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

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[1].

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[2] 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.

Notations

Cadre probabiliste et filtrations

- $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$: espace probabilisé filtré satisfaisant les conditions usuelles (complétude et continuité à droite).
- $\mathbb{P}$: mesure de probabilité historique (ou réelle), décrivant la dynamique statistique observée du marché.
- $\mathbb{Q}$: mesure de probabilité risque-neutre, équivalente à $\mathbb{P}$, sous laquelle les prix actualisés sont des martingales.
- $\mathcal{F}_t$: filtration représentant l'ensemble de l'information disponible jusqu'à la date t[3].

[1] Un produit dérivé est un instrument financier dont la valeur dépend (dérive) de celle d'un actif sous-jacent, comme une action, un indice ou une matière première.

[2] La convergence des estimateurs de Monte Carlo est garantie par la Loi Forte des Grands Nombres. La vitesse de convergence est typiquement en $\mathcal{O}(1/\sqrt{N})$, où N est le nombre de simulations.

[3] Intuitivement, une filtration modélise le flux d'information croissant au fil du temps. $\mathcal{F}_t$ représente toute l'information historique disponible jusqu'à l'instant t.

Processus stochastiques

- W_t : mouvement brownien standard sous la mesure $\mathbb{P}$.
- $W_t^{\mathbb{Q}}$: mouvement brownien standard sous la mesure $\mathbb{Q}$.
- $\Delta W_{t_n} = W_{t_{n+1}} - W_{t_n}$: incrément du mouvement brownien.
- Z_n : variables aléatoires indépendantes et identiquement distribuées telles que $Z_n \sim \mathcal{N}(0,1)$, utilisées pour la simulation des incréments brownien.

Sous-jacent et dynamique

- S_t : prix du sous-jacent à la date continue t.
- S_{t_n} : prix du sous-jacent à la date discrète t_n.
- S_0 : prix initial du sous-jacent.
- μ : dérive réelle du sous-jacent sous la mesure $\mathbb{P}$.
- r : taux sans risque constant.
- q : taux de dividende continu du sous-jacent (pris égal à 0 dans ce travail).
- σ : volatilité constante du sous-jacent.

Temps, discrétisation et simulation

- T : maturité de l'option.
- $[0,T]$: horizon temporel de valorisation.
- $(t_n)_{n=0,\dots,N}$: grille temporelle de discrétisation de l'intervalle $[0,T]$.
- $\Delta t_n = t_{n+1} - t_n$: pas de temps entre deux dates consécutives.
- $\Delta t = T/N$: pas de temps constant.
- N_{steps} : nombre de pas de temps.
- N_{paths} : nombre de trajectoires fr Monte Carlo simulées.

Espérances et opérateurs

- $\mathbb{E}^{\mathbb{P}}[\cdot]$: espérance sous la mesure historique $\mathbb{P}$.
- $\mathbb{E}^{\mathbb{Q}}[\cdot]$: espérance sous la mesure risque-neutre $\mathbb{Q}$.
- $\mathbb{E}[\cdot \mid \mathcal{F}_t]$: espérance conditionnelle à l'information disponible à la date t.
- $\mathbb{I}_A$: indicatrice de l'événement A.

Option et payoffs

- K : prix d'exercice (strike).
- $\Phi(S)$: fonction de payoff de l'option.
- $\Phi(S) = \max(K - S, 0)$: payoff d'un put.
- $\Phi(S) = \max(S - K, 0)$: payoff d'un call.

Valeur de l'option et arrêt optimal

- V_t : valeur de l'option à la date continue t.
- V_{t_n} : valeur de l'option à la date discrète t_n.
- $C(t, S)$: valeur de continuation à la date t pour un prix du sous-jacent S.
- $\widehat{C}(t_n, S_{t_n})$: estimation numérique de la valeur de continuation.
- τ : temps d'arrêt représentant la date d'exercice de l'option.
- $\mathcal{T}_n$: ensemble des temps d'arrêt τ tels que $\tau \geq t_n$.

Algorithme de Longstaff–Schwartz

- $\psi_k(\cdot)$: fonctions de base utilisées pour la régression.
- $\boldsymbol{\psi}(S) = (\psi_0(S), \ldots, \psi_K(S))$: vecteur des fonctions de base.
- β_k : coefficients de la régression par moindres carrés.
- $\boldsymbol{\beta}$: vecteur des coefficients de régression.
- $\|\cdot\|_2$: norme euclidienne.
- $\arg\min$: opérateur de minimisation.

Implémentation numérique et performance

- $\mathcal{O}(\cdot)$: notation de complexité algorithmique.
- CPU : unité centrale de calcul.
- GPU : processeur graphique.
- OpenMP : bibliothèque de parallélisation CPU.
- CUDA : modèle de programmation GPU de NVIDIA.

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.

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.

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 continu, 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
$$(t_0, \ldots, t_n) \in \mathbb{R}_+^{n+1} \quad \text{tel que} \quad 0 \leq t_0 < t_1 < \cdots < t_n,$$
les variables aléatoires
$$W_{t_1} - W_{t_0},\ W_{t_2} - W_{t_1},\ \ldots,\ W_{t_n} - W_{t_{n-1}}$$
sont indépendantes, c'est-à-dire que la connaissance de la valeur d'un incrément n'apporte aucune information sur les autres.
- **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** :
$$\forall t \geq 0, \quad h > 0, W_{t+h} - W_t \sim \mathcal{N}(0, h).$$

Les trajectoires d'un mouvement brownien sont continues mais presque sûrement nulle part dérivables, 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.

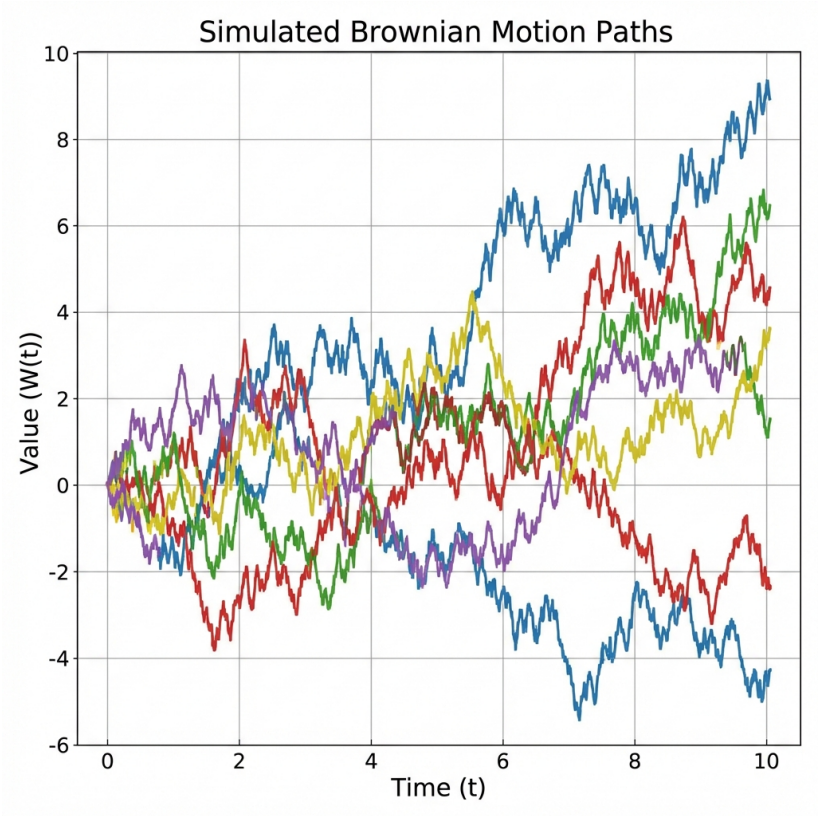


Figure 1: Réalisations de mouvements browniens standards.

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 [2]

$$\mathrm{d}S_t = \mu S_t \,\mathrm{d}t + \sigma S_t \,\mathrm{d}W_t,$$

où :

- S_t désigne la valeur du sous-jacent à l'instant $t \in \mathbb{R}_+$,
- $\mu \in \mathbb{R}$ est le paramètre de dérive, représentant le taux de croissance moyen du processus,
- $\sigma > 0$ est le paramètre de volatilité, mesurant l'intensité des fluctuations aléatoires,
- $(W_t)_{t \geq 0}$ est un mouvement brownien standard.

Cette équation est une équation différentielle stochastique linéaire au sens d'Itô[1], issue de l'équation de Black–Scholes[2].

Cette spécification implique que les variations instantanées du prix sont proportionnelles à son niveau. En particulier, si $S_0 > 0$, alors $S_t > 0$ presque sûrement pour tout $t \geq 0$, ce qui constitue une propriété essentielle pour la modélisation des prix financiers.

[1] Une équation différentielle stochastique est dite linéaire au sens d'Itô lorsque les coefficients des termes en $\mathrm{d}t$ et en $\mathrm{d}W_t$ sont des fonctions linéaires de la variable d'état. Dans ce cas, l'équation admet une solution explicite obtenue à l'aide du calcul d'Itô.

[2] Introduite par F. Black et M. Scholes dans l'article fondateur *The Pricing of Options and Corporate Liabilities* (1973), cette équation s'appuie sur le calcul stochastique développé par K. Itô dans les années 1940.

Sous ce modèle, les rendements logarithmiques

$$\ln\left(\frac{S_{t+\Delta t}}{S_t}\right)$$

sont indépendants sur des intervalles de temps disjoints et suivent une loi normale donnée par

$$\mathcal{N}\left(\left(\mu - \tfrac{1}{2}\sigma^2\right)\Delta t,\ \sigma^2 \Delta t\right).$$

Le prix S_t suit ainsi une loi log-normale et admet l'expression explicite

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

Le mouvement brownien géométrique présente alors plusieurs propriétés essentielles pour la modélisation financière :

- **Positivité du sous-jacent** : le processus S_t reste strictement positif.
- **Croissance moyenne exponentielle** : l'espérance de S_t évolue de manière exponentielle au cours du temps.
- **Volatilité proportionnelle** : l'amplitude des fluctuations aléatoires est proportionnelle au niveau du prix.

Dans le contexte de valorisation des produits dérivés, on adopte une approche sous mesure risque-neutre[1]. Sous cette mesure, en présence du taux de dividende continu $q \geq 0$, du taux d'intérêt sans risque r et de $(W_t^{\mathbb{Q}})_{t\geq 0}$, le mouvement brownien standard sous la mesure risque-neutre, la dynamique du sous-jacent s'écrit

$$\mathrm{d}S_t = (r-q)S_t\,\mathrm{d}t + \sigma S_t\,\mathrm{d}W_t^{\mathbb{Q}},$$

Dans le cadre de ce travail, on se place dans un environnement sans versement de dividendes, correspondant au cas $q = 0$. La dynamique du sous-jacent sous la mesure risque-neutre se réduit alors à

$$\mathrm{d}S_t = rS_t\,\mathrm{d}t + \sigma S_t\,\mathrm{d}W_t^{\mathbb{Q}},$$

et admet pour solution explicite

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

la démonstration du passage de l'équation différentielle stochastique à cette expression explicite étant détaillée en annexe A (inspirée de [2]).

Cette modélisation constitue le cadre théorique retenu pour la simulation des trajectoires du sous-jacent et l'application de l'algorithme de Longstaff–Schwartz dans la suite de ce travail.

2.2.2 Discrétisation temporelle

La simulation numérique des trajectoires du sous-jacent repose sur une discrétisation du temps sur une grille finie :

$$0 = t_0 < t_1 < \cdots < t_N = T, \qquad \Delta t_n = t_{n+1} - t_n.$$

Dans le cas d'un pas de temps constant, on a $\Delta t_n = \Delta t = T/N$.

Dans le cadre de ce travail, le sous-jacent est modélisé sous la mesure risque-neutre par un mouvement brownien géométrique sans dividendes, de dynamique

$$\mathrm{d}S_t = rS_t\,\mathrm{d}t + \sigma S_t\,\mathrm{d}W_t^{\mathbb{Q}}.$$

[1] Sous la mesure risque-neutre, les prix actualisés des actifs financiers sont des martingales, ce qui permet de valoriser les produits dérivés par espérance mathématique actualisée sans faire intervenir les préférences au risque des investisseurs.

Ce processus admet une solution explicite, ce qui permet de simuler exactement l'évolution du prix entre deux dates consécutives de la grille temporelle. En appliquant la formule d'Itô à la fonction logarithme, on obtient la relation de transition suivante :

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

où $(Z_n)_n$ est une suite de variables aléatoires indépendantes et identiquement distribuées suivant une loi normale centrée réduite,

$$Z_n \sim \mathcal{N}(0,1).$$

Cette discrétisation correspond à l'utilisation de la solution exacte du mouvement brownien géométrique et ne souffre donc pas d'erreur de schéma temporel. . Elle constitue la base des simulations de Monte Carlo mises en œuvre dans ce travail et permet de générer efficacement un grand nombre de trajectoires indépendantes du sous-jacent.

2.3 Algorithme de Longstaff–Schwartz (LSMC)

L'algorithme de Longstaff–Schwartz, également appelé *Least Squares Monte Carlo (LSMC)* [1], 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

Dans le cas d'une option américaine, le détenteur dispose à chaque date d'exercice possible de la liberté d'exercer immédiatement l'option ou de la conserver afin de potentiellement exercer plus tard. Cette flexibilité introduit un *problème d'arrêt optimal* : il s'agit de choisir une date d'exercice qui maximise la valeur espérée actualisée du payoff.

On se place sur une grille d'exercice discrète $(t_n)_{n=0,\dots,N}$ avec $t_N = T$ et $\Delta t_n = t_{n+1} - t_n$. 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 (par exemple, pour un put : $\Phi(S) = \max(K - S, 0)$). À la date t_n, deux choix sont possibles :

- **Exercer immédiatement :** le gain obtenu est $\Phi(S_{t_n})$.
- **Continuer et ne pas exercer :** on conserve l'option et sa valeur provient des gains futurs possibles, que l'on résume par la *valeur de continuation*.

La *valeur de continuation* $C(t_n, S_{t_n})$ est définie comme la valeur espérée actualisée de l'option à la date suivante, conditionnellement à l'information disponible à la date t_n. Sous la mesure risque-neutre $\mathbb{Q}$, elle s'écrit :

$$C(t_n, S_{t_n}) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r\Delta t_n}\, V_{t_{n+1}} \mid \mathcal{F}_{t_n}\right].$$

Dans un *cadre markovien*[1] — en particulier dans le cas du mouvement brownien géométrique — la tribu $\mathcal{F}_{t_n}$, qui représente l'ensemble de l'information accumulée jusqu'au temps t_n, est équivalente, pour la prévision de l'évolution future, à la connaissance du seul état courant S_{t_n}. Par conséquent, l'espérance conditionnelle par rapport à $\mathcal{F}_{t_n}$ se réduit à une espérance conditionnelle par rapport à S_{t_n}, ce qui justifie la notation $C(t_n, S_{t_n})$ plutôt que $C(t_n, \mathcal{F}_{t_n})$.

La règle d'exercice optimal à la date t_n consiste alors à comparer l'exercice immédiat à la continuation :

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

[1] On dit qu'un processus stochastique est étudié dans un *cadre markovien* lorsqu'il vérifie la propriété de Markov : conditionnellement à l'état présent, son évolution future est indépendante de son passé. Autrement dit, toute l'information pertinente pour prédire l'avenir du système est contenue dans son état courant. Dans le cas des équations différentielles stochastiques, cette propriété est assurée lorsque les coefficients de dérive et de diffusion dépendent uniquement du temps et de la valeur actuelle du processus, et non de son historique.

Autrement dit, l'exercice est optimal si et seulement si

$$\Phi(S_{t_n}) \geq C(t_n, S_{t_n}),$$

et dans le cas contraire il est optimal de conserver l'option. Cette comparaison définit une *région d'exercice* (où l'on exerce) et une *région de continuation* (où l'on attend) dans l'espace des états.

Formellement, la détermination de la stratégie d'exercice peut être décrite via un temps d'arrêt τ prenant ses valeurs dans l'ensemble discret des dates d'exercice $\{t_0, \ldots, t_N\}$. Pour une date donnée t_n, le problème de valorisation d'une option américaine s'écrit alors comme un problème d'arrêt optimal :

$$V_{t_n} = \sup_{\tau \in \mathcal{T}_n} \mathbb{E}^{\mathbb{Q}}\Big[e^{-r(\tau - t_n)}\, \Phi(S_\tau) \mid \mathcal{F}_{t_n}\Big],$$

où $\mathcal{T}_n = \{\tau \geq t_n \mid \tau$ est un temps d'arrêt à valeurs dans $\{t_0, \ldots, t_N\}\}$, c'est-à-dire l'ensemble des temps d'arrêt[1] τ tels que $\tau \geq t_n$. Cette formulation met en évidence que le prix de l'option américaine correspond à la meilleure espérance actualisée que l'on puisse obtenir en choisissant de manière optimale la date d'exercice.

En pratique, le principal enjeu numérique est l'évaluation de $C(t_n, S_{t_n})$: il s'agit d'une espérance conditionnelle (donc d'une *quantité fonctionnelle de l'état* [2]) qui n'admet pas, en général, de forme fermée. L'algorithme de Longstaff–Schwartz répond à cette difficulté en approximant la valeur de continuation par régression sur une base de fonctions de S_{t_n} à partir de trajectoires simulées [1], permettant ainsi d'implémenter la règle d'exercice précédente par induction arrière.

2.3.2 Régression par moindres carrés

Dans l'algorithme LSMC, la valeur de continuation n'est pas calculée analytiquement mais approximée par régression. À chaque date d'exercice, les flux futurs actualisés sont projetés sur un espace de fonctions de base $\{\psi_k(S_{t_n})\}$:

$$C(t_n, S_{t_n}) \approx \sum_k a_k\, \psi_k(S_{t_n}),$$

où les coefficients a_k sont déterminés par une régression aux moindres carrés sur les trajectoires pertinentes.

Cette approximation permet d'obtenir une estimation numérique de la valeur de continuation à partir des données simulées.

2.3.3 Stratégie d'exercice optimale

Une fois la valeur de continuation estimée, la décision d'exercice est prise en comparant, pour chaque trajectoire et chaque date, la valeur d'exercice immédiat à la valeur de continuation estimée. Si la valeur d'exercice est supérieure, l'option est exercée ; sinon, elle est conservée.

Cette procédure est appliquée de manière récursive en remontant dans le temps, ce qui permet de déterminer la stratégie d'exercice optimale et d'estimer la valeur de l'option américaine à la date initiale.

L'algorithme LSMC constitue ainsi un compromis efficace entre flexibilité du cadre stochastique et faisabilité numérique, ce qui en fait une méthode particulièrement adaptée à l'étude de la performance des architectures de calcul.

La figure 2 illustre le flux de l'algorithme LSMC dans un contexte parallèle distribué.

2.4 Synthèse algorithmique et pseudo-algorithme du LSMC

Les sections précédentes ont permis de formuler rigoureusement le problème d'arrêt optimal associé à la valorisation des options américaines, ainsi que le principe de l'approximation de la valeur de continuation par régression. On explicite à présent la manière dont ces objets théoriques sont effectivement manipulés et calculés dans le cadre numérique de l'algorithme de Longstaff–Schwartz.

[1] Un temps d'arrêt est une variable aléatoire dont la valeur est déterminée uniquement par l'information disponible jusqu'à cet instant, sans anticipation du futur (condition de non-anticipation).

[2] On entend par *quantité fonctionnelle de l'état* une grandeur qui dépend uniquement de l'état courant du système (ici la valeur du sous-jacent à la date considérée), et non de l'historique complet de sa trajectoire.

L'algorithme repose sur une discrétisation du temps et sur la simulation Monte Carlo d'un grand nombre de trajectoires du sous-jacent. Pour chaque trajectoire, l'évolution du prix est simulée sur la grille temporelle $(t_n)_{n=0,\dots,N}$. Les décisions d'exercice sont ensuite déterminées par *induction arrière*[1], en partant de la maturité et en remontant jusqu'à la date initiale.

À la date de maturité $t_N = T$, la valeur de l'option est donnée uniquement par le payoff :

$$V_{t_N}^{(i)} = \Phi(S_{t_N}^{(i)}),$$

pour chaque trajectoire i. Aucun choix n'est possible à cette date.

Pour une date intermédiaire t_n, en supposant les valeurs futures $V_{t_{n+1}}^{(i)}$ connues, l'algorithme procède selon les étapes suivantes :

- les cashflows futurs sont actualisés afin de construire la variable cible de la régression :

$$Y^{(i)} = e^{-r\Delta t_n} V_{t_{n+1}}^{(i)};$$

- seules les trajectoires *in-the-money*[2] à la date t_n sont retenues pour la régression ;
- une régression par moindres carrés est effectuée afin d'approximer la valeur de continuation $C(t_n, S_{t_n})$ sous la forme

$$\widehat{C}(t_n, S_{t_n}) = \sum_k \beta_k \psi_k(S_{t_n});$$

- pour chaque trajectoire, la valeur d'exercice immédiat $\Phi(S_{t_n}^{(i)})$ est comparée à la valeur de continuation estimée ;
- la décision d'exercice ou de continuation est prise, ce qui permet de mettre à jour la valeur $V_{t_n}^{(i)}$.

Cette procédure est répétée récursivement pour toutes les dates $t_n = t_{N-1}, \dots, t_1$. À l'issue de cette backward induction, les cashflows obtenus correspondent aux flux effectivement réalisés selon la stratégie d'exercice optimale estimée.

Le prix de l'option américaine est alors obtenu en calculant la moyenne des cashflows actualisés à la date initiale :

$$V_0 = \frac{1}{N_{\text{paths}}} \sum_{i=1}^{N_{\text{paths}}} e^{-rt_1} V_{t_1}^{(i)}.$$

L'algorithme 1 résume de manière synthétique les différentes étapes de la méthode de Longstaff–Schwartz telle qu'elle est implémentée dans ce travail.

[1] Une méthode de résolution caractéristique de la programmation dynamique, consistant à résoudre le problème à la maturité puis à remonter le temps pas à pas jusqu'à la date initiale.

[2] Une option est dite *in-the-money* (dans la monnaie) si son exercice immédiat génèrerait un flux positif. Pour un Put, cela correspond à $S < K$.

Algorithm 1 Algorithme de Longstaff–Schwartz (LSMC)

Simuler N_{paths} trajectoires du sous-jacent $(S_{t_n}^{(i)})_{n=0,\dots,N}$.
Initialiser les valeurs à maturité : $V_{t_N}^{(i)} = \Phi(S_{t_N}^{(i)})$.
for $n = N-1$ **down to** 1 **do**
  Identifier les trajectoires *in-the-money*.
  Calculer les variables cibles : $Y^{(i)} = e^{-r\Delta t_n} V_{t_{n+1}}^{(i)}$.
  Estimer la valeur de continuation par régression : $\widehat{C}(t_n, S_{t_n})$.
  for chaque trajectoire i **do**
    if $\Phi(S_{t_n}^{(i)}) \geq \widehat{C}(t_n, S_{t_n}^{(i)})$ **then**
      $V_{t_n}^{(i)} \leftarrow \Phi(S_{t_n}^{(i)})$
    else
      $V_{t_n}^{(i)} \leftarrow Y^{(i)}$
    end if
  end for
end for
Calculer le prix : $V_0 = \frac{1}{N_{\text{paths}}} \sum_i e^{-rt_1} V_{t_1}^{(i)}$.

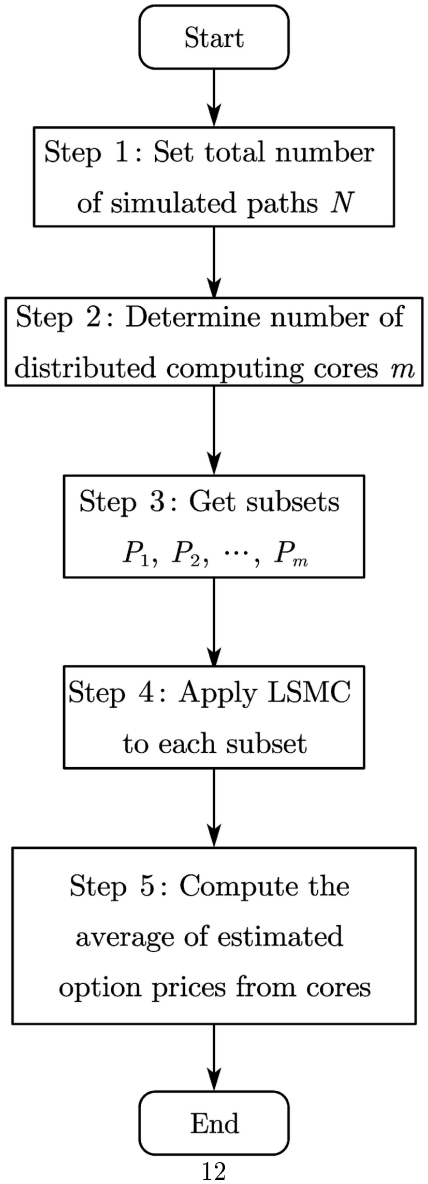


Figure 2: Diagramme de flux de l'algorithme LSMC parallélisé.

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.

Les réductions OpenMP sont utilisées pour agréger efficacement les contributions des différentes trajectoires, tout en garantissant l'absence de conditions de course[1]. 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 cœurs 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 coalescents[2] et de limiter les transferts entre l'hôte (CPU) et le périphérique (GPU).

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 vanilles 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.
- **Euler Implicite** : Schéma inconditionnellement stable, nécessitant la résolution d'un système linéaire tridiagonal à chaque pas de temps (algorithme de Thomas[3]).
- **Euler Explicite** : 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.

[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.

[3] 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)*contre*(n^3) pour l'élimination de Gauss générale.

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.

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 cœurs CUDA** et de **8 Go de mémoire GDDR6**, offrant une puissance de calcul théorique adaptée aux simulations massivement parallèles.

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 1: 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 (< 1ms) 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 2) 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
Chebyshev	CPU Séquentiel	100k	5.586	263.63	18.9 M
Chebyshev	CPU OpenMP	100k	5.586	265.64	18.8 M
Chebyshev	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
Chebyshev	CPU Séquentiel	1M	5.565	2633.49	18.9 M
Chebyshev	CPU OpenMP	1M	5.565	2638.32	18.9 M
Chebyshev	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 2: 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.

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.

4.6 Accélération par parallélisation CPU avec OpenMP

La version OpenMP exploite le parallélisme multi-cœurs 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.

Le facteur d'accélération obtenu est inférieur au nombre de cœurs 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 relativement simple et portable.

4.7 Accélération GPU avec CUDA

L'implémentation CUDA montre des gains de performance particulièrement importants pour les parties massivement parallèles de l'algorithme :

- simulation des trajectoires GBM ;
- calcul des payoffs ;
- certaines réductions statistiques.

Le GPU tire parti du parallélisme massif en assignant un thread par trajectoire, ce qui permet de traiter simultanément plusieurs centaines de milliers, voire millions de chemins.

Cependant, 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 finales dépendent fortement :

- 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.

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 cœurs. 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.

5 Difficultés rencontrées

La réalisation de ce projet a mis en évidence plusieurs difficultés, principalement liées à la nature algorithmique du LSMC et à son implémentation parallèle sur différentes architectures de calcul.

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** (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** (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.

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 $A^T A$).

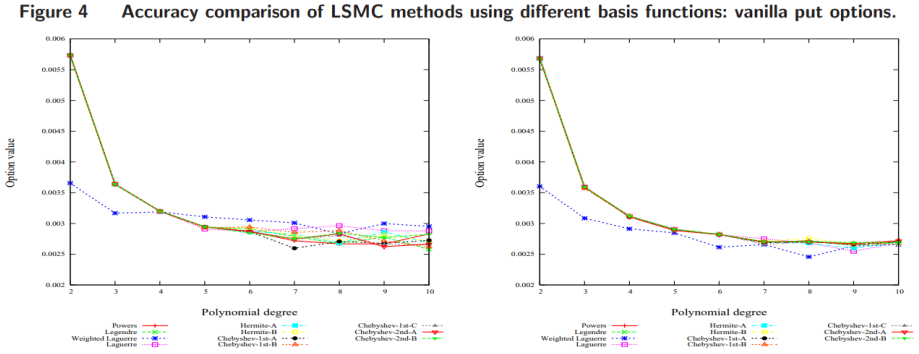
Extensions et bases avancées. Afin d'affiner la capture de la valeur de continuation, nous avons étendu l'implémentation à trois autres familles de bases, souvent citées dans la littérature spécialisée [2] :

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 3 illustre l'impact du choix de la base de régression sur la précision de l'estimation.



(a) LFIT with CONT-CE (b) SVD with CONT-LS

Figure 3: Comparaison de la précision selon les fonctions de base utilisées pour la régression LSMC (Source: [9]).

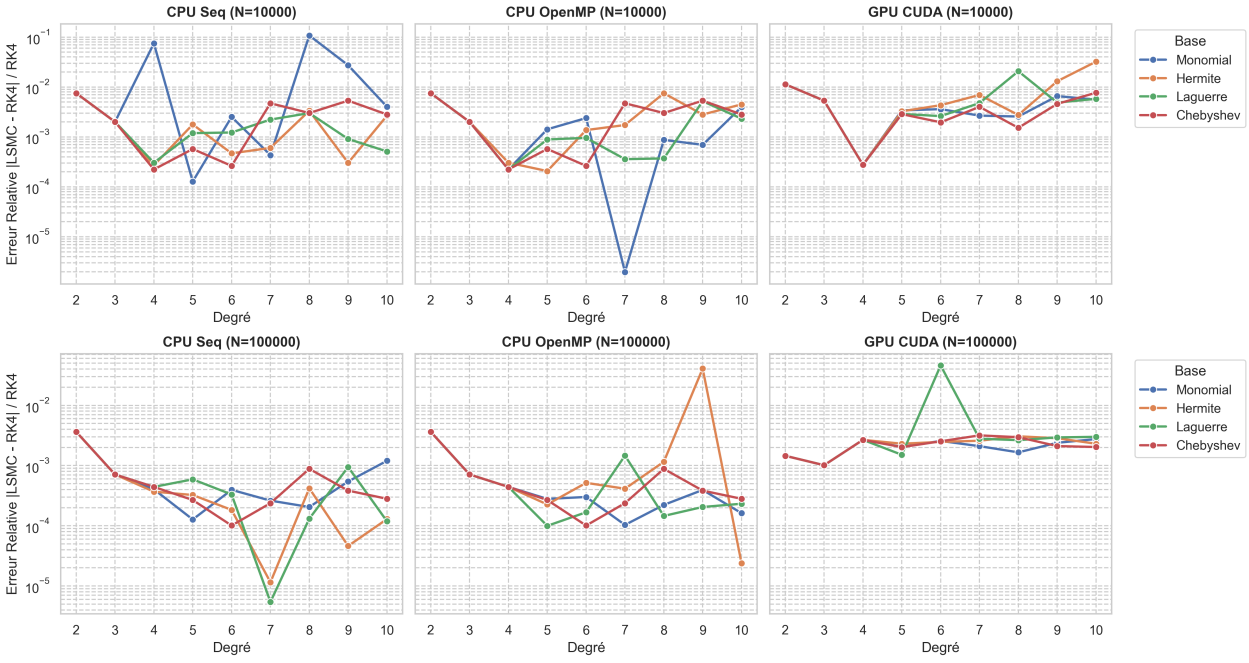


Figure 4: Comparaison de la convergence de l'erreur relative selon le nombre de trajectoires ($N = 10^4, 10^5$) et l'architecture.

Analyse de la convergence (Figure 4). Les résultats présentés en figure 4 montrent l'évolution de l'erreur relative par rapport au prix RK4 de référence. Cette dynamique peut être mise en perspective avec la Figure 3 (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).

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 Limites de la parallélisation CPU

La parallélisation sur CPU via OpenMP permet d'accélérer efficacement la simulation des trajectoires et le calcul des payoffs. Toutefois, les gains observés restent limités par la bande passante mémoire et la contention sur les caches partagés. En pratique, l'accélération n'est pas linéaire avec le nombre de cœurs, ce qui confirme le caractère fortement *memory-bound* de l'algorithme LSMC sur CPU.

5.6 Spécificités et contraintes de l'implémentation GPU

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 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. En revanche, la backward induction introduit une dépendance temporelle forte entre les dates, ce qui limite la parallélisation complète et impose des synchronisations coûteuses.

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.

5.7 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 le choix des fonctions de base utilisées pour l'approximation de la valeur de continuation. Dans ce projet, une base polynomiale simple de degré deux a été retenue pour des raisons de simplicité et de stabilité numérique. Il serait possible d'explorer :

- des bases polynomiales de degré plus élevé ;
- des polynômes orthogonaux (Laguerre, Hermite), souvent utilisés dans la littérature ;
- des bases adaptatives dépendant de la distribution du sous-jacent, comme suggéré dans des publications récentes [6].

Ces choix peuvent améliorer la précision de l'estimation de la valeur de continuation, au prix d'un coût de calcul plus élevé et d'un risque accru de sur-apprentissage.

Par ailleurs, l'utilisation de techniques de réduction de variance (antithetic variates, control variates, stratified sampling) pourrait significativement 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. Il serait intéressant de confronter les performances du LSMC à :

- .

	Méthodes par arbre	Méthodes itératives	Méthodes basées sur des simulations de Monte Carlo	Méthodes par transformée de Fourier
Compréhension	++	+	+	-
Scalabilité	-	+	++	+
Consommation mémoire	-	+	++	+
Complexité calcul	-	+	+	+
Convergence	+	+	-	++

Figure 7 Tableau dressant les atouts et contraintes des différentes méthodes.

Figure 5: Tableau comparatif des méthodes de pricing : arbres, itératives, Monte Carlo et transformée de Fourier [9].

On pourrait ainsi comparer ;

- des approches par équations aux dérivées partielles résolues numériquement ;
- des méthodes récentes basées sur l'apprentissage automatique (réseaux de neurones pour l'approximation de la valeur de continuation).

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.

7.3 Répartition des tâches et binôme

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

- **Florian Barbe** : Responsable du "cœur de calcul" (*Core Engine*).
  - 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.
- **Narjisse** : Responsable de l'expérience utilisateur et de l'interface (*Frontend*).
  - Conception de l'interface graphique (GUI) pour rendre l'outil accessible.
  - Développement d'une application interactive (via Streamlit/Python) permettant de modifier les paramètres (S_0, K, r, σ) et de lancer les simulations sans toucher au code C++.
  - Visualisation des résultats (graphiques de convergence, trajectoires) et intégration avec l'exécutable C++.

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.

7.4 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.

7.5 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 P1RV avait pour objectif principal l'étude et l'implémentation de l'algorithme de Longstaff–Schwartz pour le pricing d'options américaines, ainsi que l'analyse de ses performances selon différentes architectures de calcul. À travers ce travail, nous avons pu aborder à la fois les aspects théoriques de la finance quantitative et les problématiques concrètes liées au calcul numérique intensif.

Sur le plan théorique, le projet a permis de mettre en évidence la spécificité des options américaines par rapport aux options européennes, en particulier la problématique de l'exercice anticipé et le caractère de décision dynamique qui en découle. L'algorithme LSMC apparaît comme une méthode particulièrement adaptée pour traiter ce type de produits dérivés, en combinant la flexibilité des simulations de Monte Carlo et l'estimation statistique de la valeur de continuation par régression.

D'un point de vue algorithmique et logiciel, une implémentation complète de la méthode a été réalisée en `C++`. La version séquentielle a servi de base de validation fonctionnelle, tandis que la parallélisation sur CPU via OpenMP a permis d'exploiter efficacement le parallélisme multi-cœurs. Cette approche apporte des gains de performance significatifs, bien que limités par la bande passante mémoire et la nature statistique de l'algorithme.

L'implémentation GPU à l'aide de CUDA a mis en évidence le fort potentiel d'accélération des méthodes de Monte Carlo lorsque celles-ci sont déployées sur des architectures massivement parallèles. Les gains les plus importants sont observés lors de la simulation des trajectoires et du calcul des payoffs, tandis que la phase de backward induction reste partiellement séquentielle en raison de dépendances temporelles intrinsèques à l'algorithme LSMC.

De manière générale, ce projet illustre clairement l'importance du choix de l'architecture de calcul pour les algorithmes numériques intensifs. Il montre également que les performances ne dépendent pas uniquement

de la puissance brute du matériel, mais aussi de l'adéquation entre la structure de l'algorithme et le modèle de parallélisme sous-jacent.

Enfin, ce travail a constitué une introduction approfondie aux problématiques de calcul haute performance appliquées à la finance quantitative. Il ouvre naturellement la voie à des améliorations futures, telles que l'utilisation de bases de régression plus riches, l'optimisation avancée des réductions sur GPU ou encore l'extension à des modèles de dynamique du sous-jacent plus complexes. Au-delà des résultats obtenus, ce projet a permis de consolider des compétences transversales en modélisation mathématique, programmation parallèle et analyse de performance, essentielles dans de nombreux domaines de l'ingénierie et de la recherche.

A Rsolution de l'EDS du GBM du mouvement brownien géométrique sous la mesure risque-neutre

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

$$\mathrm{d}S_t = rS_t\,\mathrm{d}t + \sigma S_t\,\mathrm{d}W_t^{\mathbb{Q}},$$

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

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

En utilisant les règles du calcul stochastique

$$(\mathrm{d}W_t^{\mathbb{Q}})^2 = \mathrm{d}t, \qquad \mathrm{d}t\,\mathrm{d}W_t^{\mathbb{Q}} = 0, \qquad (\mathrm{d}t)^2 = 0,$$

on a

$$\mathrm{d}S_t = rS_t\,\mathrm{d}t + \sigma S_t\,\mathrm{d}W_t^{\mathbb{Q}} \quad \Longrightarrow \quad (\mathrm{d}S_t)^2 = \sigma^2 S_t^2\,(\mathrm{d}W_t^{\mathbb{Q}})^2 = \sigma^2 S_t^2\,\mathrm{d}t.$$

En substituant dans la formule d'Itô, il vient

$$\mathrm{d}\ln(S_t) = \frac{1}{S_t}\Big(rS_t\,\mathrm{d}t + \sigma S_t\,\mathrm{d}W_t^{\mathbb{Q}}\Big) - \frac{1}{2S_t^2}\Big(\sigma^2 S_t^2\,\mathrm{d}t\Big),$$

soit

$$\mathrm{d}\ln(S_t) = \left(r - \frac{\sigma^2}{2}\right)\mathrm{d}t + \sigma\,\mathrm{d}W_t^{\mathbb{Q}}.$$

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

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

Les incréments 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 Détails mathématiques de l'algorithme LSMC

B.1 Valeur de continuation

À une date t_n, la valeur de continuation est définie par :

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

Cette quantité n'admet pas de forme fermée et doit être estimée numériquement.

B.2 Approximation par régression

Dans la méthode de Longstaff–Schwartz, on approxime la valeur de continuation par une combinaison linéaire de fonctions de base :

$$C(t_n, S_{t_n}) \approx \sum_{k=0}^{K} \beta_k \psi_k(S_{t_n}).$$

Dans ce travail, les fonctions de base choisies sont :

$$\psi(S) = (1,\, S,\, S^2).$$

Les coefficients β_k sont déterminés par une régression aux moindres carrés sur les trajectoires *in-the-money*.

B.3 Critère d'exercice

La règle d'exercice est :

$$\text{Exercer si } \Phi(S_{t_n}) > \widehat{C}(t_n, S_{t_n}),$$

sinon l'option est conservée.

C Pseudo-code de l'algorithme LSMC

1. Simuler N trajectoires du sous-jacent sur $[0, T]$.
2. Initialiser les cashflows à maturité par le payoff.
3. Pour $t = T - \Delta t$ jusqu'à $t = \Delta t$:
   (a) Sélectionner les trajectoires in-the-money.
   (b) Estimer la valeur de continuation par régression.
   (c) Comparer exercice immédiat et continuation.
   (d) Mettre à jour les cashflows.
4. Actualiser et moyenner les cashflows.

D Extrait de kernel CUDA (illustratif)

E Analyse de scalabilité OpenMP

Les figures suivantes illustrent la scalabilité de l'implémentation OpenMP en fonction du nombre de trajectoires et de threads.

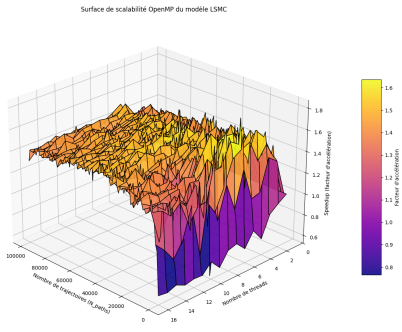


Figure 6: Surface de scalabilité OpenMP : temps d'exécution en fonction du nombre de trajectoires et de threads.

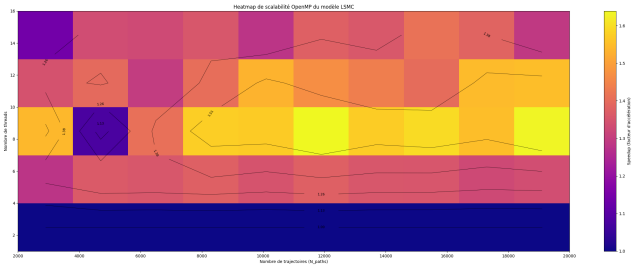


Figure 7: Heatmap des performances OpenMP.

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.