Riassunto Progetto Pacs

Nahuel Foresta, Giorgio Re

10 giugno 2014

1 Riassunto Obiettivi

Come già indicato negli altri report, l'obiettivo del progetto è creare un programma per prezzare una serie di opzioni utilizzando dei metodi basati su elementi finiti. Il valore di un'opzione al variare del sottostante (che supponiamo evolvere secondo un modello Jump-Diffusion) può essere in generale trovato come soluzione di un'equazione integro differenziale del tipo:

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

su $[0, T] \times [0, +\infty]$ con opportune condizioni al bordo e condizione finale $C(T, S) = g(S)$, payoff dell'opzione. k è un nucleo con una forte massa nell'intorno dello zero e code esponenziali o gaussiane. In due dimensioni, supponendo l'indipendenza delle componenti di salto dei due sottostanti, tale equazione diventa:

$$\frac{\delta C}{\delta t} + \frac{\sigma_1^2}{2} S_1^2 \frac{\delta^2 C}{\delta S_1^2} + \frac{\sigma_2^2}{2} S_2^2 \frac{\delta^2 C}{\delta S_2^2} + \rho \sigma_1 \sigma_2 S_1 S_2 \frac{\delta^2 C}{\delta S_1 \delta S_2} + r \frac{\delta C}{\delta S_1} + r \frac{\delta C}{\delta S_2} - rC + \\ + \int_{\mathbb{R}} \left(C(t, S_1 e^y, S_2) - C(t, S_1, S_2) - S_1(e^y - 1) \frac{\delta C}{\delta S}(t, S_1, S_2) \right) k_1(y) dy \\ + \int_{\mathbb{R}} \left(C(t, S_1, S_2 e^y) - C(t, S_1, S_2) - S_2(e^y - 1) \frac{\delta C}{\delta S}(t, S_1, S_2) \right) k_2(y) dy = 0 \quad (2)$$

su $[0, T] \times [0, +\infty]^2$ con opportune B.C. e valore finale.

Ci siamo inoltre concentrati sul seguente problema di frontiera libera:

$$\frac{\delta P}{\delta t} + \frac{\sigma^2}{2} S^2 \frac{\delta^2 P}{\delta S^2} + r \frac{\delta P}{\delta S} - rC + \\ + \int_{\mathbb{R}} \left(P(t, Se^y) - P(t, S) - S(e^y - 1) \frac{\delta P}{\delta S}(t, S) \right) k(y) dy \leq 0 \quad (3)$$

con le seguenti condizione:

$$P(t, S) \geq max(K - S_T, 0).$$

2 Problema con frontiera libera

In finanza il problema associato all'equazione (3) è abbastanza comune. La pratica standard per risolverlo è implementare un solutore iterativo (come il SOR) e controllare in ogni punto che la soluzione stia sopra all'ostacolo, ottenendo così il Projected SOR. Siccome la classe SparseMatrix di Deal II non ha questo solutore, abbiamo deciso di estenderne le funzionalità creando una classe SparseMatrix_withProjectedSOR che eredita dalla classe SparseMatrix, aggiungendole il PSOR.

3 Strutturazione del codice

Abbiamo completato la scrittura dei codici per risolvere i vari problemi con cui ci siamo confrontati (anche validando con un MonteCarlo il risultato della PIDE 2d) e ora stiamo procedendo alla strutturazione del programma.
Abbiamo creato le seguenti classi:

- classi per gestire i vari modelli in esame: classe base astratta, da cui derivano i modelli utilizzati (Black&Scholes, Kou, Merton);
- classi per gestire le condizioni al bordo e la condizione finale nei vari casi;
- una classe per gestire le diverse densità usate nella parte integrale;
- delle classi che calcolino la parte integrale;
- una classe base OptionBase<dim> astratta, che riconosce i vari modelli utilizzati e alloca dinamicamente la classe corretta per l'integrazione. Questa classe si occupa di creare la mesh e costruire la matrice di sistema;
- una classe EuropeanOption<dim>, che eredita da OptionBase<dim>, e risolve il sistema con un metodo diretto (UMFPACK);
- una classe AmericanOption<dim>, che eredita da OptionBase<dim>, e risolve il problema con frontiera libera tramite il SOR proiettato.

4 Problemi incontrati

Abbiamo riscontrato le seguenti problematiche:

- il primo problema è la distinzione fra trasformazione in logprice, ovvero $x = log(S)$, che porta l'equazione a coefficienti costanti, e la trasformazione $z = Se^y$, che rende il calcolo dell'integrale più rapido. L'idea è quella di lasciar scegliere all'utente nel costruttore della classe opzione se adottare uno o l'altro cambio di variabile. Il codice tuttavia differisce molto fra una trasformazione e l'altra. È quindi conveniente mettere un semplice if e scrivere il codice per l'una e per l'altra in due blocchi distinti, oppure sarebbe meglio fare classi separate, il che significherebbe scrivere due oggetti distinti?

- Un altro problema incontrato è legato al mesh refinement. Durante il calcolo dell'integrale infatti, occorre conoscere quale elemento della mesh è associato alla riga i-esima della matrice, e dopo uno step di refining la corrispondenza fra elementi della matrice e punti della mesh si perde. Stiamo quindi cercando di trovare una funzione che mappi posizioni sulla matrice con punti della mesh in modo da risolvere il problema. Anche il solver per il PSOR è affetto da questo problema, poiché deve controllare che la soluzione stia sopra all'ostacolo, punto per punto.

- La classe SparseMatrix di Deal II, da cui facciamo ereditare la nostra classe con il PSOR, non ha i metodi virtual, perciò chiamando i metodi di SparseMatrix con SparseMatrix_withProjectedSOR il compilatore dice che non trova le funzioni. Abbiamo risolto il problema creando due puntatori al medesimo oggetto, uno di tipo SparseMatrix *, l'altro di tipo SparseMatrix_withProjectedSOR *, dereferenziando il primo per i metodi di SparseMatrix, il secondo per il PSOR. È una pratica corretta oppure c'è un altro modo?

- L'ultimo problema riguarda la gestione della parte integrale. Siccome un modello, Black&Scholes, risolve la semplice PDE, abbiamo pensato di allocare dinamicamente la classe integrale qualora il modello non sia Black&Scholes. Il problema è che le funzioni che calcolano l'integrale sono molto diverse fra 1d e 2d e fra le varie trasformazioni. In particolare, le funzioni 1d devono calcolare un numero e un vettore (con metodi differenti in base al modello), le funzioni 2d due numeri e due vettori. Come possiamo conciliare il tutto?