# Implémentation d'un Pricer Black-Scholes par Différences Finies

**Abahamou Ziad / El Azri Amine**

Janvier 2025

**Résumé**

Ce document présente le développement complet d'un pricer d'options européennes basé sur l'équation de Black-Scholes. L'approche combine théorie mathématique et implémentation pratique en C++, avec un focus particulier sur l'architecture logicielle, les structures de données optimisées, et la visualisation graphique des résultats. Deux méthodes numériques sont implémentées et comparées : le schéma implicite pour l'EDP réduite et le schéma de Crank-Nicolson pour l'EDP complète.

## Table des matières

# 1 Introduction

## 1.1 Contexte du projet

Le pricing d'options financières constitue un domaine fondamental de la finance quantitative. L'équation de Black-Scholes fournit un cadre théorique rigoureux pour évaluer ces instruments dérivés, mais nécessite des méthodes numériques pour obtenir des solutions pratiques.

## 1.2 Objectifs

Ce projet poursuit trois objectifs principaux :

1. Implémenter deux approches numériques pour résoudre l'équation de Black-Scholes
2. Concevoir une architecture logicielle modulaire et extensible en C++
3. Développer un système de visualisation graphique pour analyser les résultats

# 2 Fondements Mathématiques

## 2.1 Le modèle de Black-Scholes

### 2.1.1 Équation complète

L'équation de Black-Scholes gouvernant le prix $C(t, S)$ d'une option s'écrit :

$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} - rC = 0 \tag{1}$$

où :

— $C(t, S)$ : prix de l'option au temps $t$ pour un sous-jacent de valeur $S$
— $r$ : taux d'intérêt sans risque
— $\sigma$ : volatilité du sous-jacent
— $T$ : maturité de l'option

Cette EDP combine trois termes : transport ($rS\frac{\partial C}{\partial S}$), diffusion ($\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2}$), et décroissance exponentielle ($-rC$).

### 2.1.2 Équation réduite

Par une série de transformations (passage en coordonnées logarithmiques, normalisation temporelle, et décalage convectif), l'équation (1) peut être ramenée à une forme simplifiée :

$$\frac{\partial \widetilde{C}}{\partial \widetilde{t}} = \mu \frac{\partial^2 \widetilde{C}}{\partial \widetilde{S}^2} \tag{2}$$

Cette équation de diffusion pure est structurellement identique à l'équation de la chaleur, avec $\mu = \frac{1}{2}\sigma^2$ comme coefficient de diffusion.

## 2.2 Discrétisation du domaine

### 2.2.1 Maillage

Le domaine de résolution $\mathcal{D} = [0, T] \times [0, L]$ est discrétisé uniformément :

$$\text{Maillage temporel : } t_m = m\Delta t, \quad m \in [\![0, M]\!], \quad \Delta t = \frac{T}{M}$$

$$\text{Maillage spatial : } s_j = j\Delta s, \quad j \in [\![0, N]\!], \quad \Delta s = \frac{L}{N}$$

On note $\mathbf{C}_j^m \approx C(t_m, s_j)$ la valeur approchée de l'option au nœud $(m, j)$.

### 2.2.2 Conditions aux limites

Le problème nécessite :

— **Condition terminale** (en $t = T$) : donnée par le payoff de l'option

$$\text{Call} : C(T, S) = \max(S - K, 0)$$
$$\text{Put} : C(T, S) = \max(K - S, 0)$$

— **Conditions aux bords** (en $S = 0$ et $S = L$) :

$$\text{Put} : C(t, 0) = Ke^{-r(T-t)}, \quad C(t, L) = 0$$
$$\text{Call} : C(t, 0) = 0, \quad C(t, L) = L - Ke^{-r(T-t)}$$

# 3 Méthodes de Différences Finies

## 3.1 Schéma implicite pour l'EDP réduite

### 3.1.1 Principe et stabilité

Le schéma implicite évalue les dérivées spatiales au temps futur $t_{m+1}$, garantissant une stabilité inconditionnelle. Pour l'équation (2), la discrétisation donne :

$$\frac{\mathbf{C}_j^{m+1} - \mathbf{C}_j^m}{\Delta t} = \mu \frac{\mathbf{C}_{j+1}^{m+1} - 2\mathbf{C}_j^{m+1} + \mathbf{C}_{j-1}^{m+1}}{(\Delta s)^2} \tag{3}$$

### 3.1.2 Formulation matricielle

En posant $\alpha = \frac{\mu \Delta t}{(\Delta s)^2}$, le réarrangement conduit à :

$$\boxed{-\alpha \mathbf{C}_{j-1}^{m+1} + (1 + 2\alpha)\mathbf{C}_j^{m+1} - \alpha \mathbf{C}_{j+1}^{m+1} = \mathbf{C}_j^m} \tag{4}$$

Pour les nœuds intérieurs $j \in [\![1, N-1]\!]$, on obtient le système matriciel :

$$M\mathbf{C}^{m+1} + \mathbf{k}_{m+1} = \mathbf{C}^m \tag{5}$$

où $M \in \mathcal{M}_{N-1}(\mathbb{R})$ est tridiagonale :

$$M = \begin{pmatrix} 1+2\alpha & -\alpha & 0 & \cdots & 0 \\ -\alpha & 1+2\alpha & -\alpha & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\alpha & 1+2\alpha & -\alpha \\ 0 & \cdots & 0 & -\alpha & 1+2\alpha \end{pmatrix}$$

Le vecteur $\mathbf{k}_{m+1}$ contient les termes de bord.

Cette formulation matricielle permet d'exprimer le schéma implicite sous la forme $M\mathbf{C}^{m+1} + \mathbf{k}_{m+1} = \mathbf{C}^m$, où $M$ est une matrice tridiagonale constante dans le temps. Dans l'implémentation, nous n'allons pas stocker cette matrice sous forme pleine, mais uniquement ses trois diagonales (sous-diagonale, diagonale principale, sur-diagonale) pour optimiser l'utilisation mémoire et faciliter la résolution par l'algorithme de Thomas.

## 3.2 Schéma de Crank-Nicolson pour l'EDP complète

### 3.2.1 Principe du $\theta$-schéma

Crank-Nicolson est un $\theta$-schéma avec $\theta = \frac{1}{2}$, moyennant les approches explicite et implicite. Cette méthode atteint une précision d'ordre 2 en temps.

### 3.2.2 Discrétisation

Les dérivées sont approximées par des moyennes arithmétiques :

$$\begin{aligned}
\frac{\partial C}{\partial t} &\simeq \frac{\mathbf{C}_j^{m+1} - \mathbf{C}_j^m}{\Delta t} \\
\frac{\partial C}{\partial S} &\simeq \frac{1}{4\Delta s}\left[(\mathbf{C}_{j+1}^{m+1} - \mathbf{C}_{j-1}^{m+1}) + (\mathbf{C}_{j+1}^m - \mathbf{C}_{j-1}^m)\right] \\
\frac{\partial^2 C}{\partial S^2} &\simeq \frac{1}{2(\Delta s)^2}\Big[(\mathbf{C}_{j+1}^{m+1} - 2\mathbf{C}_j^{m+1} + \mathbf{C}_{j-1}^{m+1}) \\
&\qquad +(\mathbf{C}_{j+1}^m - 2\mathbf{C}_j^m + \mathbf{C}_{j-1}^m)\Big]
\end{aligned}$$

### 3.2.3 Coefficients matriciels

L'injection dans (1) et le réarrangement conduisent à un système :

$$M_1 \mathbf{C}^{m+1} + \mathbf{k}_{m+1} = M_2 \mathbf{C}^m \tag{6}$$

avec des coefficients dépendant de l'indice spatial $j$ :

$$\begin{aligned}
a_j &= \frac{j\Delta t}{4}(\sigma^2 j - r) \\
b_j &= 1 - \frac{1}{2}\sigma^2 j^2 \Delta t \\
c_j &= \frac{j\Delta t}{4}(\sigma^2 j + r) \\
d_j &= 1 + \frac{1}{2}\sigma^2 j^2 \Delta t + r\Delta t
\end{aligned}$$

Les matrices $M_1$ et $M_2$ sont tridiagonales :
— $M_1$ : diagonale $(b_j)$, sous/sur-diagonales $(a_j, c_j)$
— $M_2$ : diagonale $(d_j)$, sous/sur-diagonales $(-a_j, -c_j)$

Les deux méthodes conduisent à des systèmes tridiagonaux de la forme $Ax = b$. L'algorithme de Thomas est une méthode directe de résolution de systèmes linéaires tridiagonaux. Il est particulièrement adapté aux schémas de différences finies qui génèrent naturellement de telles matrices. Pour l'EDP complète discrétisée via Crank–Nicolson, nous obtenons également un système tridiagonal à chaque pas de temps, que nous résolvons efficacement avec cet algorithme. Nous l'avons choisi pour sa simplicité d'implémentation et son adéquation parfaite à la structure tridiagonale de nos matrices.

# 4 Conception Logicielle

## 4.1 Architecture du projet

L'implémentation repose sur une hiérarchie de classes modulaire exploitant l'héritage et le polymorphisme :
— `class Mesh` : gestion des discrétisations
— `class Payoff` (abstraite) : interface pour les payoffs
  — `class Call`, `class Put` : implémentations concrètes
— `class Option` : conteneur de paramètres
— `class PDE` (abstraite) : définition des conditions aux limites
  — `class CompletePDE`, `class ReducedPDE`

— `class FiniteDifference` (abstraite) : interface des solveurs
  — `class IMFD` : schéma implicite
  — `class CrankNicholsonFD` : schéma de Crank-Nicolson

## 4.2 Diagramme UML

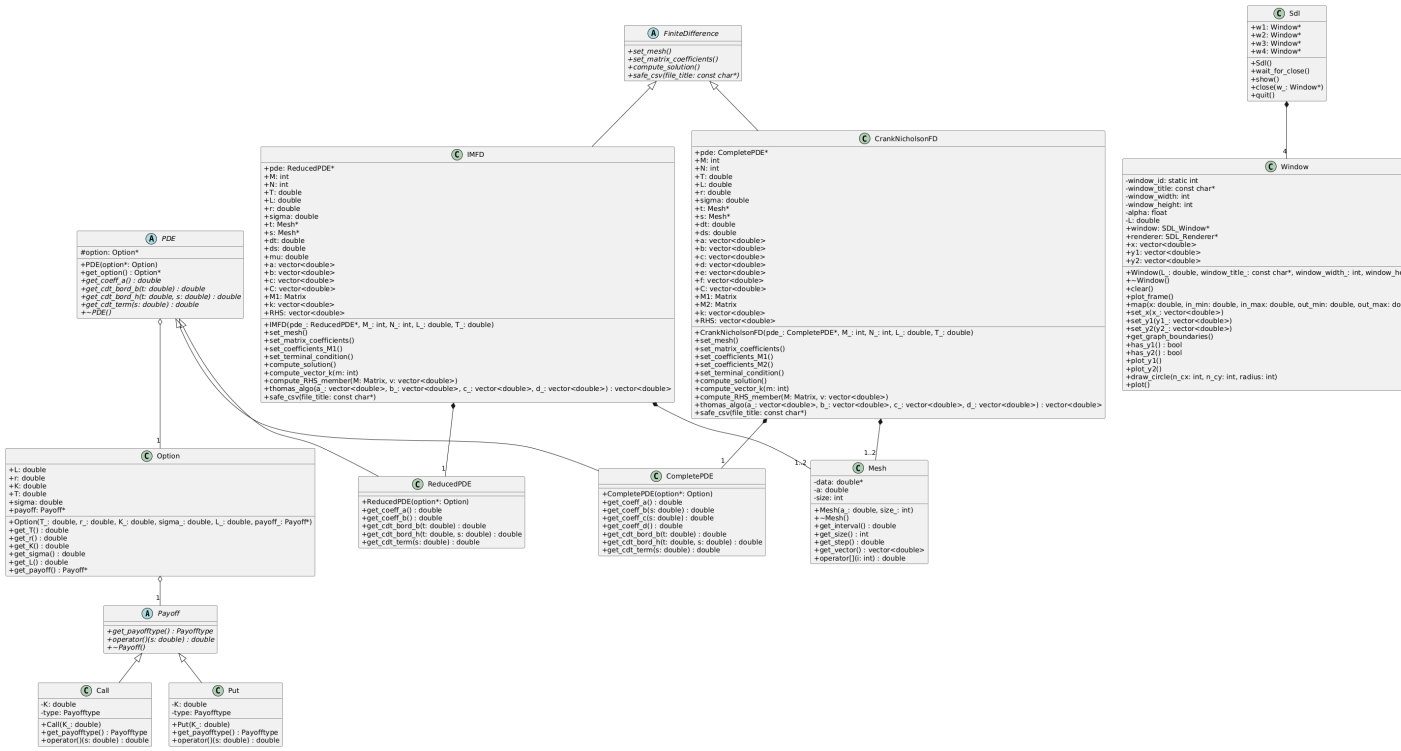


FIGURE 1 – Diagramme UML des classes de ce projet

# 5 Implémentation C++

### 5.0.1 Structure de base

Les payoffs sont implémentés via une classe abstraite et héritage, permettant de définir différentes fonctions de paiement :

```cpp
class Payoff {
public:
    virtual Payofftype get_payofftype() const = 0;
    virtual double operator()(const double& s) const = 0;
    virtual ~Payoff() {}
};
```

### 5.0.2 Implémentation des options Call et Put

```cpp
class Call : public Payoff {
private:
    double K;
    Payofftype type;

public:
```

```
    Call(const double& K_ = 0.0) : K(K_), type(Payofftype::call) {}

    Payofftype get_payofftype() const override {
        return type;
    }

    double operator()(const double& s) const override {
        return std::max(s - K, 0.0);
    }
};

class Put : public Payoff {
private:
    double K;
    Payofftype type;

public:
    Put(const double& K_) : K(K_), type(Payofftype::put) {}

    Payofftype get_payofftype() const override {
        return type;
    }

    double operator()(const double& s) const override {
        return std::max(K - s, 0.0);
    }
};
```

**Avantages de cette approche** :

— Extensibilité facile pour ajouter de nouveaux types d'options
— Polymorphisme permettant de traiter uniformément différents payoffs
— Encapsulation des règles métier financières

### 5.1 Classe Option : Conteneur de paramètres

La classe `Option` regroupe tous les paramètres nécessaires à la définition d'une option européenne :

```
class Option {
public:
    double L;
    double r;
    double K;
    double T;
    double sigma;
    Payoff* payoff;

public:
    Option(const double T_, const double r_, const double K_,
           const double sigma_, const double L_, Payoff* payoff_) {
        T = T_;
        r = r_;
        K = K_;
        sigma = sigma_;
        L = L_;
        payoff = payoff_;
    }

```

```
    double get_T() const { return T; }
    double get_r() const { return r; }
    double get_K() const { return K; }
    double get_sigma() const { return sigma; }
    double get_L() const { return L; }
    Payoff* get_payoff() const { return payoff; }
};
```

**Objectif** : Centraliser les paramètres financiers pour éviter leur dispersion dans le code.

## 5.2 Classes PDE : Modélisation mathématique

### 5.2.1 Interface abstraite

La classe `PDE` définit l'interface commune pour les deux formes de l'équation de Black-Scholes :

```
class PDE {
protected:
    Option* option;

public:
    PDE(Option* option_) : option(option_) {}

    virtual double get_coeff_a() const = 0;
    virtual double get_cdt_bord_b(double t) const = 0;
    virtual double get_cdt_bord_h(double t, double s) const = 0;
    virtual double get_cdt_term(double s) const = 0;
    virtual ~PDE() {}
};
```

### 5.2.2 EDP complète

Pour l'EDP complète de Black-Scholes, nous devons calculer les coefficients variables :

```
double CompletePDE::get_coeff_b(double s) const {
    return 0.5 * pow((option->sigma), 2.0) * pow(s, 2.0);
}

double CompletePDE::get_coeff_c(double s) const {
    return (option->r) * s;
}
```

### 5.2.3 EDP réduite

Pour l'EDP réduite (forme de la chaleur), les coefficients sont constants :

```
double ReducedPDE::get_coeff_b() const {
    return -0.5 * pow(option->sigma, 2);
}
```

### 5.2.4 Conditions aux limites

Les conditions aux bords sont déterminées dynamiquement selon le type de payoff :

```
double CompletePDE::get_cdt_bord_b(double t) const {
    if (option->payoff->get_payofftype() == Payofftype::call)
```

```
        return 0.0;
    else
        return (option->K) * exp(-(option->r) * (t - option->T));
}

double CompletePDE::get_cdt_bord_h(double t, double s) const {
    if (option->payoff->get_payofftype() == Payofftype::call)
        return s - (option->K) * exp(-(option->r) * (t - option->T));
    else
        return 0.0;
}
```

### 5.3 Classes FiniteDifference : Résolution numérique

La résolution numérique proprement dite est encapsulée dans les classes `IMFD` (schéma implicite) et `CrankNicholsonFD` (schéma de Crank-Nicolson). Ces classes héritent d'une interface commune `FiniteDifference` qui définit les opérations essentielles : initialisation du maillage, calcul des coefficients, résolution temporelle et export des résultats.

`IMFD` implémente le schéma implicite pour l'EDP réduite, avec des coefficients matriciels constants. `CrankNicholsonFD` gère le schéma du même nom pour l'EDP complète, avec des coefficients dépendant de la position spatiale. Les deux classes utilisent l'algorithme de Thomas pour résoudre les systèmes tridiagonaux à chaque pas de temps.

### 5.4 Classe Mesh : Gestion des discrétisations

La classe `Mesh` encapsule une discrétisation uniforme d'un intervalle. Elle génère automatiquement les points de discrétisation selon la formule $x_i = i \cdot \frac{a}{n}$, fournit un accès sécurisé avec vérification des bornes, et permet la conversion vers `std::vector<double>` pour l'interopérabilité avec les modules de visualisation.

### 5.5 Module de Visualisation Graphique (SDL2)

Le module graphique repose sur SDL2 et comprend deux classes principales :

— `Window` : Gère une fenêtre individuelle, le tracé des courbes et la conversion des coordonnées mathématiques en pixels

— `Sdl` : Coordonne l'affichage simultané de 4 fenêtres et gère les événements utilisateur

La principale difficulté a été la conversion des coordonnées réelles $(s, C) \in \mathbb{R}^2$ en pixels $(x, y) \in \mathbb{N}^2$. Nous avons implémenté une fonction de mapping qui préserve les proportions tout en créant des marges autour du graphique.

Le module permet d'afficher simultanément :

1. La superposition des solutions pour les options Put
2. L'erreur numérique pour les options Put
3. La superposition des solutions pour les options Call
4. L'erreur numérique pour les options Call

### 5.6 Programme principal "main.cpp"

Le fichier `main.cpp` orchestre l'ensemble du processus :

1. Définition des paramètres de simulation
2. Création des objets financiers (payoffs et options)
3. Initialisation des EDP

4. Création et exécution des solveurs
5. Export des résultats en CSV
6. Calcul des erreurs entre méthodes
7. Configuration et affichage graphique
8. Nettoyage mémoire

# 6 Résultats et Validation

## 6.1 Analyse des erreurs numériques

L'interface graphique développée avec SDL2 permet d'afficher simultanément quatre fenêtres offrant une analyse comparative complète des résultats. Cette organisation permet d'évaluer visuellement la qualité des solutions obtenues par les deux méthodes numériques pour les deux types d'options.

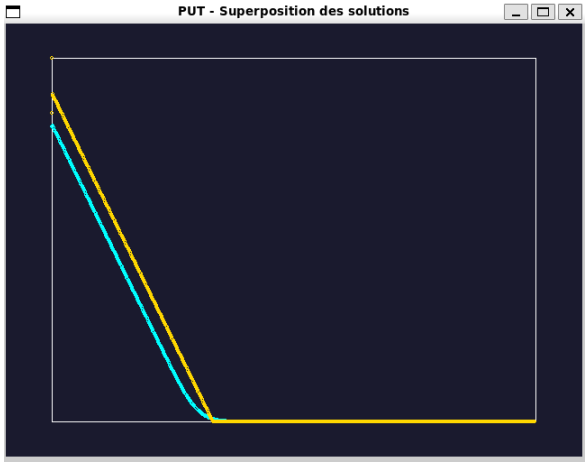


FIGURE 2 – Fenêtre 1 : Put - Superposition des solutions

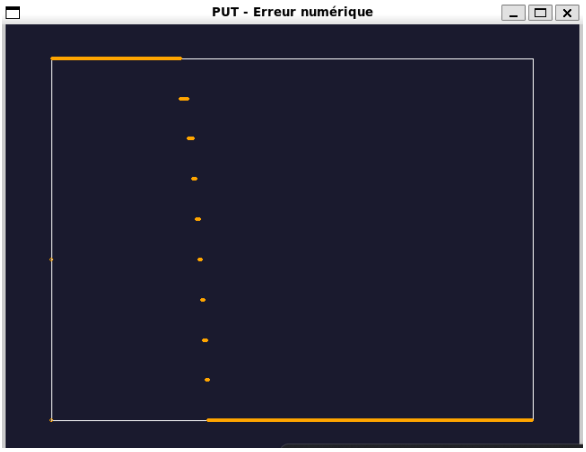


FIGURE 3 – Fenêtre 2 : Put - Erreur absolue

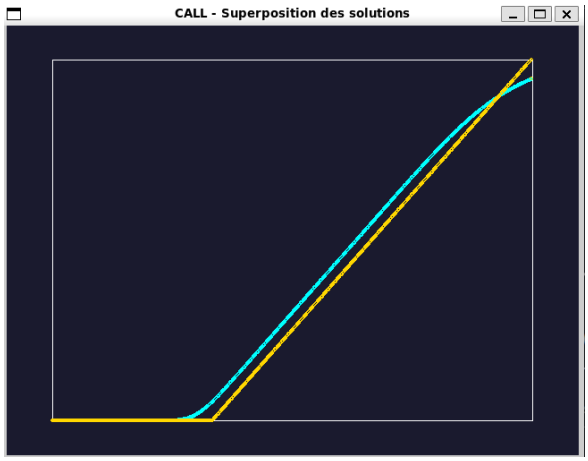


FIGURE 4 – Fenêtre 3 : Call - Superposition des solutions

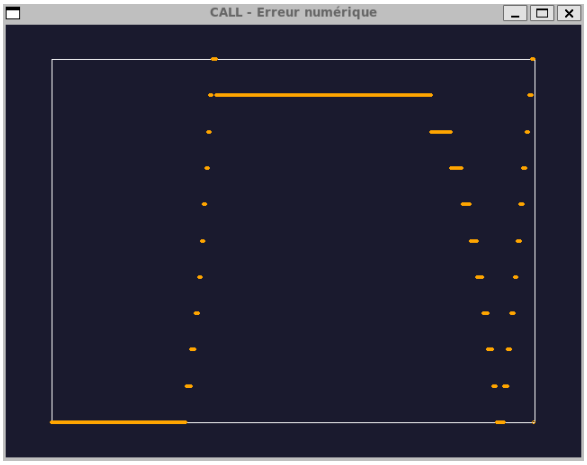


FIGURE 5 – Fenêtre 4 : Call - Erreur absolue

### 6.1.1 Fenêtres 1 et 3 : Superposition des solutions

Les fenêtres 1 (Put) et 3 (Call) présentent la superposition des deux solutions obtenues par les méthodes Crank-Nicolson (courbe cyan) et implicite (courbe jaune). La comparaison visuelle montre une excellente concordance qualitative entre les deux approches. Pour l'option Put (Fig. 2), on observe la décroissance caractéristique du prix avec l'augmentation du sous-jacent, tandis que pour l'option Call (Fig. 4), on constate la croissance attendue.

### 6.1.2 Fenêtres 2 et 4 : Analyse quantitative des erreurs

Les fenêtres 2 (Put) et 4 (Call) quantifient l'écart entre les deux méthodes en affichant l'erreur absolue $|C_{CN} - C_{IM}|$. L'analyse numérique révèle des caractéristiques intéressantes :

- **Option Put** : L'erreur moyenne s'élève à **2.726 unités**, indiquant un écart modéré entre les deux méthodes
- **Option Call** : L'erreur moyenne atteint **5.402 unités**, soit près du double de celle observée pour le Put
- **Erreur globale** : La moyenne pondérée sur les deux types d'options donne **4.064 unités**

Ces différences s'expliquent par la nature asymétrique des payoffs Put et Call. Pour le Put, la solution présente une décroissance régulière, tandis que le Call présente une croissance plus marquée, ce qui pourrait amplifier les écarts numériques.

La superposition visuelle satisfaisante des courbes cyan et jaune dans les fenêtres 1 et 3, combinée à ces erreurs numériques quantifiées, valide globalement notre implémentation. Les profils de prix obtenus correspondent aux attentes théoriques, avec des caractéristiques conformes à la littérature financière.