Rapport projet final en C++

2ème année cycle d'ingénieur

Pricing d'options et stratégie de réplication dans le modèle de Black-Scholes-Merton

---

Charles BOUCHET
Ons JDEY
Arold TOUBERT

Travail encadré par
Mme. Roxana DUMITRESCU

Table des matières

1 Introduction

Les options représentent une large part des marchés financiers actuels. Pour rappel, une option donne le droit, mais pas l'obligation, de vendre ou d'acheter un actif sous-jacent sur une période donnée à un prix d'exercice fixé, peu importe l'évolution du prix du marché. Elles peuvent donc être utilisées dans de nombreux domaines, notamment pour la spéculation et la gestion de risque. C'est donc pour cela que leur pricing s'avère être crucial dans le monde de la finance.

Pour estimer le prix de ces options, les pricer, il existe plusieurs méthodes utilisées par les ingénieurs financiers. Nous avons ainsi décidé de nous concentrer sur deux d'entre elles et de les coder à l'aide du langage C++ pour notre projet. La première est la méthode de Black-Scholes-Merton, qui s'applique aux options européennes. La deuxième méthode est celle de Monte Carlo, qui permet de simuler un grand nombre de chemins possibles à l'aide de variables aléatoires centrées réduites pour reproduire les mouvements browniens des marchés financiers. Cependant, cette méthode ne fonctionne qu'avec un grand nombre de chemins simulés, car elle utilise la loi forte des grands nombres. La programmation informatique, plus particulièrement celle en C++, est donc utile pour faire ces nombreux calculs. Nous avons aussi décidé d'utiliser la méthode de Longstaff et Schwartz pour pricer les options américaines, en raison de la complexité du calcul du payoff des options américaines que l'on peut exercer avant maturité.

Dans ce rapport, nous explorerons d'abord les différentes classes qui composent notre programme, puis nous approfondirons les différentes fonctions qui les composent, avant d'aborder les difficultés rencontrées lors de ce projet.

2 Description générale du programme

2.1 Guide d'utilisation

Pour ce programme nous avons utilisé deux librairies : `Eigen` et `cmath`. La première permet de réaliser des calculs d'algèbre linéaire, la deuxième permet l'utilisation de différentes fonctions mathématiques usuelles. Il est nécessaire d'avoir ces deux librairies pour faire fonctionner le programme.

2.2 Classe Gaussian

La classe `Gaussian` permet de générer des nombres aléatoires en simulant une variable qui suit une loi normale centrée réduite. Pour se faire, on utilise la méthode Box-Muller. Nous utilisons une classe pour générer une variable aléatoire car cela permet de réutiliser plus facilement cette méthode mais aussi de rendre le plus lisible en encapsulant la génération de nombre aléatoire au même endroit.

2.3 Classe Norm

La classe Norm permet de travailler avec une variable aléatoire suivant la loi normale centrée réduite. Elle fournit des méthodes pour calculer la densité de probabilité (*pdf*) et la fonction de répartition cumulative (*cdf*). La classe est conçue pour encapsuler les calculs associés à la loi normale centrée réduite, ce qui facilite ainsi leur utilisation dans un cadre plus large.

2.4 Classe Underlying

La classe `Underlying` permet de définir l'actif sous-jacent d'une option. Voici une description des variables membres de la classe Underlying :

- `Spot` : Prix actuel de l'actif sous-jacent.
- `Volatility` : Volatilité de l'actif, mesure la variation du prix de l'actif dans le temps.
- `Drift` : La tendance moyenne à long terme du prix de l'actif (souvent associé au taux de croissance ou au taux d'intérêt).

2.5 Classe Option

La classe `Option` est une classe abstraite qui hérite de la classe Underlying et qui permet de définir différents types d'options. Voici une description des variables membres de la classe `Option` :

— `Strike` : prix d'exercice de l'option.
— `Maturity` : échéance de l'option (en année).
— `IsCall` : Booléen qui indique si l'option est un call (`True`) ou un put (`False`).
— `Name` : nom de l'option.
— `Underlying` : Pointeur vers un objet de classe `Underlying` permettant d'avoir accès à l'actif sous-jacent de l'option.

2.6 Classes European, American, Asian, Barrier, Lookback et Digital

Ces six classes permettent de prendre en compte les spécificités de chaque type d'option (en particulier leur payoff). Il est préférable d'avoir une classe différente pour chaque type d'option pour avoir un code plus simple à comprendre et utiliser. Cela permet aussi d'ajouter plus facilement un nouveau type d'option.

2.7 Classe BlackScholesPricer

La classe `BlackScholesPricer` a pour objectif d'évaluer le prix d'une option en utilisant le modèle de Black-Scholes, basé sur des hypothèses telles qu'un marché sans friction, une volatilité constante, et un taux sans risque fixe. Ce modèle permet un calcul précis du prix des options européennes, en fonction des paramètres de l'option et de l'actif sous-jacent. Voici une description des variables de la classe `BlackScholesPricer` :

— `RiskFreeRate` : le taux sans risque, utilisé pour actualiser les prix futurs des payoffs et pour les calculs dans la formule de Black-Scholes.

2.8 Classe MonteCarloPricer

La classe `MonteCarloPricer` hérite des classes `Underlying` et `Option`. Elle a pour objectif d'évaluer le prix d'une option à l'aide de multiple simulation de nombreux chemins potentiels pour l'actif sous-jacent. On calcule ensuite le payoff moyen de l'option, que l'on actualise avec le taux sans risque. Voici une description des variables de la classe `MonteCarloPricer` :

— `NumPaths` : nombre de chemins de simulation générés par la méthode de Monte Carlo. Plus `NumPaths` est grand, plus l'estimation du payoff sera précise.
— `NumSteps` : nombre de pas dans le temps dans un price path, représente la fréquence à laquelle le prix de l'actif sous-jacent est mis à jour.

— `RiskFreeRate` : le taux sans risque, utilisé pour actualiser la valeur des payoffs.

2.9 Classe Replication

La classe `Replication` hérite de la classe `Option`, elle permet de mettre en place une stratégie de réplication à l'aide d'un portefeuille. On y considère deux types d'actifs : un actif sous-jacent et des instruments sans risque (généralement des obligations ou des bons du trésor). Voici une description des variables de la classe `Replication` :

— `UnderlyingAmount` : Quantité de l'actif sous-jacent détenue dans le portefeuille de réplication.

— `BondAmount` : Quantité d'instrument sans risque détenue dans le portefeuille.

— `PortfolioValue` : Valeur totale du portefeuille de réplication à l'instant t. C'est la somme de la valeur des instruments sans risque et des positions sur l'actif sous-jacent.

— `Option& option` : Référence à l'objet de classe `Option`, permettant d'accéder à ses caractéristiques pour calculer son payoff et effectuer la stratégie de réplication.

3 Description générale du programme

3.1 Gaussian

A l'aide du constructeur de la classe Gaussian on initialise d'abord deux objets. Le premier generator qui est un générateur de nombre aléatoires initialisé par une graine (seed). Le deuxième, uniformDist, est une distribution uniforme que l'on paramètre ici entre 0 et 1. On se sert ensuite de la méthode de Box-Muller pour générer le résultat d'une variable aléatoire centrée réduite à l'aide de la formule de Box Muller, en utilisant deux variables uniformes U_1 et U_2 entre 0 et 1 :

$$Z_0 = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$$

3.2 Underlying, Option, type spécifique d'option

La classe `Underlying` contient trois types principaux de fonctions.

Premièrement, le constructeur permet d'initialiser les variables membres de la classe lors de la création d'un objet, afin qu'elles disposent de valeurs initiales cohérentes. Deuxièmement, les getters permettent un accès en lecture aux variables membres privées, pour obtenir leurs valeurs tout en respectant le principe d'encapsulation. Pour finir, les setters permettent de modifier les valeurs des variables membres, offrant ainsi un moyen contrôlé d'ajuster les propriétés de l'objet, ce qui sera particulièrement utile lors de l'implémentation de la stratégie de réplication.

La classe `Option` reprend la même structure mais ajoute en plus une nouvelle méthode nommée **get_intrinsicValue**, qui permet de calculer la valeur intrinsèque de l'option, c'est-à-dire, la valeur qu'aurait l'option si elle était exercée dans l'immédiat. Les getters permettent d'obtenir les variables membres de l'objet et notamment d'avoir accès à l'underlying. Cependant, ici les setters ne permettent que de modifier la maturité (`Maturity`) et le prix d'exercice de l'option (`Spot`).

Chaque classe d'option spécifique possède une méthode personnelle, nommée **get_payoff**, qui permet de calculer le payoff d'une option selon les spécificités de son type. Nous détaillerons les payoffs de chaque option en annexe.

3.3 BlackScholesPricer

La classe `BlackScholesPricer` implémente un modèle de pricing analytique basé sur la formule de Black-Scholes pour évaluer la valeur des options financières européennes (calls et puts). Elle comprend un constructeur pour initialiser le taux sans risque utilisé dans les calculs. La méthode principale **price** applique les formules de Black-Scholes pour calculer le prix des options en fonction de paramètres comme le prix spot, la volatilité, la maturité, et le prix d'exercice.

3.4 MonteCarloPricer

La classe `MonteCarloPricer` implémente une méthode de pricing par simulation Monte Carlo pour évaluer la valeur des options financières. Elle comprend un constructeur pour initialiser les paramètres essentiels comme le nombre de chemins simulés, le nombre de pas de temps, et le taux sans risque. La méthode principale **GeneratePricePaths** génère des chemins de prix pour un actif sous-jacent en utilisant le modèle de Black-Scholes-Merton, avec une simulation de variables gaussiennes pour modéliser les fluctuations aléatoires. La méthode **Price** calcule la valeur de l'option en distinguant les options américaines et les autres options : les premières s'appuient sur l'algorithme de Longstaff-Schwartz pour estimer la valeur optimale de l'exercice à chaque période, tandis que les secondes utilisent directement les payoffs simulés. Le programme inclut également des getters pour accéder aux paramètres du pricer.

Pour tester la véracité des résultats obtenus par le pricing utilisant la méthode de Monte Carlo nous avons décidé de comparer les résultats à ceux obtenus avec la méthode de pricing utilisant la formule du modèle de Black Scholes Merton pour les options européennes. Nous avons aussi vérifié empiriquement que le prix des actions asiatiques était inférieur au prix des actions européennes à paramètres égaux.

3.5 Replication

Les fonctions du programme Replication mettent en œuvre une stratégie de réplication pour les options financières, permettant de reproduire leur valeur à l'aide d'un portefeuille dynamique composé d'actifs risqués et sans risque. La classe principale calcule le **delta** de l'option, représentant la sensibilité de son prix au sous-jacent, et ajuste la position en obligations pour équilibrer le portefeuille. Cette fonction prend en compte si l'option est un Call ou un Put car la formule du delta change. La méthode **strategy** effec-

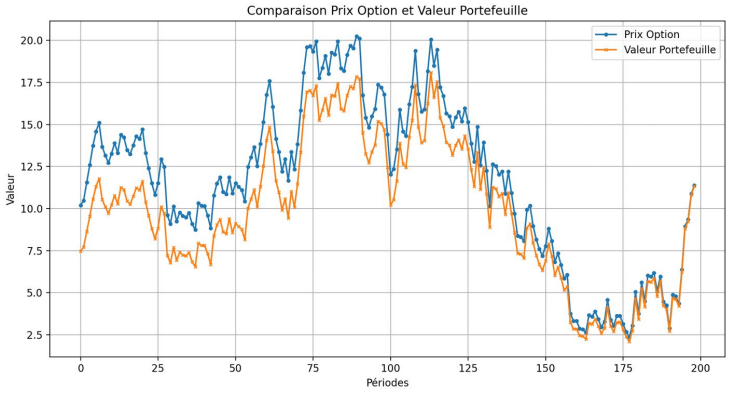

FIGURE 1 – Réplication d'une option européenne

tue une simulation pas à pas : elle utilise des chemins de prix générés par le pricer Monte Carlo et ajuste dynamiquement le portefeuille à chaque instant en fonction de la valeur actuelle de l'actif sous-jacent et de l'option. L'objectif est de vérifier que la valeur totale du portefeuille suit fidèlement celle de l'option tout au long de sa durée de vie, permettant ainsi d'évaluer l'efficacité de la réplication.

4 Retour sur les difficultés rencontrées

La première difficulté rencontrée lors du projet était liée à son architecture. En effet, c'était la première fois que nous avions un projet en C++ à rendre, et nous devions donc apprendre à structurer correctement notre code afin qu'il soit facile à comprendre, à utiliser et à réutiliser. Nous avons ainsi réfléchi au type de classe et de pointeur à utiliser pour faciliter l'ergonomie du projet, mais aussi sa modification. Cette réflexion nous a poussés à créer une première classe **Underlying**, dont hérite une deuxième, **Option**, puis à construire des classes spécifiques selon le type précis d'option (américaine, européenne, asiatique). Cela nous a permis d'ajouter davantage de types d'options au fur et à mesure du développement du programme, sans avoir à le modifier entièrement.
La deuxième difficulté rencontrée concerne la validation de nos différents codes. Pour les options européennes, nous avons pu comparer le pricing par la méthode de Monte Carlo à celui obtenu par la méthode de Black-Scholes-Merton. Cette comparaison n'était pas toujours possible pour les autres types d'options. De plus, le pricing des options américaines s'est avéré plus complexe, et nous avons dû utiliser la méthode de Longstaff-Schwartz avec une

régression linéaire.

5 Conclusion

Pour ce premier projet en C++, nous avions pour objectif de travailler sur le pricing des options et la stratégie de réplication, à l'aide du modèle de Black-Scholes-Merton. Nous avons exploré deux façons de calculer le prix des options : la première avec une méthode basée sur Black-Scholes et la deuxième des simulations Monte Carlo. Nous avons intégrées dans un programme ces deux approches bien structurées, il utilise des classes et la programmation orientée objet pour faciliter l'exécution des calculs.

L'une des forces de ce projet a justement été cette utilisation de la programmation orientée objet avec C++. Ce choix nous a permis de rendre le code à la fois plus organisé et plus robuste. Ce qui nous a vraiment convaincus, c'est la rapidité d'exécution de C++, qui est particulièrement utile pour les simulations Monte Carlo qu'on a utilisées pour le pricing des options asiatiques et américaines. Le programme a donc été conçu pour estimer efficacement les prix des options européennes, asiatiques et américaines, tout en ajustant le portefeuille au fur et à mesure de l'évolution des prix.

Au final, les résultats montrent que les modèles sont solides, notamment en comparant les simulations Monte Carlo avec les résultats du modèle analytique de Black-Scholes pour les options européennes. Nous avons rencontré quelques difficultés, comme la structuration du code ou la validation des résultats pour des options plus complexes, mais nous avons avancé étape par étape pour les surmonter.

En résumé, ce projet nous a permis d'avoir une meilleure compréhension du langage C++ dans un cadre financier et d'avoir des pistes pour améliorer le programme à l'avenir, comme en ajoutant des modèles plus complexes ou en rendant le code encore plus performant.

6 Annexe

6.1 Fondements théoriques des modèles utilisés

6.1.1 Modèle de Black-Scholes Merton

Dans le cadre de ce modèle, on dispose de la formule suivante :

$$\frac{S_{t+1} - S_t}{S_t} = \mu dt + \sigma\sqrt{dt}Z \tag{1}$$

Avec :
— μ représente la tendance
— σ représente la volatilité
— $Z \sim \mathcal{N}(0, 1)$
En posant $W_t \equiv \sqrt{dt}Z$ dans l'équation précédente, on obtient :

$$dS_t = \mu dt + \sigma dW_t \tag{2}$$

Et par la suite que :

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right) \tag{3}$$

Le prix d'une option vanille de maturité T et de payoff f à une date t est alors égal à :

$$e^{-r(T-t)} \times E(f(S(T-t))) \tag{4}$$

Avec :
— r : le taux d'intérêt sans risque
— f : le payoff de l'option à pricer
— $S(t)$: le prix de l'actif sous-jacent (une action par exemple)
Ainsi, on donne les expressions explicites des prix pour chacun des deux types d'options vanilles, à savoir les calls et les puts :

$$callPrice = S_t N(d_1) - Ke^{-r(T-t)}N(d_2) \tag{5}$$

$$putPrice = Ke^{-r(T-t)}N(-d_2) - S_t N(-d_1) \tag{6}$$

Avec :

$$d_1 = \frac{\log\left(\frac{S_k}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}} \tag{7}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} \tag{8}$$

$N(.)$ est la fonction de densité cumulative de la loi normale.

6.1.2 Modèle de Monte-Carlo

Dans les cas où on n'a pas accès directement à une formule explicite de pricing, ce qui est le cas de la majorité des options existantes en réalité, on peut procéder par la méthode de Monte-Carlo pour valoriser l'option en question.

Le principe de base consiste à générer des scénarios probables pour les prix des actifs sous-jacents au cours du temps, en utilisant des distributions statistiques appropriées pour représenter l'incertitude. Ensuite, pour chaque scénario, on peut calculer la valeur de l'option en question en utilisant une formule mathématique appropriée. La valeur de l'option peut alors être obtenue en moyennant les valeurs calculées pour chaque scénario. Autrement dit, cette méthode reposera sur l'estimation par la loi forte des grands nombres de l'espérance mathématique qui intervient dans la formule de pricing.

En effet, on a :

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W_t\right) \tag{9}$$

Donc :

$$e^{-r(T-t)} \times E(f(S(T-t))) = e^{-r(T-t)} \times E\left(f\left(S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W_t\right)\right)\right) \tag{10}$$

La loi forte des grands nombres intervient alors pour estimer par une moyenne empirique le terme :

$$E\left(f\left(S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W_t\right)\right)\right) \tag{11}$$

6.1.3 Construction d'un porte-feuille de réplication

Lorsqu'un agent effectue des transactions avec des produits financiers dérivés, il est impératif pour lui de se protéger contre le risque de fluctuations défavorables du marché. Pour cela, on construit un **portefeuille de réplication**. Une *stratégie de réplication* consiste à élaborer un portefeuille de couverture afin de se prémunir des risques de marché liés au sous-jacent sur lequel est construit le produit dérivé.

Le principe de la couverture consiste donc à investir la prime du dérivé dans un portefeuille de réplication contenant le sous-jacent S (actif risqué) et un actif non risqué (bon du trésor) noté B, évoluant au taux sans risque r. Notons $\Pi(t)$ la valeur du portefeuille à chaque instant, $a(t)$, $b(t)$ les proportions investies dans chacun des deux types d'actifs, de sorte que :

$$\forall t, \quad \Pi(t) = a(t)S(t) + b(t)B(t)$$

Pour construire notre stratégie de réplication, l'on admet l'hypothèse d'absence d'opportunité d'arbitrage. Sous cette hypothèse, pour se couvrir face au risque associé au sous-jacent (et donc au *payoff* de notre option), il faut construire un portefeuille produisant le même *payoff* que notre option à maturité, donc :

$$\Pi(T) = V(T)$$

En l'absence d'arbitrage, il s'ensuit que :

$$\forall t, \quad \Pi(t) = V(t)$$

La résolution du problème dans le cadre de Black-Scholes nous donne une solution de la forme :

$$a = \frac{\partial V}{\partial S} = \Delta = \Phi(d_1)$$

avec d_1 et d_2 définis comme dans le modèle de Black-Scholes, et on retrouve b car :

$$bB = V - aS$$

6.1.4 Méthode de Box-Muller

La méthode de Box-Muller est une technique utilisée pour générer des échantillons aléatoires suivant une distribution normale à partir de variables uniformes indépendantes. L'idée repose sur une transformation mathématique des coordonnées polaires. En partant de deux variables aléatoires U_1 et U_2 uniformément distribuées sur $[0, 1]$, on peut générer deux variables aléatoires Z_0 et Z_1 indépendantes, chacune suivant une loi normale standard (moyenne 0 et écart-type 1), à l'aide des formules suivantes :

$$Z_0 = \sqrt{-2 \ln U_1} \cos(2\pi U_2), \quad Z_1 = \sqrt{-2 \ln U_1} \sin(2\pi U_2).$$

6.1.5 Approximation de la fonction de répartition de la loi normale standard (CDF) à l'aide de la méthode d'Hastings

L'approximation de la fonction de répartition cumulative (CDF) de la loi normale standard, $\Phi(x)$, est réalisée à l'aide d'une méthode polynomiale rationnelle proposée par Hastings. Cette méthode est utilisée pour estimer la valeur de $\Phi(x)$ lorsque x est positif. La formule utilisée dans le programme suit cette approximation.

La fonction $\Phi(x)$ est approximée par :

$$\Phi(x) \approx 1 - \frac{1}{1 + A \cdot x} \cdot (C_1 + (C_2 + (C_3 + (C_4 + C_5 \cdot x) \cdot x) \cdot x) \cdot x)$$

Dans cette formule :

- $A = 0.2316419$ est un facteur d'échelle, - $C_1 = 0.319381530$, - $C_2 = -0.356563782$, - $C_3 = 1.781477937$, - $C_4 = -1.821255978$, - $C_5 = 1.330274429$ (coefficients trouvés dans [4]).

Le calcul commence en calculant une valeur intermédiaire k définie par :

$$k = \frac{1}{1 + A \cdot x}$$

Ensuite, un polynôme de degré 5 en x est évalué pour calculer k_{sum} :

$$k_{\text{sum}} = C_1 + k \cdot (C_2 + k \cdot (C_3 + k \cdot (C_4 + C_5 \cdot x)))$$

L'approximation finale de $\Phi(x)$ est alors :

$$\Phi(x) \approx 1 - \text{norm\_pdf}(x) \cdot k_{\text{sum}}$$

Où la fonction $\text{norm\_pdf}(x)$ donne la densité de probabilité de la loi normale standard, soit :

$$\text{norm\_pdf}(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

Cette approche permet de calculer la CDF de la loi normale standard de manière efficace et rapide, sans nécessiter l'intégration explicite de la fonction gaussienne, et elle est largement utilisée dans les applications financières et statistiques.

6.2 Formules utilisées

6.2.1 Option européenne

$$\text{Payoff européen} = \begin{cases} \max(S_T - K, 0), & \text{si c'est un call} \\ \max(K - S_T, 0), & \text{si c'est un put} \end{cases}$$

où S_T est le prix du sous-jacent à maturité et K est le prix d'exercice.

6.2.2 Option américaine

Payoff américain = Valeur intrinsèque au moment de l'exercice optimal

$$\text{Valeur intrinsèque} = \begin{cases} \max(S_t - K, 0), & \text{si c'est un call} \\ \max(K - S_t, 0), & \text{si c'est un put} \end{cases}$$

où S_t est le prix du sous-jacent à un instant t avant maturité.

6.2.3 Option asiatique

$$\text{Payoff asiatique} = \begin{cases} \max\left(\frac{1}{n}\sum_{i=1}^{n} S_{t_i} - K, 0\right), & \text{si c'est un call} \\ \max\left(K - \frac{1}{n}\sum_{i=1}^{n} S_{t_i}, 0\right), & \text{si c'est un put} \end{cases}$$

où $\frac{1}{n}\sum_{i=1}^{n} S_{t_i}$ est la moyenne des prix du sous-jacent sur la période de l'option.

6.2.4 Option barrière

$$\text{Payoff barrière} = \begin{cases} \text{Payoff standard}, & \text{si la barrière est atteinte (In) ou non atteinte (Out)} \\ 0, & \text{sinon} \end{cases}$$

où la barrière est une limite définie à l'avance, au-dessus ou en dessous de laquelle le payoff est affecté.

6.2.5 Option lookback

$$\text{Payoff Lookback} = \begin{cases} \max(S_{\max} - K, 0), & \text{pour un call} \\ \max(K - S_{\min}, 0), & \text{pour un put} \end{cases}$$

où $S_{\max}$ est le prix maximum atteint par le sous-jacent et $S_{\min}$ le prix minimum atteint.

6.2.6 Option digitale

$$\text{Payoff digitale (Call)} = \begin{cases} 1, & \text{si } S_T \geq K \\ 0, & \text{sinon} \end{cases}$$

où S_T est le prix de l'actif sous-jacent à l'échéance et K est le prix d'exercice. L'option digitale de type call prend la valeur 1 si le prix de l'actif sous-jacent à l'échéance est supérieur ou égal au prix d'exercice, et 0 sinon.

$$\text{Payoff digitale (Put)} = \begin{cases} 1, & \text{si } S_T \leq K \\ 0, & \text{sinon} \end{cases}$$

où S_T est le prix de l'actif sous-jacent à l'échéance et K est le prix d'exercice. L'option digitale de type put prend la valeur 1 si le prix de l'actif sous-jacent à l'échéance est inférieur ou égal au prix d'exercice, et 0 sinon.

6.2.7 Simulation de Monte Carlo

Génétation des chemins de prix

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

où $Z \sim \mathcal{N}(0,1)$ est une variable aléatoire normale, r est le taux sans risque, σ est la volatilité, et Δt est le pas de temps.

Calcul du prix de l'option

$$\text{Prix de l'option} = e^{-rT} \cdot \frac{1}{N} \sum_{i=1}^{N} \text{Payoff}_i$$

où N est le nombre de chemins simulés, r est le taux sans risque, T est la maturité, et Payoff_i est le payoff du chemin i.

6.2.8 Replication

Delta d'une option

Le delta d'une option est calculé en utilisant la formule de Black-Scholes pour une option européenne. Pour un call, le delta est donné par $N(d_1)$, et pour un put, il est $N(d_1) - 1$, où N est la fonction de répartition de la loi normale standard.

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

$$\Delta = \begin{cases} N(d_1), & \text{pour un call} \\ N(d_1) - 1, & \text{pour un put} \end{cases}$$

où S_0 est le prix de l'actif sous-jacent, K est le prix d'exercice, r est le taux sans risque, σ est la volatilité, et T est le temps restant avant la maturité.

Position en obligations
La position en obligations dans un portefeuille de réplication est calculée en fonction du prix de l'option, du delta et du prix de l'actif sous-jacent.

$$\text{Position en obligations} = \text{Prix de l'option} - \Delta \times S_0$$

où le prix de l'option est calculé à chaque étape et Δ est le delta calculé précédemment, et S_0 est le prix de l'actif sous-jacent.

Actualisation des obligations
La valeur de la position en obligations à une période future est actualisée en utilisant le taux sans risque r et le temps restant jusqu'à la maturité $T - t$.

$$\text{Valeur des obligations à l'étape suivante} = \text{Position en obligations} \times \exp(r \times (T - t))$$

où exp est la fonction exponentielle et $(T - t)$ est le temps restant jusqu'à la maturité.

Références

[1] The Eigen Project. *Eigen : A C++ Template Library for Linear Algebra.* https://eigen.tuxfamily.org/, Accessed : 2024-12-29.

[2] Black, F., Scholes, M. (1973). *The Pricing of Options and Corporate Liabilities.* Journal of Political Economy, 81(3), 637-654.

[3] Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering.* Springer-Verlag New York.

[4] Abramowitz and Stegun (1964), page 932. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* United States Department of Commerce, National Bureau of Standards.