ENSIIE

BLACK-SCHOLES
RAPPORT

Résolution numérique de l'équation différentielle de Black-Scholes

Élèves :
Amaury MANZIONE
Joris NOZI

Enseignant :
Vincent TORRI

2 janvier 2023

Table des matières

1 EDP complète

On cherche à approximer numériquement l'équation aux dérivées partielles complète de Black-Scholes complète et réduite.

Nous travaillons ici avec l'EDP complète :

$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial^2 S} = rC \quad (1)$$

Nous avons approximé (1) par la méthode des différences finies en gérant le terme en $\frac{\partial}{\partial t}$ par une approximation décentrée à gauche (car on a une condition terminale par rapport à t), le terme $\frac{\partial}{\partial S}$ par une approximation centrée et le terme $\frac{\partial^2}{\partial^2 S}$ par crank-nichelson :

$$\begin{cases} \frac{\partial C}{\partial t} = \frac{C(t_m,s_j)-C(t_{m-1},s_j)}{\Delta t} \\ \frac{\partial C}{\partial S} = \frac{C(t_m,s_{j+1})-C(t_m,s_{j-1})}{2\Delta S} \\ \frac{\partial^2 C}{\partial^2 S} = \frac{C(t_m,s_{j+1})-2C(t_m,s_j)+C(t_m,s_{j-1})+C(t_{m-1},s_{j+1})-2C(t_{m-1},s_j)+C(t_{m-1},s_{j-1})}{4\Delta^2 S} \end{cases}$$

En remplaçant ces équations dans (1) on trouve :

$$\gamma_j C(t_m,s_j) + (\beta_j+\alpha_j)C(t_m,s_{j+1}) + (-\beta_j+\alpha_j)C(t_m,s_{j-1}) = (1+2\alpha_j)C(t_{m-1},s_j)$$
$$-\alpha_j(C(t_m,s_{j+1}) - \alpha_j C(t_m,s_{j-1}))$$

avec

$$\begin{cases} \alpha_j = \frac{\sigma^2 s_j^2 \Delta t}{8\Delta^2 S} \\ \beta_j = \frac{r s_j \Delta t}{2\Delta S} \\ \gamma_j = 1 - r\Delta t - 2\alpha_j \end{cases}$$

En utilisant les conditions de bord quand $s=0$ et $s=L$ cela se traduit matriciellement par :

$$\underbrace{\begin{pmatrix} \gamma_1 & \beta_1+\alpha_1 & & \\ -\beta_2+\alpha_2 & \gamma_2 & \beta_2+\alpha_2 & \\ & \ddots & \ddots & \ddots \\ & & -\beta_{L-1}+\alpha_{L-1} & \gamma_{L-1} \end{pmatrix}}_{A_1\in\mathcal{M}_{L-1}(\mathbb{R})} \underbrace{\begin{pmatrix} C(t_m,s_1) \\ \vdots \\ \vdots \\ C(t_m,s_{L-1}) \end{pmatrix}}_{b_1\in\mathbb{R}^{L-1}} + \underbrace{\begin{pmatrix} 0 \\ \vdots \\ \vdots \\ (\beta_{L-1}+\alpha_{L-1})C(t_m,s_L)+\beta_{L-1}C(t_m \end{pmatrix}}_{k\in\mathbb{R}^{L-1}}$$

$$||$$

$$\underbrace{\begin{pmatrix} 1+2\alpha_1 & -\alpha_1 & & \\ -\alpha_2 & 1+2\alpha_2 & \alpha_2 & \\ & \ddots & \ddots & \ddots \\ & & -\alpha_{L-1} & 1+2\alpha_{L-1} \end{pmatrix}}_{A_2\in\mathcal{M}_{L-1}(\mathbb{R})} \underbrace{\begin{pmatrix} C(t_{m-1},s_1) \\ \vdots \\ \vdots \\ C(t_{m-1},s_{L-1}) \end{pmatrix}}_{b_2\in\mathbb{R}^{L-1}}$$

pour une option call et :

$$\begin{pmatrix} \gamma_1 & \beta_1+\alpha_1 & & \\ -\beta_2+\alpha_2 & \gamma_2 & \beta_2+\alpha_2 & \\ & \ddots & \ddots & \ddots \\ & & -\beta_{L-1}+\alpha_{L-1} & \gamma_{L-1} \end{pmatrix} \begin{pmatrix} C(t_m,s_1) \\ \vdots \\ \vdots \\ C(t_m,s_{L-1}) \end{pmatrix} + \begin{pmatrix} (-\beta_1+\alpha_1)C(t_m,s_0)+\alpha_1 C(t_{m-1},s_0) \\ 0 \\ \vdots \end{pmatrix}$$

$$\|$$

$$\begin{pmatrix} 1+2\alpha_1 & -\alpha_1 & & \\ -\alpha_2 & 1+2\alpha_2 & \alpha_2 & \\ & \ddots & \ddots & \ddots \\ & & -\alpha_{L-1} & 1+2\alpha_{L-1} \end{pmatrix} \begin{pmatrix} C(t_{m-1}, s_1) \\ \vdots \\ \vdots \\ C(t_{m-1}, s_{L-1}) \end{pmatrix}$$

pour une option put.

Connaissant $C(t_m, s_j) \quad \forall\ 0 \leq j \leq L$, nous cherchons à calculer $C(t_{m-1}, s_j)$ pour tout j, ce qui revient à résoudre le système :

$$A_2 b_2 = b$$

avec $b = A_1 b_1 + k$.
Nous utilisons la formule de décompostion LU pour une matrice tridiagonale vu en cours de MAN pour résoudre ce système.

2 EDP réduite

2.1 Changements de variables

On s'est d'abord documenté sur les changements de variables à faire pour obtenir cette nouvelle équation aux dérivées partielles. Voici ceux utilisés :

$$\begin{cases} \tilde{t} = \frac{\sigma^2(T-t)}{2} \\ \tilde{s} = ln(\frac{s}{K}) \\ \tilde{C} = \frac{1}{K} e^{\frac{1}{2}(k-1)\tilde{s}+\frac{1}{4}(k+1)^2\tilde{t}} C \quad avec \quad k = \frac{2r}{\sigma^2} \end{cases}$$

Ces changement de variables amènent à l'EDP suivante :

$$\frac{\partial \tilde{C}}{\partial \tilde{t}} = \frac{\partial^2 \tilde{C}}{\partial \tilde{s}^2} \quad (1')$$

Mais ces changements de variables amènent aussi à reconsidérer les maillages utilisés dans la partie 1 car ils ne sont plus valables. Même s'il n'y a pas de problème pour le maillage de $\tilde{t}$ car c'est un changement de variable affine, on doit changer celui de s car ln(0) est indéterminé.
On transforme la borne haute de l'intervalle de t avec le changement de variable : $\tilde{t}_{sup} = ln(\frac{L}{K})$ On a ensuite choisi de prendre comme borne inférieur l'opposé : $\tilde{t}_{inf} = -ln(\frac{L}{K})$.
Ainsi, si de base on avait un maillage de [0,+∞[avec [0,L], on peut supposer qu'ici on a un maillage de]-∞,+∞[avec $[-ln(\frac{L}{K}), ln(\frac{L}{K})]$.
Ce choix d'intervalle fût compliqué car ce n'est pas celui le plus intuitif. Nous avons d'abord tester de transformer le maillage de s grâce au changement de variable directement. Mais cela impliquait un maillage non uniforme (à cause du log), rendant le schéma aux différences finies implicite plus compliqué car h n'était plus constant.

2.2 Schéma aux différences finies implicite

Il nous est demandé d'utiliser le schéma aux différences finies implicite qui donne les approximations suivantes à (t_m, s_j)

$$\begin{cases} \frac{\partial \tilde{C}}{\partial \tilde{t}}(\tilde{t}_m, \tilde{s}_j) = \frac{1}{\Delta \tilde{t}}[\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_j) - \tilde{C}(\tilde{t}_m, \tilde{s}_j)] \\ \frac{\partial^2 \tilde{C}}{\partial \tilde{s}^2}(\tilde{t}_m, \tilde{s}_j) = \frac{1}{h^2}[\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_{j+1}) - 2\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_j) + \tilde{C}(\tilde{t}_{m+1}, \tilde{s}_{j-1})] \end{cases}$$

En remplaçant dans (1') à (t_m, s_j), on trouve :

$$\frac{1}{\Delta \tilde{t}}[\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_j) - \tilde{C}(\tilde{t}_m, \tilde{s}_j) = \frac{1}{h^2}[\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_{j+1}) - 2\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_j) + \tilde{C}(\tilde{t}_{m+1}, \tilde{s}_{j-1})]$$

Puis :

$$k_1\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_{j+1}) + k_2\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_j) + k_1\tilde{C}(\tilde{t}_{m+1}, \tilde{s}_{j-1}) = k_3\tilde{C}(\tilde{t}_m, \tilde{s}_j)$$

avec

$$\begin{cases} k_1 = \frac{1}{h^2} \\ k_2 = -(\frac{2}{h^2} + \frac{1}{\Delta \tilde{t}}) \\ k_3 = -\frac{1}{\Delta \tilde{t}} \end{cases}$$

On peut ainsi la traduire en un système matriciel comme suit :

$$A\tilde{C}_{m+1} = k_3\tilde{C}_m - k_1\tilde{c}_{lim}$$

avec

$$A = \begin{pmatrix} k_2 & k_1 & & \\ k_1 & k_2 & k_1 & \\ & \ddots & \ddots & \ddots \\ & & k_1 & k_2 \end{pmatrix}$$

$\tilde{c}_{lim}$ correspond aux conditions aux limites de $\tilde{C}$. En particulier, en $\tilde{s} = 0$ et $\tilde{s} = L$. Ainsi, cette valeur change en fonction du payoff : CALL ou PUT. Prenons le cas du CALL par exemple :

$$\begin{cases} \tilde{C}(\tilde{t}, 0) = C(t, 0) = 0 \\ \tilde{C}(\tilde{t}, ln(\frac{L}{K})) = \frac{1}{K}e^{\frac{1}{2}(k-1)\tilde{s} + \frac{1}{4}(k+1)^2\tilde{t}}C(t, L) \\ \qquad = e^{\frac{1}{2}(k-1)\tilde{s} + \frac{1}{4}(k+1)^2\tilde{t}}e^{-r(t-T)} \end{cases}$$

Dans cet exemple, $\tilde{c}_{lim}$ vaudrait :

$$\tilde{c}_{lim} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ e^{\frac{1}{2}(k-1)\tilde{s} + \frac{1}{4}(k+1)^2\tilde{t}}e^{-r(t-T)} \end{pmatrix}$$

3 Architecture des classes

Tout d'abord nous avons créé une classe abstraite edp qui possède comme attributs toutes les caractéristiques d'une option $(K, r, \sigma, T,$etc..) et deux std : :vector<double> qui représentent les maillages par rapport à t et s. Elle possède une méthode virtuelle pure sol_edp() qui renvoie une matrice de terme générale $C(t_m, s_j) \quad \forall \ 0 \leq j \leq L, \ 0 \leq$

$m \leq T$. Nous avons ensuite créé deux classes filles qui implémentent sol_edp, reduite qui résout l'équation reduite avec le schéma vu en 2 et crank qui résout l'équation complète avec le schéma vu en 1.

Nous avons également crée deux autres classes :

— Matrix qui représente symboliquement une matrice tridiagonale. Comme ces types de matrices ont beaucoup de zéros, nous avons jugé plus pertinent de construire un objet de type Matrix avec trois std : :vector représentant les termes de la diagonale ,ceux en dessous et au dessus. En plus des constructeurs par défauts et valués, nous avons créé un constructeur prenant en paramètres 3 double et un int (la taille de la matrice) quand la matrice diagonale a des termes constants sur ses diagonales (utile pour la classe reduite). La classe Matrix possède un opérateur externe représentant le produit d'une matrice par un vecteur. Pour les opérations vectorielles comme add_lambda et prod_vect qui n'ont pas de lien avec la classe Matrix nous avons créé un namespace algebra qui contient ces deux fonctions et la classe Matrix.

— La classe sdl gère les affichages graphiques, elle a notamment en attribut la largeur et la hauteur de la fenêtre. Elle a une méthode draw_function qui prend en argument un std : :vector<double> et le dessine sur la fenêtre.

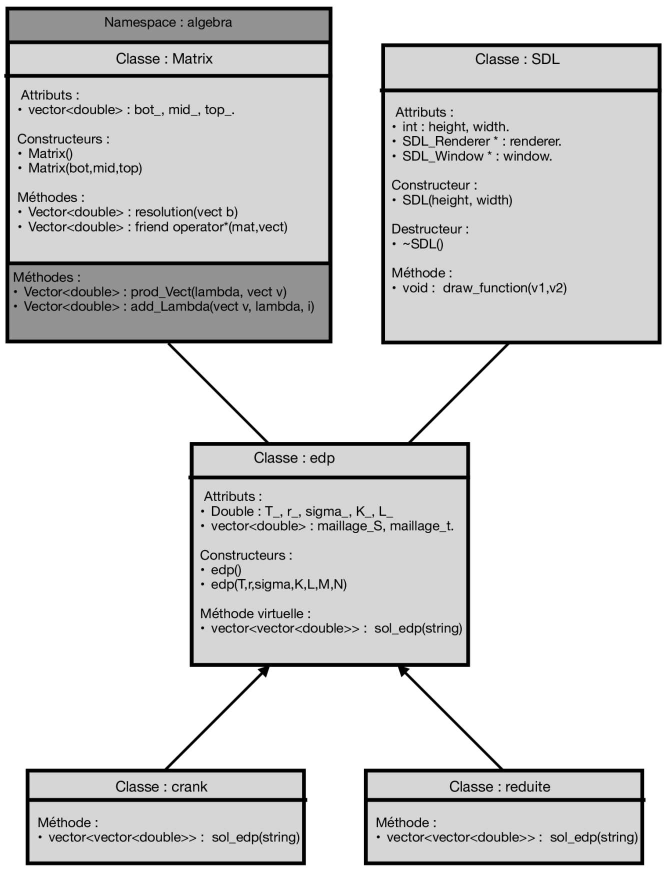

Figure 1 – Diagramme UML des classes

4 Quelques commandes

On a trouvé dans les parties précédentes que les sytèmes étaient :

$$\begin{cases} AC_{m-1} = BC_m + c_{lim} \\ A\tilde{C}_{m+1} = k_3\tilde{C}_m - k_1\tilde{c}_{lim} \end{cases}$$

Nous avons codé des fonctions pour calculer le terme de droite du système en optimisant au maximum.

— $k_3\tilde{C}_m$
Il s'agit d'un produit entre un vecteur et un réel. La fonction prod_Vect de la classe Matrix fait le produit terme à terme du vecteur par le réel.

— BC_m
Il s'agit d'un produit entre une matrice et un vecteur colonne. Cela est traité par l'opérateur externe * du namespace algebra. On profite dans cette fonction du fait que B soit tridiagonal pour optimiser le calcul également.

— $k_3\tilde{C}_m - k_1\tilde{c}_{lim}$ ou $BC_m + c_{lim}$
Il s'agit cette fois d'une addition de deux vecteurs. Mais le terme de droite a qu'une seule composante (la première ou la dernière) et tout le reste sont des 0. Donc il n'est pas nécessaire d'ajouter terme à terme chaque composante des vecteurs, mais juste additionner la première ou la dernière composante des deux vecteurs. Cela est fait par add_Lambda.

Le reste est la résolution du système AX=b avec A tridiagonale comme dans la première partie.