POLITECNICO DI MILANO
Corso di Laurea Magistrale di Ingegneria Matematica
Facoltà di Ingegneria dei Sistemi

Progetto di Programmazione Avanzata per il Calcolo Scientifico:

Metodo a elementi finiti per il pricing di opzioni multi-asset con modelli di Lévy

Nahuel Foresta, matr. 798775
Giorgio G. Re, matr. 799260

Anno Accademico 2012-2013

An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem.

John Tukey

Indice

Todo list

Introduzione

Capitolo 1

Modello di Black & Scholes

1.1 Introduzione

In questo capitolo descriviamo i modelli basilari utilizzati per descrivere il mercato finanziario, seguendo le argomentazioni di Merton (1973), trattate in [2]. Consideriamo quindi un mercato finanziario molto semplificato, costituito da un titolo *risk-free* descritto dal processo B e un titolo azionario con valore pari al processo S. Definiamo quindi questi due processi.

Definizione 1.1. Sia $(\Omega, \mathcal{F}, \mu)$ uno spazio misurabile e sia $\mathcal{F}_{t\in[0,T]}$ una filtrazione. Allora, il processo B descrive il valore di un titolo *risk-free* se la sua dinamica è del tipo:

$$dB(t) = r(t)B(t)dt,$$

dove r è un qualsiasi processo $\mathcal{F}_t$-adattato.

La caratteristica più importante quindi dei processi *risk-free* è l'assenza di aleatorietà data da un processo stocastico casuale. Integrando l'equazione precedente, otteniamo:

$$B(t) = B(0)\int_0^t r(s)ds.$$

Un caso particolare è quello in cui r è una costante deterministica, in tal modo B descrive l'andamento di un'obbligazione.

Assumiamo poi che la dinamica di S sia data da:

$$dS(t) = S(t)\mu(t, S(t))dt + S(t)\sigma(t, S(t))dW(t),$$

in cui W_t è un processo di Wiener (cioè, un moto browniano) e μ e σ due funzioni deterministiche. La funzione σ è detta volatilità del titolo, μ è il tasso di ritorno di S. Passiamo ora a definire il modello di *Black&Scholes*.

Definizione 1.2. Il modello di *Black&Scholes* consiste di due titoli con le seguenti dinamiche:

$$\begin{aligned} dB(t) &= rB(t)dt, \\ dS(t) &= \mu S(t)dt + \sigma S(t)dW(t), \end{aligned}$$

dove r, μ e σ sono costanti deterministiche.

1.2 Strumenti derivati e Opzioni

In questa sezione definiamo gli strumenti derivati e, in particolare le opzioni che abbiamo trattato nel progetto.

Definizione 1.3. In finanza, è denominato strumento derivato ogni contratto o titolo il cui valore si basa sul valore di mercato di un altro titolo o strumento finanziario, detto sottostante (ad esempio, azioni, valute, tassi di interesse o derivati stessi).

Definiamo ora il titolo derivato più semplice, ovvero l'opzione *call* europea.

Definizione 1.4. Un'opzione *call* europea con prezzo di esercizio (o *strike price*) K e scadenza T sul sottostante S è un contratto finanziario derivato con le seguenti caratteristiche:

- il titolare del contratto ha, al tempo T, il diritto di acquistare un'azione del sottostante al prezzo K dal sottoscrittore del contratto, qualsiasi sia il valore del sottostante S al tempo T;
- il titolare del contratto non ha alcun obbligo di acquistare un'azione del sottostante al tempo T;
- il diritto di acquistare un'azione del sottostante può essere esercitato solo al tempo T.

Osserviamo che la scadenza del contratto e il prezzo d'esercizio sono stabiliti alla stipula del contratto, che per noi sarà tipicamente $t = 0$.
Oltre alle opzioni *call* europee, esistono opzioni *put* europee, le quali danno al titolare del contratto il diritto a vendere (anziché comprare) un dato titolo azionario a un prezzo fissato K. Le opzioni americane invece (*call* o *put* che siano), permettono di esercitare il diritto all'acquisto o alla vendita dell'azione in ogni istante di tempo $t \in [0, T]$.

Esempio 1. Supponiamo di possedere un'opzione *call* con scadenza $T = 1$ anno, *strike price* $K = 100€$ e un sottostante che al tempo $t = 0$ vale $S_0 = 100€$. Allora, se fra un anno $S_T = 120€$, eserciteremo l'opzione, acquistando il sottostante per un prezzo pari a $K = 100€$, e il sottoscrittore del contratto pagherà i rimenenti $S_T - K = 20€$. Se invece $S_T = 80€$ non eserciteremo l'opzione, ottenendo un guadagno pari a 0.

Osserviamo quindi che il valore a scadenza, cioè il *payoff*, dell'opzione dipende soltanto dal valore del sottostante. Definiamo quindi il *payoff* come variabile aleatoria in funzione di S.

Definizione 1.5. Sia S il processo stocastico che descrive l'andamento di un titolo azionario, allora il *payoff* di un'opzione scritta su S con scadenza T e *strike* K è una variabile aleatoria $\mathcal{X} \in \mathcal{F}_T$, e:

$$\mathcal{X} = \Phi(S_T).$$

Per esempio, il *payoff* delle opzioni call e put europee è:

$$\Phi(S_T) = \max(S_T - K, 0),$$
$$\Phi(S_T) = \max(K - S_T, 0).$$

La domanda che ci poniamo ora è la seguente: qual è il prezzo equo di un'opzione? Ovvero, quanto occorre pagare oggi per avere il diritto ma non l'obbligo di acquistare al tempo T un'azione a un prezzo fissato K?

1.3 L'equazione di *Black&Scholes*

Vi sono molti modi per ricavare l'equazione di *Black&Scholes*: presentiamo qui il modo più semplice e veloce.

Prima di ricavare l'equazione spendiamo qualche riga per descrivere il concetto di neutralità al rischio in finanza. Un operatore economico si dice neutrale al rischio quando le sue preferenze lo rendono indifferente al compiere un'azione il cui risultato è una quantità aleatoria, oppure compiere un'azione il cui risultato è il valore atteso della quantità aleatoria stessa.
Per esempio, per un soggetto neutrale al rischio sono indifferenti le seguenti situazioni:

- avere 1€ con probabilità 1;
- giocare a una lotteria in cui il soggetto può ricevere 2€ con probabilità $1/2$ o 0€ con probabilità $1/2$.

Nel primo caso infatti, egli ottiene sempre 1€, nel secondo ottiene in media 1€. Perciò per un soggetto neutrale al rischio queste situazioni sono indifferenti.
Quando ci occupiamo di *pricing* di derivati, ci poniamo sempre nell'ipotesi di neutralità al rischio. In particolare,

1. assumiamo che il termine di deriva μ del modello S sia pari al tasso di interesse *risk-free* r, cioè poniamo:
$$dS(t) = rS(t)dt + \sigma S(t)dW(t);$$
2. calcoliamo il valore atteso del *payoff*, cioè $\mathbb{E}\left[\Phi(S_T)|\mathcal{F}_t\right]$;
3. scontiamo il valore atteso del *payoff* con il tasso di interesse r.

Sia quindi $C : \mathbb{R}^+ \times [0,T] \to \mathbb{R}^+$, $C = C(S(t),t)$ il processo stocastico che descrive il valore di un'opzione. In particolare, se consideriamo una *call* europea,
$$C(S,t) = e^{(-r(T-t))}\mathbb{E}\left[\max(S_T - K, 0)|\mathcal{F}_t\right].$$
Data la dinamica di S, se applichiamo a questa quantità il Lemma di Itô, otteniamo la seguente equazione:
$$dC(S,t) = \left(\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}S^2\frac{\partial^2 C}{\partial S^2}\right)dt + \sigma S\frac{\partial C}{\partial S}dW(t).$$
Dall'altro lato però, per la *risk-neutrality*, la deriva di C, come quella di ogni titolo finanziario, dovrà essere pari a rC, quindi:
$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}S^2\frac{\partial^2 C}{\partial S^2} = rC.$$

Riassumendo, posti $C = C(S,t)$ e $P = P(S,t)$ i processi che descrivono il prezzo di opzioni *call* e *put*, otteniamo le seguenti equazioni alle derivate parziali con le rispettive condizioni finali e condizioni al bordo:

$$\begin{cases} \dfrac{\partial C}{\partial t} + rS\dfrac{\partial C}{\partial S} + \dfrac{1}{2}S^2\dfrac{\partial^2 C}{\partial S^2} = rC, \\ C(S,T) = \max(S_T - K, 0), \\ C(0,t) = 0, \qquad\qquad \forall t \in [0,T], \\ \lim\limits_{S\to\infty} C(S,t) = \infty, \qquad \forall t \in [0,T], \end{cases} \tag{1.1}$$

e

$$\begin{cases} \dfrac{\partial P}{\partial t} + rS\dfrac{\partial P}{\partial S} + \dfrac{1}{2}P^2\dfrac{\partial^2 P}{\partial S^2} = rP, \\ P(S,T) = \max(K - S_T, 0), \\ P(0,t) = K, \qquad\qquad \forall t \in [0,T] \\ \lim\limits_{S\to\infty} P(S,t) = 0, \qquad \forall t \in [0,T]. \end{cases} \tag{1.2}$$

Come possiamo osservare, si tratta di equazioni paraboliche *backward* con dato finale. Il prezzo dell'opzione sarà dato dalla soluzione C o P, valutate in $S_t = S_0$, ovvero il valore dell'azione oggi, e in $t = 0$.

1.4 Opzioni Basket

Un altro tipo di opzioni scambiate sui mercati finanziari sono le opzioni basket, il cui *payoff* dipende cioè da due o più sottostanti. In particolare, in questo progetto ci siamo concentrati sul *pricing* di opzioni basket 2D, con i seguenti valori finali:

$$C(S_1, S_2, T) = \max(S_{1,T} + S_{2,T} - K, 0)$$

per la *call* e:

$$P(S_1, S_2, T) = \max(K - S_{1,T} - S_{2,T}, 0)$$

per la *put*, dove $S_{1,T}$ e $S_{2,T}$ sono i valori al tempo T dei due sottostanti S_1 e S_2. L'equazione che si ottiene con procedimenti analoghi a quelli mostrati nella sezione precedente è la seguente:

$$\frac{\partial C}{\partial t} + rS_1\frac{\partial C}{\partial S_1} + rS_2\frac{\partial C}{\partial S_2} + \frac{\sigma_1^2}{2}S_1^2\frac{\partial^2 C}{\partial S_1^2} + \frac{\sigma_2^2}{2}S_2^2\frac{\partial^2 C}{\partial S_2^2} + \rho\sigma_1\sigma_2\frac{\partial^2 C}{\partial S_1 \partial S_2} = rC, \tag{1.3}$$

in cui $C : \mathbb{R}^+ \times \mathbb{R}^+ \times [0,T] \to \mathbb{R}^+$, $C = C(S_1(t), S_2(t), t)$, σ_1 e σ_2 sono le volatilità dei due sottostanti e ρ è il coefficiente di correlazione fra S_1 e S_2.

1.5 Opzioni Americane: il problema con frontiera libera

Le opzioni americane differiscono dalle europee poiché consentono di esercitare l'opzione non solo in T, bensì in qualsiasi istante di tempo dalla stipula del contratto alla sua scadenza. Quindi, proprio perché danno al titolare del contratto dei diritti aggiuntivi, è facile capire che vale la seguente relazione:

$$V^{Am} \geq V^{Eu},$$

ovvero il valore di un'opzione americana è sempre superiore al valore dell'europea corrispondente. In particolare, il valore dell'opzione americana è sempre pari o superiore al valore del *payoff*. L'opzione europea infatti, può avere un valore inferiore al *payoff*, ma questo non può succedere per le americane. Infatti, se così non fosse, potremmo acquistare un'azione e una *put* su questa azione ed esercitare immediatamente, ottenendo un guadagno certo senza correre alcun rischio. Per questo valgono i seguenti vincoli:

Spieghiamo meglio o ce ne strafottiamo?

$$C^{Am}(S,t) \geq \max(S_T - K, 0) \qquad \forall t \in [0,T], \tag{1.4}$$

$$P^{Am}(S,t) \geq \max(K - S_T, 0) \qquad \forall t \in [0,T]. \tag{1.5}$$

Consideriamo ora il comportamento della *put* nella parte sinistra del grafico

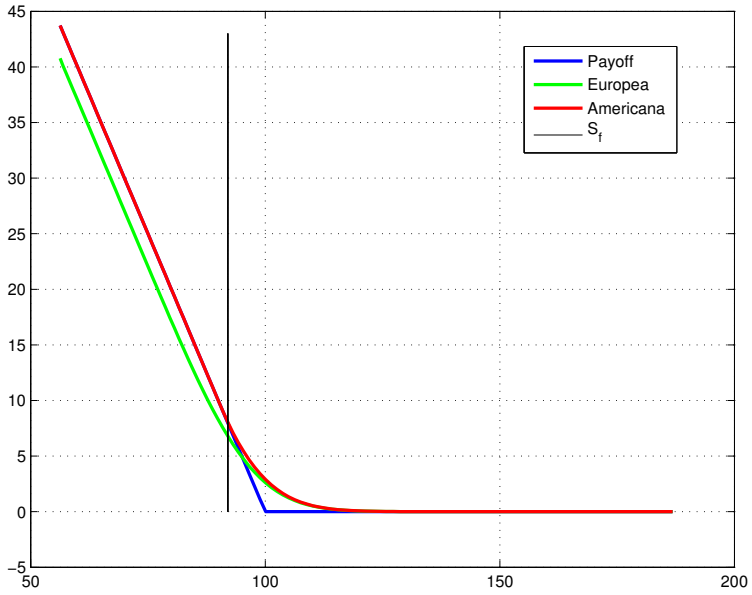


Figura 1.1: Opzioni Americane vs. Opzione Europee

riportato in figura 1.1. Senza la possibiltà di esercizio anticipato, $P^{Eu} < K - S$, ma per la disuguaglianza 1.5, $P^{Am} = K - S$. Nella parte destra della curva, invece, vale $P^{Am} \geq \max(K - S, 0)$. Quindi, per la continuità e la monotonia di P^{Am}, la curva dovrà toccare il *payoff* in un punto $S_f(t)$, $0 < S_f(t) < K$. Questo punto è definito da:

$$P^{Am}(S,t) > \max(K - S, 0) \qquad S > S_f(t),$$
$$P^{Am}(S,t) = K - S \qquad\qquad S < S_f(t).$$

Quindi, $\forall t \in [0,T]$, dobbiamo determinare il punto $S_f(t)$, attraverso il quale passa la retta che separa l'area in cui $P^{Am} =$*payoff* da qualla in cui $P^{Am} >$*payoff*. Poiché a priori questa frontiera è ignota, questo problema è detto "a frontiera

libera".

Per le *call* americane la situazione è differente, poiché $C^{Eu} \geq \max(S_T - K, 0)$[1]. Perciò il prezzo di una *call* americana è identico a quello di un'europea e in questo caso non si pone il problema con frontiera libera.

Formalmente, l'equazione differenziale che occorre risolvere per trovare il prezzo di una *put* americana è la seguente:

$$\begin{cases} \dfrac{\partial P}{\partial t} + rS\dfrac{\partial P}{\partial S} + \dfrac{1}{2}P^2\dfrac{\partial^2 P}{\partial S^2} \leq rP, \\ P(S,t) \geq \max(K - S_T, 0), \\ P(S,T) = \max(K - S_T, 0), \\ P(0,t) = K, \qquad\qquad \forall t \in [0,T] \\ \lim\limits_{S\to\infty} P(S,t) = 0, \qquad \forall t \in [0,T]. \end{cases} \tag{1.6}$$

1.6 Difetti del modello di *Black&Scholes*

Il modello di *Black&Scholes*, nonostante sia molto utilizzato in finanza, presenta alcuni problemi e può essere pericoloso utilizzarlo per valutare strumenti derivati. In particolare, empiricamente, si evidenziano le seguenti problematiche:

- il valore di S può essere discontinuo, ovvero è possibile che il sottostante presenti dei "salti";
- le code della distribuzione dei log-rendimenti dovrebbero essere normali, ma non lo sono: i valori delle code sono infatti più probabili di quanto ipotizzato dal modello, inoltre le code non sono simmetriche poiché presentano un'asimmetria verso i rendimenti negativi;
- i log-rendimenti inoltre non hanno distribuzioni indipendenti: nelle serie storiche si osservano infatti i cosiddetti *cluster*, ovvero periodi in cui la volatilità è alta e rimane alta oppure periodi con una volatilità che rimane per ridotta per lungo tempo;
- in figura 1.2 è rappresentato il grafico volatilità vs. *moneyness*[2]. In esso possiamo osservare che il valore della volatilità non rimane costante al variare dei prezzi di esercizio, bensì presenta una forma convessa. Questo fenomeno è noto come *smile* di volatilità.

Nel prossimo capitolo introdurremo una classe più ampia di processi stocastici, cioè i processi di Lévy, e descriveremo dei modelli che permettono di risolvere alcuni dei problemi sopra descritti. Questi modelli danno luogo a equazioni simili a quella di *Black&Scholes* (PDE) con l'aggiunta però di un termine integrale, per questo le chiameremo Equazioni Integro-Differenziali alle Derivate Parziali (PIDE).

Qui, 1. è da mettere il disegno? 2. Visto che merton e kou non risolvono questo problema, conviene parlare dello smile o lo saltiamo?

[1] Questa relazione si calcola immediatamente sfruttando alcune semplici relazioni che legano C, S e K. In particolare, nel qual caso: $C^{Am} \geq C^{Eu} \geq S - Ke^{-r(T-t)} > S - K$

[2] La *moneyness* di un'opzione è il rapporto fra prezzo del sottostante in $t = 0$ e *Strike*, ovvero S_0/K.

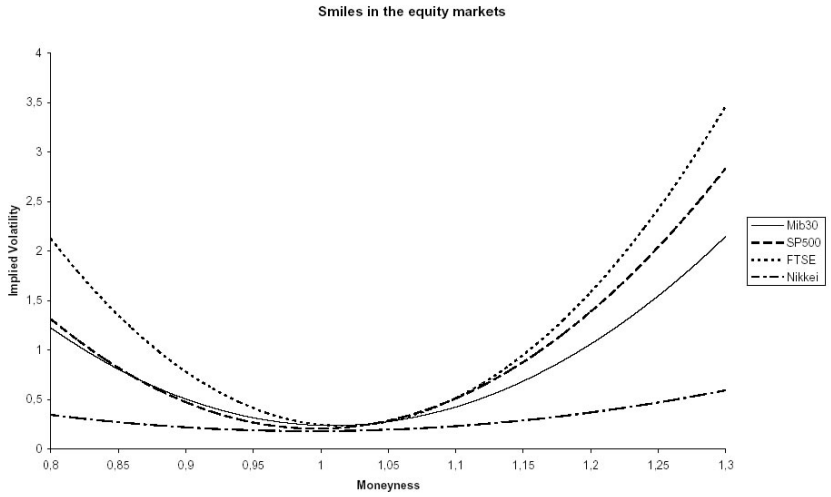


Figura 1.2: *Smile* di volatilità

Capitolo 2

Processi di Lévy e Modelli di Kou e Merton

2.1 Introduzione

In questo secondo capitolo introduciamo i processi di Lévy, una classe più ampia di processi stocastici che permettono di descrivere con più accuratezza il comportamento di un titolo azionario. Elenchiamo ora alcune definizioni e un teorema che ci permettono di definire i nuovi modelli.

Definizione 2.1. Sia $(\Omega, \mathcal{F}, \mu)$ uno spazio misurabile e sia $\mathcal{F}_{t\in[0,T]}$ una filtrazione. Sia X_t un processo stocastico *cadlag*[1], allora X_t è di Lévy se:

1. $X_0 = 0$,
2. ha incrementi indipendenti,
3. ha incrementi stazionari,
4. c'è continuità stocastica, ovvero:

$$\forall \varepsilon > 0, \qquad \lim_{h\to 0} \mathbb{P}(|X_{t+h} - X_t| \geq \varepsilon) = 0.$$

Definizione 2.2. Sia X_t un Lévy, allora poniamo

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

$\forall A \in \mathcal{B}$, e chiamiamo $\nu(A)$ la misura di Lévy di X_t.

Definizione 2.3. Sia X_t un processo di Lévy, allora X_t è un *Compound Poisson* di intensità λ e distribuzione di salti f se:

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

dove $N_t \sim Poisson(\lambda)$ e $Y_i \sim f$, Y_i i.i.d. $\forall i$.

[1] Ricordiamo che un processo è detto *cadlag* se ha traiettorie continue a destra e limitate a sinistra.

Teorema 2.1. *Decomposizione di Lévy-Itô per processi ad attività finita.*
Sia X_t un processo di Lévy con misura ν finita, allora esistono due costanti γ e σ tali che:

$$X_t = \gamma t + \sigma W_t + X_t^C,$$

dove W_t è un moto browniano e X_t^C un Compound Poisson.

Perciò un processo di Lévy è determinato univocamente dalla sua tripletta caratteristica (γ, σ, ν).

2.2 Modelli di Merton e Kou

Passiamo ora a definire i modelli di Merton e Kou. In entrambi questi modelli il prezzo dell'azione è descritto dalla seguente equazione:

$$S_t = S_0 e^{rt+X_t}, \tag{2.1}$$

dove r è il tasso di interesse e X_t è un Lévy ad attività finita, ovvero

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

Nel modello di Merton, $Y_i \sim \mathcal{N}(\mu, \delta^2)$, e la misura di Lévy è data da:

$$\nu(x) = \frac{\lambda}{\sqrt{2\pi\delta^2}} exp\left\{-\frac{(x-\mu)^2}{2\delta^2}\right\}, \tag{2.2}$$

in cui λ è l'intensità del *Poisson*.
Nel modello di Kou invece, le Y_i sono delle esponenziali con parametri diversi per salti positivi e negativi. In particolare,

$$\nu(x) = p\lambda\lambda_+ e^{\lambda_+ x}\mathcal{I}_{x>0} + (1-p)\lambda\lambda_- e^{-\lambda_- x}\mathcal{I}_{x<0},$$

dove p è la probabilità di salti positivi, λ è il solito parametro del *Poisson*, λ_+ e λ_- sono invece le intensità dei salti positivi e negativi.

2.3 *Pricing* con modelli Exponential Lévy

Riportiamo ora un risultato che permette di individuare un'equazione differenziale che permetta di risolvere il problema di *pricing* descritto nel capitolo precedente.

Teorema 2.2. *Sia S_t nella forma 2.1 con l'ipotesi:*

$$\int_{|x|>1} e^{2x}\nu(dx) < \infty.$$

Sia $C : \mathbb{R}^+ \times [0,T] \to \mathbb{R}^+$, $C = C(S_t, t)$ nella forma:

$$C(S_t, t) = e^{-r(T-t)}\mathbb{E}(\Phi(S_t)),$$

dove Φ *è un* payoff Lipshitz *che dipende dall'unico sottostante* S_t*. Allora* C *soddisfa l'equazione:*

$$\frac{\partial C}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 C}{\partial S^2} + r \frac{\partial C}{\partial S} - rC + \\ + \int_{\mathbb{R}} \left(C(t, Se^y) - C(t,S) - S(e^y - 1) \frac{\partial C}{\partial S}(t,S) \right) \nu(dy) = 0. \quad (2.3)$$

Per quanto riguarda opzioni su due *asset*, nell'ipotesi che le parti di salto dei due sottostanti siano indipendenti fra di loro, l'equazione 1.3 diventa:

$$\frac{\partial C}{\partial t} + rS_1 \frac{\partial C}{\partial S_1} + rS_2 \frac{\partial C}{\partial S_2} + \frac{\sigma_1^2}{2} S_1^2 \frac{\partial^2 C}{\partial S_1^2} + \frac{\sigma_2^2}{2} S_2^2 \frac{\partial^2 C}{\partial S_2^2} + \rho \sigma_1 \sigma_2 S_1 S_2 \frac{\partial^2 C}{\partial S_1 \partial S_2} - rC \\ + \int_{\mathbb{R}} \left(C(t, S_1 e^y, S_2) - C(t, S_1, S_2) - S_1(e^y - 1) \frac{\partial C}{\partial S_1}(t, S_1, S_2) \right) \nu_1(dy) \\ + \int_{\mathbb{R}} \left(C(t, S_1, S_2 e^y) - C(t, S_1, S_2) - S_2(e^y - 1) \frac{\partial C}{\partial S_2}(t, S_1, S_2) \right) \nu_2(dy) = 0, \quad (2.4)$$

dove S_1 e S_2 sono i due sottostanti descritti da modelli Exponential Lévy, rispettivamente con misure ν_1 e ν_2.
Nel prossimo capitolo, presentiamo dei metodi numerici che trovino una soluzione approssimata per le equazioni presentate in questi due capitoli.

Capitolo 3

Metodi numerici per PDE e PIDE

3.1 Introduzione

In questo capitolo descriviamo i metodi numerici utilizzati nel codice che abbiamo prodotto per approssimare le soluzioni dei problemi differenziali descritti sopra. Prima di procedere però vorremmo parlare di come sono trattate queste equazioni in finanza. Generalmente, in questo campo, si utilizzano sempre metodi basati sulle differenze finite. Noi, invece, abbiamo deciso di proporre un approccio a elementi finiti poiché riteniamo che, pur utilizzando domini per nulla complessi, i vantaggi degli elementi finiti siano evidenti anche per questo tipo di problemi, primo fra tutti la possibilità di raffinare e anche "de-raffinare" la *mesh* dove la soluzione lo richiede. Con questo tipo di approccio poi, non dovrebbe essere difficile estendere il problema al 3d, avendo cura di trattare correttamente gli integrali. Inoltre, una approccio FEM al problema 2.4 è molto difficile da trovare. In [4] si può trovare un analisi dell'equazione in forma debole e alcuni risultati numerici (calcolati con FreeFem++), ma non vi è nessuna indicazione sugli algoritmi usati e tantomeno il codice. Per questo motivo, per validare i prezzi ottenuti, abbiamo scritto un metodo MonteCarlo per prezzare le opzioni Basket con modelli di Lévy.

Tornando ora all'argomento del capitolo, per discretizzare le equazioni abbiamo utilizzato due cambi di variabile. Il primo, che permette di portare le equazioni a coefficienti costanti presenta l'inconveniente di dover calcolare l'integrale fuori dalla *mesh*. Il secondo invece, lascia l'equazione così com'è, ma permette di calcolare l'integrale nei soli punti della *mesh*. Lasciamo i confronti sulle prestazione dei due cambi di variabile al capitolo dedicato, tuttavia a priori è facile capire che l'assemblaggio delle matrici con il primo cambio di variabile sarà più veloce rispetto al secondo, ma il calcolo dell'integrale sarà inesorabilmente più lento.

3.2 *Log-Prices*

Ci occupiamo ora di mostrare la discretizzazione dell'equazione con il primo cambio di variabile. Trattiamo qui il solo caso della *call* europea, in quanto per la *put* e le americane la discretizzazione è la medesima, a meno di cambiare condizioni al bordo e dati finali.

3.2.1 Equazione 1d

Il primo cambio di variabili che studiamo è il seguente: $x = log(S/S_0)$, che permette di portare l'equazione a coefficienti costanti. Posta quindi $u : \mathbb{R} \times [0,T] \to \mathbb{R}^+$, $u = u(x,t)$, incognita della PIDE monodimensionale, otteniamo:

$$\begin{cases} \dfrac{\partial u}{\partial t} + \left(r - \dfrac{\sigma^2}{2}\right)\dfrac{\partial u}{\partial x} + \dfrac{\sigma^2}{2}\dfrac{\partial^2 u}{\partial x^2} - ru \\ \qquad\qquad + \displaystyle\int_{\mathbb{R}} \left(u(t,x+y) - u(t,x) - (e^y - 1)\frac{\partial u}{\partial x}\right)\nu(dy) = 0, \\ u(x,T) = \max(S_0 e^x - K, 0), \\ \lim\limits_{x\to-\infty} u(x,t) = 0, \qquad \forall t \in [0,T], \\ \lim\limits_{x\to\infty} u(x,t) = \infty, \qquad \forall t \in [0,T], \end{cases} \tag{3.1}$$

per la *call* europea.

3.2.2 Troncamento del dominio

Come è facile osservare $x \in (-\infty, \infty)$, perciò è necessario adottare un troncamento del dominio monodimensionale. Generalmente, in finanza si applica il seguente troncamento:

$$\begin{aligned} S_{min} &= (1-f)S_0 exp\left(\left(r - \frac{\sigma^2}{2}\right)T - 6\sigma\sqrt{T}\right), \\ S_{max} &= (1+f)S_0 exp\left(\left(r - \frac{\sigma^2}{2}\right)T + 6\sigma\sqrt{T}\right), \end{aligned} \tag{3.2}$$

dove $0 \le f \le 1$ è un parametro da scegliersi a piacere (nel codice è settato a 0.5 ma è possibile modificarlo con un apposito metodo). Questo troncamento è utilizzato poiché un *framework Black&Scholes* il sottostante S ha probabilità 10^{-8} di superare quei limiti (quando $f = 0$). Poniamo quindi:

$$x_{min} = log(S_{min}/S_0), \qquad x_{max} = log(S_{max}/S_0).$$

3.2.3 Discretizzazione della PDE

Concentriamoci ora solo sull'equazione senza parte integrale, consideriamo cioè la sola PDE del modello di *Black&Scholes*. Siano quindi Ω_h una triangolazione con $N+1$ nodi di $\Omega = [x_{min}, x_{max}]$ e $\mathcal{T} = \{0 = t_0 \le t_1 \le ... \le t_N = T\}$ una griglia temporale. Cerchiamo una soluzione del tipo:

$$u_h(x,t_n) = \sum_{j=0}^{N} \alpha_j(t_n)\phi_j(x),$$

dove α_j sono i valori della soluzione all'istante t_n nel nodo j della griglia, mentre $\phi_j(x)$ sono le funzioni base dello spazio $\mathbb{P}_h^1 = \{ f \in \mathbb{C}^0 \ \textit{lineari su ogni cella della triangolazione } \Omega_h \ \}$.
Passiamo ora alla formulazione debole del problema 1.1:

$$\sum_{j=1}^{N-1} \int_{\Omega_h} \left(\frac{\partial}{\partial t}(\alpha_j(t)\phi_j(x))\phi_i(x) \right) dx + \sum_{j=1}^{N-1} \left(r - \frac{\sigma^2}{2} \right) \int_{\Omega_h} \alpha_j(t)\phi_j'(x)\phi_i(x)dx$$
$$+ \sum_{j=1}^{N-1} \frac{\sigma^2}{2} \int_{\Omega_h} \alpha_j(t)\phi_j''(x)\phi_i(x)dx = \sum_{j=1}^{N-1} r \int_{\Omega_h} \alpha_j(t)\phi_j(x)\phi_i(x)dx, \quad (3.3)$$

$\forall t \in \mathcal{T}$ e $\forall i \in \{0, ..., N\}$. Applicando inoltre le condizioni al bordo, otteniamo:

$$\alpha_0 = 0, \qquad \alpha_N = S_{max} - K.$$

Applicando ora uno schema di Eulero Implicito per la derivata prima in tempo, giungiamo a:

$$\sum_{j=1}^{N-1} \int_{\Omega_h} \frac{\alpha_j(t_{n+1})\phi_j(x)}{\Delta t}\phi_i(x)dx - \sum_{j=1}^{N-1} \int_{\Omega_h} \frac{\alpha_j(t_n)\phi_j(x)}{\Delta t}\phi_i(x)dx$$
$$+ \sum_{j=1}^{N-1} \left(r - \frac{\sigma^2}{2} \right) \int_{\Omega_h} \alpha_j(t_n)\phi_j'(x)\phi_i(x)dx + \sum_{j=1}^{N-1} \frac{\sigma^2}{2} \int_{\Omega_h} \alpha_j(t_n)\phi_j''(x)\phi_i(x)dx$$
$$= \sum_{j=1}^{N-1} r \int_{\Omega_h} \alpha_j(t_n)\phi_j(x)\phi_i(x)dx, \quad (3.4)$$

$\forall t \in \mathcal{T}$ e $\forall i \in \{0, ..., N\}$. Poniamo infine:

$$a_{ij} = \int_{\Omega_h} \phi_j(x)\phi_i(x)dx,$$
$$b_{ij} = \int_{\Omega_h} \phi_j'(x)\phi_i(x)dx,$$
$$c_{ij} = \int_{\Omega_h} \phi_j'(x)\phi_i'(x)dx.$$

Otteniamo quindi la seguente discretizzazione per la PDE:

$$\sum_{j=1}^{N-1} \alpha_j(t_{n+1}) \frac{a_{ij}}{\Delta t} = \sum_{j=1}^{N-1} \alpha_j(t_n) \left(\left(\frac{1}{\Delta t} + r \right) a_{ij} - \left(r - \frac{\sigma^2}{2} \right) b_{ij} + \frac{\sigma^2}{2} c_{ij} \right).$$

3.2.4 La parte integrale

Concentriamoci ora sulla parte integrale della 3.1:

$$\int_{\mathbb{R}} \left(u(t, x+y) - u(t,x) - (e^y - 1)\frac{\partial u}{\partial x} \right) \nu(dy)$$

e osserviamo che, se la misura di probabilità è assolutamente continua rispetto alla misura di Lesbegue (e ciò è verificato per i modelli di Kou e Merton), si ha che:

$$\nu(dy) = \nu(y)dy.$$

Inoltre, siccome l'integrale della misura sullo spazio è finito nel caso dei *Compound Poisson*, possiamo separare l'integrale in tre parti distinte, e in particolare possiamo porre gli ultimi due addendi dell'integrale pari a:

$$\hat{\lambda} = \int_{\mathbb{R}} \nu(y)dy,$$
$$\hat{\alpha} = \int_{\mathbb{R}} (e^y - 1)\nu(y)dy,$$

ottenendo:

$$\frac{\partial u}{\partial t} + \left(r - \frac{\sigma^2}{2} - \hat{\alpha}\right)\frac{\partial u}{\partial x} + \frac{\sigma^2}{2}\frac{\partial^2 u}{\partial x^2} - (r + \hat{\lambda})u + \int_{\mathbb{R}} u(t, x + y)\nu(y)dy = 0.$$

Sappiamo inoltre dalla teoria che $\hat{\lambda} = \lambda$, poiché integrando la densità di Lévy otteniamo l'intensità dei salti. Per quanto riguarda invece il calcolo numerico dei due integrali, poiché le densità con cui abbiamo a che fare sono gaussiane o esponenziali, abbiamo utilizzato delle formule di quadratura rispettivamente di Gauss-Hermite e di Gauss-Laguerre, come spiegato in [1]. Per esempio, se consideriamo la densità di Lévy di Merton, ν è nella forma 2.2, perciò, a meno di costanti,

$$\begin{aligned}\hat{\alpha} &= \int_{\mathbb{R}} (e^y - 1)Ae^{-B(y-\mu)^2}dy \\ &\simeq A\sum_{j=0}^{Q}(e^{q_j} - 1)w_j,\end{aligned}$$

dove q_j sono i nodi di quadratura su $\mathbb{R}$, mentre w_j sono i pesi che inglobano già il nucleo guassiano. Lo stesso metodo viene utilizzato anche per il nucleo di Kou, con l'accortezza di spezzare l'integrale fra $(-\infty, 0)$ e $(0, \infty)$ poiché il parametro dell'esponenziale è diverso.

Focalizziamoci ora sul calcolo dell'ultimo integrale rimasto, cioè:

$$\int_{\mathbb{R}} u(t, x + y)\nu(y)dy, \tag{3.5}$$

e notiamo subito che dovendo valutare la funzione u nei punti $x+y$, questo è un termine non locale. Esso infatti è difficilmente trattabile poiché il termine $x + y$ introduce un possibile *shift* al di fuori dei nodi della griglia, sul quale occorre calcolare la soluzione. Per il calcolo numerico di questo termine, abbiamo deciso di utilizzare l'approccio più diffuso in finanza, ovvero un approccio simile a quello usato con i metodi alle differenze finite. In particolare, posti $x_i \in \Omega_h$ i nodi della griglia, poniamo

$$J_i = J(x_i) = \int_{\mathbb{R}} u(t, x_i + y)\nu(y)dy,$$

$\forall i \in \{0, ..., N\}$. Qualora $u(t, x_i + y)$ cada fuori dalla *mesh*, proiettiamo le condizioni al bordo del problema. Possiamo così scrivere la formulazione debole del termine 3.5 in questo modo:

$$\sum_{j=0}^{N} J_j \int_{\Omega_h} \phi_j(x)\phi_i(x)dx,$$

$\forall i \in \{0, ..., N\}$. Da un punto di vista matriciale, occorre quindi moltiplicare il vettore J per la matrice di massa i cui coefficienti sono i termini a_{ij} definiti sopra.

La discretizzazione completa in forma matriciale della PIDE monodimensionale è quindi la seguente:

$$M_1 u^n = M_2 u^{n+1} + MJ,$$

dove u^{n+1} e u^n sono la soluzione al passo precedente e successivo (ricordiamo infatti che l'equazione è *backward*) e le matrici sono date da:

$$M_{1,ij} = \left(\frac{1}{\Delta t} + r\right) a_{ij} - \left(r - \frac{\sigma^2}{2}\right) b_{ij} + \frac{\sigma^2}{2} c_{ij},$$

$$M_{2,ij} = \frac{a_{ij}}{\Delta t}, \qquad M_{ij} = a_{ij}.$$

Osserviamo quindi che l'integrale viene calcolato esplicitamente. D'altro canto, un calcolo implicito richiederebbe la costruzione di una matrice di sistema piena, molto pesante da invertire e praticamente impossibile da memorizzare su un solo computer. Dalla teoria sappiamo che questo metodo, detto *operator splitting*, è stabile se:

$$\Delta t \leq {}^1/\lambda,$$

una condizione molto semplice da soddisfare.

L'altro approccio possibile è il seguente: prima si discretizza la funzione u, scrivendola come $u_h \in \mathbb{P}_1^h$, poi si integra. Abbiamo quindi:

$$\int_{\Omega_h} \phi_i(x) \left(\int_{\mathbb{R}} \sum_{j=1}^{N} u_j^k \phi_j(x+y) \nu(y) dy \right) dx =$$

$$\sum_{j=1}^{N} \int_{\Omega_h} \int_{\mathbb{R}} \phi_i(x) \phi_j(x+y) \nu(y) dx dy,$$

che potremmo anche riscrivere, tramite un cambio di variabile prima della moltiplicazione per ϕ_i come:

$$d_{ij} = \sum_{j=1}^{N} \int_{\Omega_h} \int_{\mathbb{R}} \phi_i(x) \phi_j(z) \nu(z-x) dz dx.$$

In tal caso, potremmo porre $\{D\}_{ij} = d_{ij}$ ottenendo:

$$M_1 u^n = M_2 u^{n+1} + D u^{n+1}.$$

Questo metodo permette di calcolare una sola volta la matrice piena D, e poi moltiplicarla ogni volta per la soluzione al passo precedente (cioè $n+1$). Nonostante questa idea possa portare a un calcolo più veloce della parte integrale $J = Du^{n+1}$, anche questo metodo costringerebbe a tenere in memoria una matrice piena, quindi non è praticabile.

Siamo giunti quindi alla discretizzazione completa della PIDE 1d con la trasformazione *log-price*. Nei prossimi paragrafi mostriamo la discretizzazione della PIDE 2d.

3.2.5 Discretizzazione della PIDE bidimensionale

Trattiamo ora la PIDE bidimensionale riportata in 2.4. Tramite il cambio di variabile $x_1 = log(^{S_1}/_{S_{1,0}})$, $x_2 = log(^{S_2}/_{S_{2,0}})$, $u = u(t, x_1, x_2)$, otteniamo:

$$\begin{aligned}\frac{\partial u}{\partial t} + \frac{\sigma_1^2}{2}\frac{\partial^2 u}{\partial x_1^2} + \frac{\sigma_2^2}{2}\frac{\partial^2 u}{\partial x_2^2} + \rho\sigma_1\sigma_2\frac{\partial^2 u}{\partial x_1 \partial x_2} + \left(r - \sigma_1^2\right)\frac{\partial u}{\partial x_1} + \left(r - \sigma_2^2\right)\frac{\partial u}{\partial x_2} - ru \\ + \int_{\mathbb{R}} \left(u(t, x_1 + y, x_2) - u(t, x_1, x_2) - (e^y - 1)\frac{\partial u}{\partial x_1}\right) k_1(y)dy \\ + \int_{\mathbb{R}} \left(u(t, x_1, x_2 + y) - u(t, x_1, x_2) - (e^y - 1)\frac{\partial u}{\partial x_2}\right) k_2(y)dy = 0. \quad (3.6)\end{aligned}$$

Ponendo come sopra:

$$\hat{\lambda_i} = \int_{\mathbb{R}} \nu_i(y)dy \qquad \hat{\alpha_i} = \int_{\mathbb{R}} (e^y - 1)\nu_i(y)dy,$$

abbiamo:

$$\begin{aligned}\frac{\partial u}{\partial t} + \frac{\sigma_1^2}{2}\frac{\partial^2 u}{\partial x_1^2} + \frac{\sigma_2^2}{2}\frac{\partial^2 u}{\partial x_2^2} + \rho\sigma_1\sigma_2\frac{\partial^2 u}{\partial x_1 \partial x_2} + \left(r - \frac{\sigma_1^2}{2} - \hat{\alpha_1}\right)\frac{\partial u}{\partial x_1} \\ + \left(r - \frac{\sigma_2^2}{2} - \hat{\alpha_2}\right)\frac{\partial u}{\partial x_2} - (r + \hat{\lambda_1} + \hat{\lambda_2})u \\ + \int_{\mathbb{R}} u(t, x_1 + y, x_2)\nu_1(y)dy + \int_{\mathbb{R}} u(t, x_1, x_2 + y)\nu_2(y)dy = 0. \quad (3.7)\end{aligned}$$

Di nuovo, applichiamo un troncamento al dominio calcolando $\{x^1_{min}, x^1_{max}\}$ e $\{x^2_{min}, x^2_{max}\}$ come in 3.3 e definiamo una triangolazione Ω_h sull'insieme sul rettangolo di vertici (x^1_{min}, x^2_{min}), (x^1_{max}, x^2_{max}), la griglia temporale $\mathcal{T} = \{0 = t_0 \leq t_1 \leq ... \leq t_N = T\}$ e uno spazio $\mathbb{P}^1_h$ in cui cerchiamo la soluzione u_h. Prima di passare alla formulazione debole, osserviamo che l'equazione può essere scritta nel seguente modo:

$$\begin{aligned}\frac{\partial u}{\partial t} + \begin{pmatrix} r - \frac{\sigma_1^2}{2} - \hat{\alpha_1} \\ r - \frac{\sigma_2^2}{2} - \hat{\alpha_2} \end{pmatrix}\nabla u + \frac{1}{2} div\left(\begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}\nabla u\right) - (r + \hat{\lambda_1} + \hat{\lambda_2})u \\ + \int_{\mathbb{R}} u(t, x_1 + y, x_2)\nu_1(y)dy + \int_{\mathbb{R}} u(t, x_1, x_2 + y)\nu_2(y)dy = 0. \quad (3.8)\end{aligned}$$

Osserviamo inoltre che, pur essendo un'equazione bidimensionale, la parte integrale rimane in una sola dimensione, quindi per valutare i due integrali occorre calcolare due vettori, J_1 e J_2:

$$J_1^i = J_1(x^i) = \int_{\mathbb{R}} u(t, x_1^i + y, x_2^i)\nu_1(y)dy,$$
$$J_2^i = J_2(x^i) = \int_{\mathbb{R}} u(t, x_1^i, x_2^i + y)\nu_2(y)dy,$$

nei diversi nodi x^i della griglia bidimensionale. Siamo quindi pronti a scrivere la formulazione debole del problema (3.7):

$$\begin{aligned}
&\sum_{j=0}^{N} \iint_{\Omega_h} \frac{\partial}{\partial t} \left(a_j(t)\phi_j(x,y)\phi_i(x,y)\right) d\Omega \\
&\qquad + \sum_{j=0}^{N} \begin{pmatrix} r - \frac{\sigma_1^2}{2} - \hat{\alpha_1} \\ r - \frac{\sigma_2^2}{2} - \hat{\alpha_2} \end{pmatrix} \iint_{\Omega_h} a_j(t) \nabla\phi_j(x,y)\phi_i(x,y) d\Omega \\
&\quad - \frac{1}{2} \sum_{j=0}^{N} \iint_{\Omega_h} a_j(t) \nabla\phi_j(x,y)^t \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \nabla\phi_i(x,y) d\Omega \\
&\qquad - (r + \hat{\lambda_1} + \hat{\lambda_2}) \sum_{j=0}^{N} \iint_{\Omega_h} a_j(t)\phi_j(x,y)\phi_i(x,y) d\Omega \\
&+ \sum_{j=0}^{N} J_1^j \iint_{\Omega_h} a_j(t)\phi_j(x,y)\phi_i(x,y) d\Omega + \sum_{j=0}^{N} J_2^j \iint_{\Omega_h} a_j(t)\phi_j(x,y)\phi_i(x,y) d\Omega = 0,
\end{aligned} \tag{3.9}$$

e la discretizzazione in tempo con uno schema di Eulero Implicito:

$$\begin{aligned}
&\sum_{j=0}^{N} \iint_{\Omega_h} \frac{a_j(t_{n+1})}{\Delta t} \phi_j(x,y)\phi_i(x,y) d\Omega \\
&+ \sum_{j=0}^{N} J_1^j \iint_{\Omega_h} a_j(t_{n+1})\phi_j(x,y)\phi_i(x,y) d\Omega + \sum_{j=0}^{N} J_2^j \iint_{\Omega_h} a_j(t_{n+1})\phi_j(x,y)\phi_i(x,y) d\Omega = \\
&\qquad - \sum_{j=0}^{N} \begin{pmatrix} r - \frac{\sigma_1^2}{2} - \hat{\alpha_1} \\ r - \frac{\sigma_2^2}{2} - \hat{\alpha_2} \end{pmatrix} \iint_{\Omega_h} a_j(t_n) \nabla\phi_j(x,y)\phi_i(x,y) d\Omega \\
&\quad + \frac{1}{2} \sum_{j=0}^{N} \iint_{\Omega_h} a_j(t_n) \nabla\phi_j(x,y)^t \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \nabla\phi_i(x,y) d\Omega \\
&\qquad + \left(\frac{1}{\Delta t} + r + \hat{\lambda_1} + \hat{\lambda_2} \right) \sum_{j=0}^{N} \iint_{\Omega_h} a_j(t_n)\phi_j(x,y)\phi_i(x,y) d\Omega.
\end{aligned} \tag{3.10}$$

A questo punto, possiamo scrivere l'equazione in forma matriciale:

$$M_1 u^n = M_2 u^{n+1} + M J_1^{n+1} + M J_2^{n+1},$$

dove M_1 e M_2 sono le matrici di sistema date dalla formulazione debole e M è la matrice di massa.

3.3 La trasformazione *Price*

Consideriamo ora il secondo cambio di variabile, un cambio solo sull'integrale. Come già anticipato, questo cambio di variabile permette di utilizzare la stessa griglia per l'equazione che per la quadratura dell'integrale, eliminando il problema di valutare la funzione fuori dal dominio. Inoltre risolve parzialmente il problema dello *shift* dei nodi.

3.3.1 Equazione in 1d

Per semplicità, illustriamo il cambio di variabile e la relativa discretizzazione nel caso di un'equazione su un solo sottostante, per poi estenderlo al caso bidimensionale. Riportiamo l'equazione in questione:

$$\frac{\partial C}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} - rC + \\ + \int_{\mathbb{R}} \left(C(t, Se^y) - C(t,S) - S(e^y - 1)\frac{\partial C}{\partial S}(t,S) \right) \nu(y)dy = 0.$$

Come nel caso precedente, siccome la misura di Lévy ν è finita, è possibile separare gli ultimi due addendi dentro l'integrale, e definendo:

$$\hat{\alpha} = \int_{\mathbb{R}} (e^y - 1)\nu(y)dy$$
$$\hat{\lambda} = \int_{\mathbb{R}} \nu(y)dy$$

l'equazione diventa:

$$\frac{\partial C}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 C}{\partial S^2} + (r - \hat{\alpha})S\frac{\partial C}{\partial S} - (r + \hat{\lambda})C + \int_{\mathbb{R}} C(t, Se^y)\nu(y)dy = 0.$$

Come nel caso logprice, $\hat{\lambda} = \lambda$, mentre $\hat{\alpha}$ viene calcolato tramite una qualche formula di quadratura, esattamente come nel caso *log-price*. A questo punto introduciamo nell'integrale il cambio di variabile:

$$z = Se^y$$

e l'equazione diventa:

$$\frac{\partial C}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 C}{\partial S^2} + (r - \hat{\alpha})S\frac{\partial C}{\partial S} - (r + \hat{\lambda})C \\ + \int_0^{\infty} \frac{C(t,z)}{z} \nu\left(\log\left(\frac{z}{S}\right)\right) dz = 0. \quad (3.11)$$

3.3.2 Troncamento del dominio

Notiamo che il dominio della funzione e dell'integrale coincidono, ossia sono entrambi $(0, \infty)$. Scegliamo il troncamento del dominio in modo analogo al caso *log-price*, ossia definiamo S_{min} e S_{max} come in (3.3). Allo stesso modo, definiamo il dominio per l'integrale come $[S_{min}, S_{max}]$. In questo modo è possibile utilizzare la stessa griglia per il calcolo dell'integrale.

3.3.3 Discretizzazione della parte PDE

Concentriamoci prima sulla parte differenziale. Come nel caso della logprice, consideriamo una triangolazione Ω_h con $N+1$ nodi di $[S_{min}, S_{max}]$. Cerchiamo

magari dire che le funzioni di base sono quelle di lagrange?

una soluzione del tipo:

$$C_h(S,t_n) = \sum_{j=0}^{N} \gamma_j(t)\phi_j(S),$$

dove γ_j sono i valori della soluzione all'istante t nel nodo j della griglia, mentre $\phi_j(x)$ sono le funzioni base dello spazio $\mathbb{P}_h^1 = \{ f \in \mathbb{C}^0 \ \textit{lineari su ogni cella della triangolazione } \Omega_h \ \}$.
Moltiplicando l'equazione per una funzione test ϕ_i, integrando sul dominio e inserendo C_h al posto di C otteniamo la formulazione debole del problema (3.1):

$$\sum_{j=1}^{N-1} \frac{\partial}{\partial t}\gamma_j(t) \int_{\Omega_h} \phi_j(S)\phi_i(S)dS + (r-\hat{\alpha}) \sum_{j=1}^{N-1} \gamma_j(t) \int_{\Omega_h} S\phi_j'(S)\phi_i(S)dS$$
$$+\frac{\sigma^2}{2} \sum_{j=1}^{N-1} \gamma_j(t) \int_{\Omega_h} S^2\phi_j''(S)\phi_i(S)dS - (r+\lambda) \sum_{j=1}^{N-1} \gamma_j(t) \int_{\Omega_h} \phi_j(S)\phi_i(S)dS = 0$$

e integrando per parti il termine con la derivata seconda, ricordando che i termini di bordo si annullano, otteniamo:

$$\sum_{j=1}^{N-1} \frac{\partial}{\partial t}\gamma_j(t) \int_{\Omega_h} \phi_j(S)\phi_i(S)dS + (r-\sigma^2-\hat{\alpha}) \sum_{j=1}^{N-1} \gamma_j(t) \int_{\Omega_h} S\phi_j'(S)\phi_i(S)dS$$
$$-\frac{\sigma^2}{2} \sum_{j=1}^{N-1} \gamma_j(t) \int_{\Omega_h} S^2\phi_j'(S)\phi_i'(S)dS - (r+\lambda) \sum_{j=1}^{N-1} \gamma_j(t) \int_{\Omega_h} \phi_j(S)\phi_i(S)dS = 0$$

Questa equazione deve valere per ogni i, ossia per ogni elemento della base. Introduciamo ora una griglia temporale $\mathcal{T} = \{0 = t_0 \le t_1 \le ... \le t_N = T\}$ dell'intervallo $[0,T]$. Se consideriamo uno schema temporale del tipo Eulero Implicito, con la discretizzazione nel tempo, l'equazione sopra diventa:

$$\frac{\sigma^2}{2} \sum_{j=1}^{N-1} \gamma_j(t_n) \int_{\Omega_h} S^2\phi_j'(S)\phi_i'(S)dS - (r-\sigma^2-\hat{\alpha}) \sum_{j=1}^{N-1} \gamma_j(t_n) \int_{\Omega_h} S\phi_j'(S)\phi_i(S)dS$$
$$+\left(r+\lambda+\frac{1}{dt}\right) \sum_{j=1}^{N-1} \gamma_j(t_n) \int_{\Omega_h} \phi_j(S)\phi_i(S)dS = \sum_{j=1}^{N-1} \frac{\gamma_j(t_{n+1})}{dt} \int_{\Omega_h} \phi_j(S)\phi_i(S)dS$$
$$\forall i \in \{1,\dots,N-1\}, \qquad \forall n \in \{0,\dots,T-1\} \quad (3.12)$$

Definendo il vettore γ^k come il vettore dei valori $\gamma_j(t_k)$ e le matrici seguenti

$$(M_1)_{ij} = \frac{\sigma^2}{2} \int_{\Omega_h} S^2\phi_j'(S)\phi_i'(S)dS - (r-\sigma^2-\hat{\alpha}) \int_{\Omega_h} S\phi_j'(S)\phi_i(S)dS$$
$$+\left(r+\lambda+\frac{1}{dt}\right) \int_{\Omega_h} \phi_j(S)\phi_i(S)dS,$$

$$(M)_{ij} = \int_{\Omega_h} \phi_j(S)\phi_i(S)dS \quad \text{e} \quad M_2 = \frac{1}{dt}M.$$

Il sistema d'equazioni si può dunque scrivere in forma matriciale come:

$$M_1\gamma^n = M_2\gamma^{n+1} \qquad \text{per } n = T-1,\dots,0 \tag{3.13}$$

3.3.4 La parte integrale

Al sistema (3.13) va aggiunta la parte integrale. Anche in questo caso viene trattata esplicitamente e aggiunta all'*rhs*. Dobbiamo dunque calcolare il seguente integrale:

$$J(S) = \int_0^\infty \frac{C(t,z)}{z}\nu\left(\log\left(\frac{z}{S}\right)\right)dz$$

che poi andrà scritto come elemento dello spazio $\mathbb{P}_h^1$. Occorre quindi calcolare l'integrale nei punti S_i della griglia, ottenendo un vettore J dove $J_i = J(S_i)$. La (3.13) diventa dunque:

$$M_1\gamma^n = M_2\gamma^{n+1} + MJ^{n+1} \qquad \text{per } n = T-1,\dots,0 \tag{3.14}$$

Il problema si riduce dunque a calcolare la seguente quantità:

$$J^n(S_i) = \int_0^\infty \frac{C(t_n,z)}{z}\nu\left(\log\left(\frac{z}{S_i}\right)\right)dz$$

per ogni nodo della griglia.
Notiamo che rispetto al metodo *log-price* non c'è più il problema dello *shift* sui nodi, ma il termine è comunque non locale perché per ogni vertice si integra su tutto il dominio.
Per il calcolo di tale termine, abbiamo deciso di adottare la seguente procedura: per ogni cella della griglia, calcoliamo il valore della funzione:

$$\frac{C(t_k,z_l)}{z}$$

su opportuni nodi di quadratura z_l. Poi, per ogni nodo S_i della griglia calcoliamo il contributo di tale cella al valore di J_i^n valutandola contro la densità calcolata in $\log(z_l/S_i)$. Riassumendo, il termine i-esimo di J^k viene calcolato come:

$$J_i^n = \sum_{Q_k\in\Omega_h}\left(\sum_{z_l\in Q_k}\frac{C(t_n,z_l)}{z_l}\nu\left(\log\left(\frac{z_l}{S_i}\right)\right)w_l\right)$$

dove Q_k sono le celle della griglia, e w_l sono i pesi di quadratura in quella cella. Notiamo che siccome la densità è calcolata in $\log(z_l/S_i)$ non occorre più fare ricorso a formule di quadratura speciali per i modelli di Kou e Merton, come nel caso *log-price*. Per questo motivo, per calcolare questi integrali abbiamo sfruttato dei classici nodi di Gauss offerti da deal.ii.

3.3.5 Discretizzazione della PIDE bidimensionale

In modo analogo è possibile trattare il caso bidimensionale. Separando le parti dell'integrale che non dipendono dal valore della funzione, possiamo riscrivere la (2.4) come:

$$\begin{aligned}\frac{\partial C}{\partial t} + (r-\hat{\alpha}_1)S_1\frac{\partial C}{\partial S_1} + (r-\hat{\alpha}_2)S_2\frac{\partial C}{\partial S_2} + \frac{\sigma_1^2}{2}S_1^2\frac{\partial^2 C}{\partial S_1^2} + \frac{\sigma_2^2}{2}S_2^2\frac{\partial^2 C}{\partial S_2^2} \\ + \rho\sigma_1\sigma_2 S_1 S_2\frac{\partial^2 C}{\partial S_1 \partial S_2} - (r+\lambda_1+\lambda_2)C \\ + \int_{\mathbb{R}} C(t, S_1 e^y, S_2)\nu_1(y)dy + \int_{\mathbb{R}} C(t, S_1, S_2 e^y)\nu_2(y)dy = 0.\end{aligned}$$

Come nel paragrafo precedente, trattiamo in modo separato parte differenziale e parte integrale. Concentriamoci quindi sulla sola PDE e notiamo che essa può essere scritta in forma vettoriale, cioè:

$$\frac{\partial C}{\partial t} + t \cdot \nabla C + (D\nabla)\cdot\nabla C - (r+\lambda_1+\lambda_2)C = 0 \tag{3.15}$$

dove il vettore t e la matrice D sono così definite:

$$t = \begin{bmatrix}(r-\hat{\alpha}^1)S_1 \\ (r-\hat{\alpha}^2)S_2\end{bmatrix}, \qquad D = \frac{1}{2}\begin{bmatrix}\sigma_1^2 S_1^2 & \rho\sigma_1\sigma_2 S_1 S_2 \\ \rho\sigma_1\sigma_2 S_1 S_2 & \sigma_2^2 S_2^2\end{bmatrix}.$$

Analogamente al caso *log-price*, il dominio va troncato seguendo lo stesso principio.
Il passaggio alla formulazione debole della parte differenziale è stata trattata in [3], ed è analoga a quella monodimensionale. In definitiva, approssimando la funzione incognita C come un elemento dello spazio $\mathbb{P}_h^1$:

$$C_h(t, S_1, S_2) = \sum_{j=0}^{N}\gamma_j(t)\phi_j(S_1, S_2),$$

si ottiene la seguente formulazione debole per la parte differenziale (omettiamo la dipendenza di ϕ_k da (S_1, S_2)):

$$\begin{aligned}\sum_{j=0}^{N}\Bigg[\frac{\partial C}{\partial t}\gamma_j(t)\iint_{\Omega_h}\phi_i\phi_j d\Omega + \gamma_j(t)\iint_{\Omega_h}\phi_i(t - DivD)\nabla\phi_j d\Omega \\ -\gamma_j(t)\iint_{\Omega_h}(\nabla\phi_i)^t D(\nabla\phi_j)d\Omega - (r-\lambda_1-\lambda_2)\gamma_j(t)\iint_{\Omega_h}\phi_i\phi_j d\Omega\Bigg] = 0\end{aligned}$$

che deve valere per ogni elemento della base ϕ_i.
Utilizzando ancora una discretizzazione temporale del tipo Eulero Implicito otteniamo:

$$\sum_{j=0}^{N}\left[\frac{\gamma_j(t_n)}{\Delta t}\iint_{\Omega_h}\phi_i\phi_j d\Omega-\gamma_j(t_n)\iint_{\Omega_h}\phi_i(t-DivD)\nabla\phi_j d\Omega\right.$$
$$\left.+\gamma_j(t_n)\iint_{\Omega_h}(\nabla\phi_i)^t D(\nabla\phi_j)d\Omega+(r-\lambda_1-\lambda_2)\gamma_j(t_n)\iint_{\Omega_h}\phi_i\phi_j d\Omega\right]=$$
$$=\sum_{j=0}^{N}\frac{\gamma_j(t_{n+1})}{\Delta t}\iint_{\Omega_h}\phi_i\phi_j d\Omega$$

Introducendo ancora una volta il vettore γ^k delle componenti di C_h al tempo k e le matrici M_1, M_2 e M:

$$(M_1)_{ij}=\left(\frac{1}{\Delta t}+r-\lambda_1-\lambda_2\right)\int_{\Omega_h}\phi_i\phi_j d\Omega+\int\int_{\Omega_h}(\nabla\phi_i)^t D(\nabla\phi_j)d\Omega$$
$$-\iint_{\Omega_h}\phi_i(t-DivD)\nabla\phi_j d\Omega,$$
$$(M)_{ij}=\iint_{\Omega_h}\phi_i\phi_j d\Omega,\qquad M_2=\frac{1}{\Delta t}M,$$

possiamo scrivere il sistema in forma matriciale:

$$M_1\boldsymbol{\gamma}^n=M_2\boldsymbol{\gamma}^{n+1}\qquad\text{per } n=T-1,\dots,0 \tag{3.16}$$

Concentriamoci ora sulla parte integrale. Al sistema (3.16) vanno infatti aggiunte nell'*rhs* le parti integrali J_1 e J_2. Analogamente al caso monodimensionale, otteniamo:

$$M_1\gamma^n=M_2\gamma^{n+1}+MJ_1^{n+1}+MJ_2^{n+1}\qquad\text{per } n=T-1,\dots,0. \tag{3.17}$$

Per quanto riguarda il calcolo di J_1 e J_2, il caso bidimensionle è più delicato. Riportiamo le espressioni di J^1 e J^2 una volta eseguita la trasformazione *price*:

$$J^1(t,S_1,S_2)=\int_{\mathbb{R}}\frac{C(t,z,S_2)}{z}\nu_1\left(log\left(\frac{z}{S_1}\right)\right)dz,$$
$$J^1(t,S_1,S_2)=\int_{\mathbb{R}}\frac{C(t,S_1,z)}{z}\nu_1\left(log\left(\frac{z}{S_2}\right)\right)dz.$$

Per ogni punto $(S_1,S_2)_j$ della griglia, devono essere calcolati $J_1(t,S_{1,j},S_{2,j})$ e $J_2(t,S_{1,j},S_{2,j})$ lungo i segmenti contenuti nel dominio con direzione x e y che passano da $(S_1,S_2)_j$. Se la griglia è strutturata, questo corrisponde a integrare sulle facce delle celle. Una faccia parallela all'asse x contribuisce al calcolo dell'i-esimo termine di J_1 se la coordinata y della faccia è la stessa della coordinata y del punto $(S_1,S_2)_i$.
In sostanza, se definiamo per ogni nodo $(S_1,S_2)_j$ l'insieme $\mathcal{F}_j^1$ delle facce parallele all'asse x aventi come coordinata $y=S_{2,j}$ e in modo analogo l'insieme $\mathcal{F}_j^2$ per le facce parallele all'asse y, abbiamo le seguenti formule per il calcolo dei vettori J_1, J_2 al tempo k:

$$(J_1^k)_j = \sum_{\mathbb{F}\in\mathcal{F}_j^1}\sum_{z_l\in\mathbb{F}} \frac{C(t_k, z_l, S_2)}{z_l}\nu_1\left(\log\left(\frac{z_l}{S_{1,j}}\right)\right)w_l,$$
$$(J_2^k)_j = \sum_{\mathbb{F}\in\mathcal{F}_j^2}\sum_{z_l\in\mathbb{F}} \frac{C(t_k, S_1, z_l)}{z_l}\nu_2\left(\log\left(\frac{z_l}{S_{2,j}}\right)\right)w_l,$$

dove z_l, w_l sono nodi e pesi di quadratura. Queste formule vengono poi implementate nel modo descritto in 5.4.2.

3.4 Condizioni al contorno

Fin'ora non abbiamo menzionato le condizioni al contorno. Trattandosi sempre di condizioni di tipo *dirichlet*, esse vengono imposte una volta assemblato il sistema tramite tecnica di penalizzazione. Esse differiscono leggermente per price e logprice per via della trasformazione.

CB per logprice

3.5 Il problema con l'ostacolo: il SOR proiettato

Il metodo più utilizzato in finanza per risolvere problemi del tipo 1.6 è il cosiddetto SOR proiettato, un metodo di risoluzione del sistema lineare iterativo che permette ad ogni iterazione e per ogni punto della griglia di imporre che la soluzione stia sopra all'ostacolo. L'algoritmo, molto semplice da implementare, è una variazione del noto metodo SOR, *successive over-relaxation*, una variazione del metodo di *Gauss-Seidel*. Posto quindi:

$$z = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right),$$

dove b è il termine noto, a è la matrice e $x^{(k)}$ e $x^{(k+1)}$ sono le soluzione al passo precedente e successivo, al posto della classica iterata $(k+1)$-esima data da:

$$x_i^{(k+1)} = x_i^{(k)} + \omega(z - x_i^{(k)}),$$

con $0 < \omega < 2$ fissato, nel SOR proiettato imponiamo:

$$x_i^{(k+1)} = \max\left(K - S_i, \quad x_i^{(k)} + \omega(z - x_i^{(k)})\right),$$

dove S_i è il nodo i-esimo della griglia. In questo modo, qualora la soluzione scenda sotto al *payoff*, imponiamo che essa valga esattamente il *payoff* rispettando il vincolo 1.5.

3.6 *Mesh refinement*

In questo programma abbiamo implementato anche una funzione che permette di adattare la griglia al problema, in particolare di raffinare o de-raffinare la *mesh* dove ce ne sia bisogno. Per capire in quali punti adattare la griglia abbiamo

utilizzato lo stimatore di Kelly, implementato direttamente nella libreria deal.ii. Esso stima l'errore soltanto per l'equazione generalizzata di Poisson:

$$-\nabla \left(a(\underline{x}) \nabla u \right) = f,$$

tuttavia è uno stimatore utilizzato spesso perché permette di stimare la differenza fra i gradienti della soluzione in due celle vicine. Qualora questa differenza sia troppa grande, ovvero la soluzione cambi molto inclinazione fra celle vicine, allora è necessario raffinare la griglia. Se invece la differenza fra i gradienti è nulla o molto piccola, allora è possibile de-raffinare la *mesh*.

Capitolo 4

Pacchetti usati

4.1 deal.ii

deal.ii è una libreria scritta in c++ che permette di risolvere tramite metodi a elementi finiti per una grande varietà di problemi alle derivate parziali. Fra i suoi maggiori pregi, oltre alla presenza di elementi finiti di ogni ordine, troviamo la grande scalabilità (è stato infatti provato che questa libreria riesce a scalare fino a 16.000 processori), la possibilità di interfacciarsi con molte librerie come PETSc, Trilinos, METIS, BLAS, LAPACK, e la vastità e la precisione della documentazione. Soffermandoci un attimo su questo ultimo punto, vorremmo proprio sottolineare la presenza di una documentazione (quasi) sempre precisa e ben organizzata. Oltre a questa, ci sono inoltre 51 tutorial che insegnano a utilizzare la libreria dalla semplice creazione di una *mesh* bidimensionale fino alla risoluzione di problemi iperbolici di meccanica non lineare in 3d con *mesh refinement*. Per scrivere il nostro codice abbiamo sfruttato molto la massiccia documentazione della libreria, e siamo stati quasi sempre in grado di trovare le funzioni che facevano al caso nostro. In una occasione tuttavia non siamo riusciti a trovare un metodo e abbiamo deciso di scrivere sul gruppo `https://groups.google.com/forum/#!forum/dealii`, ottenendo una risposta precisa in breve tempo.

Altro?

4.2 Cmake

La libreria deal.ii utilizza Cmake per compilare i codici dei tutorial. Per ciò anche noi abbiamo utilizzato questo *tool*. In particolare, in un qualsiasi file CMakeLists.txt occorre aggiungere tutti i file sorgente, scrivere il nome dell'eseguibile desiderato e il CMake crea da solo il Makefile specificando dove si trovano gli *header file* da includere e le librerie di deal.ii da linkare. Utilizzando due sistemi operativi diversi, il CMake è stato molto utile per poter lavorare sui codici senza avere problemi di portabilità che può avere il semplice Makefile. Inoltre, siccome abbiamo utilizzato nel nostro codice OpenMP e il CMakeLists.txt di deal.ii non lo prevede, abbiamo aggiunto:

```
find_package(OpenMP)
```

questo da accorciare secondo me, basta dire che abbiamo aggiunto openMP e Doxygen

```
if (OPENMP_FOUND)
    set (CMAKE_C_FLAGS "${CMAKE_C_FLAGS}
        ${OpenMP_C_FLAGS}")
    set (CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}
        ${OpenMP_CXX_FLAGS}")
endif()
```

in modo che, se il CMake trova OpenMP, aggiunge il *flag* -fopenmp al compilatore, altrimenti non aggiunge nulla.
Abbiamo infine aggiunto:

```
FIND_PACKAGE(Doxygen)
IF(NOT DOXYGEN_FOUND)
        MESSAGE(FATAL ERROR "Doxygen not found")
ENDIF()
CONFIGURE_FILE(doxy-config \@ ONLY IMMEDIATE)
ADD_CUSTOM_TARGET(doc
        COMMAND ${DOXYGEN_EXECUTABLE} doxy-config )
```

in modo da creare il *target* doc che generi la documentazione con Doxygen.

4.3 GitHub

Durante lo sviluppo di questo progetto, abbiamo sempre utilizzato il sistema di controllo di versioni Git e il servizio di *web hosting* GitHub per lo sviluppo di progetti software. Abbiamo quindi creato una *repository* remota all'indirizzo `https://github.com/NTFrs/pacs_proj`. L'uso di git ci ha permesso di lavorare in contemporane sul progetto senza dover preoccuparci di fare cambiamenti allo stesso file, anche attraverso l'uso di diversi branch.

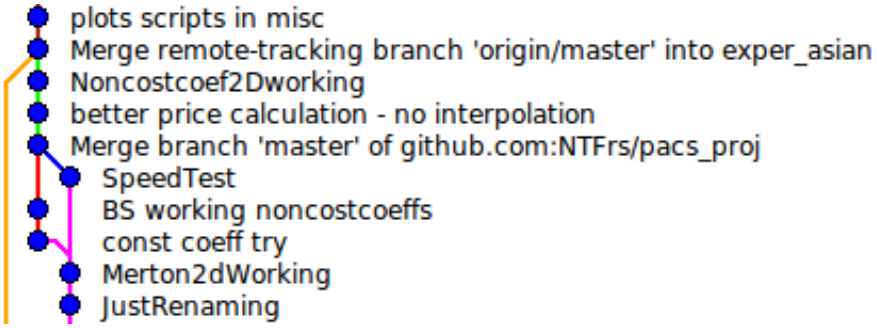


Figura 4.1: Cammino di alcuni *branch* sui quali abbiamo lavorato. Immagine tratta da gitk

4.4 Doxygen

Infine, tutto il codice è commentato con il *lexical-scanner* Doxygen, di molto facile utilizzo.

4.5 Profiler

gprof Durante la scrittura dei codici finali, abbiamo trovato che alcune parti del programma era eccessivamente più lente rispetto a quando era stata fatta su file semplici. L'utilizzo del profiler *gprof* ci ha permesso di analizzare e ristrutturare quella parte ottenendo uno speedup considerevole. In particolare, si trattava del calcolo dell'integrale con la trasformazione Price, e abbiamo scoperto che una delle funzioni di dealii di accesso a elementi della griglia era particolarmente lenta. Abbiamo risolto diminuendo quanto possibile il numero di chiamate ad essa.

Capitolo 5

Codice

5.1 Introduzione

In questo capitolo descriviamo come il problema è stato implementato da un punto di vista computazionale, elencando le varie classi scritte e le loro caratteristiche. Inizialmente mostriamo delle piccole classi che si occupano di gestire i parametri dei modelli e le loro densità. Nella seconda sezione descriviamo gli oggetti opzione che costruiscono il sistema per risolvere il problema a elementi finiti in una e due dimensione. Questi oggetti utilizzano poi le funzioni di altri oggetti, LevyIntegral, per il calcolo della parte integrale. Successivamente, descriviamo brevemente la *Factory* che abbiamo scritto per instanziare gli oggetti opzione e infine spieghiamo brevemente le linee guida per utilizzare questa libreria.

5.2 Classi per i Modelli

Il primo blocco di classi scritto nel nostro codice permette di gestire i vari modelli utilizzati per descrivere la dinamica del sottostante, in particolare i modelli di *Blach&Scholes*, *Kou* e *Merton*. Abbiamo quindi deciso di scrivere una classe astratta Model, che contenga i parametri comuni ai diversi oggetti e tutti i metodi utilizzati. Come possiamo osservare in figura 5.2, da questa classe ereditano in maniera pubblica le tre classi modello non più astratte, ed esse contengono i parametri propri di ogni modello.

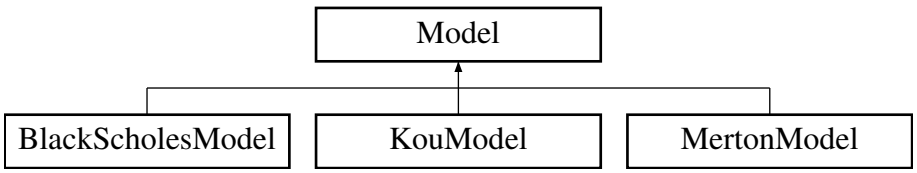


Figura 5.1: Gerarchia delle classi modello

5.3 Classi per Opzioni

Le classi per Opzioni sono la componente centrale del programma e permettono di istanziare e gestire il problema di differenziale. Sono state scritte secondo le linee guida della libreria deal.ii. Le loro caratteristiche principali sono dunque le seguenti:

- come tutte classi presentate nei *tutorial* di deal.ii, anche le nostre classi hanno la dimensione (nel nostro caso 1 o 2) come parametro template,
- i più importanti metodi privati (o protetti) e pubblici dell'oggetto Opzione sono i seguenti:
  1. virtual void setup_system();
  2. virtual void make_grid();
  3. virtual void assemble_system();
  4. virtual void refine_grid();
  5. virtual void setup_integral(), aggiunto da noi per allocare le classi che si occupano della parte integrale;
  6. virtual void solve(), metodo che risolve il sistema lineare;
  7. virtual void run(), metodo pubblico che chiama in ordine tutti i precedenti e calcola la soluzione.

A questi metodi sono stati aggiunti altre funzioni che permettono di impostare alcune variabili del problema.
Per quanto riguarda la struttura gerarchica di queste classi, poiché il nostro scopo è calcolare il prezzo di opzioni europee e americane con trasformazioni *price* e *logprice* e poiché alcuni metodi sono comuni ai diversi problemi e alle diverse trasformazioni, abbiamo deciso di sfruttare l'ereditarietà delle classi permessa dal c++ facendo chiamare al programma per ogni problema i metodi opportuni.

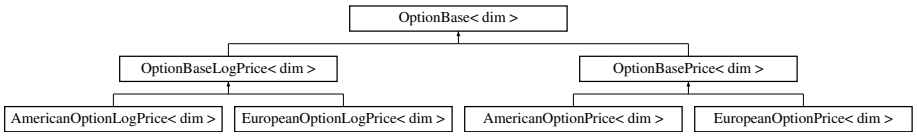


Figura 5.2: Gerarchia delle classi opzione

5.3.1 OptionBase<dim>

Entrando più nel dettaglio, come possiamo osservare nella figura 5.3, abbiamo scritto una classe base OptionBase<dim> astratta, nel cui campo protected sono contenute tutte le variabili del problema, quali gli oggetti dealii::FE_Q<dim>, dealii::DoFHandler<dim> e dealii::Triangulation<dim> che gestiscono elementi finiti, gradi di libertà e *mesh*, gli oggetti che memorizzano le matrici e i

dealii::Vector<double> della soluzione e del *right hand side*. Per quanto riguarda l'oggetto che gestisce la matrice di sistema, poiché uno dei nostri problemi necessita di un *solver* particolare non presente nella libreria utilizzata, ovvero il SOR, abbiamo deciso di scrivere un piccolo decoratore. Questo oggetto, dealii::SparseMatrix_PSOR<double, dim> eredita pubblicamente da dealii::SparseMatrix<double> e aggiunge due metodi: uno che risolve il problema con l'ostacolo nel caso *price* e uno nel caso *logprice*. Come sappiamo, i costruttori non vengono ereditati, perciò sono stati riscritti in modo che chiamino i costruttori della classe base. Inoltre, siccome i metodi di dealii::SparseMatrix<double> non sono virtual, abbiamo aggiunto all'interno della nostra classe:

```
// SparseMatrix methods
using dealii::SparseMatrix<number>::reinit;
using dealii::SparseMatrix<number>::add;
```

in modo che il compilatore capisca che deve utilizzare i metodi reinit e add di dealii::SparseMatrix<double>.

Per quanto riguarda i costruttori di questa classe, essi sono specializzati per 1 e 2 dimensioni. In particolare, il costruttore 1d prende un puntatore alla classe base Model e i vari parametri dell'opzione (quali tasso di interesse, scadenza e *strike*), mentre il costruttore 2d prende due puntatori a Model e i vari dati dell'opzione. I costruttori si occupano di creare gli oggetti relativi agli elementi finiti e li collegano alla *mesh*. I metodi invece che implementiamo qui sono void setup_system() che si occupa di creare lo *sparsity pattern* delle matrici e inizializzarle e void refine_grid() che esegue un adattamento della griglia.

5.3.2 OptionBasePrice<dim> e OptionBaseLogPrice<dim>

Le classi OptionBasePrice<dim> e OptionBaseLogPrice<dim>, anch'esse astratte, ereditano da OptionBase<dim> e definisco parte dei metodi descritti nell'introduzione. Le due classi implementano la funzione void make_grid() che crea la *mesh* nei casi *price* e *logprice*, ovvero con le opportune trasformazioni, il metodo void assemble_system() che integra gli elementi finiti e costruisce le matrici di sistema in 1d e 2d e il metodo double get_price(), che valuta la soluzione nel punto $\underline{x} = \underline{0}$ per il *logprice* e $\underline{S} = (S_0^1, S_0^2)$ per il *price*, ovvero che restituisce il prezzo dell'opzione. Infine, la classe OptionBasePrice<dim> implementa qui il metodo void setup_integral() che istanzia dinamicamente un oggetto di tipo LevyIntegral, che si occupa di calcolare la parte integrale. OptionBaseLogPrice<dim> invece non lo istanzia qui ma nel "livello" di ereditarierà successivo poiché in questa trasformazione occorre conoscere le condizioni al bordo per poter calcolare l'integrale.

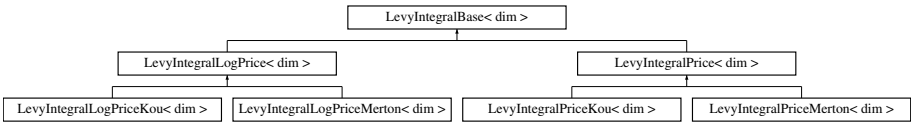
5.3.3 AmericanOption<dim> e EuropeanOption<dim>

Queste quattro classi alla base della piramide sono gli oggetti utilizzati per risolvere i vari problemi. Essi implementano le varie funzioni void solve() che risolvono il sistema lineare. In primo luogo quindi viene proiettata sulla *mesh* la condizione finale (ovviamente diversa fra Put e Call, *price* e *logprice*) e successivamente parte il ciclo temporale che applica le condizioni al bordo, calcola il vettore integral J se il modello non è *Black&Scholes* e risolve il sistema. Per

quanto riguarda quest'ultimo passaggio, per l'europea abbiamo utilizzato un *solver* utilizzato da deal.ii, ovvero SparseDirectUMFPACK, mentre per l'americana usiamo i *solver* scritti all'interno del decoratore di dealii::SparseMatrix<double>.

5.4 Classi per il calcolo degli integrali

Per il calcolo della parte integrale dell'equazione, ossia la quadratura di $\hat{\alpha}$ e il calcolo dei vettori $\mathbf{J}^i$ ad ogni iterazione temporale, sono state costruite una serie di classi. Tali classi ereditano da una classe comune di base che definisce l'interfaccia. Usando questo schema con ereditarietà (che si può vedere in figura 5.4) si riesce a mantenere un interfaccia comune, e permette di allocare classi specifiche per modelli specifici. Cosi come le classi opzione, anche queste sono classi templatizzate sulla dimensione, poiché il calcolo dell'integrale può essere anche molto diverso fra una dimensione e l'altra.



5.4.1 La classe base LevyIntegralBase

Questa classe astratta serve per definire un interfaccia che tutte le classi per la quadratura dell'integrale dovrebbero sfruttare. Inoltre a implementa alcuni metodi di base che possono essere utili alle classi figlie.
I due metodi *core* della classe sono compute_alpha(), che calcola il valore degli $\hat{\alpha}^i$, e compute_J(), che calcola il valore della parte integrale da usare nel *rhs*. È interessante notare che questa classe base definisce già una funzione che calcola $\hat{\alpha}^i$ generica, ossia con una quadratura composita con nodi di Gauss, e che funziona dunque con qualsiasi modello.

5.4.2 Le classi LevyIntegralPrice e LevyIntegralLogPrice

La seconda parte dell'integrale, i vettori $\mathbf{J}^i$ è totalmente diversa a seconda che si utilizzi la forma Price o la forma LogPrice. Per questo vi sono queste due classi che ereditano da LevyIntegralBase. Siccome sono ancora generiche e non per un modello specifico, ereditano da quest'ultima il metodo compute_alpha(). Inoltre queste funzionano con un modello qualunque.

LevyIntegralLogPrice Questa classe implementa compute_J() per la forma log-price, ed è sostanzialmente più semplice, ma anche più lenta rispetto a price.
In sostanza, sia in una che in due dimensioni (e potenzialmente in più), il calcolo di $\mathbf{J}^i$ viene svolto facendo un ciclo su tutti i nodi della griglia e in ciascuno calcolando il valore dell'integrale in quel punto e per fare ciò fa ricorso alla classe ausiliaria Solution_Trimmer. Tale classe è praticamente un funtore che, inizializzato con la soluzione e la mappatura dei gradi di libertà di Deal.II (il DoFHandler) agisce diversamente a seconda che il punto stia dentro o fuori dal dominio. Nel primo caso chiama una funzione di Deal.II che individua in che

cella si trova il nodo e restituisce il valore della funzione in tale punto, nel secondo caso impone il valore al bordo. La parte computazionalmente costosa è appunto la valutazione in un punto interno al dominio, poiché deve individuare in quale cella si trova (ricordiamo qui che la funzione incognita va valutata nel punto $y_l + x_i$, dove y_l sono i nodi di quadratura e x_i il nodo attuale, quindi ad ogni nodo in punti diversi).
Una volta ottenuto il valore della soluzione nel punto (sia esso dentro o fuori dal dominio) è possibile effettuare la quadratura dell'integrale. A questo livello, utilizza una quadratura generica poiché non vi è un modello specifico.

LevyIntegralPrice Questa classe implementa compute_J() specializzando il metodo sul parametro template, ossia la dimensione. Infatti, come già spiegato nella sezione 3.3, il modo in cui calcolare $\mathbf{J}^i$ è fondamentalmente diverso.
In una dimensione, per ogni nodo della griglia, bisogna integrare sulla stessa griglia, e quindi si scorrono tutte le celle calcolando il contributo. In due dimensioni, il contributo a $\mathbf{J}^1$ nel nodo S_i è calcolato sulla retta monodimensionale parallela all'asse delle ascisse, mentre il contributo a $\mathbf{J}^2$ è calcolato lungo la retta parallela all'asse delle ordinate. Quindi si scorrono le facce delle celle della griglia, e su ciascuna faccia si calcola il contributo a ogni nodo appartenente alla retta su cui giace la faccia in questione. Sebbene è una procedura complicata e richiede che tutti i nodi vengano visitati ma solo alcuni usati per la faccia corrente, è più rapida rispetto al calcolo con il metodo logprice. Infatti, in questo caso la funzione va valutata nei nodi di quadratura che cadono sulla faccia attuale, quindi dalla posizione nota.

richiede una griglia strutturata?

5.4.3 Le classi figlie per modelli specifici

Come si nota in figura 5.4, esistono poi delle classi derivate da LevyIntegralPrice e LevyIntegralLogPrice. Esse implementano di nuovo i metodi per il calcolo delle parti integrali con dei nodi di quadratura specifici ai modelli. In particolare LevyIntegralPriceKou e LevyIntegralLogPriceKou utilizzano nodi di Gauss-Laguerre, mentr LevyIntegralPriceMerton e LevyIntegralLogPriceMerton utilizzano nodi di Gauss-Hermite. Le due classi ereditate da LevyIntegralPrice reimplementano solo la parte relativa ad $\hat{\alpha}^i$, poiché la parte $\mathbf{J}$ rimane uguale.

Nel caso si vogliano aggiungere alla libreria alri modelli, questo design permette di creare nuove classi specifiche per l'integrale che ereditano dal secondo livello, dovendo modificare solo una piccola parte.

5.5 *Factory*

Data la complessità strutturale delle classi Opzioni, abbiamo deciso di scrivere utilizzare il *design pattern* della *factory*, altrimenti noto come costruttore virtuale, per permettere a un qualsiasi utente di istanziare l'oggetto giusto per il suo problema. Abbiamo quindi creato una classe, OptionFactory che ha costruttore di default, costruttore di copia e operatore di copia privati, perciò non è possibile istanziare la classe, e abbiamo scritto il seguente metodo:

```
static OptionFactory * instance()
        {
```

```
            static OptionFactory instance;
            return &instance;
    }
```

Questa funzione permette di istanziare uno e un solo oggetto della classe OptionFactory, che è quindi un *singleton*. Oltre a questo metodo, abbiamo due funzioni std::unique_ptr<OptionBase<dim> > create(...) con il parametro dim specializzato su una e due dimensioni che prendono come argomenti il tipo di opzione, europea o americana, put o call, il tipo di trasformazione, i modelli dei sottostanti e i vari parametri dell'opzione.

5.6 Come utilizzare la libreria

Capitolo 6

Risultati

Capitolo 7

Estensioni

Il programma che abbiamo scritto si presta molto a possibili estensioni, su molti livelli. In particolare, a partire dalle classi base OptionBasePrice<dim> e OptionBaseLogPrice<dim> è possibile scrivere dei *solver* che prezzino tutte le opzioni barriera, ovvero opzioni caratterizzate da condizioni al bordi di *Dirichlet* nulle in alcune parti o su tutto il bordo dominio.
È inoltre possibile estendere la libreria in modo che prezzi opzioni con altri modelli, per esempio ad attività infinita, caratterizzati cioè da misure di Lévy non limitate (come, ad esempio, il *Normal Inverse Gaussian* e il *Variance Gamma*). Per fare ciò occorre modificare le classi integrale, in modo che calcolino il parametro λ, che per questi modelli è incognito e calcolare all'interno della classe opzione la diffusione che approssima i piccoli salti (che in questo caso, sono infiniti).
Un'altra estensione possibile, ma abbastanza delicata, è estendere il programma al calcolo di opzioni con tre sottostanti, ovvero risolvere un PIDE 3d. Teoricamente l'equazione sarebbe formalmente la stessa, con l'aggiunta di un terzo integrale su $\mathbb{R}$. Occorrerebbe tuttavia prestare attenzione a dove integrare i vari integrali: in 2d, infatti, abbiamo calcolato gli integrali sui lati dei quadrilateri che costituiscono la *mesh*, in 3d si dovrebbe invece integrare sui lati delle facce dei parallelepipedi????
Si potrebbe provare poi a utilizzare le funzioni offerte dalla libreria deal.ii per la soluzione del problema con memoria distribuita. Per quanto riguarda la parte PDE si può sfruttare quanto implementato nella libreria, mentre per il calcolo dell'integrale in *price* occorrerebbe spedire tutta la soluzione a tutti i processi (perché il termine è non locale), in *log-price* invece tutti i processi devono conoscere tutta la griglia, per poter valutare la funzione nei punti $y + x$ (3.5).

Heston?

Bibliografia

[1] Quarteroni Alfio, Sacco Riccardo, and Saleri Fausto. *Numerical Mathematics*. Springer, 2007.

[2] Tomas Bjork. *Arbitrage Theory in Continuous Time*. Oxford Finance, 2009.

[3] Jiang Tao, Liu Xin, and Yu Zhengzhou. Finite element algorithms for pricing 2-d basket options. In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pages 4881–4886. IEEE, 2009.

[4] Jinghui Zhou. *Multi-Asset Option Pricing with Levy Process*. PhD thesis, South China University of Technology, 2009.