Progetto per il pricing di un'opzione

Corso di metodi computazionali della fisica II

Modulo di econofisica

Anno accademico 2009/2010

Mariacristina Romano Gianmaria Enea Roat

20 Aprile 2010

Indice

Capitolo 1

Introduzione

In questo progetto presentiamo un'esempio di pricing di un'opzione il cui sottostante sia un'azione, eseguito tramite la simulazione Montecarlo. Abbiamo affrontato il problema in esame realizzando una libreria scritta in codice c++, sviluppata in classi.

In particolare ci occupiano di una precisa opzione, abbastanza complessa, definita nel secondo capito. Ad ogni modo, la libreria realizzata per effettuare il princing puó essere estesa in maniera molto naturale ad una qualsiasi altra opzione, di tipo path-dependent o meno.

Nel terzo capitolo é descritta brevemente tutta la struttura della libreria e sono presenti alcune direttive per l'utilizzo. Per maggiori dettagli rimandiamo al reference della libreria. In questo capitolo ci sono anche alcuni primi test sulla corretta implementazione.

Abbiamo inoltre studiato analiticamente il particolare tipo di opzione in esame, mostrando come matematicamente si possano ottenere alcune prime previsioni qualitative (e quantitative ai limiti) riguardo al pricing dato da simulazioni ad estrazione stocastica.

Infine, nell'ultimo capitolo, mostriamo i risultati ottenuti, validati da alcuni controlli fatti sulle variabili-test ottenute nello studio analitico.

Capitolo 2

Il contratto opzionale (presentazione del problema)

Il nostro progetto verte su un particolare tipo di opzione esotica, di tipo path-dependent, ad esercizio europeo e di tipo call. Supporremo implicitamente che il prezzo $S(t)$ dell'azione segua un processo log-normale, con μ costante e volatilitá σ costante. Questo é l'usuale modello browniano utilizzato in finanza. Inoltre porremo sempre il tasso di attualizzazione del denaro (riskfree) costante ($r(t) = r = 0.02$).

2.1 L'opzione in esame

Chiamiamo S la funzione definente il valore dell'azione sottostante la nostra opzione. Ovviamente sará in funzione del tempo e dunque $S(t)$. Definiamo $t = 0$ l'istante attuale e T il tempo di maturitá della nostra opzione. Considereremo sempre azioni dal valore iniziale $S(0) = 100$. Il tempo totale T é diviso in maniera uniforme in m intervallini δ_i. Il tempo in generale é valutato in frazioni di anni. Lo strikeprice sará indicato con E.

Il payoff dell'opzione é definito come segue:

$$\text{Payoff}_{\text{performance with corridor}} = 1 \text{ euro} \cdot Max[P - E, 0] \tag{2.1}$$

dove la variabile P é definita come sommatoria:

$$P = \sum_{i=0}^{m-1} P_i \tag{2.2}$$

Ognuno dei P_i é ottenuto da: ($t_i = iT/m$)

$$P_i = Min[\ Max[\ \frac{S(t_{i+1})}{S(t_i)} - 1,\ l]\ u] \tag{2.3}$$

l e u sono funzioni definite come:

$$l = \frac{e^{r\delta t}}{k} - 1 \tag{2.4}$$

$$u = ke^{r\delta t} - 1 \tag{2.5}$$

E la funzione k é ottenuta da

$$k = 1 + \lambda\sqrt{\delta_t} \tag{2.6}$$

Dove anche λ é un dato iniziale.

2.2 Prime considerazioni

Nell'opzione definita sopra, le funzione l e u (parametrizzate dalla funzione k) definiscono una sorta di corridoio (due rette parallele nel caso di dati iniziali T, λ, E, m tutti fissati) entro e oltre il quale il rapporto scalato $\frac{S(t_{i+1})}{S(t_i)} - 1$ varia. Per come sono definiti i P_i, se questo rapporto supera (in positivo o in negativo) il corridoio, la sommatoria P ottiene il contributo massimo o minimo definito dal corridoio. Il corridoio dunque limita in modulo la variazione della sommatoria elemento per elemento.

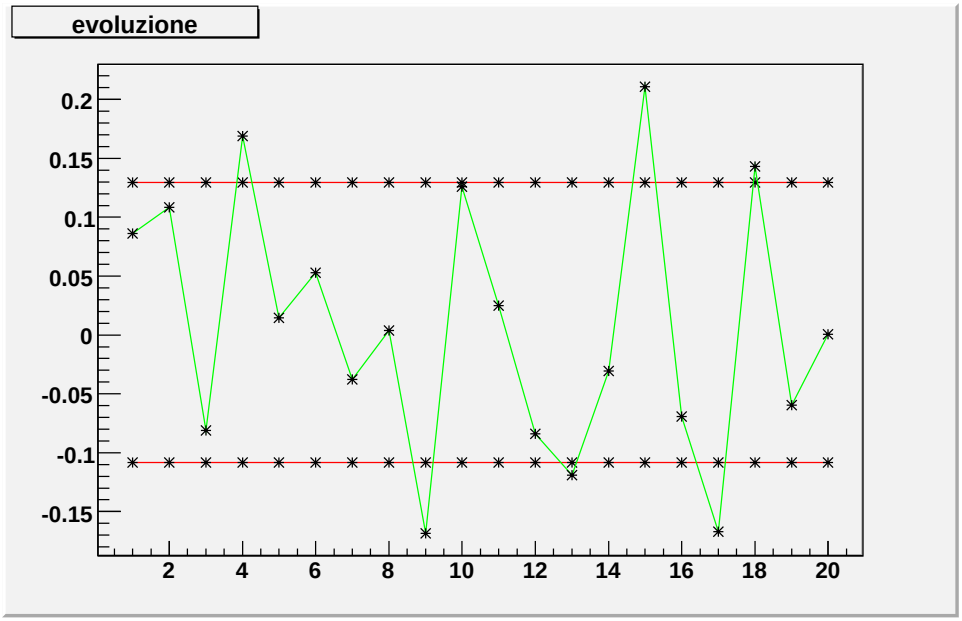

Figura 2.1: In questo grafico é mostrato un esempio di movimento dell'azione (funzione $S(t)$ nell'intervallo $[0 : T = 20]$ con $\delta_t = 1$ e volatilitá $\sigma = 0.3$) sovrapposto a uno dei possibili corridoi in esame (parametro $\lambda = 0.3$). Qualora l'azione superasse le linee del corridoio, il contributo extra non verrebbe preso in considerazione.

Capitolo 3

Libreria in c++

Per affrontare il problema e utilizzare il metodo Montecarlo, abbiamo realizzato una libreria scritta in codice c++. In questo capitolo esporremo brevemente i contenuti.

3.1 Presentazione

Diamo qui una breve descrizione delle classi, ordinate secondo il loro livello di cardinalitá. Per classi di cardinalitá zero s'intendono classi standard del c++ o, piú in generale, classi esterne alla nostra libreria non realizzate da noi. Le classi di livello uno sono classi create da noi che vengono implementate con sole classi di livello zero. Le classi di livello due sono implementate a partire da classi di livello uno ed eventualmente di livello zero. In generale classi di livello n contengono classi di ordine n-1 o piú basse.

Si veda il reference della libreria per una descrizione piú dettagliata di ogni singola classe.

3.1.1 Livello uno

Le classi del livello uno rappresentano sostanzialmente concetti reali tradotti in linguaggio informatico. Abbiamo dunque il corrispettivo della valuta (Currency), di un'opzione (Option), di un numero casuale (N_rand), di un processo generico (Process). Di livello uno sono anche le classi Statistica e Errorlist.

classe N_rand

Classe che rappresenta un contenitore di numeri generati casualmente.

classe Currency

Classe che rappresenta la valuta. É settata in modo tale da avere l'Euro (EUR) come valuta di default.

classe Option

Classe astratta che rappresenta genericamente un'opzione. Ogni tipo di opzione verrá ovviamente derivata da questa, implementando la funzione virtual pura definente il payoff.

classe Process

Classe rappresentativa di un generico processo.

classe Statistica

Classe che verrá utilizzata nella simulazione Montecarlo per avere appunto una statistica dei dati. Calcola errore e valor medio.

classe Errorlist (singleton)

Classe che rappresenta la coda di stampa degli errori di implementazione. Ogni volta che la libreria viene utilizzata erroneamente, viene inviato un messaggio di errore alla classe Errorlist che stamperá sul file lista_errori.dat l'elenco degli errori. Essendo un singleton, il costruttore é privato. Si evita in tal modo che venga costruito piú di un oggetto di tipo Errorlist, che sarebbe evidentemente un inutile spreco di memoria, e tutti i messaggi di errori vengono quindi gestiti da un unico oggetto.

3.1.2 Livello due

Anche nel livello due della libreria troviamo sostanzialmente concetti comuni espressi in forma di classe. La differenza rispetto al livello uno é che qui le classi rapprentative di istanti temporali (Time), di intervallo temporale (Deltatime), del prezzo (Price), i generatori di numeri casuali (Rand_gen e Rand_gen_root) e la classe Stima_montecarlo hanno inclusa la classe per la stampa degli errori (Errorlist).

classe Time

Classe che rappresenta un istante temporale, quindi una data e un'orario. Abbiamo trattato questa classe con particolare cura, implementando funzioni di controllo per gli anni bisestili, metodi e operatori di interazione con le altri classi temporali (in particolare la Deltatime), prestando attenzione al differente numero di giorni dei mesi.

classe Deltatime

Classe che rappresenta un intervallo temporale espresso in anni, mesi giorni, ore, minuti e secondi. Importante la ridefinizione degli operatori che permette di trattare un intervallo temporale come frazione di anni espressa da un double.

classe Price

Classe che rappresenta un prezzo, con un puntatore all'opportuna Currency.

classe Rand_gen

Classe astratta, madre dei generatori casuali.

classe Rand_gen_root (singleton)

Classe derivata dalla Rand_gen. Abbiamo scelto di formulare anche questa classe come singleton, servendo nella pratica un unico generatore di numeri casuali, ottimizzando l'utilizzo della libreria. Abbiamo qui utilizzato direttamente le classi di Root (CERN).

classe Stima_montecarlo

Classe di gestione dei risultati della simulazione montecarlo. Contiene infatti un valore e il corrispettivo errore. Utile quando una simulazione valuta piú di una grandezza contemporaneamente.

3.1.3 Livello tre

Al secondo livello troviamo le classi finanziarie elementari. Rappresentiamo qui uno schedulatore temporale (Timestruct), classi definenti la volatilitá (Volatility e Vol_std) e le classi per le curve dei tassi (Yield_curve, Yield_curve_flat, Yield_curve_term_struct).

classe Timestruct

Classe che contiene un vettore di Deltatime (e la dimensione). Rappresenta una struttura temporale, permette una migliore e piú sicura gestione dei vettori di intervalli di tempi che verranno ampliamente usati all'interno della libreria.

classe Volatility

Classe astratta che traduce il concetto di volatilitá di un'azione. Si trova al secondo livello della libreria perché la sua funzione principale viene implementata a partire dal concetto temporale espresso dalla Time.

classe Vol_std

Classe figlia della Volatility. Rappresenta il concetto di volatilitá fissa (indipendente dal tempo).

classe Yield_curve

Classe astratta. Rappresenta genericamente una curva dei tassi, con implementate le funzioni per ottenere il forward rate e il discount factor per attualizzare un prezzo. É virtual pura invece la funzione per ottenere il tasso della curva in corrispondenza di un Time.

classe Yield_curve_flat

Classe figlia della Yield_curve. Rappresenta una curva di tipo flat, quindi con un tasso fissato indipendente dal tempo.

classe Yield_curve_term_struct

Classe figlia della Yield_curve. Rappresenta una curva dei tassi complessa, costruita sull'interpolazione di dati.

3.1.4 Livello quattro

Al quarto livello troviamo l'anagrafica dell'opzione (Eq_description).

classe Eq_description

Classe per l'anagrafica di un'azione. Contiene il nome dell'azione, il codice, la curva di riferimento e la volatilitá dell'azione.

3.1.5 Livello cinque

Al quinto livello troviamo la classe rappresentativa del valore di un'azione (Eq_price).

classe Eq_price

Classe per la gestione del valore di un'azione ad un determinato istante Time. É legata all'anagrafica dell'azione corrispondente tramite un puntatore.

3.1.6 Livello sei

Il sesto livello é costituito dalle classi rappresentative dei processi azionari (in campo finanziario). Troviamo quindi la Eq_process e le varie classi derivate.

classe Eq_process

Classe astratta figlia della Process. Permette l'evoluzione di un'azione da uno stato A ad uno stato B. Contiene anche il metodo per ottenere il numero casuale da inserire nella formula di evoluzione.

classe Eq_process_lognormal_eulero

Classe figlia della Eq_process. É implementata la funzione di evoluzione tramite il metodo di Eulero per un processo lognormale. I numeri casuali da inserire nella formula provengono da un'estrazione gaussiana di media nulla e varianza unitaria.

classe Eq_process_lognormal_esatto

Classe figlia della Eq_process. É implementato il metodo di evoluzione tramite la funzione integrale esatta per un processo lognormale. I numeri casuali da inserire nella formula provengono da un'estrazione gaussiana di media nulla e varianza unitaria.

classe Eq_process_lognormal_binary

Classe figlia della Eq_process_lognormal_esatto. Eredita da questa la formula di evoluzione con la sola differenza che la generazione di numeri casuali viene reimplementata secondo un'estrazione binaria.

3.1.7 Livello sette

Il settimo livello della libreria é occupato dalla classe dei cammini evolutivi di un'azione (Path).

classe Path

Questa classe rappresenta il cammino di evoluzione di un'azione, quindi un vettore di Eq_price. Tra i membri privati compare anche la Timestruct di evoluzione (che contiene la dimensione del vettore) e il puntatore al processo con il quale l'azione é stata evoluta. Un'esempio di cammino é stato mostrato nel capitolo precedente nella figura 2.1.

3.1.8 Livello otto

Viene introdotto al livello otto il concetto di opzione basata su un sottostante di tipo azione.
Dalla classe Eq_option vengono derivate la classe Eq_op_plainvanilla, la classe Eq_op_w e la classe Eq_op_performance_with_corridor.

classe Eq_option

Classe astratta (figlia della Option) che rappresenta un'opzione. É caratterizzata dalla funzione che restituisce il payoff dato un cammino Path.

classe Eq_op_plainvanilla

Classe figlia della Eq_option. Rappresenta un opzione di tipo plainvanilla, con la funzione di payoff implementata opportunamente.

classe Eq_op_w

Classe figlia della Eq_option. Rappresenta un sorta di opzione con la funzione di payoff implementata con un algoritmo di controllo per il generatore di numeri casuali.

classe Eq_op_performance_with_corridor

Classe figlia della Eq_option. Rappresenta l'opzione del problema in esame.

3.1.9 Livello nove

Il passo successivo é il pricer. Abbiamo infatti ricostruito con il c++ tutti i concetti necessari alla descrizione del problema di tipo finanziario che ci é stato assegnato. Passiamo ora ad affrontare il problema con il metodo Montecarlo grazie alla classe Eq_pricer_montecarlo.

classe Eq_pricer

Classe astratta che rappresenta genericamente un metodo di pricing di un'opzione.

classe Eq_pricer_montecarlo

Classe figlia della Eq_pricer che implementa il metodo Montecarlo per il pricing di un'opzione con un numero N di simulazioni passato al costruttore. Tra i membri privati vi sono 3 oggetti di tipo Stima_montecarlo che permettono di salvare in memoria i risultati della simulazione sia sui cammini che sugli anticammini.

3.1.10 Livello dieci

Il decimo livello permette un rapido utilizzo dell'intera libreria.

classe Pricing

Classe ad hoc per il tema d'esame. Grazie a questa classe possiamo, passando al costruttore i dati iniziali, ottenere direttamente i vari risultati della simulazione stampati su un file di dati.

3.2 Utilizzo della libreria

Esponiamo qui alcuni aspetti della nostra libreria che vogliamo mettere in evidenza.

3.2.1 La gestione del tempo

Abbiamo prestato molta cura durante l'implementazione della libreria alla gestione del tempo, tematica molto delicata che ci é sembrato importante trattare in maniera sicura e precisa.

Abbiamo creato tre classi ad hoc per questo tema: Time, Deltatime e Timestruct.

La classe Time rappresenta un istante temporale (in anni, mesi, giorni, ore, minuti e secondi). Ovviamente ognuna di queste grandezze é controllata in costruttori e funzioni di modifica, cosí da impedire dichiarazioni errate di istanti temporali (tenendo conto della bisestilitá degli anni).

La classe Time da sola peró non esaurisce completamente i problemi di gestione del tempo. Infatti in finanza si incontra molto spesso il concetto di intervallo temporale. La classe Deltatime nasce appunto dalla necessitá di utilizzare per la rappresentazione del tempo intervalli e non solo istanti.

Le due classi sono ovviamente collegate. Intuitivamente:

$$\text{Time - Time} = \text{Deltatime}$$
$$\text{Time + Deltatime} = \text{Time}$$
$$\text{Deltatime + Deltatime} = \text{Deltatime}$$

Abbiamo quindi fatto overloading degli operatori permettendo la corretta interazione.

Nasce poi spontanea nell'affrontare il problema in esame la necessitá di dichiarare vettori di intervalli temporali. Per gestire in maniera ottimale questa tematica (vettore di Deltatime e la sua dimensione) abbiamo creato una classe, la classe Timestruct, che rappresenta appunto una struttura temporale.

3.2.2 L'evoluzione dell'azione

Per risolvere il problema in esame avevamo necessitá di simulare (N volte) il possibile andamento di una determinata azione (a volatilitá σ fissata e valore iniziale $S(0)$ fissato).

L'andamento dell'azione é stato sviluppato grazie a due formule.

La prima consiste nel metodo di Eulero:

$$S_i = S_{i-1} + S_{i-1}(r\Delta t + \sigma\sqrt{\Delta t}w) \tag{3.1}$$

Dove compare un Δt che si presume piccolo.

Il secondo consiste nella formula integrale esatta:

$$S_i = S_{i-1}e^{(r-\frac{\sigma^2}{2})\delta_t + \sigma\sqrt{\delta_t}w} \tag{3.2}$$

In entrambe le formule, w é una random variabile estratta da una distribuzione gaussiana di media nulla e varianza unitaria.

Il metodo di Eulero é un'approssimazione della formula integrale esatta. Possiamo infatti notare che consiste, a meno di un fattore correttivo $\frac{\sigma^2}{2}\delta_t$, nello sviluppo al primo ordine dell'esponenziale. Proprio per questo motivo il Δt deve essere sufficientemente piccolo perché i risultati siano corretti.

3.2.3 I Singleton

Il Singleton é un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

Utilizzare questa tecnica ci ha permesso un'ottimizzazione della libreria migliorando le prestazioni di un eventuale programma che la utilizza.

É evidente che all'interno di un programma, é inutile creare piú di un generatore di numeri casuali dello stesso tipo. Anzi, possiamo definirlo uno spreco di memoria e una quantitá (variabile) di lavoro inutile a carico del processore.

Lo stesso discorso vale per la classe ad hoc per la gestione degli errori nell'utilizzo della libreria. É infatti poco sensato che la lista di errori (che é unica e stampata su un solo file di testo) venga gestita da piú oggetti contemporaneamente.

Molto piú logico é invece permettere che venga istanziato un solo oggetto di tipo Errorlist, gestendo direttamente la chiamata del costruttore della classe (che si trova tra i membri privati) tramite un'unica funzione di getting, munita di controllo che impedisce la riallocazione del puntatore statico (membro privato) che punta all'istanza, unica, che rappresenta la nostra lista di errori.

3.2.4 I puntatori statici

Abbiamo utilizzato una variazione del design pattern Singleton per risolvere le tematiche quali l'oggetto rappresentante l'istante attuale e la curva relativa al tasso risk free.

In effetti, anche in questo caso parliamo di oggetti che dovrebbero essere unici nella loro dichiarazione. La differenza sta peró nel fatto che il loro tipo é comune a molti altri.

L'istante attuale é rappresentato da un oggetto di tipo Time (classe ad hoc per gli istanti temporali) ma l'oggetto che rappresenta l'istante attuale non deve poter essere ridichiarato, dev'essere unico in tutta la libreria.

Allo stesso modo supponiamo che la curva dei tassi risk free sia unica in tutta la libreria, rappresentata da un oggetto di tipo Yield_curve.

Abbiamo dunque inserito in entrambe le classi alcune funzioni specifiche che permettono l'istanziamento di oggetti puntati da link statici. Ovviamente il costruttore rimane unico e non ci sono altre variazioni all'interno della classe.

In questo modo abbiamo ottenuto una gestione appropriata di entrambe le tematiche.

3.2.5 Il polimorfismo

La struttura della nostra classe risulta abbastanza complessa, sviluppata su 9 livelli. Questo é in parte dovuto all'ampio utilizzo di polimorfismo che ha reso il nostro lavoro piú elastico e piú agevolmente ampliabile.

Grazie all'ereditarietá, abbiamo sviluppato per quasi ogni classe necessaria alla risoluzione del problema, due o piú livelli tramite la dichiarazione di classe madre (generalmente astratta con virtual pure) e classe/i figlia/e.

In questo modo é possibile aggiungere classi derivate alla nostra libreria e utilizzarle senza dover fare ulteriori modifiche.

Le strutture gerarchiche delle classi sono riportate nel reference della libreria.

3.3 Test della libreria

Prima di affrontare il problema di valutazione dell'opzione in esame verifichiamo che l'implementazione del metodo Montecarlo e della libreria in generale siamo corretti. A tal proposito proponiamo alcuni metodi di test delle funzioni delle classi sulle quali si baserá poi la nostra valutazione dell'opzione.

3.3.1 Opzione w (controllo del generatore di numeri casuali)

Sfruttando il polimorfismo, abbiamo creato la classe Eq_op_w (derivandola dalla classe Eq_option) con la quale possiamo testare il generatore

di numeri casuali e in generale la simulazione montecarlo. Avendo infatti definito il payoff come:

$$\text{Payoff} = \sum_i w_i^2 \tag{3.3}$$

dove w_i é la random variabile che compare nella formula (3.1), dunque un numero estratto da una distribuzione gaussiana con $< w >= 0$ e $< w^2 >= 1$ che viene estratto per ogni singola simulazione m volte. Abbiamo poi scelto un numero alto di simulazioni (30000) e un vasto range del numero m di intervalli [2:200] cosí da poter considerare la sommatoria come serie infinita. Nel limite infatti

$$<\text{Payoff}> = m * e^{-rT} \tag{3.4}$$

Verifichiamo dunque che il generatore di numeri casuali, nel nostro caso la classe singleton Rand_gen_root costruita sulle classi di Root(CERN), e la classe che utilizza il metodo Montecarlo, la Eq_pricer_montecarlo, funzionino correttamente.

Attualizziamo il payoff cosí da vedere in grafico una retta di coefficiente angolare unitario (questo calcolo viene fatto in automatico all'interno della classe Pricing).

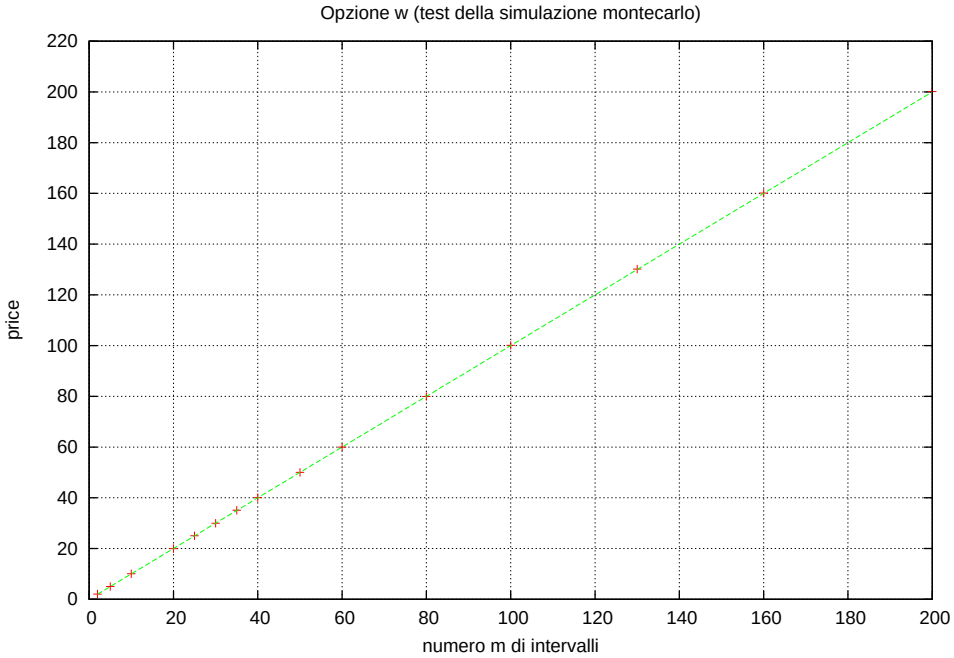

Figura 3.1: Grafico di test sulla bontá dell'estrazione gaussiana e dell'implementazione del metodo Montecarlo.

Come previsto, abbiamo un'ottima regressione lineare, dove il Payoff coincide ad ogni valutazione con il numero m di intervalli. Valori ed er-

rori sono riportati in tabella.

n	m	price	err%
1	2	2.017	0.008
2	5	5.019	0.013
3	10	10.034	0.018
4	20	19.94	0.03
5	25	25.03	0.03
6	30	29.96	0.03
7	35	35.08	0.03
8	40	40.05	0.04
9	50	50.018	0.04
10	60	59.99	0.05
11	80	79.96	0.05
12	100	100.10	0.06
13	130	130.17	0.07
14	160	160.14	0.07
15	200	200.16	0.08

3.3.2 Payoff di controllo (Plainvanilla)

Testiamo nuovamente la nostra libreria con una funzione per la quale possiamo prevedere i risultati. Definiamo il payoff come

$$\text{Payoff} = S(T) \tag{3.5}$$

ovvero esattamente il valore del sottostante al tempo T (maturity time). Si tratta sostanzialmente di un'opzione di tipo plainvanilla, con strikeprice fissato a zero.

In questo caso, con semplici passaggi algebrici :

$$e^{-rT} < \text{payoff} >= e^{-rT} < S(T) >= e^{-rT} e^{rT} S(0) = S(0) \tag{3.6}$$

e ci aspettiamo dunque un price coincidente con il valore iniziale del sottostante (su un numero N grande di simulazioni).

In grafico l'errore percentuale che otteniamo dalla simulazione (in funzione del maturity time).

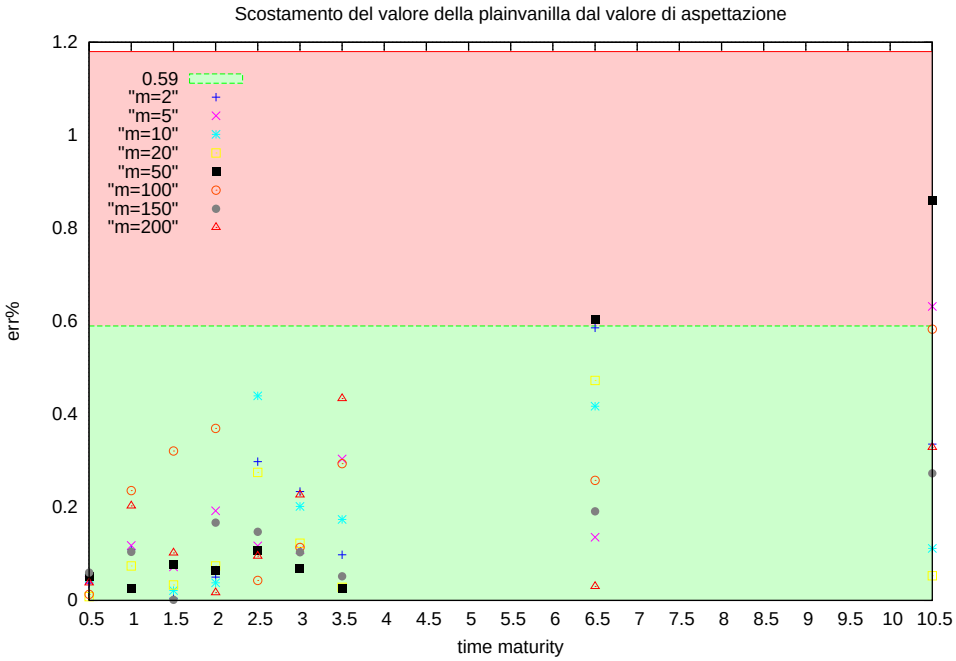

Figura 3.2: Grafico test che mostra di quanto il valore simulato di un'opzione plainvanilla con strikeprice fissato a 0 si discosta dal valore atteso. In dati che ricadono nella zona verde rientrano in una standard deviation. Nella zona rossa invece i dati che ricadono in 2 deviazioni standard.

Nel grafico si puó notare come la libreria da noi creata superi anche questo test. Infatti il valore dell'opzione tende al valore iniziale dell'azione con errori che rientrano sempre in due deviazioni standard e che crescono all'aumentare del maturity time.

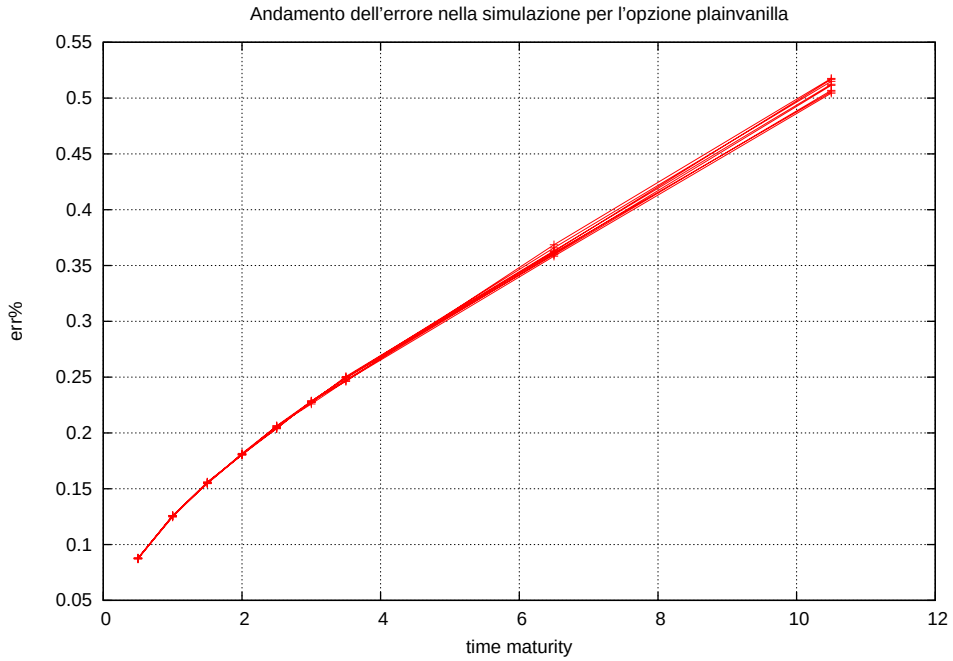

Figura 3.3: Andamento dell'errore nella simulazione della plainvanilla. Ogni curva rappresenta un m fissato ma la dipendenza da questo parametro risulta molto piccola.

Il valore dell'errore sale all'aumentare del maturity time e giustifica il grafico precedente dove i valori di price piú lontani dal valore iniziale dell'azione si verificavano appunto per valori di maturity time alti.

3.4 Valutazione degli errori

Dopo questi test preliminari controlliamo l'andamento dell'errore in funzione del numero di simulazioni. Sappiamo infatti dalla teoria che in questo metodo l'errore deve decrescere con l'aumentare del numero N di simulazioni asintoticamente a $\frac{1}{\sqrt{N}}$.

3.4.1 Errori nella simulazione Montecarlo

Abbiamo implementato all'interno della classe Pricing un costruttore che stampa su file gli errori nelle valutazioni del price di un'opzione in funzione del numero N di simulazioni.

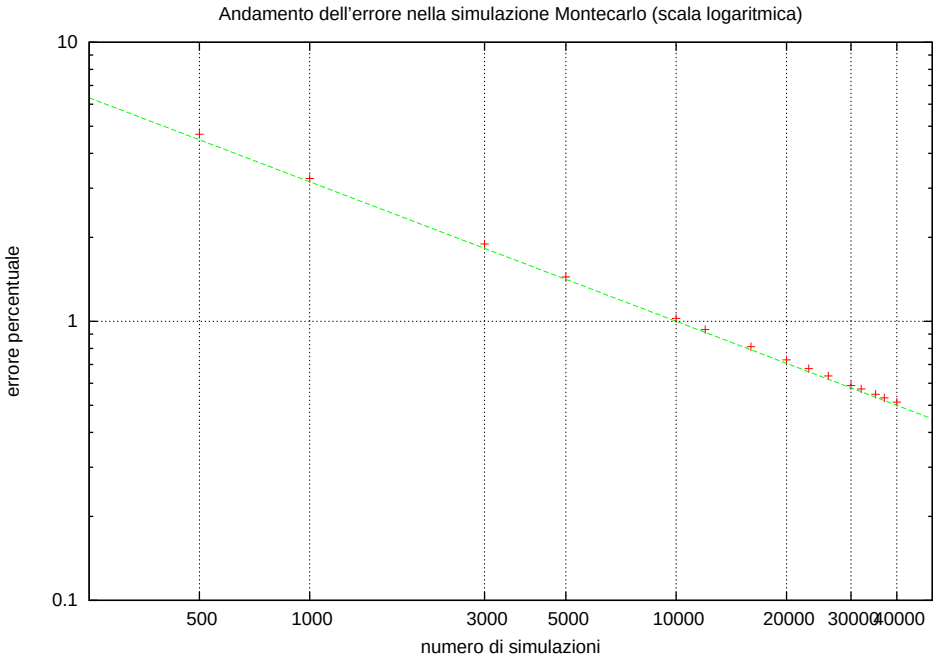

Figura 3.4: Andamento dell'errore della simulazione Montecarlo eseguita tramite la libreria che si attende asintotico a $\frac{1}{\sqrt{x}}$

Possiamo notare un perfetto fit dei dati ottenuti con la funzione $\frac{1}{\sqrt{N}}$. Nel grafico infatti (in scala logaritmica in entrambe le variabili) notiamo come i dati ottenuti si dispongano su una retta di pendenza -0.5.

N	err%
500	4.7
1000	3.3
3000	1.9
5000	1.4
10000	1.02
12000	0.93
16000	0.81
20000	0.73
23000	0.68
26000	0.64
30000	0.59
32000	0.57
35000	0.55
37000	0.53
40000	0.51

3.4.2 Variabile antitetica

In tutte le nostre simulazioni abbiamo utilizzato il metodo della variabile antitetica per la riduzione degli errori nel metodo Montecarlo.

La parte stocastica delle simulazioni consiste infatti nella creazione di un cammino che potrebbe ipoteticamente seguire l'azione sottostante all'opzione. Il metodo della variabile antitetica consiste nel considerare, per ogni cammino, anche il cammino opposto ovvero ottenuto con gli opposti della variabile random utilizzata per il cammino.

Riduzione degli errori

Grazie al metodo della variabile antitetica abbiamo a disposizione nelle nostre simulazioni cammini che sappiamo essere a due a due fortemente scorrelati fra loro. Nelle simulazioni stocastiche questo é molto importante, permette una riduzione degli errori statistici con cambiamenti a livello di implementazione della libreria praticamente nulli.

Il metodo della variabile antitetica non varia invece la dipendenza dell'errore da $\frac{1}{\sqrt{n}}$.

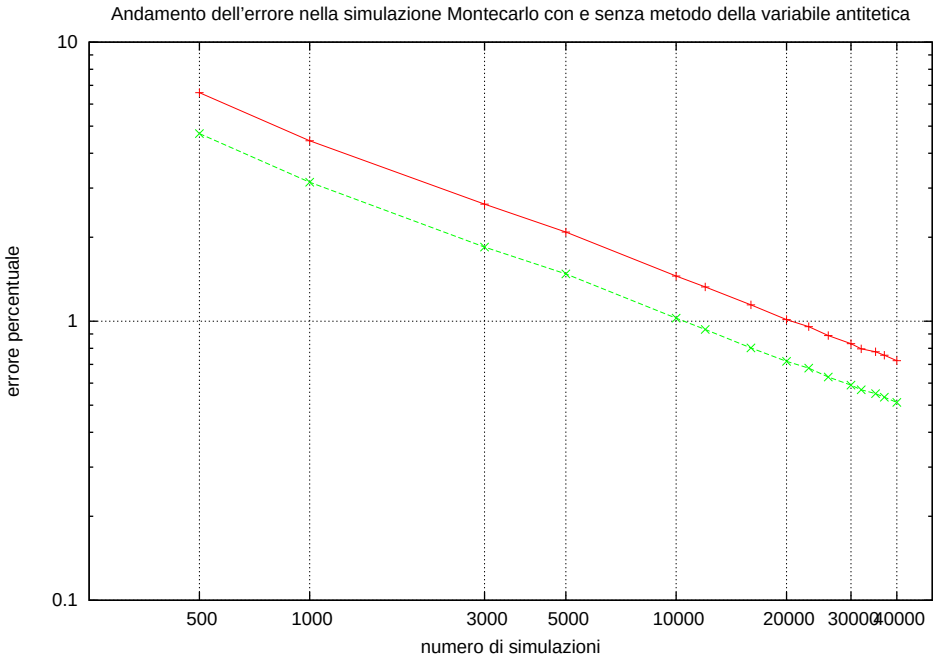

Figura 3.5: Confronto dell'andamento dell'errore nella simulazione Montecarlo con e senza il metodo della variabile antitetica. In rosso vediamo i punti di simulazione che rappresentano l'errore con l'utilizzo del metodo della variabile antitetica. In verde gli errori nella simulazione Montecarlo standard.

Vediamo in grafico come i dati delle simulazioni si dispongano su due distinte rette entrambe di coefficiente angolare -0.5. Il metodo della variabile antitetica permette infatti uno shift degli errori verso il basso, ma non ne varia la dipendenza da $\frac{1}{\sqrt{n}}$.

Capitolo 4

Studio analitico del contratto opzionale

Dopo aver controllato il corretto funzionamento dei metodi statistici della libreria, passiamo ad occuparci del particolare tipo di contratto opzionale in esame. Studiamo dunque le funzioni sulle quali il contratto é basato ottenendo delle previsioni qualitative sull'andamento del valore dell'opzione da confrontare poi con i risultati delle simulazioni.

4.1 Studio del corridoio

Studiamo ora la funzione di payoff della nostra opzione. La funzione definente i P_i:

$$P_i = Min[\ Max[\ \frac{S(t_{i+1})}{S(t_i)} - 1,\ l]\ u] \tag{4.1}$$

costruisce un corridoio limitato superiormente da u e inferiormente da l. Ribadiamo qui i concetti espressi nel secondo capitolo. Se la variazione dell'azione $\frac{S(t_{i+1})}{S(t_i)} - 1$ si mantiene all'interno del corridoio, questa da contributo alla sommatoria finale definente P. Qualora la variazione dell'azione portasse quel rapporto ad essere piú grande del valore della funzione u (o piú piccolo della funzione l), alla sommatoria finale verrá dato il contributo definito dal valore di u (o di l nel secondo caso).

Le dimensioni del corridoio sono determinate dai parametri T (maturity time),λ (parametro di k) e δ_t (intervallo temporale). In particolare, l'ultimo parametro é definito come

$$\delta_t = \frac{T}{m} \tag{4.2}$$

Cosí che risulta piú conveniente studiare le variazioni dell'intervallo in fun-

zione dei parametri T, λ e m.

$$l(T,\lambda,m) = \frac{e^{r\frac{T}{m}}}{1+\lambda\sqrt{\frac{T}{m}}} - 1 \tag{4.3}$$

$$u(T,\lambda,m) = (1+\lambda\sqrt{\frac{T}{m}})e^{r\frac{T}{m}} - 1 \tag{4.4}$$

4.1.1 Variazioni secondo λ e m

Vediamo ora come si modifica la larghezza del corridoio in funzione dei parametri λ e m. Mostriamo qui superfici in 3D che a livello qualitativo danno un'idea di ció che accade.

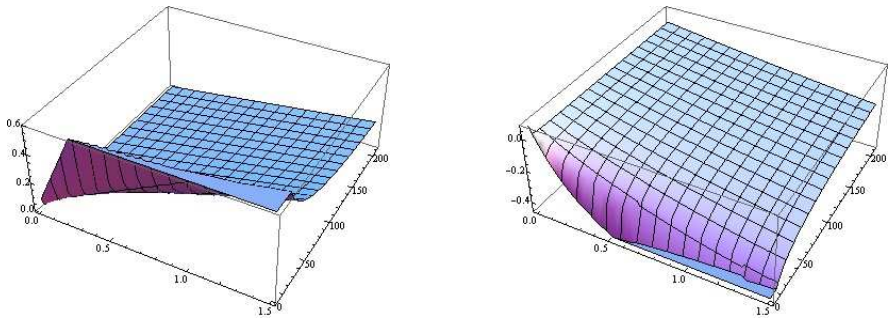

Figura 4.1: La funzione $u(\lambda, m)$ e la funzione $l(\lambda, m)$che rappresentano il limite superiore ed inferiore del corridoio

Notiamo come per bassi valori di m, il corridoio cresca con λ (che varia da 0 a 1.4), ma appena m (lasciato scorrere tra 0 e 200) supera un certo valore, il parametro λ diventi quasi ininfluente nella determinazione della larghezza del corridoio.

Inoltre si puó notare come a λ fissato uguale a zero, il corridoio si chiuda e u e l coincidano.

Vediamo in dettaglio le varie sezioni delle superfici ($l_m = l_m(\lambda)$ e $u_m = u_m(\lambda)$).

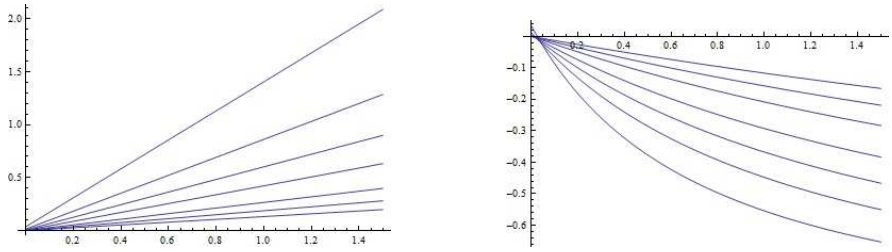

Figura 4.2: Le funzioni u e l che variano secondo il parametro λ. Nello stesso grafico vediamo le funzioni u e l a diversi fissati valori di m. Nel primo grafico, dall'alto verso il basso, vediamo rappresentate funzioni a valori di m crescenti. Viceversa per il secondo. m assume qui i valori $\{2,5,10,20,50,100,200\}$

4.1.2 Variazioni secondo T e m

Come sopra, diamo qui grafici che mostrano qualitativamente l'andamento del corridoio in superfici che rappresentano le funzioni $u(t,m)$ e $l(T,m)$.

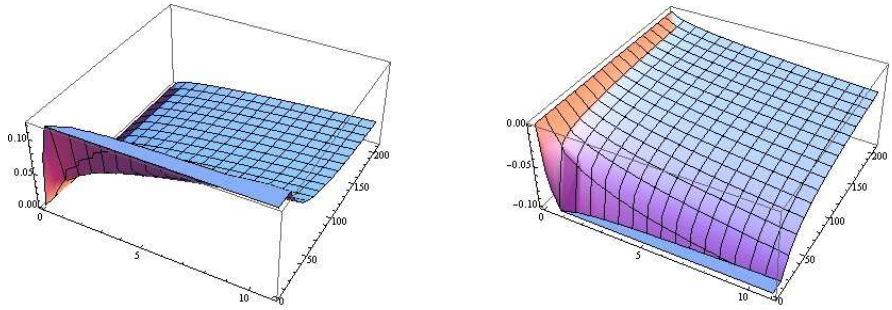

Figura 4.3: La funzione $u(T,m)$ e la funzione $l(T,m)$ che rappresentano il limite superiore ed inferiore del corridoio

Il corridoio si allarga al crescere di T e al diminuire di m, mentre tende a chiudersi per alti valori di m e bassi valori di T (fino a chiudersi del tutto per $T = 0$).

Vediamo in dettaglio le varie sezioni delle superfici ($u_m = u_m(T)$ e $l_m = l_m(T)$).

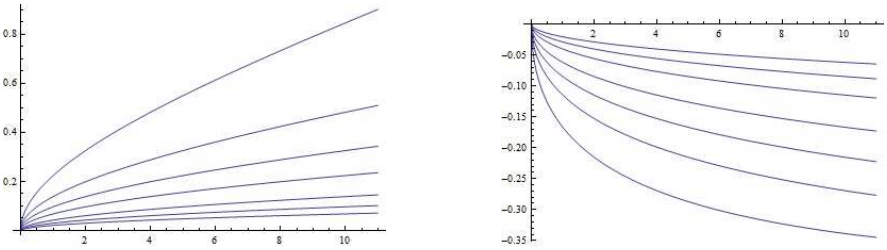

Figura 4.4: Le funzioni u e l che variano secondo la variabile T. Nello stesso grafico vediamo Le funzioni u e l a diversi fissati valori di m. Nel primo grafico, dall'alto verso il basso vediamo rappresentate funzioni a valori di m crescenti. Viceversa nel secondo. m assume qui i valori $\{2,5,10,20,50,100,200\}$

4.1.3 Variazioni secondo λ e T

Per completezza, mostriamo l'andamento del corridoio anche in funzione delle variabili λ e T, anche se gli studi successivi non verranno fatti per $u(T,\lambda)$ e $l(T,\lambda)$.

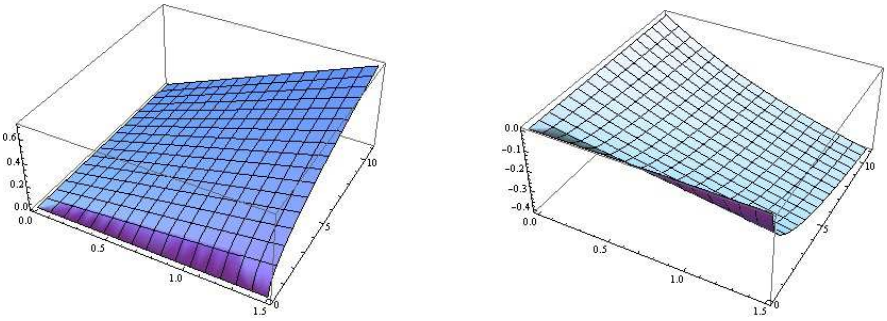

Figura 4.5: La funzione $u(T,\lambda)$, molto simile ad un piano inclinato, curvato solo ai limiti per $T \to 0$ e $\lambda \to 0$, e la funzione $l(T,m)$ che rappresenta il limite inferiore del corridoio

4.1.4 Considerazioni

Dallo studio delle funzioni $u(T,m,\lambda)$ e $l(T,m,\lambda)$ possiamo prevedere alcuni dei risultati che otterremo nella simulazione Montecarlo. Confrontando questi ultimi con i valori ottenuto dai grafici potremo testare ulteriormente la nostra libreria.

Incrementi

Dai grafici possiamo altresí dedurre come la differenza $u-l$ cresca con λ e T e decresca con m. Anche questo sará un possibile controllo nei risultati

(solo qualitativo).

Intersezioni

Come ultima considerazione, vogliamo qui far notare come siano diverse le superfici che delimitano i corridoi determinati dalla variazione dei parametri (λ, m) e dei parametri (T, m) e come le pendenze dei due singolarmente non siano lineari ai cambiamenti di una sola delle variabili (come risultava invece dalle variazioni della coppia di variabili (λ, T)).

Da questo fatto possiamo dedurre (o giustificare a posteriori) che i tassi di variazione dei price dell'opzione non saranno costanti, ma varieranno in maniera complessa. Nella pratica, le rette che collegheranno i vari punti simulati rappresentativi dei prezzi (in funzione di una delle variabili) avranno pendenze diverse da punto a punto. Avremo inoltre che i vari grafici, ognuno ad m fissato e rappresentativi di T o λ variabili, non saranno mere traslazioni l'uno del precedente, ma si potranno intersecare.

4.2 Formule chiuse

Possiamo ottenere (secondo alcuni valori dei parametri) delle formule di tipo deterministico con le quali prevedere l'andamento del price. Grazie a queste potremo altresí testare nuovamente la libreria in maniera mirata al problema in esame.

4.2.1 $\lambda = 0$

Risulta evidente dalle rappresentazioni del corridoio come ci siano alcuni valori dei parametri che azzerano la differenza delle due funzioni chiudendo il corridoio.

Prevediamo dunque che al limiti per $\lambda \to 0$ (e ovviamente $T \to 0$) il contibuto dei P_i sia costante o nullo e che l'errore in una simulazione con $\lambda = 0$ sia zero.

Al limite per $\lambda \to 0$, possiamo notare come il price sia descritto da una formula chiusa, determinata dalle condizioni iniziali del problema:

$$<\text{Price}>e^{rT} = \sum_{i=0}^{m-1} P_i = \sum_{i=0}^{m-1} l(\delta_t) = \sum_{i=0}^{m-1} (e^{(r\delta_t)} - 1) = m(e^{(r\frac{T}{m})} - 1) \quad (4.5)$$

Otteniamo quindi:

$$<\text{Price}> = m(e^{r\frac{T}{m}} - 1)e^{rT} \quad (4.6)$$

Provare la simulazione Montecarlo nel limite $\lambda = 0$ sará un ulteriore controllo sulla nostra libreria.

4.2.2 $\sigma = 0$

La volatilitá (σ) rappresenta la variazione secondo cui un'azione evolve. Vediamo cosa accade al limite per $\sigma \to 0$.

Dalla formula esatta otteniamo :

$$S_i = S_{i-1} e^{(r - \frac{\sigma^2}{2})\delta_t + \sigma\sqrt{\delta_t} w} = S_{i-1} e^{r - \frac{T}{m}} \tag{4.7}$$

Ricordando la formula per ottenere il price:

$$P = \sum_{i=0}^{m-1} P_i \tag{4.8}$$

Dove ognuno dei P_i, a volatilitá fissata a zero é ottenuto da:

$$P_i = Min[\ Max[\ e^{r - \frac{T}{m}} - 1,\ l]\ u] = \tag{4.9}$$

Le funzioni l e u diventano ininfluenti (il valore $e^{r - \frac{T}{m}} - 1$ valutato rimane sempre tra le due funzioni) ed il price (attualizzato) risulta

$$<\text{Price}> = m(e^{(r\frac{T}{m})} - 1)e^{-rT} \tag{4.10}$$

che notiamo essere esattamente la formula chiusa ricavata nella simulazione con $\lambda = 0$.

Simuleremo con il Montecarlo anche il caso $\sigma \to 0$ e confronteremo i risultati con quelli previsti e con quelli ottenuti per $\lambda \to 0$.

Capitolo 5

Simulazione Montecarlo

Veniamo ora alla simulazione Montecarlo e ai dati di pricing che abbiamo ottenuto.

Ogni curva é ottenuta con un numero N di simulazioni pari a 30000 (errore atteso $\sim 0.59\%$).

Lo strikeprice E é fissato a 0.02 (l'eventuale modifica di questo dato provacherebbe solamente una traslazione dei risultati).

5.1 Variabile λ

In questo paragrafo mostriamo i risultati ottenuti dalla simulazione a T fissato ($T = 0.5$) e λ variabile ($\lambda \epsilon [0 : 1.20]$).

Ogni grafico rappresenta le funzioni $\text{Price}_m(\lambda, T_0)$ con m che assume i valori $\{2, 5, 10, 20, 50, 150, 200\}$.

Abbiamo effettuato le simulazioni tramite la formula esatta e il metodo di Eulero.

5.1.1 Formula esatta

Per simulare l'andamento del sottostante abbiamo qui utilizzato la formula integrale esatta:

$$S_i = S_{i-1} e^{(r - \frac{\sigma^2}{2})\delta_t + \sigma\sqrt{\delta_t} w} \tag{5.1}$$

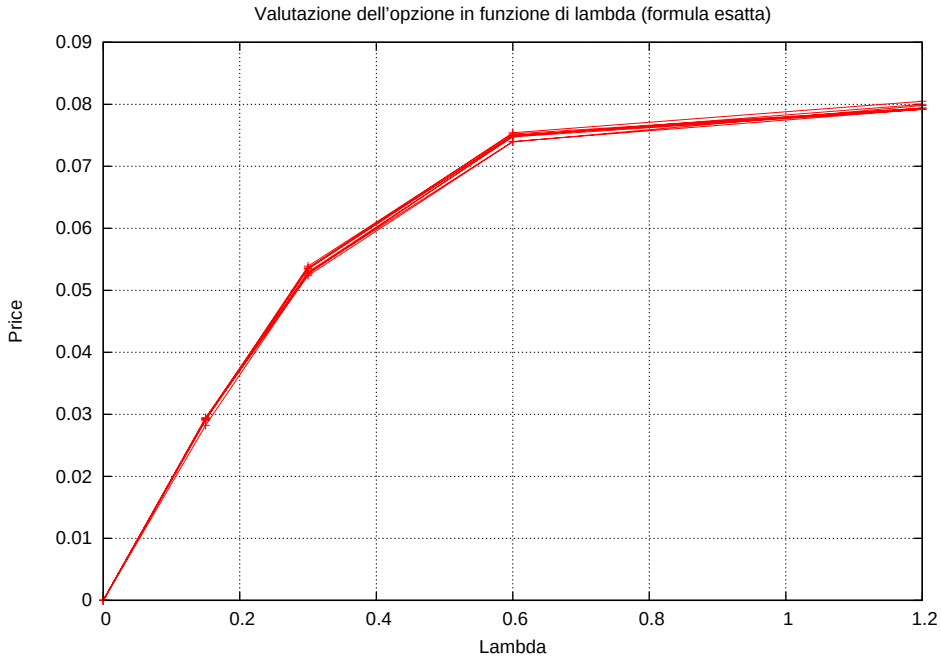

Figura 5.1: Valutazione del price dell'opzione in funzioni di λ tramite la formula esatta. Ogni curva rappresenta un m fissato.

Dal grafico si evince come il valore dell'opzione cresca al crescere di λ, partendo da un unico punto iniziale per $\lambda = 0$. Studieremo poi meglio il caso $\lambda = 0$, ad ogni modo possiamo qui anticipare che il valore dell'opzione in quel punto é determinato dallo strikeprice e sostanzialmente indipendente da m per bassi valori di maturity time.

Notiamo inoltre che, come previsto, la crescita del price non é lineare in λ, i grafici si intersecano in piú punti e il coefficiente angolare delle rette congiungenti i punti simulati diminuisce all'aumentare di λ.

I dati utilizzati in questa simulazione si trovano in appendice, Tabella A.

5.1.2 Metodo di Eulero

Per simulare l'andamento del sottostante utilizziamo ora il metodo di Eulero:

$$S_i = S_{i-1} + S_{i-1}(r\Delta t + \sigma\sqrt{\Delta t}w) \tag{5.2}$$

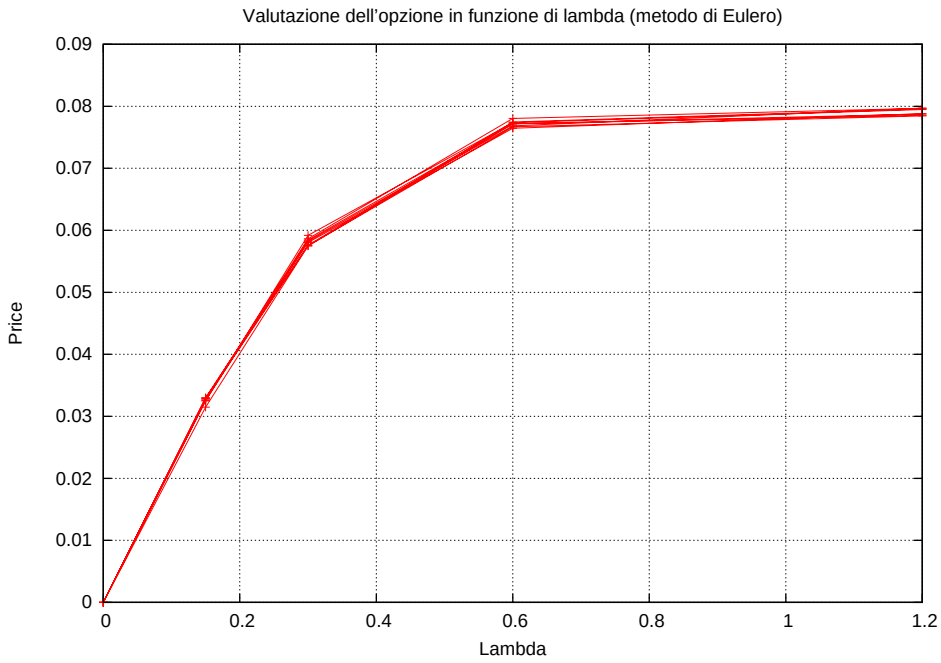

Figura 5.2: Valutazione del price dell'opzione in funzioni di λ tramite il metodo di Eulero. Ogni curva rappresenta un m fissato.

Il grafico é molto simile al grafico realizzato utilizzando la formula esatta, come atteso.

Il crescere di λ causa una distensione delle rette delimitanti il corridoio, come abbiamo giá verificato. La dipendenza del corridoio da λ giustifica la distensione delle varie curve, che si verifica per alti valori di λ.

I dati utilizzati in questa simulazione si trovano in appendice, Tabella B.

5.1.3 Confronto

Il metodo di Eulero é un'approssimazione della formula integrale esatta, come abbiamo specificato nella sezione (3.2.2). Confrontiamo in questo paragrafo i risultati ottenuti.

Differenze

Valutiamo la differenza dei valori ottenuti tramite Pricing con i due metodi in funzione di lambda.

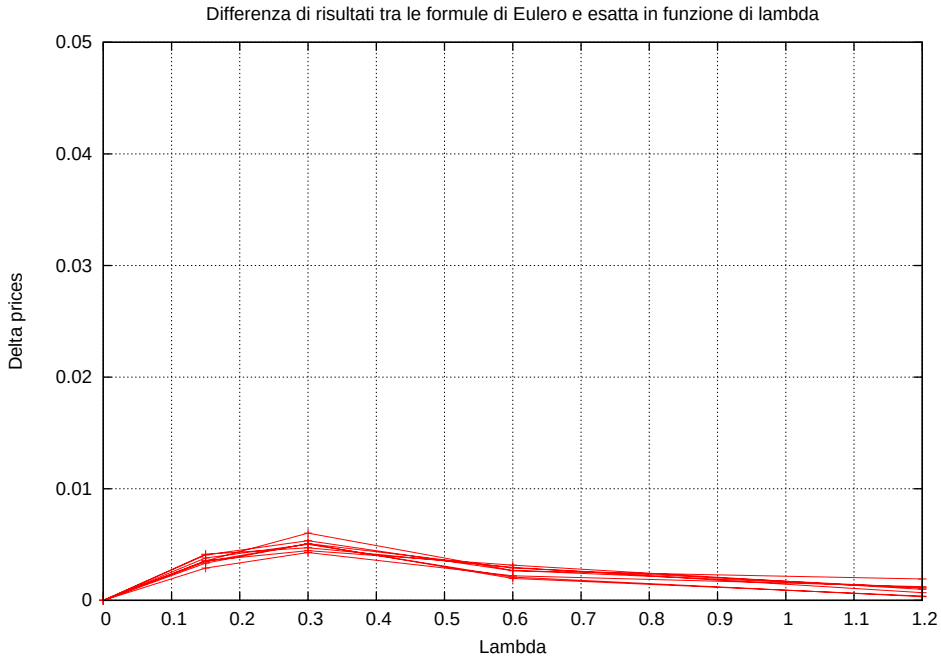

Figura 5.3: Differenza di risultati tra i metodi di Eulero e esatto esposta in funzione di λ. Ogni curva rappresenta un m fissato.

Il parametro λ non é causa di un maggior errore nell'approssimazione della formula di Eulero. Nella formula infatti, l'unico parametro che si richiede essere piccolo, é il tempo (la variabile secondo cui evolve il sottostante). Vediamo anzi che al crescere di λ, l'errore percentule diminuisce (da $\sim 10\%$ a $\sim 1\%$) fino a rendere le formule egualmente valide.

Il valore d'errore piú alto ($\sim 10\%$) é sostanzialmente determinato dal maturity time scelto per la simulazione ($T = 0.5$), come mostreremo meglio in seguito.

Errori

A seguito i grafici che rappresentano gli errori nella valutazione del price dell'opzione in funzione di λ.

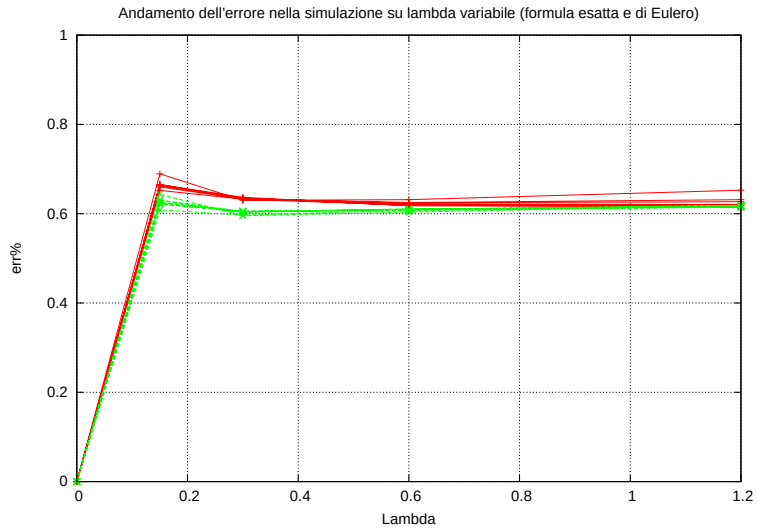

Figura 5.4: Andamento dell'errore nella valutazione dell'opzione in funzione di λ secondo la formula esatta. Ogni curva rappresenta un m fissato. In rosso vediamo le curve che rappresentano l'errore dovuto all'utilizzo della formula integrale esatta. In verde le curve in riferimento al metodo di Eulero.

I dati utilizzati in questa simulazione si trovano in appendice, Tabella A e B.

Entrambi i grafici si mantengono sotto il valore 0.7 (errore %), valore decisamente ottimo per un pricing.

Per $\lambda = 0$ l'errore si annulla, il corridoio si stringe a due rette coincidenti, come mostrato nel capitolo precedente.

5.2 Variabile T

In questo paragrafo mostriamo i risultati ottenuti dalla simulazione a λ fissato ($\lambda = 0.15$) e T variabile ($T\epsilon[0.5 : 10.5]$).

Ogni grafico rappresenta le funzioni $\text{Price}_m(\lambda_0, T)$ con m che assume i valori $\{2, 5, 10, 20, 50, 150, 200\}$.

Abbiamo effettuato le simulazioni tramite la formula esatta e il metodo di Eulero.

5.2.1 Formula esatta

Per simulare l'andamento del sottostante abbiamo qui utilizzato la formula integrale esatta:

$$S_i = S_{i-1} e^{(r - \frac{\sigma^2}{2})\delta_t + \sigma\sqrt{\delta_t} w} \tag{5.3}$$

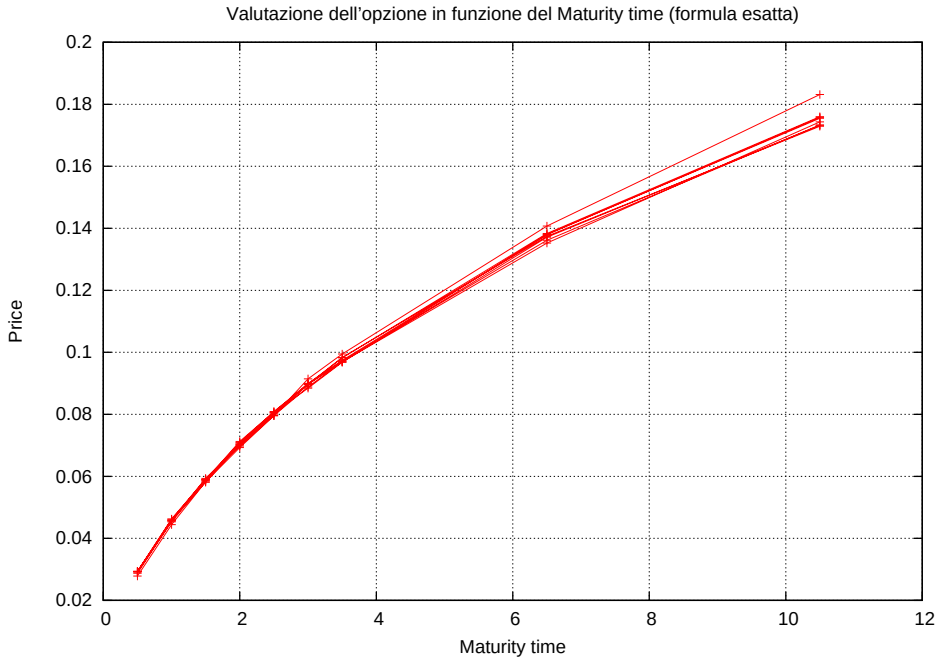

Figura 5.5: Valutazione del price dell'opzione in funzioni di T tramite la formula esatta. Ogni curva rappresenta un m fissato.

Il nostro grafico ci mostra, come atteso, un price crescente come funzione del maturity time. Come previsto, le varie curve rappresentative di m costanti hanno tassi di crescita variabili e s'intersecano in vari punti.

I dati utilizzati in questa simulazione si trovano in appendice, Tabella C.

5.2.2 Metodo di Eulero

Per simulare l'andamento del sottostante utilizziamo ora il metodo di Eulero:

$$S_i = S_{i-1} + S_{i-1}(r\Delta t + \sigma\sqrt{\Delta t}w) \tag{5.4}$$

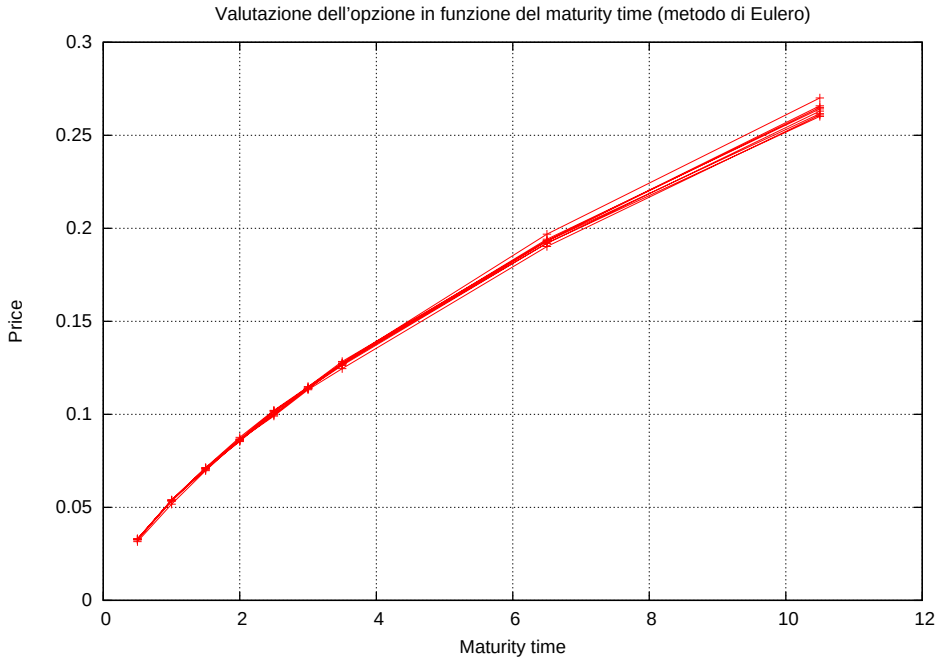

Figura 5.6: Valutazione del price dell'opzione in funzioni di T tramite il metodo di Eulero. Ogni curva rappresenta un m fissato.

I dati utilizzati in questa simulazione si trovano in appendice, Tabella D.

L'andamento del grafico si presenta molto simile al grafico ottenuto con il metodo precente,ma i valori del price si discostano notevolmente dai valori ottenuti con la formula esatta. Vediamo in dettaglio nel prossimo paragrafo.

5.2.3 Confronto

Vediamo a confronto il metodo di Eulero e la formula esatta. Ricordiamo che la formula di Eulero é un'approssimazione tanto piú precisa quanto piú $T \to 0$.

Differenze

Vediamo di quanto i risultati ottenuti tramite il metodo di Eulero si discostano dall'effettivo valore di pricing ottenuto tramite la formula esatta.

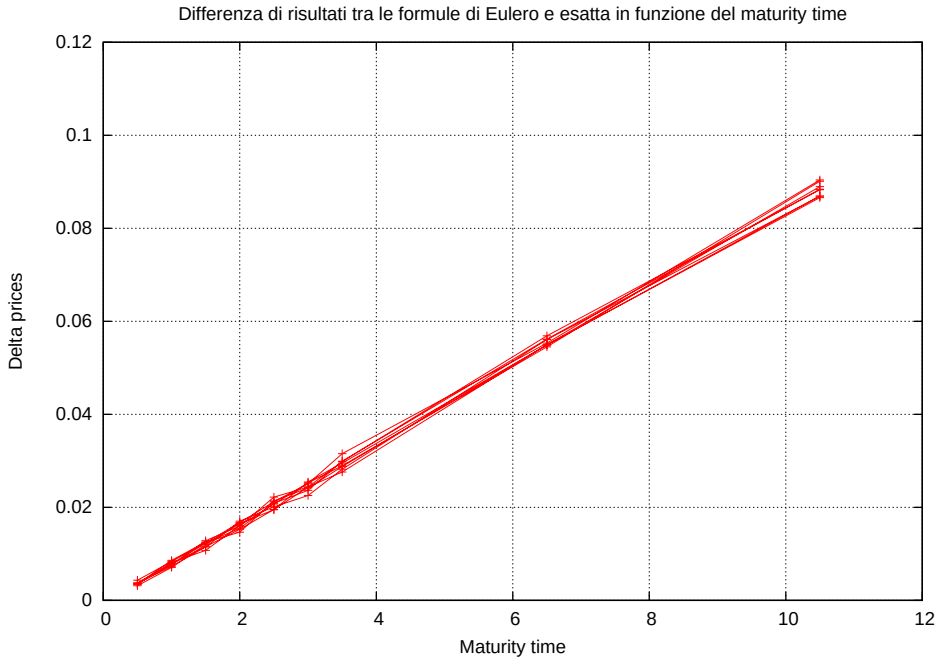

Figura 5.7: Differenza di risultati tra i metodi di Eulero e esatto esposta in funzione del maturity time. Ogni curva rappresenta un m fissato.

L'errore dei risultati ottenuti con il metodo di Eulero aumenta praticamente linearmente con il maturity time. É evidente che per T grande tale formula risulti un metodo scorretto di approssimazione.

Completando peró la formula del metodo di Eulero con lo sviluppo dell'esponenziale al primo ordine, ovvero:

$$S_i = S_{i-1} + S_{i-1}((r - \frac{\sigma^2}{2})\Delta t + \sigma\sqrt{\Delta t}w) \tag{5.5}$$

otteniamo dei risultati decisamente migliori, fino a raggiungere un massimo di errore pari al 5%. Notiamo infatti in grafico un'ottima sovrapposizione dei grafici.

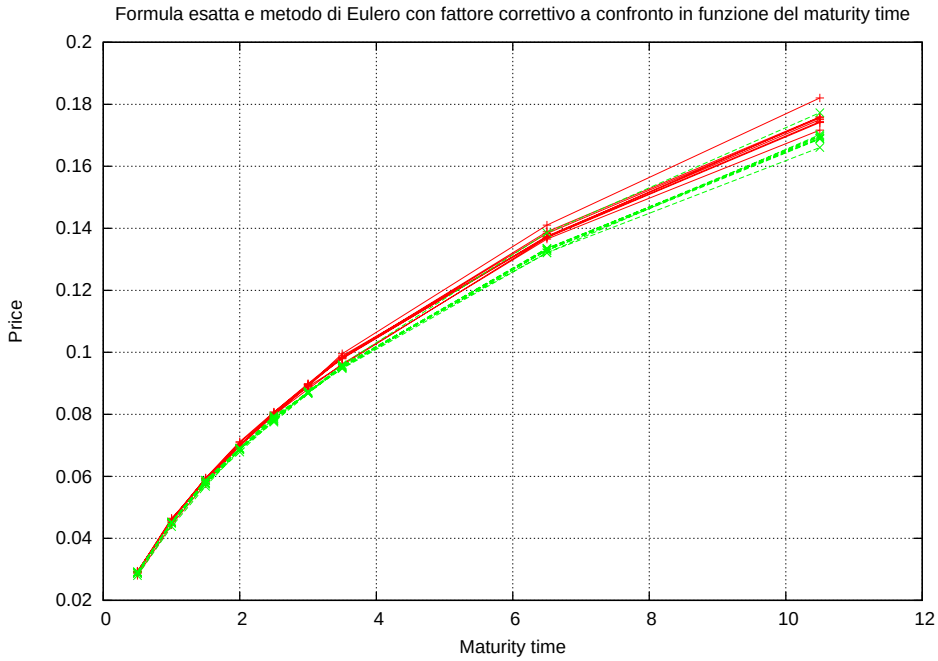

Figura 5.8: Sovrapposizione dei risultati di pricing ottenuti con la formula esatta e il metodo di Eulero completato al prim'ordine di sviluppo dell'esponenziale in funzione del maturity time. In rosso i risulati ottenuti con la formula esatta, in verde il metodo di Eulero. Ogni curva rappresenta un m fissato.

Errori

A seguito i grafici che rappresentano gli errori nella valutazione del price dell'opzione in funzione di T (maturity time).

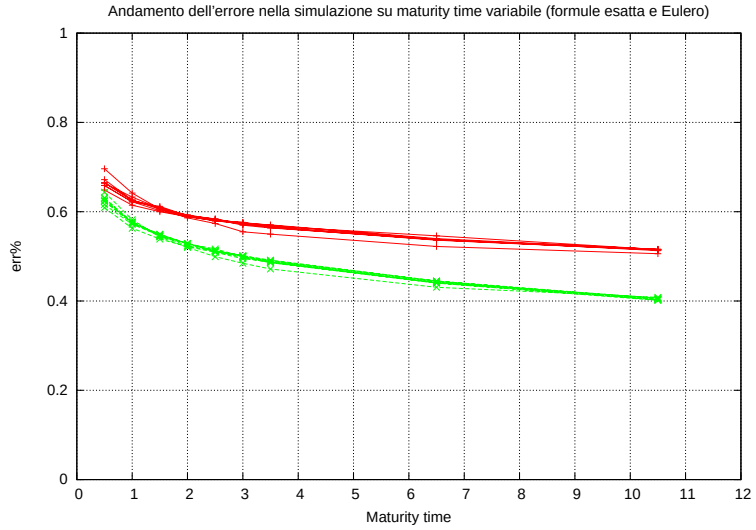

Figura 5.9: Andamento dell'errore nella valutazione dell'opzione in funzione di T secondo la formula esatta. Ogni curva rappresenta un m fissato. In rosso vediamo le curve che rappresentano l'errore dovuto all'utilizzo della formula integrale esatta. In verde le curve in riferimento al metodo di Eulero.

I dati utilizzati in questa simulazione si trovano in appendice, Tabella C e D.

Le fluttuazioni statistiche diminuiscono all'aumentare del maturity time. Da notare come gli errori nel metodo di Eulero parrebbero piú bassi degli errori dei risultati ottenuti col metodo esatto. Questo non significa che il metodo di Eulero sia piú preciso, ma semplicemente che i valori convergono piú rapidamente a un risultato che sappiamo essere solo approssimato.

5.3 Valutazioni deterministiche

Mostriamo in questa sezione alcuni risultati prevedibili in maniera deterministica che abbiamo mostrato nel capitolo precedente.

5.3.1 $\lambda = 0$

Fissato $\lambda = 0$, la simulazione Montecarlo diventa superflua. Infatti il problema diventa deterministico, il corridoio si chiude e il price dell'opzione é perfettamente determinato dalle condizioni iniziali secondo la formula:

$$<\text{Price}> = m(e^{(r\frac{T}{m})} - 1)e^{-rT} \tag{5.6}$$

In grafico i risultati attesi dalla simulazione (a meno dello strikeprice, da considerarsi nel grafico pari a zero).

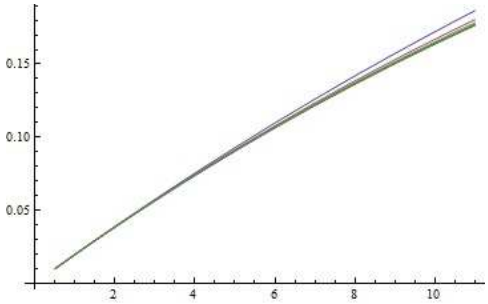

Figura 5.10: Valori del price attesi per la simuzione su $\lambda = 0$.

In particolare, lo strikeprice dell'opzione (in questo caso fissato a 0.02) porta il grafico a coincidere con la retta costante $y = 0$ sostanzialmente per i primi due punti di simulazione. All'aumentare dello strikeprice i punti di simulazione coincidenti con tale retta aumentano. Infatti, finché P (price) é minore di E (strikeprice) il payoff dell'opzione resta nullo. P aumenta fino a superare lo strikeprice e alzando la funzione sopra lo zero.

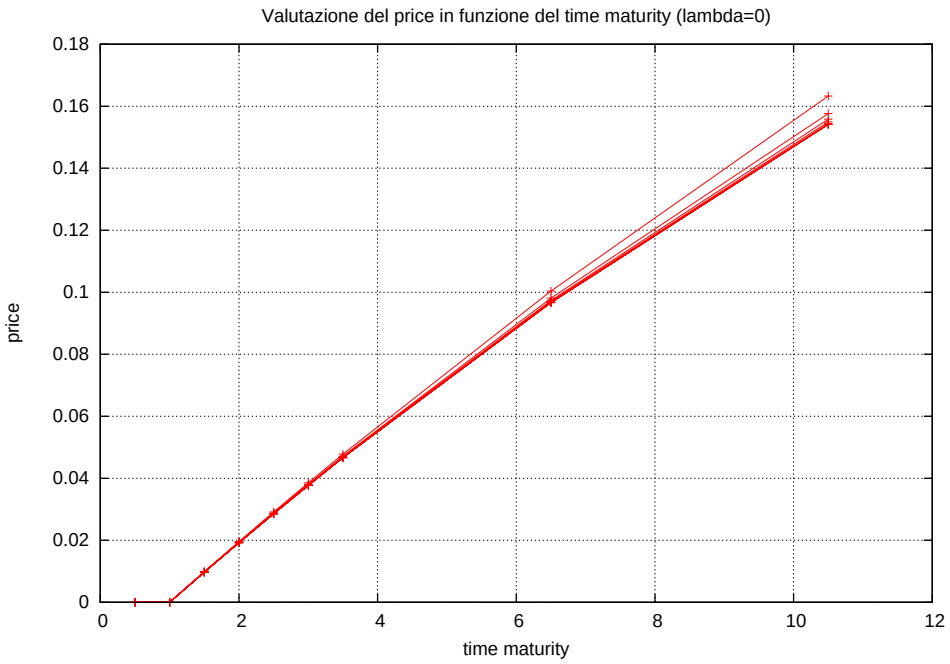

Figura 5.11: Valutazione del price dell'opzione in funzione del maturity time a λ fissato $\lambda = 0$ secondo la formula integrale esatta. Ogni curva rappresenta un m fissato. Data la netta sovrapposizione non si é fatta distinzione tra i vari m.

La nostra previsione sulla simulazione risulta molto precisa. I grafici si

comportano esattamente secondo la formula chiusa (5.6).

Essendo un problema deterministico, gli errori devono essere nulli.

In tabella a seguire i dati nella variabile $m = 2$.

n	T	price	err%
1	0.5	0	0
2	1	0.0001	0
3	1.5	0.01	0
4	2	0.02	0
5	2.5	0.03	0
6	3	0.04	0
7	3.5	0.05	0
8	6.5	0.10	0
9	10.5	0.16	0

5.3.2 Volatilitá nulla

Consideriamo ora la stessa opzione con volatilitá fissata a zero: il problema diventa nuovamente deterministico, come esposto nel capitolo precedente. La formula secondo cui evolve il price é:

$$<\text{Price}> = m(e^{(r\frac{T}{m})} - 1)e^{-rT} \tag{5.7}$$

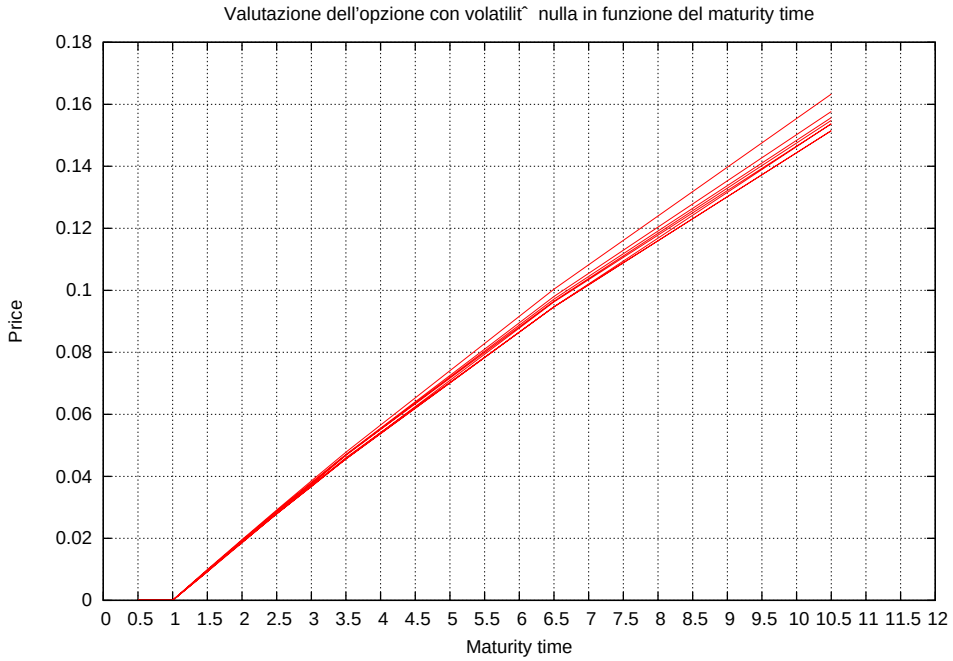

Figura 5.12: Valutazione del price dell'opzione in funzione del maturity time a σ fissata $\sigma = 0$ secondo la formula integrale esatta. Ogni curva rappresenta un m fissato.

Come previsto, a paritá di dati iniziali, il price dell'opzione si comporta esattamente come nella simulazione su $\lambda = 0$.

Ricordiamo, essendo una simulazione deterministica, gli errori devono essere pari a zero.

In tabella a seguire i dati nella variabile $m = 2$.

n	T	price	err%
1	0.5	0	0
2	1	0.0001	0
3	1.5	0.01	0
4	2	0.02	0
5	2.5	0.03	0
6	3	0.04	0
7	3.5	0.05	0
8	6.5	0.10	0
9	10.5	0.16	0

5.4 Simulazioni binarie

Concludiamo il nostro progetto con un esempio di estensione della libreria tramite il polimorfismo. Come specificato nella sezione 3.2, l'evoluzione del sottostante é stata simulata tramite le due diverse formule (Eulero e esatta) che richiedevano estrazioni di random variabili da una distribuzione gaussiana di media nulla e varianza unitaria (processo lognormale standard).

Consideriamo ora un processo alternativo ottenuto ancora con la formula integrale esatta, dove peró sostuiamo alla random variabile gaussiana w, un nuovo tipo di variabile stocastica z definita come:

$$z = \begin{cases} +1 & \text{con probabilitá } 0.5 \\ -1 & \text{con probabilitá } 0.5 \end{cases}$$

Consideriamo dunque un processo binario e valutiamo se la nostra libreria é facilmente estensibile ad una variazione di questo tipo.

Abbiamo dunque bisogno di una funzione che ci restituisca un appropriato numero random z. Per far ció é stato sufficiente aggiungere una funzione alle classi dei generatori di numeri casuali e nessun'altra modifica é stata apportata alla libreria.

Il secondo punto fondamentale consiste nell'utilizzare la stessa formula giá implementata nella classe Eq_process_lognormal_esatto con la sola variazione della variabile random.

La struttura gerarchica delle classi per la gestione dei processi di evoluzione del sottostante di tipo lognormal ci ha permesso tramite derivazione della classe Eq_process_lognormal_binary (figlia della classe madre Eq_process_lognormal_esatto) di risolvere il problema con la sola reimplementazione della funzione Get_w(), ovvero la funzione che restituisce la variabile stocastica utilizzata nella formula esatta.

La nostra libreria si é dimostrata dunque elastica e facilmente estensibile. Queste caratteristiche sono ottenibili nell'ambito della programmazione per classi, con ampio uso di polimorfismo. Nella programmazione procedurale invece una tale estensione si sarebbe rivelata molto difficoltosa e facilmente soggetta ad errori.

5.4.1 Analisi della formula

Cerchiamo di prevedere i risultati della simulazione. Dimentichiamo per un attimo il corridoio delimitante le variazioni del sottostante e valutiamo il price tramite la formula integrale esatta:

$$<\text{Price}> = \sum_{i=0}^{m-1} P_i = \sum_{i=0}^{m-1} (\frac{S_{i+1}}{S_i} - 1) = \sum_{i=1}^{m} (e^{(r-\frac{\sigma^2}{2})\delta_t + \sigma\sqrt{\delta_t} z_i} - 1) \qquad (5.8)$$

Fissati i dati iniziali, possiamo manipolare algebricamente la formula, sfruttando la distribuzione discreta di probabilitá, e ottenere:

$$<\text{Price}> = e^{(r-\frac{\sigma^2}{2})\frac{T}{m}} \sum_{i=1}^{m} (e^{\sigma\sqrt{\frac{T}{m}}z_i}) - m = e^{(r-\frac{\sigma^2}{2})\frac{T}{m}} (\frac{m}{2} e^{\sigma\sqrt{\frac{T}{m}}} + \frac{m}{2} e^{-\sigma\sqrt{\frac{T}{m}}}) - m \tag{5.9}$$

Possiamo infine ricondurci alla formula:

$$<\text{Price}> = m(e^{(r-\frac{\sigma^2}{2})\frac{T}{m}} Cosh(\sigma\sqrt{\frac{T}{m}}) - 1) \tag{5.10}$$

In grafico i valori della funzione a diversi fissati valori di m con variabile T.

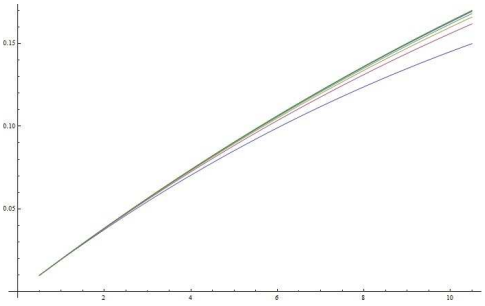

Figura 5.13: In grafico la funzione (5.9) con variabile T che rappresenta una previsione del pricing per $\lambda \to \infty$. Ogni curva rappresenta un m fissato.

Ricordiamo che tutte le considerazioni sono fatte al limite per $\lambda \to \infty$, ovvero senza considerare il corridoio. Nelle simulazioni del prossimo paragrafo, λ avrá un valore finito (fissato per ogni combinazione di T e m) che quindi abbasserá i valori dei grafici.

5.4.2 Simulazione

Testiamo la nuova classe derivata. Otteniamo con la nuova formula le medesime simulazioni di pricing dell'opzione che abbiamo eseguito con il metodo di Eulero e la formula esatta con random variabile gaussiana.

Nel complesso possiamo attenderci un comportamento del price dell'opzione molto simile a quello ottenuto nelle simulazioni eseguite con l'altro metodo. La variabile stocastica z utilizzata ha infatti media nulla e varianza unitaria, esattamente come la variabile w che compariva nelle precedenti formule (gaussiana di media nulla e varianza unitaria).

Variabile T

A seguito il grafico ottenuto in funzione del maturity time. Dal grafico notiamo che le simulazioni presentano l'andamento aspettato.

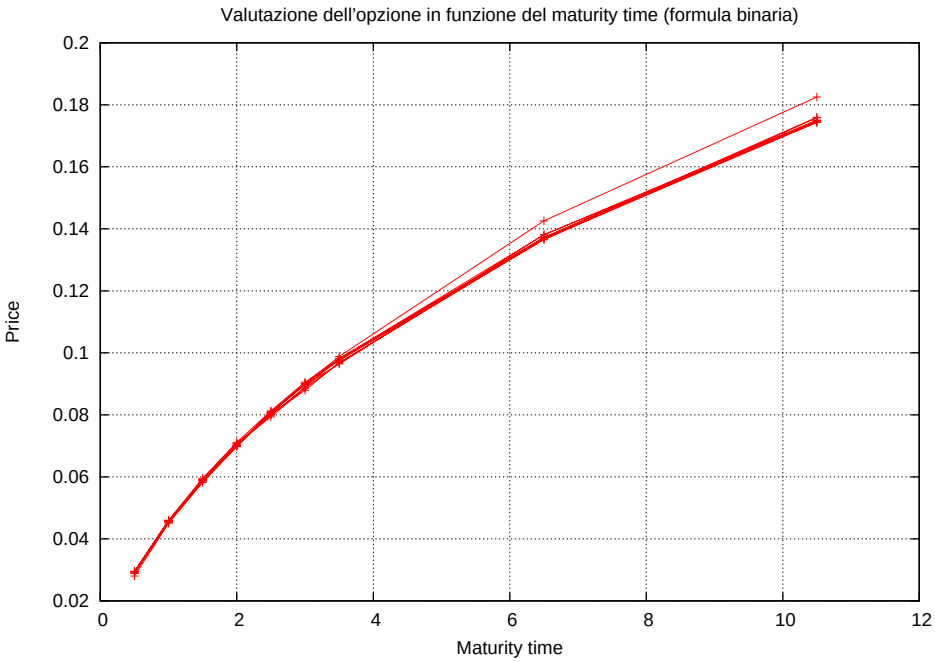

Figura 5.14: Valutazione del price dell'opzione in funzioni di T tramite la formula esatta con estrazione binaria della random variabile. Ogni curva rappresenta un m fissato.

Abbiamo previsto in maniera corretta l'andamento delle simulazioni. Consideriamo che nella simulazione λ ha un valore finito quindi il corridoio modifica in parte l'inclinazione delle curve ed i loro valori. Inoltre la previsione analitica é fatta al limite per il numero di simulazioni $N \to \infty$, questo comporta che nella simulazione la media delle variabili stocastiche non sia esattamente nulla; poiché la funzione che consideriamo é esponenziale lo scostamento dalla media implica un diverso comportamento a seconda che sia positivo o negativo.

Il grafico rappresentativo dell'andamento dell'errore nella simulazione.

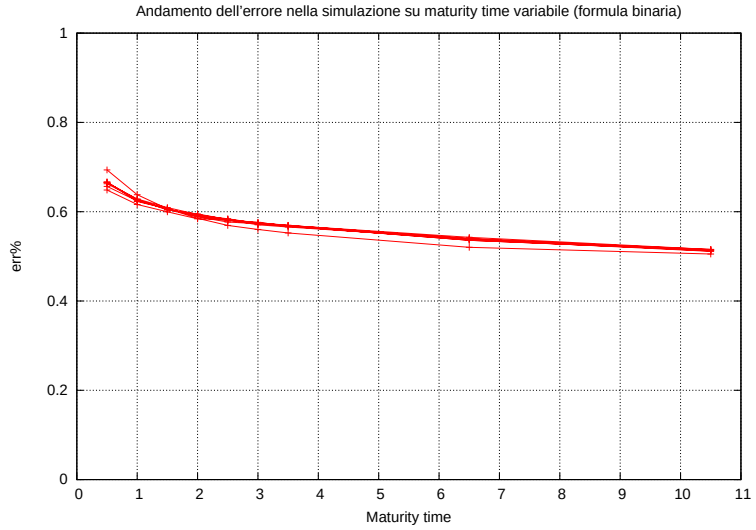

Figura 5.15: Andamento dell'errore nella valutazione dell'opzione in funzione di λ secondo la formula esatta con estrazione stocastica binaria. Ogni curva rappresenta un m fissato.

I dati utilizzati in questa simulazione si trovano in appendice, Tabella F.

Notiamo che l'andamento degli errori é molto simile a quello ottenuto con gli altri processi. Vi é infatti una diminuzione dell'errore al crescere di T. D'altra parte questo é prevedibile in quanto le distribuzioni da cui sono estratte le variabili stocastiche presentano entrambe varianza unitaria.

La variazione del price si dimostra ancora sostanzialmente la medesima del grafico ottenuto con processo esatto ad estrazione gaussiana.

Variabile λ

Al variare di λ riscontriamo il medesimo andamento del pricing per gli altri tipi di processo.

A seguito il grafico ottenuto dalla simulazione su variabile λ con curve rappresentative di m fissati.

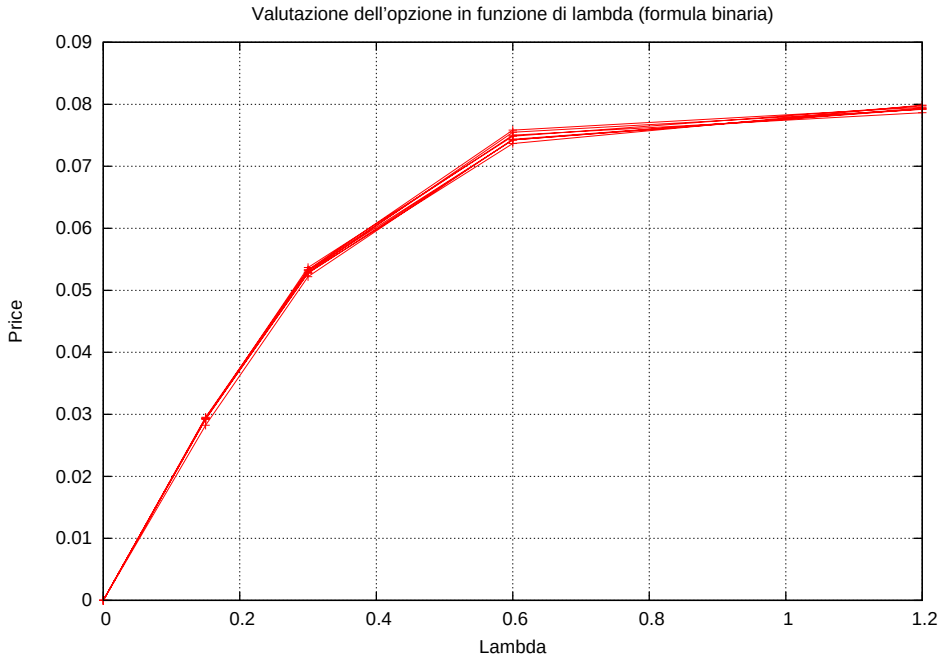

Figura 5.16: Valutazione del price dell'opzione in funzioni di λ tramite la formula esatta con estrazione binaria della random variabile. Ogni curva rappresenta un m fissato.

Per $\lambda = 0$ ritroviamo ancora price nullo a causa della chiusura del corridoio. Per λ piccolo il price sale piuttosto velocemente e presenta una maggiore dipendenza da m che si nota dallo scostamento delle curve sul grafico. Al crescere di λ la pendenza diminuisce nettamente.

Il grafico rappresentativo dell'andamento dell'errore nella simulazione.

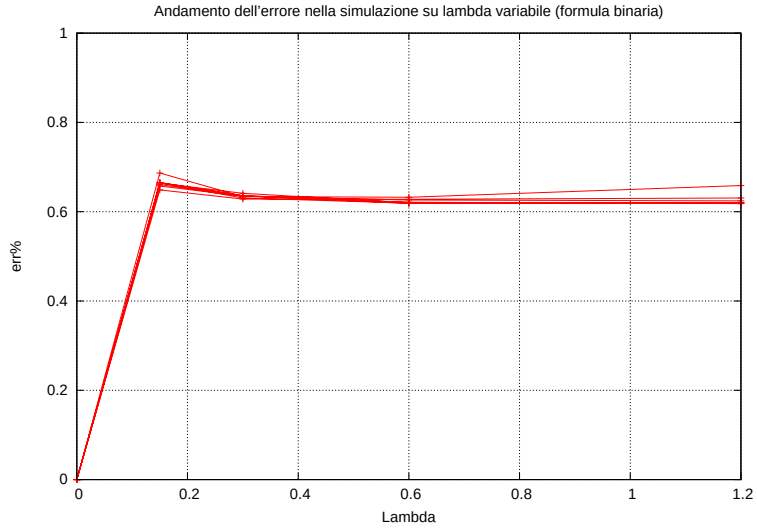

Figura 5.17: Andamento dell'errore nella valutazione dell'opzione in funzione di T secondo la formula esatta con estrazione stocastica binaria. Ogni curva rappresenta un m fissato.

Anche al variare di λ notiamo che l'andamento dell'errore é del tutto simile a quello ottenuto per gli altri processi, proprio come ci saremmo aspettati. I dati utilizzati in questa simulazione si trovano in appendice, Tabella E.

Capitolo 6

Appendice

6.1 Tabella A

Dati ottenuti dalla simulazione di pricing per λ e m variabili con formula esatta.

#N simulazioni 30000
#$S_0 = 100$
#strikeprice = 0.02
#rate = 0.02
#volatility = 0.3
#Maturity Time 0 years 6 months 0 days 0 h 0 min 0 sec
#$\lambda = \{0, 0.15, 0.3, 0.6, 1.2\}$
#$m = \{2, 5, 10, 20, 50, 100, 150, 200\}$

lambda	price	err%	err% std
$m = 2$			
0	0	0	0
0.15	0.0283	0.68	0.97
0.3	0.0530	0.62	0.88
0.6	0.0747	0.63	0.89
1.2	0.0801	0.65	0.91
lambda	price	err%	err% std
$m = 5$			
0	0	0	0
0.15	0.0293	0.64	0.92
0.3	0.0529	0.63	0.89
0.6	0.0745	0.62	0.88
1.2	0.0794	0.63	0.90

lambda	price	err%	err% std
$m = 10$			
0	0	0	0
0.15	0.0290	0.66	0.92
0.3	0.0527	0.63	0.90
0.6	0.0740	0.62	0.88
1.2	0.0791	0.62	0.88
lambda	price	err%	err% std
$m = 20$			
0	0	0	0
0.15	0.0290	0.66	0.94
0.3	0.0531	0.63	0.90
0.6	0.0744	0.62	0.88
1.2	0.0783	0.62	0.87
lambda	price	err%	err% std
$m = 50$			
0	0	0	0
0.15	0.0291	0.66	0.94
0.3	0.0520	0.64	0.91
0.6	0.0745	0.61	0.86
1.2	0.0800	0.61	0.86
lambda	price	err%	err% std
$m = 100$			
0	0	0	0
0.15	0.0291	0.66	0.93
0.3	0.0537	0.63	0.89
0.6	0.0747	0.62	0.88
1.2	0.0807	0.61	0.86
lambda	price	err%	err% std
$m = 150$			
0	0	0	0
0.15	0.0296	0.66	0.93
0.3	0.0538	0.63	0.89
0.6	0.0754	0.61	0.87
1.2	0.0804	0.61	0.87
lambda	price	err%	err% std
$m = 200$			
0	0	0	0
0.15	0.0288	0.66	0.94
0.3	0.0538	0.63	0.89
0.6	0.0755	0.61	0.87
1.2	0.0791	0.62	0.87

6.2 Tabella B

Dati ottenuti dalla simulazione di pricing per λ e m variabili con formula di Eulero.

#N simulazioni 30000
#$S_0 = 100$
#strikeprice = 0.02
#rate = 0.02
#volatility = 0.3
#Maturity Time 0 years 6 months 0 days 0 h 0 min 0 sec
#$\lambda = \{0, 0.15, 0.3, 0.6, 1.2\}$
#$m = \{2, 5, 10, 20, 50, 100, 150, 200\}$

lambda	price	err%	err% std
$m = 2$			
0	0	0	0
0.15	0.0312	0.64	0.90
0.3	0.0573	0.59	0.84
0.6	0.0769	0.60	0.85
1.2	0.0789	0.61	0.86
lambda	price	err%	err% std
$m = 5$			
0	0	0	0
0.15	0.0331	0.60	0.85
0.3	0.0580	0.59	0.84
0.6	0.0766	0.60	0.85
1.2	0.0791	0.61	0.87
lambda	price	err%	err% std
$m = 10$			
0	0	0	0
0.15	0.0324	0.62	0.87
0.3	0.0578	0.60	0.84
0.6	0.0760	0.61	0.86
1.2	0.0788	0.61	0.86
lambda	price	err%	err% std
$m = 20$			
0	0	0	0
0.15	0.0332	0.61	0.87
0.3	0.0578	0.60	0.85
0.6	0.0773	0.60	0.85
1.2	0.0790	0.61	0.87

lambda	price	err%	err% std
$m = 50$			
0	0	0	0
0.15	0.0326	0.63	0.89
0.3	0.0581	0.60	0.85
0.6	0.0772	0.60	0.86
1.2	0.0789	0.61	0.87

lambda	price	err%	err% std
$m = 100$			
0	0	0	0
0.15	0.0327	0.63	0.89
0.3	0.0582	0.60	0.86
0.6	0.0779	0.61	0.86
1.2	0.0797	0.62	0.87

lambda	price	err%	err% std
$m = 150$			
0	0	0	0
0.15	0.0330	0.63	0.88
0.3	0.0589	0.60	0.86
0.6	0.0784	0.61	0.86
1.2	0.0793	0.62	0.87

lambda	price	err%	err% std
$m = 200$			
0	0	0	0
0.15	0.0329	0.63	0.88
0.3	0.0592	0.60	0.85
0.6	0.0782	0.61	0.86
1.2	0.0810	0.61	0.87

6.3 Tabella C

Dati ottenuti dalla simulazione di pricing per T e m variabili con formula esatta.

#N simulazioni 30000
#$S_0 = 100$
#strikeprice $= 0.02$
#rate $= 0.02$
#volatility $= 0.3$
#$\lambda = 0.15$
#Maturity Time $= \{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 6.5, 10.5\}$
#$m = \{2, 5, 10, 20, 50, 100, 150, 200\}$

Deltatime	price	err%	err% std
$m = 2$			
0.5	0.0279	0.70	0.98
1	0.0444	0.64	0.91
1.5	0.0582	0.61	0.86
2	0.0693	0.59	0.83
2.5	0.0795	0.57	0.81
3	0.0914	0.56	0.78
3.5	0.0994	0.55	0.78
6.5	0.1407	0.52	0.74
10.5	0.1832	0.51	0.71
Deltatime	price	err%	err% std
$m = 5$			
0.5	0.0293	0.65	0.92
1	0.0462	0.61	0.87
1.5	0.0591	0.60	0.85
2	0.0698	0.59	0.83
2.5	0.0796	0.58	0.82
3	0.0889	0.57	0.81
3.5	0.0969	0.57	0.81
6.5	0.1352	0.55	0.77
10.5	0.1743	0.52	0.73
Deltatime	price	err%	err% std
$m = 10$			
0.5	0.0293	0.66	0.93
1	0.0459	0.62	0.87
1.5	0.0590	0.60	0.85
2	0.0704	0.59	0.84
2.5	0.0806	0.58	0.82
3	0.0885	0.58	0.81
3.5	0.0968	0.57	0.80
6.5	0.1371	0.54	0.76
10.5	0.1732	0.51	0.73

Deltatime	price	err%	err% std
$m = 20$			
0.5	0.0293	0.66	0.93
1	0.0459	0.62	0.88
1.5	0.0581	0.61	0.87
2	0.0702	0.59	0.84
2.5	0.0796	0.58	0.82
3	0.0898	0.57	0.80
3.5	0.0972	0.56	0.80
6.5	0.1361	0.54	0.76
10.5	0.1734	0.52	0.73
Deltatime	price	err%	err% std
$m = 50$			
0.5	0.0294	0.67	0.94
1	0.0457	0.63	0.89
1.5	0.0594	0.61	0.86
2	0.0701	0.59	0.84
2.5	0.0801	0.58	0.83
3	0.0896	0.57	0.81
3.5	0.0984	0.57	0.80
6.5	0.1374	0.54	0.76
10.5	0.1729	0.52	0.73
Deltatime	price	err%	err% std
$m = 100$			
0.5	0.0291	0.66	0.94
1	0.0453	0.63	0.90
1.5	0.0586	0.61	0.86
2	0.0707	0.59	0.83
2.5	0.0805	0.58	0.82
3	0.0898	0.57	0.81
3.5	0.0982	0.57	0.80
6.5	0.1383	0.54	0.76
10.5	0.1760	0.51	0.73

Deltatime	price	err%	err% std
$m = 150$			
0.5	0.0293	0.67	0.94
1	0.0462	0.63	0.89
1.5	0.0588	0.61	0.86
2	0.0712	0.59	0.83
2.5	0.0808	0.58	0.82
3	0.0900	0.58	0.82
3.5	0.0974	0.57	0.81
6.5	0.1380	0.54	0.76
10.5	0.1758	0.51	0.72
Deltatime	price	err%	err% std
$m = 200$			
0.5	0.0287	0.67	0.95
1	0.0456	0.63	0.89
1.5	0.0590	0.61	0.86
2	0.0709	0.59	0.83
2.5	0.0809	0.58	0.82
3	0.0897	0.58	0.81
3.5	0.0969	0.57	0.81
6.5	0.1381	0.54	0.76
10.5	0.1755	0.52	0.73

6.4 Tabella D

Dati ottenuti dalla simulazione di pricing per T e m variabili con formula di Eulero.

#N simulazioni 30000
#$S_0 = 100$
#strikeprice $= 0.02$
#rate $= 0.02$
#volatility $= 0.3$
#$\lambda = 0.15$
#Maturity Time $= \{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 6.5, 10.5\}$
#$m = \{2, 5, 10, 20, 50, 100, 150, 200\}$

Deltatime	price	err%	err% std
$m = 2$			
0.5	0.0315	0.64	0.91
1	0.0520	0.58	0.83
1.5	0.0696	0.54	0.77
2	0.0852	0.52	0.74
2.5	0.0997	0.50	0.71
3	0.1138	0.49	0.69
3.5	0.1270	0.47	0.67
6.5	0.1929	0.43	0.61
10.5	0.2591	0.41	0.58
Deltatime	price	err%	err% std
$m = 5$			
0.5	0.0331	0.61	0.86
1	0.0534	0.56	0.79
1.5	0.0706	0.54	0.76
2	0.0857	0.52	0.74
2.5	0.0997	0.51	0.72
3	0.1139	0.50	0.70
3.5	0.1250	0.49	0.69
6.5	0.1913	0.44	0.62
10.5	0.2579	0.41	0.58
Deltatime	price	err%	err% std
$m = 10$			
0.5	0.0330	0.62	0.87
1	0.0529	0.57	0.81
1.5	0.0703	0.55	0.78
2	0.0858	0.52	0.74
2.5	0.0999	0.51	0.73
3	0.1134	0.50	0.70
3.5	0.1259	0.49	0.69
6.5	0.1913	0.44	0.62
10.5	0.2608	0.40	0.57

Deltatime	price	err%	err% std
$m = 20$			
0.5	0.0329	0.62	0.88
1	0.0534	0.58	0.82
1.5	0.0710	0.55	0.77
2	0.0863	0.53	0.75
2.5	0.1009	0.51	0.73
3	0.1130	0.50	0.71
3.5	0.1274	0.49	0.69
6.5	0.1919	0.44	0.62
10.5	0.2605	0.40	0.57
Deltatime	price	err%	err% std
$m = 50$			
0.5	0.0330	0.63	0.88
1	0.0534	0.58	0.82
1.5	0.0712	0.55	0.77
2	0.0871	0.53	0.75
2.5	0.1015	0.51	0.72
3	0.1143	0.50	0.71
3.5	0.1276	0.49	0.69
6.5	0.1927	0.44	0.62
10.5	0.2629	0.40	0.57
Deltatime	price	err%	err% std
$m = 100$			
0.5	0.0327	0.63	0.89
1	0.0536	0.58	0.82
1.5	0.0712	0.55	0.77
2	0.0863	0.53	0.75
2.5	0.1009	0.51	0.73
3	0.1145	0.50	0.71
3.5	0.1275	0.49	0.69
6.5	0.1915	0.45	0.63
10.5	0.2632	0.40	0.57

Deltatime	price	err%	err% std
$m = 150$			
0.5	0.0333	0.62	0.88
1	0.0539	0.57	0.82
1.5	0.0711	0.55	0.78
2	0.0863	0.53	0.75
2.5	0.1009	0.52	0.73
3	0.1152	0.50	0.71
3.5	0.1286	0.49	0.69
6.5	0.1938	0.44	0.63
10.5	0.2634	0.40	0.57
Deltatime	price	err%	err% std
$m = 200$			
0.5	0.0328	0.63	0.89
1	0.0539	0.58	0.81
1.5	0.0712	0.55	0.78
2	0.0868	0.53	0.75
2.5	0.1013	0.51	0.73
3	0.1153	0.50	0.71
3.5	0.1279	0.49	0.70
6.5	0.1925	0.45	0.63
10.5	0.2626	0.41	0.57

6.5 Tabella E

Dati ottenuti dalla simulazione di pricing per λ e m variabili con formula esatta a estrazione della RV binaria.

#N simulazioni 30000
#$S_0 = 100$
#strikeprice = 0.02
#rate = 0.02
#volatility = 0.3
#Maturity Time 0 years 6 months 0 days 0 h 0 min 0 sec
#$\lambda = \{0, 0.15, 0.3, 0.6, 1.2\}$
#$m = \{2, 5, 10, 20, 50, 100, 150, 200\}$

lambda	price	err%	err% std
$m = 2$			
0	0	0	0
0.15	0.0282	0.68	0.96
0.3	0.0521	0.63	0.89
0.6	0.0743	0.63	0.89
1.2	0.0792	0.65	0.92
lambda	price	err%	err% std
$m = 5$			
0	0	0	0
0.15	0.0293	0.64	0.92
0.3	0.0528	0.62	0.88
0.6	0.0736	0.62	0.89
1.2	0.0798	0.63	0.89
lambda	price	err%	err% std
$m = 10$			
0	0	0	0
0.15	0.0292	0.65	0.92
0.3	0.0527	0.63	0.89
0.6	0.0742	0.62	0.88
1.2	0.0792	0.62	0.88
lambda	price	err%	err% std
$m = 20$			
0	0	0	0
0.15	0.0291	0.66	0.94
0.3	0.0531	0.63	0.89
0.6	0.0742	0.61	0.87
1.2	0.0793	0.61	0.87
lambda	price	err%	err% std
$m = 50$			
0	0	0	0
0.15	0.0293	0.66	0.94
0.3	0.0532	0.63	0.90
0.6	0.0750	0.62	0.88
1.2	0.0786	0.61	0.87
lambda	price	err%	err% std
$m = 100$			
0	0	0	0
0.15	0.0292	0.66	0.93
0.3	0.0536	0.63	0.90
0.6	0.0748	0.62	0.87
1.2	0.0797	0.61	0.87

lambda	price	err%	err% std
$m = 150$			
0	0	0	0
0.15	0.0295	0.66	0.92
0.3	0.0533	0.63	0.90
0.6	0.0758	0.61	0.86
1.2	0.0795	0.61	0.87
lambda	price	err%	err% std
$m = 200$			
0	0	0	0
0.15	0.0294	0.66	0.94
0.3	0.0528	0.64	0.90
0.6	0.0754	0.61	0.87
1.2	0.0791	0.62	0.87

6.6 Tabella F

Dati ottenuti dalla simulazione di pricing per T e m variabili con formula esatta a estrazione della RV binaria.

#N simulazioni 30000
#$S_0 = 100$
#strikeprice $= 0.02$
#rate $= 0.02$
#volatility $= 0.3$
#$\lambda = 0.15$
#Maturity Time $= \{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 6.5, 10.5\}$
#$m = \{2, 5, 10, 20, 50, 100, 150, 200\}$

Deltatime	price	err%	err% std
$m = 2$			
0.5	0.0280	0.69	0.97
1	0.0450	0.64	0.89
1.5	0.0582	0.61	0.86
2	0.0698	0.59	0.82
2.5	0.0807	0.57	0.80
3	0.0901	0.56	0.79
3.5	0.0988	0.55	0.78
6.5	0.1426	0.52	0.73
10.5	0.1825	0.51	0.72

Deltatime	price	err%	err% std
$m = 5$			
0.5	0.0294	0.65	0.92
1	0.0457	0.62	0.88
1.5	0.0591	0.60	0.85
2	0.0708	0.58	0.82
2.5	0.0794	0.58	0.82
3	0.0886	0.57	0.81
3.5	0.0966	0.57	0.80
6.5	0.1371	0.54	0.77
10.5	0.1746	0.51	0.73
Deltatime	price	err%	err% std
$m = 10$			
0.5	0.0295	0.66	0.93
1	0.0457	0.62	0.88
1.5	0.0584	0.61	0.85
2	0.0700	0.59	0.83
2.5	0.0802	0.58	0.81
3	0.0879	0.58	0.81
3.5	0.0969	0.57	0.80
6.5	0.1367	0.54	0.75
10.5	0.1744	0.51	0.73
Deltatime	price	err%	err% std
$m = 20$			
0.5	0.0289	0.67	0.93
1	0.0459	0.62	0.89
1.5	0.0588	0.61	0.86
2	0.0701	0.59	0.84
2.5	0.0806	0.58	0.82
3	0.0897	0.57	0.81
3.5	0.0964	0.57	0.80
6.5	0.1381	0.54	0.76
10.5	0.1750	0.51	0.73

Deltatime	price	err%	err% std
$m = 50$			
0.5	0.0290	0.67	0.94
1	0.0455	0.63	0.88
1.5	0.0591	0.61	0.85
2	0.0703	0.59	0.84
2.5	0.0799	0.58	0.83
3	0.0889	0.57	0.81
3.5	0.0981	0.57	0.80
6.5	0.1372	0.54	0.76
10.5	0.1749	0.51	0.72
Deltatime	price	err%	err% std
$m = 100$			
0.5	0.0294	0.66	0.94
1	0.0454	0.63	0.89
1.5	0.0592	0.60	0.85
2	0.0700	0.60	0.84
2.5	0.0811	0.58	0.82
3	0.0900	0.58	0.82
3.5	0.0976	0.57	0.81
6.5	0.1372	0.54	0.76
10.5	0.1759	0.51	0.72
Deltatime	price	err%	err% std
$m = 150$			
0.5	0.0294	0.66	0.95
1	0.0458	0.63	0.89
1.5	0.0588	0.61	0.86
2	0.0706	0.59	0.83
2.5	0.0805	0.58	0.82
3	0.0901	0.57	0.81
3.5	0.0976	0.57	0.80
6.5	0.1365	0.54	0.77
10.5	0.1759	0.52	0.73

Deltatime	price	err%	err% std
$m = 200$			
0.5	0.0295	0.67	0.94
1	0.0460	0.63	0.89
1.5	0.0595	0.61	0.86
2	0.0711	0.59	0.85
2.5	0.0813	0.58	0.82
3	0.0906	0.57	0.80
3.5	0.0981	0.57	0.80
6.5	0.1381	0.54	0.76
10.5	0.1743	0.52	0.72

6.7 Specifiche

Relazione sviluppata con LaTeX.
Reference della libreria generato con Doxygen.
Grafici di studio delle funzioni generati con Mathematica.
Grafici di illustrazione dei dati ottenuti generati con GNUplot.
Grafico che simula l'andamento dell'azione generato con Root(CERN).

6.8 Bibliografia

Elementi di finanza, metodi numerici e loro implementazione, Marco Airoldi [Version 7.9].
Wikipedia.