Rapport de Projet C++ : Pricer

Simon Teissier et Tilian Bourachot

Ensae 2024 - 2025

Introduction

Ce projet consiste à déterminer le prix d'une option financière en utilisant le modèle de Black-Scholes-Merton, ainsi qu'à mettre en œuvre une stratégie de réplication. Dans les cas où les formules analytiques ne s'appliquent pas, les prix sont calculés par la méthode de Monte-Carlo, et avec un arbre binomial pour des options américaines.

Nous avons aussi décider d'implémenter des stratégies de couverture *Delta Hedging* et *Delta-Gamma Hedging* dans un contexte d'options financières. Le programme développé permet de simuler ces stratégies et d'en visualiser les résultats grâce à des graphiques et des exports CSV. Les méthodes employées s'appuient sur les concepts mathématiques enseignés dans le cours *Instruments financiers* de Mme Vidal, suivi au premier semestre.

Description des fichiers et structure

Le projet est organisé en plusieurs fichiers pour assurer une modularité et une lisibilité optimales.

- **Option.h et Option.cpp :** Fichiers de base définissant une classe abstraite `Option`, qui sert de modèle pour les options européennes et américaines. Contient des méthodes pour calculer des grecs et des prix basés sur le modèle de Black-Scholes-Merton.
- **CallOption.h et CallOption.cpp :** Implémentation d'une option européenne de type `Call`, avec des méthodes pour calculer son prix et ses grecs (Delta, Gamma, Vega, Theta, Rho).
- **PutOption.h et PutOption.cpp :** Implémentation d'une option européenne de type `Put`, similaire à `CallOption`.
- **AmericanOption.h et AmericanOption.cpp :** Définit une classe abstraite `AmericanOption` qui étend la classe `Option`, avec un focus sur les options américaines utilisant un arbre binomial.
- **AmericanCallOption.h et AmericanCallOption.cpp :** Implémentation d'une option américaine de type `Call`, basée sur un arbre binomial pour gérer l'exercice anticipé.
- **AmericanPutOption.h et AmericanPutOption.cpp :** Implémentation d'une option américaine de type `Put`, également basée sur un arbre binomial.
- **Hedging.h et Hedging.cpp :** Implémentent les stratégies de couverture :
  - `delta_hedging` : Gère la couverture dynamique basée sur le Delta.
  - `delta_gamma_hedging` : Combine Delta et Gamma pour une couverture plus précise.
- **Hedge.cpp :** Implémente les stratégies de couverture *Delta Hedging* et *Delta-Gamma Hedging*.
- **Price.cpp :** Effectue les calculs des prix et grecs des options européennes et américaines.

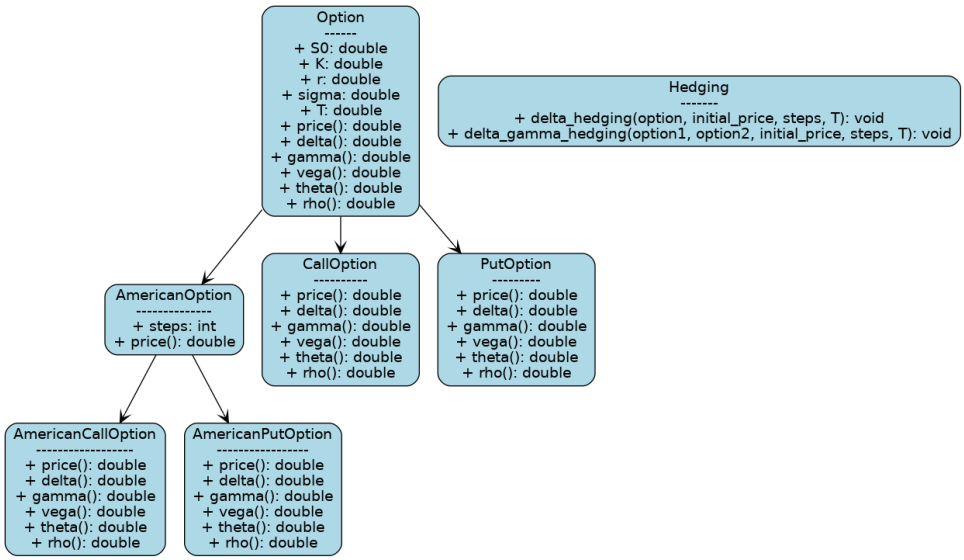

Figure 1: UML diagramme

Exécution des fichiers

Pour exécuter les fichiers, ouvrez un terminal, placez-vous dans le répertoire du projet, puis lancez la commande ./build.sh avec les paramètres suivants.

- `build.sh price` génère un exécutable `Price.exe` permettant d'obtenir uniquement les prix et les grecs des options.
- `build.sh hedge` permet de traiter les stratégies de couverture *Delta Hedging* et *Delta-Gamma Hedging*. Cela génère un exécutable `Hedge.exe` qui, une fois exécuté, crée deux fichiers CSV pouvant être visualisés en exécutant le script `graphique.py`.

```
$ ./build.sh
Launch build.sh with following parameters:
        - clean : remove all generated files
        - price : build the programme to price the products
        - hedge : build the programme to show delta hedging and gamma delta hedging
        - all   : build price & hedge
```

Figure 2: lignes de commandes

Architecture et concepts de C++

Polymorphisme, Pointeurs et Simulation Monte-Carlo

Le polymorphisme et les pointeurs jouent un rôle central dans ce projet, permettant une architecture flexible et extensible. La classe abstraite `Option` sert de base commune pour les différents types d'options financières (`CallOption`, `PutOption`), et définit des méthodes virtuelles comme `price()` et `delta()` qui sont redéfinies dans les classes dérivées selon leur logique spécifique.

Ce mécanisme garantit que des fonctions génériques, telles que `delta_hedging` ou `monte_carlo_pricer`, peuvent manipuler des objets `Option*` sans se soucier de leur type réel (call ou put). Dans `monte_carlo_pricer`, par exemple, un pointeur vers `Option` permet de calculer le payoff de manière polymorphe, en appelant dynamiquement les implémentations spécifiques des sous-classes.

Les pointeurs intelligents, tels que `std::unique_ptr`, sont utilisés pour gérer dynamiquement la mémoire tout en évitant les fuites. Par exemple :

```
std::unique_ptr<Option> call = std::make_unique<CallOption>(S0, K, r, sigma, T);
```

Ces pointeurs automatisent la libération des ressources tout en restant compatibles avec le polymorphisme, assurant ainsi une gestion robuste des objets dynamiques.

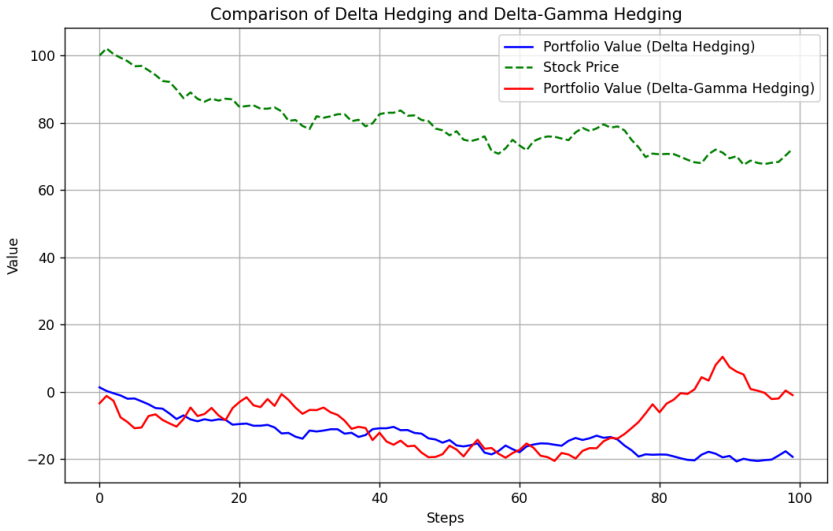

Figure 3: Graphique obtenu avec graphique.py

Pour le pricing des options via Monte-Carlo, nous utilisons la fonction `simulate_path` pour générer des trajectoires simulées du sous-jacent, basées sur le modèle de Black-Scholes-Merton. Chaque trajectoire est divisée en plusieurs pas de temps, et les mouvements aléatoires sont modélisés avec une loi normale. La fonction `monte_carlo_pricer` calcule ensuite le prix de l'option en actualisant la moyenne des payoffs simulés.

Grâce à cette conception modulaire et générique, le même algorithme peut s'adapter à différents types d'options (call, put) et être facilement étendu pour inclure de nouvelles classes ou modèles stochastiques. En combinant polymorphisme et gestion efficace des pointeurs, notre architecture offre une solution robuste aux problématiques financières complexes.

Interface utilisateur et personnalisation

Dans un souci de rendre le programme plus convivial et user-friendly, nous avons intégré une interaction initiale avec l'utilisateur. Lors de l'exécution, le programme demande à l'utilisateur s'il souhaite utiliser des valeurs par défaut pour les paramètres (comme le prix initial, le taux sans risque ou la volatilité) ou s'il préfère entrer ses propres valeurs personnalisées. Cette flexibilité permet une adaptation facile du programme à différents scénarios, que ce soit pour des tests rapides avec les valeurs standards ou pour des analyses spécifiques basées sur des données réelles ou hypothétiques.

1 Problèmes rencontrés et solutions

Lors du développement de notre projet, plusieurs défis ont été rencontrés :

- **Fermeture immédiate de la fenêtre d'exécution :** Cela était dû à l'absence d'une pause à la fin du programme, nous empéchant ainsi d'observer les résultats. Nous avons ajouté une commande qui permet d'attendre une touche pour se fermer.
- **Incohérences des résultats avec certains paramètres :** Lors des premières simulations, les valeurs des grecs calculées par Monte-Carlo divergeaient pour des valeurs extrêmes de volatilité ou de taux sans risque. Nous avons ajusté la précision des calculs en augmentant le nombre de trajectoires simulées.
- **Complexité dans la gestion des méthodes virtuelles :** La classe abstraite `Option` définit des méthodes virtuelles pures pour les grecs (`gamma`, `vega`, `theta`, `rho`), ce qui oblige les classes dérivées à fournir une implémentation. Cependant, ces grecs ne sont pas toujours applicables aux options américaines évaluées par des arbres binomiaux. Pour résoudre ce problème tout en respectant les exigences de conception, nous avons choisi de retourner des valeurs par défaut (`0.0`) pour ces

méthodes dans les classes correspondantes. Cette décision a nécessité une réflexion approfondie sur l'équilibre entre modularité et pertinence des implémentations.

Un aspect important de notre projet a été de réfléchir à la structure des classes et à leurs dépendances respectives. Il était essentiel de concevoir une architecture efficace, claire et compréhensible, tout en respectant les principes de propreté du code.

2 Conclusion

Ce projet a été une opportunité unique pour explorer les bases du langage C++, notamment à travers l'utilisation des concepts fondamentaux comme les pointeurs et le polymorphisme. Ces outils ont permis de construire un programme modulaire, efficace, et extensible pour traiter des problématiques financières complexes telles que le pricing et les stratégies de couverture.

L'implémentation des stratégies Delta Hedging et Delta-Gamma Hedging a également permis d'appréhender des notions avancées de gestion des risques en finance, tout en consolidant nos compétences en programmation et en modélisation mathématique. Le programme offre une visualisation claire des performances des stratégies et un cadre flexible pour des extensions futures, comme l'intégration d'autres modèles financiers ou l'optimisation des calculs Monte-Carlo.

Annexe : Pricing des options américaines et stratégies de couverture

Pricing des options américaines avec un arbre binomial

Pour les options américaines, qui permettent un exercice anticipé à tout moment avant l'échéance, nous avons utilisé un modèle d'arbre binomial pour le pricing. Cette méthode consiste à discrétiser l'évolution du prix du sous-jacent sur une période donnée en divisant celle-ci en plusieurs pas de temps (steps). À chaque pas, le prix de l'actif peut augmenter (multiplié par un facteur u) ou diminuer (multiplié par un facteur d).

Dans notre implémentation, nous avons construit un arbre pour représenter les prix de l'actif sous-jacent à chaque étape, en calculant les payoffs de l'option au dernier nœud (à l'échéance). Ensuite, nous avons remonté l'arbre en calculant la valeur présente de l'option à chaque nœud intermédiaire. Pour chaque nœud, nous avons comparé la valeur de l'option si elle est maintenue (*hold*) à celle si elle est exercée immédiatement (*exercise*). La valeur maximale entre ces deux alternatives est choisie pour représenter la valeur de l'option au nœud.

Matériellement, le modèle repose sur trois éléments principaux :

- Le facteur $u = e^{\sigma\sqrt{\Delta t}}$ pour les augmentations de prix et $d = \frac{1}{u}$ pour les diminutions.
- La probabilité de montée $p = \frac{e^{r\Delta t} - d}{u - d}$, où r est le taux sans risque.
- Une actualisation pour tenir compte de la valeur temporelle de l'argent, avec un facteur d'actualisation $e^{-r\Delta t}$.

Cette méthode est particulièrement utile pour le pricing des options américaines, car elle permet de capturer la possibilité d'exercice anticipé, ce qui est une caractéristique fondamentale de ces options. Notre implémentation calcule le payoff final de l'option, remonte l'arbre binomial pour évaluer les valeurs intermédiaires et fournit ainsi un prix précis et cohérent avec la nature américaine des options.

Stratégie Delta-Gamma Hedging et choix de la deuxième option

Dans notre implémentation de la stratégie *Delta-Gamma Hedging*, nous avons utilisé une approche avancée qui consiste à gérer simultanément la sensibilité du portefeuille aux variations du prix du sous-jacent (Delta) et à la convexité de ces variations (Gamma). Contrairement au *Delta Hedging* qui se contente de neutraliser l'effet de Delta, le *Delta-Gamma Hedging* nécessite une deuxième option pour ajuster Gamma.

La première option utilisée (`option1`) est celle pour laquelle on souhaite couvrir le portefeuille. La deuxième option (`option2`) est choisie pour son Gamma, qui est utilisé pour compenser le Gamma de la première option. La position dans cette deuxième option est déterminée par la formule suivante :

$$\text{gamma\_position} = -\frac{\Gamma_1}{\Gamma_2}$$

où Γ_1 et Γ_2 représentent les gammas des deux options. Ensuite, un Delta combiné est calculé en tenant compte à la fois du Delta de la première option et de celui de la deuxième, pondéré par la position Gamma :

$$\text{combined\_delta} = \Delta_1 + \text{gamma\_position} \cdot \Delta_2$$

Dans notre implémentation, la deuxième option (`option2`) est généralement choisie comme une option Put, car son Gamma est souvent plus élevé que celui d'une option Call correspondante, rendant la couverture plus efficace. Cette flexibilité dans le choix des options est permise par l'utilisation de pointeurs et du polymorphisme. Les pointeurs constants (`const Option*`) permettent à la fonction de manipuler des options de type Call ou Put sans modification supplémentaire du code.

Le programme simule ensuite les trajectoires du prix du sous-jacent, ajuste les positions dans les options et enregistre les résultats (valeur du portefeuille, Delta combiné, position Gamma) dans un fichier CSV. Ces données peuvent être analysées et visualisées pour évaluer la performance de la stratégie.