Riassunto Progetto Pacs

Nahuel Foresta, Giorgio Re

8 maggio 2014

1 Riassunto Obiettivi

Come già indicato nel primo report, l'obiettivo del progetto è di creare un programma per prezzare una serie di opzioni utilizzando dei metodi basati a 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.

2 Strumenti Aggiunti rispetto al report precedente

2.1 Libreria di Integrazione

Poiché le densità all'interno dell'integrale sono del tipo:

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

per il modello di Kou e

$$k(y) = \frac{\lambda}{\delta\sqrt{2\pi}} exp\left\{-\frac{(x-\mu)^2}{2\delta^2}\right\}$$

per il modello di Merton, abbiamo usato dei nodi di quadratura che inglobassero già il peso esponenziale. Sfruttando quindi le funzioni offerte dalla libreria legendre_rule.hpp utilizzata in uno dei laboratori, abbiamo utilizzato i nodi di Laguerre per calcolare l'integrale con il metodo di Kou (ottenendo risultati più precisi rispetto a quelli di Gauss), e pensiamo di utilizzare i nodi di Hermite per il modello di Merton.

3 Cosa è stato fatto in breve

3.1 Fino al report precedente

Al momento dell'ultimo report lo stato era:

- Costruito un programma che risolve l'equazione PDE (senza parte integrale) in una dimensione in un caso semplice utilizzando unicamente gli strumenti forniti dalla libreria.
- Abbiamo risolto lo stesso problema nel caso bi-dimensionale. Sebbene il comportamento qualitativo della soluzione è quello aspettato, i valori esatti non sono ancora giusti (confrontati con dei risultati dati da tool che risolvono l'equazione con metodi alle differenze finite).
- Abbiamo provato a risolvere l'equazione integro differenziale in una dimensione in diversi modi. In due casi siamo riusciti ad ottenere un risultato corretto, ma non pienamente soddisfacenti.

3.2 Fino ad ora

Dopo l'ultimo incontro, abbiamo deciso di esplorare sia l'utilizzo di formule di quadrature più adatte per trattare il termine integrale con nucleo esponenziale, sia di utilizzare funzioni della libreria deal II che permettono di valutare la funzione in un punto qualsiasi. In particolare:

- Abbiamo corretto la PDE in 2D ottenendo il risultato atteso.
- Abbiamo implementato la quadratura diretta del termine integrale utilizzando l'equazione nella forma (3), sia in 1D che in 2D. Si veda la sezione 4 Metodologia per i dettagli.
- Abbiamo implementato il calcolo dell'integrale con i nodi di Laguerre per la quadratura del termine integrale con nucleo esponenziale (modello di Kou).

Abbiamo inoltre uniformato l'uso dei "funtori". Deal II utilizza infatti una classe `Function<dim>` (BC, valore iniziale, coefficienti) e dalla quale è possibile far ereditare altre funzioni. In particolare tale classe implementa due metodi `virtual`, `value` e `value_list` che valutati su un punto (o un vettore di punti) restituiscono il valore nel punto (o nei punti).

4 Metodologia

La PDE (e la PIDE) in questione è trasformabile in un equazione a coefficienti costanti con la trasformazione $x_i = \ln S_i$ (oppure $x_i = \ln {}^{S_i}/_{S_i^0}$) e $C(t, S_1, S_2) = u(t, x_1, x_2)$. In tal caso diventa (2D):

$$\begin{aligned}\frac{\delta u}{\delta t} + \frac{\sigma_1^2}{2}\frac{\delta^2 u}{\delta x_1^2} + \frac{\sigma_2^2}{2}\frac{\delta^2 u}{\delta x_2^2} + \rho\sigma_1\sigma_2\frac{\delta^2 u}{\delta x_1 \delta x_2} + \left(r - \sigma_1^2\right)\frac{\delta u}{\delta x_1}\left(r - \sigma_2^2\right)\frac{\delta u}{\delta x_2} - ru + \\ + \int_{\mathbb{R}}\left(u(t, x_1 + y, x_2) - u(t, x_1, x_2) - (e^y - 1)\frac{\delta u}{\delta x_1}\right)k_1(y)dy + \\ + \int_{\mathbb{R}}\left(u(t, x_1, x_2 + y) - u(t, x_1, x_2) - (e^y - 1)\frac{\delta u}{\delta x_2}\right)k_2(y)dy = 0 \quad (3)\end{aligned}$$

Definendo

$$\hat{\lambda_i} = \int_{\mathbb{R}} u(t, x, y)k_i(y)dy \qquad \hat{\alpha_i} = \int_{\mathbb{R}}(e^y - 1)\frac{\delta u}{\delta x_i}k_i(y)dy$$

L'equazione (3) diventa:

$$\begin{aligned}\frac{\delta u}{\delta t} + \frac{\sigma_1^2}{2}\frac{\delta^2 u}{\delta x_1^2} + \frac{\sigma_2^2}{2}\frac{\delta^2 u}{\delta x_2^2} + \rho\sigma_1\sigma_2\frac{\delta^2 u}{\delta x_1 \delta x_2} + \left(r - \sigma_1^2 - \hat{\alpha_1}\right)\frac{\delta u}{\delta x_1}\left(r - \sigma_2^2 - \hat{\alpha_2}\right)\frac{\delta u}{\delta x_2} + \\ - (r + \lambda_1 + \lambda_2)u + \int_{\mathbb{R}} u(t, x_1 + y, x_2)k_1(y)dy + \int_{\mathbb{R}} u(t, x_1, x_2 + y)k_2(y)dy = 0 \quad (4)\end{aligned}$$

Senza entrare nei dettagli (vedere il primo report), otteniamo una discretizzazione del tipo:

$$M_1 u^k = M_2 u^{k+1} + J^{k+1} \qquad \text{per } k = M \dots 1 \qquad \text{e} \qquad u^M(S) = g(S) \quad (5)$$

Dove M_1 e M_2 sono la somma delle matrici date dagli elementi finiti (mass, stiffnes, ...) date da una discretizzazione temporale di tipo theta-metodo. Dall'ultimo report, ci siamo concentrati su un modo di calcolare il termine esplicito J.

4.1 La parte integrale J

Nell'ultimo report erano spiegati i problemi che il calcolo di questa parte introduce così come due approcci possibili. Ci siamo concentrati sull'approccio che non genera una matrice densa, ma richiede il calcolo del vettore J a ogni iterata temporale.
Riassumendo, si tratta di calcolare in ogni nodo $x^i = (x_1^i, x_2^i)$ della griglia due valori di J:

$$J_1^i = J_1(x^i) = \int_{\mathbb{R}} u(t, x_1^i + y, x_2^i)k_1(y)dy \text{ e } J_2^i = J_2(x^i) = \int_{\mathbb{R}} u(t, x_1^i, x_2^i + y)k_2(y)dy$$

nei diversi nodi x^i della griglia. In seguito si potrà scrivere dunque

$$\sum_{j=1}^{N} J_i^j \int_{x_{min}}^{x_{max}} \phi_i(x)\phi_j(x)dx$$

scrivibile come $M\underline{J}$, con M matrice di massa, e aggiungere tale termine all'*rhs*. L'eq da risolvere a ogni passo temporale è dunque:

$$M_1 u^k = M_2 u^{k+1} + M\underline{J_1}^{k+1} + M\underline{J_2}^{k+1} \qquad \text{per } k = M \dots 1$$

Per calcolare tale integrale è necessario il valore di u sia in punti interni al dominio che però non appartengono alla mesh, sia fuori dalla mesh. Per quanto riguarda i punti all'interno della mesh abbiamo usato una funzione interna alla libreria deal II che permette di valutare la soluzione in un punto dato. Siccome valutare la funzione in un punto qualunque richiede una ricerca sulla griglia per individuare in quale cella si trova il nodo, questo processo può essere lento. Per quanto riguarda i valori fuori dal dominio, imponiamo il valore del payoff, procedura standard in questo caso.

In sostanza abbiamo più griglie:

- **Griglia di dominio** Una bidimensionale, la griglia che discretizza il dominio.
- **Griglie di integrazione** Una griglia monodimensionale per ogni direzione di interpolazione (due nel caso bidimensionale).

Siccome nella valutazione della soluzione in un punto la ricerca della cella di appartenenza è un operazione lenta, abbiamo notato che è meglio utilizzare una griglia di integrazione con meno celle e più nodi per cella.

4.1.1 Un integrazione più corretta

Come anticipato sopra, abbiamo implementato l'uso di nodi di Gauss-Laguerre per la quadratura dell'integrale. In questo caso l'uso di un troncamento B_{min}, B_{max} non è necessario. A livello di tempi c'è un guadagno (circa il 30%). Per ora è stato testato solo in 1D, ma non ci dovrebbero essere problemi a estenderlo al caso 2D siccome gli integrali nell'rhs sono comunque monodimensionali.

4.1.2 Possibile parallelizzazione

Per come viene eseguito il calcolo del vettore J, una strada che sembrerebbe interessante è la parallelizzazione di tale operazione. Essendo questo il cono di bottiglia del metodo, porterebbe essere un ottimo modo di aumentare le prestazioni. Per il momento abbiamo provato una parallelizzazione con *openmp* semplicemente dividendo il vettore J fra i vari thread. Come si può osservare in Appendice A, ciò introduce una buona miglioria.

5 Conclusioni parziali e futuri passi

Riassumendo, il calcolo di J effettuando una quadratura ad ogni iterazione temporale dà risultati corretti. Per poter effettuare tale quadratura, è necessario valutare la funzione in punti non della griglia il che introduce un calo nella velocità.

5.1 Prossimi passi

I prossimi passi previsti sono:

- Controllare l'opzione asiatica: Il valore di un opzione monodimensionale di questo tipo può essere trovato con una PDE/PIDE bidimensionale molto simile a quella trattata fin'ora. Un primo tentativo restituisce un valore non corretto, bisogna capire il perché.
- Riscrivere l'integrazione nel modello di Kou bidimensionale utilizzando nodi di quadratura di Laguerre.
- Implementare il modello di Merton: al posto di un nucleo esponenziale si ha un nucleo gaussiano, da trattare in principio con una quadratura apposita (nodi di Hermite).
- Trattare l'opzione di tipo americano: tali opzioni introducono un problema di frontiera libera in cui occorre risolvere la (1) e la (2) con il segno di disuguaglianza $\leq$ al posto dell'uguaglianza. L'approccio standard è di utilizzare un solver iterativo (tipicamente SOR) ed aggiungervi la condizione che la soluzione stia sopra l'ostacolo (il PayOff). Nel caso matrice tridiagonale (1D) è immediato (ed è già stato implementato), mentre bisogna studiare quale sarebbe il miglior metodo per matrici generate dal problema 2D.
- Parallelizzare il calcolo del vettore integrale J. Con openmp dovrebbe essere abbastanza facile, è fattibile/migliorabile utilizzando MPI?
- Valutare la possibilità di fare adattazione di griglia e in questo caso quale stima dell'errore usare.

Oltre a questo, dare una forma al programma, decidendo il design da utilizzare nei vari punti.

5.2 Approcci abbandonati

Per il momento abbiamo deciso di non esplorare il cambio di variabili $Se^y = z$ a livello equazione. In tal caso si otterrebbe un equazione a coefficienti non costanti, ma la parte integrale potrebbe essere più semplice. Tuttavia questa trasformazione potrebbe dare luogo a instabilità.
L'altro approccio abbandonato è la scrittura della funzione u nella base ad elementi finiti direttamente prima dell'integrazione. Questo darebbe però luogo a una matrice densa, per questo abbiamo deciso di non continuare su quella strada (vedere il primo report).

A Tabella di convergenza

Di seguito i risultati nel caso monodimensionale

```
 ****** CONVERGENCE TABLE 1D V012

Results for 100 time iterations (serial):
Grid 16 Price 12.5057 clocktime 0.52997s realtime 0.53684s
Grid 32 Price 12.4252 clocktime 1.36887s realtime 1.37507s
Grid 64 Price 12.3916 clocktime 3.76746s realtime 3.78003s
Grid 128 Price 12.3837 clocktime 10.9453s realtime 11.1629s
Grid 256 Price 12.3823 clocktime 36.8916s realtime 38.1013s
Grid 512 Price 12.3814 clocktime 119.291s realtime 119.976s
Grid 1024 Price 12.3814 clocktime 422.728s realtime 425.07s

real 10m0.041s
user 9m55.040s
sys 0m0.511s


Results for 100 time iterations (parallel dual-core):
Grid 16 Price 12.5057 clocktime 0.63368s realtime 0.35037s
Grid 32 Price 12.4252 clocktime 1.63260s realtime 0.87694s
Grid 64 Price 12.3916 clocktime 4.38279s realtime 2.49441s
Grid 128 Price 12.3837 clocktime 12.6560s realtime 6.77907s
Grid 256 Price 12.3823 clocktime 38.2390s realtime 20.8882s
Grid 512 Price 12.3814 clocktime 126.062s realtime 79.3415s
Grid 1024 Price 12.3814 clocktime 441.518s realtime 250.999s

real 6m1.768s
user 10m17.468s
sys 0m7.691s
```