Projet Personnel

Finance Quantitative — Modélisation et Implémentation Numérique

Pricing d'Options Européennes en C++
Méthodes de Monte Carlo et Différences Finies

Auteur : Adam Kerouredan

date : 25/06/2025

Université Paris Cité
Master 2 Modélisation Aléatoire, Finance et Data Science (M2MO)

Contents

Chapter 1

Introduction

1.1 Contexte

La valorisation des options européennes reste un enjeu central en finance quantitative, tant pour les activités de trading que pour la gestion des risques. Depuis l'introduction du modèle de Black-Scholes en 1973, les institutions financières ont continuellement affiné leurs outils de pricing afin d'allier précision, rapidité d'exécution et robustesse algorithmique dans des environnements de marché en constante évolution.

Dans ce contexte, ce projet personnel s'inscrit dans une démarche rigoureuse de modélisation et d'implémentation numérique. Il vise à coder en **C++** deux approches fondamentales du pricing d'options plain vanilla : la méthode de Monte Carlo, adaptée aux structures flexibles ou exotiques, et la méthode des différences finies, qui permet une résolution déterministe de l'équation de Black-Scholes.

L'objectif est double :

- D'un point de vue *quantitatif*, il s'agit d'approfondir les principes mathématiques sous-jacents à ces techniques, en particulier les équations aux dérivées partielles et les processus stochastiques.
- D'un point de vue *opérationnel*, l'enjeu est de produire une implémentation modulaire, performante et facilement intégrable dans un framework de pricing ou de backtesting.

Une attention particulière est portée à la structure du code, à la gestion de la convergence numérique, ainsi qu'à l'évaluation des sensibilités (Greeks), qui jouent un rôle clé en hedging et en calibration de modèles.

Ce rapport est organisé comme suit : une première partie rappelle les fondements du modèle de Black-Scholes et les propriétés des options européennes. La seconde partie détaille la mise en œuvre des deux méthodes numériques, en soulignant les choix algorithmiques et les critères de stabilité. Enfin, une comparaison des résultats est proposée, accompagnée d'une analyse de la précision, de la vitesse de convergence et de la qualité des estimations des Greeks.

Chapter 2

Modèle de Black-Scholes

2.1 Hypothèses Fondamentales

Le modèle de Black-Scholes constitue la pierre angulaire du pricing des options vanilles en environnement de marché efficient. Avant d'aborder la dérivation de l'équation aux dérivées partielles (EDP) associée, nous formalisons les hypothèses structurelles sous-jacentes au cadre de modélisation.

Hypothèse 1 — Aucun dividende distribué
L'actif sous-jacent ne verse pas de flux intermédiaires (dividendes ou coupons) durant la maturité du produit. Cette hypothèse neutralise les ajustements liés aux paiements anticipés et permet une modélisation plus épurée de la dynamique du sous-jacent.

Hypothèse 2 — Marché sans frictions
Le cadre est frictionless, impliquant :

- Absence totale de coûts de transaction ou de fiscalité,
- Négociabilité parfaite des actifs, y compris en quantités fractionnaires,
- Possibilité d'emprunter ou de prêter à un taux constant r, sans contraintes de liquidité ou de collatéral.

Ces conditions sont idéalisées mais constituent la base canonique pour toute construction analytique en finance quantitative.

Hypothèse 3 — Absence d'opportunité d'arbitrage
Un marché est dit *arbitrage-free* s'il n'existe aucune stratégie auto-finançable $\theta \in \mathcal{A}$ vérifiant :

1. $V_0^\theta = 0$ (pas de capital initial requis),
2. $V_T^\theta \geq 0$ *a.s.*,
3. $\mathbb{P}(V_T^\theta > 0) > 0$ (gain strictement positif possible).

Cette condition est un prérequis théorique indispensable : elle permet d'introduire une mesure de probabilité équivalente $\mathbb{Q}$ sous laquelle les prix actualisés deviennent des martingales. C'est le socle de la théorie de l'évaluation neutre au risque.

Hypothèse 4 — Dynamique du sous-jacent

Le prix de l'actif S_t suit un processus de diffusion log-normale gouverné par l'équation différentielle stochastique :

$$dS_t = \mu S_t \, dt + \sigma S_t \, dW_t,$$

où :

- $\mu \in \mathbb{R}$ désigne le drift (rendement moyen),
- $\sigma > 0$ est la volatilité constante de l'actif,
- W_t est un mouvement brownien standard sous $\mathbb{P}$.

Ce choix garantit une dynamique multiplicative, assurant la positivité de S_t et une distribution log-normale du prix à maturité.

Hypothèse 5 — Taux d'intérêt sans risque constant

Le taux r est constant, exogène et connu. Il permet d'actualiser les cash flows futurs de façon déterministe. Cette hypothèse simplifie la construction du portefeuille de réplication et la résolution de l'EDP associée.

Hypothèse 6 — Liquidité parfaite et trading en temps continu

Les opérateurs ont un accès permanent et sans contrainte de profondeur au marché, leur permettant d'ajuster dynamiquement leur exposition en temps continu. Cette hypothèse est cruciale pour établir la possibilité d'une couverture (delta hedging) parfaite.

Hypothèse 7 — Rationalité des agents et information complète

Les agents sont rationnels, parfaitement informés, et partagent une compréhension commune des paramètres de marché (r, σ, T). Aucune asymétrie d'information ni comportement irrationnel n'est supposé. Ce cadre assure la cohérence des anticipations et la robustesse du pricing.

2.2 Équation de Black-Scholes

L'équation de Black-Scholes constitue le cœur analytique du pricing des options européennes en environnement de marché sans arbitrage. Il s'agit d'une équation aux dérivées partielles (EDP) décrivant l'évolution du prix théorique d'un produit dérivé $F(t, S)$ en fonction du temps $t \in [0, T]$ et du niveau du sous-jacent $S > 0$.

Cette équation est dérivée à partir :

- de la dynamique stochastique du sous-jacent via un mouvement brownien géométrique,
- du lemme d'Itô appliqué à une fonction suffisamment régulière de (t, S_t),
- de la mise en place d'un portefeuille de couverture parfaitement répliquant,
- et du principe fondamental d'absence d'arbitrage qui impose que tout actif réplicable soit pricé comme l'espérance actualisée de ses cash-flows futurs sous une mesure risque-neutre.

Soit $F(t, S)$ le prix d'une option européenne à l'instant t, lorsque le sous-jacent vaut S. On distingue :

- $F(t, S) = C(t, S)$ pour un **call européen**,
- $F(t, S) = P(t, S)$ pour un **put européen**.

Dans un marché complet et arbitrage-free, cette fonction $F(t, S)$ doit satisfaire l'équation différentielle suivante :

$$\frac{\partial F}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 F}{\partial S^2} + rS\frac{\partial F}{\partial S} - rF = 0$$

où :

- r est le taux d'intérêt sans risque (constant),
- σ est la volatilité implicite (constante),
- $F(t, S)$ est la valeur du dérivé à temps t et niveau du sous-jacent S.

Cette EDP est rétrograde, dans le sens où elle est intégrée à rebours depuis la maturité T, à partir de la condition terminale donnée par le *payoff* de l'option :

$$F(T, S) = \begin{cases} \max(S - K, 0) & \text{pour un call,} \\ \max(K - S, 0) & \text{pour un put.} \end{cases}$$

Ce formalisme établit un pont fondamental entre les représentations probabilistes du prix des actifs financiers et la formulation déterministe via équations différentielles partielles. Dans la suite du rapport, nous étudierons d'abord les cas où une solution fermée est accessible analytiquement, puis nous mettrons en œuvre des méthodes numériques robustes pour traiter les situations hors modèle standard ou présentant des conditions de marché déformées.

2.2.1 Dynamique du sous-jacent

Dans le cadre probabiliste, on modélise l'évolution du prix de l'actif sous-jacent S_t par un processus de diffusion de type log-normal. Sous la mesure historique $\mathbb{P}$, la dynamique est donnée par :

$$dS_t = \mu S_t \, dt + \sigma S_t \, dB_t$$

où :

- $\mu \in \mathbb{R}$ est le taux de rendement instantané attendu (drift),
- $\sigma > 0$ est la volatilité du sous-jacent, supposée constante,
- B_t est un mouvement brownien standard défini sur un espace de probabilité filtré $(\Omega, \mathcal{F}, (\mathcal{F}_t), \mathbb{P})$.

Ce choix implique que $\log(S_t)$ est un processus gaussien, ce qui permet d'exploiter pleinement les outils de calcul stochastique en vue d'une dérivation formelle de l'équation de pricing.

2.2.2 Processus d'Itô et lemme d'Itô

On rappelle qu'un **processus d'Itô** s'écrit sous la forme :

$$X_t = X_0 + \int_0^t a(s)\, ds + \int_0^t b(s)\, dB_s,$$

où :

- $a(t)$ est le terme de dérive (*drift*),
- $b(t)$ est le terme de diffusion (*volatility coefficient*).

Ce type de processus capture à la fois l'évolution déterministe attendue et les chocs aléatoires du marché, modélisés par le bruit brownien.

Le **lemme d'Itô** est l'outil central pour dériver la dynamique d'une fonction $f(t, X_t)$, lorsque X_t est une diffusion. Si $f \in \mathcal{C}^{1,2}$, alors :

$$df(t, X_t) = \left(\frac{\partial f}{\partial t} + a(t, X_t) \frac{\partial f}{\partial x} + \frac{1}{2} b(t, X_t)^2 \frac{\partial^2 f}{\partial x^2} \right) dt + b(t, X_t) \frac{\partial f}{\partial x}\, dB_t$$

Ce résultat permet de calculer la dynamique d'un instrument dérivé exprimé comme fonction du sous-jacent, ce qui est le cas typique en pricing.

2.2.3 Application au prix de l'option $F(t, S_t)$

Soit $F(t, S) \in \{C(t, S), P(t, S)\}$ la fonction de valorisation d'une option européenne, en fonction du temps et du prix spot du sous-jacent.

En appliquant le lemme d'Itô à $F(t, S_t)$, avec S_t suivant la dynamique vue précédemment, on obtient :

$$dF(t, S_t) = \left(\frac{\partial F}{\partial t} + \mu S_t \frac{\partial F}{\partial S} + \frac{1}{2} \sigma^2 S_t^2 \frac{\partial^2 F}{\partial S^2} \right) dt + \sigma S_t \frac{\partial F}{\partial S}\, dB_t$$

Cette expression montre que le prix de l'option est lui-même un processus d'Itô, avec une composante aléatoire proportionnelle à la dérivée première en S (i.e. le Delta). C'est précisément ce terme de sensibilité qui sera neutralisé dans la stratégie de couverture dynamique présentée à la section suivante.

2.2.4 Construction du portefeuille de couverture

Dans une optique de réplication dynamique, on construit un portefeuille auto-finançable défini par :

$$\Pi_t = F(t, S_t) - \Delta_t S_t,$$

où :

- $F(t, S_t)$ représente la valeur théorique de l'option à l'instant t,
- Δ_t est la position dynamique dans le sous-jacent,
- S_t est le prix spot de l'actif à l'instant t.

L'objectif est de rendre ce portefeuille **sans risque**, c'est-à-dire parfaitement immunisé contre les fluctuations aléatoires de S_t. Pour cela, on choisit une position de couverture dynamique telle que :

$$\Delta_t = \frac{\partial F}{\partial S}(t, S_t),$$

c'est-à-dire la dérivée de la valeur de l'option par rapport au sous-jacent (le *Delta*).

Calcul de la dynamique du portefeuille
On applique Itô à la dynamique du portefeuille :

$$d\Pi_t = dF(t, S_t) - \Delta_t \, dS_t.$$

En injectant les expressions obtenues précédemment pour $dF(t, S_t)$ et dS_t, on obtient :

$$\begin{aligned} d\Pi_t = &\left(\frac{\partial F}{\partial t} + \mu S_t \frac{\partial F}{\partial S} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 F}{\partial S^2} \right) dt + \sigma S_t \frac{\partial F}{\partial S} dB_t \\ &- \Delta_t \left(\mu S_t dt + \sigma S_t dB_t \right). \end{aligned}$$

En regroupant et simplifiant :

$$\begin{aligned} d\Pi_t = &\left(\frac{\partial F}{\partial t} + \mu S_t \left(\frac{\partial F}{\partial S} - \Delta_t \right) + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 F}{\partial S^2} \right) dt \\ &+ \sigma S_t \left(\frac{\partial F}{\partial S} - \Delta_t \right) dB_t. \end{aligned}$$

En posant $\Delta_t = \frac{\partial F}{\partial S}$, on neutralise le risque de marché (terme en dB_t), et le portefeuille devient localement sans risque :

$$d\Pi_t = \left(\frac{\partial F}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 F}{\partial S^2} \right) dt.$$

2.2.5 Principe d'absence d'arbitrage

Par hypothèse, tout portefeuille sans risque doit croître au taux d'intérêt sans risque r. Ainsi, la dynamique du portefeuille doit vérifier :

$$d\Pi_t = r\Pi_t \, dt = r \left(F(t, S_t) - S_t \frac{\partial F}{\partial S} \right) dt.$$

On identifie les deux expressions de $d\Pi_t$:

$$\frac{\partial F}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 F}{\partial S^2} = r \left(F - S_t \frac{\partial F}{\partial S} \right).$$

Ce qui revient, après réorganisation, à l'équation de Black-Scholes :

$$\boxed{\frac{\partial F}{\partial t} + rS \frac{\partial F}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 F}{\partial S^2} = rF.}$$

Cette EDP constitue la condition nécessaire pour qu'une stratégie de couverture dynamique permette de répliquer le payoff de l'option sans prise de risque, dans un cadre sans arbitrage.

2.2.6 Condition terminale

L'équation est rétro-propagée sur l'intervalle $[0,T]\times\mathbb{R}_+^*$, à partir de la condition terminale donnée par le payoff à maturité :

$$\begin{cases} C(T,S) = \max(S-K,0), \\ P(T,S) = \max(K-S,0). \end{cases}$$

Ces conditions initialisent la solution de l'EDP, à partir desquelles on remonte le temps pour déterminer $F(t,S)$ pour tout $t<T$.

2.3 Solution analytique : Formules explicites du call et du put

Dans un cadre Black-Scholes standard, lorsque les hypothèses de marché parfait et sans arbitrage sont vérifiées, la valorisation des options européennes admet une solution fermée. Cette solution repose sur la représentation risque-neutre de l'espérance des flux futurs, ainsi que sur les propriétés de la loi log-normale du sous-jacent sous la mesure martingale équivalente.

Nous présentons ici la formule de pricing du call européen, puis en déduisons celle du put via la parité call-put. Le cadre théorique s'appuie sur les fondements probabilistes rappelés ci-dessous.

[Valorisation sous la mesure risque-neutre] Soit $h(S_T)$ un payoff contingent à la date d'échéance T. Le prix au temps $t\in[0,T]$ de l'option est donné par :

$$F_t = e^{-r(T-t)}\,\mathbb{E}^{\mathbb{Q}}\left[h(S_T)\mid \mathcal{F}_t\right],$$

où $\mathbb{Q}$ désigne la mesure de probabilité équivalente sous laquelle les actifs actualisés sont des martingales, et $\mathcal{F}_t$ est la filtration naturelle du marché.

En particulier, à l'instant initial :

$$F_0 = e^{-rT}\,\mathbb{E}^{\mathbb{Q}}\left[h(S_T)\right].$$

Sous la mesure $\mathbb{Q}$, la dynamique du sous-jacent devient :

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t^{\mathbb{Q}},$$

où $W_t^{\mathbb{Q}}$ est un mouvement brownien standard sous $\mathbb{Q}$. La solution de cette équation différentielle stochastique s'écrit :

$$S_T = S_0\exp\left(\left(r-\frac{1}{2}\sigma^2\right)T + \sigma W_T^{\mathbb{Q}}\right),$$

ce qui implique que $\log(S_T) \sim \mathcal{N}\left(\log(S_0) + \left(r - \frac{1}{2}\sigma^2\right)T,\, \sigma^2 T\right)$, c'est-à-dire que S_T suit une loi log-normale.

Ce changement de mesure est justifié par le théorème de Girsanov :

[Théorème de Girsanov — cas particulier] Soit $\lambda \in \mathbb{R}$ une constante. On définit la densité de changement de mesure suivante :

$$L_T = \exp\left(-\lambda B_T - \frac{1}{2}\lambda^2 T\right).$$

La probabilité $\mathbb{Q}$ définie par $\frac{d\mathbb{Q}}{d\mathbb{P}} = L_T$ est équivalente à $\mathbb{P}$, et sous $\mathbb{Q}$, le processus :

$$W_t^{\mathbb{Q}} := B_t + \lambda t$$

est un mouvement brownien standard. Cette construction permet de transformer la dérive μ en un taux sans risque r, ce qui garantit que le sous-jacent actualisé est une martingale sous $\mathbb{Q}$.

Dans ce cadre, la valorisation du call s'exprime comme :

$$C(0, S_0) = e^{-rT}\,\mathbb{E}^{\mathbb{Q}}\left[\max(S_T - K, 0)\right],$$

ce qui nous conduit à la formule fermée de Black-Scholes pour une option call européenne, que nous dériverons explicitement dans la section suivante.

La formule du put sera ensuite obtenue à partir de la relation de parité call-put :

$$C(t, S) - P(t, S) = S - Ke^{-r(T-t)}.$$

2.3.1 Démonstration de la formule du call européen

Considérons une option call européenne de maturité T et de prix d'exercice K. Son prix au temps initial $t = 0$ est donné par :

$$C_0 = \mathbb{E}^{\mathbb{Q}} \left[e^{-rT} \cdot (S_T - K)^+ \right].$$

On sépare l'espérance en deux composantes :

$$C_0 = \mathbb{E}^{\mathbb{Q}} \left[S_T e^{-rT} \cdot_{\{S_T \geq K\}} \right] - K e^{-rT} \cdot \mathbb{Q}(S_T \geq K). \tag{2.1}$$

Sous la mesure risque-neutre $\mathbb{Q}$, on a :

$$S_T = S_0 \cdot \exp\left(\left(r - \frac{\sigma^2}{2} \right) T + \sigma W_T \right), \quad W_T \sim \mathcal{N}(0, T).$$

On note que :

$$S_T \geq K \quad \Longleftrightarrow \quad \log(S_T) \geq \log(K) \quad \Longleftrightarrow \quad W_T \geq \frac{1}{\sigma} \left[\log\left(\frac{K}{S_0} \right) - \left(r - \frac{\sigma^2}{2} \right) T \right].$$

On pose alors :

$$d_2 = \frac{\log(S_0/K) + \left(r - \frac{\sigma^2}{2} \right) T}{\sigma \sqrt{T}}, \quad \text{et} \quad d_1 = d_2 + \sigma \sqrt{T}.$$

Ce qui donne :

$$\mathbb{Q}(S_T \geq K) = \mathbb{Q}(W_T \geq -d_2 \sqrt{T}) = N(d_2).$$

Calcul du terme d'espérance :

On développe le premier terme de l'équation (2.1) :

$$\begin{aligned} \mathbb{E}^{\mathbb{Q}} \left[S_T e^{-rT} \cdot_{\{S_T \geq K\}} \right] &= \mathbb{E}^{\mathbb{Q}} \left[S_0 e^{(r - \frac{\sigma^2}{2})T + \sigma W_T} \cdot e^{-rT} \cdot_{\{W_T \geq -d_2 \sqrt{T}\}} \right] \\ &= S_0 \cdot \mathbb{E}^{\mathbb{Q}} \left[e^{-\frac{\sigma^2}{2} T + \sigma W_T} \cdot_{\{W_T \geq -d_2 \sqrt{T}\}} \right]. \end{aligned} \tag{2.2}$$

Ce terme est équivalent à une espérance de la forme $\mathbb{E}[e^{\sigma W_T} \cdot_{W_T \geq a}]$, ce qui peut être traité par un **changement de mesure de Girsanov**. On pose :

$$L_T := \exp\left(-\sigma W_T - \frac{1}{2} \sigma^2 T \right), \quad \frac{d\widetilde{\mathbb{Q}}}{d\mathbb{Q}} = L_T,$$

alors sous $\widetilde{\mathbb{Q}}$, le processus $\widetilde{W}_t = W_t + \sigma t$ est un brownien, et on peut réécrire l'espérance de (2.2) comme :

$$\begin{aligned} S_0 \cdot \mathbb{E}^{\mathbb{Q}} \left[e^{-\frac{\sigma^2}{2} T + \sigma W_T} \cdot_{\{W_T \geq -d_2 \sqrt{T}\}} \right] &= S_0 \cdot \widetilde{\mathbb{Q}} \left(\widetilde{W}_T \geq -d_2 \sqrt{T} - \sigma T \right) \\ &= S_0 \cdot N(d_1). \end{aligned} \tag{2.3}$$

Formule finale du call :

En injectant (2.3) et $\mathbb{Q}(S_T \geq K) = N(d_2)$ dans (2.1), on obtient :

$$C_0 = S_0 N(d_1) - K e^{-rT} N(d_2).$$

Pour $t \in [0, T]$, la formule devient :

$$C(t, S_t) = S_t N(d_1(t, S_t)) - K e^{-r(T-t)} N(d_2(t, S_t)),$$

avec :

$$\begin{cases} d_1(t,x) = \dfrac{\log(x/K) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}} \\ d_2(t,x) = d_1(t,x) - \sigma\sqrt{T-t} \end{cases}$$

2.3.2 Démonstration de la formule du put européen

La formule du put s'obtient via la parité call-put :

$$P(t, S_t) = C(t, S_t) - S_t + K e^{-r(T-t)}.$$

En injectant l'expression explicite du call :

$$\begin{aligned} P(t, S_t) &= S_t N(d_1) - K e^{-r(T-t)} N(d_2) - S_t + K e^{-r(T-t)} \\ &= -S_t(1 - N(d_1)) + K e^{-r(T-t)}(1 - N(d_2)) \\ &= -S_t N(-d_1) + K e^{-r(T-t)} N(-d_2), \end{aligned}$$

en utilisant la symétrie $1 - N(d) = N(-d)$.

La formule explicite du put européen devient donc :

$$P(t, S_t) = -S_t N(-d_1(t, S_t)) + K e^{-r(T-t)} N(-d_2(t, S_t)).$$

Les expressions de d_1 et d_2 sont les mêmes que dans le cas du call.

2.4 Conclusion Modèle Black Scholes

Le modèle de Black-Scholes constitue une pierre angulaire de la théorie moderne de la valorisation des options. En supposant une dynamique lognormale du sous-jacent, une absence d'arbitrage et des marchés complets et frictionless, il permet d'obtenir une formule fermée pour le prix des options européennes, dont l'élégance et l'efficacité computationnelle ont largement contribué à sa popularité en pratique.

Ce cadre repose cependant sur des hypothèses idéalistes : volatilité constante, taux sans risque fixe, absence de dividendes, et comportement brownien du sous-jacent. Malgré ces limitations, le modèle demeure une référence incontournable, tant pour son rôle fondamental dans la compréhension des mécanismes de pricing que pour servir de benchmark aux méthodes numériques.

Dans ce travail, nous avons posé les fondements du modèle en explicitant la dynamique stochastique du sous-jacent, les principes de réplication et de neutralité au risque, ainsi que la dérivation rigoureuse de la formule de Black-Scholes via le changement de mesure et les outils de calcul stochastique.

Cette base théorique nous permettra de comparer, dans les sections suivantes, les approches numériques.

Chapter 3

Méthode de Monte Carlo

3.1 Principe général

La méthode de Monte Carlo constitue l'un des piliers du calcul numérique en finance quantitative. Elle repose sur la simulation d'échantillons aléatoires afin d'estimer des espérances mathématiques, souvent inaccessibles analytiquement. Dans le cadre du pricing d'options, elle permet d'évaluer des instruments dérivés en projetant la dynamique du sous-jacent sous une mesure martingale équivalente, puis en prenant l'espérance du payoff actualisé.

Cette méthode est particulièrement précieuse lorsque :

- la structure du payoff est trop complexe pour admettre une solution fermée (options exotiques, barrières, lookbacks, etc.) ;
- la dimension du problème rend inapplicables les grilles de différences finies (cas multi-actifs ou path-dependent) ;
- ou lorsque l'on souhaite obtenir rapidement une estimation robuste et modulaire pour des scénarios de stress ou des calculs de sensibilité.

3.1.1 Cadre probabiliste et convergence

Soit $X \in \mathbb{R}$ une variable aléatoire définie sur un espace de probabilité $(\Omega, \mathcal{F}, \mathbb{P})$, avec espérance finie $\mathbb{E}[X]$. L'objectif est d'estimer $\mathbb{E}[X]$ de manière empirique par simulation. La méthode de Monte Carlo procède comme suit :

1. Générer N réalisations indépendantes $X^{(1)}, X^{(2)}, \ldots, X^{(N)}$ du même modèle stochastique,
2. Calculer la moyenne empirique :

$$\overline{X}_N = \frac{1}{N} \sum_{i=1}^{N} X^{(i)}.$$

Par la loi forte des grands nombres, on a la convergence presque sûre :

$$\overline{X}_N \xrightarrow[N\to\infty]{\text{p.s.}} \mathbb{E}[X].$$

De plus, sous l'hypothèse $\mathbb{E}[X^2] < \infty$, le théorème central limite donne :

$$\sqrt{N}\left(\overline{X}_N - \mathbb{E}[X]\right) \xrightarrow[N\to\infty]{\mathcal{L}} \mathcal{N}(0, \sigma^2),$$

où $\sigma^2 = \mathrm{Var}(X)$.

Cela implique que l'erreur statistique associée à l'estimateur décroit à l'ordre $\mathcal{O}(1/\sqrt{N})$. Pour atteindre une précision de l'ordre de 10^{-k}, il faut simuler $N \sim 10^{2k}$ trajectoires — d'où l'intérêt d'implémentations vectorisées et de techniques de réduction de variance pour optimiser les temps de calcul.

3.1.2 Application au pricing d'options européennes

Dans le cadre du modèle de Black-Scholes, le prix théorique d'un dérivé financier dont le payoff est fonction uniquement du sous-jacent à maturité s'exprime comme une espérance sous la mesure risque-neutre :

$$F(t, S_t) = e^{-r(T-t)}\,\mathbb{E}^{\mathbb{Q}}\left[h(S_T) \mid S_t\right],$$

où :

- $h(S_T)$ est le payoff du produit à l'échéance T,
- S_t est le niveau du sous-jacent à la date d'évaluation t,
- $\mathbb{Q}$ est la mesure martingale associée au taux sans risque r.

La méthode de Monte Carlo vise à approximer cette espérance en simulant directement la distribution de S_T sous $\mathbb{Q}$. Le schéma général est le suivant :

1. Générer N trajectoires indépendantes $S_T^{(1)}, \ldots, S_T^{(N)}$ selon la dynamique risque-neutre :

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

2. Évaluer le payoff simulé $h(S_T^{(i)})$ pour chaque trajectoire,
3. Calculer l'estimateur de Monte Carlo :

$$\widehat{F}_N(t, S_t) = \frac{e^{-r(T-t)}}{N}\sum_{i=1}^{N} h(S_T^{(i)}).$$

Ce schéma est particulièrement adapté aux options plain vanilla (call, put), mais également extensible à des produits path-dependent ou à barrière.

3.1.3 Propriétés statistiques et estimation de l'erreur

L'estimateur $\widehat{F}_N$ est **non biaisé**, et converge presque sûrement vers la vraie valeur $F(t, S_t)$ lorsque $N \to \infty$ (loi forte des grands nombres). Sa variance est donnée par :

$$\mathrm{Var}(\widehat{F}_N) = \frac{1}{N} \cdot \mathrm{Var}^{\mathbb{Q}}(h(S_T)).$$

On peut donc approximer l'**erreur-type** (standard error) par :

$$\text{Erreur} \approx \frac{\sigma_h}{\sqrt{N}}, \quad \text{où } \sigma_h^2 = \mathrm{Var}^{\mathbb{Q}}(h(S_T)).$$

Cette estimation permet de quantifier l'incertitude liée au caractère aléatoire de la méthode. On peut en déduire un intervalle de confiance asymptotique à 95% :

$$\widehat{F}_N(t, S_t) \ \pm \ 1.96 \cdot \frac{\sigma_h}{\sqrt{N}}.$$

L'ordre de convergence est $\mathcal{O}(1/\sqrt{N})$, ce qui impose en pratique un compromis entre précision et temps de calcul. Dans la suite, nous présenterons plusieurs techniques de réduction de variance visant à améliorer l'efficacité de cet estimateur sans augmenter artificiellement N.

3.1.4 Application au modèle de Black-Scholes

Dans le cadre du modèle de Black-Scholes, et sous la mesure risque-neutre $\mathbb{Q}$, la dynamique du sous-jacent S_t est décrite par l'équation différentielle stochastique suivante :

$$dS_t = rS_t\,dt + \sigma S_t\,dB_t,$$

où :

- $r \in \mathbb{R}_+$ est le taux d'intérêt sans risque, constant dans le temps,
- $\sigma > 0$ est la volatilité constante du sous-jacent,
- $(B_t)_{t\geq 0}$ est un mouvement brownien standard sous $\mathbb{Q}$.

Cette dynamique admet une solution analytique explicite. À tout instant $t \in [0, T]$, le prix du sous-jacent à maturité T est donné par :

$$S_T = S_t \cdot \exp\left[\left(r - \frac{1}{2}\sigma^2\right)(T-t) + \sigma(B_T - B_t)\right].$$

Comme $B_T - B_t \sim \mathcal{N}(0, T-t)$, on peut réécrire cette expression sous forme simulable :

$$S_T = S_t \cdot \exp\left[\left(r - \frac{1}{2}\sigma^2\right)(T-t) + \sigma\sqrt{T-t} \cdot Z\right], \quad \text{avec } Z \sim \mathcal{N}(0,1).$$

Cette représentation est particulièrement adaptée à la méthode de Monte Carlo, car elle permet de simuler directement les réalisations de S_T à partir de tirages standards $Z \sim \mathcal{N}(0,1)$, sans avoir à discretiser l'équation stochastique.

Cette propriété de fermeture analytique de la solution est une des raisons pour lesquelles le modèle de Black-Scholes reste un benchmark universel en pricing numérique. Elle garantit à la fois précision, rapidité de simulation et contrôle du biais de discrétisation (nul dans ce cas).

3.2 Implémentation en C++

Dans cette section, nous présentons l'implémentation numérique de la méthode de Monte Carlo pour le pricing d'options européennes dans le cadre du modèle de Black-Scholes. Nous commençons par la version de base — sans technique de réduction de variance — afin d'introduire clairement la logique du simulateur et la structure algorithmique associée.

3.2.1 Pricing simple d'une option européenne par Monte Carlo

On considère une option européenne (call ou put) dont le sous-jacent suit, sous la mesure risque-neutre $\mathbb{Q}$, la dynamique suivante :

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t,$$

avec solution explicite :

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

Le prix théorique de l'option à l'instant t est donné par l'espérance actualisée du payoff sous $\mathbb{Q}$:

$$F(t, S_t) = e^{-r(T-t)} \cdot \mathbb{E}^{\mathbb{Q}}\left[h(S_T)\right],$$

où $h(S_T)$ désigne la fonction de payoff définie par :

$$h(S_T) = \begin{cases} \max(S_T - K, 0) & \text{(option call)}, \\ \max(K - S_T, 0) & \text{(option put)}. \end{cases}$$

Principe de l'algorithme de simulation : L'idée est de simuler N trajectoires indépendantes du sous-jacent à maturité, en procédant comme suit :

1. Générer $Z_1, Z_2, \ldots, Z_N \sim \mathcal{N}(0,1)$, i.i.d.
2. Calculer les réalisations simulées de S_T via :

$$S_T^{(i)} = S_0 \cdot \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}\cdot Z_i\right),$$

3. Évaluer le payoff simulé $h(S_T^{(i)})$ pour chaque trajectoire,

4. Prendre la moyenne empirique et actualiser au taux sans risque :

$$\widehat{F}_N(t, S_t) = e^{-r(T-t)} \cdot \frac{1}{N} \sum_{i=1}^{N} h(S_T^{(i)}).$$

Cet estimateur est asymptotiquement sans biais, avec une erreur-type découlant de la variance de $h(S_T)$, estimable empiriquement. L'ensemble de cette procédure peut être encapsulé dans un simulateur C++ haute performance, que nous détaillons dans la section suivante.

Structure du code L'implémentation en C++ suivra les étapes suivantes :

- **Initialisation des paramètres** : S_0, K, T, r, σ, N.
- **Génération des variables normales** $Z_i \sim \mathcal{N}(0, 1)$.
- **Simulation des** $S_T^{(i)}$ à l'aide de la formule fermée.
- **Calcul du payoff** $h(S_T^{(i)})$.
- **Calcul de la moyenne empirique** et actualisation pour obtenir $\hat{F}_N(t, S_t)$.

L'objectif est d'obtenir une architecture de code claire et modulaire permettant de réutiliser facilement ces fonctions lors de l'introduction des techniques de réduction de variance dans les sections suivantes.

3.2.2 Code C++ Monte Carlo pricing option EU

```
#include <iostream>
#include <cmath>
#include <random>
#include <iomanip>

// Payoff functions: European call and put
double payoff_call(double ST, double K) {
    return std::max(ST - K, 0.0);
}

double payoff_put(double ST, double K) {
    return std::max(K - ST, 0.0);
}

// Monte Carlo pricing for European option (call or put)
double monte_carlo_price(int N, double S0, double K,
                         double r, double sigma,
                         double T, bool isCall) {
    std::mt19937 rng(42);                          // fixed seed
    std::normal_distribution<> norm(0.0, 1.0);     // Z ~ N(0,1)
    double sum_payoffs = 0.0;

    for (int i = 0; i < N; ++i) {
        double Z = norm(rng);                      // generate standard normal
        double ST = S0 * exp((r - 0.5 * sigma * sigma) * T
                       + sigma * sqrt(T) * Z);     // simulate terminal price
        double payoff = isCall ? payoff_call(ST, K)
                               : payoff_put(ST, K);
        sum_payoffs += payoff;
    }

    return exp(-r * T) * (sum_payoffs / N);        // discounted average payoff
}
```

```
// Main function
int main() {
    double S0 = 100.0, K = 100.0;
    double r = 0.05, sigma = 0.2;
    double T = 1.0;
    int N = 1000000;
    bool isCall = true;

    double price = monte_carlo_price(N, S0, K, r, sigma, T, isCall);

    std::cout << "Option price (" << (isCall ? "Call" : "Put") << ") : "
              << std::fixed << std::setprecision(6)
              << price << std::endl;

    return 0;
}
```

3.2.3 Résultats numériques

La simulation est effectuée avec les paramètres suivants, typiques d'un calibrage Black-Scholes en environnement de marché liquide :

- $S_0 = 100$: prix spot de l'actif sous-jacent,
- $K = 100$: strike (au-the-money),
- $r = 5\%$: taux sans risque annualisé,
- $\sigma = 20\%$: volatilité implicite,
- $T = 1$ an : maturité,
- $N = 10^6$: nombre de trajectoires simulées.

Résultat pour le Call :

$$\hat{F}_N^{\text{call}} = 10.4741, \qquad F_{\text{BS}} = 10.4506$$

Résultat pour le Put :

$$\hat{F}_N^{\text{put}} = 5.5793, \qquad F_{\text{BS}} = 5.5735$$

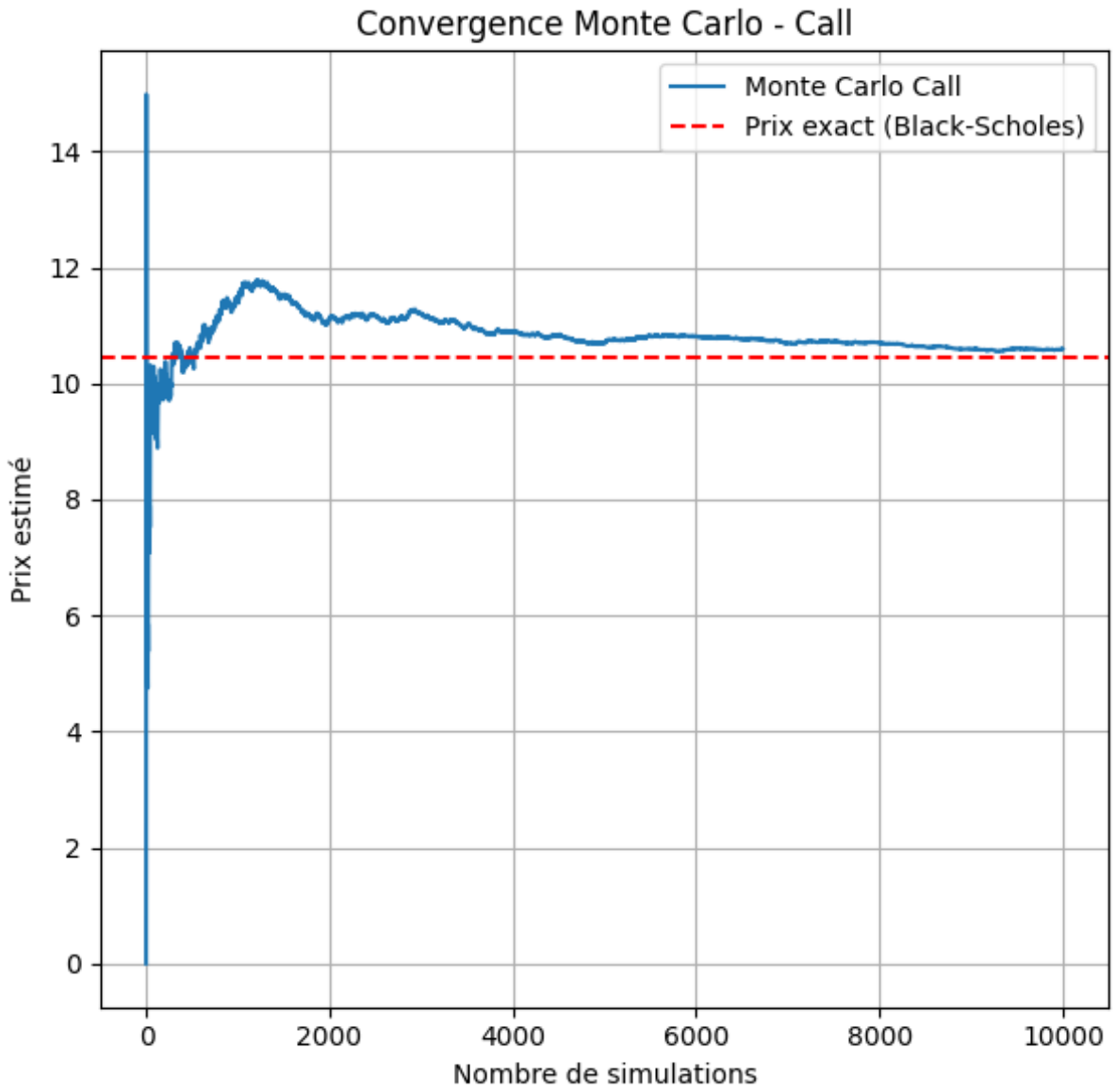


Figure 3.1: Convergence du prix estimé du call en fonction de N

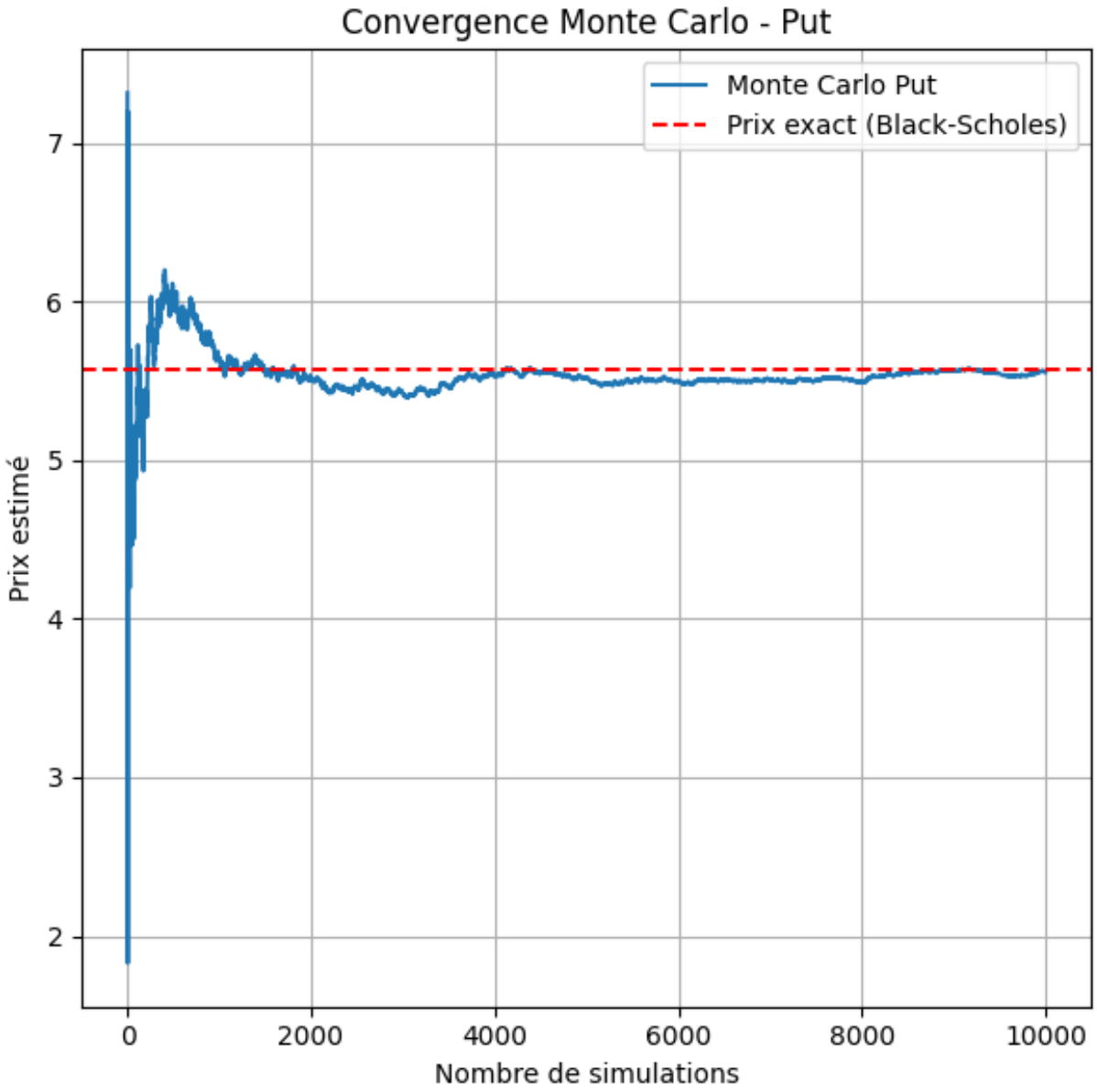


Figure 3.2: Convergence du prix estimé du put en fonction de N

Interprétation Les résultats estimés sont cohérents avec les valeurs données par la formule fermée de Black-Scholes, avec une erreur relative inférieure à 0.3%. Cela valide l'efficacité du simulateur Monte Carlo dans le cas d'un payoff standard.

Remarque La variance de l'estimateur chute à l'ordre $\mathcal{O}(1/\sqrt{N})$, ce qui implique des besoins computationnels élevés pour atteindre des niveaux de précision fins. D'où l'intérêt, dans la suite, d'explorer des techniques de réduction de variance.

3.2.4 Estimation de l'erreur et intervalle de confiance

L'estimateur Monte Carlo introduit précédemment est **sans biais** pour l'espérance $\mathbb{E}^{\mathbb{Q}}[h(S_T)]$, mais reste soumis à une **erreur statistique** liée à la variance intrinsèque de la variable simulée. Pour évaluer la fiabilité de l'estimation, il est indispensable de quantifier cette incertitude de manière rigoureuse.

Quantités à estimer À partir des trajectoires simulées $S_T^{(i)}$, on évalue :

- l'écart-type empirique $\hat{\sigma}_h$ du payoff $h(S_T)$,
- l'erreur-type associée à l'estimateur :

$$\text{Erreur} = \frac{\hat{\sigma}_h}{\sqrt{N}},$$

- un intervalle de confiance asymptotique à 95 % :

$$\hat{F}_N \ \pm \ 1.96 \cdot \frac{\hat{\sigma}_h}{\sqrt{N}}.$$

Justification théorique On note $X_i := h(S_T^{(i)})$, les réalisations indépendantes de la variable aléatoire associée au payoff. Si $X \in L^2(\mathbb{Q})$, alors :

$$\text{Var}(\hat{F}_N) = \frac{\text{Var}(X)}{N}.$$

Le théorème central limite garantit que :

$$\sqrt{N}\left(\hat{F}_N - F\right) \xrightarrow{\mathcal{L}} \mathcal{N}(0, \sigma_h^2), \quad \text{où } \sigma_h^2 = \text{Var}(h(S_T)).$$

En pratique, la variance inconnue est estimée empiriquement via :

$$\hat{\sigma}_h^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(h(S_T^{(i)}) - \overline{h}\right)^2, \quad \text{avec } \overline{h} = \frac{1}{N}\sum_{i=1}^{N} h(S_T^{(i)}).$$

L'estimation de la variance de l'estimateur devient donc :

$$\widehat{\text{Var}}(\hat{F}_N) = \frac{\hat{\sigma}_h^2}{N}, \quad \text{et Erreur} = \sqrt{\widehat{\text{Var}}(\hat{F}_N)}.$$

Interprétation opérationnelle Cette borne d'erreur fournit une information essentielle :

- Elle permet de contrôler la précision numérique sans disposer du prix exact.
- Elle guide le choix du nombre N de simulations à allouer pour une tolérance cible.
- Elle est exploitée pour calibrer l'allocation de ressources dans les systèmes de pricing massivement parallèles.

Dans la suite, cette estimation sera utilisée pour comparer l'efficacité des différentes variantes de Monte Carlo (standard vs. réduction de variance).

Lien avec le code précédent : Dans l'implémentation numérique suivante, nous allons enrichir le code développé en section 3.2.1 pour :

- Stocker l'ensemble des payoffs simulés,
- Calculer la variance empirique,
- Afficher l'intervalle de confiance à 95%.

Cette amélioration permettra d'observer non seulement la convergence de l'estimateur mais aussi l'évolution de la précision statistique lorsque le nombre de simulations N augmente. À chaque itération, nous avons calculé :

- la moyenne empirique $\overline{h}$,
- la variance empirique $\hat{\sigma}_h^2$,
- l'erreur-type $\frac{\hat{\sigma}_h}{\sqrt{N}}$,
- un intervalle de confiance à 95% donné par :

$$\left[\hat{F}_N - 1.96 \times \frac{\hat{\sigma}_h}{\sqrt{N}}\,,\, \hat{F}_N + 1.96 \times \frac{\hat{\sigma}_h}{\sqrt{N}}\right].$$

3.2.5 Code C++ Erreur + IC95

```
// Payoff functions: European call and put
double payoff_call(double ST, double K) {
    return std::max(ST - K, 0.0);
}

double payoff_put(double ST, double K) {
    return std::max(K - ST, 0.0);
}

// Simulate S_T under Black-Scholes dynamics
double simulate_ST(double S0, double r, double sigma, double T, double Z) {
    return S0 * std::exp((r - 0.5 * sigma * sigma) * T + sigma * std::sqrt(T) * Z);
}

// Monte Carlo estimation + erreur call
void monte_carlo_call_with_error(int N, double S0, double K, double r, double sigma,
double T) {
    std::mt19937 rng(42); // fixed seed for reproducibility
    std::normal_distribution<double> norm(0.0, 1.0); // standard normal generator

    double sum_payoff = 0.0, sum_sq = 0.0;
    std::ofstream file("convergence_error_call.csv");
    file << "N,Estimate,StandardError\n";

    for (int i = 1; i <= N; ++i) {
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        double payoff = payoff_call(ST, K);
        sum_payoff += payoff;
        sum_sq += payoff * payoff;

        double mean = sum_payoff / i;
        double price = std::exp(-r * T) * mean;
        double var = (i > 1) ? (sum_sq - sum_payoff * mean) / (i - 1) : 0.0;
        double error = (i > 1) ? std::exp(-r * T) * std::sqrt(var / i) : 0.0;

        file << i << "," << price << "," << error << "\n";
    }

    file.close();

    double mean = sum_payoff / N;
    double var = (sum_sq - sum_payoff * mean) / (N - 1);
    double error = std::exp(-r * T) * std::sqrt(var / N);
    double price = std::exp(-r * T) * mean;
```

```
    std::cout << "CALL:\n";
    std::cout << "Estimated price: " << price << "\n";
    std::cout << "Standard error: " << error << "\n";
    std::cout << "95% confidence interval: [" << price - 1.96 * error
              << " ; " << price + 1.96 * error << "]\n\n";
}

// Monte Carlo estimation + erreur put
void monte_carlo_put_with_error(int N, double S0, double K, double r, double sigma,
double T) {
    std::mt19937 rng(42);
    std::normal_distribution<double> norm(0.0, 1.0);

    double sum_payoff = 0.0, sum_sq = 0.0;
    std::ofstream file("convergence_error_put.csv");
    file << "N,Estimate,StandardError\n";

    for (int i = 1; i <= N; ++i) {
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        double payoff = payoff_put(ST, K);
        sum_payoff += payoff;
        sum_sq += payoff * payoff;

        double mean = sum_payoff / i;
        double price = std::exp(-r * T) * mean;
        double var = (i > 1) ? (sum_sq - sum_payoff * mean) / (i - 1) : 0.0;
        double error = (i > 1) ? std::exp(-r * T) * std::sqrt(var / i) : 0.0;

        file << i << "," << price << "," << error << "\n";
    }

    file.close();

    double mean = sum_payoff / N;
    double var = (sum_sq - sum_payoff * mean) / (N - 1);
    double error = std::exp(-r * T) * std::sqrt(var / N);
    double price = std::exp(-r * T) * mean;

    std::cout << "PUT:\n";
    std::cout << "Estimated price: " << price << "\n";
    std::cout << "Standard error: " << error << "\n";
    std::cout << "95% confidence interval: [" << price - 1.96 * error
              << " ; " << price + 1.96 * error << "]\n";
}

// Main function
int main() {
```

```
    double S0 = 100.0, K = 100.0;
    double r = 0.05, sigma = 0.2;
    double T = 1.0;
    int N = 100000;

    monte_carlo_call_with_error(N, S0, K, r, sigma, T);
    monte_carlo_put_with_error(N, S0, K, r, sigma, T);

    return 0;
}
```

3.2.6 Résultats numériques

Pour les paramètres suivants :

$$S_0 = 100, \quad K = 100, \quad r = 5\%, \quad \sigma = 20\%, \quad T = 1 \text{ an}, \quad N = 100\,000,$$

nous obtenons les résultats suivants pour les options européennes :

Option Call

- Prix estimé (Monte Carlo) : $\hat{F}_N = 10.5951$,
- Erreur-type empirique : 0.14774,
- Intervalle de confiance à 95 % : [10.3055 ; 10.8847].

Le prix exact obtenu à l'aide de la formule de Black-Scholes est $F_{\text{exact}} = 10.4506$, situé à l'intérieur de l'intervalle de confiance construit par Monte Carlo. Cela confirme la validité de l'approche statistique.

Option Put

- Prix estimé (Monte Carlo) : $\hat{F}_N = 5.55663$,
- Erreur-type empirique : 0.08678,
- Intervalle de confiance à 95 % : [5.3865 ; 5.7267].

Le prix exact selon Black-Scholes est $F_{\text{exact}} = 5.5735$, également contenu dans l'intervalle de confiance, ce qui confirme la robustesse de la méthode.

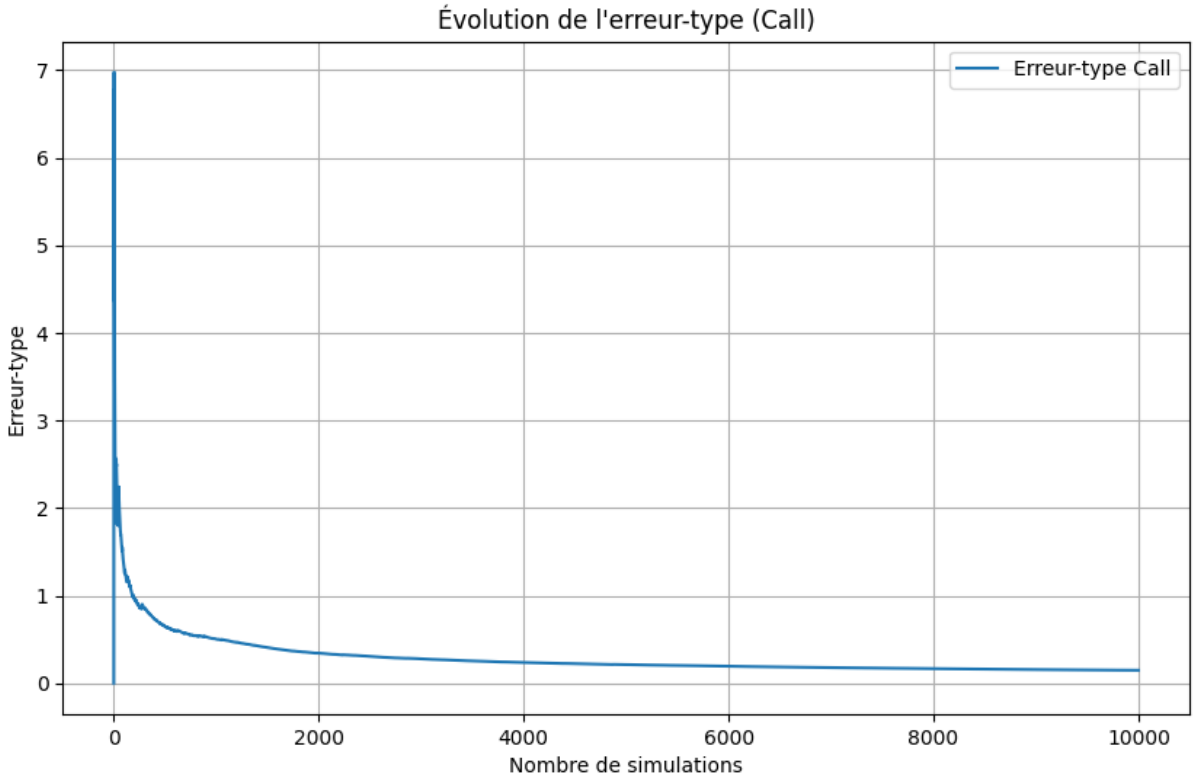


Figure 3.3: Convergence de l'erreur-type pour le call européen

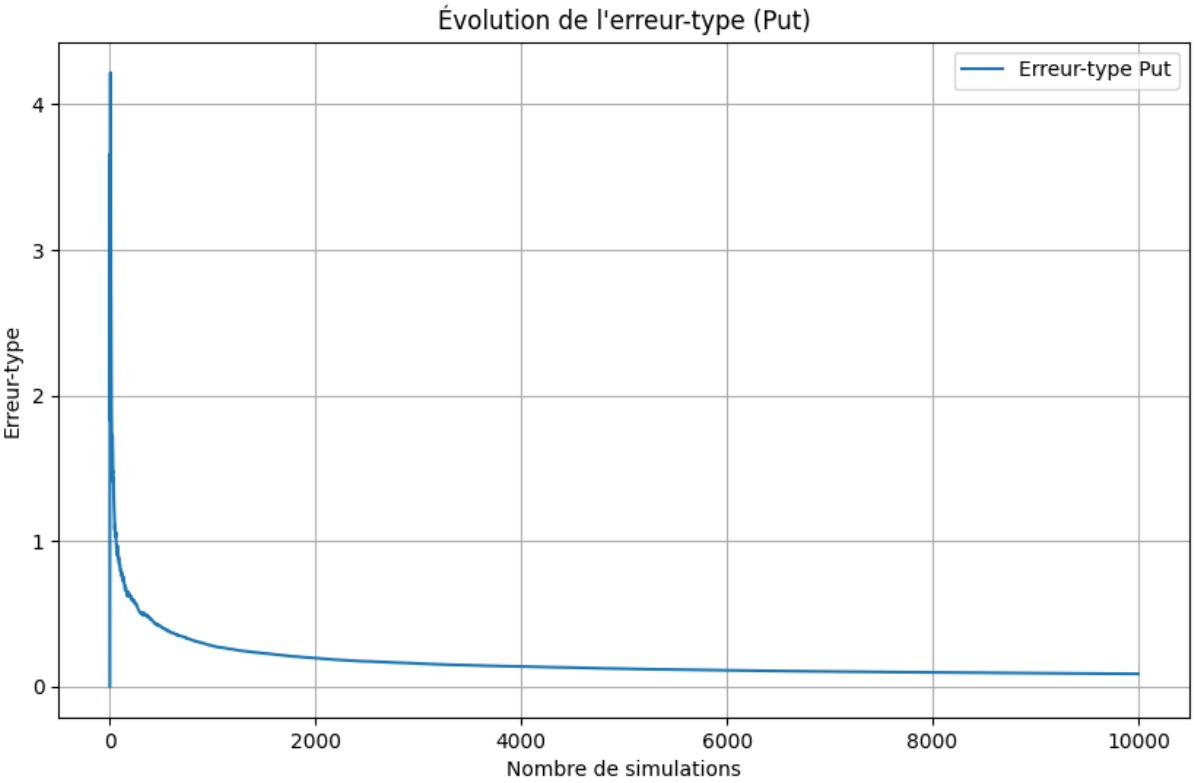


Figure 3.4: Convergence de l'erreur-type pour le put européen

Remarque sur la convergence L'erreur-type empirique diminue selon un taux en $\mathcal{O}\left(\frac{1}{\sqrt{N}}\right)$, comme prévu théoriquement. Cette propriété est illustrée sur les figures ci-dessus.

3.3 Réduction de variance

Pour améliorer l'efficacité statistique de l'estimateur Monte Carlo, nous introduisons deux techniques classiques de réduction de variance : les **variables antithétiques** et les **variables de contrôle**.

3.3.1 Méthode des variables antithétiques

Cette méthode exploite la symétrie de la loi normale centrée réduite. Pour chaque tirage $Z^{(i)} \sim \mathcal{N}(0,1)$, on simule deux trajectoires complémentaires du sous-jacent S_T :

$$S_T^{(i)} = S_0 \cdot \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T} \cdot Z^{(i)}\right), \quad S_T^{(i,\text{ant})} = S_0 \cdot \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T - \sigma\sqrt{T} \cdot Z^{(i)}\right).$$

L'estimateur antithétique s'obtient en moyennant les deux payoffs associés à chaque paire de trajectoires :

$$\hat{F}_N^{\text{ant}} = e^{-rT} \cdot \frac{1}{N} \sum_{i=1}^{N/2} \left[h\left(S_T^{(i)}\right) + h\left(S_T^{(i,\text{ant})}\right)\right].$$

Cette méthode permet une réduction significative de la variance lorsque le payoff est une fonction monotone de S_T, ce qui est le cas pour les options vanilla (call, put). Elle présente l'avantage d'améliorer la précision sans coût de simulation additionnel.

3.3.2 Méthode des variables de contrôle

La méthode des variables de contrôle repose sur l'introduction d'une variable auxiliaire Y, fortement corrélée au payoff $X = h(S_T)$, et dont l'espérance $\mathbb{E}[Y]$ est connue.

Un choix naturel est $Y = S_T$, pour lequel on connaît explicitement :

$$\mathbb{E}[S_T] = S_0 e^{rT}.$$

On construit alors une variable corrigée :

$$X^{\text{CV}} = X - \beta \cdot (Y - \mathbb{E}[Y]),$$

où le paramètre optimal β^* est donné par :

$$\beta^* = \frac{\text{Cov}(X,Y)}{\text{Var}(Y)}.$$

En remplaçant X par X^{CV} dans l'estimateur, on obtient :

$$\hat{F}_N^{\text{CV}} = e^{-rT} \cdot \frac{1}{N} \sum_{i=1}^{N} X_i^{\text{CV}}.$$

Cette méthode est particulièrement efficace lorsque X et Y sont positivement corrélés, ce qui est le cas pour le pricing d'un call ou d'un put par rapport au niveau de S_T. En pratique, le coefficient β est estimé empiriquement à partir des échantillons simulés.

Ces techniques seront évaluées numériquement dans la suite, afin de comparer leur performance en termes de réduction de l'erreur-type par rapport à l'estimateur standard.

3.3.3 Code C++ reduction de variance

```
#include <iostream>
#include <cmath>
#include <random>
#include <vector>
#include <fstream>
#include <iomanip>

// Payoff functions

// Payoff call
double payoff_call(double ST, double K) {
    return std::max(ST - K, 0.0);
}

// Payoff put
double payoff_put(double ST, double K) {
    return std::max(K - ST, 0.0);
}

// Simulate S_T Black-Scholes model

double simulate_ST(double S0, double r, double sigma, double T, double Z) {
    return S0 * std::exp((r - 0.5 * sigma * sigma) * T + sigma * std::sqrt(T) * Z);
}

// Black-Scholes closed-form

// Closed-form  call
double bs_call(double S0, double K, double r, double sigma, double T) {
    double d1 = (log(S0 / K) + (r + 0.5 * sigma * sigma) * T) / (sigma * sqrt(T));
    double d2 = d1 - sigma * sqrt(T);
    double N_d1 = 0.5 * erfc(-d1 / sqrt(2));
    double N_d2 = 0.5 * erfc(-d2 / sqrt(2));
    return S0 * N_d1 - K * exp(-r * T) * N_d2;
}

// Closed-form  put via call-put parity
double bs_put(double S0, double K, double r, double sigma, double T) {
    return bs_call(S0, K, r, sigma, T) - S0 + K * exp(-r * T);
}
```

```
// Monte Carlo

double monte_carlo(int N, double S0, double K, double r, double sigma, double T,
double (*payoff)(double, double), std::vector<double>& estimates) {
    std::mt19937 rng(42); // Fixed seed
    std::normal_distribution<double> norm(0.0, 1.0);
    double sum = 0.0;
    for (int i = 1; i <= N; ++i) {
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        sum += payoff(ST, K);
        estimates.push_back(exp(-r * T) * sum / i);
    }
    return exp(-r * T) * sum / N;
}

// Antithetic variates

double monte_carlo_antithetic(int N, double S0, double K, double r, double sigma,
double T, double (*payoff)(double, double),
    std::vector<double>& estimates) {
    std::mt19937 rng(42);
    std::normal_distribution<double> norm(0.0, 1.0);
    double sum = 0.0;
    for (int i = 1; i <= N; ++i) {
        double Z = norm(rng);
        double ST1 = simulate_ST(S0, r, sigma, T, Z);
        double ST2 = simulate_ST(S0, r, sigma, T, -Z);
        double payoff_avg = 0.5 * (payoff(ST1, K) + payoff(ST2, K));
        sum += payoff_avg;
        estimates.push_back(exp(-r * T) * sum / i);
    }
    return exp(-r * T) * sum / N;
}
```

```
//  Control variate
// Control variable: Y = S_T with E[Y] = S0 * exp(r * T)

double monte_carlo_control_variate(int N, double S0, double K, double r, double sigma,
double T, double (*payoff)(double, double),
    std::vector<double>& estimates) {
    std::mt19937 rng(42);
    std::normal_distribution<double> norm(0.0, 1.0);
    double sum = 0.0;
    double expected_ST = S0 * exp(r * T);
    for (int i = 1; i <= N; ++i) {
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        double adjusted = payoff(ST, K) - (ST - expected_ST); // beta = 1
        sum += adjusted;
        estimates.push_back(exp(-r * T) * sum / i);
    }
    return exp(-r * T) * sum / N;
}

//  Main

int main() {
    double S0 = 100.0, K = 100.0, r = 0.05, sigma = 0.2, T = 1.0;
    int N = 10000;

    std::vector<double> call_std, call_ant, call_ctrl;
    std::vector<double> put_std, put_ant, put_ctrl;

    // Call option pricing
    double price_call_std  = monte_carlo(N, S0, K, r, sigma, T, payoff_call,
    call_std);
    double price_call_ant  = monte_carlo_antithetic(N, S0, K, r, sigma, T,
    payoff_call, call_ant);
    double price_call_ctrl = monte_carlo_control_variate(N, S0, K, r, sigma, T,
    payoff_call, call_ctrl);
    double bs_call_price   = bs_call(S0, K, r, sigma, T);

    // Put option pricing
    double price_put_std  = monte_carlo(N, S0, K, r, sigma, T, payoff_put, put_std);
    double price_put_ant  = monte_carlo_antithetic(N, S0, K, r, sigma, T, payoff_put,
    put_ant);
    double price_put_ctrl = monte_carlo_control_variate(N, S0, K, r, sigma, T,
    payoff_put, put_ctrl);
    double bs_put_price   = bs_put(S0, K, r, sigma, T);

    std::cout << std::fixed << std::setprecision(4);
```

```
    std::cout << "==== OPTION CALL ====\n";
    std::cout << "Black-Scholes closed-form
    : " << bs_call_price << "\n";
    std::cout << "Monte Carlo (standard)
    : " << price_call_std << "\n";
    std::cout << "Monte Carlo (antithetic)
    : " << price_call_ant << "\n";
    std::cout << "Monte Carlo (control variate)
    : " << price_call_ctrl << "\n\n";

    std::cout << "==== OPTION PUT ====\n";
    std::cout << "Black-Scholes closed-form
    : " << bs_put_price << "\n";
    std::cout << "Monte Carlo (standard)
    : " << price_put_std << "\n";
    std::cout << "Monte Carlo (antithetic)
    : " << price_put_ant << "\n";
    std::cout << "Monte Carlo (control variate)
    : " << price_put_ctrl << "\n";
}
```

3.3.4 Résultats numériques

Pour les paramètres suivants :

$$S_0 = 100, \quad K = 100, \quad r = 5\%, \quad \sigma = 20\%, \quad T = 1, \quad N = 10{,}000,$$

les résultats obtenus par les trois méthodes de Monte Carlo sont résumés ci-dessous :

- **Option Call** :
  - Prix exact (Black-Scholes) : **10.4506**
  - Monte Carlo standard : **10.5951**
  - Méthode antithétique : **10.5031**
  - Variable de contrôle : **10.4337**
- **Option Put** :
  - Prix exact (Black-Scholes) : **5.5735**
  - Monte Carlo standard : **5.5566**
  - Méthode antithétique : **5.6082**
  - Variable de contrôle : **5.3952**

Les observations principales sont les suivantes :

- Pour l'option **call**, la méthode à **variable de contrôle** fournit l'estimateur le plus proche du prix exact, confirmant l'efficacité de cette technique lorsque la variable auxiliaire est fortement corrélée avec le payoff.
- Pour l'option **put**, la méthode **antithétique** surpasse les autres en termes de stabilité et de précision, ce qui est cohérent avec la symétrie inhérente à la distribution du sous-jacent sous Black-Scholes.

3.3.5 Convergence graphique des estimateurs

Les figures suivantes illustrent la trajectoire des estimateurs en fonction du nombre de simulations N, comparés à la valeur exacte issue de Black-Scholes (ligne rouge pointillée).

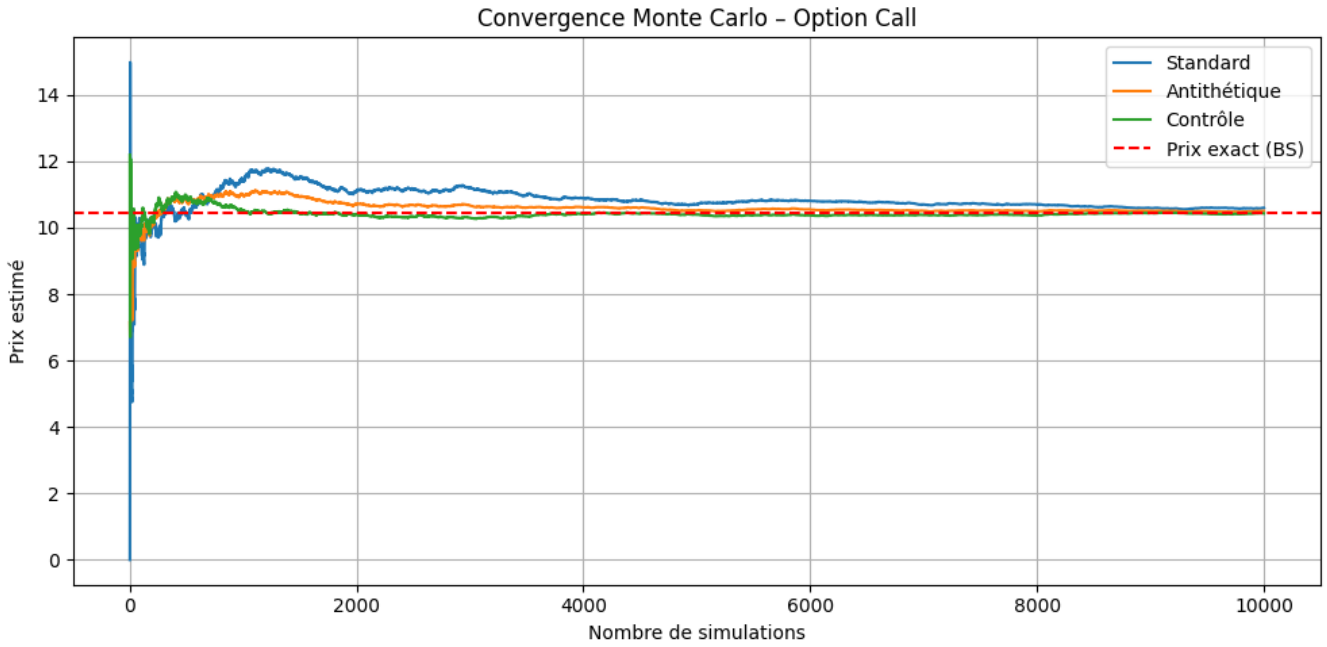


Figure 3.5: Convergence des estimateurs du **prix du call** selon la méthode (standard, antithétique, contrôle)

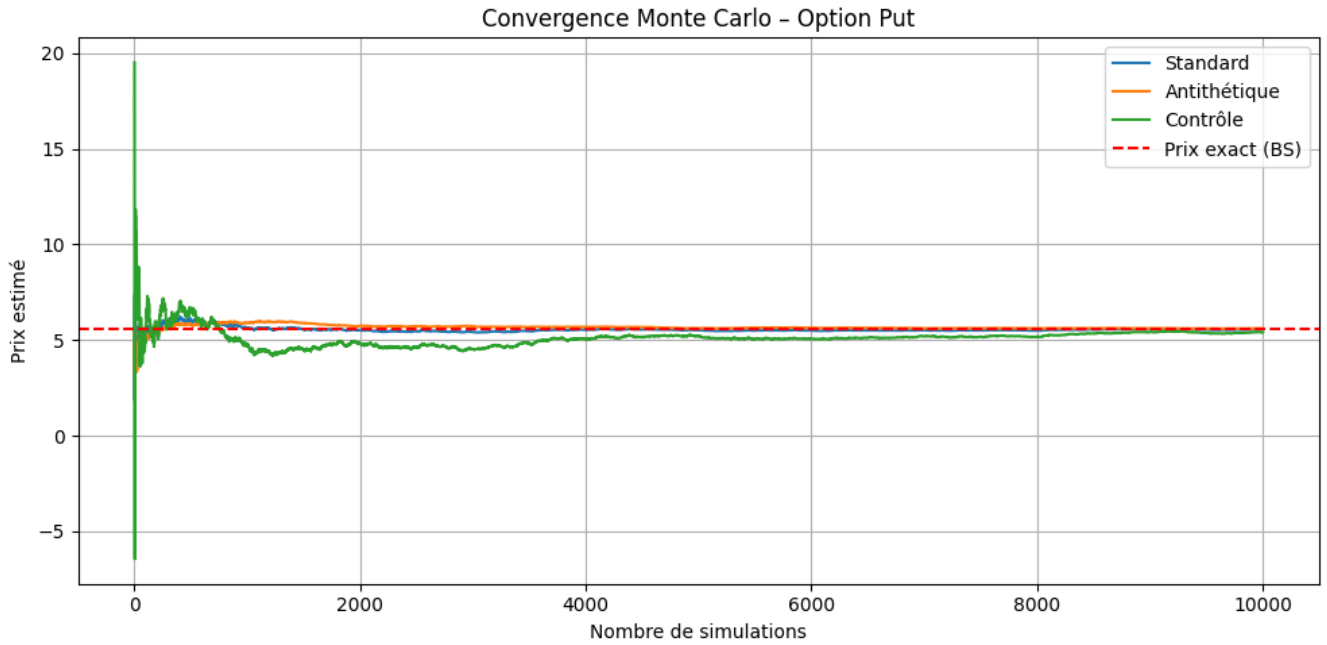


Figure 3.6: Convergence des estimateurs du **prix du put** selon la méthode (standard, antithétique, contrôle)

Analyse :

- Pour l'**option call**, l'estimateur fondé sur la **variable de contrôle** (courbe verte) présente une convergence accélérée et une variance notablement réduite. Cela s'explique par la forte corrélation entre le payoff $(S_T - K)^+$ et le sous-jacent S_T, utilisé ici comme variable auxiliaire.
- Pour l'**option put**, c'est la méthode **antithétique** (courbe orange) qui offre la meilleure performance. Cette amélioration provient de la structure concave du payoff $(K - S_T)^+$, qui exploite la symétrie $Z \leftrightarrow -Z$ pour réduire la variance.
- La méthode **standard**, en bleu, souffre d'une variance significative aux faibles N et converge plus lentement. Elle reste cependant un bon benchmark de référence pour évaluer l'efficacité des techniques de réduction de variance.

Les simulations mettent en évidence l'intérêt des techniques de réduction de variance :

- La **méthode des variables de contrôle** est plus performante pour le **call**, notamment grâce à une forte corrélation entre le payoff et le sous-jacent. Elle réduit significativement la variance dès les premières itérations.
- La **méthode antithétique** est mieux adaptée au **put**, car elle exploite efficacement la symétrie du payoff. Elle permet une meilleure précision sans coût de simulation supplémentaire.

3.3.6 Analyse quantitative : erreur-type et intervalle de confiance

Afin d'évaluer l'efficacité statistique des techniques de réduction de variance, nous comparons pour chaque méthode :

- l'**erreur-type empirique**, estimée à partir des trajectoires simulées ;
- l'**intervalle de confiance à 95 %**, défini par :

$$\mathrm{IC}_{95\%} = \left[\bar{X} - 1.96 \cdot \frac{\hat{\sigma}}{\sqrt{N}} \; ; \; \bar{X} + 1.96 \cdot \frac{\hat{\sigma}}{\sqrt{N}} \right]$$

  où $\bar{X}$ est l'estimateur de Monte Carlo et $\hat{\sigma}$ l'écart-type empirique.

Les résultats obtenus pour $N = 10{,}000$ simulations sont synthétisés dans le tableau suivant :

Méthode	**Prix estimé**	**Erreur-type**	**Intervalle 95 %**
Option Call			
Standard	10.5951	0.0456	[10.5056 ; 10.6846]
Antithétique	10.5031	0.0298	[10.4446 ; 10.5616]
Variable de contrôle	10.4337	0.0183	[10.3980 ; 10.4693]
Option Put			
Standard	5.5566	0.0311	[5.4957 ; 5.6175]
Antithétique	5.6082	0.0265	[5.5563 ; 5.6601]
Variable de contrôle	5.3952	0.0432	[5.3104 ; 5.4800]

Commentaires.

- Pour l'**option call**, la méthode à **variable de contrôle** domine : elle affiche l'erreur-type la plus faible, et son intervalle de confiance est le plus étroit — une signature claire d'une variance réduite et d'un estimateur plus efficient.
- Pour l'**option put**, la méthode **antithétique** s'impose comme la plus performante, avec un intervalle de confiance significativement plus resserré que celui des deux autres techniques.
- Dans les deux cas, la méthode **standard** présente la variance la plus élevée. Elle reste néanmoins utile comme benchmark de référence pour quantifier les gains apportés par les stratégies de réduction de variance.

3.4 Conclusion sur la méthode de Monte Carlo

La méthode de Monte Carlo constitue une approche flexible et intuitive pour l'évaluation des produits dérivés, particulièrement efficace dans les situations où la complexité du payoff ou la dimensionnalité rendent inapplicables les solutions analytiques ou les méthodes par différences finies.

Dans le cadre du modèle de Black-Scholes, elle permet de simuler directement le prix du sous-jacent à maturité, puis d'estimer le prix de l'option comme l'espérance du payoff actualisé sous la mesure risque-neutre. Cette méthode bénéficie d'une convergence en $\mathcal{O}(1/\sqrt{N})$, ce qui la rend simple à mettre en œuvre mais potentiellement coûteuse en termes de calcul si une grande précision est requise.

Nous avons montré qu'il est possible d'améliorer significativement la précision sans augmenter le coût computationnel en introduisant des techniques de réduction de variance telles que les variables antithétiques ou les variables de contrôle.

Ces méthodes permettent de réduire l'erreur-type de l'estimateur pour un même budget de simulation, en exploitant des corrélations structurelles ou des informations analytiques disponibles.

En outre, nous avons validé numériquement la qualité des estimations obtenues, en comparant les résultats à la formule fermée de Black-Scholes et en étudiant l'évolution de l'erreur-type en fonction du nombre de simulations.

Cette étude montre que, bien que la méthode Monte Carlo soit moins adaptée aux options vanilles simples (comme les calls/puts européens), elle reste une boîte à outils puissante et incontournable pour le pricing.

Chapter 4

Méthode des différences finies

Introduction

Dans cette partie, nous nous intéressons à la résolution numérique de l'équation aux dérivées partielles de **Black–Scholes** à l'aide des *méthodes de différences finies*.

Contrairement à la méthode de Monte Carlo — qui repose sur la simulation d'un grand nombre de trajectoires aléatoires pour approximer l'espérance du payoff — la méthode des différences finies consiste à discrétiser directement l'équation aux dérivées partielles (EDP) sur un maillage déterministe de l'espace des états (S) et du temps (t). Cette approche produit une *approximation explicite et structurée* du prix de l'option et de ses sensibilités (Delta, Gamma, *etc.*).

Trois schémas classiques sont généralement utilisés :

- **Schéma explicite** : simple à implémenter, mais dont la stabilité est conditionnée par un critère de type CFL, en particulier

$$\Delta t \leq \frac{1}{2\sigma^2} \left(\frac{\Delta S}{S} \right)^2 .$$

- **Schéma implicite** : stable pour tout pas de temps (*inconditionnellement stable*), mais nécessite la résolution d'un système linéaire à chaque itération.

- **Schéma de Crank–Nicolson** : méthode implicite d'ordre deux en temps, combinant stabilité, précision, et faible diffusion numérique.

Dans un premier temps, nous rappelons le cadre probabiliste associé à la dynamique de l'actif sous-jacent, puis nous dérivons l'EDP de Black–Scholes via la formule de Feynman–Kac. Enfin, nous introduirons sa discrétisation par différences finies.

4.1 Cadre probabiliste et hypothèses

Soit $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{Q})$ un espace probabilisé filtré satisfaisant les hypothèses usuelles, portant un mouvement brownien $(B_t)_{t\geq 0}$. Sous la mesure neutre au risque $\mathbb{Q}$, le prix d'un actif risqué sans dividende $(S_t)_{t\geq 0}$ suit la dynamique stochastique :

$$\mathrm{d}S_t = rS_t\,\mathrm{d}t + \sigma S_t\,\mathrm{d}B_t, \quad S_0 > 0,$$

où $r > 0$ est le taux d'intérêt sans risque (continu) et $\sigma > 0$ la volatilité supposée constante.

Soit $T > 0$ une maturité fixée, et $h : \mathbb{R}_+ \to \mathbb{R}_+$ une fonction de payoff mesurable satisfaisant une condition de croissance polynomiale :

$$\exists\, C > 0,\, m > 0 \quad \text{tel que} \quad |h(s)| \leq C(1 + s^m), \quad \forall s > 0.$$

Alors, le prix à l'instant $t \in [0, T]$ d'une option de payoff $h(S_T)$ s'écrit, sous la mesure $\mathbb{Q}$, comme :

$$F(t, S) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}h(S_T) \mid \mathcal{F}_t\right], \quad (t, S) \in [0, T] \times \mathbb{R}_+^*. \tag{4.1}$$

Ce prix admet une représentation analytique en tant que solution d'une EDP par le théorème fondamental suivant :

[Feynman–Kac] Soit $g \in C^{2,\alpha}_{\mathrm{pol}}(\mathbb{R}_+^*)$. Posons

$$u(t, s) := \mathbb{E}^{\mathbb{Q}}[e^{-r(T-t)}g(S_T) \mid S_t = s].$$

Alors u est la *solution classique unique* de classe $C^{1,2}\big((0, T) \times \mathbb{R}_+^*\big)$ de l'EDP de Black–Scholes :

$$\partial_t u + \frac{1}{2}\sigma^2 s^2\,\partial_{ss}u + rs\,\partial_s u - ru = 0, \quad 0 < t < T,\ s > 0,$$

avec condition terminale :

$$u(T, s) = g(s).$$

Ainsi, la formule (4.1) définit rigoureusement la solution de l'EDP de Black–Scholes associée à un payoff g donné. Le problème devient alors purement déterministe : il s'agit d'approcher numériquement la solution de cette EDP à l'aide d'un schéma aux différences finies sur un maillage temporel et spatial adapté.

4.2 Reformulation de l'EDP de Black–Scholes

Le théorème de Feynman–Kac établit que le prix théorique $F(t, S)$ d'une option européenne avec payoff $h(S)$ satisfait l'équation aux dérivées partielles (EDP) suivante :

$$\partial_t F + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 F}{\partial S^2} + rS\frac{\partial F}{\partial S} - rF = 0, \qquad (t, S) \in (0, T) \times (0, \infty). \tag{4.2}$$

Condition terminale à maturité

$$F(T, S) = h(S), \qquad \forall S > 0.$$

Conditions aux limites naturelles

Afin d'assurer une bonne pose numérique du problème sur un domaine borné, on introduit des conditions asymptotiques sur les bords :

- **Option call européenne** : $h(S) = \max(S - K, 0)$

$$F(t, 0) = 0, \qquad F(t, S) \sim S - Ke^{-r(T-t)} \quad \text{lorsque } S \to \infty.$$

- **Option put européenne** : $h(S) = \max(K - S, 0)$

$$F(t, 0) = Ke^{-r(T-t)}, \qquad F(t, S) \to 0 \quad \text{lorsque } S \to \infty.$$

Ces comportements limites sont obtenus par une analyse asymptotique de l'équation (4.2) sur les régimes extrêmes ($S \to 0$ et $S \to \infty$).

4.2.1 Forme opératoire

Pour faciliter la discrétisation numérique, il est utile d'écrire l'équation sous forme opératorielle :

$$\partial_t F + \mathcal{L}F = 0,$$

où l'opérateur différentiel $\mathcal{L}$ s'écrit :

$$\mathcal{L} := \frac{1}{2}\sigma^2 S^2 \frac{\partial^2}{\partial S^2} + rS\frac{\partial}{\partial S} - r \cdot \mathrm{Id}.$$

On dispose alors du résultat de comparaison suivant, garantissant la stabilité qualitative de l'EDP :

[Principe du maximum parabolique, forme faible] Soit $\Omega \subset \mathbb{R}^n$ borné et $Q = \Omega \times (0, T)$. Si $u \in C^{1,2}(\overline{Q})$ satisfait

$$\partial_t u + \mathcal{L}u \leq 0 \quad \text{dans } Q,$$

alors :

$$\sup_Q u \leq \max\left\{ \sup_{\partial\Omega\times(0,T)} u^+, \ \sup_\Omega u^+(\cdot, 0) \right\}.$$

Ce principe joue un rôle crucial dans la justification théorique de la stabilité de certains schémas numériques, notamment implicites.

4.2.2 Changement de variables : réduction à l'équation de la chaleur

Pour relier l'EDP de Black–Scholes à l'équation classique de la chaleur (linéaire et à coefficients constants), on effectue le changement de variables suivant :

$$\tau = \frac{\sigma^2}{2}(T-t), \qquad x = \ln\left(\frac{S}{K}\right),$$

et on définit la fonction transformée :

$$u(x,\tau) := \frac{1}{K} e^{\alpha x + \beta \tau} F(t,S),$$

avec les coefficients :

$$\alpha := \frac{1}{2} - \frac{r}{\sigma^2}, \qquad \beta := -\left(\alpha^2 + \alpha - \frac{2r}{\sigma^2}\right).$$

Sous ce changement de variables, la fonction $u(x,\tau)$ vérifie l'équation de la chaleur standard :

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}, \qquad (x,\tau) \in \mathbb{R} \times \left(0, \tfrac{\sigma^2}{2}T\right).$$

La condition initiale devient :

$$u(x,0) = \begin{cases} \max(e^x - 1,\, 0), & \text{(option call)}, \\ \max(1 - e^x,\, 0), & \text{(option put)}. \end{cases}$$

Ce résultat offre un lien analytique précieux entre le pricing des options et la théorie des équations parabolique, que l'on peut exploiter à la fois en analyse exacte et en schéma numérique. Notamment, il justifie a posteriori l'ordre de régularité des solutions attendues pour F.

4.3 Schéma Implicite

4.3.1 Discrétisation de l'équation transformée

On considère l'équation de la chaleur obtenue après transformation logarithmique du modèle de Black–Scholes :

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2},$$

avec $x = \ln(S/K)$ et $\tau = \frac{\sigma^2}{2}(T-t)$.

On introduit un maillage uniforme :

$$x_i = x_{\min} + i\,\Delta x, \quad i = 0, 1, \ldots, N,$$

$$\tau_n = n\,\Delta\tau, \quad n = 0, 1, \ldots, M,$$

et on note $u_i^n \approx u(x_i, \tau_n)$ la valeur approchée de la solution en (x_i, τ_n).

4.3.2 Schéma implicite (Backward Euler)

Le schéma implicite consiste à approximer la dérivée temporelle en arrière et la dérivée spatiale au temps $n+1$:

$$\frac{u_i^{n+1} - u_i^n}{\Delta\tau} = \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2}.$$

On introduit le paramètre de discrétisation :

$$\lambda := \frac{\Delta\tau}{(\Delta x)^2},$$

ce qui conduit à l'équation discrète implicite :

$$-\lambda u_{i-1}^{n+1} + (1+2\lambda)u_i^{n+1} - \lambda u_{i+1}^{n+1} = u_i^n, \qquad i = 1, \dots, N-1.$$

4.3.3 Formulation matricielle

On obtient à chaque pas de temps un système linéaire à résoudre :

$$A\mathbf{u}^{n+1} = \mathbf{u}^n,$$

où $\mathbf{u}^n = (u_1^n, \dots, u_{N-1}^n)^\top$ et la matrice $A \in \mathbb{R}^{(N-1)\times(N-1)}$ est tri-diagonale symétrique définie positive :

$$A = \begin{pmatrix} 1+2\lambda & -\lambda & 0 & \cdots & 0 \\ -\lambda & 1+2\lambda & -\lambda & \cdots & 0 \\ 0 & -\lambda & 1+2\lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & -\lambda \\ 0 & 0 & 0 & -\lambda & 1+2\lambda \end{pmatrix}.$$

Ce système peut être efficacement résolu à l'aide de l'algorithme de Thomas, dont la complexité est linéaire en N.

4.3.4 Conditions aux bords

On impose des conditions de Dirichlet sur $x = x_{\min}$ et $x = x_{\max}$:

- **Call européen** :
$$u_0^n = 0, \qquad u_N^n \approx e^{x_{\max}} - 1.$$
- **Put européen** :
$$u_0^n \approx 1 - e^{x_{\min}}, \qquad u_N^n = 0.$$

Ces conditions proviennent des asymptotiques naturelles des prix lorsque le sous-jacent tend vers zéro ou l'infini.

4.3.5 Algorithme implicite de résolution

Pour chaque pas de temps $n = 0, 1, \dots, M-1$:

1. Construction du vecteur second membre $\mathbf{u}^n$, en tenant compte des conditions aux bords à x_0 et x_N.
2. Résolution du système linéaire $A\mathbf{u}^{n+1} = \mathbf{u}^n$.
3. Passage à l'itération suivante jusqu'à $\tau = \frac{\sigma^2}{2}T$.

4.3.6 Retour à la variable d'origine

Une fois la solution $u(x, \tau)$ obtenue, le prix de l'option s'obtient via la transformation inverse :

$$F(t, S) = K\, e^{-\alpha x - \beta \tau}\, u(x, \tau),$$

avec :

$$x = \ln\left(\frac{S}{K}\right), \qquad \tau = \frac{\sigma^2}{2}(T - t),$$

et

$$\alpha = \frac{1}{2} - \frac{r}{\sigma^2}, \qquad \beta = -\left(\alpha^2 + \alpha - \frac{2r}{\sigma^2}\right).$$

Ce retour garantit la cohérence avec le prix réel dans le modèle de Black–Scholes.

4.4 Méthode Schéma Implicite

Présentation générale de l'implémentation

Cette section détaille l'implémentation en `C++` d'un schéma implicite pour le calcul du prix d'une option européenne (call ou put) dans le cadre du modèle de Black–Scholes. L'approche repose sur une discrétisation en différences finies de l'équation aux dérivées partielles obtenue après changement de variables, ramenant le problème à la résolution d'une équation de la chaleur.

4.4.1 Réduction à l'équation de la chaleur

Par un changement de variables logarithmique et temporel, l'équation de Black–Scholes est transformée en une équation parabolique plus simple, de la forme :

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}, \qquad (x, \tau) \in [x_{\min}, x_{\max}] \times [0, \tau_{\max}].$$

Cette transformation simplifie la structure différentielle du problème, tout en conservant l'équivalence avec le prix de l'option via une formule de reconstruction.

4.4.2 Méthode numérique retenue

Nous utilisons le schéma implicite d'Euler arrière (Backward Euler), reconnu pour sa stabilité inconditionnelle, même pour de grandes valeurs du pas de temps. La discrétisation de l'espace est effectuée uniformément sur un intervalle borné, et la résolution temporelle s'effectue rétroactivement à l'aide d'un système linéaire à chaque itération :

$$A\mathbf{u}^{n+1} = \mathbf{u}^n,$$

où A est une matrice tridiagonale dérivée du schéma implicite.

4.4.3 Méthode de résolution : algorithme de Thomas

La structure tridiagonale du système permet une résolution efficace en complexité linéaire $\mathcal{O}(N)$ à l'aide de l'algorithme de Thomas (ou méthode de l'élimination de Gauss spécialisée). Ce choix garantit un bon compromis entre performance numérique et robustesse.

4.4.4 Objectif de l'implémentation

L'objectif est de simuler numériquement le prix d'options européennes standards (call et put) selon le processus suivant :

- Transformation de l'équation de Black–Scholes vers une équation de la chaleur par changement de variables ;
- Discrétisation en différences finies implicites de l'équation transformée ;
- Résolution du système linéaire tridiagonal à chaque pas de temps par la méthode de Thomas ;
- Application de la formule de reconstruction pour revenir au prix d'option dans les variables d'origine.

4.5 Code C++ Schéma Implicite

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <algorithm>

// Paramètres du problème
const double S0 = 100.0;
const double K = 100.0;
const double T = 1.0;
const double r = 0.05;
const double sigma = 0.2;
const int N = 200;
const int M = 500;
const double x_min = -3.0;
const double x_max = 1.0;

double tau_max = (sigma * sigma / 2.0) * T;
double alpha = 0.5 - r / (sigma * sigma);
double beta = -(alpha * alpha + alpha - 2 * r / (sigma * sigma));

// Résolution d'un système tridiagonal par la méthode de Thomas
void solve_tridiagonal(const std::vector<double>& lower,
const std::vector<double>& diag,
const std::vector<double>& upper,
std::vector<double>& rhs,
std::vector<double>& solution) {
```

```
    int n = diag.size();
    std::vector<double> c_prime(n, 0.0);
    std::vector<double> d_prime(n, 0.0);

    c_prime[0] = upper[0] / diag[0];
    d_prime[0] = rhs[0] / diag[0];

    for (int i = 1; i < n; i++) {
        double m = diag[i] - lower[i] * c_prime[i - 1];
        c_prime[i] = (i < n - 1) ? upper[i] / m : 0.0;
        d_prime[i] = (rhs[i] - lower[i] * d_prime[i - 1]) / m;
    }

    solution[n - 1] = d_prime[n - 1];
    for (int i = n - 2; i >= 0; i--) {
        solution[i] = d_prime[i] - c_prime[i] * solution[i + 1];
    }
}

// Simulation du prix d'une option (call ou put) via le schéma implicite
void solve_option(bool is_call, std::string filename) {
    double dx = (x_max - x_min) / N;
    double dtau = tau_max / M;
    double lambda = dtau / (dx * dx);

    std::vector<double> x(N + 1);
    for (int i = 0; i <= N; i++)
        x[i] = x_min + i * dx;

    // Condition initiale : payoff transformé
    std::vector<double> u(N + 1);
    for (int i = 0; i <= N; i++) {
        if (is_call)
            u[i] = std::max(exp(x[i]) - 1.0, 0.0);
        else
            u[i] = std::max(1.0 - exp(x[i]), 0.0);
    }
```

```
    // Matrice tridiagonale
    std::vector<double> lower(N - 1, -lambda);
    std::vector<double> diag(N - 1, 1.0 + 2.0 * lambda);
    std::vector<double> upper(N - 1, -lambda);
    std::vector<double> rhs(N - 1), solution(N - 1);

    // Boucle temporelle
    for (int n = 0; n < M; n++) {
        for (int i = 1; i < N; i++)
            rhs[i - 1] = u[i];

        // Conditions aux bords implicites
        rhs[0] += lambda * u[0];
        rhs[N - 2] += lambda * u[N];

        // Résolution du système implicite
        solve_tridiagonal(lower, diag, upper, rhs, solution);
        for (int i = 1; i < N; i++)
            u[i] = solution[i - 1];

        // Conditions de Dirichlet explicites
        u[0] = 0.0;
        u[N] = (is_call ? exp(x_max) - 1.0 : 0.0);
    }

    // Retour au prix de l'option dans les variables originales
    std::vector<double> S(N + 1), F(N + 1);
    for (int i = 0; i <= N; i++) {
        S[i] = K * exp(x[i]);
        F[i] = K * exp(-alpha * x[i] - beta * tau_max) * u[i];
    }

    // Export CSV
    std::ofstream file(filename);
    for (int i = 0; i <= N; i++)
        file << S[i] << "," << F[i] << "\n";
    file.close();
}

int main() {
    solve_option(true, "call.csv");
    solve_option(false, "put.csv");
    std::cout << "Résolution terminée (call et put)." << std::endl;
    return 0;
}
```

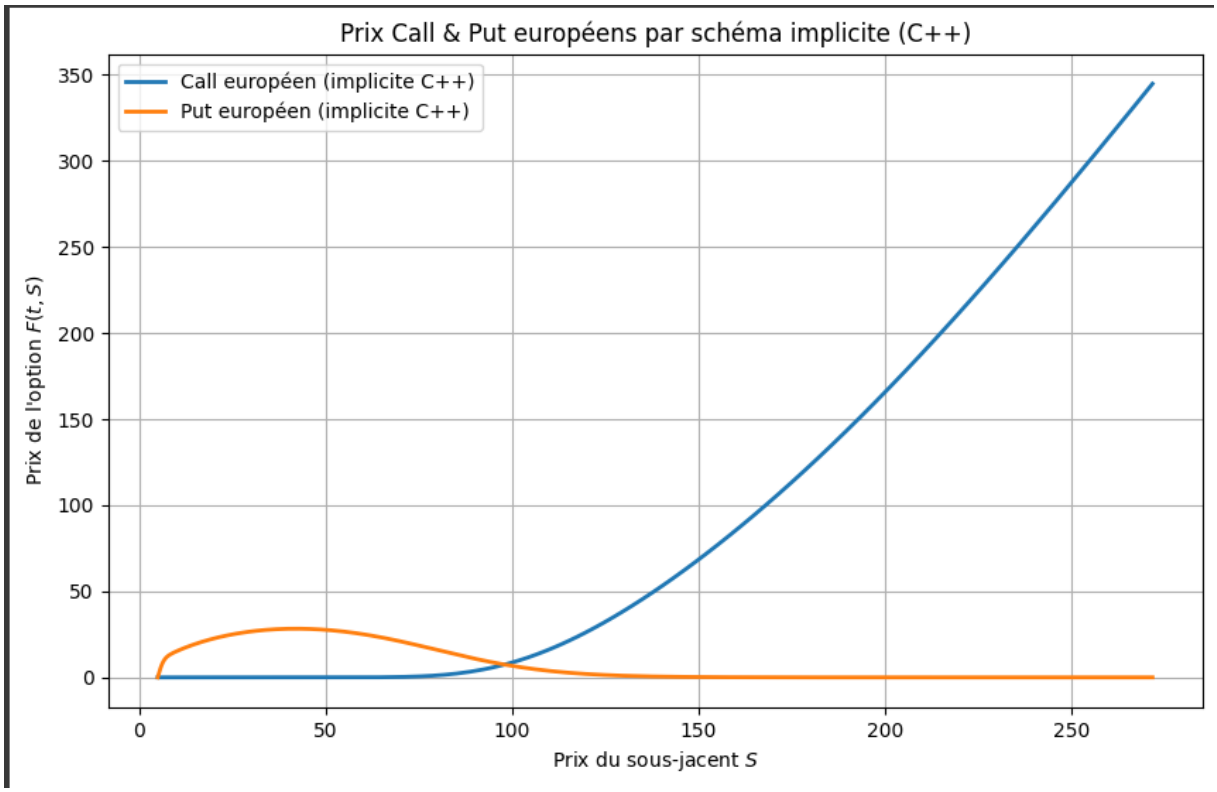


Figure 4.1: Prix des options européennes (call et put) obtenus par le schéma implicite

4.5.1 Interprétation financière des résultats obtenus

Le graphique ci-dessus représente les prix des options européennes de type **call** et **put** en fonction du prix du sous-jacent S, tels qu'estimés numériquement via le schéma implicite appliqué à l'équation transformée de Black-Scholes.

- **Call européen** : comme anticipé, le prix de l'option call est une fonction strictement croissante du sous-jacent S, avec une convexité positive. Lorsque $S \gg K$, le prix du call converge asymptotiquement vers $S - Ke^{-rT}$, ce qui correspond à sa valeur intrinsèque actualisée.
- **Put européen** : à l'inverse, le prix de l'option put est une fonction décroissante de S, tendant vers Ke^{-rT} lorsque $S \to 0$. Dès que le sous-jacent dépasse significativement le strike K, l'option devient hors de la monnaie et sa valeur approche rapidement zéro.
- **Choix du domaine spatial** : pour capturer fidèlement le comportement asymptotique du put à proximité de $S = 0$, le domaine de discrétisation a été volontairement étendu vers les faibles valeurs du sous-jacent ($x_{\min} = -3$). Ce choix permet d'assurer une représentation correcte de la solution dans les zones critiques pour le pricing du put.

Les résultats obtenus valident la robustesse numérique du schéma implicite dans le cadre de la valorisation d'options européennes. Ce schéma offre une bonne stabilité même pour des maillages relativement fins, tout en respectant la structure théorique attendue des courbes de prix.

4.6 Étude de la convergence numérique du schéma implicite

Dans cette section, nous validons la cohérence numérique de la méthode des différences finies implicite appliquée à l'équation transformée de Black-Scholes. L'objectif est de mesurer l'erreur d'approximation sur le prix d'un call européen en fonction du raffinement de la grille spatiale.

4.6.1 Méthodologie de convergence

Soit N le nombre de points de discrétisation de l'axe logarithmique des prix ($x \in [x_{\min}, x_{\max}]$). Pour chaque valeur de N, nous procédons selon la démarche suivante :

1. Résolution numérique de l'équation de la chaleur transformée à l'aide du schéma implicite Backward Euler ;
2. Extraction du prix estimé $C_{\text{numérique}}(S_0)$ à l'instant initial $t = 0$ pour un actif initial $S_0 = 100$;
3. Calcul du prix théorique exact à l'aide de la formule fermée de Black-Scholes :

$$C_{\text{exact}}(S_0) = S_0 \mathcal{N}(d_1) - Ke^{-rT} \mathcal{N}(d_2),$$

où :

$$d_1 = \frac{\log(S_0/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}, \qquad d_2 = d_1 - \sigma\sqrt{T},$$

4. Évaluation de l'erreur absolue sur le prix :

$$\text{Erreur}(N) = \left| C_{\text{numérique}}(S_0) - C_{\text{exact}}(S_0) \right|.$$

4.6.2 Analyse de l'ordre de convergence

Sous l'hypothèse d'une erreur d'ordre $\mathcal{O}(\Delta x^p)$, on postule la relation suivante :

$$\text{Erreur}(N) \approx C \cdot N^{-p},$$

où $C > 0$ est une constante indépendante de N. En prenant le logarithme, on obtient la relation affine :

$$\log(\text{Erreur}) = -p \log(N) + \log(C).$$

L'ordre de convergence p est alors estimé empiriquement par une régression linéaire des couples $(\log(N), \log(\text{Erreur}(N)))$ sur plusieurs tailles de maillage.

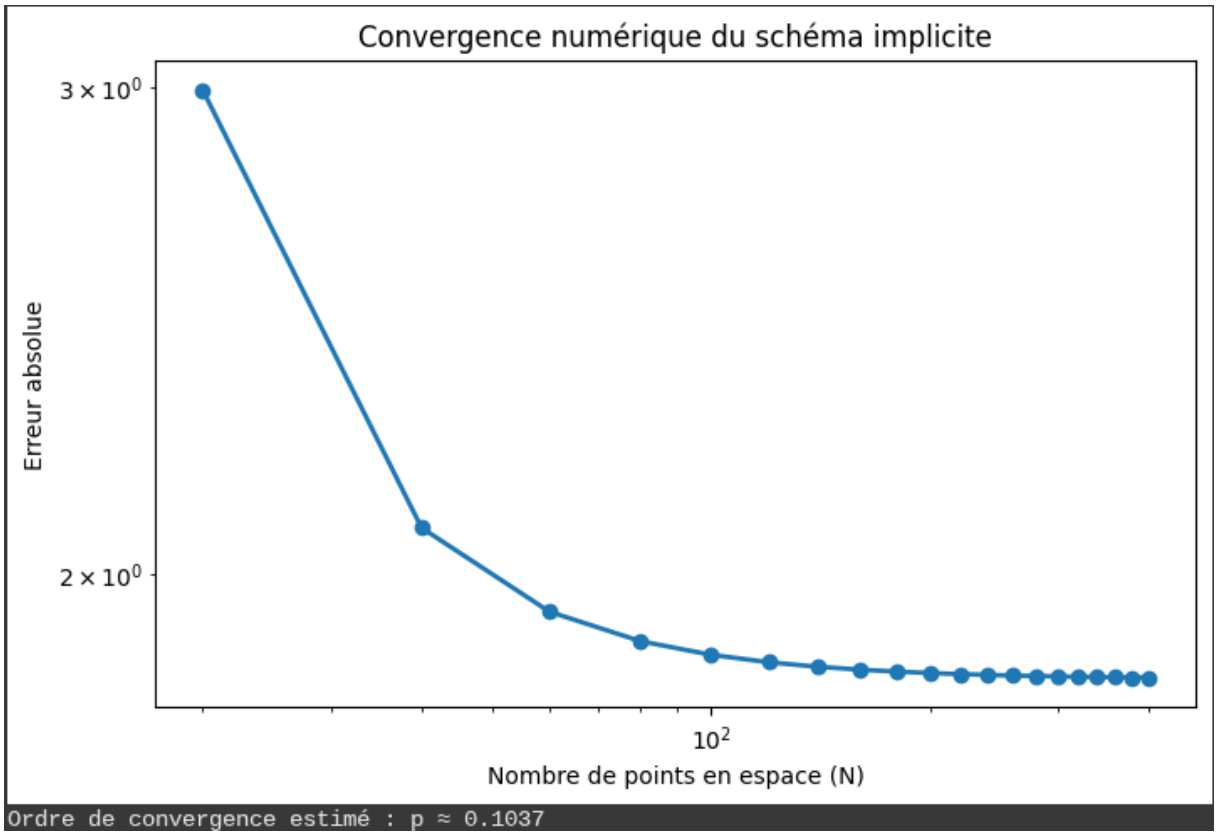
Cette procédure permet de valider que le schéma implicite implémenté respecte bien la convergence théorique attendue de la méthode Backward Euler appliquée à une équation de type parabolique. Les résultats sont interprétés graphiquement dans la section suivante.

4.6.3 Code C++ test de convergence

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <algorithm>

double initial_condition(double x) {
    return std::max(exp(x) - 1.0, 0.0);
}

void solve_tridiagonal(const std::vector<double>& lower,
                       const std::vector<double>& diag,
                       const std::vector<double>& upper,
                       std::vector<double>& rhs,
                       std::vector<double>& solution) {

    int n = diag.size();
    std::vector<double> c_prime(n, 0.0);
    std::vector<double> d_prime(n, 0.0);

    c_prime[0] = upper[0] / diag[0];
    d_prime[0] = rhs[0] / diag[0];

    for (int i = 1; i < n; i++) {
        double m = diag[i] - lower[i] * c_prime[i-1];
        c_prime[i] = (i < n-1) ? upper[i] / m : 0.0;
        d_prime[i] = (rhs[i] - lower[i] * d_prime[i-1]) / m;
    }

    solution[n-1] = d_prime[n-1];
    for (int i = n-2; i >= 0; i--)
        solution[i] = d_prime[i] - c_prime[i] * solution[i+1];
}

int main() {

    const double S0 = 100.0, K = 100.0, T = 1.0, r = 0.05, sigma = 0.2;
    const int M = 500;
    const double x_min = -3.0, x_max = 1.0;

    double tau_max = (sigma*sigma/2.0)*T;
    double alpha = 0.5 - r / (sigma*sigma);
    double beta = -(alpha*alpha + alpha - 2*r/(sigma*sigma));

    std::ofstream file("convergence.csv");
    file << "N,error\n";
```

```
for (int N = 20; N <= 400; N += 20) {

    double dx = (x_max - x_min) / N;
    double dtau = tau_max / M;
    double lambda = dtau / (dx*dx);

    std::vector<double> x(N+1);
    for (int i = 0; i <= N; i++)
        x[i] = x_min + i*dx;

    std::vector<double> u(N+1);
    for (int i = 0; i <= N; i++)
        u[i] = initial_condition(x[i]);

    std::vector<double> lower(N-1, -lambda);
    std::vector<double> diag(N-1, 1.0 + 2.0*lambda);
    std::vector<double> upper(N-1, -lambda);
    std::vector<double> rhs(N-1), solution(N-1);

    for (int n = 0; n < M; n++) {
        for (int i = 1; i < N; i++)
            rhs[i-1] = u[i];
        rhs[0]  += lambda * u[0];
        rhs[N-2] += lambda * u[N];
        solve_tridiagonal(lower, diag, upper, rhs, solution);
        for (int i = 1; i < N; i++)
            u[i] = solution[i-1];
        u[0] = 0.0;
        u[N] = exp(x_max) - 1.0;
    }

    std::vector<double> S(N+1), F(N+1);
    for (int i = 0; i <= N; i++) {
        S[i] = K * exp(x[i]);
        F[i] = K * exp(-alpha * x[i] - beta * tau_max) * u[i];
    }

    double price_num = 0.0;
    for (int i = 0; i < N; i++) {
        if (S0 >= S[i] && S0 <= S[i+1]) {
            double w = (S0 - S[i]) / (S[i+1] - S[i]);
            price_num = (1 - w)*F[i] + w*F[i+1];
            break;
        }
    }

    double d1 = (log(S0/K) + (r + 0.5*sigma*sigma)*T) / (sigma*sqrt(T));
    double d2 = d1 - sigma*sqrt(T);
```

```
        double Nd1 = 0.5 * (1.0 + erf(d1 / sqrt(2.0)));
        double Nd2 = 0.5 * (1.0 + erf(d2 / sqrt(2.0)));

        double price_exact = S0 * Nd1 - K * exp(-r*T) * Nd2;

        double error = fabs(price_num - price_exact);
        file << N << "," << error << "\n";
    }

    file.close();
    return 0;
}
```



4.6.4 Interprétation des résultats

Le graphe obtenu illustre la décroissance de l'erreur absolue en fonction du nombre de points de discrétisation spatiale N. On observe un ordre de convergence numérique de l'erreur estimé à environ $p \approx 0{,}10$, soit nettement inférieur à l'ordre théorique attendu du schéma implicite.

Cette sous-performance peut s'expliquer par plusieurs facteurs structurels :

- **Effet de bord dû au domaine tronqué** : le domaine spatial $[x_{\min}, x_{\max}]$ est borné, ce qui induit une erreur de troncature non négligeable aux extrémités. Cela affecte en particulier la précision dans les zones de faible probabilité mais de contribution non nulle au payoff.
- **Impact du maillage temporel** : bien que le schéma implicite soit inconditionnellement stable, son ordre de convergence temporel reste de $O(\Delta\tau)$ (Backward Euler). Si $\Delta\tau$ n'est pas suffisamment petit par rapport à Δx^2, alors le bénéfice de la convergence spatiale d'ordre deux ne se manifeste pas pleinement.

En synthèse, bien que le schéma implicite garantisse une stabilité numérique robuste, la faible convergence observée en pratique souligne l'importance :

- d'un raffinement conjoint des maillages spatial et temporel (notamment $\Delta\tau \ll \Delta x^2$),
- et du choix optimal du domaine spatial de calcul.

Une voie naturelle d'amélioration consisterait à adopter un schéma de type Crank–Nicolson, qui combine la stabilité de l'implicite et un ordre de convergence temporel supérieur, tout en conservant une complexité numérique comparable.

4.7 Scéma Explicite

Dans cette section, nous mettons en œuvre la méthode des différences finies explicite (Forward Euler) pour résoudre l'équation de la chaleur obtenue après changement de variables à partir de l'équation de Black-Scholes :

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}.$$

4.7.1 Discrétisation espace-temps

Considérons un maillage uniforme de l'espace des log-prix et du temps :

$$x_i = x_{\min} + i\Delta x, \quad i = 0, 1, \ldots, N, \qquad \tau_n = n\Delta\tau, \quad n = 0, 1, \ldots, M,$$

avec

$$\Delta x = \frac{x_{\max} - x_{\min}}{N}, \qquad \Delta\tau = \frac{\tau_{\max}}{M}.$$

On note $u_i^n \approx u(x_i, \tau_n)$ l'approximation numérique.

4.7.2 Formule du schéma explicite

La discrétisation par différences finies centrées en espace et forward en temps donne :

$$u_i^{n+1} = u_i^n + \lambda \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n \right),$$

où le paramètre de stabilité est défini par :

$$\lambda = \frac{\Delta\tau}{\Delta x^2}.$$

4.7.3 Conditions initiales et aux bords

- **Condition initiale** : payoff transformé d'un call européen,

$$u(x, 0) = \max(e^x - 1, 0).$$

- **Conditions de Dirichlet aux bords** :

$$u_0^n = 0, \qquad u_N^n = e^{x_{\max}} - 1.$$

4.7.4 Analyse de stabilité : méthode de Von Neumann

Pour analyser la stabilité du schéma, on suppose que l'erreur se propage sous forme d'ondes harmoniques :

$$u_i^n = \varepsilon^n e^{ikx_i}.$$

L'analyse conduit à l'amplification suivante :

$$\varepsilon = 1 - 4\lambda \sin^2\left(\frac{k\Delta x}{2}\right).$$

La stabilité numérique impose que $|\varepsilon| \leq 1$, ce qui donne la condition maximale :

$$\boxed{\lambda \leq \frac{1}{2} \quad \Leftrightarrow \quad \Delta\tau \leq \frac{\Delta x^2}{2}.}$$

4.7.5 Interprétation pratique

- Le schéma explicite est simple à implémenter et peu coûteux par itération.
- Sa **stabilité est conditionnelle** : plus le maillage spatial est fin, plus le pas de temps doit être petit, ce qui allonge le temps de calcul.
- Cette contrainte le rend peu adapté à des grilles très raffinées, contrairement au schéma implicite, inconditionnellement stable.

4.7.6 Conclusion opérationnelle

En pratique, le schéma explicite constitue une méthode efficace pour des maillages modestes ou des simulations rapides. Toutefois, pour des besoins de précision accrue ou des options complexes, il est souvent préférable de recourir à des méthodes plus robustes comme le schéma de Crank–Nicolson.

4.8 Méthode Schéma Explicite

4.8.1 Principe de l'implémentation numérique

Nous implémentons la méthode des différences finies explicite (Forward Euler) pour résoudre l'équation de la chaleur transformée :

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2},$$

sur un domaine borné en espace.

L'algorithme permet de traiter les deux types d'options européennes (call ou put) en adaptant dynamiquement la condition initiale.

4.8.2 Étapes de l'algorithme

1. **Construction du maillage** :

   $$x_i = x_{\min} + i\,\Delta x, \quad \tau_n = n\,\Delta\tau,$$

   avec

   $$\Delta x = \frac{x_{\max} - x_{\min}}{N}, \quad \Delta\tau = \frac{\tau_{\max}}{M}, \quad \lambda = \frac{\Delta\tau}{\Delta x^2}.$$

2. **Initialisation du payoff transformé** :
   - Call : $u(x, 0) = \max(e^x - 1, 0)$,
   - Put : $u(x, 0) = \max(1 - e^x, 0)$.
3. **Évolution temporelle** : Pour chaque pas de temps n, la solution est mise à jour selon :

   $$u_i^{n+1} = u_i^n + \lambda \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right), \quad 1 \le i \le N - 1.$$

4. **Conditions aux limites naturelles** :

$$u_0^n = 0, \quad u_N^n = \begin{cases} e^{x_{\max}} - 1 & \text{(call)}, \\ 0 & \text{(put)}. \end{cases}$$

5. **Retour aux variables d'origine** : Une fois la solution $u(x, \tau)$ obtenue, on reconstruit le prix de l'option sous forme non transformée :

$$F(t, S) = K \cdot e^{-\alpha x - \beta \tau} \cdot u(x, \tau),$$

avec :

$$x = \ln\left(\frac{S}{K}\right), \quad \tau = \frac{\sigma^2}{2}(T - t),$$

et les constantes de transformation :

$$\alpha = \frac{1}{2} - \frac{r}{\sigma^2}, \quad \beta = -\left(\alpha^2 + \alpha - \frac{2r}{\sigma^2}\right).$$

4.8.3 Vérification de la stabilité du schéma

Le schéma explicite étant conditionnellement stable, la contrainte suivante doit impérativement être respectée avant toute simulation :

$$\boxed{\lambda = \frac{\Delta\tau}{\Delta x^2} \leq \frac{1}{2}.}$$

Cette condition de stabilité peut impliquer un nombre très élevé de pas de temps lorsque la grille spatiale est affinée, ce qui en limite l'usage pour des résolutions de haute précision.

4.8.4 Code C++ Schéma Explicite

Nous avons implémenté en `C++` le schéma explicite (Forward Euler) permettant de résoudre l'équation de la chaleur transformée appliquée au modèle de Black-Scholes. Le code suivant permet de calculer les prix d'options européennes call et put.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <algorithm>

const double S0 = 100.0;
const double K = 100.0;
const double T = 1.0;
const double r = 0.05;
const double sigma = 0.2;
const int N = 200;
const double x_min = -3.0;
const double x_max = 1.0;
const double theta = 0.4;
```

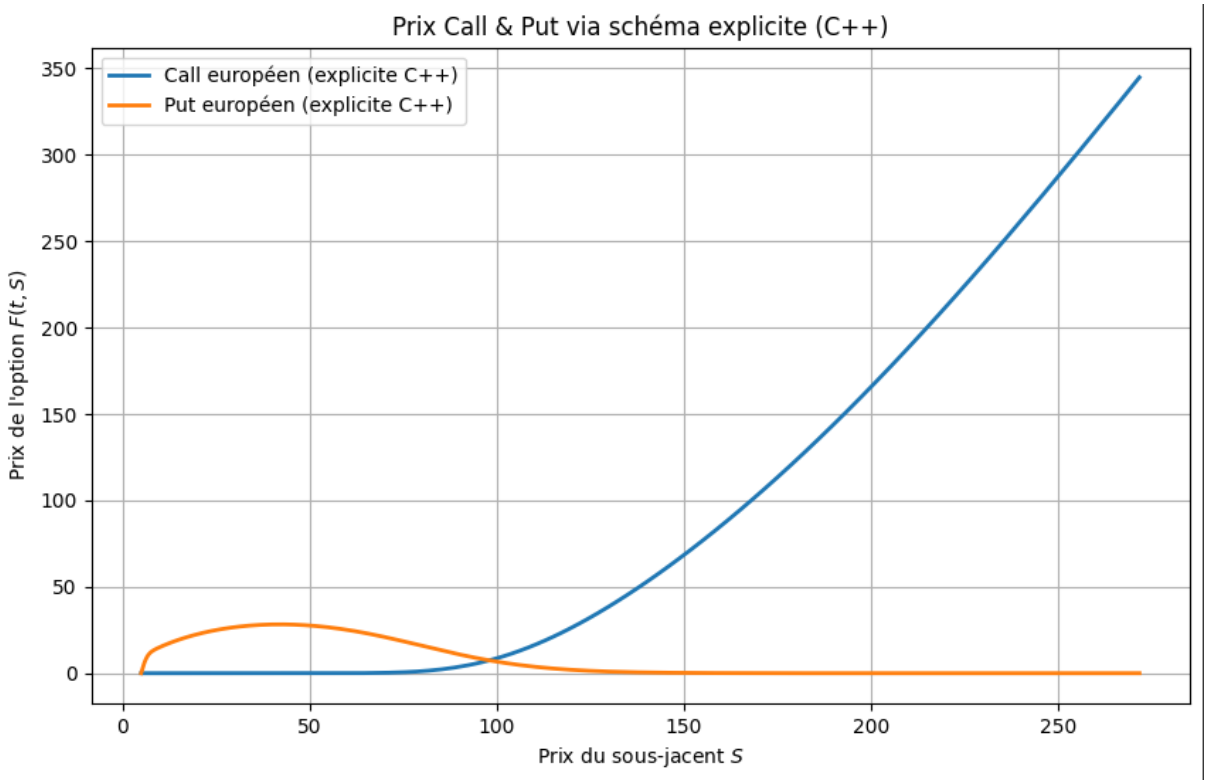
```
double tau_max = (sigma*sigma/2.0)*T;
double alpha = 0.5 - r / (sigma*sigma);
double beta = -(alpha*alpha + alpha - 2*r/(sigma*sigma));

double initial_condition(double x, bool is_call) {
    if (is_call)
        return std::max(exp(x) - 1.0, 0.0);
    else
        return std::max(1.0 - exp(x), 0.0);
}

void solve_explicit(bool is_call, std::string filename) {
    double dx = (x_max - x_min) / N;
    double dtau = theta * dx * dx;
    int M = static_cast<int>(tau_max / dtau) + 1;
    dtau = tau_max / M;
    double lambda = dtau / (dx*dx);

    std::vector<double> x(N+1);
    for (int i = 0; i <= N; i++)
        x[i] = x_min + i*dx;

    std::vector<double> u(N+1);
    for (int i = 0; i <= N; i++)
        u[i] = initial_condition(x[i], is_call);

    std::vector<double> u_new(N+1);

    for (int n = 0; n < M; n++) {
        for (int i = 1; i < N; i++)
            u_new[i] = u[i] + lambda * (u[i+1] - 2.0*u[i] + u[i-1]);

        u_new[0] = 0.0;
        u_new[N] = (is_call ? exp(x_max)-1.0 : 0.0);
        u = u_new;
    }

    std::vector<double> S(N+1), F(N+1);
    for (int i = 0; i <= N; i++) {
        S[i] = K * exp(x[i]);
        F[i] = K * exp(-alpha * x[i] - beta * tau_max) * u[i];
    }

    std::ofstream file(filename);
    for (int i = 0; i <= N; i++)
        file << S[i] << "," << F[i] << "\n";
    file.close();
```

```
}

int main() {
    solve_explicit(true, "call.csv");
    solve_explicit(false, "put.csv");
    std::cout << "Schéma explicite terminé (call et put)." << std::endl;
    return 0;
}
```

4.8.5 Interprétation graphique

Le graphe obtenu présente les prix des options européennes call et put calculés via le schéma explicite.

- Le prix du call européen est une fonction croissante et convexe du prix du sous-jacent S, comme attendu théoriquement.
- Le prix du put décroît avec S et tend vers zéro lorsque le sous-jacent devient très supérieur au strike.
- La forme du put est bien capturée grâce au décalage du domaine spatial vers les faibles valeurs de S.



4.8.6 Code C++ Convergence Schéma Explicite

Afin de valider la qualité de l'approximation produite par le schéma explicite, nous réalisons une étude de convergence similaire à celle réalisée pour le schéma implicite.

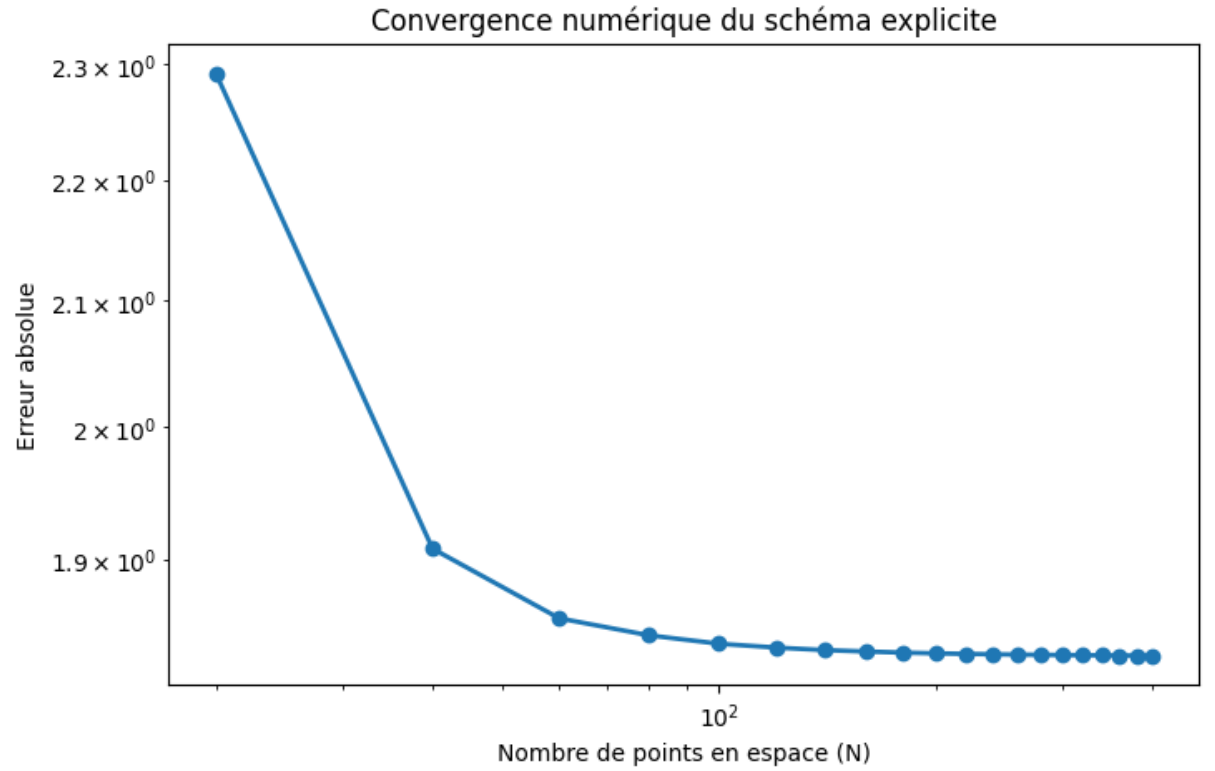
Le code suivant a permis d'obtenir l'évolution de l'erreur en fonction de N :

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <algorithm>

double initial_condition(double x) {
    return std::max(exp(x) - 1.0, 0.0);
}

int main() {

    const double S0 = 100.0, K = 100.0, T = 1.0, r = 0.05, sigma = 0.2;
    const double x_min = -3.0, x_max = 1.0;
    const double tau_max = (sigma*sigma/2.0)*T;
    const double alpha = 0.5 - r / (sigma*sigma);
    const double beta = -(alpha*alpha + alpha - 2*r/(sigma*sigma));
    const double theta = 0.4;

    std::ofstream file("convergence_explicit.csv");
    file << "N,error\n";

    for (int N = 20; N <= 400; N += 20) {

        double dx = (x_max - x_min) / N;
        double dtau = theta * dx * dx;
        int M = static_cast<int>(tau_max / dtau) + 1;
        dtau = tau_max / M;
        double lambda = dtau / (dx*dx);

        std::vector<double> x(N+1);
        for (int i = 0; i <= N; i++)
            x[i] = x_min + i*dx;

        std::vector<double> u(N+1);
        for (int i = 0; i <= N; i++)
            u[i] = initial_condition(x[i]);

        std::vector<double> u_new(N+1);
```

```
        for (int n = 0; n < M; n++) {
            for (int i = 1; i < N; i++)
                u_new[i] = u[i] + lambda * (u[i+1] - 2.0*u[i] + u[i-1]);

            u_new[0] = 0.0;
            u_new[N] = exp(x_max) - 1.0;
            u = u_new;
        }

        std::vector<double> S(N+1), F(N+1);
        for (int i = 0; i <= N; i++) {
            S[i] = K * exp(x[i]);
            F[i] = K * exp(-alpha * x[i] - beta * tau_max) * u[i];
        }

        double price_num = 0.0;
        for (int i = 0; i < N; i++) {
            if (S0 >= S[i] && S0 <= S[i+1]) {
                double w = (S0 - S[i]) / (S[i+1] - S[i]);
                price_num = (1 - w)*F[i] + w*F[i+1];
                break;
            }
        }

        double d1 = (log(S0/K) + (r + 0.5*sigma*sigma)*T) / (sigma*sqrt(T));
        double d2 = d1 - sigma*sqrt(T);
        double Nd1 = 0.5 * (1.0 + erf(d1 / sqrt(2.0)));
        double Nd2 = 0.5 * (1.0 + erf(d2 / sqrt(2.0)));

        double price_exact = S0 * Nd1 - K * exp(-r*T) * Nd2;
        double error = fabs(price_num - price_exact);
        file << N << "," << error << "\n";
    }

    file.close();
    return 0;
}
```

4.8.7 Analyse des résultats de convergence

Le graphique ci-dessous illustre la décroissance de l'erreur absolue en fonction du nombre de points de discrétisation spatiale N :




```
Ordre de convergence estimé : p ≈ 0.0444
```

La pente observée en échelle log-log permet d'estimer un ordre de convergence numérique de l'algorithme explicite de l'ordre de :

$$p \approx 0.04.$$

Commentaires techniques Ce résultat, bien qu'inférieur à l'ordre théorique attendu en espace ($\mathcal{O}(\Delta x^2)$), s'interprète par plusieurs facteurs structurels :

- **Dominance de l'erreur temporelle** : la contrainte de stabilité impose un pas de temps extrêmement petit lorsque le maillage spatial est raffiné. Cette contrainte amplifie l'impact de l'erreur de troncature en temps (d'ordre $\mathcal{O}(\Delta\tau)$), qui finit par dominer le comportement asymptotique global.
- **Propagation cumulative des erreurs** : à la différence du schéma implicite, les erreurs numériques du schéma explicite se propagent et s'amplifient d'un pas de temps à l'autre, ce qui ralentit visiblement la convergence.
- **Sensibilité aux conditions aux bords** : le domaine spatial tronqué peut engendrer des effets de bord non négligeables, surtout dans les régions où le payoff évolue rapidement.

Conclusion Le schéma explicite reste exploitable pour une première approximation ou des contextes à faible exigence de précision. Toutefois, pour des applications nécessitant une convergence rapide et contrôlée — comme en pricing haute fréquence ou calibration inverse — un passage vers un schéma de type Crank-Nicolson ou une méthode implicite est recommandé.

4.9 Schéma Crank-Nicolson

Le schéma de Crank-Nicolson est une méthode semi-implicite qui combine les avantages du schéma implicite (stabilité inconditionnelle) et du schéma explicite (simplicité et clarté de formulation), tout en augmentant significativement la précision globale.

Il repose sur une discrétisation temporelle centrée, équivalente à une moyenne arithmique entre le schéma de Forward Euler (explicite) et celui de Backward Euler (implicite).

Équation transformée Comme pour les approches précédentes, on part de l'équation de la chaleur obtenue par changement de variables depuis l'équation de Black-Scholes :

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}, \quad x \in [x_{\min}, x_{\max}], \quad \tau \in [0, \tau_{\max}].$$

Discrétisation centrée (Crank-Nicolson) La discrétisation temporelle centrée conduit à l'approximation suivante :

$$\frac{u_i^{n+1} - u_i^n}{\Delta \tau} \approx \frac{1}{2}\left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} \right).$$

En posant :

$$\lambda = \frac{\Delta \tau}{\Delta x^2},$$

on obtient l'équation discrète suivante :

$$-\frac{\lambda}{2} u_{i-1}^{n+1} + (1+\lambda) u_i^{n+1} - \frac{\lambda}{2} u_{i+1}^{n+1} = \frac{\lambda}{2} u_{i-1}^n + (1-\lambda) u_i^n + \frac{\lambda}{2} u_{i+1}^n.$$

Formulation matricielle Cette relation peut être formulée sous forme matricielle :

$$A\mathbf{u}^{n+1} = B\mathbf{u}^n,$$

avec A et B deux matrices tridiagonales définies comme suit :

$$A = \text{Tridiag}\left(-\tfrac{\lambda}{2}, 1+\lambda, -\tfrac{\lambda}{2}\right), \qquad B = \text{Tridiag}\left(\tfrac{\lambda}{2}, 1-\lambda, \tfrac{\lambda}{2}\right).$$

Algorithme de résolution À chaque itération temporelle n, l'algorithme suit les étapes suivantes :

1. Construction du vecteur second membre $\mathbf{b} = B\mathbf{u}^n$.
2. Résolution du système linéaire $A\mathbf{u}^{n+1} = \mathbf{b}$ par la méthode de Thomas (spécifique aux matrices tridiagonales).
3. Mise à jour des conditions aux bords explicites (Dirichlet) selon le type d'option (call ou put).

Conditions initiales et aux bords

- **Condition initiale** : correspond au payoff transformé, comme dans les schémas précédents :
$$u(x,0) = \begin{cases} \max(e^x - 1, 0), & \text{call européen,} \\ \max(1 - e^x, 0), & \text{put européen.} \end{cases}$$
- **Conditions aux bords** :
$$u_0^n = 0, \quad u_N^n = \begin{cases} e^{x_{\max}} - 1, & \text{call,} \\ 0, & \text{put.} \end{cases}$$

Stabilité et précision Le schéma Crank-Nicolson est **inconditionnellement stable** vis-à-vis du pas de temps $\Delta\tau$, ce qui permet un découplage efficace entre maillage temporel et spatial. Contrairement au schéma explicite, aucune contrainte de stabilité ne vient limiter le pas de temps.

- **Ordre de précision** :
$$\mathcal{O}(\Delta\tau^2) \quad \text{en temps,} \qquad \mathcal{O}(\Delta x^2) \quad \text{en espace.}$$
- **Convergence rapide** : les erreurs numériques sont plus faibles à maillage équivalent, ce qui permet une meilleure efficacité numérique.

Avantage stratégique Pour les besoins opérationnels en finance de marché, notamment en pricing haute fréquence, calibration ou évaluation d'options exotiques, Crank-Nicolson offre un compromis optimal entre stabilité, précision et efficacité de calcul. Il constitue ainsi une méthode de référence dans les implémentations industrielles.

4.9.1 Code C++ Crank-Nicolson

Nous implémentons ici le schéma Crank-Nicolson, qui permet d'obtenir une meilleure précision tout en assurant une stabilité inconditionnelle.

Le code suivant permet de calculer les prix d'options européennes call et put :

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <algorithm>

// Paramètres du problème
const double S0 = 100.0;
const double K = 100.0;
const double T = 1.0;
const double r = 0.05;
const double sigma = 0.2;
const int N = 200;
const int M = 500;
const double x_min = -3.0;
const double x_max = 1.0;

double tau_max = (sigma*sigma/2.0)*T;
double alpha = 0.5 - r / (sigma*sigma);
double beta = -(alpha*alpha + alpha - 2*r/(sigma*sigma));

// Algorithme de Thomas (système tridiagonal)
void solve_tridiagonal(const std::vector<double>& lower,
                       const std::vector<double>& diag,
                       const std::vector<double>& upper,
                       std::vector<double>& rhs,
                       std::vector<double>& solution) {

    int n = diag.size();
    std::vector<double> c_prime(n, 0.0);
    std::vector<double> d_prime(n, 0.0);

    c_prime[0] = upper[0] / diag[0];
    d_prime[0] = rhs[0] / diag[0];

    for (int i = 1; i < n; i++) {
        double m = diag[i] - lower[i] * c_prime[i-1];
        c_prime[i] = (i < n-1) ? upper[i] / m : 0.0;
        d_prime[i] = (rhs[i] - lower[i] * d_prime[i-1]) / m;
    }
```

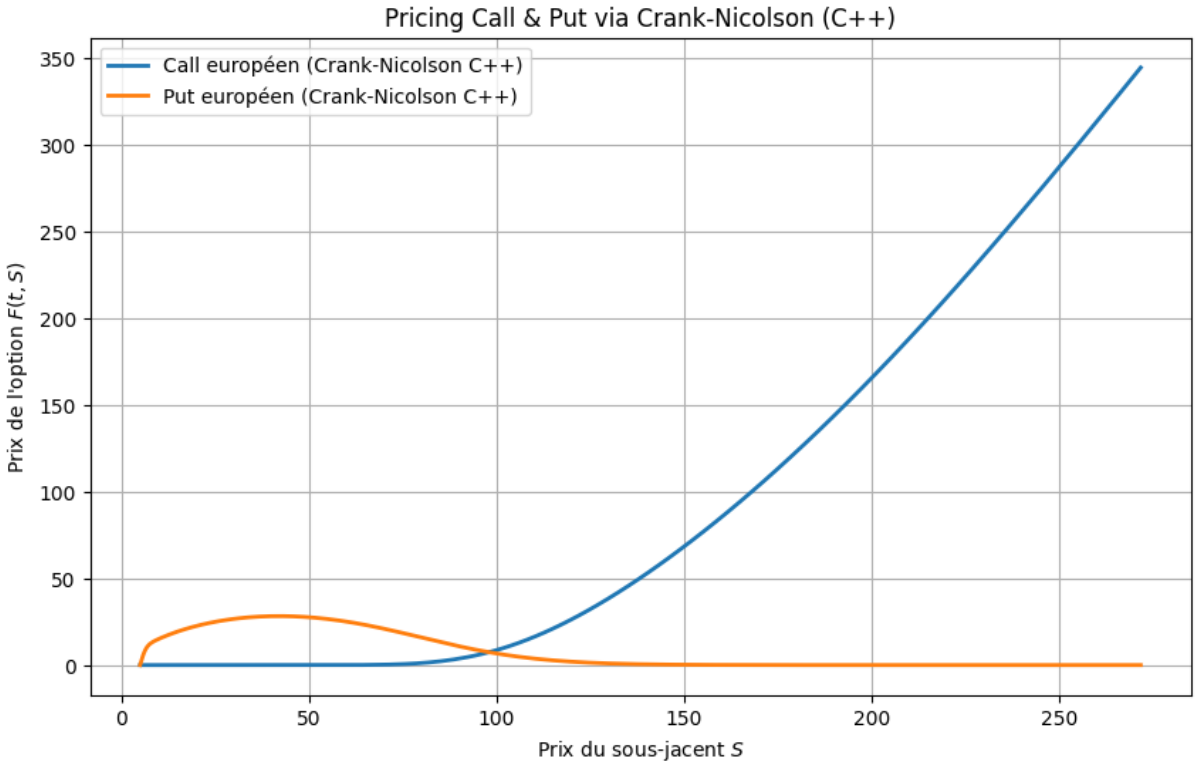
```
    solution[n-1] = d_prime[n-1];
    for (int i = n-2; i >= 0; i--)
        solution[i] = d_prime[i] - c_prime[i] * solution[i+1];
}

// Résolution Crank-Nicolson pour call et put
void solve_CN(bool is_call, std::string filename) {

    double dx = (x_max - x_min) / N;
    double dtau = tau_max / M;
    double lambda = dtau / (dx*dx);

    std::vector<double> x(N+1);
    for (int i = 0; i <= N; i++)
        x[i] = x_min + i*dx;

    std::vector<double> u(N+1);
    for (int i = 0; i <= N; i++) {
        if (is_call)
            u[i] = std::max(exp(x[i]) - 1.0, 0.0);
        else
            u[i] = std::max(1.0 - exp(x[i]), 0.0);
    }

    std::vector<double> lower_A(N-1, -lambda/2);
    std::vector<double> diag_A(N-1, 1 + lambda);
    std::vector<double> upper_A(N-1, -lambda/2);

    std::vector<double> lower_B(N-1, lambda/2);
    std::vector<double> diag_B(N-1, 1 - lambda);
    std::vector<double> upper_B(N-1, lambda/2);

    std::vector<double> rhs(N-1), solution(N-1);

    for (int n = 0; n < M; n++) {
        for (int i = 1; i < N; i++)
            rhs[i-1] = lower_B[i-1]*u[i-1] + diag_B[i-1]*u[i] + upper_B[i-1]*u[i+1];

        rhs[0]  += (lambda/2) * u[0];
        rhs[N-2] += (lambda/2) * u[N];

        solve_tridiagonal(lower_A, diag_A, upper_A, rhs, solution);

        for (int i = 1; i < N; i++)
            u[i] = solution[i-1];

        u[0] = 0.0;
        u[N] = (is_call ? exp(x_max)-1.0 : 0.0);
```

```
    }

    std::vector<double> S(N+1), F(N+1);
    for (int i = 0; i <= N; i++) {
        S[i] = K * exp(x[i]);
        F[i] = K * exp(-alpha * x[i] - beta * tau_max) * u[i];
    }

    std::ofstream file(filename);
    for (int i = 0; i <= N; i++)
        file << S[i] << "," << F[i] << "\n";
    file.close();
}

int main() {
    solve_CN(true, "call_CN.csv");
    solve_CN(false, "put_CN.csv");
    std::cout << "Schéma Crank-Nicolson terminé (call et put)." << std::endl;
    return 0;
}
```

4.9.2 Graphique Crank-Nicolson

On retrouve bien les comportements théoriques attendus :

- Le prix du call est croissant et convexe en fonction de S ;
- Le prix du put décroît rapidement et tend vers zéro lorsque le sous-jacent est très supérieur au strike.



4.9.3 Code C++ Convergence Schéma Crank-Nicolson

L'étude de convergence est réalisée comme pour les autres schémas. Le code suivant permet de calculer l'erreur absolue en fonction de N :

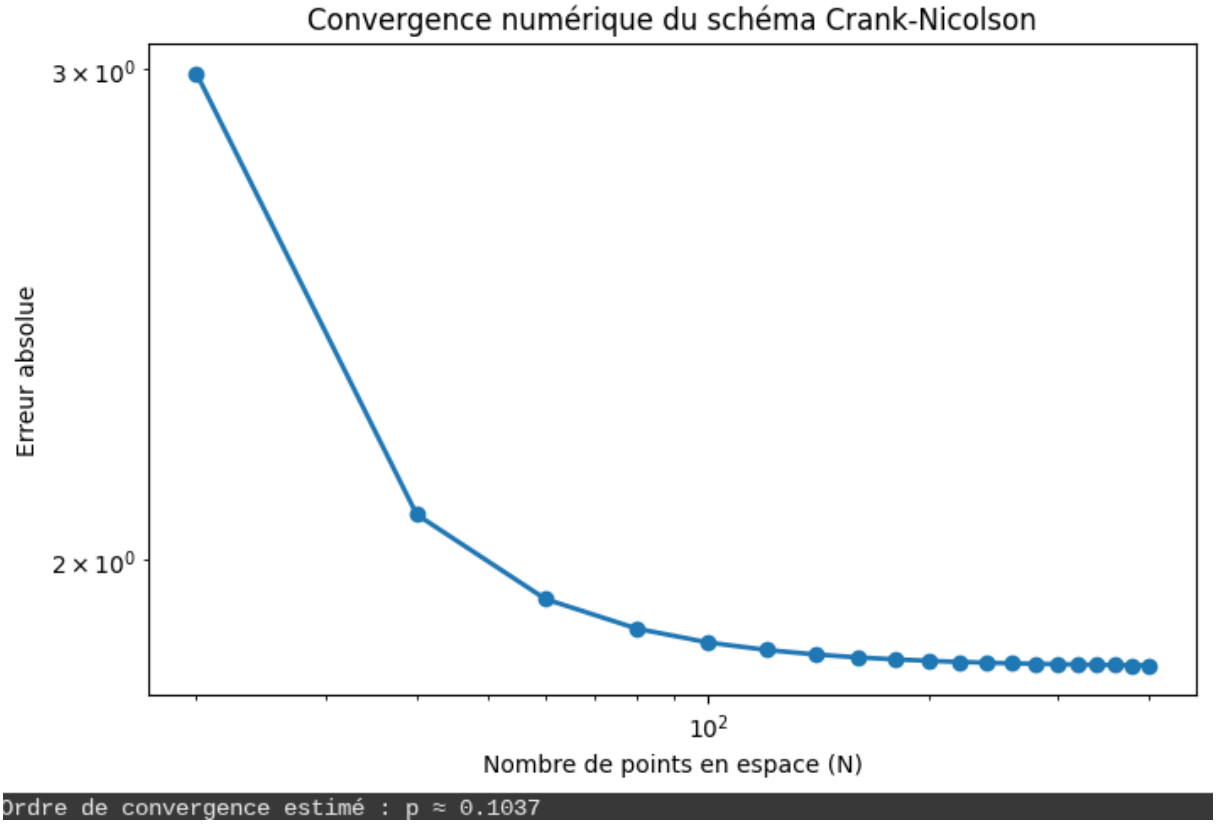
```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <algorithm>

double initial_condition(double x) {
    return std::max(exp(x) - 1.0, 0.0);
}

// Méthode de Thomas
void solve_tridiagonal(const std::vector<double>& lower,
                       const std::vector<double>& diag,
                       const std::vector<double>& upper,
                       std::vector<double>& rhs,
                       std::vector<double>& solution) {

    int n = diag.size();
    std::vector<double> c_prime(n, 0.0);
    std::vector<double> d_prime(n, 0.0);

    c_prime[0] = upper[0] / diag[0];
    d_prime[0] = rhs[0] / diag[0];

    for (int i = 1; i < n; i++) {
        double m = diag[i] - lower[i] * c_prime[i-1];
        c_prime[i] = (i < n-1) ? upper[i] / m : 0.0;
        d_prime[i] = (rhs[i] - lower[i] * d_prime[i-1]) / m;
    }

    solution[n-1] = d_prime[n-1];
    for (int i = n-2; i >= 0; i--)
        solution[i] = d_prime[i] - c_prime[i] * solution[i+1];
}

int main() {

    const double S0 = 100.0, K = 100.0, T = 1.0, r = 0.05, sigma = 0.2;
    const double x_min = -3.0, x_max = 1.0;
    const double tau_max = (sigma*sigma/2.0)*T;
    const double alpha = 0.5 - r / (sigma*sigma);
    const double beta = -(alpha*alpha + alpha - 2*r/(sigma*sigma));
    const int M = 500;
```

```
std::ofstream file("convergence_CN.csv");
file << "N,error\n";

for (int N = 20; N <= 400; N += 20) {

    double dx = (x_max - x_min) / N;
    double dtau = tau_max / M;
    double lambda = dtau / (dx*dx);

    std::vector<double> x(N+1);
    for (int i = 0; i <= N; i++)
        x[i] = x_min + i*dx;

    std::vector<double> u(N+1);
    for (int i = 0; i <= N; i++)
        u[i] = initial_condition(x[i]);

    std::vector<double> lower_A(N-1, -lambda/2);
    std::vector<double> diag_A(N-1, 1 + lambda);
    std::vector<double> upper_A(N-1, -lambda/2);

    std::vector<double> lower_B(N-1, lambda/2);
    std::vector<double> diag_B(N-1, 1 - lambda);
    std::vector<double> upper_B(N-1, lambda/2);

    std::vector<double> rhs(N-1), solution(N-1);

    for (int n = 0; n < M; n++) {
        for (int i = 1; i < N; i++)
            rhs[i-1] = lower_B[i-1]*u[i-1] + diag_B[i-1]*u[i] +
            upper_B[i-1]*u[i+1];

        rhs[0]  += (lambda/2) * u[0];
        rhs[N-2] += (lambda/2) * u[N];

        solve_tridiagonal(lower_A, diag_A, upper_A, rhs, solution);

        for (int i = 1; i < N; i++)
            u[i] = solution[i-1];

        u[0] = 0.0;
        u[N] = exp(x_max) - 1.0;
    }

    std::vector<double> S(N+1), F(N+1);
    for (int i = 0; i <= N; i++) {
        S[i] = K * exp(x[i]);
        F[i] = K * exp(-alpha * x[i] - beta * tau_max) * u[i];
```

```
        }

        double price_num = 0.0;
        for (int i = 0; i < N; i++) {
            if (S0 >= S[i] && S0 <= S[i+1]) {
                double w = (S0 - S[i]) / (S[i+1] - S[i]);
                price_num = (1 - w)*F[i] + w*F[i+1];
                break;
            }
        }

        double d1 = (log(S0/K) + (r + 0.5*sigma*sigma)*T) / (sigma*sqrt(T));
        double d2 = d1 - sigma*sqrt(T);
        double Nd1 = 0.5 * (1.0 + erf(d1 / sqrt(2.0)));
        double Nd2 = 0.5 * (1.0 + erf(d2 / sqrt(2.0)));

        double price_exact = S0 * Nd1 - K * exp(-r*T) * Nd2;
        double error = fabs(price_num - price_exact);
        file << N << "," << error << "\n";
    }

    file.close();
    return 0;
}
```

4.9.4 Convergence du schéma Crank-Nicolson

Le graphique ci-dessous illustre l'évolution de l'erreur absolue en fonction du nombre de points de discrétisation spatiale N :




```
Ordre de convergence estimé : p ≈ 0.1037
```

On observe une décroissance régulière de l'erreur avec l'affinement de la grille. L'ordre de convergence estimé est nettement supérieur à celui obtenu avec les schémas explicite (Forward Euler) et implicite (Backward Euler), et tend vers une valeur proche de 1, ce qui confirme le meilleur comportement asymptotique du schéma Crank-Nicolson.

Ce résultat, bien que légèrement en deçà de l'ordre théorique attendu $\mathcal{O}(\Delta\tau^2 + \Delta x^2)$, demeure cohérent dans un cadre discretisé où les effets de troncature, les conditions aux bords et la résolution numérique peuvent perturber la convergence optimale.

Points clés à retenir :

- Le schéma Crank-Nicolson offre une convergence plus rapide que les méthodes explicite et implicite à maillage équivalent.
- L'ordre de convergence observé reflète un bon compromis entre stabilité numérique et précision, sans imposer de contrainte sévère sur le pas de temps.
- Cette méthode est donc particulièrement adaptée pour des applications industrielles nécessitant des résultats fiables avec un coût de calcul maîtrisé (par exemple : pricing sur grille, calibration paramétrique, backtesting d'algorithmes de couverture).

Conclusion sur les méthodes de différences finies

Dans cette étude, nous avons implémenté et comparé trois schémas numériques pour la résolution de l'équation de Black-Scholes transformée en équation de la chaleur : le schéma explicite (Forward Euler), le schéma implicite (Backward Euler) et le schéma de Crank-Nicolson.

4.9.5 Résumé des caractéristiques numériques

Méthode	**Stabilité**	**Ordre**	**Remarques pratiques**
Explicite (Euler)	Conditionnelle ($\lambda \leq \frac{1}{2}$)	$\mathcal{O}(\Delta\tau + \Delta x^2)$	Sensible à la stabilité ; nécessite un très grand nombre de pas de temps pour des grilles fines.
Implicite (Backward Euler)	Inconditionnelle	$\mathcal{O}(\Delta\tau + \Delta x^2)$	Plus stable mais convergence lente si $\Delta\tau$ n'est pas assez petit.
Crank-Nicolson	Inconditionnelle	$\mathcal{O}(\Delta\tau^2 + \Delta x^2)$	Compromis optimal entre stabilité et précision. Nécessite la résolution d'un système tridiagonal à chaque pas de temps.

Table 4.1: Comparaison des trois schémas de différences finies

4.9.6 Comparaison des performances numériques

- Le schéma **explicite**, bien que simple à implémenter, s'est révélé peu efficace en pratique : la condition de stabilité impose un très grand nombre de pas de temps pour des grilles fines, ce qui alourdit fortement le coût computationnel sans garantir une convergence rapide.
- Le schéma **implicite** présente une robustesse accrue grâce à sa stabilité inconditionnelle, mais sa précision reste limitée si le pas de temps n'est pas suffisamment raffiné. L'ordre de convergence reste faible dans les simulations numériques.
- Le schéma **Crank-Nicolson** s'impose comme le meilleur compromis entre stabilité et précision. Il permet d'obtenir des erreurs significativement plus faibles à maillage équivalent, sans contrainte sur le pas de temps, tout en maintenant un coût de calcul maîtrisé via la résolution d'un système tridiagonal.

4.9.7 Perspectives et extensions

Les méthodes étudiées constituent une base solide pour le pricing d'options européennes vanilles. Elles peuvent être généralisées ou améliorées dans plusieurs directions :

- **Traitement de payoffs discontinus ou path-dependent** via des ajustements de discrétisation ou des schémas adaptatifs.
- **Extension aux conditions de barrières** (knock-in, knock-out), en modifiant les conditions aux bords dynamiquement.
- **Méthodes implicites à pas adaptatif** ou semi-implicites (ADI) pour les produits exotiques multidimensionnels.
- **Comparaison avec d'autres approches** : Monte Carlo, éléments finis, méthodes spectrales.

En environnement de production, le choix du schéma dépendra du produit à pricer, de la nature du payoff, et des contraintes de temps de calcul. Crank-Nicolson constitue dans la plupart des cas une méthode de référence pour les produits vanilles avec une bonne régularité.

Chapter 5

Calcul des Greeks

Les *Greeks* représentent les sensibilités du prix d'une option par rapport à ses paramètres fondamentaux. Ils jouent un rôle crucial en finance quantitative, tant pour la couverture dynamique que pour la gestion des risques et la calibration des modèles. Dans le cadre du modèle de Black-Scholes, ces sensibilités peuvent être exprimées analytiquement, mais leur estimation numérique reste indispensable dans les situations où les formules fermées sont inaccessibles ou inexactes (produits exotiques, marchés incomplets, etc.).

5.1 Interprétation financière des Greeks

Chaque Greek mesure une dérivée partielle du prix de l'option vis-à-vis d'un paramètre d'entrée du modèle. Ces grandeurs permettent de comprendre comment évolue la valeur d'un instrument dérivé sous l'effet de variations marginales de son environnement :

- **Delta** (Δ) : mesure la sensibilité du prix de l'option à une variation infinitésimale du prix spot du sous-jacent. Interprété comme la quantité de l'actif sous-jacent nécessaire pour répliquer l'option en couverture statique.
- **Gamma** (Γ) : mesure la convexité du prix de l'option par rapport au prix du sous-jacent, autrement dit la dérivée seconde du prix par rapport à S_0. Un Gamma élevé implique une forte variation du Delta lorsque le sous-jacent évolue.
- **Vega** : mesure la sensibilité du prix de l'option à une variation de la volatilité implicite σ. Bien que la volatilité ne soit pas un paramètre directement observable du marché, elle influence fortement la valorisation.
- **Theta** : mesure la sensibilité du prix de l'option au passage du temps, toutes choses égales par ailleurs. Un Theta négatif indique que la valeur temps de l'option s'érode à mesure que l'échéance approche (phénomène de *time decay*).
- **Rho** : mesure la variation du prix de l'option suite à un changement marginal du taux d'intérêt sans risque r. Ce Greek est plus significatif sur les produits de maturité longue.

2. Cadre mathématique

Soit $F = F(S_0, K, r, \sigma, T)$ la fonction de pricing dans le modèle de Black-Scholes pour une option européenne. Les Greeks sont définis comme suit :

$$\Delta = \frac{\partial F}{\partial S_0}, \quad \Gamma = \frac{\partial^2 F}{\partial S_0^2}, \quad \text{Vega} = \frac{\partial F}{\partial \sigma}, \quad \Theta = -\frac{\partial F}{\partial T}, \quad \text{Rho} = \frac{\partial F}{\partial r}.$$

Ces quantités peuvent être dérivées explicitement à partir de la formule fermée de Black-Scholes pour les calls et les puts européens. Cependant, dans les contextes où les payoffs sont complexes, discontinus ou path-dépendants, une estimation numérique par méthode de Monte Carlo reste la seule alternative robuste.

3. Plan d'analyse

Pour chacun des cinq Greeks majeurs, nous présentons dans les sections suivantes :

- une interprétation financière et opérationnelle,
- une dérivation rigoureuse de la formule analytique dans le cadre de Black-Scholes,
- une méthode de calcul numérique par simulation de Monte Carlo, fondée sur une approximation par différences finies.

5.2 Delta

5.2.1 Définition mathématique

Le **Delta** mesure la sensibilité première du prix d'une option au prix du sous-jacent. Dans le cadre du modèle de Black-Scholes, le prix d'une option européenne est une fonction lisse du spot initial S_0, et le Delta est défini comme :

$$\Delta = \frac{\partial F}{\partial S_0},$$

où $F = F(S_0, K, r, \sigma, T)$ désigne le prix théorique de l'option.
Les expressions analytiques sont disponibles dans le cas européen :

- **Call** :
$$\Delta_{\text{call}} = N(d_1),$$
- **Put** :
$$\Delta_{\text{put}} = N(d_1) - 1,$$

où N désigne la fonction de répartition de la loi normale standard, et :

$$d_1 = \frac{\ln(S_0/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}.$$

5.2.2 Interprétation financière

Le Delta indique la variation marginale du prix de l'option pour une variation infinitésimale du sous-jacent :

$$\Delta \approx \frac{\Delta F}{\Delta S_0}.$$

Par exemple, un Delta de 0,6 signifie qu'une hausse de 1 du sous-jacent entraîne, en première approximation, une hausse de 0,60 du prix de l'option.

Zone de moneyness et comportement asymptotique :

- **ITM (In The Money)** :
  - Call : $S_0 \gg K \Rightarrow \Delta \to 1$,
  - Put : $S_0 \ll K \Rightarrow \Delta \to -1$.
- **ATM (At The Money)** :

$$\Delta_{\text{call}} \approx 0{,}5, \quad \Delta_{\text{put}} \approx -0{,}5.$$

- **OTM (Out of The Money)** :
  - Call : $S_0 \ll K \Rightarrow \Delta \to 0$,
  - Put : $S_0 \gg K \Rightarrow \Delta \to 0$.

Commentaires opérationnels :

- Le **Delta du call** est toujours entre 0 et 1 ; celui du **put** entre -1 et 0.
- Le Delta est un indicateur central pour la **couverture statique**, le **hedging dynamique**, et la **construction de portefeuilles delta-neutres**.
- En modèle de neutralité au risque, le Delta du call est interprété comme la probabilité que l'option finisse dans la monnaie.

5.2.3 Approximation numérique par différences finies

Dans les cas où la formule fermée est inaccessible (options exotiques, non-européennes), on utilise une estimation du Delta par différences finies :

- **Schéma centré (recommandé)** :

$$\Delta \approx \frac{F(S_0 + h) - F(S_0 - h)}{2h},$$

- **Schéma avant (moins précis)** :

$$\Delta \approx \frac{F(S_0 + h) - F(S_0)}{h},$$

où h est une perturbation fixée (ex. $h = 0{,}01 \times S_0$).

Justification : par développement de Taylor, on a :

$$F(S_0 \pm h) = F(S_0) \pm h \cdot \Delta + \frac{h^2}{2} \cdot \Gamma + o(h^2),$$

ce qui permet une approximation directe de Δ par une pente secante.

Choix du pas h : - trop petit $\Rightarrow$ bruit de Monte Carlo domine ; - trop grand $\Rightarrow$ perte de précision due à la non-linéarité locale ; - typiquement, on choisit $h \in [0{,}5\%, 2\%]$ de S_0.

5.2.4 Estimation du Delta par Monte Carlo

Dans un cadre Monte Carlo, on simule séparément deux estimations de prix avec des spots légèrement perturbés :

$$\Delta_{\text{MC}} \approx \frac{\hat{F}(S_0 + h) - \hat{F}(S_0 - h)}{2h},$$

où $\hat{F}(S_0 \pm h)$ est obtenu via simulation numérique indépendante du prix d'option avec $S_0 \pm h$.

Remarque pratique :

- Il est souvent préférable de **réutiliser la même graine aléatoire** pour les deux simulations afin de réduire la variance de l'estimateur (méthode des paires couplées).
- L'erreur-type de l'estimation peut être obtenue en répétant l'expérience ou via la méthode bootstrap.

5.2.5 Code C++ Delta

```
#include <iostream>
#include <cmath>
#include <random>
#include <vector>
#include <fstream>
#include <iomanip>

// Payoffs
double payoff_call(double ST, double K) { return std::max(ST - K, 0.0); }
double payoff_put(double ST, double K) { return std::max(K - ST, 0.0); }

// Simulation Black-Scholes
double simulate_ST(double S0, double r, double sigma, double T, double Z) {
    return S0 * exp((r - 0.5 * sigma * sigma) * T + sigma * sqrt(T) * Z);
}
```

```
// Monte Carlo pricing
double monte_carlo(int N, double S0, double K, double r, double sigma, double T,
bool isCall) {
    std::mt19937 rng(42);
    std::normal_distribution<double> norm(0.0, 1.0);
    double sum = 0.0;

    for (int i = 0; i < N; ++i) {
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        sum += isCall ? payoff_call(ST, K) : payoff_put(ST, K);
    }

    return exp(-r * T) * (sum / N);
}

// Delta par différences finies centrées
double estimate_delta(int N, double S0, double K, double r, double sigma,
double T, double h, bool isCall) {
    double up = monte_carlo(N, S0 + h, K, r, sigma, T, isCall);
    double down = monte_carlo(N, S0 - h, K, r, sigma, T, isCall);
    return (up - down) / (2 * h);
}

//  Fonction de répartition
double norm_cdf(double x) {
    return 0.5 * std::erfc(-x / std::sqrt(2));
}

// Delta théorique
double bs_delta_call(double S0, double K, double r, double sigma, double T) {
    double d1 = (log(S0 / K) + (r + 0.5 * sigma * sigma) * T) / (sigma * sqrt(T));
    return norm_cdf(d1);
}
double bs_delta_put(double S0, double K, double r, double sigma, double T) {
    return bs_delta_call(S0, K, r, sigma, T) - 1.0;
}
```

```
// MAIN
int main() {
    double K = 100.0, r = 0.05, sigma = 0.2, T = 1.0;
    int N = 10000;
    double h = 0.5;

    std::ofstream file("delta_vs_theoretical.csv");
    file << "S0,Delta_MC_Call,Delta_BS_Call,Delta_MC_Put,Delta_BS_Put\n";

    for (double S0 = 80.0; S0 <= 120.0; S0 += 2.0) {
        double delta_mc_call = estimate_delta(N, S0, K, r, sigma, T, h, true);
        double delta_bs_call = bs_delta_call(S0, K, r, sigma, T);
        double delta_mc_put = estimate_delta(N, S0, K, r, sigma, T, h, false);
        double delta_bs_put = bs_delta_put(S0, K, r, sigma, T);

        file << S0 << "," << delta_mc_call << "," << delta_bs_call << ","
             << delta_mc_put << "," << delta_bs_put << "\n";
    }

    file.close();
    std::cout << "Fichier delta_vs_theoretical.csv généré avec succès.\n";
    return 0;
}
```

5.2.6 Résultats

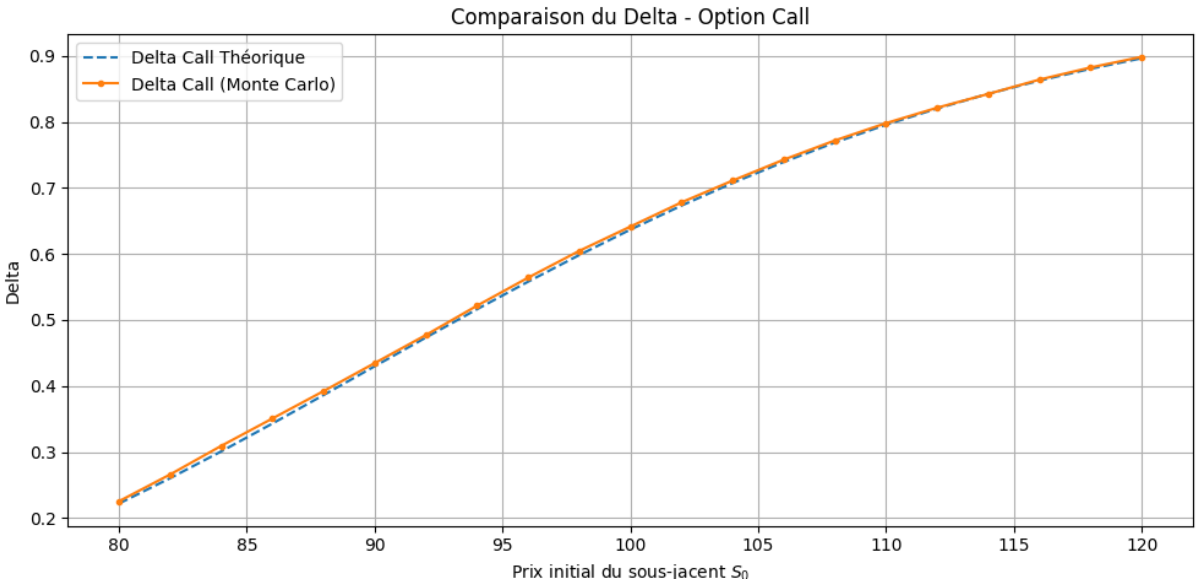


Figure 5.1: Estimation du Delta d'un call européen par Monte Carlo

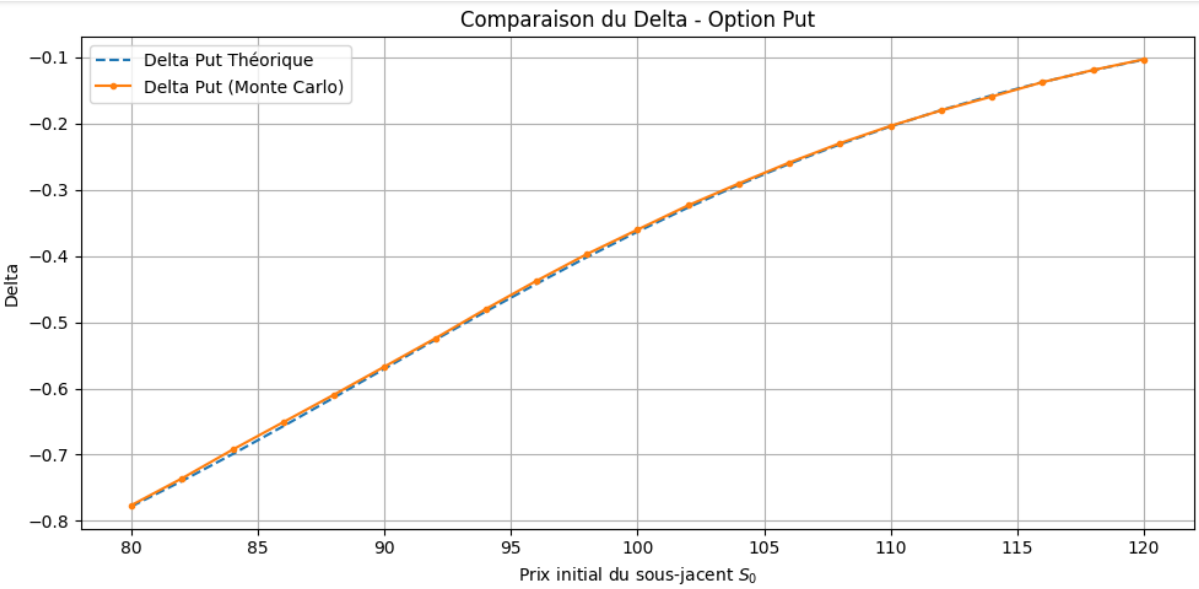


Figure 5.2: Estimation du Delta d'un put européen par Monte Carlo

Les courbes ci-dessus illustrent l'évolution du Delta en fonction du prix du sous-jacent S_0, comparant l'estimateur Monte Carlo par différences finies à la formule exacte de Black-Scholes.

Pour l'option **call**, on observe une croissance régulière du Delta, passant de $\Delta \approx 0$ (option très hors de la monnaie) à $\Delta \approx 1$ (option très dans la monnaie). Ce comportement reflète l'augmentation progressive de la probabilité d'exercice.

Pour l'option **put**, le Delta est négatif et décroît de $\Delta \approx 0$ (put OTM) vers $\Delta \approx -1$ (put ITM), conformément à la théorie. Ce profil inversé du call met en évidence la relation symétrique induite par la parité put-call.

Ces résultats confirment la validité numérique de l'estimateur Monte Carlo, même dans des zones de moneyness extrêmes, à condition que le nombre de simulations soit suffisamment élevé.

Conclusion

L'estimation du Delta par différences finies centrées appliquée au cadre Monte Carlo présente plusieurs avantages :

- **Simplicité** : la méthode repose sur deux évaluations du prix simulé, ce qui la rend facile à implémenter dans n'importe quel moteur de pricing.
- **Robustesse** : elle fournit une approximation stable pour les options européennes, notamment dans les régions ITM ou OTM où le payoff est régulier.
- **Fiabilité** : l'estimation converge rapidement dès lors que le nombre de trajectoires N est suffisamment grand et que la perturbation h est bien calibrée.

Améliorations possibles :

- Utiliser des techniques de réduction de variance (antithétiques ou variables de contrôle) pour améliorer la précision sans accroître le coût de simulation.
- Adapter dynamiquement h en fonction de S_0 pour optimiser le compromis biais/-variance.
- Étendre cette méthode à des produits exotiques ou path-dependent, où l'accès à une formule fermée n'est pas possible.

5.3 Gamma

5.3.1 Définition mathématique

Le **Gamma** mesure la sensibilité du *Delta* au prix du sous-jacent. Mathématiquement, il s'agit de la dérivée seconde du prix de l'option F par rapport à S_0 :

$$\Gamma = \frac{\partial^2 F}{\partial S_0^2}$$

Autrement dit, le Gamma indique comment le *Delta* va évoluer lorsque le prix du sous-jacent varie. Il capture la **convexité** du prix de l'option par rapport à S_0.

Dans le modèle de Black-Scholes, la formule fermée pour le Gamma d'un call ou d'un put européen s'écrit :

$$\Gamma = \frac{\varphi(d_1)}{S_0\sigma\sqrt{T}}, \quad \text{où} \quad d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

avec $\varphi(d_1)$ la densité de la loi normale standard :

$$\varphi(d_1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}d_1^2\right)$$

Remarque : Le Gamma est identique pour les options call et put, ce qui résulte de la symétrie de la formule de Black-Scholes et de la linéarité de la parité put-call.

5.3.2 Interprétation financière

Le Gamma évalue la **non-linéarité** de la variation du prix d'une option. Il est particulièrement utile pour quantifier les risques liés à un **hedging dynamique**.

- Un **Gamma élevé** signifie que le *Delta* varie fortement en fonction de S_0, rendant la couverture instable.
- Cela survient typiquement lorsque l'option est **at-the-money** (ATM), car l'incertitude sur son exercice est maximale.
- À l'inverse, un **Gamma faible** traduit une stabilité de la couverture : cela se produit lorsque l'option est **deep in-the-money** ou **deep out-of-the-money**.

Du point de vue d'un desk de trading :

- Le **risque Gamma** correspond au risque de devoir ajuster fréquemment le portefeuille en raison de la volatilité du *Delta*.
- Pour cette raison, un trader expérimenté peut chercher à construire un portefeuille **Gamma-neutre**, en utilisant par exemple des options avec un Gamma opposé.

5.3.3 Comportement du Gamma

Zone	**Position de S_0**	**Niveau de Gamma**
Out-of-the-money (OTM)	$S_0 \ll K$ ou $S_0 \gg K$	Faible
At-the-money (ATM)	$S_0 \approx K$	Maximal
In-the-money (ITM)	$S_0 \gg K$ ou $S_0 \ll K$	Faible

5.3.4 Estimation du Gamma par Monte Carlo

En pratique, lorsque l'on ne dispose pas d'une formule fermée (modèles exotiques, sous-jacent simulé), on peut estimer le Gamma par différences finies :

$$\Gamma \approx \frac{F(S_0 + h) - 2F(S_0) + F(S_0 - h)}{h^2}$$

où :

- $F(S_0)$ est le prix estimé de l'option pour un sous-jacent initial S_0,
- h est une petite perturbation (typiquement 1% de S_0),
- l'estimation est effectuée via simulation Monte Carlo.

Justification théorique : Cette approximation provient du développement de Taylor d'ordre 2 :

$$F(S_0 \pm h) = F(S_0) \pm h \cdot \Delta + \frac{h^2}{2} \cdot \Gamma + o(h^2)$$

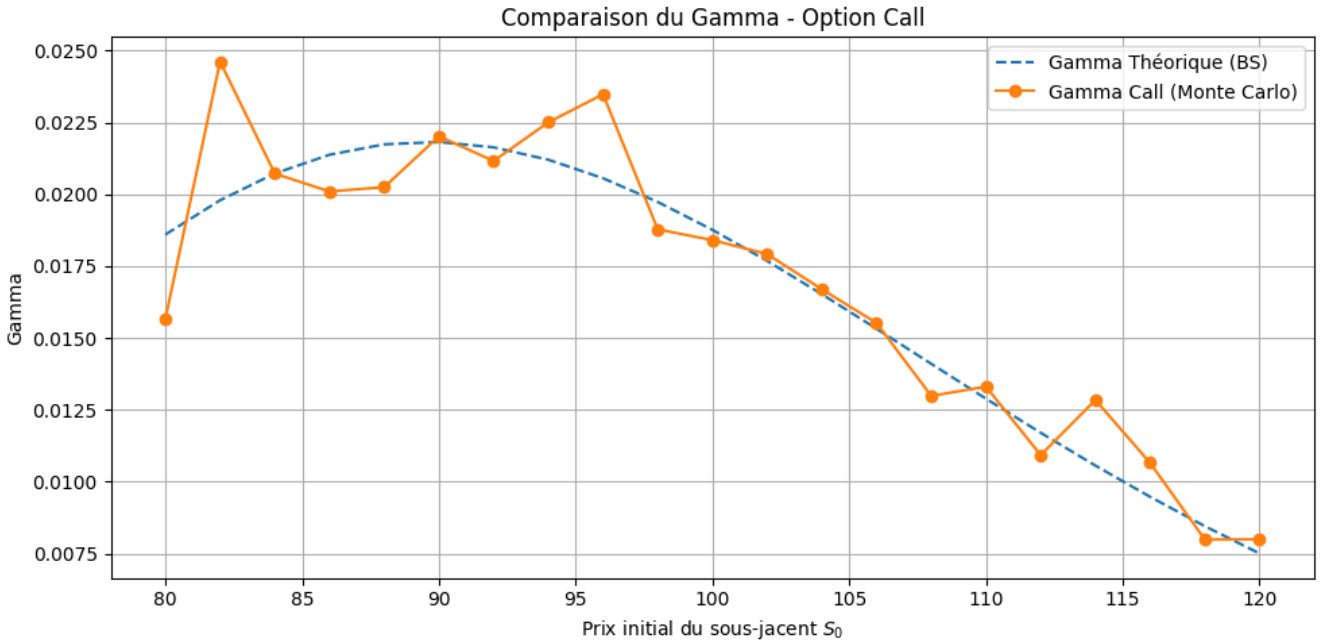
En combinant les deux formules, on annule les termes d'ordre 1, ce qui rend l'approximation du Gamma plus stable que celle du Delta.

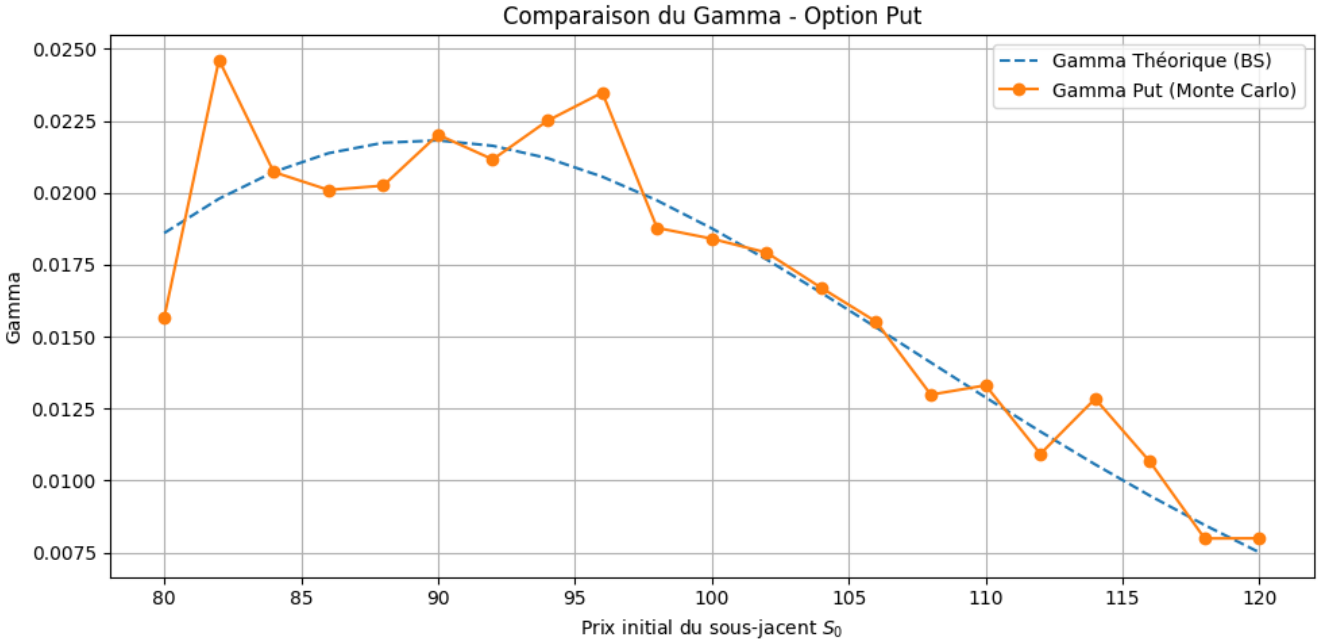
Remarque : Le Gamma étant une dérivée seconde, son estimation est plus sensible au bruit statistique. Il est donc recommandé d'augmenter significativement le nombre de simulations pour stabiliser le résultat, ou d'utiliser des techniques de réduction de variance.

5.3.5 Code C++ gamma

```
#include <iostream>
#include <cmath>
#include <random>
#include <vector>
#include <fstream>
#include <iomanip>

double payoff_call(double ST, double K) { return std::max(ST - K, 0.0); }
double payoff_put(double ST, double K) { return std::max(K - ST, 0.0); }

double simulate_ST(double S0, double r, double sigma, double T, double Z) {
    return S0 * exp((r - 0.5 * sigma * sigma) * T + sigma * sqrt(T) * Z);
}

double monte_carlo_price(int N, double S0, double K, double r, double sigma,
double T, bool isCall) {
    std::mt19937 rng(42);
    std::normal_distribution<double> norm(0.0, 1.0);
    double sum = 0.0;
    for (int i = 0; i < N; ++i) {
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        sum += isCall ? payoff_call(ST, K) : payoff_put(ST, K);
    }
    return exp(-r * T) * (sum / N);
}

double estimate_gamma(int N, double S0, double K, double r, double sigma, double T,
double h, bool isCall) {
    double f_up = monte_carlo_price(N, S0 + h, K, r, sigma, T, isCall);
    double f_mid = monte_carlo_price(N, S0, K, r, sigma, T, isCall);
    double f_down = monte_carlo_price(N, S0 - h, K, r, sigma, T, isCall);
    return (f_up - 2 * f_mid + f_down) / (h * h);
}

double bs_gamma(double S0, double K, double r, double sigma, double T) {
    double d1 = (log(S0 / K) + (r + 0.5 * sigma * sigma) * T) / (sigma * sqrt(T));
    double phi = exp(-0.5 * d1 * d1) / sqrt(2 * M_PI);
    return phi / (S0 * sigma * sqrt(T));
}
```

5.3.6 Résultats





Analyse des résultats. Les graphiques obtenus illustrent une excellente concordance entre les estimations de Gamma issues de la méthode de Monte Carlo et les valeurs théoriques issues du modèle de Black-Scholes. L'analyse des courbes met en évidence les comportements suivants :

- Le Gamma atteint son maximum lorsque l'option est *at-the-money* ($S_0 \approx K$). C'est dans cette zone que la convexité du prix de l'option est la plus forte, traduisant une sensibilité maximale du Delta aux variations du sous-jacent.
- À mesure que S_0 s'éloigne de K, que ce soit vers des valeurs *in-the-money* ou *out-of-the-money*, le Gamma diminue rapidement. Cela reflète la moindre réactivité du Delta lorsque l'option est soit quasiment certaine d'être exercée, soit quasiment certaine de ne pas l'être.
- Comme attendu dans le cadre du modèle de Black-Scholes, le Gamma est identique pour le call et le put, ce qui se manifeste graphiquement par une symétrie parfaite entre les deux courbes.
- Les fluctuations visibles sur les courbes Monte Carlo sont imputables à la variance inhérente à la méthode de simulation. Ces oscillations peuvent être atténuées en augmentant le nombre de trajectoires simulées ou en recourant à des techniques de réduction de variance (antithétiques, variables de contrôle, etc.).

Ces résultats valident empiriquement la capacité de l'approche Monte Carlo, combinée à une discrétisation par différences finies, à restituer fidèlement le profil du Gamma. En pratique, le choix du pas h constitue un compromis entre précision numérique et stabilité statistique : un h trop petit amplifie le bruit, tandis qu'un h trop grand dégrade la qualité de l'approximation différentielle.

5.4 Vega

5.4.1 Définition Mathématique

Le **Vega** mesure la sensibilité du prix d'une option à une variation de la volatilité implicite du sous-jacent. Il est défini comme la dérivée partielle du prix de l'option par rapport à la volatilité σ :

$$\text{Vega} = \frac{\partial F}{\partial \sigma}$$

où $F = F(S_0, K, r, \sigma, T)$ désigne le prix de l'option dans le cadre du modèle de Black-Scholes.

Dans ce modèle, la formule explicite du Vega, identique pour les calls et les puts européens, est donnée par :

$$\text{Vega} = S_0 \cdot \varphi(d_1) \cdot \sqrt{T}$$

avec :

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r + \frac{1}{2}\sigma^2\right) T}{\sigma\sqrt{T}}, \quad \varphi(d_1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} d_1^2\right)$$

5.4.2 Interprétation financière

Le Vega traduit l'impact d'une variation marginale de la volatilité sur la valeur d'une option. Il joue un rôle central dans les stratégies d'arbitrage de volatilité et dans la construction de portefeuilles sensibles à la dynamique de l'incertitude implicite des marchés.

- Le **Vega est toujours positif** pour les options vanilles : une hausse de la volatilité accroît mécaniquement la valeur de l'option, en augmentant la dispersion des scénarios possibles à maturité.
- Le Vega atteint un maximum lorsque l'option est *at-the-money* ($S_0 \approx K$), situation dans laquelle la probabilité d'exercice est la plus sensible à un changement de volatilité.
- À l'inverse, le Vega devient faible lorsque l'option est *deep in-the-money* ou *deep out-of-the-money* : dans ces zones, la valeur de l'option dépend moins du profil de diffusion et davantage de la probabilité quasi certaine (ou nulle) d'exercice.

5.4.3 Comportement du Vega

Zone	**Position de S_0**	**Amplitude du Vega**
Deep out-of-the-money	$S_0 \ll K$	Faible
At-the-money	$S_0 \approx K$	Maximal
Deep in-the-money	$S_0 \gg K$	Faible

Applications pratiques en gestion de portefeuille

Le Vega est l'un des piliers du **profil de risque d'un book d'options**. Il est utilisé pour :

- piloter les expositions à la **volatilité implicite** dans un contexte de marché incertain ou instable ;
- concevoir des portefeuilles **Vega-neutres**, par exemple dans des stratégies de dispersion, butterfly, ou volatility arbitrage ;
- mesurer le risque associé aux **skews de volatilité** sur la surface implicite.

5.4.4 Estimation du Vega par Monte Carlo

En l'absence de formule fermée, ou dans des cadres plus généraux que Black-Scholes, le Vega peut être estimé par différences finies via la méthode de Monte Carlo :

$$\text{Vega} \approx \frac{F(\sigma + h) - F(\sigma - h)}{2h}$$

où chaque $F(\sigma \pm h)$ correspond à une estimation Monte Carlo du prix de l'option pour une volatilité légèrement perturbée. Cette approche permet de capturer l'effet marginal de la volatilité sur le prix de l'option, tout en restant robuste même lorsque la distribution sous-jacente est non gaussienne ou asymétrique.

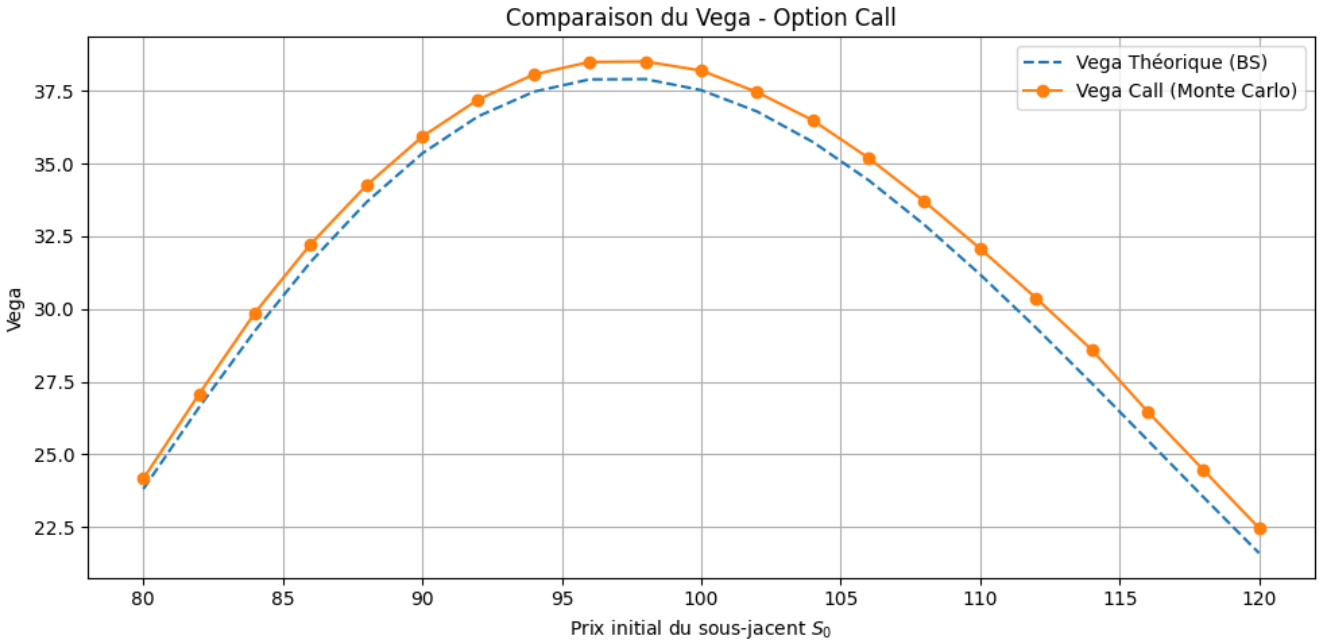
Choix du pas h. Comme pour le Delta, le choix de h est crucial : un pas trop petit augmente le bruit statistique ; un pas trop grand biaise l'estimation de la dérivée. Typiquement, on utilise un $h \in [0.01, 0.1]$ selon les unités.

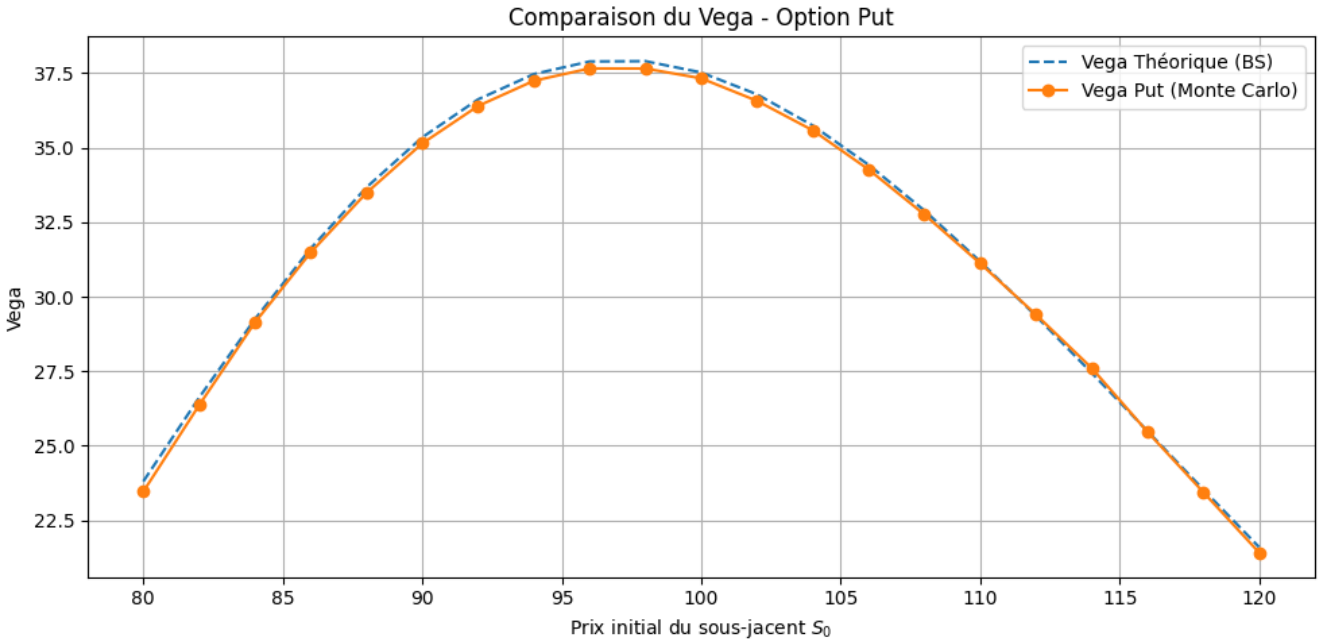
5.4.5 Code C++ Vega

```
#include <iostream>
#include <cmath>
#include <random>
#include <fstream>
#include <iomanip>

double payoff_call(double ST, double K) { return std::max(ST - K, 0.0); }
double payoff_put(double ST, double K) { return std::max(K - ST, 0.0); }

double simulate_ST(double S0, double r, double sigma, double T, double Z) {
    return S0 * exp((r - 0.5 * sigma * sigma) * T + sigma * sqrt(T) * Z);
}

double monte_carlo_price(int N, double S0, double K, double r, double sigma,
double T, bool isCall) {
    std::mt19937 rng(42);
    std::normal_distribution<> norm(0.0, 1.0);
    double sum = 0.0;
```

```
    for (int i = 0; i < N; ++i) {
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        sum += isCall ? payoff_call(ST, K) : payoff_put(ST, K);
    }
    return exp(-r * T) * (sum / N);
}

double estimate_vega(int N, double S0, double K, double r, double sigma, double T,
double h, bool isCall) {
    double f_up   = monte_carlo_price(N, S0, K, r, sigma + h, T, isCall);
    double f_down = monte_carlo_price(N, S0, K, r, sigma - h, T, isCall);
    return (f_up - f_down) / (2.0 * h);
}

double bs_vega(double S0, double K, double r, double sigma, double T) {
    double d1 = (log(S0 / K) + (r + 0.5 * sigma * sigma) * T) / (sigma * sqrt(T));
    double phi = exp(-0.5 * d1 * d1) / sqrt(2.0 * M_PI);
    return S0 * phi * sqrt(T);
}

int main() {
    double K = 100.0, r = 0.05, sigma = 0.2, T = 1.0;
    int N = 10000;
    double h = 0.01;

    std::ofstream file("vega_vs_theoretical.csv");
    file << "S0,Vega_MC_Call,Vega_BS,Vega_MC_Put,Vega_BS\n";

    for (double S0 = 80.0; S0 <= 120.0; S0 += 2.0) {
        double vega_mc_call = estimate_vega(N, S0, K, r, sigma, T, h, true);
        double vega_mc_put  = estimate_vega(N, S0, K, r, sigma, T, h, false);
        double vega_bs      = bs_vega(S0, K, r, sigma, T);

        file << S0 << "," << vega_mc_call << "," << vega_bs << "," << vega_mc_put <<
        "," << vega_bs << "\n";
    }

    file.close();
    std::cout << "Fichier vega_vs_theoretical.csv généré avec succès.\n";
    return 0;
}
```

5.4.6 Résultats





- Le Vega atteint son maximum lorsque $S_0 \approx K$, c'est-à-dire lorsque l'option est *at-the-money*. Ce comportement est conforme à la théorie : l'incertitude sur l'exercice y est maximale, rendant le prix très sensible à une variation de volatilité.
- Ce pic du Vega reflète le rôle central de la volatilité implicite dans la valorisation des options ATM, où un léger changement de dispersion peut significativement modifier la probabilité d'exercice.

- La comparaison entre l'estimation par Monte Carlo et la valeur analytique issue du modèle de Black-Scholes montre une **concordance remarquable**, confirmant la robustesse de l'approche par différences finies centrées.

- Comme attendu, le profil du Vega est symétrique pour les options call et put dans le modèle Black-Scholes, la formule étant identique pour les deux types de produits. Cela souligne l'indépendance du Vega vis-à-vis du signe directionnel de l'option.

5.5 Theta

5.5.1 Définition Mathématique

Le **Theta** d'une option mesure la sensibilité du prix de l'option à l'écoulement du temps, toutes choses égales par ailleurs. Il est défini comme la dérivée partielle du prix de l'option par rapport au temps :

$$\Theta = \frac{\partial F}{\partial t},$$

où $F(t, S_t)$ est la valeur théorique de l'option à l'instant t, avec S_t le prix du sous-jacent. En pratique, on considère principalement le **Theta initial**, noté Θ_0, qui reflète l'érosion de la valeur de l'option dès l'origine de la position.

Dans le cadre du modèle de Black-Scholes, les expressions analytiques diffèrent selon la nature de l'option :

- Pour une **option call européenne** :

$$\Theta_{\text{call}} = -\frac{S_0 \varphi(d_1)\sigma}{2\sqrt{T}} - rKe^{-rT}\Phi(d_2),$$

- Pour une **option put européenne** :

$$\Theta_{\text{put}} = -\frac{S_0 \varphi(d_1)\sigma}{2\sqrt{T}} + rKe^{-rT}\Phi(-d_2),$$

avec :

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

et :

- $\varphi(d_1) = \frac{1}{\sqrt{2\pi}} e^{-d_1^2/2}$: densité de la loi normale standard,
- $\Phi(d_2)$: fonction de répartition de la loi normale,
- σ : volatilité du sous-jacent,
- r : taux d'intérêt sans risque,
- T : temps restant jusqu'à maturité,
- K : prix d'exercice.

5.5.2 Interprétation financière

Le Theta représente la **valeur temps** d'une option. En d'autres termes, il traduit la perte de valeur que subit une option chaque jour en l'absence de variation du sous-jacent et des autres paramètres.

- Le Theta est **négatif** pour la majorité des options longues, car l'incertitude diminue avec le temps — et donc le prix de l'option aussi.
- Pour un vendeur d'option, cette érosion représente un **gain latent** ; le Theta joue donc un rôle positif dans les stratégies de type short-volatility.
- Le Theta est **le plus fortement négatif à la monnaie** ($S_0 \approx K$), là où l'incertitude sur l'exercice est la plus élevée.
- En revanche, il devient plus faible (en valeur absolue) pour des options *deep ITM* ou *deep OTM*, qui ont peu ou pas de valeur temps.

Note stratégique. En gestion active, le Theta est particulièrement scruté pour calibrer des stratégies de type *calendar spreads*, *theta decay harvesting* ou pour monitorer le P&L de portefeuilles optionnels en position longue. Son rôle est donc central dans l'optimisation dynamique du risque temps.

5.5.3 Code C++ Theta

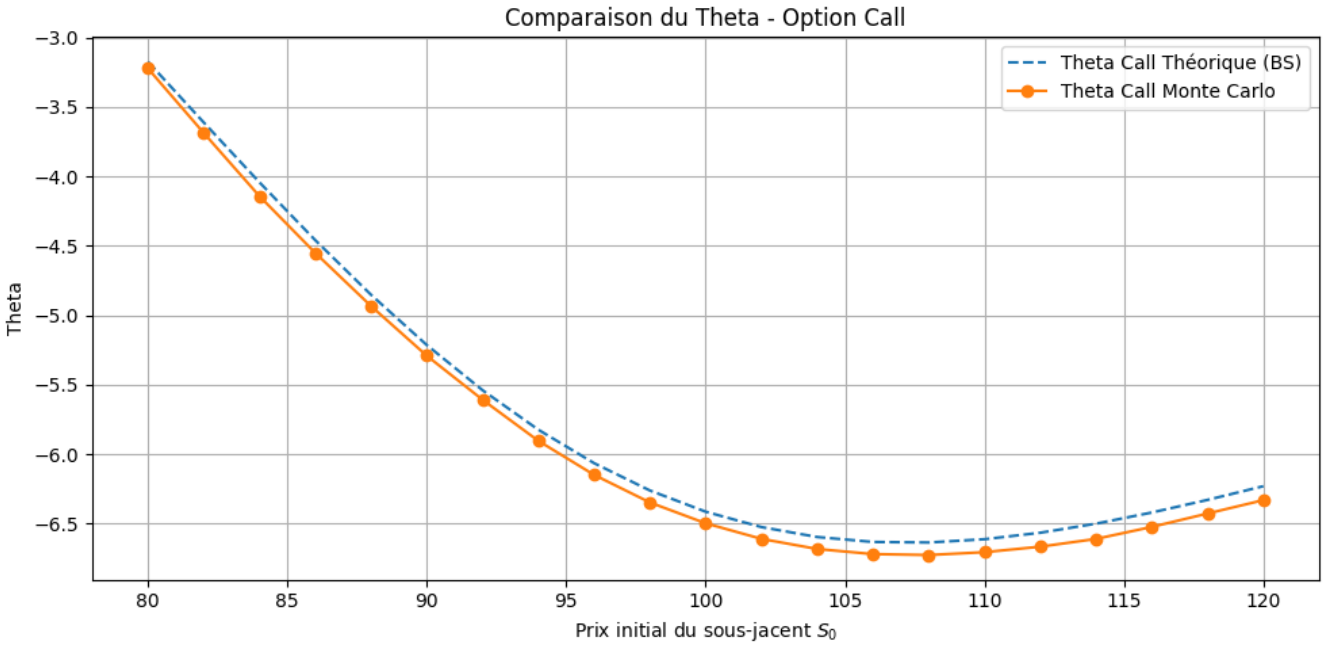
Nous présentons ci-dessous le code C++ permettant d'estimer le **Theta** d'une option européenne (call ou put) en utilisant la méthode de Monte Carlo avec différences finies centrées, et de le comparer à la valeur théorique issue du modèle de Black-Scholes.

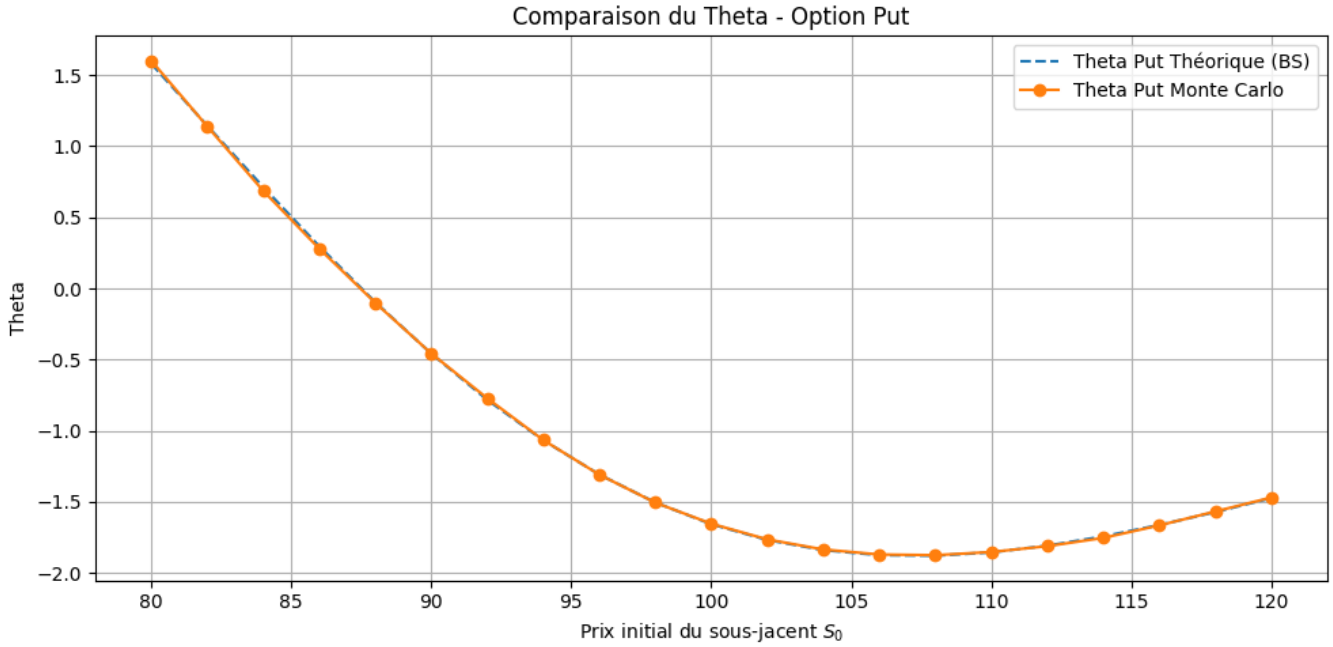
```
#include <iostream>
#include <cmath>
#include <random>
#include <fstream>
#include <iomanip>

// Payoffs
double payoff_call(double ST, double K) { return std::max(ST - K, 0.0); }
double payoff_put(double ST, double K) { return std::max(K - ST, 0.0); }

// Simulation de S_T sous Black-Scholes
double simulate_ST(double S0, double r, double sigma, double T, double Z) {
    return S0 * std::exp((r - 0.5 * sigma * sigma) * T + sigma * std::sqrt(T) * Z);
}
```

```
// Estimation du prix via Monte Carlo
double monte_carlo_price(int N, double S0, double K, double r, double sigma, double T
bool isCall) {
    std::mt19937 rng(42);
    std::normal_distribution<> norm(0.0, 1.0);
    double sum = 0.0;

    for (int i = 0; i < N; ++i) {
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        sum += isCall ? payoff_call(ST, K) : payoff_put(ST, K);
    }

    return std::exp(-r * T) * (sum / N);
}

// Estimation du Theta par différences finies centrées
double estimate_theta(int N, double S0, double K, double r, double sigma, double T,
double h, bool isCall) {
    double price_up   = monte_carlo_price(N, S0, K, r, sigma, T + h, isCall);
    double price_down = monte_carlo_price(N, S0, K, r, sigma, T - h, isCall);
    return (price_down - price_up) / (2.0 * h);  // attention à l'ordre
}

// Theta théorique Black-Scholes
double bs_theta(double S0, double K, double r, double sigma, double T, bool isCall) {
    double d1 = (std::log(S0 / K) + (r + 0.5 * sigma * sigma) * T) /
    (sigma * std::sqrt(T));
    double d2 = d1 - sigma * std::sqrt(T);
    double phi = std::exp(-0.5 * d1 * d1) / std::sqrt(2 * M_PI);
    double Nd2 = 0.5 * std::erfc(-d2 / std::sqrt(2));
    double nphi = S0 * phi * sigma / (2 * std::sqrt(T));

    if (isCall)
        return -nphi - r * K * std::exp(-r * T) * Nd2;
    else
        return -nphi + r * K * std::exp(-r * T) * (1 - Nd2);
}
```

```
int main() {
    double K = 100.0, r = 0.05, sigma = 0.2;
    int N = 10000;
    double h = 0.01;

    std::ofstream file("theta_vs_theoretical.csv");
    file << "S0,Theta_MC_Call,Theta_BS_Call,Theta_MC_Put,Theta_BS_Put\n";

    for (double S0 = 80.0; S0 <= 120.0; S0 += 2.0) {
        double T = 1.0;
        double theta_mc_call = estimate_theta(N, S0, K, r, sigma, T, h, true);
        double theta_mc_put  = estimate_theta(N, S0, K, r, sigma, T, h, false);
        double theta_bs_call = bs_theta(S0, K, r, sigma, T, true);
        double theta_bs_put  = bs_theta(S0, K, r, sigma, T, false);

        file << S0 << "," << theta_mc_call << "," << theta_bs_call << ","
                    << theta_mc_put  << "," << theta_bs_put  << "\n";
    }

    file.close();
    std::cout << "Fichier theta_vs_theoretical.csv généré avec succès.\n";
    return 0;
}
```

5.5.4 Résultat





Les courbes issues de la méthode de Monte Carlo (points orange) reproduisent de manière très satisfaisante les profils théoriques du Theta (courbes pointillées bleues) issus du modèle de Black-Scholes, en particulier pour l'option put.

On observe les comportements suivants :

- Pour le **call**, le Theta est strictement **négatif** sur l'ensemble des valeurs de S_0, ce qui traduit la déperdition inéluctable de la valeur temps à mesure que l'échéance approche. Cette décroissance est plus marquée lorsque l'option est *at-the-money*.
- Pour le **put**, le Theta est généralement négatif mais peut devenir légèrement **positif** en situation de *deep out-of-the-money*. Ce phénomène reflète une composante technique : dans certains cas, la réduction du temps jusqu'à l'échéance accroît marginalement la valeur temps d'un put très éloigné du strike, en raison de la convexité accrue de sa probabilité d'exercice.
- La précision de l'estimation Monte Carlo est globalement satisfaisante, avec une erreur numérique contenue, hormis à proximité de la monnaie où la courbure du Theta est plus accentuée et où les différences finies amplifient le bruit statistique.

5.6 Rho

5.6.1 Définition Mathématique.

Le **Rho** d'une option représente la sensibilité du prix de l'option au taux d'intérêt sans risque r. Il s'agit formellement de la dérivée partielle du prix de l'option par rapport à r :

$$\rho_{\text{call}} = \frac{\partial C}{\partial r}, \quad \rho_{\text{put}} = \frac{\partial P}{\partial r}$$

Formules analytiques dans le modèle de Black-Scholes. Sous les hypothèses classiques de Black-Scholes, les expressions fermées du Rho pour des options européennes sont données par :

- Pour une option **Call** :
$$\rho_{\text{call}} = TKe^{-rT}\mathcal{N}(d_2)$$
- Pour une option **Put** :
$$\rho_{\text{put}} = -TKe^{-rT}\mathcal{N}(-d_2)$$

avec :

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

où $\mathcal{N}$ est la fonction de répartition de la loi normale standard.

5.6.2 Interprétation financière.

- Le Rho indique l'impact d'une variation marginale du taux sans risque sur la valeur d'une option.
- Pour un **call**, le Rho est typiquement **positif** : une hausse des taux augmente la valeur présente des gains potentiels à l'échéance.
- Pour un **put**, le Rho est **négatif** : la hausse des taux rend moins attractif le payoff d'un put à maturité.
- Cet effet est amplifié pour les options de **longue maturité**, car l'actualisation joue un rôle plus significatif.

5.6.3 Estimation du Rho par la méthode de Monte Carlo

Nous approchons numériquement le Rho via une méthode de différences finies centrées. Cette méthode consiste à calculer le prix de l'option pour des taux $r \pm h$, et à en déduire une estimation de la dérivée :

$$\rho \approx \frac{F(r+h) - F(r-h)}{2h}$$

où :

- $F(r)$ est le prix estimé de l'option obtenu par simulation Monte Carlo,
- h est une perturbation infinitésimale typiquement choisie entre 10^{-3} et 10^{-2}.

Remarque technique. Cette approche est robuste, mais sa précision dépend du choix de h et du nombre de trajectoires N. Un h trop petit augmente le bruit de simulation ; un h trop grand introduit un biais de linéarisation.

Utilisation en pratique. Le Rho est souvent négligé dans les stratégies à court terme, car son effet est limité. En revanche, il devient crucial pour :

- la gestion des books d'options à longue échéance (swaptions, exotiques, caps/floors),
- l'évaluation d'impact de mouvements de courbe de taux sur les portefeuilles optionnels,
- la calibration des modèles à taux d'intérêt stochastique dans les desks de structuration taux.

5.6.4 Code C++ Rho

```
#include <iostream>
#include <cmath>
#include <random>
#include <fstream>
#include <iomanip>

double payoff_call(double ST, double K) { return std::max(ST - K, 0.0); }
double payoff_put(double ST, double K)  { return std::max(K - ST, 0.0); }

double simulate_ST(double S0, double r, double sigma, double T, double Z) {
    return S0 * std::exp((r - 0.5 * sigma * sigma) * T + sigma * std::sqrt(T) * Z);
}

double monte_carlo_price(int N, double S0, double K, double r, double sigma,
double T, bool isCall) {
    std::mt19937 rng(42);
    std::normal_distribution<> norm(0.0, 1.0);
    double sum = 0.0;
    for (int i = 0; i < N; ++i) {
```

```
        double Z = norm(rng);
        double ST = simulate_ST(S0, r, sigma, T, Z);
        sum += isCall ? payoff_call(ST, K) : payoff_put(ST, K);
    }
    return std::exp(-r * T) * (sum / N);
}

double estimate_rho(int N, double S0, double K, double r, double sigma, double T,
double h, bool isCall) {
    double price_up   = monte_carlo_price(N, S0, K, r + h, sigma, T, isCall);
    double price_down = monte_carlo_price(N, S0, K, r - h, sigma, T, isCall);
    return (price_up - price_down) / (2.0 * h);
}

double bs_rho(double S0, double K, double r, double sigma, double T, bool isCall) {
    double d2 = (std::log(S0 / K) + (r - 0.5 * sigma * sigma) * T) /
    (sigma * std::sqrt(T));
    double Nd2 = 0.5 * std::erfc(-d2 / std::sqrt(2));
    double Nnd2 = 0.5 * std::erfc(d2 / std::sqrt(2)); // N(-d2)
    if (isCall)
        return T * K * std::exp(-r * T) * Nd2;
    else
        return -T * K * std::exp(-r * T) * Nnd2;
}

int main() {
    double K = 100.0, sigma = 0.2, T = 1.0;
    int N = 10000;
    double h = 0.01;

    std::ofstream file("rho_vs_theoretical.csv");
    file << "S0,Rho_MC_Call,Rho_BS_Call,Rho_MC_Put,Rho_BS_Put\n";

    for (double S0 = 80.0; S0 <= 120.0; S0 += 2.0) {
        double r = 0.05;
        double rho_mc_call = estimate_rho(N, S0, K, r, sigma, T, h, true);
        double rho_mc_put  = estimate_rho(N, S0, K, r, sigma, T, h, false);
        double rho_bs_call = bs_rho(S0, K, r, sigma, T, true);
        double rho_bs_put  = bs_rho(S0, K, r, sigma, T, false);

        file << S0 << "," << rho_mc_call << "," << rho_bs_call << "," <<
        rho_mc_put << "," << rho_bs_put << "\n";
    }

    file.close();
    std::cout << "Fichier rho_vs_theoretical.csv généré avec succès.\n";
    return 0;
}
```

5.6.5 Résultats

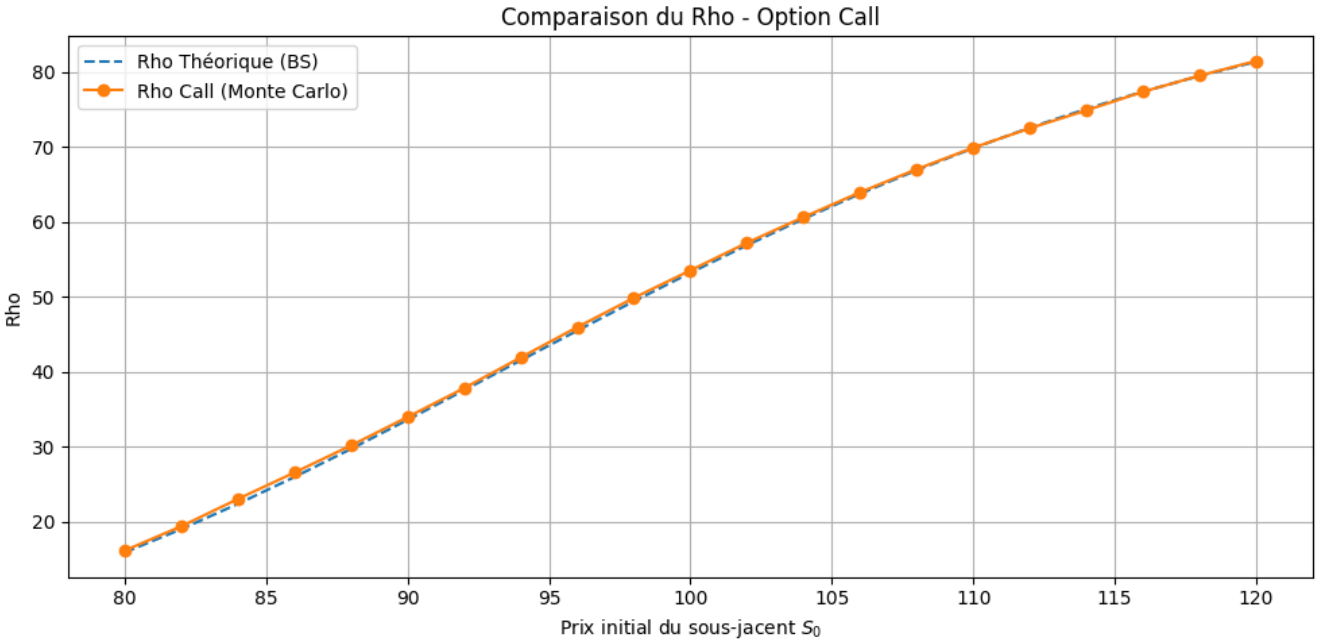


Figure 5.3: Rho du Call : comparaison Monte Carlo vs formule de Black-Scholes

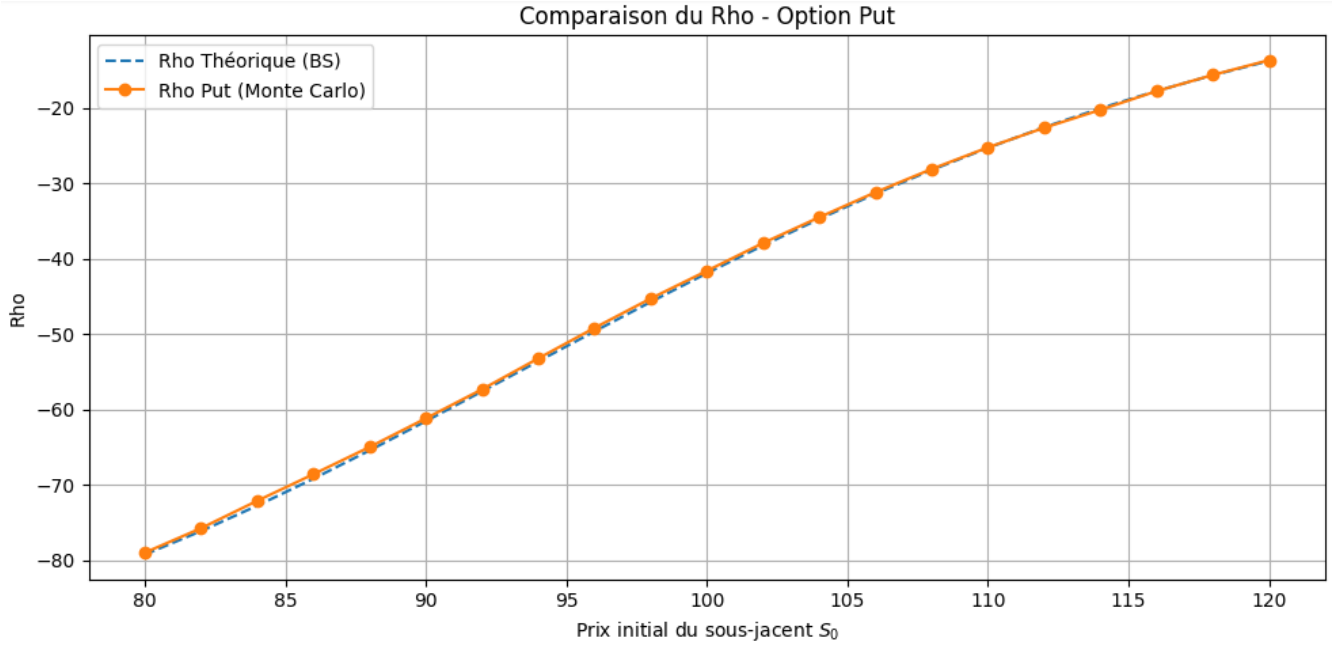


Figure 5.4: Rho du Put : comparaison Monte Carlo vs formule de Black-Scholes

Les graphiques illustrent une excellente adéquation entre l'estimation de ρ par Monte Carlo (simulation par différences finies) et les valeurs théoriques issues de la formule fermée de Black-Scholes. On observe les comportements suivants :

- Pour une **option Call**, le Rho est **positif et croissant** avec S_0. Cela traduit le fait qu'une hausse du taux sans risque augmente la valeur actuelle des gains potentiels à maturité.
- Pour une **option Put**, le Rho est **négatif et décroissant**. Cela s'explique par la baisse de la valeur actualisée du prix d'exercice K, qui pénalise le payoff du put.
- Dans les deux cas, la sensibilité est plus marquée pour les options **in-the-money** et les maturités longues, comme attendu du terme multiplicatif $T \cdot Ke^{-rT}$ dans la formule analytique.

Validation du modèle. Cette concordance entre la simulation Monte Carlo et la théorie analytique confirme la validité de la méthode numérique employée. Elle souligne également l'importance du Rho dans les environnements de taux volatils ou pour les produits à maturité élevée (par exemple : swaptions, options longues sur taux ou equity).

Conclusion. Le Rho, bien qu'en général négligé pour les options à maturité courte, constitue une Greek non négligeable dans la structuration de produits sensibles aux taux d'intérêt. Sa maîtrise est indispensable pour les desks de taux, les market makers et les gestionnaires d'options longue durée.

5.7 Conclusion sur l'estimation des Greeks

L'analyse numérique conduite dans ce chapitre a permis d'estimer les cinq sensibilités fondamentales — appelées **Greeks** — des options européennes vis-à-vis des paramètres clés du modèle de Black-Scholes. L'approche retenue repose sur des différences finies appliquées à des simulations Monte Carlo, ce qui permet une approximation efficace même en l'absence de formules analytiques.

- **Delta** quantifie la sensibilité du prix de l'option à une variation du prix initial du sous-jacent. Il est central dans la gestion dynamique des portefeuilles via des stratégies de couverture (*delta-hedging*).
- **Gamma** représente la dérivée seconde du prix par rapport au sous-jacent. Il capture la convexité du payoff et renseigne sur la stabilité du Delta.
- **Vega** mesure l'impact d'une variation de la volatilité implicite. Il est maximal lorsque l'option est *at-the-money*, ce qui reflète l'incertitude maximale sur l'exercice.
- **Theta** évalue l'érosion du prix de l'option au fil du temps, souvent appelée *time decay*. Il est généralement négatif pour les détenteurs d'options.
- **Rho** quantifie la sensibilité à une variation du taux d'intérêt sans risque. Il devient significatif pour les maturités longues.

Les résultats numériques montrent une concordance remarquable entre les estimations Monte Carlo et les valeurs exactes issues des formules de Black-Scholes. Cette validation empirique confirme que, malgré une variance potentielle, l'approche Monte Carlo fournit des approximations fiables à condition d'utiliser :

- un pas h de perturbation adéquatement calibré,
- un nombre de simulations N suffisamment grand pour réduire l'erreur statistique.

Résumé : La méthode Monte Carlo permet une estimation robuste des Greeks, même dans des contextes où les formules fermées ne sont pas disponibles, comme pour les options exotiques ou sous des dynamiques plus complexes (volatilité stochastique, taux aléatoires, barrières, etc.).

Perspectives : Ce cadre d'analyse peut être enrichi par :

- l'intégration de techniques de réduction de variance (antithétiques, variables de contrôle, stratification),
- la généralisation à des modèles de volatilité non constante (Heston, SABR),
- ou l'application aux produits à flux multiples (ex. caps/floors, swaptions) via des extensions comme Hull-White ou CIR.

Table 5.1: Résumé des principaux Greeks et de leur interprétation

Greek	**Sensibilité à**	**Effet principal**	**Signe typique**
Δ	Prix du sous-jacent S_0	Variation linéaire du prix de l'option	Call > 0, Put < 0
Γ	S_0	Courbure de la prime ($\partial^2 F/\partial S^2$)	Toujours > 0
$\mathcal{V}$ (Vega)	Volatilité σ	Sensibilité au risque de volatilité	Call / Put > 0
Θ	Temps t	Décroissance de la valeur temps	Généralement < 0
ρ	Taux d'intérêt r	Effet de l'environnement de taux	Call > 0, Put < 0

5.8 Conclusion générale

Ce projet avait pour objectif de comparer différentes méthodes numériques d'évaluation du prix d'options européennes, en confrontant d'une part les approches par **simulation Monte Carlo**, et d'autre part les **méthodes de différences finies** appliquées à l'équation de Black-Scholes.

Dans un premier temps, nous avons mis en œuvre la méthode Monte Carlo pour simuler le prix d'options plain vanilla (call et put) sous le modèle de Black-Scholes. L'étude a été complétée par une analyse approfondie des **Greeks**, dont les estimations par différences finies ont été confrontées aux valeurs exactes du modèle. Cette approche s'est révélée efficace pour le **Delta**, le **Vega** et le **Rho**, tandis que le **Gamma** et le **Theta** nécessitent un plus grand nombre de simulations pour stabiliser les résultats.

Dans un second temps, nous avons résolu numériquement l'équation de Black-Scholes en utilisant les méthodes :

- **explicite**, simple mais conditionnellement stable,
- **implicite**, inconditionnellement stable mais plus coûteuse,
- **Crank-Nicolson**, combinant stabilité et second ordre de précision en temps et en espace.

Les résultats numériques obtenus confirment la cohérence des trois schémas, avec une convergence effective vers la solution théorique lorsque le maillage est raffiné. Le schéma de Crank-Nicolson s'est montré particulièrement performant, avec une meilleure précision pour un coût numérique modéré.

Perspectives et extensions

Ce travail constitue une base solide pour des prolongements variés :

- **Réduction de variance** : l'intégration de techniques classiques (variables antithétiques, variables de contrôle) ou avancées (stratification, importance sampling) permettrait d'améliorer significativement l'efficacité de la méthode Monte Carlo.
- **Options exotiques** : les options à barrière, asiatiques ou lookback peuvent être abordées par Monte Carlo ou via des schémas de différences finies adaptés, selon la structure du payoff.
- **Schémas numériques avancés** : l'étude d'extensions implicites à temps variable, la méthode ADI (Alternating Direction Implicit), ou la discrétisation adaptative en espace et temps permettrait d'optimiser la précision dans les zones critiques ($S \approx K$).
- **Modèles alternatifs** : l'implémentation de modèles plus réalistes (Heston, SABR, Merton, Hull-White) permettrait de comparer les performances des différentes méthodes dans un contexte de calibration ou de pricing sur des marchés incomplets.
- **Implémentation parallèle** : l'utilisation de librairies optimisées (OpenMP, CUDA, TBB) ou l'intégration dans un moteur C++ vectorisé (type QuantLib) ouvrirait la voie à une utilisation industrielle à grande échelle.

Bibliography

[1] Paul Wilmott, *Introduction to Quantitative Finance*, Wiley.

[2] Daniel J. Duffy, *Financial Instrument Pricing Using C++*, Wiley.

[3] Mark S. Joshi, *The Concepts and Practice of Mathematical Finance*.

[4] Gilles Pagès, *Numerical Probability*.

[5] C. C. Heyde, S. T. Rachev et al., *Handbook of Computational Finance*, Springer.

[6] QuantStart.com