heston possiamo scrivere il prezzo di un'opzione Europea come valore atteso condizionato. Prendiamo ad esempio il prezzo

di un'opzione Call $C(t, x, v)$:

$$
\begin{aligned}
C(t,x) &= e^{-r(T-t)}\mathbb{E}\left[\varphi(x_T)|x_t, v_t\right] , && (4.46)\\
&= e^{-r(T-t)}\int_{\mathbb{R}^+}\int_{\mathbb{R}} \varphi(x_T) f(x_T, v_T|x_t, v_t)\, dx_T\, dv_T , && (4.47)
\end{aligned}
$$

dove notiamo l'aumento di dimensionalità del problema rispetto a (4.39), dovuto alla volatilità stocastica.
Se la densità f soddisfacesse l'ipotesi:

$$
f(x_T, v_T|x_t, v_t) = f(x_T - x_t, v_T|v_t)\,, \tag{4.48}
$$

potremmo applicare il metodo CONV come mostreremo tra poco.
Per semplificare la trattazione poniamo a zero il tasso r e il fattore α. Poniamo $y = x_T$, $x = x_t$ e $z = y - x$, otteniamo:

$$
C(t,x) = \int_{\mathbb{R}^+}\int_{\mathbb{R}} \varphi(x+z) f(z, v_T|v_t)\, dz\, dv_T\ . \tag{4.49}
$$

Applichiamo l'anti-trasformata di Fourier rispetto ad x:

$$
\begin{aligned}
\mathcal{F}^{-1}\left[C(t,x)\right] &= \mathcal{F}^{-1}\left[\int_{\mathbb{R}^+}\int_{\mathbb{R}} \varphi(x+z) f(z, v_T|v_t)\, dz\, dv_T\right] , && (4.50)\\
&= \int_{\mathbb{R}^+} \mathcal{F}^{-1}\left[\int_{\mathbb{R}} \varphi(x+z) f(z, v_T|v_t)\, dz\right] dv_T , && (4.51)\\
&= \int_{\mathbb{R}^+} \left\{\mathcal{F}^{-1}\left[\varphi(y)\right](\omega)\frac{1}{2\pi}\int_{\mathbb{R}} e^{-i\omega z} f(z, v_T|v_t)\, dz\right\} dv_T , && (4.52)\\
&= \frac{1}{2\pi}\mathcal{F}^{-1}\left[\varphi(y)\right](\omega)\int_{\mathbb{R}^+}\int_{\mathbb{R}} e^{-i\omega z} f(z, v_T|v_t)\, dz\, dv_T , && (4.53)\\
&= \frac{1}{2\pi}\mathcal{F}^{-1}\left[\varphi(y)\right](\omega)\Phi(-\omega) , && (4.54)
\end{aligned}
$$

Giungiamo quindi alla formula (4.54) che non è altro che la (4.44). Applicando infine la trasformata di Fourier otteniamo la formula del metodo CONV (4.45), provando quindi che vale anche per il modello di Heston, a patto che valga la condizione (4.48), come dimostreremo qui di seguito.

4.3.3 Densità di probabilità nel modello di Heston

La condizione (4.48) compare in un articolo di Guardasoni e Sanfelici [28] (in realtà per il modello di Bates, ma è equivalente per Heston) senza alcuna dimostrazione. In realtà l'ipotesi (4.48) può essere sostituita da un'altra molto simile, che ci permette comunque di derivare il metodo CONV per il modello di Heston. Sia $f(x, v|x_0, v_0)$ la densità di probabilità del modello di Heston, condizionata ai valori iniziali x_0 e v_0, definiamo la densità marginale di x in questo modo:

$$\widetilde{f}(x|x_0, v_0) = \int_{\mathbb{R}^+} f(x, v|x_0, v_0)\, dv\,. \tag{4.55}$$

È sufficiente che $\widetilde{f}(x|x_0, v_0) = \widetilde{f}(x - x_0|v_0)$ per poter applicare il metodo CONV al modello di Heston. Infatti nelle equazioni da (4.50) a (4.53), il payoff $\varphi(x + z)$ non dipende in alcun modo da v_T, si può quindi portare all'interno l'integrale in dv_T e ottenere:

$$\begin{aligned}
\mathcal{F}^{-1}\left[C(t, x)\right] &= \mathcal{F}^{-1}\left[\int_{\mathbb{R}^+}\int_{\mathbb{R}} \varphi(x + z) f(z, v_T|v_t)\, dz\, dv_T\right]\,, && (4.56)\\
&= \mathcal{F}^{-1}\left[\int_{\mathbb{R}} \varphi(x + z) \int_{\mathbb{R}^+} f(z, v_T|v_t) dv_T\, dz\right]\,, && (4.57)\\
&= \mathcal{F}^{-1}\left[\int_{\mathbb{R}} \varphi(x + z)\widetilde{f}(z, |v_t)\, dz\right]\,, && (4.58)\\
&= \frac{1}{2\pi}\mathcal{F}^{-1}\left[\varphi(y)\right](\omega) \int_{\mathbb{R}} e^{-i\omega z}\widetilde{f}(z|v_t)\, dz\,, && (4.59)\\
&= \frac{1}{2\pi}\mathcal{F}^{-1}\left[\varphi(y)\right](\omega)\Phi(-\omega)\,. && (4.60)
\end{aligned}$$

È possibile ricavare una formula analitica per $\widetilde{f}$, vedi [17], non è però facilmente trattabile. A noi interessa solo mostrare che $\widetilde{f}(x|x_0, v_0) = \widetilde{f}(x - x_0|v_0)$, procederemo quindi senza utilizzare l'espressione analitica della densità marginale.
Sappiamo che la funzione caratteristica non è altro che la trasformata di Fourier della densità marginale rispetto ad x:

$$\begin{aligned}
\varphi(\omega) &= \mathbb{E}\left[e^{i\omega x}\right], && (4.61)\\
&= \int_{\mathbb{R}^+}\int_{\mathbb{R}} e^{i\omega x} f(x, v|x_0, v_0)\, dx\, dv, && (4.62)\\
&= \int_{\mathbb{R}} e^{i\omega x} \widetilde{f}(x|x_0, v_0)\, dx, && (4.63)\\
&= \mathcal{F}\left[\widetilde{f}(x|x_0, v_0)\right]. && (4.64)
\end{aligned}$$

Grazie all'articolo di Heston [30] sappiamo che la forma della funzione caratteristica è del tipo:

$$\varphi^H(\omega) = e^{g(T-t,v_0,\omega)} e^{i\omega x_0}, \tag{4.65}$$

come riportato all'inizio di questo capitolo.
Partendo da questa forma, possiamo antitrasformare la (4.64), ottenendo quindi:

$$\begin{aligned}
\widetilde{f}(x|x_0, v_0) &= \mathcal{F}^{-1}\left[\mathcal{F}\left[\widetilde{f}(x|x_0, v_0)\right]\right], && (4.66)\\
&= \mathcal{F}^{-1}\left[\varphi^H\right], && (4.67)\\
&= \mathcal{F}^{-1}\left[e^{g(T-t,v_0,\omega)} e^{i\omega x_0}\right], && (4.68)\\
&= \frac{1}{2\pi}\int_{\mathbb{R}} e^{-i\omega x} e^{i\omega x_0} e^{g(T-t,v_0,\omega)}, && (4.69)\\
& && (4.70)
\end{aligned}$$

$$= \frac{1}{2\pi}\int_{\mathbb{R}} e^{-i\omega(x-x_0)} e^{g(T-t,v_0,\omega)}\,, \tag{4.71}$$

$$= \mathcal{F}^{-1}\left[e^{g(T-t,v_0,\omega)}\right](x-x_0)\,, \tag{4.72}$$

che pertanto è funzione di $x - x_0$, cioè $\widetilde{f}(x|x_0, v_0) = \widetilde{f}(x - x_0|v_0)$.

Confronto CONV-FST

Riportiamo in Tabella 4.7 i prezzi ottenuti per due opzioni, con modello di Heston, al fine di confrontare CONV e FST.

	Prezzo FST	Prezzo CONV
Call	23.55268	23.55270
Put	23.08479	23.08476

Tabella 4.7: Prezzi per due opzioni Europee ottenuti con il modello di Heston. Parametri $(r, q, \theta, k, \rho, \sigma) = (0.05, 0, 0.2, 0.3, -0.2, 0.4)$, maturity 2 anni, spot=100, strike=110.

4.4 FST e PIDE Forward

La PIDE che abbiamo risolto nei capitoli precedenti, con l'utilizzo del metodo FST classico (Sezione 2.4), consente di ottenere i prezzi di una generica opzione Europea al variare del prezzo sottostante. Questo è molto utile per avere una visione di come lo spotprice influenzi il prezzo dell'opzione, ci consente di prezzare anche opzioni più complesse di quelle Europee. Nel mercato però lo spot è unico, mentre a parità di tutti gli altri parametri, sono presenti opzioni con strike differenti. Ci piacerebbe poter prezzare in una sola volta tutte le opzioni Europee al variare dello strike, come avviene per esempio applicando l'algoritmo di pricing di Carr-Madan. Questa possibilità sarebbe particolarmente utile dal punto di vista della calibrazione del modello. Calibrare vuol dire trovare quei parametri che rendono i prezzi ottenuti dal modello maggiormente simili

a quelli di mercato; per questo motivo gli strumenti su cui si calibra devono essere i più liquidi possibili, al fine di ottenere dei parametri corretti, senza introdurre errori dovuti al mispricing spesso presente nel mercato per strumenti illiquidi. Dato che in questa tesi ci occupiamo principalmente di derivati su Equity (non che il modello non possa essere utilizzato per altri tipi di sottostante), gli strumenti più liquidi in assoluto sono le opzioni Europee plain-vanilla. Quando si calibra bisogna ricorrere ad un algoritmo di minimizzazione, che richiede di calcolare i prezzi delle opzioni un gran numero di volte. La possibilità di calcolare con un solo algoritmo FST il prezzo di tante opzioni aventi la stessa maturity e diversi strike, invece che eseguire un pricing differente per ogni strike, velocizzerebbe notevolmente il processo di calibrazione.
Presentiamo ora alcuni passaggi della derivazione dell'equazione Forward ricavata da Dupire per un modello lognormale [15] [24], al fine di dare almeno un'idea di come quest'equazione nasca. La derivazione dell'equivalente per i processi con salti segue sostanzialmente lo stesso procedimento, alcune complicazioni sono però presenti. Rimandando quindi agli articoli [3] e [4] per la dimostrazione formale, riporteremo solo il risultato ottenuto.

Un sottostante lognormale evolve secondo la seguente SDE:

$$dS_t = \mu S_t dt + \sigma S_t dW_t\,, \tag{4.73}$$

dove W_t è un processi di Wiener.
Sia $C(t, S_t)$ il prezzo di un'opzione Call Europea con strike K e scadenza T, a meno del fattore di sconto, possiamo scrivere il prezzo di tale opzione come:

$$C(t, S_t, K) = \mathbb{E}[(S_T - K)^+ | S_t] = \int_{\mathbb{R}} (x - K)^+ f(x, T|S_t)\, dx\,, \tag{4.74}$$

dove $f(x, T|S_t)$ è la distribuzione di $x = S_T$ condizionata al valore iniziale S_t.
Derivando la (4.74) rispetto a K rispettivamente una e due volte otteniamo:

$$\frac{\partial C}{\partial K} = -\int_K^\infty f(x, T|S_t)\, dx\,, \tag{4.75}$$

$$\frac{\partial^2 C}{\partial K^2} = f(K, T|S_t)\,. \tag{4.76}$$

È inoltre noto che un processo del tipo (4.73) soddisfa l'equazione di Fokker-Planck:

$$\frac{\partial f}{\partial t} = -\frac{\partial}{\partial S}(\mu S f(S,t)) + \frac{1}{2}\frac{\partial^2}{\partial S^2}(\sigma^2 S^2 f(S,t))\,. \tag{4.77}$$

Derivando ora rispetto a T l'equazione (4.74) e sfruttando l'equazione (4.77) in $t = T$, otteniamo:

$$\frac{\partial C}{\partial T} = -\int_K^\infty (x-K)\left\{\frac{\partial}{\partial T} f(x, T|S_t)\right\} dx\,, \tag{4.78}$$

$$= -\int_K^\infty (x-K)\left\{-\frac{\partial}{\partial x}(\mu x f(x,T)) + \frac{1}{2}\frac{\partial^2}{\partial x^2}(\sigma^2 x^2 f(,T))\right\} dx\,. \tag{4.79}$$

Integrando per parti due volte ed utilizzando le relazioni (4.75) e (4.76), giungiamo all'equazione Forward di Dupire:

$$\frac{\partial C}{\partial T} = \frac{1}{2}\sigma^2 K^2 \frac{\partial^2 C}{\partial K^2} + (r-q)\left(C - K\frac{\partial C}{\partial K}\right) - rC\,, \tag{4.80}$$

dove il tasso risk free r ed il tasso di dividendo continuo q sono stati reinseriti, anche se per semplicità erano stati omessi durante la dimostrazione. Per una trattazione dettagliata su come giungere all'equazione (4.80) rimandiamo a [47].

Mostriamo ora, senza dimostrarlo, il risultato ottenuto in [4] sull'equazione Forward per processi con salti. Sia S_t un processo di Lévy definito come in (2.7), il prezzo $V(t, S_t)$ di una generica opzione Europea con scadenza T e strike K soddifa la seguente equazione

forward:

$$\begin{aligned}\frac{\partial V}{\partial T} = & - (r-q)K\frac{\partial V}{\partial K} + \frac{1}{2}\sigma^2 K^2 \frac{\partial^2 V}{\partial K^2} - qV + \\ & + \int_{\mathbb{R}} \nu(dz)\, e^z \left[V(T, Ke^{-z}) - V(T,K) - K(e^{-z}-1)\frac{\partial V}{\partial K} \right]. \end{aligned} \tag{4.81}$$

L'equazione (4.81) è molto simile alla PIDE Backward (2.10) ma necessita ancora di qualche trasformazione per poter applicare il metodo FST. Applichiamo il cambio di variabile $k = \log K$ e otteniamo:

$$\begin{aligned}\frac{\partial V}{\partial T} = & - \left(r - q + \frac{\sigma^2}{2}\right)\frac{\partial V}{\partial k} + \frac{1}{2}\sigma^2 \frac{\partial^2 V}{\partial k^2} - qV + \\ & + \int_{\mathbb{R}} \nu(dz)\, e^z \left[V(T, k-z) - V(T,k) - (e^{-z}-1)\frac{\partial V}{\partial k} \right]. \end{aligned} \tag{4.82}$$

Applichiamo ora la trasformata di Fourier rispetto a k:

$$\frac{\partial \widehat{V}}{\partial T}(\omega) = \widehat{V}(\omega) \left\{ -i\omega \left(r - q + \frac{\sigma^2}{2} \right) - \omega^2 \frac{\sigma^2}{2} - q + \int_{\mathbb{R}} \nu(dz)\, e^{(1-i\omega)z} - e^z + i\omega(e^z - 1) \right\}. \tag{4.83}$$

Applichiamo il cambio di variabile $i\epsilon = (1 - i\omega)$, dove ω è il corrispondente di x nello spazio delle frequenze. Sostituiamo all'interno della (4.83) e otteniamo:

$$\frac{\partial \widehat{V}}{\partial T}(\omega) = \widehat{V}(\omega) \left\{ i\epsilon \left(r - q - \frac{\sigma^2}{2} \right) - \epsilon^2 \frac{\sigma^2}{2} - r + \int_{\mathbb{R}} \nu(dz)\, e^{i\epsilon z} - 1 - i\epsilon(e^z - 1) \right\}. \tag{4.84}$$

Ricordando la formula dell'esponente caratteristico (2.16), otteniamo:

$$\frac{\partial \hat{V}}{\partial T}(\omega) = \hat{V}(\omega)\,(\Psi(\epsilon) - r)\ . \tag{4.85}$$

Riportiamo infine la (4.85) nella sola variabile ω:

$$\frac{\partial \hat{V}}{\partial T}(\omega) = \hat{V}(\omega)\,[\Psi(-\omega - i) - r]\ . \tag{4.86}$$

Risolvendo la ODE (4.86) giungiamo alla soluzione continua del metodo fFST:

$$V(T, x) = \mathcal{F}^{-1}\left[\mathcal{F}[V](t, \omega)\, e^{(\Psi(-\omega - i) - r)\,(T - t)}\right](x)\,. \tag{4.87}$$

Essendo l'equazione (4.87) in avanti, necessita di una condizione iniziale. Nel caso di un'opzione Call avremo quindi $V(0, x) = (S_0 - K)^+$, mentre per un'opzione di tipo Put $V(0, x) = (K - S_0)^+$.
Per quanto riguarda l'implementazione numerica, valgono tutte le considerazioni fatte nella Sezione 2.5.
Giungiamo quindi al seguente algoritmo:

$$v_m = \text{IFFT}[\,\text{FFT}[v_{m-1}] \cdot e^{(\Psi(-\omega - i) - r)(t_m - t_{m-1})}\,]\,. \tag{4.88}$$

4.4.1 Confronto con il metodo FST

Come detto in precedenza, il metodo fFST ci consente di ottenere in una sola volta il prezzo di molte opzioni, al variare dello strike K. La griglia K non coincide però con i valori degli strike che vogliamo osservare, è quindi necessario interpolare i valori sulla griglia trovata (noi utilizzeremo un'interpolazione di tipo Spline). Mostriamo in Tabella 4.8 i risultati ottenuti per il pricing di una serie di opzioni con differenti strike, con un solo utilizzo dell'algoritmo fFST e con i diversi pricing necessari utilizzando l'algoritmo

FST. In Tabella 4.9 mostriamo i tempi di calcolo necessari per prezzare le opzioni della Tabella 4.8.

Stike	Prezzo FST	Prezzo fFST	errore$\cdot 10^{-8}$
30	81.4661522489	81.4661522489	0.0011823431
40	71.9852593939	71.9852593939	0.0047293724
50	62.6414842878	62.6414842876	0.0197935889
60	53.6810760867	53.6810760866	0.0149391610
70	45.4280967809	45.4280967808	0.0112798659
80	38.1443870837	38.1443870822	0.1551185846
90	31.9418322195	31.9418322185	0.1030240781
100	26.7888441096	26.7888441096	0.0016171952
110	22.5659494345	22.5659494346	0.0045204728
120	19.1216250838	19.1216250825	0.1361851076
130	16.3082177983	16.3082177945	0.3831694073
140	13.9983348566	13.9983348548	0.1876685473
150	12.0887888717	12.0887888712	0.0541668043
160	10.4985660384	10.4985660361	0.2332960491
170	9.1648132194	9.1648132178	0.1693175377
180	8.0387756811	8.0387756807	0.0446220838
190	7.0823924245	7.0823924225	0.2065692505
200	6.2656693852	6.2656693844	0.0822415913

Tabella 4.8: Risultati ottenuti per 18 Call Europee con modello di Merton jump-diffusion, $T = 1$, 2^{20} punti per ogni FST e per la fFST.

	FST	fFST
tempo (sec.)	98.280978	5.643039

Tabella 4.9: Confronto dei tempi di calcolo tra FST e fFST.

Notiamo che il tempo di calcolo necessario per prezzare diverse opzioni Europee, con la stessa maturity e strike differenti, è nettamente più veloce effettuando una singola fFST rispetto al pricing con un numero di FST pari al numero di opzioni. I tempi sono complessivi delle interpolazioni necessarie, sia sulla griglia K che nel caso della FST sulla griglia S. I prezzi sono accuratissimi anche utilizzando una sola fFST e

un'interpolazione spline, come si può notare dall'ordine di grandezza della differenza tra i prezzi.

4.4.2 Esercizio di calibrazione

Il problema della calibrazione è di per se molto complesso, consiste nel risolvere il seguente problema:

$$\min_{\mathbf{x}} \sum_{i} (V^i_{mkt} - V^i(\mathbf{x}))^2 \,, \tag{4.89}$$

dove V^i_{mkt} è il prezzo di mercato dell'opzione i-esima e $V^i(\mathbf{x})$ il prezzo dell'opzione i-esima calcolato con il modello da calibrare, per un vettore di parametri $\mathbf{x}$. In generale il termine quadratico viene moltiplicato per dei pesi w_i, questo principalmente per due motivi: il primo è che alcuni dati potrebbero essere più attendibili di altri, il secondo è che i prezzi generati dal nostro modello sono sostanzialmente appartenenti ad una forma parametrica. Non è detto quindi che il mercato segua tale forma, per questo motivo un fitting perfetto su tutti i dati è pressochè impossibile. L'utilizzo dei pesi consente quindi di selezionare i dati per i quali siamo interessati ad un fitting più preciso.
Non intendiamo affrontare il problema di calibrazione nel suo complesso, argomento sul quale si potrebbe scrivere un'intera tesi, ma vogliamo solamente mettere a confronto il metodo fFST con l'FST standard. Per questo motivo sfrutteremo dei dati simulati, che ci consentono di utilizzare pesi costanti pari ad uno, in quanto tutti i dati sono ugualmente attendibili ed è possibile replicare in maniera esatta la forma funzionale dei prezzi.

Presentiamo ora i risultati ottenuti per la calibrazione del modello Variance-Gamma, i dati di mercato sono stati generati utilizzando come parametri del modello $(r, q, \sigma, \theta, k) = (0.05, 0.01, 0.19071, -0.28113, 0.49083)$, per 25 opzioni Call con scadenza 6 mesi, spot pari a 100 e strike K=40,45,...,160.
La calibrazione è stata effettuata supponendo incogniti σ, θ e k, prendendo come dato

iniziale il vettore $\mathbf{x}_0 = (0.15071, -0.32113, 0.45083)$.
Abbiamo effettuato tre calibrazioni:

- La prima con il metodo FST, senza preoccuparci di far cadere il valore dello spot S_0 sulla griglia, dovendo quindi interpolare dopo ogni FST.
- La seconda sempre con il metodo FST, facendo però attenzione a far coincidere S_0 con un punto della griglia, non dovendo quindi interpolare.
- La terza con il metodo fFST, calcolando quindi una sola volta l'fFST per tutti i prezzi, ma dovendo necessariamente interpolare i valori trovati.

I risultati ottenuti sono riportati in Tabella 4.10.

	FST con interp.	FST senza interp.	fFST
Iterazioni	18	18	18
σ	0.190710	0.190710	0.190710
θ	-0.281130	-0.281130	-0.281130
k	0.490829	0.490829	0.490830
tempo (sec.)	55.833082	18.632448	2.136334

Tabella 4.10: Confronto della calibrazione effettuata con FST e fFST. A tale scopo abbiamo utilizzato la funzione *fminunc* di Matlab.

Notiamo che il metodo fFST risulta molto più performante rispetto all'FST classico. Rendendolo quindi ideale per la calibrazione. Come descritto in [13], esistono metodi di pricing più veloci di quelli basati sulla FFT per il calcolo di una singola opzione, ma per calcolare tante opzioni con strike differenti e stessa maturity i metodi FFT sono i più performanti. Proponiamo quindi tale metodo in quanto, rispetto a quello di Carr-Madan, non richiede l'espressione analitica del payoff nello spazio delle frequenze, risulta quindi molto più flessibile.

Capitolo 5

Calcolo su GPU

I risultati presentati sino ad ora sono stati prodotti utilizzando il software MATLAB. Il calcolo delle opzioni di tipo Europeo risulta particolarmente rapido, non richiede quindi nessun tipo di ulteriore speedup, anche perchè i comandi FFT e IFFT di Matlab vengono già eseguiti in modalità multi-thread. Per il pricing di opzioni Americane e Barriera i tempi di esecuzione risultano maggiori, infatti ogni step temporale corrisponde alla valutazione di un'opzione Europea; sarebbe quindi utile poter utilizzare le nuove architetture multi-core per velocizzare l'esecuzione del codice. È vero che gli step temporali devono essere eseguiti in maniera seriale, se si riuscisse però ad aumentare considerevolmente l'esecuzione di una singola FFT, diciamo per esempio a renderla dieci volte più veloce, il risultato sarebbe quello di rendere approssimativamente dieci volte più veloce l'algoritmo nel suo complesso.
Per questo motivo abbiamo deciso di implementare alcuni codici in grado di essere eseguiti su GPU (graphic processing unit), nello specifico alcuni codici scritti in C/C++ che sfruttano le estensioni CUDA.
A causa della notevole differenza di prestazioni tra linguaggio interpretato (Matlab) e compilato (C/C++), l'implementazione con CUDA verrà confrontata con un codice seriale C/C++. Rimandiamo a [36] e [53] per una trattazione estensiva sulla programmazione in C/C++.
Un'implementazione molto efficiente della FFT seriale è reperibile attraverso la libreria

FFTW [23]. Per l'implementazione parallela su GPU utilizzeremo invece la libreria cuFFT [44], che consente di utilizzare in maniera ottimale l'hardware CUDA[1].

5.1 Introduzione su CUDA

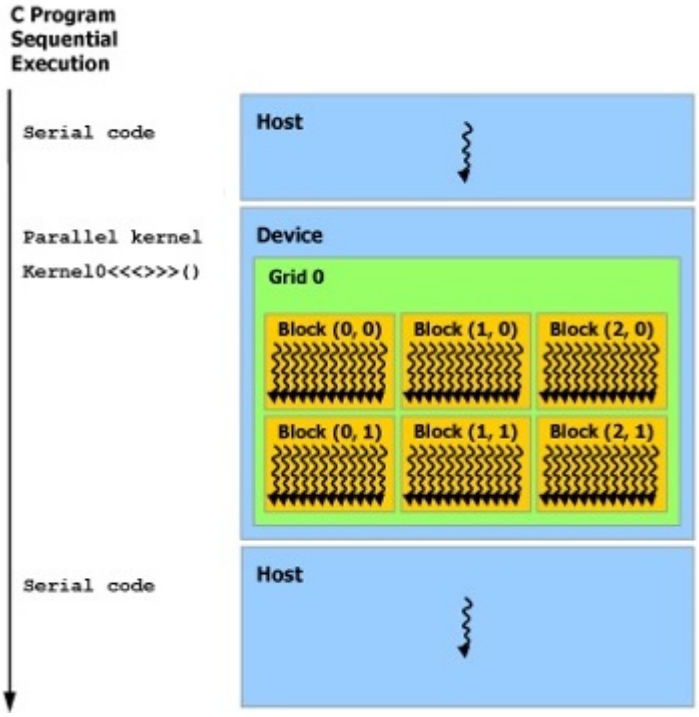

Figura 5.1: Esecuzione di codice ibrido. L'esecuzione di un programma C/C++ avviene in modo sequenziale, le operazioni vengono eseguite dall'Host fino a che non si incontra la richiesta di un'operazione parallela da eseguire su GPU (*parallel kernel*). A a questo punto l'host alloca sul device i dati necessari, il device esegue le operazioni richieste e al termine comunica il risultato all'host che, una volta recuperati i dati, continua l'esecuzione della parte seriale del codice.

CUDA è un'architettura hardware per l'elaborazione parallela creata da NVIDIA [43], tale architettura permette di ottenere aumenti di prestazioni di computing sfruttan-

[1]Inoltre se è necessario eseguire più FFT della stessa dimensione, la modalità BATCH permette di sfruttare alcune ottimizzazioni ulteriori.

do l'elaborazione dei dati su GPU. L'ambiente di sviluppo CUDA, disponibile come estensione dei linguaggi più diffusi (C/C++, Java, Pyhton, Matlab e altri), consente lo sviluppo di codici basati sul paradigma del *co-processing*, cioè un'implementazione ibrida CPU-GPU, vedi Figura 5.1. Una CPU è dotata solitamente di un numero di core non molto elevato, nella maggior parte dei casi non superano le quattro unità, anche se esistono CPU che dispongono di 16 core. Una GPU anche di fascia bassa (nel nostro caso una GT540M con 96 CUDA core) possiede un numero di core decisamente più elevato, mentre quelle disegnate appositamente per il calcolo scientifico possiedono un numero di core che può superare tranquillamente le 2000 unità. I core di una CPU sono pensati per gestire operazioni di complessità molto variabile, possiedono inoltre frequenze di clock molto maggiori rispetto a quelli di una GPU; sono quindi pochi core velocissimi adatti ad eseguire operazioni complesse. Il calcolo su GPU è pensato in maniera differente, tantissimi core eseguono parallelamente operazioni molto semplici, riducendo al massimo i tempi di latency (Figura 5.2).

Inoltre l'architettura hardware CUDA può essere utilizzata non solo con le librerie

Figura 5.2: Composizione hardware di CPU e GPU. Una CPU è composta da pochi core (in questo caso indicati come *ALU*-Arithmetic Logic Unit) che condividono memoria cache e vengono coordinate da un'unità di controllo. La GPU invece possiede molte più unità aritmetiche-logiche, suddivise in gruppi. Ognuno di questi gruppi condivide una memoria cache e un'unità di controllo. I core di ogni gruppo sono quelli che vengono eseguiti fisicamente in parallelo. Vedremo più avanti che ogni gruppo corrisponde sostanzialmente a quello che chiameremo *warp*.

specifiche di NVIDIA, ma anche per eseguire programmi scritti con librerie basate su OpenCL. Presentiamo ora, in maniera sintetica, alcune informazioni necessarie per sviluppare dei codici con estensioni CUDA.

5.2 Programmazione CUDA

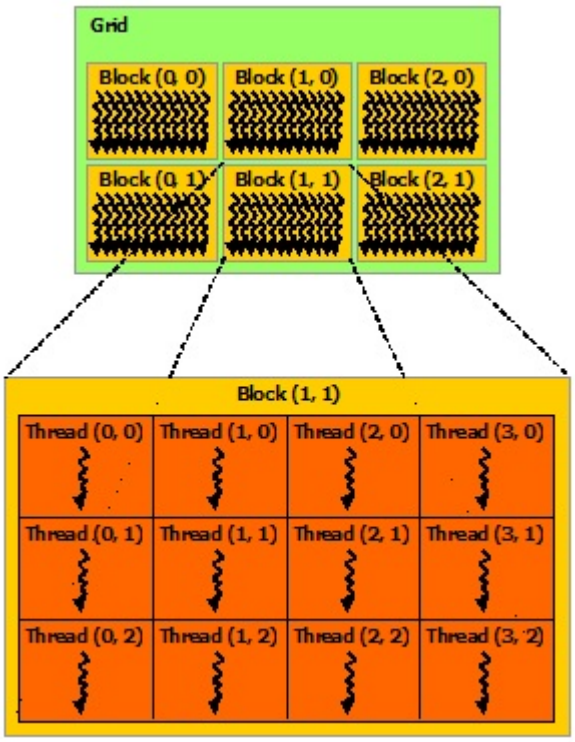

Figura 5.3: Organizzazione dei thread.

Come detto in precedenza, l'idea si basa sul paradigma del *co-processing*, quindi della scrittura di codici ibridi. Nel nostro caso, un codice C eseguito sulla CPU (Host), effettua alcune chiamate alla GPU (Device) che, una volta eseguiti i calcoli necessari, restituisce il risultato all'Host. Tali chiamate sono molto semplici da eseguire per le funzioni già implementate nelle librerie di NVIDIA, la scrittura da parte dell'utente di una funzione eseguibile su GPU richiede invece maggiore accortezza, soprattutto a causa della complessa gestione della memoria. Per quanto ci riguarda, abbiamo dovuto imple-

mentare le funzioni per il calcolo delle operazioni element-wise tra numeri complessi, e le funzioni che impongono le condizioni al bordo per opzioni path-dependent.
L'unità fondamentale nel calcolo su GPU è il thread, che esegue le singole operazioni. Questi sono organizzati in blocchi, a loro volta disposti all'interno di una griglia, come mostrato in Figura 5.3. All'interno di ogni blocco, gruppi da 32 thread formano un *warp*: questi sono i thread che vengono fisicamente eseguiti in parallelo (per le schede più datate viene eseguito in parallelo un *half-warp* di 16 thread). Per ulteriori informazioni si vedano [45] e [46].

5.2.1 Gestione della memoria

La struttura organizzativa dei thread, che abbiamo mostrato nel paragrafo precedente, prevede una gestione della memoria complessa, che deve essere utilizzata al meglio per ottenere le massime performance. La memoria attraverso cui comunicano Host e Device è chiamata *Global Memory*: è il tipo di memoria più lento ma è accessibile ad ogni thread, da Host e da Device. Lo spazio nella *Global Memory* viene allocato dall'Host, il lifetime dei dati è pari all'esecuzione dell'intero programma.
Tutti i thread all'interno di un blocco possono invece accedere alla *Shared Memory*: tale memoria consente degli accessi molto più veloci, non è però visibile all'esterno del blocco e il suo lifetime corrisponde all'esecuzione del blocco stesso, dopodiche viene deallocata in maniera automatica. La *Shared Memory* è posizionata fisicamente *on-chip*.
Infine ogni thread ha a disposizione una propria memoria non condivisa, molto piccola e velocissima, chiamata *Register Memory*: tale memoria è posizionata fisicamente *on-chip*. Qui vengono salvate le variabili dichiarate nello scope del singolo thread, devono però essere variabili dalla dimensione molto contenuta. Gli array dichiarati all'interno dello scope del singolo thread vengono invece allocati in una memoria fittizia, detta *Local Memory*. Tale memoria è sostanzialmente una parte della *Global Memory*, che viene temporaneamente riservata all'uso del singolo thread, risulta quindi più lenta di *Shared* e *Register Memory*. *Register* e *Local Memory* hanno come lifetime l'esecuzione del singolo thread, vengono subito deallocate a fine chiamata. In Figura 5.4 presentiamo

schematicamente la suddivisione della memoria, per maggiori dettagli si veda [46].

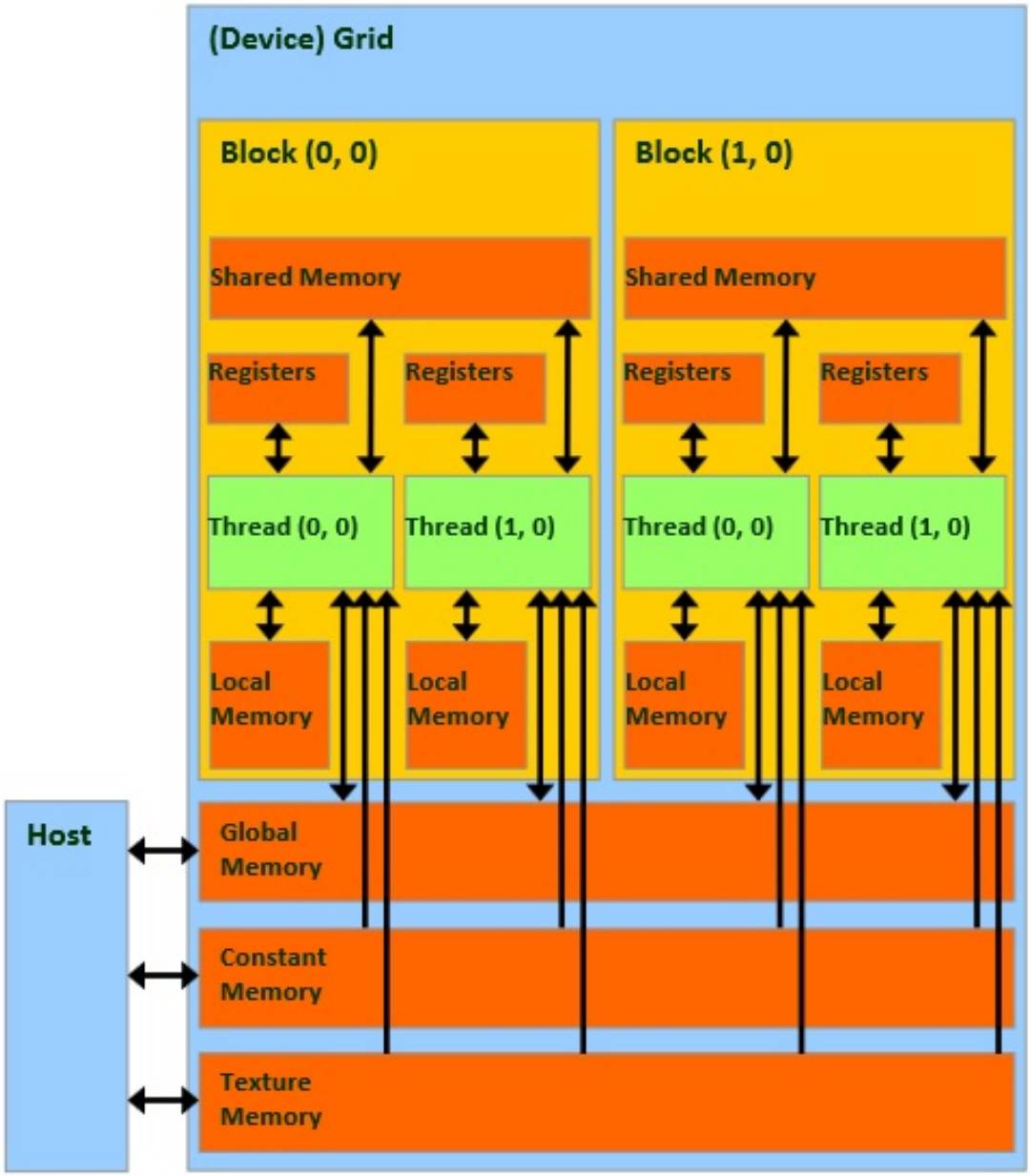

Figura 5.4: Composizione della memoria di una GPU.

L'accesso da parte dei thread a *Global* e *Shared Memory* può essere ottimizzata avendo grande accortezza nella scrittura delle funzioni. La trattazione di tale tecnica di programmazione esula dallo scopo di questa tesi, rimandiamo quindi a [45] per maggiori dettagli.

5.2.2 Chiamate alle funzioni

L'esecuzione di una funzione richiede che venga stanziato un certo numero di thread, ed organizzato in blocchi all'interno di una griglia. La porzione di codice che esegue delle operazioni su GPU con una singola chiamata, occupandosi sia della gestione dei thread che dell'esecuzione delle funzioni, viene chiamata *Kernel*. La chiamata di un *Kernel* è identificata dalla scrittura <<< , >>> prima dei parametri da passare alla funzione, dove nei due spazi vuoti vanno inserite le dimensioni della griglia e dei singoli blocchi.
Per far si che l'esecuzione del *Kernel* sia efficiente, il numero di thread stanziati deve essere coerente con quelli effettivamente utilizzati dalla GPU. La creazione di un elevato numero di thread inutilizzati rallenta in maniera sostanziale l'esecuzione del programma, come riportato in Tabella 5.1. La ripartizione dei thread all'interno dei blocchi, e dei blocchi all'interno della griglia può essere fatta a sua volta in maniera ottimale. L'intenzione non è certo quella di effettuare una trattazione esaustiva di questo tipo di ottimizzazioni, che ancora una volta esulano dallo scopo di questa tesi, ma di presentare alcuni problemi che abbiamo incontrato, e parzialmente risolto, durante la stesura dei codici. Per maggiori dettagli rimandiamo a [45].
Le chiamate alla GPU possono essere sincrone o asincrone. Nel primo caso l'esecuzione del programma viene bloccata sino a che la GPU non finisce le operazioni richieste. Nel secondo caso invece, una volta chiamata una funzione da eseguire sulla GPU, il programma seriale può continuare mentre la GPU finisce di eseguire le operazioni richieste. L'utilizzo di funzioni asincrone deve essere gestito con attenzione, potrebbe accadere di leggere un dato da una cella dove si pensa che si trovi il valore corretto, mentre questo non è ancora stato calcolato. È possibile inserire dei blocchi nel codice

seriale, che interrompono l'esecuzione fino a che la GPU non ha finito di eseguire le operazioni richieste, tramite la funzione *cudaThreadSynchronize()*.

Thread stanziati	Thread utilizzati	tempo (ms)	tempo ottimale (ms)
2^{22}	2^{6}	132.251	2.505
2^{22}	2^{7}	132.421	2.766
2^{22}	2^{8}	132.805	3.000
2^{22}	2^{9}	133.112	3.283
2^{22}	2^{10}	133.667	3.875
2^{22}	2^{11}	134.380	4.579
2^{22}	2^{12}	136.846	6.986
2^{22}	2^{13}	142.933	13.331
2^{22}	2^{14}	154.872	25.478
2^{22}	2^{15}	177.533	48.600
2^{22}	2^{16}	226.481	98.586
2^{22}	2^{17}	331.657	205.832
2^{22}	2^{18}	559.703	437.931
2^{22}	2^{19}	1016.287	902.713
2^{22}	2^{20}	2003.863	1906.459
2^{22}	2^{21}	3933.664	3868.692
2^{22}	2^{22}	8138.758	8138.454

Tabella 5.1: Confronto dei tempi di esecuzione ottenuti stanziando un numero fisso di thread (colonna tempo), anche se superiore a quelli necessari, rispetto alla configurazione ottimale, con il numero di thread stanziati pari a quelli necessari (colonna tempo ottimale). I risultati sono relativi al pricing di due opzioni Barriera con 64 step temporali in modalità BATCH (Call e Put).

5.2.3 FST in CUDA

Mostriamo schematicamente come avviene l'implementazione del metodo FST su GPU, riportando per chiarezza alcune parti di codice. Per i codici completi rimandiamo all'Appendice C
Come prima cosa viene creato un codice C/C++, che serve come base per il nostro programma, da cui si chiameranno poi le funzioni in grado di eseguire alcuni calcoli su GPU, nell'ordine:

1. Oltre alla dichiarazione delle variabili, vengono definiti i parametri del modello, l'esponente caratteristico e il vettore contenente la condizione finale V_M, il tutto nel codice seriale.

2. Viene allocato sulla GPU lo spazio per contenere i dati necessari:
   *cudaMalloc((void **)&CallDevice, sizeof(cufftDoubleReal)*(Dimension));*

3. Si procede quindi alla copia dei dati dalla RAM alla memoria della GPU:
   cudaMemcpy(CallDevice,CallHost,sizeof(cufftDoubleReal)(Dimension), cudaMemcpyHostToDevice);*

4. Vengono creati i *cufftPlan*, cioè gli oggetti che si occupano di gestire la FFT su GPU. Contengono le informazioni sul tipo di dati, il verso della trasformazione ed altre informazioni:
   cufftPlan1d(&forward, N , CUFFT_D2Z, 1);

5. Viene calcolato il prezzo dell'opzione V_0, eseguendo tutte le operazioni necessarie sulla scheda video. In particolare oltre alle singole FFT, anche le moltiplicazioni vettoriali, matriciali e l'imposizione dei vincoli ad ogni step temporale. Infatti tutte le operazioni devono essere eseguite senza spostare continuamente i dati tra GPU e RAM, poichè come vedremo, questo tipo di operazioni è inefficiente.
   cufftExecD2Z(forward,CallDevice,FourierCallDevice);
   Cproduct<<<blocchi,thread>>>(..parametri funzione..);
   cufftExecZ2D(backward,FourierCallDevice,CallDevice);

6. Si attende che la GPU finisca di eseguire le operazioni richieste, inserendo un blocco nel codice:
   cudaThreadSynchronize();

7. Il vettore dei prezzi V_0 viene copiato dalla scheda video alla RAM.
   cudaMemcpy(CallHost,CallDevice,sizeof(cufftDoubleReal)(Dimension), cudaMemcpyDeviceToHost);*

8. Si distruggono le variabili CUDA, in quanto vengono gestite in maniera C classica e necessitano di essere deallocate manualmente:
   cufftDestroy(forward);
   cudaFree(CallDevice);

In Appendice B e C riportiamo alcuni codici C/C++ e CUDA utilizzati nello sviluppo della tesi. Si può notare che le chiamate alla GPU vengono eseguite all'interno del codice seriale C/C++.

5.3 Risultati

Come osservato in precedenza, i vari step temporali devono essere eseguiti in maniera seriale, ciò che può essere parallelizzato è l'esecuzione della singola FFT. Per questo motivo siamo interessati a vedere come si comportano le due implementazioni all'aumentare della dimensione della trasformata e non del numero di step temporali.
Riportiamo in Tabella 5.2 alcuni risultati ottenuti con il codice seriale e il codice parallelo; i codici sono stati eseguiti su un PC dotato di processore Intel Core i7 da 2.2Ghz, mentre la scheda video utilizzata è una Nvidia GT540M da 2GB con 96 CUDA core. All'interno delle tabelle indicheremo con FFTW i codici eseguiti senza l'utilizzo della GPU, con cuFFT quelli eseguiti in parallelo su GPU.

Osserviamo che il calcolo su GPU risulta molto più rapido del corrispettivo seriale per tutti i vettori considerati, questo anche in presenza di una scheda video non molto potente. Le moderne schede video con supporto CUDA, sviluppate per il calcolo scientifico, sono dotate di un numero di core che può arrivare fino a 2880, con una velocità di

Dimensione	FFTW (ms)	cuFFT (ms)	Passaggio dati (ms)	Performance GPU
2^{10}	7.466	3.157	0.039	2.34
2^{11}	13.949	3.662	0.047	3.76
2^{12}	28.495	5.536	0.054	5.10
2^{13}	58.903	7.911	0.102	7.35
2^{14}	128.815	13.482	0.168	9.44
2^{15}	240.927	25.481	0.268	9.36
2^{16}	487.594	50.481	0.488	9.57
2^{17}	1027.134	103.808	0.943	9.81
2^{18}	2179.747	220.029	2.157	9.81
2^{19}	5117.748	451.606	4.718	11.22
2^{20}	10954.624	951.036	9.003	11.41
2^{21}	23122.075	1929.426	17.217	11.88

Tabella 5.2: Confronto delle performance ottenute con FFTW seriale e cuFFT parallela, Call Barriera con 64 step temporali. La performance GPU è calcolata come tempo di esecuzione del pricing tramite FFTW fratto tempo totale per il pricing con cuFFT, comprensivo del passaggio di dati.

calcolo e di trasferimento dati molto maggiore della GPU che abbiamo utilizzato per i test. L'aumento di prestazioni da noi ottenuto è quindi molto inferiore alle potenzialità che il calcolo su GPU ci offre.
Per quanto riguarda il trasferimento dei dati, non sembra impattare molto sulle performance. Questo è vero perchè eseguiamo, dopo aver passato i dati, 128 FFT su GPU (64 in avanti e 64 all'indietro) oltre alle moltiplicazioni e alle imposizioni dei vincoli sulla barriera. Se volessimo utilizzare la GPU per eseguire il pricing di un'opzione Europea (solo due FFT), che non richiede time-stepping, il tempo di trasferimento dati non sarebbe più trascurabile, vedi Tabella 5.3.

L'utilizzo del calcolo su GPU non impatta in alcun modo sulla precisione, infatti le moderne schede video hanno il pieno supporto sia delle operazioni *floating point* che *double precision*. Tutte le prove eseguite in questa tesi sono state effettuate in *double precision*, riportiamo un esempio in Tabella 5.4.

Dimensione	FFTW (ms)	cuFFT tot (ms)	cuFFT (ms)
2^{11}	0.173	0.117	0.067
2^{12}	0.291	0.160	0.093
2^{13}	0.604	0.242	0.132
2^{14}	1.141	0.399	0.208
2^{15}	2.362	0.690	0.386
2^{16}	5.150	1.383	0.767
2^{17}	10.743	2.669	1.559
2^{18}	21.996	5.422	3.302
2^{19}	53.383	11.404	6.761

Tabella 5.3: Confronto delle performance ottenute con FFTW seriale e cuFFT parallela, Call Europea. La colonna cuFFT contiene i tempi di esecuzione non comprensivi del passaggio dati tra Host e Device, la colonna cuFFT tot riporta invece il tempo complessivo di calcolo e passaggio dei dati. Notiamo che, in questo caso, il passaggio dei dati impatta in maniera evidente sui tempi di esecuzione, che rimangono comunque inferiori a quelli seriali.

Spot	FFTW	cuFFT
100	5.042297085	5.042297085
104	6.997406378	6.997406378
108	9.331449794	9.331449794
112	12.008835646	12.008835646
116	14.980512118	14.980512118

Tabella 5.4: Confronto dei prezzi ottenuti con FFTW seriale e cuFFT parallela, Call Barriera Knock-and-Out con 64 step temporali. Modello di Merton con parametri: $(r, q, \sigma, \lambda, \mu, \delta) = (0.05, 0.0, 0.15, 0.1, -0.22, 0.4)$, maturity un anno, strike 110, barriere a 20 e 300.

Presentiamo infine un esempio di pricing per un'opzione Barriera 2D, vedi Tabella 5.5. A tal proposito, non avendo ancora introdotto il problema di pricing per un'opzione con più sottostanti, riportiamo la PDE multidimensionale corrispondente.
Sia $\mathbf{X}(t)$ un processo di Lévy multidimensionale con tripletta caratteristica $(\gamma, \mathbf{\Sigma}, \nu)$, dove γ è il vettore dei drift, $\mathbf{\Sigma}$ è la matrice di varianza-covarianza della parte diffusiva e ν è la densità di Lévy multidimensionale. Utilizzando come modello per il sottostante un processo di tipo exponential-Lévy, il prezzo spot risulta essere:

$$\mathbf{S}(t) = \mathbf{S}(0)e^{\mathbf{X}(t)}\,,$$

dove il vettore $\mathbf{S}(t)$ è composto dai diversi sottostanti: $S_1(t) \dots S_n(t)$.
Il prezzo $V(t, \mathbf{x})$ di un'opzione Europea scritta su $\mathbf{S}(t)$ risolve la seguente PIDE:

$$\begin{cases} (\partial_t + \mathcal{L} - r)V(t, \mathbf{x}) = 0\,, \\ V(T, \mathbf{x}) = f(T)\,, \end{cases}$$

con l'operatore $\mathcal{L}$ definito nella maniera seguente:

$$\mathcal{L}f(\mathbf{x}) = \left(\gamma \cdot \partial_{\mathbf{x}} + \frac{1}{2}\partial_{\mathbf{x}} \cdot \Sigma \cdot \partial_{\mathbf{x}}\right) f(\mathbf{x}) + \int_{\mathbb{R}^n \setminus \{\mathbf{0}\}} \left(f(\mathbf{x}+\mathbf{y}) - f(\mathbf{x}) - \mathbf{1}_{|\mathbf{y}|<1}\mathbf{y} \cdot \partial_{\mathbf{x}} f(\mathbf{x})\right) \nu(d\mathbf{y})\,.$$

Come fatto in precedenza, abbiamo applicato il cambio di variabile in logspot $\mathbf{x}$.
Definiamo inoltre il payoff $f(T)$ di un'opzione Europea 2D in questo modo:

$$\begin{aligned} f(T) &= (S_1(T) + S_2(T) - K)^+ \qquad \text{per un'opzione Call}\,, \\ f(T) &= (K - S_1(T) + S_2(T))^+ \qquad \text{per un'opzione Put}\,. \end{aligned}$$

N	FFTW(ms)	cuFFT(ms)	FFTW(prezzo)	cuFFT(prezzo)	Performance GPU
2^{10}	6674.553	1256.999	79.580	79.580	5.31
2^{11}	28606.215	5095.416	79.952	79.952	5.61
2^{12}	137761.697	21913.547	80.183	80.183	6.29

Tabella 5.5: Confronto dei prezzi e dei tempi ottenuti, al variare della griglia spaziale con N punti, con FFTW seriale e cuFFT parallela, Call Barriera 2D Knock-and-Out con 64 step temporali. Modello di Merton con parametri: $(r, q, \sigma_1, \lambda_1, \mu_1, \delta_1, \sigma_2, \lambda_2, \mu_2, \delta_2, \rho) = (0.05, 0.0, 0.15, 0.1, -0.22, 0.4, 0.20, 0.15, -0.42, 0.3, 0.5)$, maturity un anno, strike 110, barriere a 50 e 150, $S_1 = S_2 = 100$. I tempi cuFFT sono comprensivi del passaggio di dati, la performance è definita come in precedenza.

Conclusioni

Giunti alla fine di questa trattazione possiamo riassumere ciò che è stato fatto ed i risultati raggiunti.
Il metodo FST è applicabile ad un'ampia classe di modelli: gli exponential-Lévy. L'utilizzo del Regime-Switching ne aumenta inoltre la flessibilità. L'estensione apportata, per i modelli a volatilità stocastica di Heston, Bates ed Heston-Hull-White, lo rendono incredibilmente versatile, almeno per i prodotti di tipo equity.
L'utilizzo di tale metodo ci consente di superare i seguenti problemi: quelli relativi al termine integro-differenziale, presente negli approcci PIDE con metodi variazionali, l'inefficienza computazionale dei metodi MC, e la scarsa applicabilità a payoff generici dei metodi classici basati sulla trasformata di Fourier.
La risoluzione analitica nella variabile temporale lo rende estremamente efficace per quelle opzioni che non sono fortemente path-dependent (plain-vanilla o pochi monitoraggi discreti). Ma anche nel caso di opzioni fortemente path-dependent, il numero di valutazioni temporali è inferiore, o al limite uguale, a quello degli altri metodi.
Il pricing di opzioni con esercizio anticipato viene svolto in maniera estremamente semplice e precisa. Inoltre, grazie all'utilizzo dell'estrapolazione di Richardson, l'estensione all'esercizio continuo non richiede un grande sforzo computazionale.
Il metodo risulta estremamente preciso nel confronto con i metodi più utilizzati in letteratura, con ottime proprietà di convergenza e stabilità.
Nel metodo FST gli step temporali devono essere eseguiti in maniera sequenziale, come qualsiasi metodo che preveda l'utilizzo di una discretizzazione nel tempo; ma grazie alla natura dell'FFT il codice risulta comunque facilmente parallelizzabile. Abbiamo fornito

alcune implementazioni di codici su GPU, con performance che arrivano ad essere anche 11 volte superiori a quelle di un'implementazione seriale.
Infine, ma non per importanza, ricordiamo l'utilizzo del metodo fFST per l'equazione Forward. La possibilità di ottenere una grande quantità di prezzi, al variare dello strike, lo rende estremamente efficace per la calibrazione dei modelli. Il metodo che in letteratura è riconosciuto essere il più rapido, per la calibrazione su opzioni Europee, è il metodo di Carr-Madan [13]. Ad ogni valutazione tale metodo esegue una FFT e un'interpolazione, l'fFST necessita di due FFT e un'interpolazione. Rispetto al metodo di Carr-Madan, l'fFST consente l'utilizzo di payoff molto più generici; basti pensare che per le opzioni Put il metodo di Carr-Madan richiede una riscrittura analitica, quindi un'implementazione diversa, rispetto alle opzioni Call.

Possibili sviluppi futuri

Ci sono ancora alcune cose su cui, secondo noi, sarebbe interessante lavorare:

- L'estensione del metodo ad una classe più ampia di opzioni. Altre persone, che come noi si sono interessate allo sviluppo di questo metodo, hanno utilizzato l'FST per il pricing di opzioni di vario genere, come abbiamo mostrato in questa tesi; ma data la grande varietà di prodotti presenti sul mercato, probabilmente è possibile estendere la metodologia ad altri derivati.

- Lo sviluppo dell'fFST. L'equazione Forward che abbiamo presentato è utilizzabile solo per opzioni non path-dependent, infatti la discretizzazione in strike al posto che in spotprice non rende possibile l'imposizione delle condizioni sul bordo. Per esempio, non è possibile imporre la condizione che il derivato valga zero se il sottostante supera la barriera. Ma oltre alle opzioni Call e Put, sono presenti molti prodotti che dipendono dal valore del sottostante solo a scadenza.

- Non dovrebbe essere difficile utilizzare il metodo per risolvere l'equazione di Kolmogorov Forward (Fokker-Plank). Noi abbiamo utilizzato quest'ultima per

derivare l'equazione di Dupire, ma in alcune applicazioni può essere molto comodo avere a disposizione la densità di transizione del processo.

- L'estensione al modello di Heston apre moltissime strade. Riteniamo che il metodo FST sia estendibile a tutti i modelli che ammettono una forma analitica per la funzione caratteristica. Sia il prezzo di un'opzione che la funzione caratteristica del modello non sono altro che un valore atteso rispetto alla densità di transizione; grazie al teorema di Feynman-Kac soddisfano quindi la stessa PDE (o PIDE) con differenti condizioni al bordo. Seguendo la trattazione svolta per il modello di Heston si potrebbe provare ad estenderlo ad una classe di modelli affini.

- Grazie alla considerazione precedente e alla trattazione originale di V.Surkov, dove viene presentata l'estensione ai modelli mean-reverting, si aprono buone possibilità di utilizzo nel mondo tassi d'interesse.

- Infine si può pensare ad uno sviluppo di tipo numerico. L'aumento di dimensionalità del problema, per esempio per opzioni Basket, richiede uno sforzo computazionale non indifferente. A tale proposito si potrebbero sfruttare delle implementazioni multicore, come abbiamo presentato nella tesi, magari proprio ottimizzando alcuni codici eseguibili su GPU.

Appendice A

Codici MATLAB

Presentiamo alcuni codici Matlab utilizzati per lo svolgimento della tesi. La maggior parte dei codici è stata scritta per processi di tipo exponential-Lévy, cambiando l'esponente caratteristico è possibile utilizzare altri modelli di questo tipo. Fanno eccezione i codici (A.4) e (A.8), scritti specificatamente per il modello di Heston, ed il codice (A.9) scritto per il modello di Heston-Hull-White approssimato (H1HW).

A.1 FST per Opzioni Europee

Funzione per il calcolo di opzioni Europee tramite il metodo FST, con modello di Merton.

```
function [prezzo_call,prezzo_put]=FST_Eur_MJD(S0,K,T,r,q,...
                                                sigma,lambda,mu,delta,M,L)

%% VARIABILI
% S0=spot, K=strike , r=risk-free , q=dividend-yeld, T=maturity,
% (sigma,lambda,mu,delta)=parametri del modello
% M->usiamo una discretizzazione con N^M punti spaziali
% L=ampiezza intervallo.

% funzione caratteristica con condizione risk-neutral
```

```
gamma=r-q-0.5*sigma^2 - lambda*(exp(0.5*delta^2+mu)-1);
phi= @(u) -0.5*(sigma^2)*(u.^2) + 1i*u*gamma + ...
                                lambda*(exp(-0.5*(delta^2)*(u.^2)+1i*u*mu)-1);

% griglie
N=2^M;
xmin=-L; xmax=L;
k=(0:N-1);
dx=(xmax-xmin)/N;
x=xmin+k*dx;
wmin=-pi/dx; wmax=pi/dx;
dw=2*wmax/N;
w=[0:dw:wmax, wmin+dw:dw:-dw];

% payoff
v_call=max(0,S0*exp(x)-K);
v_put=max(0,K-S0*exp(x));
v_cap_call=fft(v_call);
v_cap_put=fft(v_put);
v_call=real(ifft(v_cap_call.*exp(T* (phi(w)-r))));
v_put=real(ifft(v_cap_put.*exp(T* (phi(w)-r))));

s=S0*exp(x);
prezzo_call=interp1(s,v_call,S0,'spline');
prezzo_put=interp1(s,v_put,S0,'spline');
```

A.2 FST per Opzioni Americane

Funzione per il calcolo di opzioni Bermuda tramite il metodo FST, con modello di Merton.

```
function [prezzo_call,prezzo_put]=FST_Am_MJD(S0,K,T,r,q,...
                                                sigma,lambda,mu,delta,M,NT,L)

%% VARIABILI
% S0=spot, K=strike , r=risk-free , q=dividend-yeld, T=maturity,
% (sigma,lambda,mu,delta)=parametri del modello
% M->usiamo una discretizzazione con N^M punti spaziali
% NT=numero di passi temporali
% L=ampiezza intervallo.

% funzione caratteristica con condizione risk-neutral
gamma=r-q-0.5*sigma^2 - lambda*(exp(0.5*delta^2+mu)-1);
phi= @(u) -0.5*(sigma^2)*(u.^2) + 1i*u*gamma + ...
                              lambda*(exp(-0.5*(delta^2)*(u.^2)+1i*u*mu)-1);

% grid
dt=T/NT;
N=2^M;
xmin=-L; xmax=L;
k=(0:N-1);
dx=(xmax-xmin)/N;
x=xmin+k*dx;
wmin=-pi/dx; wmax=pi/dx;
dw=2*wmax/N;
w=[0:dw:wmax, wmin+dw:dw:-dw];

% payoff
v_call=max(0,S0*exp(x)-K);
v_put=max(0,K-S0*exp(x));
vc=v_call;
```

```
vp=v_put;
for i=1:NT
    vc=max(real(ifft(fft(vc).*exp(dt*(phi(w)-r)) )), v_call );
    vp=max(real(ifft(fft(vp).*exp(dt*(phi(w)-r)) )), v_put );
end

s=S0*exp(x);
prezzo_call=interp1(s,vc,S0,'spline');
prezzo_put=interp1(s,vp,S0,'spline');
```

A.3 FST per Opzioni Barriera

Funzione per il calcolo di opzioni Barriera di tipo Out tramite il metodo FST, con modello Variance Gamma.

```
function [prezzo_call,prezzo_put]=FST_Barrier_VG(S0,K,T,r,q,...
                                                 sigmaVG,theta,k,M,NT,L,Down,Up)

%% VARIABILI
% S0=spot, K=strike , r=risk-free , q=dividend-yeld, T=maturity,
% (sigmaVG,theta,k)=parametri del modello
% M->usiamo una discretizzazione con N^M punti spaziali
% NT=numero di passi temporali
% Down e Up -> barriere
% L=ampiezza intervallo.

% funzione caratteristica con condizione risk-neutral
phi= @(u) -(1/k)*log(1+0.5*k*(u*sigmaVG).^2-1i*theta*k*u);
gamma=-phi(-1i);
phi= @(u) 1i*u*(gamma+r-q) -(1/k)*log(1+0.5*k*(u*sigmaVG).^2-1i*theta*k*u);

% grid
dt=T/NT;
N=2^M;
xmin=-L; xmax=L;
k=(0:N-1);
dx=(xmax-xmin)/N;
x=xmin+k*dx;
wmin=-pi/dx; wmax=pi/dx;
dw=2*wmax/N;
w=[0:dw:wmax, wmin+dw:dw:-dw];
Dx=log(Down/S0);
Ux=log(Up/S0);

% payoff
```

```
v_call=max(0,S0*exp(x)-K);
v_put=max(0,K-S0*exp(x));
vc=v_call;
vp=v_put;
index=find(x<=Dx | x>=Ux);
barrier=ones(size(x));
barrier(index)=0;
for i=1:NT
    vc=real(ifft(fft(vc).*exp(dt*(phi(w)-r)) ));
    vc=vc.*barrier;
    vp=real(ifft(fft(vp).*exp(dt*(phi(w)-r)) ));
    vp=vp.*barrier;
end

s=S0*exp(x);
prezzo_call=interp1(s,vc,S0,'spline');
prezzo_put=interp1(s,vp,S0,'spline');
```

A.4 FST per Opzioni Europee con Modello di Heston

Funzione per il calcolo di opzioni Europee tramite il metodo FST, con modello di Heston.

```
function [prezzo_call,prezzo_put]=FST_Eur_H(S0,K,T,r,q,...
                                            v0,vlong,lambda,rho,sigma,M,L)

%% VARIABILI
% S0=spot, K=strike , r=risk-free , q=dividend-yeld, T=maturity,
% v0=volatilita istantanea,
% (vlong,lambda,rho,sigma)=parametri del modello di Heston
% M->usiamo una discretizzazione con N^M punti spaziali
% L=ampiezza intervallo.

% grid
N=2^M;
xmin=-L; xmax=L;
k=(0:N-1);
dx=(xmax-xmin)/N;
x=xmin+k*dx;
wmin=-pi/dx;
wmax=pi/dx;
dw=2*wmax/N;
w=[0:dw:wmax, wmin+dw:dw:-dw];

% payoff
v_call=max(0,S0*exp(x)-K);
v_put=max(0,K-S0*exp(x));
v_cap_call=fft(v_call);
v_cap_put=fft(v_put);
phi=exp(cf_heston(w,T,r,q,v0,vlong,lambda,sigma,rho));
v_call=real(ifft(v_cap_call.*phi.*exp(-T*r)));
v_put=real(ifft(v_cap_put.*phi.*exp(-T*r)));
```

```
s=S0*exp(x);
prezzo_call=interp1(s,v_call,S0,'spline');
prezzo_put=interp1(s,v_put,S0,'spline');

end

%% Funzione caratteristica di Heston, x0=0;
function y = cf_heston(u,T,r,d,v0,vlong,lambda,sigma,rho)

alfa = -.5*(u.*u + u*1i);
beta = lambda - rho*sigma*u*1i;
sigma2 = sigma * sigma;
gamma = .5 * sigma2;
D = sqrt(beta .* beta - 4.0 * alfa .* gamma);
bD = beta - D;
eDt = exp(- D * T);
G = bD ./ (beta + D);
B = (bD ./ sigma2) .* ((1.0 - eDt) ./ (1.0 - G .* eDt));
psi = (G .* eDt - 1.0) ./(G - 1.0);
A = ((lambda * vlong) / (sigma2)) * (bD * T - 2.0 * log(psi));
y = A + B*v0 + 1i*u*((r-d)*T);

end
```

A.5 Regime-Switching FST per Opzioni Europee

Funzione per il calcolo di opzioni Europee tramite il metodo Regime-Switching FST, con modello Merton e 5 stati del mondo.

```
function [Call,Put]=rsFST_Eur_MJD(S0,K,T,r,q,sigma1,sigma2,sigma3,...
                sigma4,sigma5,p1,p2,p3,p4,p5,lambda,mu,delta,M,L,tasso,tag)

%% VARIABILI
% S0=spot, K=strike , r=risk-free , q=dividend-yeld, T=maturity,
% (lambda,mu,delta)=parametri della parte Jump
% sigma1,sigma2,... volatilita dei vari stati del mondo
% p1,p2,... probabilita dei vari stati del mondo
% M->usiamo una discretizzazione con N^M punti spaziali
% tasso=tasso di transizione, supponiamo che il processo possa saltare
% solo negli stati adiacenti, tutti con lo stesso tasso di transizione
% tag=1,...,5 stato del mondo in cui ci troviamo noto, tag=0 non noto
% L=ampiezza intervallo.

% esponente caratteristico con condizione risk-neutral
gamma1=r-q-0.5*sigma1^2 - lambda*(exp(0.5*delta^2+mu)-1);
gamma2=r-q-0.5*sigma2^2 - lambda*(exp(0.5*delta^2+mu)-1);
gamma3=r-q-0.5*sigma3^2 - lambda*(exp(0.5*delta^2+mu)-1);
gamma4=r-q-0.5*sigma4^2 - lambda*(exp(0.5*delta^2+mu)-1);
gamma5=r-q-0.5*sigma5^2 - lambda*(exp(0.5*delta^2+mu)-1);
phi1= @(u) -0.5*(sigma1^2)*(u.^2) + 1i*u*gamma1 + ...
           lambda*(exp(-0.5*(delta^2)*(u.^2)+1i*u*mu)-1);
phi2= @(u) -0.5*(sigma2^2)*(u.^2) + 1i*u*gamma2 + ...
           lambda*(exp(-0.5*(delta^2)*(u.^2)+1i*u*mu)-1);
phi3= @(u) -0.5*(sigma3^2)*(u.^2) + 1i*u*gamma3 + ...
           lambda*(exp(-0.5*(delta^2)*(u.^2)+1i*u*mu)-1);
phi4= @(u) -0.5*(sigma4^2)*(u.^2) + 1i*u*gamma4 + ...
           lambda*(exp(-0.5*(delta^2)*(u.^2)+1i*u*mu)-1);
phi5= @(u) -0.5*(sigma5^2)*(u.^2) + 1i*u*gamma5 + ...
           lambda*(exp(-0.5*(delta^2)*(u.^2)+1i*u*mu)-1);
```

```
% grid
N=2^M;
xmin=-L; xmax=L;
k=(0:N-1);
dx=(xmax-xmin)/(N-1);
x=xmin+k*dx;
wmin=-pi/dx; wmax=pi/dx;
dw=2*wmax/N;
w=[0:dw:wmax, wmin+dw:dw:-dw];
B=[-2*tasso 2*tasso 0 0 0; tasso -2*tasso tasso 0 0 ; ...
    0 tasso -2*tasso tasso 0 ; 0 0 tasso -2*tasso tasso ; ...
    0 0 0 2*tasso -2*tasso];
A=zeros(5,5,N);
P=[p1 p2 p3 p4 p5];

if (sum(P)~=1)
    display('ERRORE LE PROBABILITA NON SOMMANO AD UNO');
    return
end

for ii=1:N
    A(:,:,ii)=B;
    A(1,1,ii)=A(1,1,ii)+phi1(w(ii))-r;
    A(2,2,ii)=A(2,2,ii)+phi2(w(ii))-r;
    A(3,3,ii)=A(3,3,ii)+phi3(w(ii))-r;
    A(4,4,ii)=A(4,4,ii)+phi4(w(ii))-r;
    A(5,5,ii)=A(5,5,ii)+phi5(w(ii))-r;
    A(:,:,ii)=expm(A(:,:,ii)*T);
end


% condizione finale
v_call=[max(0,S0*exp(x)-K);max(0,S0*exp(x)-K);max(0,S0*exp(x)-K); ...
                                        max(0,S0*exp(x)-K);max(0,S0*exp(x)-K)];
v_put=[max(0,K-S0*exp(x));max(0,K-S0*exp(x));max(0,K-S0*exp(x)); ...
```

```
                                        max(0,K-S0*exp(x));max(0,K-S0*exp(x))];

% in ogni riga ho una Vk,calcolare la fft per ogni riga (Dim 2)
v_cap_call=fft(v_call,[],2);
v_cap_put=fft(v_put,[],2);

v_cap_exp_call=zeros(5,N);
v_cap_exp_put=zeros(5,N);
for kk=1:N
    v_cap_exp_call(:,kk)=A(:,:,kk)*v_cap_call(:,kk);
    v_cap_exp_put(:,kk)=A(:,:,kk)*v_cap_put(:,kk);
end
v_call=real(ifft(v_cap_exp_call,[],2));
v_put=real(ifft(v_cap_exp_put,[],2));

s=S0*exp(x);
if (tag==0)
v_finale_call=v_call(1,:)*P(1)+v_call(2,:)*P(2)+v_call(3,:)*P(3)+ ...
                                            v_call(4,:)*P(4)+v_call(5,:)*P(5);
v_finale_put=v_put(1,:)*P(1)+v_put(2,:)*P(2)+v_put(3,:)*P(3)+ ...
                                              v_put(4,:)*P(4)+v_put(5,:)*P(5);
elseif(tag==1 || tag==2 || tag==3 || tag==4 || tag==5)
v_finale_call=v_call(tag,:)*P(tag);
v_finale_put=v_put(tag,:)*P(tag);
else
display('abbiamo supposto non noto lo stato del mondo in cui ci troviamo')
v_finale_call=v_call(1,:)*P(1)+v_call(2,:)*P(2)+v_call(3,:)*P(3)+ ...
                                            v_call(4,:)*P(4)+v_call(5,:)*P(5);
v_finale_put=v_put(1,:)*P(1)+v_put(2,:)*P(2)+v_put(3,:)*P(3)+ ...
                                              v_put(4,:)*P(4)+v_put(5,:)*P(5);
end

Call=interp1(s,v_finale_call,S0);
Put=interp1(s,v_finale_put,S0);
```

A.6 FST per Equazione Forward

Funzione per il calcolo di opzioni Europee tramite il metodo Forward FST, con modello Merton.

```
function [prezzo_call,prezzo_put,k]=fFST_Eur_MJD(S0,K,T,r,q,...
                                                    sigma,lambda,mu,delta,M,L)

%% VARIABILI
% S0=spot, K=strike , r=risk-free , q=dividend-yeld, T=maturity,
% (sigma,lambda,mu,delta)=parametri del modello
% M->usiamo una discretizzazione con N^M punti spaziali
% L=ampiezza intervallo.

% funzione caratteristica con condizione risk-neutral
gamma=r-q-0.5*sigma^2 - lambda*(exp(0.5*delta^2+mu)-1);
phi= @(u) -0.5*(sigma^2)*(u.^2) + 1i*u*gamma + ...
                                lambda*(exp(-0.5*(delta^2)*(u.^2)+1i*u*mu)-1);

% griglie
N=2^M;
xmin=-L; xmax=L;
kk=(0:N-1);
dx=(xmax-xmin)/N;
x=xmin+kk*dx;
wmin=-pi/dx;
wmax=pi/dx;
dw=2*wmax/N;
w=[0:dw:wmax, wmin+dw:dw:-dw];
k=K*exp(x);

% payoff
v_call=max(0,S0-k);
v_put=max(0,k-S0);
v_cap_call=fft(v_call);
```

```
v_cap_put=fft(v_put);
v_call=real(ifft(v_cap_call.*exp(T* (phi(-1i-w)-r))));
v_put=real(ifft(v_cap_put.*exp(T* (phi(-1i-w)-r))));

prezzo_call=v_call;
prezzo_put=v_put;
```

A.7 Multi-Asset FST per Opzioni Europee

Funzione per il calcolo di opzioni Europee Multi-Asset tramite il metodo FST, con modello Merton e caso 2D.

```
function [prezzo_call,prezzo_put]=FST_Eur_MJD_2D(S1,S2,K,T,r,q,...
          sigma1,lambda1,mu1,delta1,sigma2,lambda2,mu2,delta2,rho,M,L1,L2)

%% VARIABILI
% S0=spot, K=strike , r=risk-free , q=dividend-yeld, T=maturity,
% (sigma1,lambda1,mu1,delta1)=parametri del modello per S1
% (sigma2,lambda2,mu2,delta2)=parametri del modello per S2
% rho=correlazione tra S1 e S2
% M->usiamo una discretizzazione con N^M punti spaziali
% L1 e L2 ampiezza intervalli per S1 e S2.

% funzione caratteristica con condizione risk-neutral
gamma1=r-q-0.5*sigma1^2 - lambda1*(exp(0.5*delta1^2+mu1)-1);
phi1= @(u) -0.5*(sigma1^2)*(u.^2) + 1i*u*gamma1 + ...
                          lambda1*(exp(-0.5*(delta1^2)*(u.^2)+1i*u*mu1)-1);
gamma2=r-q-0.5*sigma2^2 - lambda2*(exp(0.5*delta2^2+mu2)-1);
phi2= @(u) -0.5*(sigma2^2)*(u.^2) + 1i*u*gamma2 + ...
                          lambda2*(exp(-0.5*(delta2^2)*(u.^2)+1i*u*mu2)-1);
interaction= @(u1,u2) -rho*sigma1*sigma2*u1*u2;

% grid
N=2^M;
phi=zeros(N,N);
xmin=-L1; xmax=L1;
dx=(xmax-xmin)/(N-1);
k=(0:N-1);
x=xmin+k*dx;
wmax=pi/dx;
dw=2*wmax/N;
w=[0:dw:wmax, -wmax+dw:dw:-dw];
```

```
ymin=-L2; ymax=L2;
dy=(ymax-ymin)/(N-1);
y=ymin+k*dy;
y=y';
wwmax=pi/dy;
dww=2*wwmax/N;
ww=[0:dww:wwmax, -wwmax+dww:dww:-dww];

% esponente caratteristico del processo bidimensionale
for i=1:N
    for j=1:N
        phi(i,j)=phi1(w(j))+phi2(ww(i))+interaction(w(j),ww(i));
    end
end
clear w ww

% condizione finale, payoff
v_call=zeros(N,N);
v_put=zeros(N,N);
for i=1:N
    for j=1:N
        v_call(i,j)=max(0,S1*exp(x(j))+S2*exp(y(i))-K);
        v_put(i,j)=max(0,K-S1*exp(x(j))-S2*exp(y(i)));
    end
end
v_cap_call=fft2(v_call);
v_cap_put=fft2(v_put);
v_call=real(ifft2(v_cap_call.*exp(T* (phi-r))));
v_put=real(ifft2(v_cap_put.*exp(T* (phi-r))));

prezzo_call=interp2(S1*exp(x),S2*exp(y),v_call,S1,S2,'spline');
prezzo_put=interp2(S1*exp(x),S2*exp(y),v_put,S1,S2,'spline');
```

A.8 CONV per Opzioni Europee con Modello di Heston

Funzione per il calcolo di opzioni Europee tramite il metodo CONV, con modello di Heston.

```
function [PCall,PPut]=CONV_Heston(S0,K,r,q,...
                                  T,v0,vlong,lambda,rho,sigma,M,tag,L,alpha)

%% VARIABILI
% S0=spot, K=strike , r=risk-free , q=dividend-yeld, T=maturity,
% v0=volatilita istantanea,
% (vlong,lambda,rho,sigma)=parametri del modello di Heston
% M->usiamo una discretizzazione con N^M punti spaziali
% L=ampiezza intervallo,
% alpha=parametro di dumping
% tag->discretizzazione CONV: tag=1 ,tag=2 .

if(2*lambda*vlong < sigma*sigma)
    display('FELLER CONDITION NON VERIFICATA');
end

N=2^M;
j=(0:N-1)';
dy=L/N ; du=2*pi /L ;
if tag==1
    x=(j-N/2)*dy ;
    y=x ;
    u=(j-N/2)*du ;
elseif tag==2
    x=(j-N/2)* dy ;
    y=log ( K/S0 )+(j-N/2)* dy ;
    u=(j-N/2)* du ;
end

%payoff
```

```
payoffC=@(t) max( S0*exp(t)-K , 0 );
payoffP=@(t) max( K-S0*exp(t) , 0 );

%pesi quadratura
w=ones(length(j),1); w(1)=0.5; w(end)=0.5;

sgn=(-ones(length(j),1) ).^j;

% payoff with dump
vC=exp(alpha*y).*payoffC(y);
vP=exp(alpha*y).*payoffP(y);

Dj_C=N*ifft(sgn.*w.*vC);
Dj_P=N*ifft(sgn.*w.*vP);
H_CF=cf_heston(-u+1i*alpha,T,r,q,v0,vlong,lambda,sigma,rho);
Dp_C=1/N*fft(exp(1i*j*(y(1)-x(1))*du).*H_CF.*Dj_C);
Dp_P=1/N*fft(exp(1i*j*(y(1)-x(1))*du).*H_CF.*Dj_P);
Call=exp(-r*T)*exp(1i*u(1)*( y(1)-x(1)))*exp(-alpha*x ).*sgn.*Dp_C;
Put=exp(-r*T)*exp(1i*u(1)*( y(1)-x(1)))*exp(-alpha*x ).*sgn.*Dp_P;
S=S0*exp(y);

PCall=interp1(S,real(Call),S0,'spline');
PPut=interp1(S,real(Put),S0,'spline');
end

%% Funzione caratteristica di Heston, x0=0;
function y = cf_heston(u,T,r,d,v0,vlong,lambda,sigma,rho)

alfa = -.5*(u.*u + u*1i);
beta = lambda - rho*sigma*u*1i;
sigma2 = sigma * sigma;
gamma = .5 * sigma2;
D = sqrt(beta .* beta - 4.0 * alfa .* gamma);
bD = beta - D;
eDt = exp(- D * T);
G = bD ./ (beta + D);
```

```
B = (bD ./ sigma2) .* ((1.0 - eDt) ./ (1.0 - G .* eDt));
psi = (G .* eDt - 1.0) ./(G - 1.0);
A = ((lambda * vlong) / (sigma2)) * (bD * T - 2.0 * log(psi));
y = A + B*v0 + 1i*u*((r-d)*T);

end
```

A.9 FST per Opzioni Europee con Modello H1HW

Funzione per il calcolo di opzioni Europee tramite il metodo FST, con modello di Heston-Hull-White approssimato (H1HW).

```
function [prezzo_call,prezzo_put]=FST_Eur_H1HW(S0,K,T,r,v0,vlong,...
                lambda_h,rho_h,sigma_h,lambda_r,theta_r,eta_r,rho_xr,M,L)

%% VARIABILI
% S0=spot, K=strike, T=maturity,
% v0=volatilita istantanea, r=tasso istantaneo,
% (vlong,lambda_h,rho_h,sigma_h)=parametri del modello di Heston
% (theta_r,lambda_,rho_xr,eta_r)=parametri del modello di Hull-White
% M->usiamo una discretizzazione con N^M punti spaziali
% L=ampiezza intervallo.

% grid
N=2^M;
xmin=-L; xmax=L;
k=(0:N-1);
dx=(xmax-xmin)/N;
x=xmin+k*dx;
wmin=-pi/dx; wmax=pi/dx;
dw=2*wmax/N;
w=[0:dw:wmax, wmin+dw:dw:-dw];

% payoff
v_call=max(0,S0*exp(x)-K);
v_put=max(0,K-S0*exp(x));
v_cap_call=fft(v_call);
v_cap_put=fft(v_put);
phi=exp(cf_h1hw(w,T,r,v0,vlong,lambda_h,sigma_h,lambda_r,eta_r,...
                                                rho_h,rho_xr,theta_r));
v_call=real(ifft(v_cap_call.*phi));
v_put=real(ifft(v_cap_put.*phi));
```

```
s=S0*exp(x);
prezzo_call=interp1(s,v_call,S0,'spline');
prezzo_put=interp1(s,v_put,S0,'spline');

end

%% Funzione caratteristica di H1HW
function y = cf_h1hw(u,T,r0,V0,theta,kappa,omega,lambda,eta,...
                                                    rho12,rho13,thetar)

    D1 = sqrt((omega*rho12*1i*u-kappa).^2-omega^2*1i*u.*(1i*u-1));
    g = (kappa-omega*rho12*1i*u-D1)./(kappa-omega*rho12*1i*u+D1);
    a = sqrt(theta - .125 * omega^2/kappa);
    b = sqrt(V0) - a;
    ct=.25*omega^2*(1-exp(-kappa))/kappa;
    lambdat=4*kappa*V0*exp(-kappa)/(omega^2*(1-exp(-kappa)));
    d2=4*kappa*theta/omega^2;
    F1 = sqrt(ct*(lambdat-1)+ct*d2+ct*d2/(2*(d2+lambdat)));
    c = -log((F1-a)/b);
    I1 = thetar * (1i*u-1) * (T+(exp(-lambda * T)-1)/lambda);
    I2 = kappa*theta/omega^2*(T*(kappa-omega*rho12*1i*u-D1)-...
                                        2*log((1-g.*exp(-D1*T))./(1-g)));
    I3 = eta^2*(1i+u).^2/(4*lambda^3)*(3+exp(-2*lambda*T)-...
                                          4*exp(-lambda*T)-2*lambda*T);
    I4 = -eta*rho13/lambda *(1i*u+u.^2)*(b/c*(1-exp(-c*T))+a*T+...
         a/lambda*(exp(-lambda*T)-1)+b/(c-lambda)*exp(-c*T)*...
        (1-exp(-T*(lambda-c))));

    A = I1+I2+I3+I4;
    BV = (1-exp(-D1*T))./(omega^2.*(1-g.*exp(-D1*T))).*...
                                          (kappa-omega*rho12*1i*u-D1);
    Br = (1i*u-1)/lambda*(1-exp(-lambda*T));
    y = A  + BV * V0  + Br * r0;
end
```

Appendice B

Codici C/C++

Presentiamo alcuni codici C/C++ utilizzati per lo svolgimento della tesi. I codici sono stati scritti per il processo di Merton, cambiando l'esponente caratteristico è possibile utilizzare altri modelli.

B.1 Header file

Dichiarazione delle funzioni.

```
#ifndef __FST_H__
#define __FST_H__

#include <cstdlib>
#include <iostream>
#include <cmath>
#include <vector>
#include<complex.h>
#include <fftw3.h>

// Funzione per il calcolo del prezzo di un'opzione Call Europea,
// modello di Merton, metodo FST.
double Call_FST_MJD(double S0,double K,double T,double r,double sigma,
```

```
                    double gamma,double mu,double delta,int M);

// Funzione per il calcolo del prezzo di un'opzione Call Europea,
// modello di Merton, metodo FST.
double AmCall_FST_MJD(double S0,double K,double T,double r,
                      double sigma,double lambda,double mu,
                      double delta,int M, int Nt);

// Funzione per il calcolo del prezzo di un'opzione Call Barriera,
// tipo Knock-and-Out (doppia barriera),
// modello di Merton, metodo FST.
double BarrierCall_FST_MJD(double S0,double K,double Up,double Down,
                           double T,double r,double sigma,double lambda,
                           double mu,double delta,int M, int Nt);

// Funzione per il calcolo del prezzo di un'opzione Call Barriera 2D,
// tipo Knock-and-Out (doppia barriera),
// modello di Merton, metodo FST,
// payoff max(0,S1(T)+S2(T)-K) se l'opzione rimane attiva.
double Call_FST_MJD_Barr_2D(double S01,double S02,double K,double T,
       double r,double sigma1,double sigma2,double rho,double lambda1,
       double lambda2,double mu1,double mu2,double delta1,
       double delta2,int M,double U,double D,int Nt);

#endif
```

B.2 File .cpp

Implementazione delle funzioni.

```
#include "FST.h"

using namespace std;

// Funzione per il calcolo del prezzo di un'opzione Call Europea,
// modello di Merton, metodo FST.
double Call_FST_MJD(double S0,double K,double T,double r,double sigma,
                    double lambda,double mu,double delta,int M)
{
    int N=pow(2,M);
    double xmin=-10.0; double xmax=10.0;
    double dx=(xmax-xmin)/N;
    double *x;
    x = (double*) fftw_malloc(sizeof(double) * N);
    double *v_Call;
    v_Call = (double*) fftw_malloc(sizeof(double) * N);
    double wmin=-M_PI/dx; double wmax=M_PI/dx;
    double dw=(wmax-wmin)/N;
    double gamma=r-0.5*sigma*sigma-lambda*(exp(0.5*delta*delta+mu)-1);
    fftw_complex *phi; //stesso tipo della complex.h
    phi = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*(N/2+1));

    for(int k=0; k<(N/2+1) ;k++)
    {
        x[k]=xmin+k*dx;
        phi[k]=-0.5*(sigma*sigma)*(k*dw*k*dw) + I*k*dw*gamma +
            lambda*(cexp(-0.5*(delta*delta)*(k*dw*k*dw)+I*k*dw*mu)-1);
    }

    for(int k=(N/2+1); k<N ;k++)
    {
```

```
    x[k]=xmin+k*dx;
}

fftw_complex *out;
out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*(N/2+1));
fftw_plan pf,pb;
pf = fftw_plan_dft_r2c_1d(N, v_Call, out,FFTW_ESTIMATE);
pb = fftw_plan_dft_c2r_1d(N, in, prezzi, FFTW_ESTIMATE);

for(int k=0; k<N ;k++)
{
    v_Call[k] = max(0.0,(S0*exp(x[k])-K));
}

fftw_execute(pf);

double *prezzi;
prezzi = (double*) fftw_malloc(sizeof(double) * N);
fftw_complex *in;
in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*(N/2+1));

for(int k=0; k<(N/2+1) ;k++)
{
    in[k]=out[k]*cexp(T*(phi[k]-r));
}

fftw_execute(pb);

for(int k=0; k<N ;k++)
{
    prezzi[k]=prezzi[k]/N;
}

double price=prezzi[N/2+1];

fftw_destroy_plan(pf); fftw_destroy_plan(pb);
```

```
    fftw_free(v_Call); fftw_free(out); fftw_free(in);
    fftw_free(prezzi); fftw_free(phi); fftw_free(x);

    return price;
}

// Funzione per il calcolo del prezzo di un'opzione Call Europea,
// modello di Merton, metodo FST.
double AmCall_FST_MJD(double S0,double K,double T,double r,
                      double sigma,double lambda,double mu,
                      double delta,int M, int Nt)
{
    int N=pow(2,M);
    double xmin=-10.0; double xmax=10.0;
    double dx=(xmax-xmin)/N;
    double *x;
    x = (double*) fftw_malloc(sizeof(double) * N);
    double *v_Call;
    v_Call = (double*) fftw_malloc(sizeof(double) * N);
    double wmin=-M_PI/dx; double wmax=M_PI/dx;
    double dw=(wmax-wmin)/N;
    double gamma=r-0.5*sigma*sigma-lambda*(exp(0.5*delta*delta+mu)-1);
    fftw_complex *phi; //stesso tipo della complex.h
    phi = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*(N/2+1));
    double *prezzi;
    prezzi = (double*) fftw_malloc(sizeof(double) * N);
    double dt=(T/Nt);

    for(int k=0; k<(N/2+1) ;k++)
    {
        x[k]=xmin+k*dx;
        phi[k]=-0.5*(sigma*sigma)*(k*dw*k*dw) + I*k*dw*gamma +
            lambda*(cexp(-0.5*(delta*delta)*(k*dw*k*dw)+I*k*dw*mu)-1);
    }

    for(int k=(N/2+1); k<N ;k++)
```

```
    {
        x[k]=xmin+k*dx;
    }

    fftw_complex *out;
    out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*(N/2+1));
    fftw_plan pf,pb;
    pf = fftw_plan_dft_r2c_1d(N, prezzi, out,FFTW_ESTIMATE);
    pb = fftw_plan_dft_c2r_1d(N, out, prezzi, FFTW_ESTIMATE);

    for(int k=0; k<N ;k++)
    {
        v_Call[k] = max(0.0,(S0*exp(x[k])-K));
        prezzi[k]=v_Call[k];
    }


for (int i=0; i<Nt ; i++)
{
    fftw_execute(pf);

    for(int k=0; k<(N/2+1) ;k++)
    {
        out[k]=out[k]*cexp(dt*(phi[k]-r));
    }

    fftw_execute(pb);

    for(int k=0; k<N ;k++)
    {
        prezzi[k]=max( (prezzi[k]/N) , v_Call[k] );
    }
}

    double price=prezzi[N/2+1];
```

```
    fftw_free(x); fftw_free(phi);
    fftw_destroy_plan(pf);
    fftw_destroy_plan(pb);
    fftw_free(v_Call); fftw_free(out); fftw_free(prezzi);

    return price;
}

// Funzione per il calcolo del prezzo di un'opzione Call Barriera,
// tipo Knock-and-Out (doppia barriera),
// modello di Merton, metodo FST.
double BarrierCall_FST_MJD(double S0,double K,double Up,double Down,
                           double T,double r,double sigma,double lambda,
                           double mu,double delta,int M, int Nt)

{
    int N=pow(2,M);
    double xmin=-10.0; double xmax=10.0;
    double dx=(xmax-xmin)/N;
    double *x;
    x = (double*) fftw_malloc(sizeof(double) * N);
    double *v_Call;
    v_Call = (double*) fftw_malloc(sizeof(double) * N);
    double wmin=-M_PI/dx; double wmax=M_PI/dx;
    double dw=(wmax-wmin)/N;
    double gamma=r-0.5*sigma*sigma-lambda*(exp(0.5*delta*delta+mu)-1);
    fftw_complex *phi; //stesso tipo della complex.h
    phi = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*(N/2+1));
    double *prezzi;
    prezzi = (double*) fftw_malloc(sizeof(double) * N);
    double dt=(T/Nt);

    for(int k=0; k<(N/2+1) ;k++)
    {
        x[k]=xmin+k*dx;
        phi[k]=-0.5*(sigma*sigma)*(k*dw*k*dw) + I*k*dw*gamma +
```

```
        lambda*(cexp(-0.5*(delta*delta)*(k*dw*k*dw)+I*k*dw*mu)-1);
}

for(int k=(N/2+1); k<N ;k++)
{
    x[k]=xmin+k*dx;
}

fftw_complex *out;
out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*(N/2+1));
fftw_plan pf,pb;
pf = fftw_plan_dft_r2c_1d(N, prezzi, out,FFTW_ESTIMATE);
pb = fftw_plan_dft_c2r_1d(N, out, prezzi, FFTW_ESTIMATE);

for(int k=0; k<N ;k++)
{
    v_Call[k] = max(0.0,(S0*exp(x[k])-K));
    prezzi[k]=v_Call[k];
}

for (int i=0; i<Nt ; i++)
{

    fftw_execute(pf);

    for(int k=0; k<(N/2+1) ;k++)
    {
        out[k]=out[k]*cexp(dt*(phi[k]-r));
    }

    fftw_execute(pb);
    for(int k=0; k<N ;k++)
    {
        if ((S0*exp(xmin+k*dx))<=Down || (S0*exp(xmin+k*dx))>=Up)
        prezzi[k]=0;
        else
```

```
            prezzi[k]=prezzi[k]/N;
        }
    }

    double price=prezzi[N/2+1];

    fftw_free(x); fftw_free(phi);
    fftw_destroy_plan(pf);
    fftw_destroy_plan(pb);
    fftw_free(v_Call); fftw_free(out); fftw_free(prezzi);

    return price;
}

// Funzione per il calcolo del prezzo di un'opzione Call Barriera 2D,
// tipo Knock-and-Out (doppia barriera),
// modello di Merton, metodo FST,
// payoff max(0,S1(T)+S2(T)-K) se l'opzione rimane attiva.
double Call_FST_MJD_Barr_2D(double S0_1,double S0_2,double K,double T,
       double r,double sigma_1,double sigma_2,double rho,double lambda_1,
       double lambda_2,double mu_1,double mu_2,double delta_1,
       double delta_2,int M,double U,double D,int Nt)
{
    double dt=T/Nt;
    int N=pow(2,M);
    double xmin=-8.0;
    double xmax=8.0;
    double dx=(xmax-xmin)/N;
    double *v_Call;
    double* ris;
    double wmin=-M_PI/dx;
    double wmax=M_PI/dx;
    double dw=(wmax-wmin)/N;
    double gamma1=r-0.5*sigma_1*sigma_1-lambda_1*
                  (exp(0.5*delta_1*delta_1+mu_1)-1);
    double gamma2=r-0.5*sigma_2*sigma_2-lambda_2*
```

```
                (exp(0.5*delta_2*delta_2+mu_2)-1);
fftw_complex *phi;
phi = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N*(N/2+1));
v_Call = (double*) fftw_malloc(sizeof(double)*N*N);
fftw_complex *exponent;
exponent=(fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N*(N/2+1));
ris = (double*) fftw_malloc(sizeof(double)*N*N);

for (int i=0;i<(N/2+1);i++){
    for(int k=0; k<(N/2+1) ;k++)
    {
        phi[i*(N/2+1)+k]=-0.5*(sigma_1*sigma_1)*(k*dw*k*dw)+
        I*k*dw*gamma1 + lambda_1*( cexp(-0.5*(delta_1*delta_1)*
        (k*dw*k*dw)+I*k*dw*mu_1)-1)-0.5*(sigma_2*sigma_2)*
        (i*dw*i*dw) + I*i*dw*gamma2 + lambda_2*(cexp(-0.5*
        (delta_2*delta_2)*(i*dw*i*dw)+I*i*dw*mu_2)-1)-
        rho*sigma_1*sigma_2*(k*dw)*(i*dw)    ;
    }
}

for (int i=(N/2+1);i<N;i++){
    for(int k=0; k<(N/2+1) ;k++)
    {
        phi[i*(N/2+1)+k]=-0.5*(sigma_1*sigma_1)*(k*dw*k*dw)+
        I*k*dw*gamma1+lambda_1*(cexp(-0.5*(delta_1*delta_1)
        *k*dw*k*dw)+I*k*dw*mu_1)-1)-0.5*(sigma_2*sigma_2)*
        ((wmin+(i-N/2)*dw)*(wmin+(i-N/2)*dw))+I*(wmin+(i-N/2)*dw)
        *gamma2+lambda_2*(cexp(-0.5*(delta_2*delta_2)*
        ((wmin+(i-N/2)*dw)*(wmin+(i-N/2)*dw))+I*(wmin+(i-N/2)*dw)
        *mu_2)-1)-rho*sigma_1*sigma_2*(k*dw)*(wmin+(i-N/2)*dw);
    }
}

for (int i=0;i<N;i++){
    for(int k=0;k<N;k++)
    {
```

```
        v_Call[i*N+k]=max(0.0,(S0_1*exp(xmin+k*dx)+
                              S0_2*exp(xmin+i*dx)-K));
        ris[i*N+k]=v_Call[i*N+k];
    }
}

for (int i=0;i<N;i++){
    for(int k=0;k<N/2+1;k++)
    {
        exponent[i*(N/2+1)+k] =cexp(dt*(phi[i*(N/2+1)+k]-r)) ;
    }
}

fftw_complex *out;
fftw_plan pf,pb;
out=(fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N*(N/2+1));
pf=fftw_plan_dft_r2c_2d(N,N,ris,out,FFTW_ESTIMATE);
pb=fftw_plan_dft_c2r_2d(N,N,out,ris,FFTW_ESTIMATE);

for (int kk=0;kk<Nt;kk++)
{

    fftw_execute(pf);
    for (int i=0;i<N;i++){
        for(int k=0;k<N/2+1;k++)
        {
            out[i*(N/2+1)+k]=out[i*(N/2+1)+k]*exponent[i*(N/2+1)+k];
        }
    }
    fftw_execute(pb);

    for (int i=0;i<N;i++){
        for(int k=0;k<N;k++)
        {
            if ( S0_1*exp(xmin+dx*i)<D || S0_1*exp(xmin+dx*i)>U ||
                 S0_2*exp(xmin+dx*k)<D || S0_2*exp(xmin+dx*k)>U ){
```

```
                    ris[i*N+k]=0.0;
                }
                else {
                    ris[i*N+k]=ris[i*N+k]/(N*N) ;
                }
            }
        }
    }

    double price=ris[N*(N/2)+N/2];

    fftw_free(phi);
    fftw_destroy_plan(pf);
    fftw_destroy_plan(pb);
    fftw_free(v_Call); fftw_free(out); fftw_free(exponent);

    return price;
}
```

Appendice C

Codici CUDA

Presentiamo alcuni codici CUDA utilizzati per lo svolgimento della tesi. I codici sono stati scritti per il processo di Merton, cambiando l'esponente caratteristico è possibile utilizzare altri modelli.

C.1 Header file

Dichiarazione delle funzioni.

```
#ifndef __cuFST_H__
#define __cuFST_H__

#include<cstdlib>
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
#include<complex.h>
#include<cuComplex.h>
#include<cufft.h>
```

```
////////////////////////////////////////////////////////////////
// funzioni ibride eseguite in parte su CPU e in parte su GPU //
////////////////////////////////////////////////////////////////

// Funzione per il calcolo del prezzo di un'opzione Call Europea,
// modello di Merton, metodo FST.
double Call_FST_MJD_CUDA(double S0,double K,double T,double r,
                         double sigma,double gamma,double mu,
                         double delta,int M);

// Funzione per il calcolo del prezzo di un'opzione Call Americana,
// modello di Merton, metodo FST.
double AmCall_FST_MJD_CUDA(double S0,double K,double T,double r,
                           double sigma,double lambda,double mu,
                           double delta,int M,int Nt);

// Funzione per il calcolo del prezzo di opzioni Barriera,
// tipo Knock-and-Out (doppia barriera),
// modello di Merton, metodo FST,
// la funzione stampa a video i prezzi di Call e Put Barriera,
// utilizziamo la modalità BATCH della cuFFT per velocizzare il calcolo
// di più FFT aventi la stessa dimensione.
double Barrier_FST_MJD_CUDA(double S0,double K,double Down,double Up,
                            double T,double r,double sigma,double lambda,
                            double mu,double delta,int M, int Nt);

// Funzione per il calcolo del prezzo di un'opzione Call Europea 2D,
// modello di Merton, metodo FST,
// payoff max(0,S1(T)+S2(T)-K).
double Call_FST_MJD_CUDA_2D(double S1,double S2,double K,double T,
       double r,double sigma1,double lambda1,double mu1,double delta1,
       double sigma2,double lambda2,double mu2,double delta2,
       double rho,int M);

// Funzione per il calcolo del prezzo di un'opzione Call Barriera 2D,
// tipo Knock-and-Out (doppia barriera),
```

```
// modello di Merton, metodo FST,
// payoff max(0,S1(T)+S2(T)-K) se l'opzione rimane attiva.
double Call_FST_MJD_CUDA_Barr_2D(double S1,double S2,double K,double T,
       double r,double sigma1,double lambda1,double mu1,double delta1,
       double sigma2,double lambda2,double mu2,double delta2,
       double rho,int M,double D,double U,int Nt);

////////////////////////////////////////////////////////
// funzioni richiamabili da codice C++ ed eseguite su GPU //
////////////////////////////////////////////////////////

// Prodotto di vettori complessi element-wise.
__global__ void Cproduct(cufftDoubleComplex *a,cufftDoubleComplex *b,
                         cufftDoubleComplex *c,int N);

// Prodotto di matrici complesse element-wise.
__global__ void Cproduct2D(cufftDoubleComplex *a,cufftDoubleComplex *b,
                           cufftDoubleComplex *c,int N,int M);

// Applicazione della condizione al bordo per Call Americana.
__global__ void NormFrontCall(cufftDoubleReal *a,int N,double S0,
                              double K,double xmin,double dx);

// Applicazione della condizione al bordo per opzioni Barriera.
__global__ void NormBarrier(cufftDoubleReal *a,int N,double S0,
                            double K,double xmin,double dx,
                            double logDown,double logUp);

// Applicazione della condizione al bordo per opzioni Barriera 2D.
__global__ void NormBarrier2D(cufftDoubleReal *a,int N,double S0,
                              double xmin,double dx,double D,double U);

#endif
```

C.2 File .cu

Implementazione delle funzioni.

```
#include "cuFST.cuh"

using namespace std;

// Funzione per il calcolo del prezzo di un'opzione Call Europea,
// modello di Merton, metodo FST.
double Call_FST_MJD_CUDA(double S0,double K,double T,double r,
                         double sigma,double lambda,double mu,
                         double delta,int M)
{
    int N=pow(2,M);
    //variabili CUDA
    cufftHandle forward,backward;
    cufftDoubleComplex *out,*cesp;
    cufftDoubleReal *in;
    cudaMalloc((void **)&out, sizeof(cufftDoubleComplex)*(N/2+1) );
    cudaMalloc((void **)&cesp, sizeof(cufftDoubleComplex)*(N/2+1) );
    cudaMalloc((void **)&in, sizeof(cufftDoubleReal)*(N) );

    double xmin=-10.0; double xmax=10.0;
    double dx=(xmax-xmin)/N;
    vector<double> X (N,0); double *x=&X[0];
    vector<double> SS (N,0); double *S=&SS[0];
    vector<double> Call (N,0); double *v_Call=&Call[0];
    vector<double> risultato (N,1); double *ris=&risultato[0];
    double wmin=-M_PI/dx; double wmax=M_PI/dx;
    double dw=(wmax-wmin)/N;
    double gamma=r-0.5*sigma*sigma-lambda*(exp(0.5*delta*delta+mu)-1);
    vector<double complex> Phi (N/2 + 1,0);
    double complex *phi=&Phi[0];
```

```
for(int k=0; k<(N/2+1) ;k++)
{
    X[k]=xmin+k*dx;
    SS[k]=S0*exp(X[k]);
    phi[k]=-0.5*(sigma*sigma)*(k*dw*k*dw) + I*k*dw*gamma +
        lambda*(cexp(-0.5*(delta*delta)*(k*dw*k*dw)+I*k*dw*mu)-1);
}
for(int k=(N/2+1); k<N ;k++)
{
    X[k]=xmin+k*dx;
    SS[k]=S0*exp(X[k]);
}
for(int k=0; k<N ;k++)
{
    Call[k] = max(0.0,(S0*exp(x[k])-K));
}
for(int k=0; k<(N/2+1) ;k++)
{
    phi[k]=cexp(T*(phi[k]-r));
}
cudaMemcpy(cesp,phi,sizeof(cufftDoubleComplex)*(N/2 + 1),
           cudaMemcpyHostToDevice);
cudaMemcpy(in,v_Call,sizeof(cufftDoubleReal)*(N),
           cudaMemcpyHostToDevice);
cufftPlan1d(&forward, N , CUFFT_D2Z, 1); //BATCH=1 una sola fft
cufftPlan1d(&backward, N , CUFFT_Z2D, 1);

if (M>22)
{
    printf("M deve essere minore o uguale a 22");
    return 0;
}
int blocchi=M-8;
if (blocchi<1) blocchi=1;
else blocchi=pow(2,blocchi);
```

```
    cufftExecD2Z(forward,in,out);
    Cproduct<<<blocchi,256>>>(out,cesp,out,(N/2 + 1) );
    cufftExecZ2D(backward,out,in);
    cudaThreadSynchronize();
    cudaMemcpy(ris,in,sizeof(cufftDoubleReal)*(N),
               cudaMemcpyDeviceToHost);

    for(int k=0; k<N ;k++)
    {
        ris[k]=ris[k]/N;
    }

    cufftDestroy(forward); cufftDestroy(backward);
    cudaFree(in); cudaFree(out); cudaFree(cesp);

return ris[N/2 + 1];
}

// Prodotto di vettori complessi element-wise.
__global__ void Cproduct(cufftDoubleComplex *a,cufftDoubleComplex *b,
                         cufftDoubleComplex *c,int N)
{
    // Complex multiplication
    // Array dimension N
    int idx=threadIdx.x + blockIdx.x*blockDim.x;
    if (idx < N ){
        double real_a=a[idx].x;
        double imag_a=a[idx].y;
        double real_b=b[idx].x;
        double imag_b=b[idx].y;
        c[idx].x=real_a*real_b-imag_a*imag_b;
        c[idx].y=real_a*imag_b+imag_a*real_b;
    }
}

// Applicazione della condizione al bordo per Call Americana.
```

```
__global__ void NormFrontCall(cufftDoubleReal *a,int N,double S0,
                              double K,double xmin,double dx)
{
    int idx=threadIdx.x + blockIdx.x*blockDim.x;
    if (idx < N ){
        a[idx]=max( (a[idx]/N) , max(0.0,S0*exp(xmin+idx*dx)-K) );
    }
}

// Applicazione della condizione al bordo per opzioni Barriera.
__global__ void NormBarrier(cufftDoubleReal *a,int N,double S0,
                            double K,double xmin,double dx,
                            double logDown,double logUp)
{
    int idx=threadIdx.x + blockIdx.x*blockDim.x;
    if (idx < N ){
        if( (xmin+dx*idx)<=logDown || (xmin+dx*idx)>=logUp ){
        a[idx]=0;
        }
        else {a[idx]=(a[idx]/N);
        }
    }
}

// Funzione per il calcolo del prezzo di un'opzione Call Americana,
// modello di Merton, metodo FST.
double AmCall_FST_MJD_CUDA(double S0,double K,double T,double r,
                           double sigma,double lambda,double mu,
                           double delta,int M, int Nt)
{

    int N=pow(2,M);
    //variabili CUDA
    cufftHandle forward,backward;
    cufftDoubleComplex *out,*cesp;
    cufftDoubleReal *in;
```

```
cudaMalloc((void **)&out, sizeof(cufftDoubleComplex)*(N/2+1) );
cudaMalloc((void **)&cesp, sizeof(cufftDoubleComplex)*(N/2+1) );
cudaMalloc((void **)&in, sizeof(cufftDoubleReal)*(N) );

double xmin=-10.0; double xmax=10.0;
double dx=(xmax-xmin)/N;
vector<double> X (N,0); double *x=&X[0];
vector<double> SS (N,0); double *S=&SS[0];
vector<double> Call (N,0); double *v_Call=&Call[0];
vector<double> risultato (N,1); double *ris=&risultato[0];
double wmin=-M_PI/dx; double wmax=M_PI/dx;
double dw=(wmax-wmin)/N;
double gamma=r-0.5*sigma*sigma-lambda*(exp(0.5*delta*delta+mu)-1);
vector<double complex> Phi (N/2 + 1,0);
double complex *phi=&Phi[0];
double dt=(T/Nt);

for(int k=0; k<(N/2+1) ;k++)
{
    X[k]=xmin+k*dx;
    SS[k]=S0*exp(X[k]);
    phi[k]=-0.5*(sigma*sigma)*(k*dw*k*dw) + I*k*dw*gamma +
        lambda*(cexp(-0.5*(delta*delta)*(k*dw*k*dw)+I*k*dw*mu)-1);
}
for(int k=(N/2+1); k<N ;k++)
{
    X[k]=xmin+k*dx;
    SS[k]=S0*exp(X[k]);
}
for(int k=0; k<N ;k++)
{
    Call[k] = max(0.0,(S0*exp(x[k])-K));
}
for(int k=0; k<(N/2+1) ;k++)
{
    phi[k]=cexp(dt*(phi[k]-r));
```

```
    }
    cudaMemcpy(in,v_Call,sizeof(cufftDoubleReal)*(N),
               cudaMemcpyHostToDevice);
    cudaMemcpy(cesp,phi,sizeof(cufftDoubleComplex)*(N/2 + 1),
               cudaMemcpyHostToDevice);
    cufftPlan1d(&forward, N , CUFFT_D2Z, 1);
    cufftPlan1d(&backward, N , CUFFT_Z2D, 1);

    if (M>22)
    {
        printf("M deve essere minore o uguale a 22");
        return 0;
    }
    int blocchi=M-8;
    if (blocchi<1) blocchi=1;
    else blocchi=pow(2,blocchi);
    for (int i=0; i<Nt ;i++)
    {
        cufftExecD2Z(forward,in,out);
        Cproduct<<<blocchi,256>>>(out,cesp,out,(N/2 + 1) );
        cufftExecZ2D(backward,out,in);
        NormFrontCall<<<blocchi,256>>>(in,N,S0,K,xmin,dx);
    }

    cudaMemcpy(ris,in,sizeof(cufftDoubleReal)*(N),
               cudaMemcpyDeviceToHost);

    for(int k=0; k<N ;k++)
    {
        ris[k]=ris[k];
    }

    cufftDestroy(forward); cufftDestroy(backward);
    cudaFree(in); cudaFree(out); cudaFree(cesp);

return ris[N/2 + 1];
```

```
}


// Funzione per il calcolo del prezzo di opzioni Barriera,
// tipo Knock-and-Out (doppia barriera),
// modello di Merton, metodo FST,
// la funzione stampa a video i prezzi di Call e Put Barriera,
// utilizziamo la modalitÃà BATCH della cuFFT per velocizzare il calcolo
// di piÃź FFT aventi la stessa dimensione.
double Barrier_FST_MJD_CUDA(double S0,double K,double Down,double Up,
                            double T,double r,double sigma,double lambda,
                            double mu,double delta,int M, int Nt)
{

    int N=pow(2,M);

    cufftHandle forward,backward;
    cufftDoubleComplex *out,*cesp;
    cufftDoubleReal *in;
    cudaMalloc((void **)&out, sizeof(cufftDoubleComplex)*(N/2+1)*2);
    cudaMalloc((void **)&cesp, sizeof(cufftDoubleComplex)*(N/2+1));
    cudaMalloc((void **)&in, sizeof(cufftDoubleReal)*(N)*2 );

    double logDown=log(Down/S0); double logUp=log(Up/S0);
    double xmin=-10.0; double xmax=10.0;
    double dx=(xmax-xmin)/N;
    vector<double> SS (N,0); double *S=&SS[0];
    vector<double> Call (2*N,0); double *v_Call=&Call[0];
    vector<double> risultato (2*N,1); double *ris=&risultato[0];
    double wmin=-M_PI/dx; double wmax=M_PI/dx;
    double dw=(wmax-wmin)/N;
    double gamma=r-0.5*sigma*sigma-lambda*(exp(0.5*delta*delta+mu)-1);
    vector<double complex> Phi (N/2 + 1,0);
    double complex *phi=&Phi[0];
    double dt=(T/Nt);
```

```
for(int k=0; k<(N/2+1) ;k++)
{
    SS[k]=S0*exp(xmin+k*dx);
    phi[k]=-0.5*(sigma*sigma)*(k*dw*k*dw) + I*k*dw*gamma +
        lambda*(cexp(-0.5*(delta*delta)*(k*dw*k*dw)+I*k*dw*mu)-1);
}
for(int k=(N/2+1); k<N ;k++)
{
    SS[k]=S0*exp(xmin+k*dx);
}
for(int k=0; k<N ;k++)
{
    Call[k] = max(0.0,SS[k]-K);
    Call[N+k] = max(0.0,K-SS[k]);
}
for(int k=0; k<(N/2+1) ;k++)
{
    phi[k]=cexp(dt*(phi[k]-r));
}

cudaMemcpy(in,v_Call,sizeof(cufftDoubleReal)*(N)*2,
           cudaMemcpyHostToDevice);
cudaMemcpy(cesp,phi,sizeof(cufftDoubleComplex)*(N/2 + 1),
           cudaMemcpyHostToDevice);

cufftPlan1d(&forward, N , CUFFT_D2Z, 2);
cufftPlan1d(&backward, N , CUFFT_Z2D, 2);

if (M>22)
{
    printf("M deve essere minore o uguale a 22");
    return 0;
}
int blocchi=M-8;
if (blocchi<1) blocchi=1;
else blocchi=pow(2,blocchi);
```

```
    for (int i=0; i<Nt ;i++)
    {
        cufftExecD2Z(forward,in,out);//FFT
        Cproduct<<<blocchi,256>>>(out,cesp,out,(N/2 + 1) );
        Cproduct<<<blocchi,256>>>(&out[N/2+1],cesp,&out[N/2+1],N/2+1);
        cufftExecZ2D(backward,out,in);//IFFT
        NormBarrier<<<blocchi,256>>>(in,N,S0,K,xmin,dx,logDown,logUp);
        NormBarrier<<<blocchi,256>>>(&in[N],N,S0,K,xmin,dx,logDown,logUp);
        cudaThreadSynchronize();
    }

    cudaMemcpy(ris,in,sizeof(cufftDoubleReal)*N*2,
               cudaMemcpyDeviceToHost);

    double price_call=ris[N/2+1]; double price_put=ris[N+N/2+1];
    cout<<"I prezzi di Call e Put sono: "<<price_call
        <<" "<<price_put<<endl;

    cufftDestroy(forward); cufftDestroy(backward);
    cudaFree(in); cudaFree(out); cudaFree(cesp);

return 0.0;
}

// Prodotto di matrici complesse element-wise.
__global__ void Cproduct2D(cufftDoubleComplex *a,cufftDoubleComplex *b,
                           cufftDoubleComplex *c,int N,int M)
{
    int idx=threadIdx.x + (blockIdx.x)*(blockDim.x);
    int idy=threadIdx.y + (blockIdx.y)*(blockDim.y);
    double real,imag;
    if (idx < (N)  && idy < (M) ){
            real = a[M*idx+idy].x * b[M*idx+idy].x -
                   a[M*idx+idy].y * b[M*idx+idy].y;
            imag = a[M*idx+idy].x * b[M*idx+idy].y +
                   a[M*idx+idy].y * b[M*idx+idy].x;
```

```
        c[M*idx+idy].x=real;
        c[M*idx+idy].y=imag;
    }
}

// Applicazione della condizione al bordo per opzioni Barriera 2D.
__global__ void NormBarrier2D(cufftDoubleReal *a,int N,double S0,
                              double xmin,double dx,double D,double U)
{
    int idx=threadIdx.x + (blockIdx.x)*(blockDim.x);
    int idy=threadIdx.y + (blockIdx.y)*(blockDim.y);
    if (idx < N && idy<N ){
        if( S0*exp(xmin+dx*idx)<D || S0*exp(xmin+dx*idx)>U ||
            S0*exp(xmin+dx*idy)<D || S0*exp(xmin+dx*idy)>U ){
            a[idx*N+idy]=0.0;
        }
        else {
            a[idx*N+idy]=a[idx*N+idy]/(N*N) ;
        }
    }
}

// Funzione per il calcolo del prezzo di un'opzione Call Europea 2D,
// modello di Merton, metodo FST,
// payoff max(0,S1(T)+S2(T)-K).
double Call_FST_MJD_CUDA_2D(double S1,double S2,double K,double T,
                            double r,double sigma1,double lambda1,
                            double mu1,double delta1,double sigma2,
                            double lambda2,double mu2,double delta2,
                            double rho,int M)
{
    int N=pow(2,M);
    cufftHandle forward,backward;
    cufftDoubleComplex *out,*cesp;
    cufftDoubleReal *in;
    cudaMalloc((void **)&out, sizeof(cufftDoubleComplex)*N*(N/2+1));
```

```
cudaMalloc((void **)&cesp, sizeof(cufftDoubleComplex)*N*(N/2+1));
cudaMalloc((void **)&in, sizeof(cufftDoubleReal)*N*N);
double xmin=-10.0; double xmax=10.0;
double dx=(xmax-xmin)/N;
double* Call=new double[N*N];
double wmin=-M_PI/dx; double wmax=M_PI/dx;
double dw=(wmax-wmin)/N;
double gamma1=r-0.5*sigma1*sigma1-lambda1*(exp(0.5*delta1*delta1+mu1)-1);
double gamma2=r-0.5*sigma2*sigma2-lambda2*(exp(0.5*delta2*delta2+mu2)-1);
double complex * Phi=new double complex [N*(N/2+1)];
int dim_blocco=0;

if (M-5<1){
    dim_blocco=1;
}
else{
    dim_blocco=pow(2,M-5);
}
dim3 grid(dim_blocco,dim_blocco);
dim3 block(32,32); //2^10 thread per blocco

for(int i=0;i<N;i++){
    for(int k=0;k<(N/2+1);k++)
    {
        Call[i*N+k] = max(0.0,S1*exp(xmin+i*dx)+S2*exp(xmin+k*dx)-K);
        Phi[i*(N/2+1)+k]=-0.5*(sigma1*sigma1)*(k*dw*k*dw) +
            I*k*dw*gamma1 + lambda1*( cexp(-0.5*(delta1*delta1)*
            (k*dw*k*dw)+I*k*dw*mu1)-1)-0.5*(sigma2*sigma2)*
            (i*dw*i*dw)+I*i*dw*gamma2+lambda2*(cexp(-0.5*
            (delta2*delta2)*(i*dw*i*dw)+I*i*dw*mu2)-1)-rho*sigma1*
            sigma2*(k*dw)*(i*dw)    ;
        Phi[i*(N/2+1)+k]=cexp(T*(Phi[i*(N/2+1)+k]-r));
    }
}

for(int i=(N/2+1);i<N;i++){
```

```
    for(int k=0;k<(N/2+1);k++)
    {
        Phi[i*(N/2+1)+k]=-0.5*(sigma1*sigma1)*(k*dw*k*dw)+
        I*k*dw*gamma1 + lambda1*(cexp(-0.5*(delta1*delta1)*
        (k*dw*k*dw)+I*k*dw*mu1)-1)-0.5*(sigma2*sigma2)*
        ((wmin+(i-N/2)*dw)*(wmin+(i-N/2)*dw))+I*(wmin+(i-N/2)*dw)
        *gamma2+lambda2*(cexp(-0.5*(delta2*delta2)*((wmin+(i-N/2)
        *dw)*(wmin+(i-N/2)*dw))+I*(wmin+(i-N/2)*dw)*mu2)-1)-
        rho*sigma1*sigma2*(k*dw)*(wmin+(i-N/2)*dw);
        Phi[i*(N/2+1)+k]=cexp(T*(Phi[i*(N/2+1)+k]-r));
    }
}


for(int i=0;i<N;i++){
    for(int k=(N/2+1);k<N;k++)
    {
        Call[i*N+k]=max(0.0,S1*exp(xmin+i*dx)+S2*exp(xmin+k*dx)-K);
    }
}

cudaMemcpy(in,Call,sizeof(cufftDoubleReal)*N*N,
           cudaMemcpyHostToDevice);
cudaMemcpy(cesp,Phi,sizeof(cufftDoubleComplex)*N*(N/2+1),
           cudaMemcpyHostToDevice);
cufftPlan2d(&forward,N,N,CUFFT_D2Z);
cufftPlan2d(&backward,N,N,CUFFT_Z2D);

cufftExecD2Z(forward,in,out);
Cproduct2D<<<grid,block>>>(out,cesp,out,N,(N/2+1));
cufftExecZ2D(backward,out,in);

cudaMemcpy(Call,in,sizeof(cufftDoubleReal)*N*N,
           cudaMemcpyDeviceToHost);

for(int i=0;i<N;i++){
```

```
        for(int k=0; k<N ;k++)
        {
            Call[i*N+k]=Call[i*N+k]/(N*N);
        }
    }

    double price=Call[N*N/2+N/2];

    cufftDestroy(forward); cufftDestroy(backward);
    cudaFree(in); cudaFree(out); cudaFree(cesp);
    delete[] Call; delete[] Phi;

return price;

}

// Funzione per il calcolo del prezzo di un'opzione Call Barriera 2D,
// tipo Knock-and-Out (doppia barriera),
// modello di Merton, metodo FST,
// payoff max(0,S1(T)+S2(T)-K) se l'opzione rimane attiva.
double Call_FST_MJD_CUDA_Barr_2D(double S1,double S2,double K,double T,
                        double r,double sigma1,double lambda1,double mu1,
                        double delta1,double sigma2,double lambda2,
                        double mu2,double delta2,double rho,int M,
                        double D,double U,int Nt)
{

    int N=pow(2,M);
    double dt=T/Nt;
    cufftHandle forward,backward;
    cufftDoubleComplex *out,*cesp;
    cufftDoubleReal *in;
    cudaMalloc((void **)&out, sizeof(cufftDoubleComplex)*N*(N/2+1));
    cudaMalloc((void **)&cesp, sizeof(cufftDoubleComplex)*N*(N/2+1));
    cudaMalloc((void **)&in, sizeof(cufftDoubleReal)*(N)*N );
    double xmin=-8; double xmax=8;
```

```
double dx=(xmax-xmin)/N;
double* Call=new double[N*N];
double wmin=-M_PI/dx; double wmax=M_PI/dx;
double dw=(wmax-wmin)/N;
double gamma1=r-0.5*sigma1*sigma1-lambda1*(exp(0.5*delta1*delta1+mu1)-1);
double gamma2=r-0.5*sigma2*sigma2-lambda2*(exp(0.5*delta2*delta2+mu2)-1);
double complex * Phi=new double complex [N*(N/2+1)];
int dim_blocco=0;

if (M-5<1){
    dim_blocco=1;
}
else{
    dim_blocco=pow(2,M-5);
}
dim3 grid(dim_blocco,dim_blocco);
dim3 block(32,32); //2^10 thread per blocco

for(int i=0;i<N;i++){
    for(int k=0; k<(N/2+1) ;k++)
    {
        Call[i*N+k]=max(0.0,S1*exp(xmin+i*dx) +S2*exp(xmin+k*dx)-K);
        Phi[i*(N/2+1)+k]=-0.5*(sigma1*sigma1)*(k*dw*k*dw)+
        I*k*dw*gamma1+lambda1*( cexp(-0.5*(delta1*delta1)*(k*dw*k*dw)
        +I*k*dw*mu1)-1)-0.5*(sigma2*sigma2)*(i*dw*i*dw)+I*i*dw*gamma2
        + lambda2*(cexp(-0.5*(delta2*delta2)*(i*dw*i*dw)+I*i*dw*mu2)
        -1)-rho*sigma1*sigma2*(k*dw)*(i*dw) ;
        Phi[i*(N/2+1)+k]=cexp(dt*(Phi[i*(N/2+1)+k]-r));
    }
}

for(int i=(N/2+1);i<N;i++){
    for(int k=0; k<(N/2+1) ;k++)
    {
        Phi[i*(N/2+1)+k]=-0.5*(sigma1*sigma1)*(k*dw*k*dw)+
        I*k*dw*gamma1+lambda1*(cexp(-0.5*(delta1*delta1)*(k*dw*k*dw)
```

```
        +I*k*dw*mu1) -1)-0.5*(sigma2*sigma2)*((wmin+(i-N/2)*dw)*
        (wmin+(i-N/2)*dw))+I*(wmin+(i-N/2)*dw)*gamma2+lambda2*
        (cexp(-0.5*(delta2*delta2)*((wmin+(i-N/2)*dw)*(wmin+(i-N/2)
        *dw))+I*(wmin+(i-N/2)*dw)*mu2)-1)-rho*sigma1*sigma2*(k*dw)
        *(wmin+(i-N/2)*dw);
        Phi[i*(N/2+1)+k]=cexp(dt*(Phi[i*(N/2+1)+k]-r));
    }
}


for(int i=0;i<N;i++){
    for(int k=(N/2+1); k<N ;k++)
    {
        Call[i*N+k]=max(0.0,S1*exp(xmin+i*dx)+S2*exp(xmin+k*dx)-K);
    }
}

cufftPlan2d(&forward,N,N,CUFFT_D2Z);
cufftPlan2d(&backward,N,N,CUFFT_Z2D);

cudaMemcpy(in,Call,sizeof(cufftDoubleReal)*N*N,
           cudaMemcpyHostToDevice);
cudaMemcpy(cesp,Phi,sizeof(cufftDoubleComplex)*N*(N/2+1),
           cudaMemcpyHostToDevice);

for (int i=0;i<Nt;i++){
    cufftExecD2Z(forward,in,out);
    Cproduct2D<<<grid,block>>>(out,cesp,out,N,(N/2+1) );
    cufftExecZ2D(backward,out,in);
    NormBarrier2D<<<grid,block>>>(in,N,S0,xmin,dx,D,U);
}
cudaThreadSynchronize();

cudaMemcpy(Call,in,sizeof(cufftDoubleReal)*N*N,
           cudaMemcpyDeviceToHost);
```

```
    double price=Call[N*N/2+N/2];

    cufftDestroy(forward); cufftDestroy(backward);
    cudaFree(in); cudaFree(out); cudaFree(cesp);
    delete[] Call; delete[] Phi;

return price;
}
```

Bibliografia

[1] M. Attari, *Option Pricing Using Fourier Transforms: A Numerically Efficient Simplification*. Working Paper disponibile su SSRN, 2004.

[2] P. Baldi, *Equazioni Differenziali Stocastiche e Applicazioni*. Quaderni dell'Unione Matematica Italiana, Vol. 28, Pitagora Editrice, 2000.

[3] J. Bect, *A Unifying Formulation of the Fokker-Planck-Kolmogorov Equation for General Stochastic Hybrid Systems*. Nonlinear Analysis: Hybrid Systems, Vol. 4, Issue 2, pp. 357-370, 2010.

[4] A. Bentata e R. Cont, *Forward equations for option prices in semimartingale models*. Université Pierre et Marie Curie - Paris VI, Working Paper, 2011.

[5] T. Björk, *Arbitrage Theory in Continuous Time*. Oxford Finance, 2009.

[6] T. Bollerslev, M. Gibson e H. Zhou, *Dynamic estimation of volatility risk premia and investor risk aversion from option-implied and realized volatilities*. Journal of Econometrics, Vol. 160, Issue 1, pp. 235-245, 2011.

[7] P.W. Buchen, O. Konstandatos e T.J. Kyng, *Images and Barriers on the road to Real Options Valuation*. UTS ePress, 2009.

[8] P.W. Buchen, *Pricing European Barrier Options*. University of Sydney, Technical Report 25, 1996.

[9] P.P. Carr e D.B. Madan, *Option valuation using the Fast Fourier Transform*. Journal of Computational Finance, Vol. 2, pp. 61-73,, 1999.

[10] G. Cheang, C. Chiarella e A. Ziogas, *An Analysis of American Options under Heston Stochastic Volatility and Jump-Diffusion Dynamics*. UTS, Research Paper 256, 2009.

[11] C. Chiarella, B. Kang, G.H. Meyer e A. Ziogas, *The Evaluation of American Option Prices under Stochastic Volatility and Jump-Diffusion Dynamics using the Method of Lines*. International Journal of Theoretical and Applied Finance, Vol. 12, No. 3, pp. 393-425, 2009.

[12] K. Chourdakis, *Switching Lévy Models in Continuous Time: Finite Distributions and Options Pricing*. University of Essex, Working Paper disponibile su SSRN, 2005.

[13] R. Cont e P. Tankov, *Financial Modelling with Jump Processes*. Chapman & Hall/CRC Financial Mathematics Series, 2004.

[14] Y. d'Halluin, P.A. Forsyth e K.R. Vetzal, *Robust Numerical Methods for Contingent Claims under Jump Diffusion Processes*. IMA Journal of Numerical Analysis 25, pp. 87-112, 2005.

[15] M.H.A. Davis, *The Dupire Formula*. Imperial College London, Finite Difference Methods Course material, 2011.

[16] D. Ding, *Fourier Transform Methods for Option Pricing*. Chapter 11, Fourier Transform Applications, InTech, pp. 265-290, 2012.

[17] A.A. Drăgulescu e V.M. Yakovenko, *Probability distribution of returns in the Heston model with stochastic volatility*. University of Maryland, Working Paper, 2002.

[18] F. Fang e C.W. Oosterlee *Pricing Early-Exercise and Discrete Barrier Options by Fourier-Cosine Series Expansions*. Numerische Mathematik, Vol. 114, Issue 1, pp. 27-62, Springer, 2009.

[19] F. Fang e C.W. Oosterlee *A Fourier-based Valuation Method for Bermudan and Barrier Options under Heston's Model*. SIAM Journal on Financial Mathematics, Vol. 2, Issue 1, pp. 439-463, 2010.

[20] F. Fang *The COS Method: An Effcient Fourier Method for Pricing Financial Derivatives, PhD thesis*. Friedrich-Alexander-Universität Erlangen-Nürnberg, 2010.

[21] I. Florescu, R. Liu, M.C. Mariani e G. Sewellx *Numerical Schemes for Option Pricing in Regime-Switching Jump Dffusion Models*. International Journal of Theoretical and Applied Finance, Vol. 16, Issue 8, 2013.

[22] I. Florescu, R. Liu e M.C. Mariani, *Solutions to a Partial Integro-Differential Parabolic System Arising in the Pricing Of Financial Options in Regime-Switching Jump Diffusion Models*. Electronic Journal of Differential Equations, No. 231, pp. 1-12., 2012.

[23] M. Frigo e S.G. Johnson, *FFTW3*. http://www.fftw.org/, 2014.

[24] J. Gatheral, *The Volatility Surface-A Practitioner's Guide*. Wiley Finance, 2006.

[25] G. Gilardi, *Analisi tre*. McGraw-Hill Libri Italia, 1994.

[26] L. Grzelak e C.W. Oosterlee, *On The Heston Model with Stochastic Interest Rates*. SIAM Journal on Financial Mathematics, Vol. 2, Issue 1, pp. 255-286, 2011.

[27] L.A. Grzelak, C.W. Oosterlee e S. van Weerenb, *Extension of Stochastic Volatility Equity Models with Hull-White Interest Rate Process*. Working Paper disponibile su SSRN, 2009.

[28] C. Guardasoni e S. Sanfelici, *Fast Numerical Pricing of Barriers Options under Stochastic Volatility and Jumps*. Working Paper, Università di Parma.

[29] S. Guo, L.A. Grzelak e C.W. Oosterlee, *Analysis of an affine version of the Heston-Hull-White option pricing partial differential equation*. Applied Numerical Mathematics, pp. 143-159, 2013.

[30] S.L. Heston, *A Closed-Form Solution for Options with Stochastic Volatility with Apllications to Bond and Currency Options*. The Review of Financial Studies, Vol. 6, Issue 2, pp. 327-343, 1993.

[31] J.E. Ingersoll Jr., *Theory of Financial Decision Making*. Rowman & Littlefiled, 1987.

[32] P. Jäckel, *Monte Carlo Methods in Finance*. Wiley Finance, 2002.

[33] K.R. Jackson, S. Jaimungal e V. Surkov, *Fourier Space Time-stepping for Option Pricing with Lévy Models*. University of Toronto, Working Paper, disponibile su SSRN, 2007.

[34] K.R. Jackson, S. Jaimungal e V. Surkov, *Option Pricing with Regime Switching Lévy Processes using Fourier Space Time-stepping*. University of Toronto, Working Paper, disponibile su SSRN, 2007.

[35] J. Kienitz e D. Wetterau, *Financial Modelling: Theory, Implementation and Practice with MATLAB Source*. Wiley Finance, 2012.

[36] S.B. Lippman, J. Lajoie e B.E. Moo, *C++ Primer*. Addison Wesley, 2005.

[37] R. Lord, F. Fang, F. Bervoets e C.W. Oosterlee, *A Fast Method for Pricing Early-Exercise Options with the FFT*. Springer-Verlag Berlin Heidelberg, 2007.

[38] R. Lord, F. Fang, F. Bervoets e K. Oosterlee, *A Fast and Accurate FFT-Based Method for Pricing Early-Exercise Options under Lévy Processes*. MPRA, Working Paper, 2007.

[39] R. Lord, *Efficient Pricing Algorithms for Exotic Derivatives, PhD thesis*. Erasmus Universiteit Rotterdam, 2008.

[40] D.Marazzina, G.Fusai e G.Germano, *Pricing Credit Derivatives in a Wiener-Hopf Framework*. All'interno di: *Topics in Numerical Methods for Finance*, Springer Proceedings in Mathematics & Statistics, Vol. 19, pp. 139-154, Springer-Verlag, 2012.

[41] G.H. Meyer e J. van der Hoek, *The Evaluation of American Options with the Method of Lines*. Working Paper, disponilie su mathnet.or.kr, 1994.

[42] H.T. Nyquist e C.E. Shannon, *Teorema del campionamento di Nyquist-Shannon*. Wikipedia.org.

[43] NVIDIA Corporation, http://www.nvidia.it/object/cuda-parallel-computing-it.html.

[44] NVIDIA Corporation, *CUFFT Library User's Guide*. Nvidia CUDA Guide, disponilie su developer.nvidia.com, 2013.

[45] NVIDIA Corporation, *CUDA C Best Practices Guide*. Nvidia CUDA Guide, disponilie su developer.nvidia.com, 2013.

[46] NVIDIA Corporation, *CUDA C Programming Guide*. Nvidia CUDA Guide, disponilie su developer.nvidia.com, 2013.

[47] F.D. Rouah, *The derivation of local volatility*. www.javaquant.net, 1998.

[48] M. Schmelzle, *Option Pricing Formulae using Fourier Transform: Theory and Application*. Working Paper, disponibile su pfadintegral.com, 2010.

[49] R.U. Seydel, *Tools for Computational Finance*. Springer, 2012.

[50] V. Surkov, *Option Pricing using Fourier Space Time-stepping Framework, PhD thesis*. University of Toronto, Working Paper, disponibile su SSRN, 2009.

[51] V. Surkov, *Parallel Option Pricing with Fourier Space Time-stepping Method on Graphics Processing Units*. University of Toronto, Working Paper, disponibile su SSRN, 2007.

[52] G. Winkler, T. Apel e U. Wystup, *Valuation of Options in Heston's Stochastic Volatility Model Using Finite Element Methods*. Foreign Exchange Risk, Risk Publications, 2001.

[53] D. Yang, *C++ and Object-Oriented Numeric Computing for Scientists and Engineers*. Springer, 2001.

[54] B. Zhang e C.W. Oosterlee, *Acceleration of Option Pricing Technique on Graphics Processing Units*. Processing Units-Concurrency and Computation: Practice and Experience, Wiley, 2010.