Institut Polytechnique de Paris

École Nationale de la Statistique et de l'Administration Économique

Projet language C++

PRICER

Elaboré par :

MAGHREBI Mohamed Aziz

DERBEL Malek

MOKLINE Mohamed Iyed

2ème Année ENSAE

Année universitaire : 2025/2026

Table des matières

Introduction

La valorisation et la couverture des produits dérivés constituent des problématiques centrales en finance quantitative. Comprendre les mécanismes qui permettent de déterminer le prix d'un produit financier et de maîtriser les risques qui lui sont associés représente ainsi un exercice fondamental. C'est dans cette optique que nous avons choisi de traiter le sujet intitulé *« Pricer »*.

Le présent projet consiste à développer un pricer d'options financières en langage C++, capable de valoriser différents types d'options (européennes et exotiques) et de calculer leurs sensibilités (*grecques*), en combinant le modèle analytique de Black–Scholes–Merton avec la méthode de montecarlo.

Le modèle de Black–Scholes–Merton reste une référence fondamentale pour l'évaluation des options européennes, en reposant sur des hypothèses classiques (marché sans arbitrage, taux d'intérêt et volatilité constants, dynamique log-normale du sous-jacent) qui permettent d'obtenir des formules analytiques explicites pour les prix et les grecques. Toutefois, ces formules fermées deviennent inapplicables dès que l'on considère des options plus complexes (asiatiques, digitales, lookback) ou lorsque certaines hypothèses du modèle sont relâchées. Dans ces cas, les méthodes de simulation, notamment la méthode de Monte-Carlo, offrent une approche puissante et flexible pour estimer à la fois les prix et les sensibilités des options, tout en fournissant des intervalles de confiance statistiques.

L'objectif principal de ce projet consiste à :

— Implémenter le modèle analytique de Black–Scholes–Merton pour le calcul exact des prix et des grecques d'options européennes.

— Développer un pricer Monte-Carlo polyvalent capable de simuler le prix et les grecques de divers types d'options (européennes, asiatiques, digitales et lookback).

— Calculer et illustrer la stratégie de réplication dynamique de l'option via le *delta-hedging*, permettant de couvrir les risques en ajustant continûment la position dans le sous-jacent.

— Proposer une interface interactive permettant la saisie des paramètres de marché et des caractéristiques des options.

Étude théorique

Cadre général du pricing des options

Une option financière est un produit dérivé dont la valeur dépend de l'évolution d'un actif sous-jacent (action, indice, taux, etc.). On distingue principalement :

— les options européennes, exerçables uniquement à maturité,

— les options exotiques, dont le payoff dépend de toute la trajectoire du sous-jacent.

L'objectif d'un pricer est de :

— Déterminer la valeur théorique d'une option,

— Calculer ses sensibilités (Grecs),

— Simuler une stratégie de réplication (hedging).

Dans ce projet, deux approches sont utilisées :

— le modèle analytique de Black–Scholes–Merton,

— la méthode de Monte-Carlo.

1.1 Modèle de Black–Scholes–Merton

1.1.1 Hypothèses du modèle

Le modèle de Black–Scholes–Merton repose sur les hypothèses suivantes :

— Le marché est sans arbitrage.

— Le taux d'intérêt sans risque r est constant.

— La volatilité σ est constante.

— Il n'y a ni dividendes, ni coûts de transaction.

— Le sous-jacent suit un mouvement brownien géométrique.

1.1.2 Dynamique du sous-jacent

Sous la mesure risque-neutre, le prix du sous-jacent S_t suit une équation différentielle stochastique de type Black–Scholes–Merton :

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t, \tag{1.1.1}$$

où r est le taux d'intérêt sans risque, σ la volatilité constante et W_t un mouvement brownien standard.

La solution explicite de l'équation (1.1.1) est donnée par :

$$S_T = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}Z\right), \quad Z \sim \mathcal{N}(0,1), \tag{1.1.2}$$

expression qui est utilisée pour la génération des trajectoires simulées dans les méthodes de Monte-Carlo.

1.1.3 Formules de Black–Scholes

Pour une option européenne de maturité T et de strike K, on définit :

$$d_1 = \frac{\ln(S_0/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T} \tag{1.1.3}$$

Prix d'un call européen

$$C = S_0\Phi(d_1) - Ke^{-rT}\Phi(d_2) \tag{1.1.4}$$

Prix d'un put européen

$$P = Ke^{-rT}\Phi(-d_2) - S_0\Phi(-d_1) \tag{1.1.5}$$

où $\Phi(\cdot)$ désigne la fonction de répartition de la loi normale standard.

1.1.4 Grecs analytiques

Les sensibilités du prix par rapport aux paramètres du marché sont données par :

Delta

$$\Delta_{\text{call}} = \Phi(d_1), \quad \Delta_{\text{put}} = \Phi(d_1) - 1 \tag{1.1.6}$$

Gamma

$$\Gamma = \frac{\phi(d_1)}{S_0\sigma\sqrt{T}} \tag{1.1.7}$$

Vega

$$\text{Vega} = S_0\sqrt{T}\,\phi(d_1) \tag{1.1.8}$$

où $\phi(\cdot)$ est la densité de la loi normale.

Ces expressions sont implémentées directement dans le code pour validation des résultats Monte-Carlo.

1.2 Méthode de Monte-Carlo

1.2.1 Principe général

Sous l'hypothèse d'absence d'arbitrage, le prix d'une option est donné par l'espérance actualisée de son payoff sous la mesure risque-neutre :

$$V_0 = e^{-rT}\,\mathbb{E}^{\mathbb{Q}}[\text{Payoff}], \tag{1.2.1}$$

conformément au théorème fondamental de l'évaluation des actifs.

Cette espérance est approchée numériquement par la méthode de Monte-Carlo :

$$V_0 \approx e^{-rT}\,\frac{1}{N}\sum_{i=1}^{N}\text{Payoff}^{(i)}, \tag{1.2.2}$$

où chaque payoff est calculé à partir d'une trajectoire simulée du sous-jacent.

1.2.2 Discrétisation temporelle

Pour les options dépendant du chemin, la dynamique du sous-jacent (1.1.1) est discrétisée sur n pas de temps :

$$S_{t+\Delta t} = S_t \exp\left(\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}Z\right), \tag{1.2.3}$$

avec $Z \sim \mathcal{N}(0,1)$ indépendant à chaque pas.

1.2.3 Calcul des Grecs par différences finies

Les sensibilités sont estimées numériquement :

Delta

$$\Delta \approx \frac{V(S_0+\varepsilon) - V(S_0-\varepsilon)}{2\varepsilon}, \tag{1.2.4}$$

Gamma

$$\Gamma \approx \frac{V(S_0 + \varepsilon) - 2V(S_0) + V(S_0 - \varepsilon)}{\varepsilon^2}, \tag{1.2.5}$$

Vega

$$\text{Vega} \approx \frac{V(\sigma + \varepsilon) - V(\sigma)}{\varepsilon}. \tag{1.2.6}$$

1.3 Options européennes

Les options européennes sont les options financières les plus classiques. Leur payoff dépend uniquement de la valeur du sous-jacent à maturité T, notée S_T, et ne dépend pas de l'évolution du prix avant cette date. Dans le cadre de ce projet, les options européennes servent de référence avant l'introduction des options exotiques dépendantes du chemin.

1.3.1 Option européenne Call

Une option européenne de type call donne à son détenteur le droit, mais non l'obligation, d'acheter le sous-jacent à la date de maturité T au prix d'exercice K.

Son payoff est donné par :

$$\text{Payoff}_{\text{Call Européen}} = \max(S_T - K, 0) \tag{1.3.1}$$

1.3.2 Option européenne Put

Une option européenne de type put donne à son détenteur le droit, mais non l'obligation, de vendre le sous-jacent à la date de maturité T au prix d'exercice K.

Son payoff est défini par :

$$\text{Payoff}_{\text{Put Européen}} = \max(K - S_T, 0) \tag{1.3.2}$$

1.4 Options exotiques considérées

Dans cette section, nous présentons les principales options exotiques étudiées. Contrairement aux options européennes standards, le payoff de ces options dépend de l'ensemble de la trajectoire du sous-jacent et non uniquement de sa valeur à maturité.

1.4.1 Options asiatiques

Les options asiatiques sont des options dont le payoff dépend de la moyenne arithmétique des prix du sous-jacent observés à différentes dates $t_1, t_2, \ldots, t_n$ sur l'intervalle $[0, T]$.

La moyenne arithmétique est définie par :

$$\bar{S} = \frac{1}{n} \sum_{i=1}^{n} S_{t_i} \tag{1.4.1}$$

Les payoffs considérés sont :

$$\text{Payoff}_{\text{Call Asiatique}} = \max(\bar{S} - K, 0) \tag{1.4.2}$$

$$\text{Payoff}_{\text{Put Asiatique}} = \max(K - \bar{S}, 0) \tag{1.4.3}$$

1.4.2 Options digitales

Les options digitales, également appelées options binaires, offrent un paiement fixe si une condition est satisfaite à maturité, et nul sinon. Le payoff dépend uniquement de la valeur finale du sous-jacent S_T.

Pour une option digitale de type call :

$$\text{Payoff}_{\text{Call Digital}} = \begin{cases} 1 & \text{si } S_T > K, \\ 0 & \text{sinon.} \end{cases} \tag{1.4.4}$$

Pour une option digitale de type put :

$$\text{Payoff}_{\text{Put Digital}} = \begin{cases} 1 & \text{si } S_T < K, \\ 0 & \text{sinon.} \end{cases} \tag{1.4.5}$$

1.4.3 Options lookback

Les options lookback sont des options dépendantes du chemin, dont le payoff repose sur le maximum ou le minimum atteint par le sous-jacent sur toute la durée de vie du contrat.

Pour une option lookback à strike fixe de type call :

$$\text{Payoff}_{\text{Call Lookback}} = \max\left(\max_{t \in [0,T]} S_t - K, 0 \right) \tag{1.4.6}$$

Pour une option lookback à strike fixe de type put :

$$\text{Payoff}_{\text{Put Lookback}} = \max\left(K - \min_{t \in [0,T]} S_t, 0 \right) \tag{1.4.7}$$

1.5 Stratégie de réplication et couverture

Dans le cadre du modèle de Black–Scholes–Merton, le marché est supposé complet et sans arbitrage. Il est alors possible de répliquer parfaitement le payoff d'une option européenne à l'aide d'un portefeuille auto-finançant composé de :

— Une position dynamique sur le sous-jacent S_t.

— Un compte bancaire rémunéré au taux sans risque r.

La valeur du portefeuille de réplication à l'instant t est donnée par :

$$\Pi_t = \Delta_t S_t + B_t \tag{1.5.1}$$

où S_t représente le prix du sous-jacent, Δ_t le nombre d'actions détenues et B_t le montant placé (ou emprunté) sur le compte bancaire.

1.5.1 Principe du delta-hedging

La stratégie de *delta-hedging* consiste à ajuster le portefeuille à chaque instant selon :

$$\Delta_t = \frac{\partial V}{\partial S}(S_t, t) \tag{1.5.2}$$

où $V(S_t, t)$ est le prix de l'option obtenu à partir de la formule de Black–Scholes. Cette approche permet de neutraliser localement l'exposition du portefeuille aux variations infinitésimales du sous-jacent.

Dans un cadre théorique idéal, lorsque les ajustements de Δ_t sont continus, le portefeuille de réplication vérifie :

$$\Pi_T = \text{Payoff de l'option} \tag{1.5.3}$$

ce qui implique une couverture parfaite et un P&L nul à maturité.

1.5.2 Discrétisation temporelle

Pour l'étude théorique, on considère un découpage du temps jusqu'à maturité T en N intervalles :

$$\Delta t = \frac{T}{N} \tag{1.5.4}$$

Ce découpage permet d'analyser l'impact du choix d'un pas de temps sur la précision de la réplication et de quantifier l'erreur théorique due à une couverture non continue.

1.5.3 Initialisation du portefeuille

Au temps initial $t = 0$, le portefeuille est construit pour répliquer exactement le prix de l'option :

$$\Pi_0 = V_0 = \Delta_0 S_0 + B_0 \tag{1.5.5}$$

d'où :

$$B_0 = V_0 - \Delta_0 S_0 \tag{1.5.6}$$

Les valeurs initiales V_0 et Δ_0 sont déterminées à l'aide des formules analytiques de Black–Scholes.

1.5.4 Bilan à maturité et erreur de couverture

À la maturité T, la valeur théorique du portefeuille est :

$$\Pi_T = \Delta_T S_T + B_T \tag{1.5.7}$$

L'erreur de couverture théorique (ou P&L de hedging) s'écrit :

$$\text{Erreur} = \Pi_T - \text{Payoff} \tag{1.5.8}$$

Dans le cadre théorique idéal, lorsque la couverture est continue et les hypothèses du modèle respectées, cette erreur est nulle.

Cette étude illustre les fondements mathématiques de la réplication parfaite et de la stratégie de delta-hedging dans le modèle de Black–Scholes, sans tenir compte des contraintes pratiques de marché.

Conclusion

Cette étude théorique met en évidence la complémentarité entre le modèle analytique de Black–Scholes–Merton et la méthode de Monte-Carlo. Le modèle analytique permet des calculs rapides et exacts pour les options européennes, tandis que la simulation de Monte-Carlo offre une grande flexibilité pour le pricing et l'analyse des options exotiques ainsi que pour l'étude des stratégies de couverture.

Conception

Introduction

Ce chapitre présente la conception logicielle du pricer développé en langage `C++`. L'objectif est de décrire l'architecture du programme, les structures de données mises en place ainsi que les principaux choix de conception adoptés afin d'assurer :

— la modularité du code,

— la réutilisabilité des composants,

— l'extensibilité vers de nouveaux types d'options ou de modèles.

Le projet repose sur une approche orientée objet, où chaque concept financier (option, modèle, méthode de pricing, couverture) est représenté par une classe dédiée. La première brique fondamentale de cette architecture est la représentation des options financières.

2.1 Classe abstraite `Option`

La classe `Option` constitue la classe mère de toutes les options implémentées dans le projet. Elle permet de regrouper les attributs communs à l'ensemble des produits dérivés et de définir une interface générique pour le calcul du payoff.

Cette classe est déclarée abstraite grâce à la méthode virtuelle pure `payoff`, ce qui impose à chaque classe dérivée de fournir sa propre implémentation du calcul du payoff conformément aux expressions mathématiques présentées précédemment.

Attributs

— `maturity` : maturité de l'option,

— `strike` : prix d'exercice,

— `name` : nom descriptif de l'option.

Méthodes

- `getMaturity() const`,
- `getStrike() const`,
- `getName() const`,
- `payoff(const std::vector<double>& path) const` : méthode virtuelle pure calculant le payoff à partir d'une trajectoire simulée du sous-jacent.

Le vecteur `path` représente la trajectoire discrétisée du sous-jacent, telle que définie dans la section consacrée à la méthode de Monte-Carlo.

2.1.1 Options européennes

Les options européennes sont implémentées à travers les classes `CallEuropeen` et `PutEuropeen`, toutes deux dérivées de la classe `Option`.

Conformément aux équations de payoff des options européennes présentées pour le call avec l'équation 1.3.1 et pour le put avec l'équation 1.3.2, le calcul repose uniquement sur la valeur finale du sous-jacent, cette valeur correspond au dernier élément du vecteur `path`, accessible via la méthode `path.back()`.

2.1.2 Options asiatiques

Les options asiatiques sont implémentées par les classes `CallAsiatique` et `PutAsiatique`. Leur payoff est calculé conformément aux expressions mathématiques définies dans la section dédiée aux options asiatiques pour le call avec l'équation 1.4.2 et pour le put avec l'équation 1.4.3.

Le calcul repose sur la moyenne arithmétique des valeurs contenues dans le vecteur `path`, représentant les prix du sous-jacent aux différentes dates de discrétisation.

Cette moyenne est ensuite utilisée dans la méthode `payoff` de chaque classe dérivée pour évaluer le payoff.

2.1.3 Options digitales

Les options digitales sont modélisées par les classes `CallDigital` et `PutDigital`. Leur payoff, tel que défini dans la section consacrée aux options digitales, dépend uniquement de la valeur finale du sous-jacent pour le call avec l'équation 1.4.4 et pour le put avec l'équation 1.4.5.

Dans l'implémentation, la méthode `payoff` compare la valeur `path.back()` au prix d'exercice `strike` à l'aide d'une condition logique simple, reflétant directement la définition mathématique du produit.

2.1.4 Options lookback

Les options lookback sont implémentées par les classes `CallLookback` et `PutLookback`. Leur payoff dépend respectivement du maximum ou du minimum atteint par le sous-jacent sur l'ensemble de la trajectoire, conformément aux définitions théoriques introduites précédemment pour le call avec l'équation 1.4.6 et pour le put avec l'équation 1.4.7.

Dans le code C++, ces quantités sont déterminées à l'aide des fonctions standards `std::max_element` et `std::min_element` appliquées au vecteur `path`.

2.1.5 Polymorphisme et extensibilité

Grâce à l'utilisation d'une méthode virtuelle pure dans la classe `Option`, toutes les options peuvent être manipulées via des pointeurs ou références de type `Option`. Cette approche permet d'évaluer différents types d'options de manière polymorphique, sans modifier le code du pricer.

L'architecture adoptée facilite ainsi l'extension du programme, notamment par l'ajout de nouveaux produits dérivés ou de nouvelles méthodes de calcul, tout en préservant la cohérence entre la modélisation théorique et l'implémentation numérique.

2.2 Modélisation du marché : classe `BlackScholesModel`

La classe `BlackScholesModel` représente le modèle de marché utilisé pour le pricing et la couverture des options. Elle encapsule l'ensemble des paramètres financiers nécessaires ainsi que les méthodes permettant :

— le calcul analytique des prix et des sensibilités,

— la simulation de trajectoires du sous-jacent.

Cette classe constitue un élément central du pricer, car elle est utilisée à la fois par le module analytique de Black–Scholes et par la méthode de Monte-Carlo.

2.2.1 Attributs

La classe `BlackScholesModel` contient les paramètres suivants :

— `spot` (S_0) : prix initial du sous-jacent,

— `rate` (r) : taux d'intérêt sans risque supposé constant,

— `volatility` (σ) : volatilité constante du sous-jacent.

Ces paramètres sont fixés à l'initialisation de l'objet et sont accessibles uniquement via des méthodes `get`, garantissant l'encapsulation des données.

2.2.2 Fonctions statistiques

Deux fonctions privées sont implémentées afin de manipuler la loi normale standard :

— `normalCDF(x)` : fonction de répartition $\Phi(x) = \mathbb{P}(Z \leq x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}}\, dt$

— `normalPDF(x)` : densité de probabilité $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \quad x \in \mathbb{R}$.

Ces fonctions sont utilisées dans les formules analytiques de Black–Scholes pour le calcul des prix et des Grecs.

2.2.3 Formules fermées de Black–Scholes

La classe implémente les expressions analytiques du modèle de Black–Scholes–Merton pour les options européennes.

Prix La méthode `bsPrice` calcule le prix d'un call ou d'un put européen à l'aide des formules fermées, en fonction du strike K, de la maturité T et du type de l'option en faisant référence à l'équation 1.1.5 pour le put et l'équation 1.1.4 pour le call.

Delta La méthode `bsDelta` retourne la sensibilité du prix par rapport au prix du sous-jacent, utilisée notamment dans la stratégie de couverture dynamique en faisant référence à l'équation 1.1.6.

Gamma La méthode `bsGamma` mesure la convexité du prix de l'option par rapport au sous-jacent en faisant référence à l'équation 1.1.7.

Vega La méthode `bsVega` mesure la sensibilité du prix de l'option aux variations de la volatilité en faisant référence à l'équation 1.1.8.

Ces formules servent également de référence pour valider la précision des résultats obtenus par la méthode de Monte-Carlo.

2.2.4 Génération de trajectoires

La méthode `generatePath` permet de simuler une trajectoire du sous-jacent selon un mouvement brownien géométrique.

La dynamique discrétisée est donnée par l'équation 1.2.3 et la trajectoire est stockée dans un vecteur contenant l'ensemble des valeurs du sous-jacent, de l'instant initial jusqu'à la maturité. Cette approche permet le pricing d'options dépendant du chemin, telles que les options asiatiques ou lookback.

2.2.5 Rôle dans l'architecture du pricer

La classe `BlackScholesModel` est utilisée par :

— le module de pricing analytique pour les options européennes,

— le moteur de simulation Monte-Carlo,

— le simulateur de couverture dynamique (hedging).

Elle joue ainsi un rôle central dans l'architecture du pricer en assurant la cohérence entre les calculs analytiques, les simulations numériques et les stratégies de réplication.

2.3 Méthode de Monte-Carlo : classe `MonteCarlo`

La classe `MonteCarlo` implémente une méthode numérique de valorisation des options fondée sur la simulation de trajectoires du sous-jacent. Elle est particulièrement adaptée aux options pour lesquelles aucune formule analytique n'est disponible, notamment les options dépendantes du chemin.

D'un point de vue logiciel, cette classe est conçue comme une classe utilitaire : elle ne contient que des méthodes statiques et ne nécessite donc aucune instanciation. Cette approche simplifie son intégration dans le pricer et garantit une utilisation cohérente dans l'ensemble du programme.

2.3.1 Principe général

Le principe de valorisation repose sur l'approximation numérique de l'espérance du payoff sous la mesure risque-neutre. Dans l'implémentation, cette espérance est approchée par une moyenne empirique calculée à partir d'un grand nombre de trajectoires simulées par le modèle de Black–Scholes.

La génération des trajectoires est assurée par la méthode `generatePath` de la classe `BlackScholesModel`, tandis que l'évaluation du payoff repose sur le polymorphisme de la classe abstraite `Option` via la méthode virtuelle `payoff`.

2.3.2 Pricing par simulation

La méthode `price` constitue le cœur du moteur de Monte-Carlo. Son fonctionnement peut être résumé comme suit :

— Génération de trajectoires du sous-jacent à l'aide du modèle de Black–Scholes.

— Calcul du payoff associé à chaque trajectoire en appelant `option.payoff(path)`.

— Agrégation des payoffs simulés et actualisation au taux sans risque fourni par le modèle.

Le nombre de pas de temps utilisés dans chaque trajectoire est déterminé automatiquement à partir de la maturité de l'option, en considérant une base de 252 pas par an, correspondant au nombre moyen de jours de bourse. Un mécanisme de sécurité garantit qu'au moins un pas de temps est utilisé, même pour des maturités très courtes.

2.3.3 Calcul des Grecs par différences finies

En l'absence de formules fermées pour les options exotiques, les sensibilités sont estimées numériquement à l'aide de différences finies en choquant les paramètres par epsilon choisi selon l'ordre de grandeur de la variable qu'on choque.

Delta La méthode `delta` repose sur la création de deux modèles de Black–Scholes dont le prix initial du sous-jacent est perturbé positivement et négativement. Le delta est ensuite estimé à partir de la différence des prix obtenus par la méthode `price` appliquée à ces deux modèles en faisant référence à l'équation 1.2.4.

Gamma La méthode `gamma` utilise une approche similaire, en évaluant le prix de l'option pour un sous-jacent augmenté, diminué et inchangé. Cette méthode permet d'estimer la convexité du prix par rapport au sous-jacent à partir de trois valorisations successives en faisant référence à l'équation 1.2.5.

Vega La méthode `vega` mesure la sensibilité du prix à la volatilité. Elle est implémentée en comparant le prix de l'option obtenu avec une volatilité perturbée à celui obtenu avec la volatilité initiale du modèle en faisant référence à l'équation 1.2.6.

Les paramètres de perturbation sont passés en argument avec des valeurs par défaut, ce qui permet un compromis entre précision numérique et stabilité des estimations.

2.3.4 Avantages et limites

L'approche de Monte-Carlo présente plusieurs avantages :

— grande flexibilité pour la valorisation de produits complexes,

— indépendance vis-à-vis de l'existence de formules analytiques,

— architecture modulaire facilitant l'extension à de nouveaux produits.

Cependant, certaines limites doivent être soulignées :

— convergence lente nécessitant un grand nombre de simulations,

— coût de calcul important pour l'estimation des Grecs,

— présence de bruit numérique, en particulier pour les sensibilités de second ordre.

2.3.5 Rôle dans l'architecture du pricer

Dans l'architecture globale du pricer, la classe `MonteCarlo` joue un rôle central. Elle est utilisée comme moteur principal de valorisation pour les options exotiques, comme outil de validation numérique des formules de Black–Scholes pour les options standards, et comme base de calcul des Grecs nécessaires à l'analyse et à la gestion du risque.

Elle constitue ainsi un composant fondamental assurant la robustesse, la cohérence et l'extensibilité du pricer.

2.4 Simulation de réplication : classe `HedgingSimulator`

La classe `HedgingSimulator` implémente une stratégie de réplication de type *delta-neutre*, conformément aux principes théoriques exposés précédemment. Elle permet d'étudier numériquement la performance d'une couverture dynamique d'options européennes en simulant l'évolution du portefeuille constitué du sous-jacent S_t et du compte bancaire B_t tel que défini en équation 1.5.1.

2.4.1 Principe du delta-hedging

La simulation repose sur le calcul du *delta* Δ_t d'une option, défini par la relation théorique équation 1.5.2. À chaque pas de temps, le portefeuille est ajusté pour rester localement insensible aux variations du sous-jacent, ce qui traduit numériquement le concept de portefeuille auto-finançant équation 1.5.3.

2.4.2 Fonctionnement de la simulation

Le simulateur (la méthode `run`)suit les étapes suivantes :

1. **Initialisation du portefeuille** : les valeurs initiales V_0, Δ_0 et B_0 sont calculées à partir des formules de Black–Scholes, conformément aux équations équation 1.5.5 et équation 1.5.6.
2. **Simulation du sous-jacent** : les trajectoires de S_t sont générées selon le modèle de Black–Scholes, en discretisant le temps selon l'équation 1.2.3
3. **Mise à jour du compte bancaire** : à chaque pas de temps, B_t est ajusté pour maintenir le portefeuille auto-finançant :

$$B_{t+\Delta t} = B_t e^{r\Delta t} - (\Delta_{t+\Delta t} - \Delta_t) S_{t+\Delta t}.$$

4. **Répétition des étapes** jusqu'à maturité T, en recalculant Δ_t à chaque intervalle.
5. **Évaluation finale** : le portefeuille simulé Π_T est comparé au payoff théorique pour calculer l'erreur de couverture équation 1.5.8.

2.4.3 Objectifs et limites

Cette simulation permet de :

— évaluer numériquement la qualité de la couverture dynamique d'une option,

— mesurer l'écart entre la valeur théorique V_T et le portefeuille répliqué (Π_T),

— illustrer l'impact de la discrétisation du temps sur la performance du delta-hedging.

Cependant, certaines limites doivent être notées :

— le recalcul discret du delta introduit un *risque de couverture*,

— le modèle suppose une volatilité constante et ignore les coûts de transaction,

— la simulation repose sur les hypothèses théoriques de Black–Scholes, qui peuvent différer des conditions réelles de marché.

2.4.4 Rôle dans l'architecture du pricer

`HedgingSimulator` constitue un outil, permettant :

— d'étudier l'efficacité des stratégies de couverture et les risques résiduels,

— d'illustrer concrètement le lien entre la théorie du modèle et sa mise en œuvre numérique.

Conclusion

Ce chapitre a présenté la conception orientée objet du pricer en `C++`, en mettant en évidence une architecture modulaire et extensible. La structuration autour des classes `Option`, `BlackScholesModel`, `MonteCarlo` et `HedgingSimulator` assure une cohérence entre théorie financière et implémentation numérique, tout en facilitant l'extension du pricer à de nouveaux produits financiers.

Compilation et interface utilisateur

Introduction

Le fichier `main.cpp` constitue le point d'entrée du pricer d'options développé dans ce projet. Il assure l'interaction avec l'utilisateur, la configuration du marché financier, la création dynamique des options ainsi que l'appel aux différents modules de calcul (pricing, calcul des grecs et réplication dynamique).

L'architecture du programme repose sur une séparation claire entre :

— l'interface utilisateur (menus, saisies, affichages),

— le modèle de marché (`BlackScholesModel`),

— les produits financiers (hiérarchie de la classe abstraite `Option`),

— les moteurs de calcul (Monte Carlo et couverture dynamique).

3.1 Initialisation et saisie sécurisée des paramètres de marché

Au lancement du programme, l'utilisateur est invité à saisir les paramètres fondamentaux du modèle de Black–Scholes :

— le prix initial du sous-jacent S_0,

— le taux d'intérêt sans risque r,

— la volatilité σ.

La saisie est sécurisée grâce à la fonction utilitaire `double getIn(string txt)` qui vérifie la validité des entrées et empêche toute erreur de type (saisie non numérique). Ces paramètres permettent ensuite de construire dynamiquement un objet

```
BlackScholesModel model(S0, r, sigma);
```

Le nombre de simulations Monte Carlo est initialisé par défaut à $N = 50\,000$, garantissant un compromis entre précision et temps de calcul.

3.2 Menu interactif et boucle principale

Le cœur du programme repose sur une boucle :

```
while (choix != 0)
```

qui maintient l'application active tant que l'utilisateur ne choisit pas explicitement de quitter (choix `0`).

À chaque itération, un menu avancé est affiché, proposant les fonctionnalités suivantes :

- **Choix 1 :** pricing des options européennes (Call et Put) avec calcul complet des grecs,
- **Choix 2 :** pricing des options asiatiques à moyenne arithmétique,
- **Choix 3 :** pricing des options digitales (binaires),
- **Choix 4 :** pricing des options lookback à strike fixe,
- **Choix 5 :** simulation de réplication dynamique (delta-hedging),
- **Choix 6 :** modification des paramètres de marché (S_0, r, σ) et du nombre de simulations Monte Carlo,
- **Choix 0 :** sortie propre du programme.

Cette organisation claire du menu permet une navigation intuitive et offre la possibilité d'enchaîner plusieurs calculs successifs sans redémarrer l'application, tout en conservant une flexibilité maximale dans l'analyse des produits financiers.

3.3 Création dynamique des options et polymorphisme

Pour les choix correspondant au pricing (options 1 à 4), l'utilisateur saisit :

- le strike K,
- la maturité T,
- le type de payoff (Call ou Put).

La création des options repose sur le polymorphisme en C++. Un pointeur générique vers la classe abstraite `Option` est utilisé `Option* opt = nullptr;`

Selon le choix de l'utilisateur, ce pointeur est instancié dynamiquement avec la classe fille appropriée (`CallEuropeen`, `PutAsiatique`, etc.). Ce mécanisme permet

de traiter uniformément tous les produits financiers, indépendamment de leur nature exacte, illustrant ainsi le principe de liaison dynamique à l'exécution (*runtime polymorphism*).

3.4 Calcul du prix et des grecs

Une fois l'option créée, elle est transmise à la fonction :

```
afficherDetailsOption(Option* opt, BlackScholesModel& model, int N)
```

Cette fonction :

— calcule le prix par la méthode de Monte Carlo,

— estime les sensibilités principales (Delta, Gamma, Vega),

— affiche les résultats de manière structurée.

Pour les options européennes, une vérification supplémentaire est effectuée grâce à l'utilisation de `dynamic_cast` afin de comparer les résultats Monte Carlo avec les formules analytiques exactes du modèle de Black–Scholes (prix et grecs).

Enfin, la mémoire est correctement libérée par l'instruction `delete opt;` ce qui garantit une gestion sûre des ressources dynamiques.

3.5 Simulation de réplication dynamique

Le menu permet également de lancer une simulation de couverture dynamique (*delta-hedging*) pour une option européenne. Cette fonctionnalité est implémentée dans la classe `HedgingSimulator`

La méthode statique `run` :

— génère une trajectoire du sous-jacent selon la dynamique de Black–Scholes,

— ajuste dynamiquement la position en actif risqué,

— simule l'évolution du portefeuille de réplication,

— compare la valeur finale du portefeuille au payoff théorique.

Cette partie illustre concrètement le lien entre la théorie de la couverture continue et son implémentation numérique.

3.6 Mise à jour dynamique des paramètres

À tout moment, l'utilisateur peut modifier :

— le spot S_0,

— le taux r,

— la volatilité σ,

— le nombre de simulations N.

Le modèle de marché est alors reconstruit dynamiquement, sans interruption du programme, ce qui renforce la flexibilité et l'interactivité de l'application.

3.7 Gestion de l'interface et robustesse

Le programme intègre plusieurs mécanismes assurant sa robustesse :

— contrôle systématique des saisies utilisateur,

— gestion propre des allocations mémoire,

— affichage clair et structuré des résultats,

— possibilité de quitter le programme à tout moment.

3.8 Problèmes rencontrés et solutions adoptées

Le développement de ce pricer a soulevé plusieurs difficultés techniques et numériques, typiques des méthodes de valorisation par simulation.

Variance de la méthode de Monte-Carlo Les estimateurs Monte-Carlo présentent un bruit statistique, en particulier pour les options exotiques et pour le calcul des sensibilités. *Solution :* un nombre de simulations élevé est utilisé par défaut, et l'erreur statistique est quantifiée à l'aide de l'écart-type et de l'erreur standard.

Choix de la discrétisation temporelle Une discrétisation trop grossière dégrade la précision du pricing des options dépendantes du chemin. *Solution :* le temps est discrétisé automatiquement sur la base de 252 pas par an, avec une contrainte garantissant au moins un pas de temps.

Calcul numérique des Grecs L'estimation des Grecs par différences finies dépend du choix du paramètre de perturbation ε et peut introduire un compromis biais/variance. *Solution :* des chocs adaptés à l'ordre de grandeur des paramètres sont utilisés, et des différences centrées sont privilégiées pour Delta et Gamma.

Couverture dynamique en temps discret La stratégie de delta-hedging n'est pas parfaitement réplicative en temps discret, ce qui génère une erreur de couverture. *Solution :* cette erreur est explicitement simulée et analysée afin d'illustrer les limites pratiques de la couverture théorique continue.

Environnements de développement hétérogènes Les membres de l'équipe ont travaillé sur des systèmes d'exploitation différents, principalement **Windows** et **macOS**, tout en utilisant un environnement de travail commun basé sur l'éditeur **Visual Studio Code**. Cette diversité a nécessité une attention particulière à la portabilité du code et à la méthode de compilation.

Solution : afin de garantir un comportement identique du programme sur les différentes plateformes, l'équipe a adopté une approche uniforme reposant sur une **compilation en ligne de commande** avec le compilateur `g++`, exécutée depuis le terminal intégré de Visual Studio Code. La procédure comporte trois étapes principales :

1. **Positionnement dans le répertoire du projet :** se placer dans le dossier contenant tous les fichiers sources `.cpp` et `.h`, par exemple : `cd pricer`.
2. **Compilation :** compiler l'ensemble des fichiers sources avec la commande : `g++ *.cpp -o pricer` qui génère un exécutable (`pricer.exe` sous Windows ou `pricer` sous macOS).
3. **Exécution :** lancer le programme directement depuis le terminal : `.\pricer.exe` sur Windows ou `./pricer` sur macOS.

Cette démarche a permis d'assurer la cohérence du code source et le bon fonctionnement du pricer sur tous les environnements utilisés par l'équipe.

Synthèse

Cette interface utilisateur, combinée à une architecture modulaire, permet :

1. l'évaluation précise des options européennes via des formules analytiques,
2. le pricing des options exotiques par simulation Monte Carlo,
3. l'illustration pratique des stratégies de couverture dynamique.

L'organisation du code facilite l'extension future du pricer vers de nouveaux produits, de nouveaux grecs ou des stratégies de gestion du risque plus avancées.

Conclusion

Le projet présenté consistait à développer un pricer d'options financières en langage C++, combinant le modèle analytique de Black–Scholes et des simulations Monte Carlo pour le calcul des prix et des sensibilités (*grecs*) des options.

Ce travail a permis de mettre en œuvre plusieurs concepts clés de la finance quantitative et de la programmation :

— la modélisation du marché financier à l'aide du modèle de Black–Scholes,

— le calcul numérique des prix et des grecs via Monte Carlo pour différents types d'options (européennes, asiatiques, digitales et lookback),

— la simulation de réplication dynamique (*delta-hedging*) pour illustrer la couverture des risques,

— la gestion interactive et sécurisée des entrées utilisateurs et des paramètres de marché.

L'approche modulaire adoptée a permis de structurer le programme de manière claire et extensible, facilitant l'ajout de nouveaux types d'options, de méthodes de calcul ou de stratégies de couverture qui ont offert une illustration concrète de la gestion des risques et permis de visualiser les limites de la couverture dynamique en temps discret, matérialisées par l'erreur de réplication (Hedging Error).

En conclusion, ce projet a non seulement renforcé les compétences en programmation orientée objet, mais il a également permis de mieux comprendre la relation entre les modèles financiers théoriques et les simulations numériques. Il constitue ainsi une base solide pour des développements et des connaissances futurs dans le domaine de la finance.