Rapport de notre projet de programmation en C++

Pricing par EDP

Malo David, Maxime Quille, Thomas Lambelin

Décembre 2024

Résumé

Ce document présente notre implémentation d'une méthode de résolution numérique de l'équation de Black-Scholes pour le pricing d'options européennes, basée sur la méthode explicite des différences finies. Dans un premier temps, nous décrivons et expliquons le code que nous avons développé pour aborder ce problème (disponible ici : `https://github.com/malo-david/CPP-Pricing-Using-EDP`). Ensuite, nous retraçons l'évolution du projet, en mettant en lumière les ajustements et améliorations réalisés au fil de l'implémentation. Enfin, nous discutons des perspectives d'amélioration et des possibilités d'extension de cet algorithme.

Table des matières

1 Explication de notre code

1.1 Structure utilisée

La structure du code repose sur une classe centrale, `FiniteDifferencePricer`, qui encapsule l'ensemble des paramètres et des méthodes nécessaires pour évaluer le prix d'une option européenne. Nous avons fait ce choix afin d'améliorer d'une part la clarté du code, et d'autre part sa modularité.

Les caractéristiques de l'option sont encapsulées dans la structure `Parameters`, membre imbriqué dans la classe centrale. Cette structure permet de centraliser les caractéristiques de l'option et de simplifier leur passage et leur manipulation au sein de la classe.

Les membres publics du code sont la méthode `run` et la structure `Parameters`. Les attributs et méthodes internes au calcul sont définies comme privés. Cela garantit notamment l'intégrité des paramètres de discrétisation (`N,M,dt,dS` etc.) et des vecteurs `U` et `U_old`. En somme, on garantit que la configuration des paramètres ne peut être déclenchée qu'à travers l'exécution de `run()`, simplifiant ainsi l'interface utilisateur exposée (détaillée en 1.3).

1.2 Approche mathématique et algorithmique

Nous avons choisi d'utiliser le schéma des différences finies explicite, nous semblant plus accessible d'implémentation dans un premier temps. Cette méthode repose sur une récurrence finie, permettant de *remonter dans le temps* une fois l'échelle des temps et des prix discrétisées, et les conditions limites calculées. La formule que nous avons utilisée est la suivante :

$$U_j^{n+1} = (\alpha - \beta)U_{j-1}^n + (1 - r\,dt - 2\alpha)U_j^n + (\alpha + \beta)U_{j+1}^n$$

Où U_j^n désigne le prix de l'option à n instants en partant de la fin (donc de $t = T$, soit à $t = T - n \cdot dt$) et j correspond au prix de l'actif sous-jacent ($S = jdS$). Les constantes α et β sont définies comme suit :

$$\alpha = \frac{\sigma^2 S^2 dt}{2dS^2}, \quad \beta = \frac{rSdt}{2dS}$$

En pratique, nous n'avons pas de U^n, mais seulement deux vecteurs : le vecteur `U` est utilisé pour stocker les résultats des calculs à chaque itération temporelle, et un second vecteur `U_old` est employé pour sauvegarder l'état précédent. Ce choix est plus léger en mémoire qu'une matrice complète de dimensions $N \times M$, tout en étant suffisant pour résoudre l'équation de manière efficace.

De plus, la prise en charge des options put est intégrée directement via la parité put-call, évitant ainsi une duplication inutile des calculs. Une fois le prix de l'option call calculée, on obtient le prix du put :

$$P_t = C_t - S_0 + K \cdot \exp(-rT)$$

De même pour le calcul des Grecques qui au lieu de reposer sur la méthode des différences finies (comme pour le call) repose sur cette même parité qui mène analytiquement à :

$$\Delta_p = \Delta_c - 1, \quad \Gamma_p = \Gamma_c$$

Enfin, tout au long de l'algorithme, nous utilisons la condition de stabilité suivante, assurant que le schéma des différences finies ne diverge pas :

$$dt \leqslant \frac{dS^2}{\sigma^2 S_{\max}^2}$$

Cette dernière (`checkStability()`) est systématiquement vérifiée avant d'appeler la méthode implémentant le calcul du prix de l'option call : `computeCallOptionPrice()`, et est également utilisée dans la méthode `configureMode(mode)` où cette dernière est saturée afin de choisir le plus grand pas dt possible, adapté au pas dS souhaité (et configuré via le mode choisi).

1.3 Fonctionnement orienté utilisateur

L'utilisateur n'a pas besoin d'appeler de fonctions, il suffit d'exécuter le code puis de rentrer les paramètres de l'option (exécution de `inputParameters()`) et de choisir le mode (exécution de `chooseMode()`). Une fois cela fait, la console affiche le prix de l'option, la précision utilisée (valeur et nombre de pas temporels) ainsi que les sensibilités Delta et Gamma (méthodes `displayResults()` et `configureMode(mode)`).

2 Progression et évolution du projet

La première étape fut de choisir le type de produit dérivé que nous allions pricer, et nous nous sommes tournés vers les options européennes car nous avions vu leur fonctionnement détaillé en cours, et que nous connaissions les bases de leur pricing.

Par la suite, nous nous sommes concentrés sur l'implémentation du schéma des différences finies explicites pour un call, puisque nous avions trouvé de la documentation sur celle-ci (cf. [1]).

Au départ, afin de tester notre programme (la partie algorithmique en tant que telle) nous avions fixé les paramètres d'une l'option dont on connaissait le prix afin de comparer ce dernier avec celui que notre programme calculait. Une fois choisis les paramètres de discrétisation de sorte à ce que la condition de stabilité soit vérifiée, nous avons obtenu une estimation du prix correcte.

C'est alors que l'on décida de permettre à l'utilisateur de choisir les caractéristiques de l'option. Pour les quelques tests que nous avons effectué, le prix calculé était correct. On permis également à l'utilisateur de choisir entre plusieurs modes de précisions (qui initalement correspondaient à des valeurs plus ou moins grandes mais fixées de `M` et `N`). Seulement, en testant des valeurs peu générales pour la volatilité, nous avons retrouvé des cas de divergence du schéma. C'est la raison pour laquelle `M` est choisi en saturant la contrainte de stabilité à partir de `N`. Enfin, un dernier ajustement a été nécessaire : l'ajout du `min` dans le calcul de `dt` pour éviter un nombre de pas temporel `M` trop petit dans le cas de volatilités faibles (par saturation de la condition de stabilité, voir Figure 1 ci-dessous).

```
Entrez les paramètres de l'option :
Type d'option (0 pour put, 1 pour call) : 1
Prix de l'actif sous-jacent initial (S0) : 100
Prix Strike (K) : 50
Taux sans risque (r) : 0.03
Volatilité (sigma) : 0.01
Maturité (T) : 1
Choisissez un mode :
1. Précis (petits pas, mais exécution lente)
2. Rapide (grands pas, mais moins précis)
3. Personnalisé (nombre de pas spatial à choisir)
2
Pas temporel calculé (dt) : 1, Nombre de pas temporels (M) : 2
Prix de l'option : 52.955
Delta : 1
Gamma : -1.77636e-15
```

FIGURE 1 – Avant ajustement, la saturation automatique de la condition de stabilité donnait une discrétisation triviale de l'échelle temporelle (dt=T) pour des volatilités trop faibles.

Puis, nous avons généralisé le code en permettant le pricing d'options put en utilisant la parité put-call, y compris pour le calcul des Grecques (cf. [2]).

Enfin, nous avons complété le code par l'ajout de messages d'erreurs et de boucles dans `inputParameters()` dans le cas où l'utilisateur renseigne des valeurs aberrantes (par exemple 3 au lieu de 0.03 pour le taux d'intérêt, ou bien un nombre négatif pour le prix de l'actif sous-jacent).

3 Perspectives

À l'avenir, nous pourrions améliorer le programme en permettant le pricing d'options américaines (ce qui nécessiterait de faire le choix d'une matrice plutôt que d'un double vecteur), mais aussi d'implémenter une interface graphique plus poussée avec par exemple une grille des prix de l'option en fonction du prix du sous-jacent et du temps à maturité. Enfin, il serait pertinent de boucler sur les paramètres de précision (temporels et spatiaux) afin d'obtenir l'évolution de la justesse du prix de l'option calculé en fonction du temps de calcul, et éventuellement d'en déduire un palier de précision à partir duquel le prix calculé est juste en général.

4 Bibliographie

[1] Olivier Guibé. *Méthodes numériques pour la finance*. 2010. p.29. url : `http://math.mad.free.fr/wordpress/wp-content/uploads/M2MNF_cours.pdf`

[2] Fanny Vidal. *Financial Instruments Lecture 5 : Vanilla Options and Valuation with (a) Binomial Trees and (b) Black-Scholes Model.*