

Rapport de Projet P1RV

Optimisation et Parallélisation du Pricing d'Options Américaines

Méthode de Monte Carlo Longstaff-Schwartz sur CPU et GPU

Auteurs :

Florian BARBE
Narjisse EL MANSSOURI

École Centrale de Nantes
Année Universitaire 2025 – 2026

Table des matières

1	Introduction	3
2	Cadre théorique	4
2.1	Options américaines et problématique de l'exercice anticipé	4
2.2	Modélisation stochastique du sous-jacent	5
2.2.1	Mouvement brownien géométrique	5
2.2.2	Discrétisation temporelle	6
2.3	Algorithme de Longstaff–Schwartz (LSMC)	7
2.3.1	Valeur de continuation et problème d'arrêt optimal	7
2.3.2	Régression par moindres carrés	7
2.4	Synthèse algorithmique et pseudo-algorithme du LSMC	7
3	Travail réalisé : implémentation et méthodologie	11
3.1	Environnement de développement	11
3.2	Implémentation CPU séquentielle	11
3.3	Parallélisation CPU avec OpenMP	11
3.4	Accélération GPU avec CUDA	12
3.5	Méthodes de Différences Finies (FDM) pour comparaison	14
3.6	Structure du code et organisation des fichiers	15
4	Résultats et analyse des performances	16
4.1	Configuration matérielle	16
4.2	Première approche : comparaison des méthodes de parallélisme	16
4.3	Pour aller plus loin : analyse de l'impact de la base de régression	17
4.4	Validation numérique	18
4.5	Performances CPU séquentiel	18
4.6	Accélération par parallélisation CPU avec OpenMP	19
4.7	Accélération GPU avec CUDA	21
4.8	Comparaison globale des architectures	23
4.9	Discussion sur les limites du parallélisme	23
5	Analyse Critique et Limitations	25
5.1	Organisation du projet et évolution des dépôts	25
5.2	Échec de l'intégration CUDA dans Visual Studio	25
5.3	Complexité de l'exercice anticipé	25
5.4	Choix, stabilité et extensions de la régression	26
5.5	Compromis précision–performance	27
6	Perspectives et améliorations possibles	28
6.1	Améliorations algorithmiques	28
6.2	Extensions du modèle financier	28
6.3	Optimisations CPU avancées	28
6.4	Optimisation et généralisation de l'implémentation GPU	29
6.5	Perspectives méthodologiques	29
7	Organisation du travail	30
7.1	Découpage du projet	30
7.2	Organisation du développement et itérations	30
7.3	Répartition des tâches et binôme	32
7.4	Retours d'expérience et témoignages	32
7.5	Outils et environnement collaboratif	33
7.6	Gestion du temps et avancement	33

8 Conclusion	35
Annexes	36
A Résolution de l'EDS du GBM	36
B Rappels de Probabilités et propriétés du Mouvement Brownien	37
B.1 Définition et propriétés du Mouvement Brownien	37
B.2 Propriétés des incrément	37
B.3 Loi des Grands Nombres et Monte Carlo	37
C Propriétés des Bases Polynomiales et Relations de Récurrence	38
C.1 Base Canonique (Monomiale)	38
C.2 Polynômes de Laguerre $L_n(x)$	38
C.3 Polynômes d'Hermite $H_n(x)$	38
C.4 Polynômes de Legendre $P_n(x)$	38
C.5 Polynômes de Chebyshev (1ère espèce) $T_n(x)$	38
D Notations et Conventions	39
D.1 Probabilités et Processus Stochastiques	39
D.2 Modélisation Financière	39
D.3 Méthodes Numériques (LSMC)	39
D.4 Implémentation et Architecture	40

1 Introduction

Ce rapport présente le travail réalisé durant le premier semestre de la deuxième année de notre cursus ingénieur à Centrale Nantes, dans le cadre de notre projet P1RV. Ce projet s'inscrit dans une démarche d'analyse de performance et de mise en œuvre d'algorithmes numériques avancés, appliqués ici au domaine de la finance quantitative.

Le sujet central du projet porte sur l'évaluation et la comparaison de différentes stratégies de calcul pour le pricing d'options américaines, en s'appuyant sur l'algorithme LSMC (*Least Squares Monte Carlo*), également appelé méthode de Monte Carlo avec régression par moindres carrés ou méthode de Longstaff–Schwartz. Cette approche est aujourd'hui largement utilisée pour la valorisation de produits dérivés complexes lorsque les solutions analytiques fermées ne sont pas disponibles.

La valorisation des options américaines est intrinsèquement plus complexe que celle des options européennes en raison de la possibilité d'un exercice anticipé à tout instant avant la maturité. Cette caractéristique impose la détermination d'une politique d'exercice optimale, rendant inadaptées les méthodes classiques fondées uniquement sur des formules fermées (c'est-à-dire une formule explicite, finie, calculable directement et sans itération). L'algorithme LSMC tente de répondre à cette difficulté en combinant des simulations de Monte Carlo avec des régressions par moindres carrés, permettant d'estimer, à chaque date d'exercice possible, la valeur de continuation associée au maintien de l'option.

Au-delà de l'aspect théorique, ce projet s'inscrit dans une problématique de performance numérique, car les méthodes de Monte Carlo nécessitent la simulation d'un très grand nombre de trajectoires afin de garantir la convergence statistique et la précision des résultats, entraînant des coûts de calcul élevés. Le travail réalisé s'étend donc également à l'évaluation de l'impact de différentes architectures de calcul sur les performances de notre algorithme.

Nous avons donc décidé de décliner notre objectif principal en trois volets complémentaires :

- **Implémentation séquentielle sur CPU** : développement initial d'une version simple en fonctionnement normal sur CPU, dans le but de valider le fonctionnement de l'algorithme. Il servira en point de comparaison de référence pour la suite.
- **Parallélisation sur CPU** : utilisation d'OpenMP afin d'exploiter le parallélisme multi-cœurs du processeur et de réduire, en toute logique, le temps d'exécution des simulations.
- **Implémentation accélérée sur GPU** : recours à la librairie CUDA pour déporter les calculs les plus intensifs, notamment la simulation des trajectoires et certaines opérations de régression, vers une architecture presque complètement parallèle, dans le but de gagner davantage de performances.

2 Cadre théorique

2.1 Options américaines et problématique de l'exercice anticipé

Une option américaine confère à son détenteur le droit, mais non l'obligation, d'acheter ou de vendre un actif sous-jacent à un prix fixé à l'avance, appelé *strike*, à tout instant compris entre la date d'émission et la date de maturité. Cette caractéristique la distingue des options européennes, pour lesquelles l'exercice n'est autorisé qu'à maturité.

Caractéristique	Options Américaines	Options Européennes
Moment d'exercice	À tout moment avant maturité.	Seulement à maturité.
Risque d'exercice anticipé	Oui, particulièrement en présence de dividendes.	Aucun exercice anticipé.
Modèle de pricing	Prime possible liée à la flexibilité d'exercice.	Suit typiquement le modèle Black-Scholes.
Parité Put-Call	Peut ne pas tenir (dû à l'exercice anticipé).	Tient strictement (marché efficient).
Sous-jacents courants	Actions US, ETFs, indices.	Indices européens.

Table 1: Comparaison entre Options Américaines et Européennes.

La possibilité d'un exercice anticipé introduit une complexité dans le processus de valorisation. En effet, à chaque date d'observation, le détenteur de l'option doit comparer le gain associé à un exercice immédiat avec celui attendu en conservant l'option. Cette décision repose sur l'identification d'une stratégie d'exercice optimal, dépendant à la fois du temps restant jusqu'à maturité et de l'évolution future du prix du sous-jacent.

La valorisation des options américaines s'apparente ainsi à un problème de décision dynamique, pour lequel il est généralement impossible d'obtenir une solution analytique fermée. Cette difficulté motive le recours à des méthodes numériques capables de traiter explicitement l'exercice anticipé.

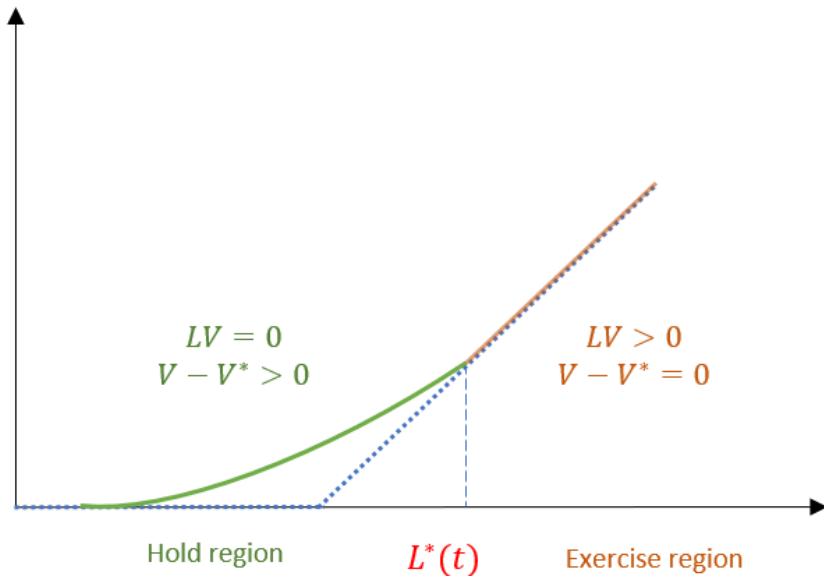


Figure 1: La figure illustre la règle de décision d'une option américaine : à chaque instant, la valeur de continuation est comparée au payoff immédiat. La frontière $L^*(t)$ délimite la région dans laquelle il est optimal d'exercer l'option de celle où il est préférable de la conserver.

Légende : L'axe des abscisses représente le prix du sous-jacent S_t , l'ordonnée la valeur de l'option. $L^*(t)$ est la frontière d'exercice optimal. V est la valeur de l'option, V^* sa valeur intrinsèque. LV est l'opérateur de Black-Scholes.

2.2 Modélisation stochastique du sous-jacent

La valorisation des options américaines par des méthodes de Monte Carlo nécessite un modèle probabiliste décrivant l'évolution du prix du sous-jacent au cours du temps. Dans ce projet, on adopte un cadre classique de finance quantitative fondé sur un modèle stochastique continu¹, permettant de simuler un grand nombre de trajectoires réalistes du prix de l'actif.

2.2.1 Mouvement brownien géométrique

Un *mouvement brownien* (ou processus de Wiener) est un processus stochastique continu $(W_t)_{t \geq 0}$ modélisant une source d'aléa pure, sans mémoire, évoluant dans le temps. Il peut être interprété comme la limite d'une marche aléatoire lorsque le pas de temps tend vers zéro.

Un mouvement brownien standard est caractérisé par les propriétés suivantes :

- **Condition initiale** : $W_0 = 0$.
- **Continuité des trajectoires** : Les trajectoires $t \mapsto W_t$ sont continues presque sûrement.
- **Incréments indépendants** : Pour tout entier $n \geq 1$ et pour tout vecteur de temps (t_0, \dots, t_n) tel que $0 \leq t_0 < t_1 < \dots < t_n$, les variables aléatoires $W_{t_1} - W_{t_0}, \dots, W_{t_n} - W_{t_{n-1}}$ sont indépendantes.
- **Incréments stationnaires** : Pour tout $t \geq 0$ et tout $h > 0$, la variable aléatoire $W_{t+h} - W_t$ a une loi qui dépend uniquement de h .
- **Incréments gaussiens centrés** : $W_{t+h} - W_t \sim \mathcal{N}(0, h)$.

Les trajectoires d'un mouvement brownien sont continues mais presque sûrement nulle part dérивables, traduisant une évolution extrêmement irrégulière. Ces propriétés impliquent notamment une absence de mémoire du processus ainsi qu'une croissance de l'incertitude proportionnelle au temps.

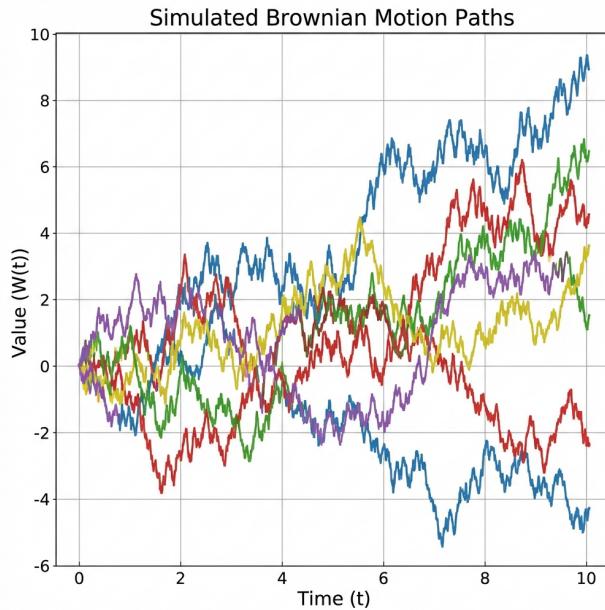


Figure 2: Réalisations de mouvements browniens standards. Les trajectoires sont continues mais presque sûrement nullepart dérivables, ce qui reflète l'absence de régularité locale et la nature purement aléatoire du processus.

¹Défini formellement par un espace probabilisé filtré $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \geq 0}, \mathbb{P})$ satisfaisant les conditions habituelles, où l'évolution des prix est modélisée par des processus d'Itô.

Pour modéliser l'évolution d'un prix d'actif financier, l'utilisation d'un tel mouvement brownien n'est cependant pas appropriée, notamment parce qu'elle autoriserait des valeurs négatives du prix. On préfère donc une dynamique multiplicative plutôt qu'additive, dans laquelle l'aléa porte sur les rendements plutôt que sur les variations absolues de prix. Cette approche conduit naturellement au *mouvement brownien géométrique*, qui constitue le modèle de référence pour la dynamique du prix d'un sous-jacent S_t .

Le mouvement brownien géométrique est défini comme la solution de l'équation différentielle stochastique

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

où μ est le drift¹ et σ la volatilité. Cette équation est une équation différentielle stochastique linéaire au sens d'Itô², issue de l'équation de Black-Scholes³. Sous ce modèle, le prix S_t suit une loi log-normale⁴ et admet l'expression explicite (voir le détail de la résolution en **Annexe A**) :

$$S_t = S_0 \exp \left(\left(\mu - \frac{1}{2} \sigma^2 \right) t + \sigma W_t \right).$$

Dans le cadre de ce travail ($q = 0$, mesure risque-neutre⁵), la dynamique se réduit à :

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

et la solution explicite est :

$$S_t = S_0 \exp \left(\left(r - \frac{1}{2} \sigma^2 \right) t + \sigma W_t^{\mathbb{Q}} \right).$$

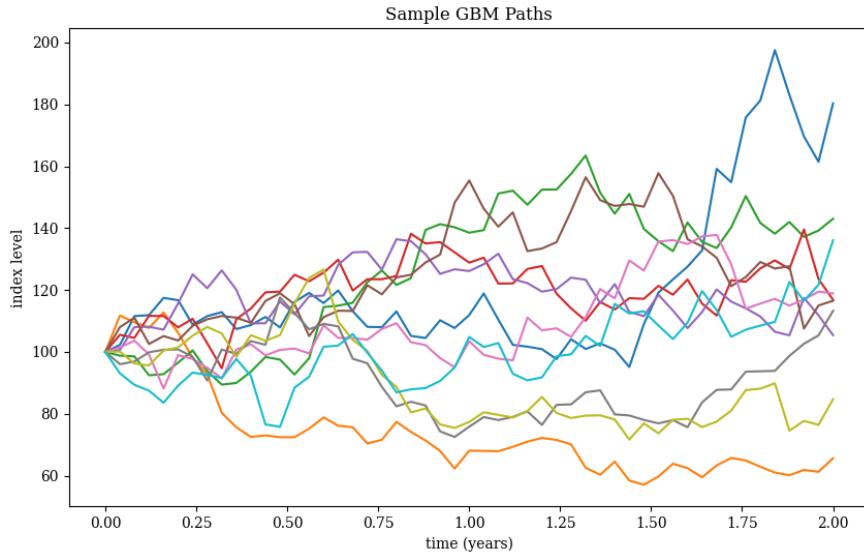


Figure 3: Exemples de trajectoires de Mouvement Brownien Géométrique.

2.2.2 Discrétisation temporelle

La simulation numérique repose sur une discrétisation exacte de cette solution :

$$S_{t_{n+1}} = S_{t_n} \exp \left(\left(r - \frac{\sigma^2}{2} \right) \Delta t_n + \sigma \sqrt{\Delta t_n} Z_n \right), \quad Z_n \sim \mathcal{N}(0, 1).$$

¹Tendance déterministe moyenne du cours, représentant le taux de rendement espéré de l'actif.

²Une équation différentielle stochastique (EDS) est dite linéaire au sens d'Itô si les coefficients de dérive et de diffusion sont des fonctions affines de la variable d'état S_t .

³Voir [2] pour une dérivation complète de l'équation aux dérivées partielles de Black-Scholes.

⁴Une variable aléatoire suit une loi log-normale si son logarithme népérien suit une loi normale. Ceci implique que le prix S_t reste strictement positif.

⁵Mesure de probabilité \mathbb{Q} sous laquelle le prix actualisé de tout actif non versant de dividendes est une martingale. Elle est fondamentale pour la valorisation d'actifs contingents (options).

Cette discréétisation correspond à l'utilisation de la solution exacte du mouvement brownien géométrique et ne souffre donc pas d'erreur de schéma temporel.

2.3 Algorithme de Longstaff–Schwartz (LSMC)

L'algorithme de Longstaff–Schwartz, également appelé *Least Squares Monte Carlo (LSMC)*, est une méthode de Monte Carlo spécifiquement conçue pour la valorisation des options américaines. Il permet d'estimer la valeur de continuation à chaque date d'exercice possible à partir des trajectoires simulées.

2.3.1 Valeur de continuation et problème d'arrêt optimal

On note V_{t_n} la valeur de l'option à la date t_n et $\Phi(S_{t_n})$ sa valeur d'exercice immédiat. Le prix de l'option américaine correspond au problème d'arrêt optimal :

$$V_{t_n} = \max(\Phi(S_{t_n}), C(t_n, S_{t_n})),$$

où la valeur de continuation $C(t_n, S_{t_n})$ est définie par l'espérance conditionnelle :

$$C(t_n, S_{t_n}) = \mathbb{E}^{\mathbb{Q}} [e^{-r\Delta t_n} V_{t_{n+1}} | S_{t_n}].$$

2.3.2 Régression par moindres carrés

Dans l'algorithme LSMC, la valeur de continuation est approximée par régression sur une base de fonctions $\{\psi_k(S_{t_n})\}$:

$$\hat{C}(t_n, S_{t_n}) \approx \sum_{k=0}^{K_{reg}} a_k \psi_k(S_{t_n}).$$

Le choix de cette base est fondamental pour la qualité de l'approximation. Dans le cadre de ce projet, nous avons implémenté et comparé plusieurs familles de polynômes orthogonaux classiques :

- **Base Canonique (Monomiale)** : Constituée des termes simples $1, X, X^2, \dots$. Bien que naturelle, elle engendre des matrices mal conditionnées pour des degrés élevés (> 3), ce qui peut dégrader la précision numérique.
- **Polynômes de Laguerre** : Orthogonaux sur \mathbb{R}^+ avec le poids e^{-x} , ils sont théoriquement adaptés aux variables positives comme les prix d'actifs S_t .
- **Polynômes d'Hermite** : Orthogonaux sur \mathbb{R} pour la mesure gaussienne. Ils sont pertinents dans le modèle Black-Scholes où le log-prix est gaussien.
- **Polynômes de Legendre** : Orthogonaux sur un segment fini, ils offrent une bonne stabilité mais nécessitent un redimensionnement des données.
- **Polynômes de Chebyshev** : Reconnus pour minimiser l'erreur d'interpolation (phénomène de Runge), ils contribuent à stabiliser la régression.

Les définitions mathématiques précises et les relations de récurrence de ces bases sont détaillées en **Annexe C**.

Nous avons analysé l'influence de la nature et du degré (2, 3, voire plus) de ces bases sur la convergence du prix et la stabilité du calcul.

2.4 Synthèse algorithmique et pseudo-algorithme du LSMC

L'algorithme repose sur une discréétisation du temps et sur la simulation Monte Carlo d'un grand nombre de trajectoires. Les décisions d'exercice sont déterminées par *induction arrière*.

Le pseudo-code suivant résume de manière synthétique les différentes étapes de la méthode.

Algorithm 1 Pseudo-code de l'algorithme de Longstaff–Schwartz (LSMC)

```
1: Simuler  $N_{\text{paths}}$  trajectoires  $(S_{t_n}^{(i)})_n$ .  
2: Initialiser  $V_{t_N}^{(i)} = \Phi(S_{t_N}^{(i)})$ .  
3: for  $n = N - 1$  down to 1 do  
4:    $Y^{(i)} = e^{-r\Delta t_n} V_{t_{n+1}}^{(i)}$ .  
5:   Régression  $Y^{(i)} \approx \widehat{C}(t_n, S_{t_n}^{(i)})$ .  
6:   for chaque trajectoire  $i$  do  
7:     if  $\Phi(S^{(i)}) \geq \widehat{C}(S^{(i)})$  then  
8:        $V^{(i)} \leftarrow \Phi(S^{(i)})$ .  
9:     else  
10:       $V^{(i)} \leftarrow Y^{(i)}$ .  
11:    end if  
12:   end for  
13: end for  
14:  $V_0 = \text{mean}(e^{-rt_1} V_{t_1}^{(i)})$ .
```

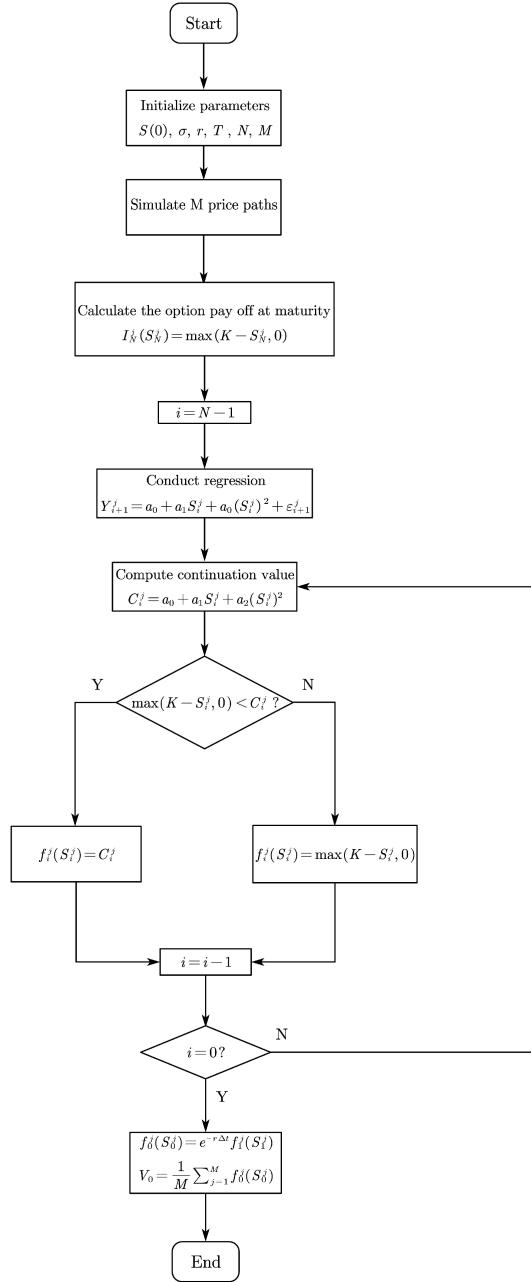


Figure 4: Illustration schématique de la méthode LSMC : Régression sur les trajectoires In-The-Money.

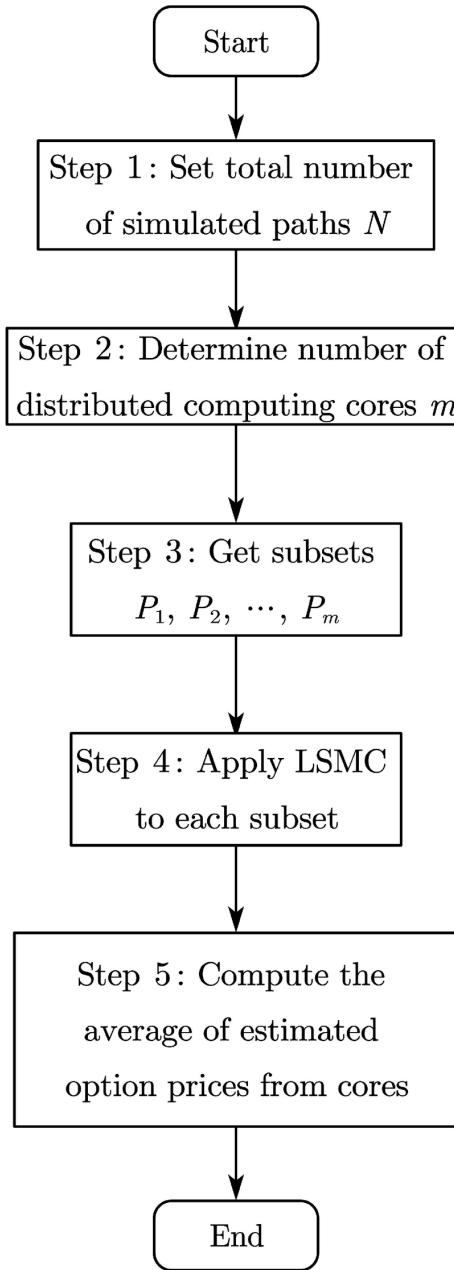


Figure 5: Flux de l'algorithme LSMC : Simulation parallèle vs Régression séquentielle.

Synthèse : La structure du LSMC est intrinsèquement hybride. Si la phase de simulation est un candidat idéal pour le parallélisme massif (qualifié d'*embarrassingly parallel*¹), la phase de backward induction impose une synchronisation globale à chaque pas de temps, limitant le potentiel d'accélération sur GPU (loi d'Amdahl).

¹ Terme technique standard dans la littérature du calcul haute performance (et la documentation CUDA) désignant des tâches pouvant être exécutées de manière totalement indépendante, sans communication entre elles.

3 Travail réalisé : implémentation et méthodologie

Cette section décrit les choix techniques et méthodologiques retenus pour l'implémentation de l'algorithme de Longstaff–Schwartz, ainsi que les stratégies de parallélisation mises en œuvre sur CPU et GPU. L'objectif est double : assurer la validité numérique des résultats tout en évaluant l'impact des architectures de calcul sur les performances.

3.1 Environnement de développement

L'ensemble du projet a été développé en C++, avec une attention particulière portée aux performances et à la gestion mémoire. Les principaux outils et technologies utilisés sont :

- compilateur g++ compatible C++17;
- bibliothèque OpenMP pour la parallélisation CPU;
- CUDA C++ pour l'implémentation GPU;
- générateurs pseudo-aléatoires indépendants par thread;
- système de build basé sur CMake.

Les calculs ont été réalisés sur une machine équipée d'un processeur multi-cœurs et d'un GPU NVIDIA compatible CUDA. Les expériences ont été conduites en privilégiant la reproductibilité et la stabilité numérique.

3.2 Implémentation CPU séquentielle

Une première version séquentielle de l'algorithme LSMC a été implémentée afin de servir de référence fonctionnelle et de point de comparaison en termes de performance.

Cette version suit strictement les étapes théoriques :

- simulation des trajectoires du sous-jacent selon un mouvement brownien géométrique;
- calcul des payoffs à chaque date;
- application de la backward induction;
- estimation de la valeur de continuation par régression polynomiale;
- calcul de la moyenne finale des cashflows actualisés.

L'implémentation séquentielle permet de valider la cohérence des résultats numériques et de mesurer le coût de calcul intrinsèque de l'algorithme avant toute parallélisation.

3.3 Parallélisation CPU avec OpenMP

La version parallèle CPU repose sur l'observation que la majorité des calculs dans l'algorithme LSMC sont indépendants par trajectoire. Cette propriété est exploitée à l'aide d'OpenMP.

Les sections parallélisées sont notamment :

- la génération des trajectoires du sous-jacent;
- le calcul des payoffs;
- l'accumulation des termes des équations normales pour la régression;

- la mise à jour des cashflows lors de la backward induction;
- le calcul de la moyenne finale.

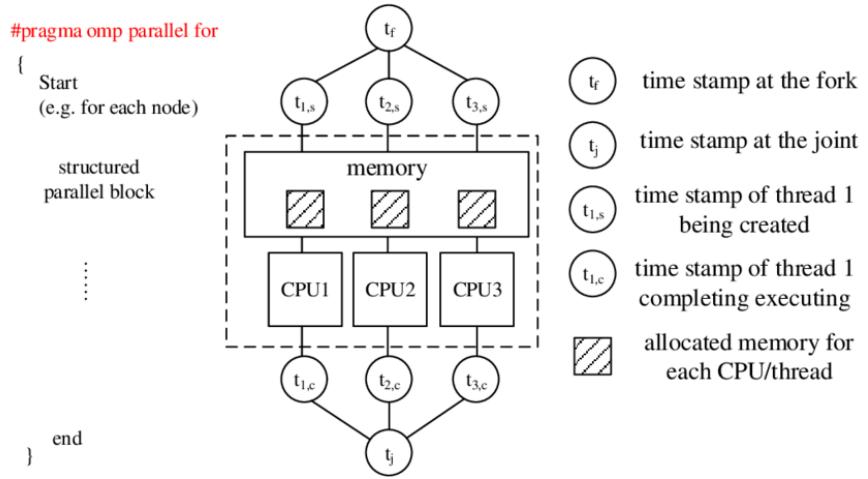


Figure 6: Illustration du modèle d'exécution Fork-Join utilisé par OpenMP (Source : Fork-Join model for OpenMP parallelisation).

Les réductions OpenMP sont utilisées pour agréger efficacement les contributions des différentes trajectoires, tout en garantissant l'absence de conditions de course¹. Le schéma `schedule(static)` est privilégié afin d'assurer une répartition homogène de la charge de travail et un bon comportement mémoire.

Cette parallélisation permet d'exploiter efficacement les coeurs du processeur, mais reste limitée par la bande passante mémoire et la nature statistique de l'algorithme.

3.4 Accélération GPU avec CUDA

Une version accélérée sur GPU a été développée afin d'exploiter le parallélisme massif offert par les architectures CUDA, une approche explorée dans des travaux similaires [4, 5]. Le GPU est particulièrement adapté à la simulation Monte Carlo, chaque trajectoire pouvant être associée à un thread indépendant.

Les principales étapes déportées sur le GPU sont :

- la simulation des trajectoires du mouvement brownien géométrique;
- le calcul des payoffs;
- certaines phases de réduction nécessaires à la régression.

La backward induction impose cependant une dépendance temporelle forte entre les dates successives, ce qui limite la parallélisation complète de l'algorithme. L'approche retenue consiste donc à paralléliser intensivement les calculs spatiaux (trajectoires) tout en conservant une synchronisation globale entre les dates.

Une attention particulière est portée à l'organisation mémoire des données afin de garantir des accès coalescents² et de limiter les transferts entre l'hôte (CPU) et le périphérique (GPU).

¹ Une *race condition* (condition de course) survient lorsque le résultat d'un programme dépend de l'ordre d'exécution imprévisible de threads accédant simultanément à des données partagées en écriture.

² L'accès coalescent désigne le regroupement par le matériel de plusieurs requêtes mémoire provenant de threads voisins en une seule transaction physique, optimisant ainsi l'utilisation de la bande passante.

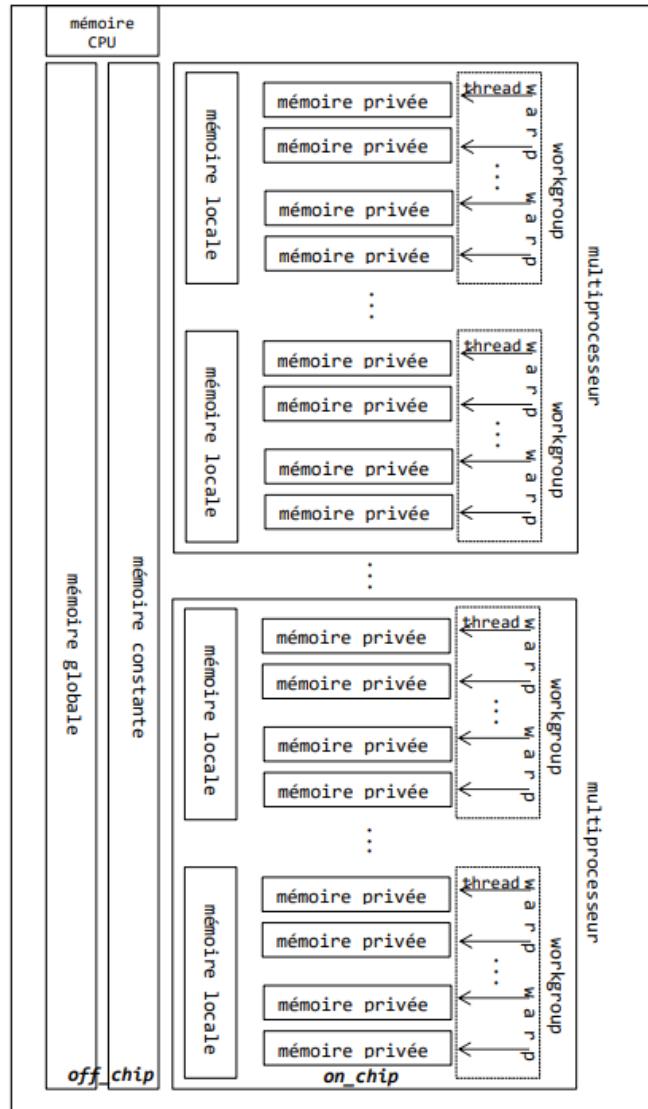


Figure 3 Découpage logique de la mémoire et des threads d'un GPU, sans scindement des warp.

Figure 7: Découpage logique de la mémoire et des threads d'un GPU (Architecture CUDA) [7].



Figure 8: Architecture d'un Streaming Multiprocessor (SM) Ada Lovelace. Chaque SM contient 4 partitions avec des unités FP32/INT32, des Tensor Cores de 4ème génération, 128 Ko de cache L1/mémoire partagée, et un RT Core de 3ème génération.

Un paramètre clé de l'optimisation CUDA est la taille des blocs de threads. En CUDA, les threads sont organisés en blocs, et plusieurs blocs sont exécutés sur un même SM. La taille de bloc (nombre de threads par bloc) peut être ajustée par multiples de 32 (taille d'un warp) jusqu'à un maximum de 1024 threads. Ce paramètre influence directement l'occupation des SMs et donc les performances globales. L'impact de ce paramètre sera analysé dans la Section 4.7.

Cette implémentation permet d'évaluer concrètement les gains de performance apportés par le GPU et de mettre en évidence les limites structurelles du LSMC dans un contexte massivement parallèle.

3.5 Méthodes de Différences Finies (FDM) pour comparaison

Afin de valider nos résultats Monte Carlo et de disposer d'un point de comparaison déterministe performant, nous avons également implémenté des solveurs basés sur les différences finies (FDM) pour l'équation de Black-Scholes-Merton (PDE) [2]. Bien que ces méthodes soient limitées en dimension (difficiles à étendre au-delà de 2 ou 3 sous-jacents), elles sont extrêmement efficaces pour les options vanilles¹ et américaines sur un seul sous-jacent.

Trois schémas numériques ont été implémentés dans le fichier `fdm.cpp` :

- Le schéma de **Runge-Kutta 4 (RK4)** a été choisi comme méthode de référence ("baseline") pour sa haute précision ($O(\Delta t^4)$). Bien que plus coûteux en calcul que les méthodes d'Euler, il fournit une valeur quasi-exacte pour valider les résultats Monte Carlo.

¹ Les options vanilles désignent les options standard (Call et Put) ayant des caractéristiques classiques (prix d'exercice, date d'échéance) et ne présentant pas de clauses exotiques (barrières, asiatiques, lookback, etc.).

- **Euler Implicit** : Schéma inconditionnellement stable, nécessitant la résolution d'un système linéaire tridiagonal à chaque pas de temps (algorithme de Thomas¹).
- **Euler Explicit** : Schéma simple et rapide, mais conditionnellement stable (nécessite un pas de temps suffisamment petit pour éviter les instabilités numériques).

Ces méthodes nous ont servi de "vérité terrain" pour vérifier la convergence de nos prix LSMC.

3.6 Structure du code et organisation des fichiers

Le projet est organisé de manière modulaire pour séparer les responsabilités (modélisation, calcul, utilitaires). Voici une description de l'arborescence :

- **src/** (dossier racine P1RV_CUDA/) :
 - **lsmc.cu** / **lsmc.cpp** : Cœur de l'algorithme Longstaff-Schwartz. La version .cu contient les kernels CUDA pour le GPU.
 - **gbm.cu** : Simulation du Mouvement Brownien Géométrique.
 - **fdm.cpp** : Implémentation des méthodes de différences finies (solvers PDE).
 - **main.cu** : Point d'entrée, gestion des arguments et lancement des benchmarks.
- **Utils/** :
 - **csv_writer.hpp** : Utilitaires pour l'export des résultats.
 - Scripts Python : Pour l'interface utilisateur et la visualisation.

¹ L'algorithme de Thomas, également appelé TDMA (TriDiagonal Matrix Algorithm), est une forme simplifiée de l'élimination de Gauss optimisée pour les systèmes tridiagonaux. Sa complexité est $(n) \text{contre}(n^3)$ pour l'élimination de Gauss générale.

4 Résultats et analyse des performances

4.1 Configuration matérielle

Tous les benchmarks présentés dans cette section ont été réalisés sur une machine équipée d'un processeur **Intel® Core™ i7-13620H** (13ème génération) et d'une carte graphique **NVIDIA GeForce RTX 4060**. Cette dernière, basée sur l'architecture Ada Lovelace, dispose de **3072 coeurs CUDA** et de **8 Go de mémoire GDDR6**, offrant une puissance de calcul théorique adaptée aux simulations massivement parallèles.

Propriété	Valeur
Architecture	Ada Lovelace
Compute Capability	8.9
Mémoire totale	8 Go GDDR6
Driver CUDA	581.80
Max threads par bloc	1024
Max threads par SM	1536
Taille de warp	32
Max warps par SM	48
Max blocs par SM	16

Table 2: Spécifications techniques du GPU RTX 4060 utilisé pour les benchmarks.

Cette section présente les résultats obtenus lors de l'exécution de l'algorithme de Longstaff–Schwartz selon les différentes architectures étudiées : CPU séquentiel, CPU parallélisé avec OpenMP et GPU via CUDA. L'analyse porte à la fois sur la validité numérique des résultats et sur les performances de calcul observées.

4.2 Première approche : comparaison des méthodes de parallélisme

Les performances ont été évaluées sur un ensemble de simulations Monte Carlo pour une option de type put américain. Nous avons comparé les temps d'exécution et la précision des prix obtenus par nos trois implémentations (CPU Séquentiel, OpenMP, GPU CUDA) ainsi que par les méthodes de différences finies.

Les résultats ci-dessous ont été obtenus sur une machine équipée d'un GPU NVIDIA.

Mode	Pas (N)	Trajectoires (M)	Prix (€)	Temps (ms)	Écart / FDM
FDM Implicite	1000	-	6.067	0.67	Ref
FDM Explicite	1000	-	6.079	60.18	+0.012
FDM RK4	1000	-	6.079	231.88	+0.012
CPU Séquentiel	50	100,000	6.057	559.91	-0.010
OpenMP	50	100,000	6.057	540.23	-0.010
GPU CUDA	50	100,000	6.070	41.96	+0.003
CPU Séquentiel	50	1,000,000	6.059	6914.19	-0.008
OpenMP	50	1,000,000	6.059	6562.99	-0.008
GPU CUDA	50	1,000,000	6.047	455.63	-0.020
CPU Séquentiel	50	5,000,000	6.057	35352.25	-0.010

Table 3: Comparaison des temps de calcul et précision pour un Put Américain ($S_0 = 100, K = 100, r = 0.05, \sigma = 0.2, T = 1$). (Matériel : i7-13620H + RTX 4060)

Analyse des résultats :

- **Précision** : Tous les modes LSMC convergent vers un prix très proche de la référence FDM (6.067). Les écarts observés sont de l'ordre du centime, ce qui est acceptable pour une méthode de Monte Carlo avec ces paramètres.

- **Performance GPU** : Le GPU démontre une accélération spectaculaire. Pour 1 million de trajectoires, le calcul prend environ **455 ms** sur GPU contre près de **7 secondes** (6914 ms) sur CPU séquentiel, soit un facteur d'accélération (speedup) d'environ $\times 15$.
- **Comparaison FDM** : Les méthodes de différences finies (surtout l'implicite) sont extrêmement rapides ($< 1\text{ms}$) pour ce problème 1D. Cela confirme que pour des options simples, les PDE restent supérieures. Cependant, l'intérêt du LSMC (et donc de notre implémentation GPU) réside dans sa capacité à traiter des problèmes de plus haute dimension où les méthodes de grille échouent.

4.3 Pour aller plus loin : analyse de l'impact de la base de régression

Les benchmarks réalisés ("boosted", voir tableau 4) révèlent que l'augmentation du degré de la base est bénéfique. La base Cubique permet d'atteindre un prix de ≈ 6.06 , nettement plus proche de la référence théorique (≈ 6.08) que les bases quadratiques/monomiales (≈ 5.58 dans cette configuration CPU non-optimisée).

Cette amélioration de précision justifie le léger surcoût calculatoire lié à la résolution de systèmes linéaires 4×4 , désormais gérée par notre solveur de Gauss générique sur GPU.

Base	Architecture	Trajectoires	Prix (€)	Temps (ms)	Throughput (ops/s)
N = 100,000					
Monomiale	CPU Séquentiel	100k	5.586	261.96	19.1 M
Monomiale	CPU OpenMP	100k	5.586	274.97	18.2 M
Monomiale	GPU	100k	6.070	45.46	109.9 M
Hermite	CPU Séquentiel	100k	5.586	265.94	18.8 M
Hermite	CPU OpenMP	100k	5.586	266.81	18.7 M
Hermite	GPU	100k	6.070	43.40	115.2 M
Laguerre	CPU Séquentiel	100k	5.586	268.81	18.6 M
Laguerre	CPU OpenMP	100k	5.586	265.83	18.8 M
Laguerre	GPU	100k	6.070	35.44	141.1 M
Cheby shev	CPU Séquentiel	100k	5.586	263.63	18.9 M
Cheby shev	CPU OpenMP	100k	5.586	265.64	18.8 M
Cheby shev	GPU	100k	6.070	41.66	120.0 M
Cubique	CPU Séquentiel	100k	5.586	266.56	18.7 M
Cubique	CPU OpenMP	100k	5.586	265.21	18.8 M
Cubique	GPU	100k	6.086	38.66	129.3 M
N = 1,000,000					
Monomiale	CPU Séquentiel	1M	5.565	2653.88	18.8 M
Monomiale	CPU OpenMP	1M	5.565	2688.26	18.6 M
Monomiale	GPU	1M	6.047	334.53	149.5 M
Hermite	CPU Séquentiel	1M	5.565	2672.32	18.7 M
Hermite	CPU OpenMP	1M	5.565	2704.67	18.5 M
Hermite	GPU	1M	6.047	594.25	84.1 M
Laguerre	CPU Séquentiel	1M	5.565	2710.66	18.4 M
Laguerre	CPU OpenMP	1M	5.565	2841.14	17.6 M
Laguerre	GPU	1M	6.047	649.80	76.9 M
Cheby shev	CPU Séquentiel	1M	5.565	2633.49	18.9 M
Cheby shev	CPU OpenMP	1M	5.565	2638.32	18.9 M
Cheby shev	GPU	1M	6.047	626.32	79.8 M
Cubique	CPU Séquentiel	1M	5.565	2677.14	18.7 M
Cubique	CPU OpenMP	1M	5.565	2628.52	19.0 M
Cubique	GPU	1M	6.063	687.69	72.7 M

Table 4: Benchmarks "Boosted" complets : Impact du choix de la base et de l'architecture. (Matériel : i7-13620H + RTX 4060)

Les temps mesurés confirment l'efficacité de l'approche massivement parallèle pour la phase de simulation. Le goulot d'étranglement restant sur GPU est la régression séquentielle à chaque pas de temps.

Une expérimentation complémentaire a été menée pour tenter d'éliminer totalement les synchronisations CPU durant la phase de régression (implémentation **Parallélisation Complète de la Régression sur GPU** ou "Full GPU"). Le tableau 5 présente les résultats obtenus.

Base	Architecture	Traj.	Prix (€)	Temps (ms)	Ops/s	Speedup vs GPU Std
N = 100 000						
Monomiale	GPU Full	100k	6.085	51.55	1.9 M	0.88×
Hermite	GPU Full	100k	6.085	86.05	1.2 M	0.50×
Laguerre	GPU Full	100k	6.085	93.10	1.1 M	0.38×
Chebyshev	GPU Full	100k	3.812	102.95	1.0 M	0.40×

Table 5: Comparaison de performance : Nouveau Solveur GPU "Full" vs Standard (N=100k)

Les résultats présentés dans le tableau 5 montrent que l'approche dite *Full GPU*, visant à déporter l'intégralité de la phase de régression sur le GPU, conduit à des temps d'exécution significativement plus élevés que l'implémentation GPU standard. Ce comportement, a priori contre-intuitif, s'explique par plusieurs facteurs structurels liés à la nature du calcul et à l'architecture GPU.

Tout d'abord, la phase de régression repose sur des opérations de réduction globale (construction des matrices normales, calcul des produits scalaires, résolution de systèmes linéaires de petite taille) qui présentent un parallélisme intrinsèquement limité. Pour des tailles de matrices faibles (typiquement 3×3 ou 4×4), le coût de lancement des kernels, les synchronisations entre blocs et la gestion de la mémoire partagée deviennent prépondérants face au coût arithmétique réel, ce qui dégrade fortement les performances.

De plus, la suppression des normalisations dynamiques (Min/Max) dans certaines bases, bien qu'elle permette d'éliminer des synchronisations coûteuses, peut engendrer des problèmes de conditionnement numérique et de stabilité, limitant les gains théoriques attendus. Le GPU se révèle ainsi particulièrement efficace pour des calculs massivement parallèles et compute-bound, mais moins adapté à des opérations séquentielles fines et faiblement dimensionnées comme la résolution répétée de petits systèmes linéaires.

Enfin, cette implémentation *Full GPU* n'a pas fait l'objet d'optimisations approfondies (kernel fusion, utilisation systématique de bibliothèques spécialisées telles que cuBLAS ou CUB, restructuration mémoire avancée). Compte tenu des contraintes de temps du projet, il n'a pas été possible d'explorer plus en détail ces pistes d'optimisation, l'objectif principal restant l'analyse comparative des architectures CPU, OpenMP et GPU standard dans le cadre du LSMC.

Ces résultats restent néanmoins instructifs, car ils mettent en évidence les limites pratiques d'une approche naïvement « tout GPU » et soulignent l'intérêt d'architectures hybrides, dans lesquelles le GPU est réservé aux phases massivement parallèles tandis que certaines opérations restent plus efficacement traitées sur CPU.

4.4 Validation numérique

Avant toute analyse de performance, la cohérence numérique des différentes implémentations a été vérifiée. Les prix obtenus avec :

- l'implémentation CPU séquentielle,
- l'implémentation CPU OpenMP,
- l'implémentation GPU CUDA,

sont compatibles entre eux à l'intérieur des intervalles d'erreur statistique attendus pour une méthode de Monte Carlo.

Lorsque le nombre de trajectoires augmente, la variance de l'estimateur décroît conformément au taux théorique $\mathcal{O}(1/\sqrt{N_{\text{paths}}})$, ce qui confirme la bonne implémentation de l'algorithme LSMC sur les trois architectures.

4.5 Performances CPU séquentiel

L'implémentation séquentielle sert de référence de performance. Le temps d'exécution croît linéairement avec le nombre de trajectoires simulées, ce qui est cohérent avec la complexité algorithmique du LSMC :

$$\mathcal{O}(N_{\text{paths}} \times N_{\text{steps}}).$$

Cette version met en évidence le caractère fortement coûteux des méthodes de Monte Carlo lorsque la précision recherchée impose un grand nombre de trajectoires. Elle justifie pleinement le recours à des techniques de parallélisation.

Les figures ci-dessous illustrent la validation empirique de la complexité linéaire de l'implémentation séquentielle.

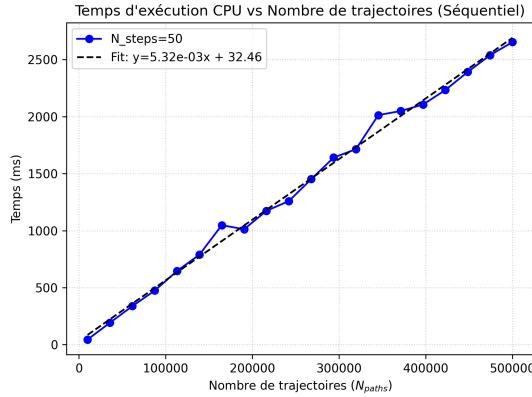


Figure 9: Temps vs N_{paths} (CPU Séquentiel)

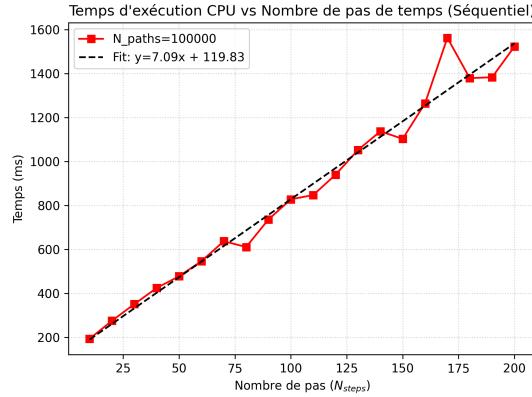


Figure 10: Temps vs N_{steps} (CPU Séquentiel)

L'alignement quasi-parfait des points mesurés avec la régression linéaire confirme que l'implémentation respecte la complexité théorique sans overhead significatif cachés.

4.6 Accélération par parallélisation CPU avec OpenMP

La version OpenMP exploite le parallélisme multi-coeurs du processeur. Les gains observés sont significatifs pour les phases suivantes :

- simulation des trajectoires du sous-jacent;
- calcul des payoffs;
- accumulation des équations normales pour la régression;
- mise à jour des cashflows.

Les résultats de benchmark avec 4 threads OpenMP confirment une amélioration des temps de calcul, tout en conservant une complexité linéaire globale (voir figures ci-dessous).

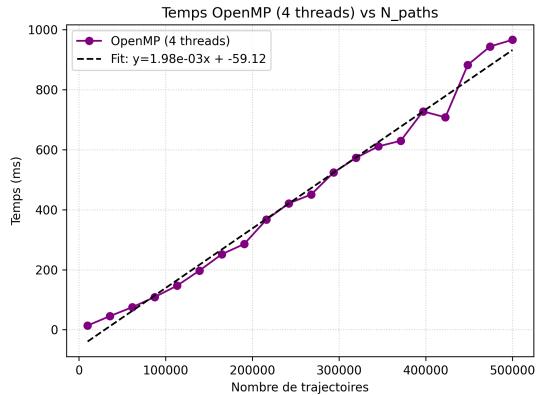


Figure 11: Temps vs N_{paths} (OpenMP 4 threads)

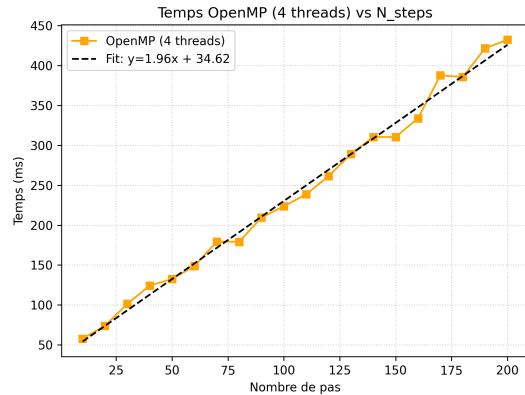


Figure 12: Temps vs N_{steps} (OpenMP 4 threads)

Le facteur d'accélération obtenu est inférieur au nombre de coeurs disponibles, ce qui s'explique par :

- la bande passante mémoire limitée;
- les accès concurrents aux structures de données partagées;
- la présence de phases séquentielles incompressibles (notamment la progression temporelle de la backward induction).

Néanmoins, OpenMP permet une réduction notable du temps de calcul, tout en conservant une implémentation relatively simple et portable.

Synthèse : Le parallélisme OpenMP est efficace pour les phases *compute-bound* (simulations) mais sature rapidement la bande passante mémoire lors des opérations vectorielles massives, plafonnant l'accélération bien en deçà du nombre de coeurs physiques.

Analyse du passage à l'échelle selon la taille du problème. La figure 17 illustre l'évolution du temps d'exécution en fonction du nombre de coeurs pour différentes tailles de problème. Pour de petits nombres de trajectoires ($N=1000$), le parallélisme n'apporte que peu de bénéfices car le temps de calcul est dominé par les synchronisations et le lancement des threads. En revanche, lorsque le problème grandit ($N=100\,000$ ou $N=1\,000\,000$), l'accélération devient très significative, bien qu'elle plafonne rapidement au-delà de 4-6 coeurs en raison de la saturation de la bande passante mémoire.

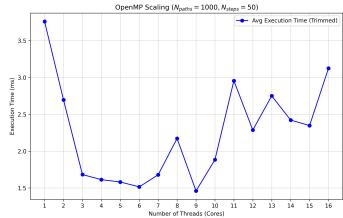


Figure 13: *
N=1 000

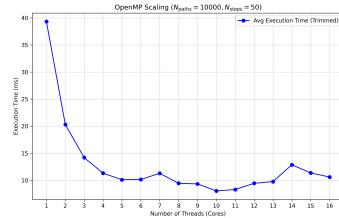


Figure 14: *
N=10 000

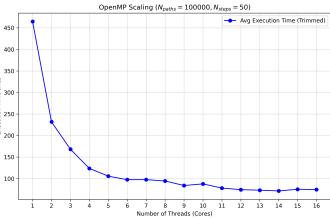


Figure 15: *
N=100 000

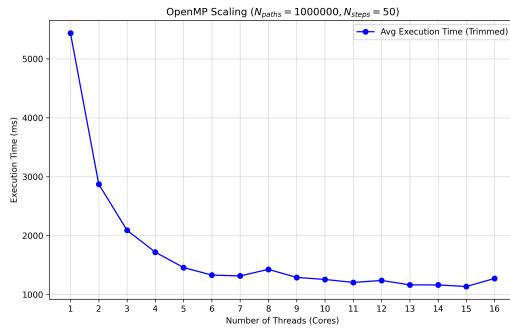


Figure 16: *
N=1 000 000

Figure 17: Passage à l'échelle OpenMP selon la taille du problème (moyenne lissée sur 10 exécutions, min/max exclus). L'accélération est négligeable pour les petits problèmes mais devient significative pour $N \geq 100\,000$, avant de plafonner à cause de la bande passante mémoire.

4.7 Accélération GPU avec CUDA

L'implémentation GPU avec CUDA a mis en évidence [7] un contraste marqué entre les différentes phases de l'algorithme. La simulation des trajectoires et le calcul des payoffs bénéficient pleinement du parallélisme massif, avec des accélérations importantes lorsque le nombre de trajectoires augmente.

Les résultats expérimentaux (voir Figures ci-dessous) montrent un gain de performance croissant avec la taille du problème, atteignant des facteurs d'accélération supérieurs à 30 pour de grands nombres de trajectoires.

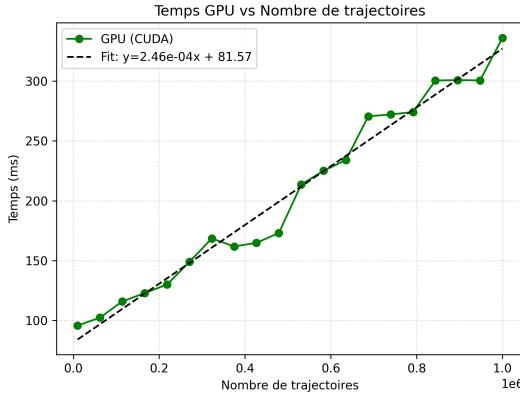


Figure 18: Temps vs N_{paths} (GPU CUDA)

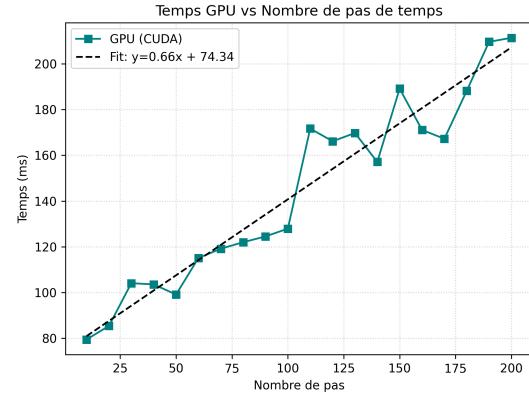


Figure 19: Temps vs N_{steps} (GPU CUDA)

En revanche, la backward induction impose une dépendance temporelle forte entre les dates successives. Cette contrainte limite la parallélisation complète de l'algorithme et réduit le gain théorique maximal. Les performances dépendent notamment :

- du nombre de trajectoires simulées;
- de l'occupation effective du GPU;
- de l'efficacité des réductions parallèles;
- de la limitation des transferts mémoire CPU-GPU.

Synthèse : Le GPU excelle sur le parallélisme de données (trajectoires), offrant des speedups d'un ordre de grandeur, mais l'accélération globale reste structurellement bornée par la nature séquentielle temporelle de l'algorithme (loi d'Amdahl appliquée à la backward induction).

Par ailleurs, une légère différence entre les prix CPU et GPU a été observée. Elle s'explique par la nature statistique de la méthode de Monte Carlo, les différences de générateurs pseudo-aléatoires et les effets d'arrondis en arithmétique flottante. Ces écarts restent toutefois compatibles avec la variance attendue de l'estimateur.

En ce qui concerne la stabilité des temps de calcul, on observe une variance plus élevée sur GPU que sur CPU. Ceci s'explique par plusieurs facteurs intrinsèques au fonctionnement d'un GPU dans un environnement non temps-réel :

- **Nature asynchrone :** Les lancements de kernels et les transferts mémoire impliquent une interaction complexe avec le driver, soumise à des latences variables.
- **Gestion dynamique des fréquences :** Le mécanisme de "boost clock" ajuste la fréquence du GPU en temps réel selon la charge et la température, introduisant des fluctuations de performance sur des exécutions courtes.
- **Ressource partagée :** Le GPU étant également sollicité par le système d'affichage (WDDM sous Windows), des micro-interruptions peuvent survenir, contrairement aux threads CPU qui peuvent être plus isolés.

Nous avons également évalué l'impact de la taille des blocs de threads sur les performances GPU :

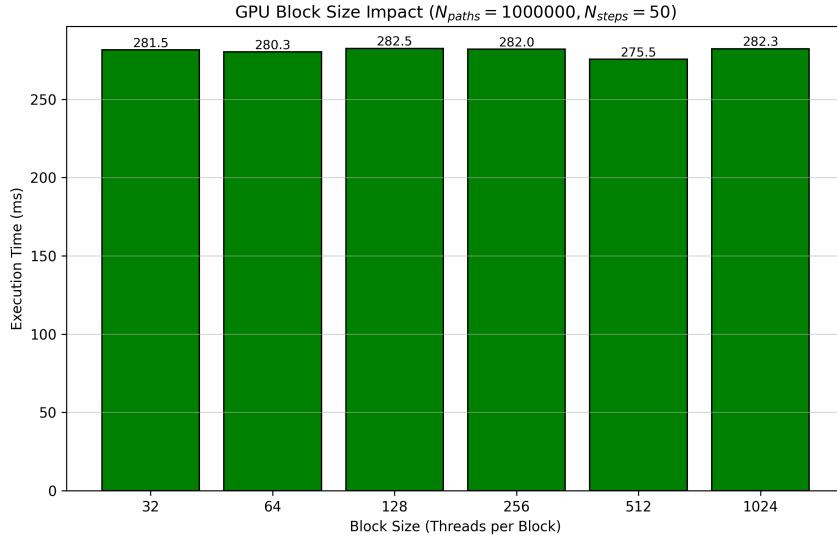


Figure 20: Impact de la taille des blocs GPU sur le temps d'exécution ($N=1\,000\,000$ trajectoires, 50 pas). Moyenne lissée sur 10 exécutions (min/max exclus).

Les résultats montrent une différence notable entre la plus petite taille (32 threads, soit un seul warp) et les tailles supérieures. Au-delà de 64 threads par bloc, les performances sont relativement stables, suggérant que la taille de 256 threads (valeur par défaut) représente un bon compromis.

4.8 Comparaison globale des architectures

De manière synthétique :

- le CPU séquentiel fournit une référence simple mais peu performante;
- OpenMP permet un gain modéré, limité par la bande passante mémoire;
- CUDA offre les meilleures performances pour les grandes tailles de problèmes, en particulier lorsque le nombre de trajectoires est très élevé.

Le GPU s'avère donc particulièrement adapté aux méthodes de Monte Carlo à grande échelle, tandis que le CPU reste pertinent pour des tailles de problèmes plus modérées ou lorsque la simplicité d'implémentation est prioritaire.

4.9 Discussion sur les limites du parallélisme

Les résultats confirment que l'algorithme LSMC est naturellement bien adapté au parallélisme spatial, mais intrinsèquement limité par sa structure séquentielle dans le temps. En effet, l'algorithme repose sur une **induction arrière** (backward induction) : le calcul des valeurs à l'étape t dépend mathématiquement des résultats de l'étape $t+1$ (nécessaires pour construire la variable cible de la régression).

Il est donc impossible de paralléliser le traitement des différents pas de temps pour une même option. Le GPU doit impérativement attendre la fin du calcul de l'étape $t+1$ avant de commencer l'étape t , ce qui crée une barrière de synchronisation inévitable. L'accélération maximale est ainsi plafonnée par cette contrainte séquentielle, même avec un nombre infini de coeurs. La seule manière d'accroître davantage le parallélisme serait de traiter simultanément plusieurs options distinctes (batching).

Ces observations expliquent pourquoi les gains GPU, bien que très importants, ne peuvent pas être parfaitement linéaires avec la puissance de calcul.

Ces résultats mettent en évidence l'intérêt d'architectures hybrides CPU-GPU, ainsi que l'importance de choix d'implémentation fins (organisation mémoire, réductions efficaces, limitation des synchronisations) pour exploiter pleinement les capacités du matériel moderne.

Synthèse Globale : Si le GPU déverrouille la scalabilité spatiale du stock de trajectoires, la dimension temporelle reste séquentielle. L'optimum de performance réside dans un équilibre : assez de trajectoires pour saturer le GPU, mais pas au-delà des besoins de précision, car le coût de régression devient alors prépondérant.

5 Analyse Critique et Limitations

La réalisation de ce projet a mis en évidence plusieurs défis techniques et méthodologiques, dont l'analyse permet de mieux cerner les contraintes du calcul haute performance en finance.

5.1 Organisation du projet et évolution des dépôts

Le développement du projet s'est articulé autour de **deux dépôts Git distincts**, reflétant une évolution technique majeure :

1. **lsmc-openmp**¹ (Octobre 2025) : Le premier dépôt a été consacré au développement de la logique métier de l'algorithme LSMC en C++ avec parallélisation OpenMP. Le système de build reposait sur des solutions **Visual Studio** (.sln, .vcxproj). Ce dépôt inclut également une interface graphique en Python (Streamlit/Matplotlib) pour la visualisation des trajectoires.
2. **CUDA-Implementation**² (Janvier 2026) : Face aux difficultés d'intégration CUDA dans le premier dépôt, un second dépôt a été créé avec une architecture repensée autour de **CMake**. C'est dans ce dépôt que les fonctionnalités avancées (5 bases de régression, solveur générique, kernels optimisés) ont été développées et validées.

Cette organisation permet de conserver une **version CPU de référence** (premier dépôt) tout en disposant d'une **version HPC/GPU complète** (second dépôt).

5.2 Échec de l'intégration CUDA dans Visual Studio

L'historique des commits du premier dépôt témoigne des tentatives infructueuses d'intégration CUDA :

- "*Début intégration CUDA*" : ajout initial des fichiers .cu et configuration NVCC.
- "*On continue de debug CUDA car rien ne marche*" : erreurs de compilation persistantes.
- "*Suppression complète de tout le code lié à CUDA/GPU*" : abandon de l'approche.

Cause identifiée : Visual Studio peinait à gérer correctement la cohabitation entre le compilateur C++ standard (MSVC) et le compilateur CUDA (nvcc). Les conflits de flags de compilation, les incompatibilités d'architectures cibles et la gestion des dépendances rendaient la configuration instable.

Solution adoptée : La migration vers **CMake** a résolu ces problèmes. CMake intègre nativement le support CUDA via `enable_language(CUDA)` et permet une séparation propre entre le code hôte (.cpp) et le code device (.cu). Cette architecture a permis de finaliser l'implémentation GPU avec succès.

5.3 Complexité de l'exercice anticipé

La difficulté théorique majeure provient de la possibilité d'un exercice anticipé pour les options américaines. Cette caractéristique transforme le problème de valorisation en un problème d'arrêt optimal, nécessitant une estimation fiable de la valeur de continuation à chaque date. Une mauvaise compréhension de cette quantité conduit rapidement à des erreurs de logique dans la backward induction. Un soin particulier a donc été apporté à la séparation claire entre valeur d'exercice immédiat et valeur de continuation estimée.

¹<https://github.com/FlorianBarbe/lsmc-openmp>

²<https://github.com/FlorianBarbe/CUDA-Implementation-of-the-Longstaff-Schwartz-Method-for-American-Option-Pricing>

5.4 Choix, stabilité et extensions de la régression

L'estimation de la valeur de continuation par régression constitue un point sensible de l'algorithme. Le choix des fonctions de base représente un compromis entre précision et stabilité numérique. Des bases trop simples introduisent un biais, tandis que des bases trop riches peuvent engendrer des instabilités ou un surcoût de calcul.

Étude initiale : Bases quadratiques. Dans un premier temps, une étude comparative a été menée entre la base canonique $(1, S, S^2)$ et la base de **polynômes d'Hermite de degré 2** $(1, S, S^2 - 1)$. Bien que les benchmarks montrent peu de différence en termes de précision pour ce problème spécifique, la base d'Hermite a été retenue par défaut pour ses meilleures propriétés théoriques de conditionnement (matrice $A^T A$).

Extensions et bases avancées. Afin d'affiner la capture de la valeur de continuation, nous avons étendu l'implémentation à trois autres familles de bases, souvent citées dans la littérature spécialisée [2] :

1. **Laguerre** : Polynômes orthogonaux sur $[0, \infty]$, historiquement suggérés par Longstaff et Schwartz [1] pour leur adaptation naturelle aux prix d'actifs positifs (pondération par exponentielle décroissante).
2. **Tchebychev** : Polynômes minimisant l'erreur d'interpolation sur $[-1, 1]$. Cette base a nécessité l'implémentation d'un kernel de réduction Min-Max sur GPU pour normaliser les prix à chaque pas de temps.
3. **Cubique** : Base monômiale enrichie au degré 3 $(1, S, S^2, S^3)$.

La figure 21 illustre l'impact du choix de la base de régression sur la précision de l'estimation.

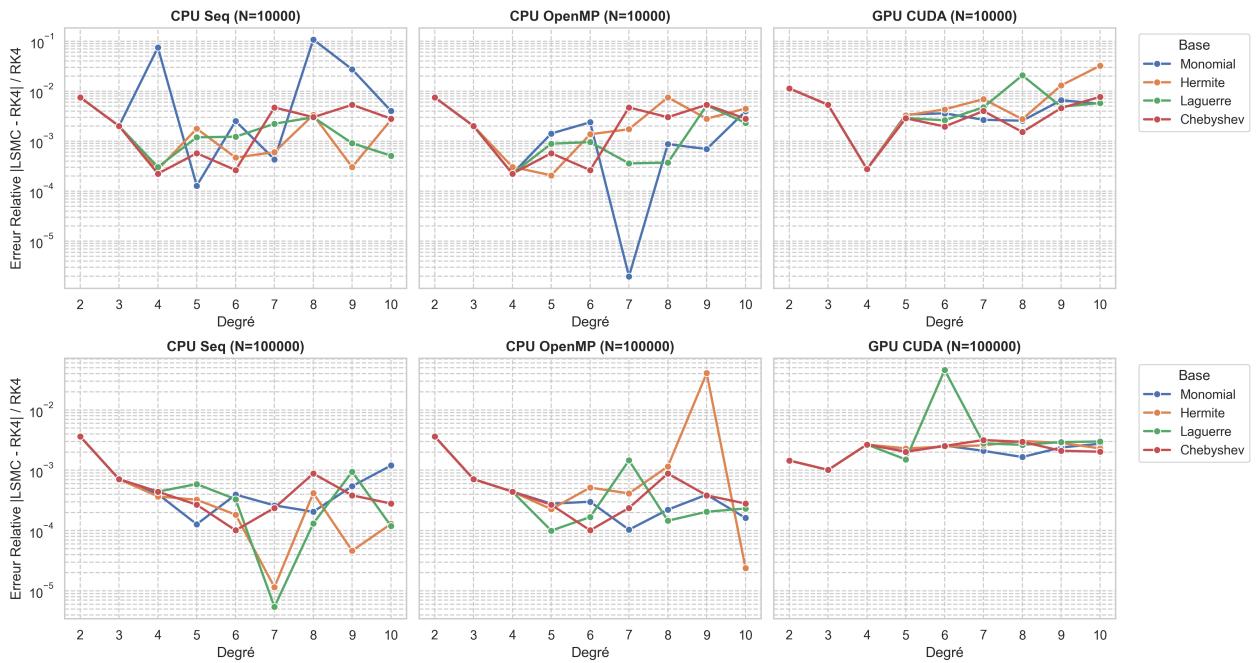


Figure 21: Comparaison de la convergence de l'erreur relative selon le nombre de trajectoires ($N = 10^4, 10^5$) et l'architecture.

Figure 4 Accuracy comparison of LSMC methods using different basis functions: vanilla put options.

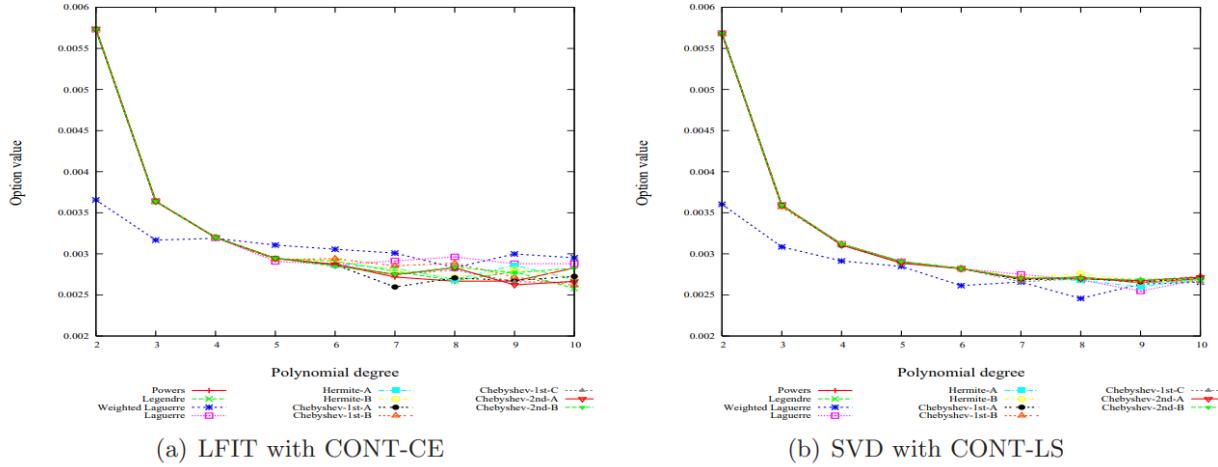


Figure 22: Évolution théorique attendue de la précision (Source : [9]). Voici ce que l'on devrait observer idéalement.

Analyse de la convergence (Figure 21). Les résultats présentés en figure 21 montrent l'évolution de l'erreur relative par rapport au prix RK4 de référence. Cette dynamique peut être mise en perspective avec la Figure 22 (issue de [9]) qui illustre les attentes théoriques de convergence. Contrairement à l'intuition théorique qui suggère une amélioration monotone, les courbes observées ne présentent pas une décroissance régulière. Plusieurs facteurs peuvent expliquer ce comportement mitigé et "non concluant" sur ce cas test spécifique :

- **Phénomène de Runge :** L'utilisation de bases polynomiales de degré élevé sur des points non uniformément répartis (les trajectoires stochastiques) peut induire de fortes oscillations aux bords du domaine (valeurs extrêmes de S_t), dégradant la qualité globale de la régression [3].
- **Conditionnement de la matrice :** Pour des degrés élevés ($d > 5$), la matrice de Gram $A^T A$ devient mal conditionnée, en particulier pour la base Monomiale, ce qui amplifie les erreurs numériques lors de la résolution du système linéaire. Bien que les bases orthogonales (Laguerre, Hermite) soient censées atténuer ce problème, le bruit de Monte Carlo semble ici dominer les gains théoriques.
- **Compromis Biais-Variance :** Augmenter le degré réduit le biais de modélisation (capacité à fitter la "vraie" fonction de continuation) mais augmente la variance de l'estimateur, car le modèle capture le bruit statistique des trajectoires (sur-apprentissage).
- **Changement de référence :** Il serait également pertinent de comparer ces résultats avec des schémas d'Euler explicite ou implicite, la méthode RK4 pouvant présenter des instabilités dans certains régimes stochastiques.

Ces observations soulignent la difficulté pratique d'obtenir une convergence "parfaite" avec des méthodes de régression globale sur des degrés élevés sans un nombre de trajectoires extrêmement grand ($N \gg 10^5$).

5.5 Compromis précision–performance

Enfin, ce projet a mis en évidence le compromis fondamental entre précision numérique et temps de calcul. L'augmentation du nombre de trajectoires améliore la convergence statistique, mais accroît fortement le coût de calcul, en particulier lors des phases de régression et de backward induction. Ce compromis a guidé le choix des paramètres expérimentaux et l'analyse comparative des architectures CPU et GPU.

6 Perspectives et améliorations possibles

Le travail réalisé dans le cadre de ce projet a permis de mettre en œuvre une version fonctionnelle et performante de l'algorithme de Longstaff–Schwartz sur différentes architectures de calcul. Plusieurs pistes d'amélioration et d'extension peuvent toutefois être envisagées, tant sur le plan algorithmique que sur le plan des performances et des modèles financiers considérés.

6.1 Améliorations algorithmiques

Une première piste d'amélioration concerne non pas le degré de la base polynomiale, mais la nature même de l'approximation utilisée pour estimer la valeur de continuation. Bien que différentes bases polynomiales aient été explorées au cours du projet, celles-ci montrent des gains limités au-delà d'un certain degré, ce qui suggère une limite structurelle des approximations globales dans un contexte stochastique bruité.

Dans cette optique, plusieurs extensions méthodologiques pourraient être envisagées :

- l'intégration de méthodes de **Monte Carlo Multi-Niveaux (MLMC)** [11], permettant de réduire la variance et le coût de calcul en combinant des simulations de différentes résolutions temporelles ;
- la mise en œuvre de **régressions locales** (Local Least Squares) [10], plus adaptées à la capture de non-linéarités locales de la fonction de continuation ;
- l'exploration d'approches de **parallélisation temporelle**, telles que les méthodes de type Parareal [12], visant à atténuer la contrainte séquentielle imposée par la backward induction, bien que leur intégration dans le cadre du LSMC reste non triviale.

Ces approches offrent un potentiel d'amélioration de la précision et de l'efficacité de l'estimation de la valeur de continuation, au prix d'une complexité algorithmique accrue et d'un risque plus élevé de surapprentissage.

Par ailleurs, l'utilisation de techniques classiques de réduction de variance (antithetic variates, control variates, stratified sampling) constitue une piste complémentaire pertinente pour améliorer la convergence statistique de la méthode de Monte Carlo, en réduisant le nombre de trajectoires nécessaires pour atteindre une précision donnée.

6.2 Extensions du modèle financier

Le cadre retenu dans ce projet repose sur un mouvement brownien géométrique, modèle de référence mais relativement simplificateur. Plusieurs extensions naturelles peuvent être envisagées :

- introduction d'un taux de dividende continu;
- prise en compte d'une volatilité locale ou stochastique (modèles de Heston, SABR);
- extension à des options multi-actifs ou dépendant de plusieurs sous-jacents;
- valorisation de produits exotiques présentant des payoffs path-dépendants.

L'algorithme de Longstaff–Schwartz conserve une grande flexibilité face à ces extensions, ce qui constitue l'un de ses principaux avantages par rapport aux méthodes analytiques.

6.3 Optimisations CPU avancées

Sur CPU, plusieurs optimisations supplémentaires pourraient être envisagées :

- vectorisation explicite via les instructions SIMD (AVX2, AVX-512);

- meilleure gestion de la localité mémoire pour réduire la pression sur la bande passante RAM;
- utilisation de bibliothèques numériques optimisées pour les régressions linéaires.

Ces optimisations permettraient d'exploiter plus finement l'architecture matérielle, en particulier pour des tailles de problème intermédiaires où le GPU n'est pas toujours optimal.

6.4 Optimisation et généralisation de l'implémentation GPU

Du côté GPU, plusieurs améliorations sont envisageables :

- implémentation complète de la régression par moindres carrés directement sur GPU, sans synchronisation CPU;
- utilisation de bibliothèques CUDA spécialisées (cuBLAS, CUB, Thrust) pour les réductions et opérations linéaires;
- exploration de stratégies multi-GPU pour des simulations de très grande dimension.

Une telle approche permettrait de réduire encore les coûts de communication et d'atteindre des gains de performance plus proches du potentiel théorique du GPU.

6.5 Perspectives méthodologiques

Enfin, ce projet ouvre la voie à des comparaisons plus larges entre différentes approches numériques pour le pricing d'options américaines, notamment :

- les méthodes par arbres** (Binomiaux, Trinomiaux) : très utilisées pour leur flexibilité sur les options américaines (simple dimension);
- les méthodes itératives** (Différences Finies) : qui résolvent l'EDP de Black-Scholes-Merton avec condition de frontière libre ($P \geq (K - S)^+$);
- les méthodes spectrales** (Transformée de Fourier) : permettant des évaluations très rapides pour certains modèles (e.g., Heston, Lévy), bien que moins naturellement adaptées à l'exercice anticipé américain (nécessitant des techniques type méthodes de Spitzer).

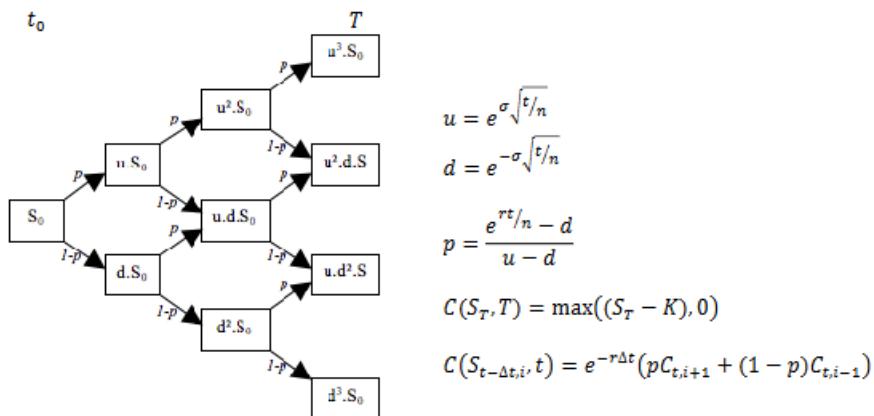


Figure 23: Illustration d'un arbre binomial pour la valorisation d'options.

Ces perspectives permettraient d'évaluer plus finement les compromis entre précision, coût de calcul et flexibilité des différentes méthodes, dans un contexte de finance quantitative moderne.

7 Organisation du travail

Le projet P1RV a été mené sur l'ensemble du premier semestre selon une organisation progressive, combinant approfondissement théorique, développement logiciel itératif et analyse des performances. Le travail s'est appuyé sur une démarche incrémentale, avec des phases successives de conception, d'implémentation, de refactorisation et d'optimisation, comme en témoigne l'historique détaillé du dépôt Git.

7.1 Découpage du projet

Le développement du projet a été structuré autour des grandes étapes suivantes :

- étude du cadre théorique : options américaines, Monte Carlo, backward induction et algorithme de Longstaff–Schwartz;
- implémentation d'une version CPU séquentielle servant de référence;
- restructuration complète du code afin de séparer clairement simulation, régression et logique LSMC;
- parallélisation CPU avec OpenMP;
- ajout progressif d'un backend GPU CUDA expérimental;
- mise en place d'outils d'export des résultats (CSV) et de visualisation;
- campagnes de tests, mesures de performances et nettoyage final du code;
- rédaction et structuration du rapport.

Ce découpage a permis de valider progressivement chaque brique fonctionnelle avant d'aborder les aspects avancés de parallélisation et d'optimisation.

7.2 Organisation du développement et itérations

Le développement s'est appuyé sur un processus itératif, visible dans l'historique du dépôt Git :

- création initiale du projet et mise en place de la structure `src/include`;
- premières implémentations du GBM, de la régression OLS et du LSMC;
- phases de refonte complètes du code afin d'améliorer la lisibilité, la modularité et les performances;
- intégration progressive d'OpenMP sur les boucles critiques;
- ajout d'une interface de visualisation et d'export des résultats (scripts Python, Streamlit);
- implémentation d'un backend CUDA complet incluant la simulation GBM, le calcul des payoffs, la backward induction et les réductions nécessaires à la régression;
- nettoyage approfondi du dépôt après des problèmes liés à des fichiers CSV volumineux et à l'historique Git.

Cette approche incrémentale a permis de maintenir un code fonctionnel à chaque étape tout en introduisant progressivement des optimisations plus complexes.

Dans une phase avancée du projet (dépôt `lsmc-openmp`), une interface graphique utilisateur a été développée en Python afin de faciliter la visualisation des résultats et la compréhension des mécanismes sous-jacents à l'algorithme LSMC. Cette interface permet de paramétriser la simulation (S_0, K, r, σ, T), de lancer l'exécutable C++ en arrière-plan et de visualiser dynamiquement les trajectoires Monte Carlo exportées (Figure 24).

Conçue comme un outil d'aide à l'analyse et à la compréhension, cette interface ne vise pas à remplacer les scripts en ligne de commande utilisés pour les benchmarks de performance, mais constitue un support pédagogique essentiel. Elle a permis de valider le comportement de la diffusion du sous-jacent, d'explorer l'influence des paramètres du modèle et d'illustrer concrètement le fonctionnement des méthodes de Monte Carlo et de l'exercice anticipé.

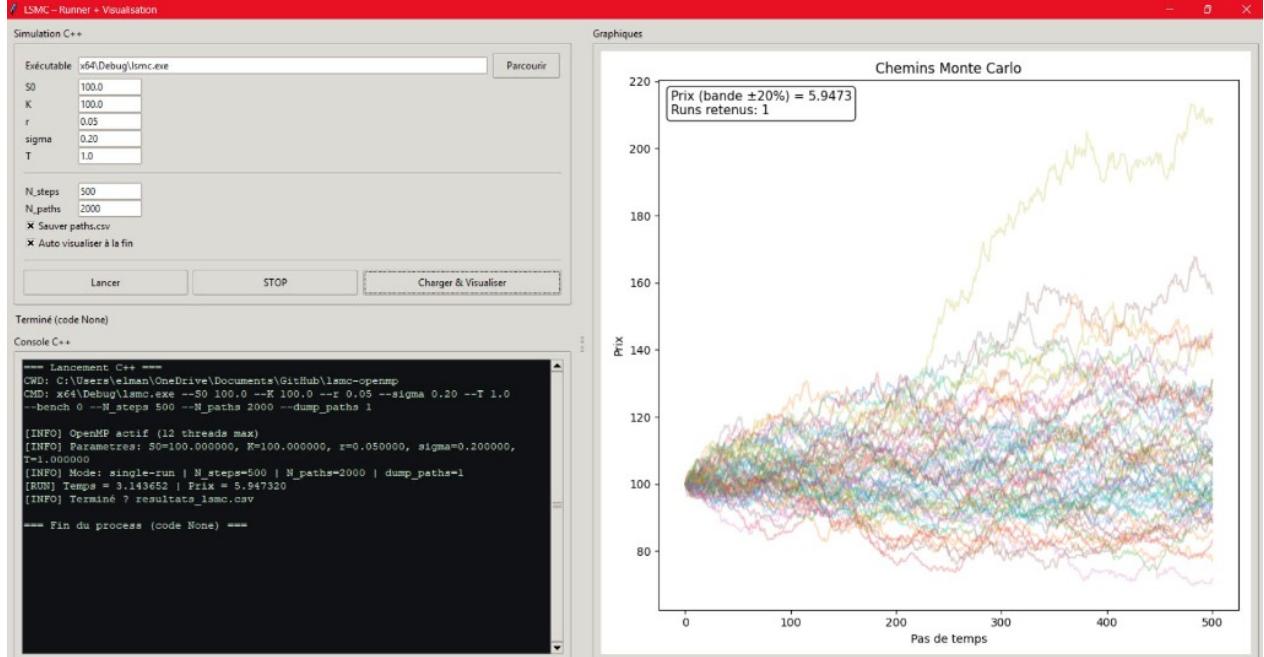


Figure 24: Interface de paramétrage et de lancement (Projet Initial lsmc-openmp).

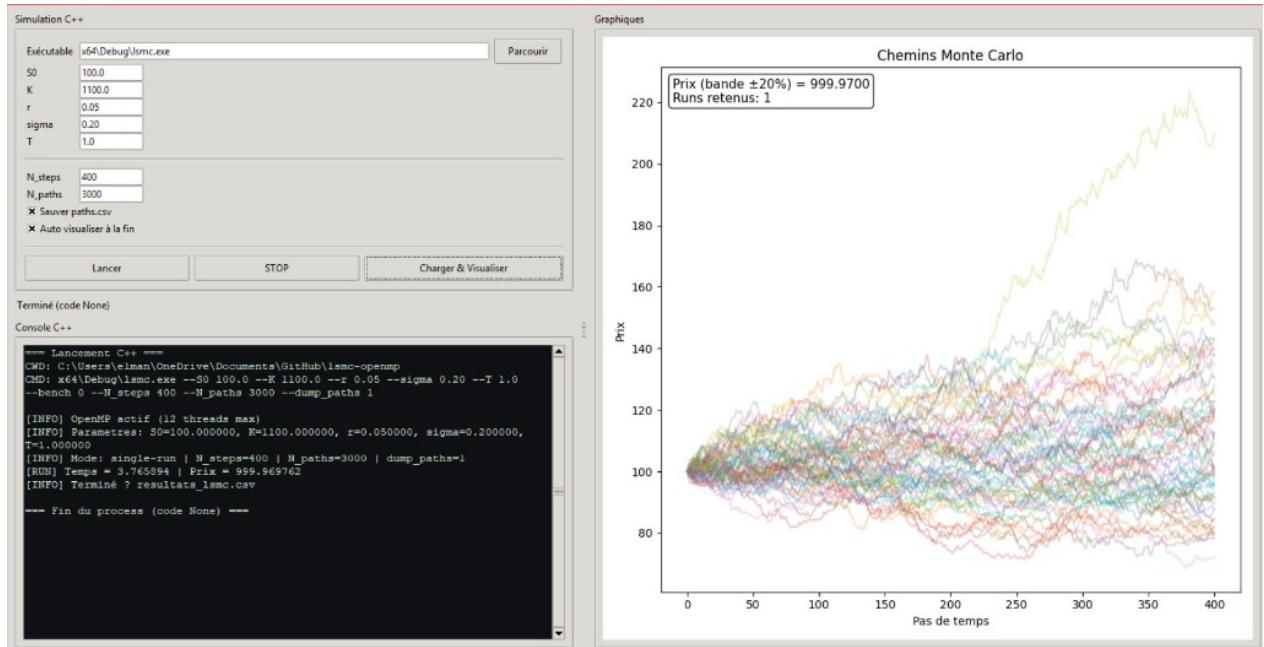


Figure 25: Visualisation des trajectoires Monte Carlo et du prix estimé via l'interface Python.

7.3 Répartition des tâches et binôme

Afin d'optimiser notre efficacité, nous nous sommes répartis les tâches selon nos affinités et compétences techniques.

Responsabilités Techniques

Narjisso El Manssouri a pris la responsabilité de l'expérience utilisateur, de l'interface (*Frontend*) et de la validation théorique :

- Conception de l'interface graphique (GUI) pour rendre l'outil accessible.
- Développement de l'application interactive Python.
- Visualisation des résultats (trajectoires, convergence) et validation de la cohérence théorique.

Florian Barbe a pris la responsabilité du "coeur de calcul" (*Core Engine*). Son travail s'est concentré sur la performance et l'architecture logicielle C++/CUDA :

- Modélisation mathématique et implémentation C++.
- Développement de l'algorithme LSMC et des solveurs FDM.
- Parallélisation CPU avec OpenMP.
- Développement complet du kernel CUDA et de l'implémentation GPU.
- Benchmarking et optimisation des performances bas niveau.

7.4 Retours d'expérience et témoignages

Au-delà de la répartition technique, ce projet a été l'occasion d'une montée en compétence individuelle.

Narjisso El Manssouri

"La principale difficulté a été de savoir dans quelle direction avancer au début du projet : je comprenais déjà l'objet financier d'une option américaine, mais pas les mathématiques et la logique algorithmique derrière le pricing. De ce fait, l'entrée en matière a été plus lente, notamment pour comprendre la structure du code que l'on pouvait et devait faire, ainsi que la façon dont l'exercice anticipé est traité avant de pouvoir coder efficacement."

Ma contribution s'est donc d'abord orientée vers la partie théorique, afin de clarifier les hypothèses, le déroulé de la méthode de simulation et le rôle de chaque bloc du programme, puis vers la réalisation d'une interface graphique en Python pour rendre l'outil exploitable et lisible : elle permet de saisir les paramètres (S_0, K, r, σ, T), de choisir le nombre de pas de temps et de trajectoires, de lancer l'exécutable C++ et suivre son exécution via une console intégrée, d'exporter les trajectoires en CSV, puis de charger et visualiser les chemins Monte Carlo sur un graphique avec un récapitulatif du prix estimé.

J'ai particulièrement apprécié ce projet car il correspond exactement à ce que je cherchais dans mon option : consolider des bases en C++ et en simulation de données, tout en les appliquant à un cas concret de finance de marché, avec un vrai enjeu de compréhension méthodologique et de mise en production via un outil de visualisation."

Florian Barbe

"Ma contribution s'est principalement concentrée sur la partie algorithmique et les aspects de performance du projet. L'une des premières difficultés a été de bien me situer dans le cadre financier des options américaines. Si le principe général était clair, la logique de l'exercice anticipé et son impact sur la structure du calcul demandaient une compréhension plus fine pour pouvoir être traduits correctement en algorithme. Cette phase a été plus longue que prévu, mais elle a été essentielle pour comprendre le fonctionnement global du LSMC et éviter une implémentation purement mécanique.

La mise en œuvre de l'accélération GPU avec CUDA a également représenté un défi important. Les premières tentatives d'intégration se sont révélées complexes, tant sur le plan de l'architecture du code que de la gestion mémoire et des synchronisations imposées par la backward induction. Ces difficultés ont conduit à la décision de créer un second dépôt Git, spécifiquement dédié à une implémentation plus propre et mieux adaptée au calcul sur GPU.

Ce projet a néanmoins été particulièrement stimulant sur le plan technique. Le moment où les premiers tests sur GPU ont montré des temps de calcul largement inférieurs aux versions CPU a été un soulagement, vu le nombre d'heures que j'avais passées dessus sans succès. Voir concrètement des calculs de Monte Carlo s'exécuter en une fraction du temps initial a été très motivant et a donné un sens immédiat aux efforts fournis en optimisation et en parallélisation. Cette expérience a renforcé mon intérêt pour le calcul parallèle et m'a permis de mesurer de manière très concrète l'impact des choix d'architecture et d'implémentation sur les performances."

Cette séparation claire (Back-end en C++/CUDA vs Front-end en Python) nous a permis d'avancer en parallèle : pendant que le moteur de calcul était optimisé, l'interface utilisateur était développée pour consommer les résultats produits.

7.5 Outils et environnement collaboratif

Le projet s'est appuyé sur les outils suivants :

- **Git / GitHub** pour le suivi du développement et l'historique des itérations;
- **CMake** pour la configuration et la compilation du projet;
- **OpenMP** pour la parallélisation CPU;
- **CUDA C++** pour l'implémentation GPU;
- **Python** et **Streamlit** pour l'analyse et la visualisation des résultats;
- **Overleaf** pour la rédaction du rapport.

L'utilisation intensive de Git a joué un rôle central, notamment pour gérer les phases de refonte, corriger des erreurs structurelles (fichiers volumineux, historique trop lourd) et stabiliser le dépôt en fin de projet.

7.6 Gestion du temps et avancement

Le travail a été réparti sur l'ensemble du semestre avec une montée en complexité progressive :

- début de semestre : compréhension théorique, premières implémentations CPU séquentielles;
- milieu de semestre : restructuration du code, parallélisation OpenMP, premières analyses de performance;
- fin de semestre : implémentation CUDA complète, optimisation mémoire, nettoyage du dépôt Git, analyse comparative et rédaction finale.

Cette organisation a permis d'anticiper les difficultés techniques, notamment celles liées au calcul parallèle, à la gestion mémoire et aux limitations structurelles de l'algorithme LSMC, tout en assurant la livraison d'un projet fonctionnel, documenté et cohérent avec les objectifs pédagogiques du P1RV.

8 Conclusion

Ce projet a permis de concevoir, implémenter et analyser une version performante de l’algorithme de Longstaff–Schwartz pour le pricing d’options américaines, en combinant rigueur mathématique, validation numérique et étude approfondie des performances sur différentes architectures de calcul.

L’implémentation séquentielle sur CPU a servi de référence fonctionnelle et a permis de valider la cohérence de l’algorithme. La parallélisation via OpenMP a montré des gains mesurés mais limités, principalement contraints par la bande passante mémoire et la présence de phases séquentielles incompressibles. En revanche, l’implémentation GPU avec CUDA a démontré des accélérations significatives pour les phases massivement parallèles, en particulier la simulation Monte Carlo des trajectoires, avec des speedups pouvant atteindre un ordre de grandeur par rapport au CPU.

Les résultats obtenus mettent toutefois en évidence une limite structurelle fondamentale du LSMC : la backward induction impose une dépendance temporelle forte qui empêche toute parallélisation complète dans la dimension du temps. Le goulet d’étranglement réside donc dans l’estimation séquentielle de la valeur de continuation à chaque pas de temps, ce qui plafonne l’accélération théorique, même sur des architectures massivement parallèles.

La validation croisée avec des méthodes déterministes de différences finies a confirmé la cohérence numérique des prix obtenus, les écarts restant compatibles avec la variance statistique attendue d’une méthode de Monte Carlo. L’étude du choix des bases de régression a par ailleurs mis en évidence le compromis biais–variance inhérent à l’algorithme, ainsi que les limites pratiques des régressions polynomiales de degré élevé en présence de bruit stochastique.

En conclusion, ce travail montre que le GPU constitue une solution particulièrement adaptée pour les méthodes de Monte Carlo à grande échelle, tout en soulignant que les gains de performance sont intrinsèquement bornés par la structure algorithmique du LSMC. Ces résultats ouvrent la voie à des approches hybrides (batching d’options, parallélisme asynchrone) visant à contourner la barrière séquentielle temporelle, seule restriction à l’essor du LSMC sur les architectures exascale.

Annexes

A Résolution de l'EDS du GBM

On considère le processus $(S_t)_{t \geq 0}$ solution, sous la mesure risque-neutre \mathbb{Q} , de l'équation différentielle stochastique

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

où r est le taux sans risque, $\sigma > 0$ la volatilité constante, et $(W_t^{\mathbb{Q}})_{t \geq 0}$ un mouvement brownien standard sous \mathbb{Q} .

Comme $S_t > 0$ presque sûrement, la fonction \ln est bien définie. En appliquant la formule d'Itô à $f(S_t) = \ln(S_t)$, on obtient

$$d\ln(S_t) = f'(S_t) dS_t + \frac{1}{2} f''(S_t) (dS_t)^2 = \frac{1}{S_t} dS_t - \frac{1}{2S_t^2} (dS_t)^2.$$

En utilisant les règles du calcul stochastique

$$(dW_t^{\mathbb{Q}})^2 = dt, \quad dt dW_t^{\mathbb{Q}} = 0, \quad (dt)^2 = 0,$$

on a

$$dS_t = rS_t dt + \sigma S_t dW_t^{\mathbb{Q}} \implies (dS_t)^2 = \sigma^2 S_t^2 (dW_t^{\mathbb{Q}})^2 = \sigma^2 S_t^2 dt.$$

En substituant dans la formule d'Itô, il vient

$$d\ln(S_t) = \frac{1}{S_t} \left(rS_t dt + \sigma S_t dW_t^{\mathbb{Q}} \right) - \frac{1}{2S_t^2} (\sigma^2 S_t^2 dt),$$

soit

$$d\ln(S_t) = \left(r - \frac{\sigma^2}{2} \right) dt + \sigma dW_t^{\mathbb{Q}}.$$

En intégrant entre t_n et t_{n+1} (avec $\Delta t = t_{n+1} - t_n$) :

$$\ln(S_{t_{n+1}}) - \ln(S_{t_n}) = \left(r - \frac{\sigma^2}{2} \right) \Delta t + \sigma (W_{t_{n+1}}^{\mathbb{Q}} - W_{t_n}^{\mathbb{Q}}).$$

Les incrémentations du mouvement brownien sous \mathbb{Q} vérifient

$$W_{t_{n+1}}^{\mathbb{Q}} - W_{t_n}^{\mathbb{Q}} \sim \mathcal{N}(0, \Delta t).$$

Il existe donc $Z_n \sim \mathcal{N}(0, 1)$ tel que

$$W_{t_{n+1}}^{\mathbb{Q}} - W_{t_n}^{\mathbb{Q}} = \sqrt{\Delta t} Z_n.$$

En prenant l'exponentielle, on obtient la formule discrète exacte

$$S_{t_{n+1}} = S_{t_n} \exp \left(\left(r - \frac{\sigma^2}{2} \right) \Delta t + \sigma \sqrt{\Delta t} Z_n \right),$$

et, en particulier, l'expression explicite en temps continu

$$S_t = S_0 \exp \left(\left(r - \frac{\sigma^2}{2} \right) t + \sigma W_t^{\mathbb{Q}} \right).$$

B Rappels de Probabilités et propriétés du Mouvement Brownien

B.1 Définition et propriétés du Mouvement Brownien

Un mouvement brownien standard $(W_t)_{t \geq 0}$ est un processus stochastique caractérisé par :

1. $W_0 = 0$ presque sûrement.
2. Ses trajectoires $t \mapsto W_t$ sont continues.
3. Ses accroissements sont indépendants et stationnaires (loi normale $\mathcal{N}(0, t - s)$ pour $W_t - W_s$).

B.2 Propriétés des incrément

Pour la simulation numérique, nous utilisons la propriété d'indépendance des accroissements sur des intervalles disjoints. Si $0 \leq t_1 < t_2 < \dots < t_n$, les variables aléatoires $(W_{t_{i+1}} - W_{t_i})$ sont mutuellement indépendantes.

B.3 Loi des Grands Nombres et Monte Carlo

La convergence des méthodes de Monte Carlo repose sur la Loi Forte des Grands Nombres (LFGN) :

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N f(X_i) = \mathbb{E}[f(X)] \quad \text{p.s.}$$

L'erreur d'approximation décroît en $\mathcal{O}(1/\sqrt{N})$, conformément au Théorème Central Limite.

C Propriétés des Bases Polynomiales et Relations de Récurrence

Cette annexe détaille les propriétés mathématiques des bases de régression implémentées.

C.1 Base Canonique (Monomiale)

Base la plus simple, définie par $P_n(x) = x^n$. Elle n'est pas orthogonale pour le produit scalaire standard, ce qui conduit à des matrices de Gram ($A^T A$) mal conditionnées (problème de la matrice de Hilbert) lorsque le degré augmente.

C.2 Polynômes de Laguerre $L_n(x)$

Orthogonaux sur $[0, +\infty[$ pour le poids $w(x) = e^{-x}$. Adaptés pour modéliser des fonctions de variables positives (comme les prix d'actifs).

- $L_0(x) = 1$
- $L_1(x) = 1 - x$
- Récurrence : $(n+1)L_{n+1}(x) = (2n+1-x)L_n(x) - nL_{n-1}(x)$

C.3 Polynômes d'Hermite $H_n(x)$

Orthogonaux sur $]-\infty, +\infty[$ pour le poids $w(x) = e^{-x^2}$. Naturellement liés à la loi Normale.

- $H_0(x) = 1$
- $H_1(x) = 2x$
- Récurrence : $H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$

C.4 Polynômes de Legendre $P_n(x)$

Orthogonaux sur $[-1, 1]$ pour le poids $w(x) = 1$. Nécessitent de redimensionner les variables (scaling) dans l'intervalle $[-1, 1]$ avant régression.

- $P_0(x) = 1$
- $P_1(x) = x$
- Récurrence : $(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$

C.5 Polynômes de Chebyshev (1ère espèce) $T_n(x)$

Orthogonaux sur $[-1, 1]$ pour le poids $w(x) = \frac{1}{\sqrt{1-x^2}}$. Ils minimisent l'effet de Runge (oscillation aux bords) en interpolation.

- $T_0(x) = 1$
- $T_1(x) = x$
- Récurrence : $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$

D Notations et Conventions

D.1 Probabilités et Processus Stochastiques

Notation	Description
$\Omega, \mathcal{F}, \mathbb{P}$	Espace probabilisé filtré standard.
\mathbb{Q}	Mesure risque-neutre, sous laquelle les actifs actualisés sont des martingales.
$(\mathcal{F}_t)_{t \geq 0}$	Filtration naturelle engendrée par le mouvement brownien, représentant l'information disponible à l'instant t .
$W_t^{\mathbb{Q}}$	Mouvement Brownien standard sous la mesure \mathbb{Q} .
S_t	Valeur du sous-jacent à l'instant t .
$\mathbb{E}^{\mathbb{Q}}[\cdot \mathcal{F}_t]$	Espérance conditionnelle sous la mesure \mathbb{Q} sachant l'information à l'instant t .

D.2 Modélisation Financière

Notation	Description
S_0	Prix initial du sous-jacent ($t = 0$).
K	Prix d'exercice (Strike) de l'option.
T	Maturité de l'option (en années).
r	Taux d'intérêt sans risque constant.
σ	Volatilité constante du sous-jacent.
$\Phi(S)$	Fonction de payoff ou valeur intrinsèque. Pour un Put : $\Phi(S) = (K - S)^+$.
V_t	Prix de l'option américaine à l'instant t .
$C(t, S_t)$	Valeur de continuation à l'instant t sachant S_t .
τ	Temps d'arrêt correspondant à la stratégie d'exercice optimale.

D.3 Méthodes Numériques (LSMC)

Notation	Description
N ou N_{steps}	Nombre de pas de temps de la discréétisation.
Δt	Pas de temps uniforme ($\Delta t = T/N$).
t_n	Dates de discréétisation ($t_n = n\Delta t$).
M ou N_{paths}	Nombre de trajectoires Monte Carlo simulées.

$S_{t_n}^{(i)}$	Valeur simulée du sous-jacent pour la trajectoire i à l'instant t_n .
$\psi_k(S)$	k -ième fonction de base utilisée pour la régression.
K_{reg}	Nombre total de fonctions de base (e.g., 3 pour quadratique).
β (ou a_k)	Vecteur des coefficients de régression estimés.
$\hat{C}(t_n, S)$	Estimateur numérique de la valeur de continuation obtenue par régression.
A	Matrice de design (Vandermonde généralisée) utilisée dans les moindres carrés ($A^T A \beta = A^T Y$).

D.4 Implémentation et Architecture

Notation	Description
OpenMP	<i>Open Multi-Processing</i> , API pour la programmation parallèle sur mémoire partagée (CPU).
CUDA	<i>Compute Unified Device Architecture</i> , plateforme de calcul parallèle NVIDIA.
Thread	Unité d'exécution de base. Sur GPU, un thread traite typiquement une trajectoire simulée.
Block	Regroupement de threads sur GPU partageant une mémoire locale (Shared Memory).
Grid	Ensemble des blocs lancés pour l'exécution d'un Kernel CUDA.
Kernel	Fonction exécutée sur le GPU par chaque thread de la grille.
Speedup	Gain de performance, défini comme Temps _{CPU} /Temps _{GPU} .

References

- [1] Longstaff, F. A., & Schwartz, E. S. (2001). *Valuing American Options by Simulation : A Simple Least-Squares Approach*. The Review of Financial Studies, 14(1), 113-147.
- [2] Hull, J. C. *Options, Futures, and Other Derivatives*. Pearson Education.
- [3] Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*. Springer.
- [4] Oger, G. *Mémoire de Magistère : Valorisation d'options américaines sur GPU*.
- [5] Croain, D., & Poulette. *Projet GPU : Pricing d'options américaines*.
- [6] Risks Journal (2023). *Recent Advances in American Option Pricing*. risks-11-00145.
- [7] Benguigui, M. (2015). *Valorisation d'options américaines et Value At Risk sur cluster GPU/CPU hétérogène*. Thèse de doctorat, Université Nice Sophia Antipolis.
- [8] Reesor, R. M., Stentoft, L., & Zhu, X. (2024). *A Critical Analysis of the Weighted Least Squares Monte Carlo Method for Pricing American Options*. Finance Research Letters.
- [9] Areal, N., Rodrigues, A., & Armada, M. J. R. *Improvements to the Least Squares Monte Carlo Option Valuation Method*. Source file: *Areal_Parallel_Methods.pdf*.
- [10] Detra (2023). *Risk Management with Local Least Squares Monte Carlo*. Technical Note.
- [11] Ghodssi, Z. (2021). *Multilevel Monte Carlo Calculation of American Option Prices*. Source file: *Ghodssi_2021 MLMC Valuation.pdf*.
- [12] Parareal Methods for American Options. Source file: *Parareal_American_Options.pdf*.