



PROJET DE MONTE-CARLO

---

## Pricing d'options asiatiques avec méthodes de Monte-Carlo

---

*Thomas AHN, Tristan AMADEI, Thibaut METZ, Thomas SALOMON*

Avril 2023

# Table des matières

|  |           |
|--|-----------|
| <b>Introduction</b>  | <b>1</b>  |
| <b>1 Pricing d'options asiatiques</b>  | <b>3</b>  |
| 1.1 Prérequis . . . . .  | 3         |
| 1.2 Simulations de Monte-Carlo pour le pricing . . . . .                     | 3         |
| <b>2 Simulation du prix du sous-jacent</b>                                   | <b>4</b>  |
| 2.1 Simulation d'un processus Cox-Ingersoll-Ross (CIR) . . . . .             | 4         |
| 2.2 Simulation des incrémentations browniennes . . . . .                     | 4         |
| 2.2.1 Méthode Monte-Carlo standard . . . . .                                 | 5         |
| 2.2.2 Méthode Randomized Quasi Monte-Carlo . . . . .                         | 5         |
| <b>3 Approximation du prix de l'option et calcul de la Mean Square Error</b> | <b>8</b>  |
| 3.1 Méthode Monte-Carlo Standard . . . . .                                   | 8         |
| 3.2 Méthode Multi-Level Monte-Carlo . . . . .                                | 8         |
| 3.2.1 Présentation de la méthode . . . . .                                   | 8         |
| 3.2.2 Choix des paramètres de MLMC . . . . .                                 | 9         |
| 3.3 Méthode Randomized Quasi Monte-Carlo . . . . .                           | 10        |
| 3.4 Calcul de la Mean Square Error . . . . .                                 | 11        |
| <b>4 Comparaison des approximations du prix en termes de MSE et CPU time</b> | <b>12</b> |
| 4.1 Multiprocessing . . . . .  | 12        |
| 4.2 Comparaison entre MC Standard et MLMC . . . . .                          | 12        |
| 4.2.1 MLMC avec paramètres non optimisés . . . . .                           | 12        |
| 4.2.2 MLMC avec paramètres optimisés . . . . .                               | 14        |
| 4.3 Comparaison entre RQMC et QMC MLMC . . . . .                             | 14        |
| 4.3.1 QMC MLMC avec paramètres non optimisés . . . . .                       | 15        |
| 4.3.2 QMC MLMC avec paramètres optimisés . . . . .                           | 16        |
| 4.4 Évolution de la MSE et de la CPU time . . . . .                          | 16        |
| 4.4.1 Variation de la MSE avec le nombre de simulations total . . . . .      | 16        |
| 4.4.2 Variation de la MSE avec $\epsilon$ . . . . .                          | 17        |
| <b>5 Utilisation d'un pont brownien</b>                                      | <b>18</b> |
| 5.1 Séquence de Van der Corput . . . . .                                     | 18        |
| 5.2 La méthode du pont brownien . . . . .                                    | 18        |
| <b>6 Conclusion</b>  | <b>20</b> |
| <b>7 Annexes</b>   | <b>21</b> |
| 7.1 Démonstration $N_l \propto \sqrt{V_l h_l}$ . . . . .                     | 21        |
| 7.2 Évolution des $N_l$ . . . . .  | 21        |
| 7.2.1 Multi-Level Monte-Carlo . . . . .                                      | 21        |
| 7.2.2 Multi-Level Quasi Monte-Carlo . . . . .                                | 23        |

# Introduction

L'objectif de ce projet est de comparer différentes méthodes de pricing d'une option asiatique. Les comparaisons seront effectuées en termes de MSE, de CPU time et de variance (empirique) des estimateurs. Les méthodes de pricing comparées dans ce projet sont :

- Monte-Carlo standard (MC)
- Multi-Level Monte-Carlo (MLMC)
- Randomized Quasi Monte-Carlo (RQMC)
- Multi-Level Randomized Quasi Monte-Carlo (QMC MLMC)

Nous tenons à informer le lecteur que l'exécution du notebook fourni avec ce rapport pourrait prendre du temps, étant donné que le code a été exécuté sur un environnement différent afin de bénéficier d'une puissance de calcul accrue.

# 1 Pricing d'options asiatiques

## 1.1 Prérequis

Une option asiatique est un type d'option dont le payoff dépend du prix moyen d'un actif sous-jacent sur une période donnée, plutôt que du prix de l'actif à une date spécifique. Le prix moyen est souvent calculé sur une période d'une semaine, d'un mois ou d'un trimestre.

Le payoff d'une option asiatique est généralement exprimé sous forme d'une moyenne arithmétique des prix de l'actif sous-jacent sur la période de référence, moins le prix d'exercice de l'option. Si le prix moyen du sous-jacent est supérieur au prix d'exercice, le détenteur de l'option reçoit un paiement qui correspond à la différence entre le prix moyen et le prix d'exercice. Si le prix moyen du sous-jacent est inférieur au prix d'exercice, l'option expire sans valeur. Le prix d'une option asiatique peut être calculé de différentes manières, en fonction des termes et conditions de l'option définissant le payoff. Dans ce projet, nous considérons une option asiatique d'achat (call asiatique) dont la forme du prix adopté est la suivante :

$$C = \mathbb{E} \left[ e^{-rT} \cdot \max \left( \frac{1}{k} \sum_{i=1}^k S(t_i) - K, 0 \right) \right]$$

avec :  
-  $t_0 = 0 < t_1 < \dots < t_k = T$   
-  $r > 0$   
-  $K > 0$  fixé  
-  $S$  : un processus défini sur  $[0, T]$

## 1.2 Simulations de Monte-Carlo pour le pricing

Le prix d'une option asiatique peut être évalué en utilisant des simulations de Monte-Carlo. Cette méthode implique de simuler plusieurs trajectoires du prix de l'actif sous-jacent. Pour chaque trajectoire simulée, une estimation du prix de l'option est réalisée, et le prix final est obtenu en prenant la moyenne des estimations. Pour générer les trajectoires du sous-jacent, il est possible d'utiliser plusieurs modèles. Dans ce projet, le processus adopté pour la trajectoire du sous-jacent est le processus Cox-Ingersoll-Ross(CIR).

## 2 Simulation du prix du sous-jacent

### 2.1 Simulation d'un processus Cox-Ingersoll-Ross (CIR)

Le processus CIR est un processus stochastique défini par l'équation différentielle stochastique suivante :  $dS_t = a(b - S_t)dt + \sigma\sqrt{S_t}dW_t$  avec  $W_t$  un mouvement brownien.

En utilisant la méthode d'Euler-Maruyama permettant d'obtenir une solution numérique d'une équation différentielle stochastique :

Pour  $i \geq 0$ ,  $S_{t_{i+1}} - S_{t_i} = a(b - S_{t_i})(t_{i+1} - t_i) + \sigma\sqrt{S_{t_i}}(W_{t_{i+1}} - W_{t_i})$  avec  $t_i = i\frac{T}{n}$

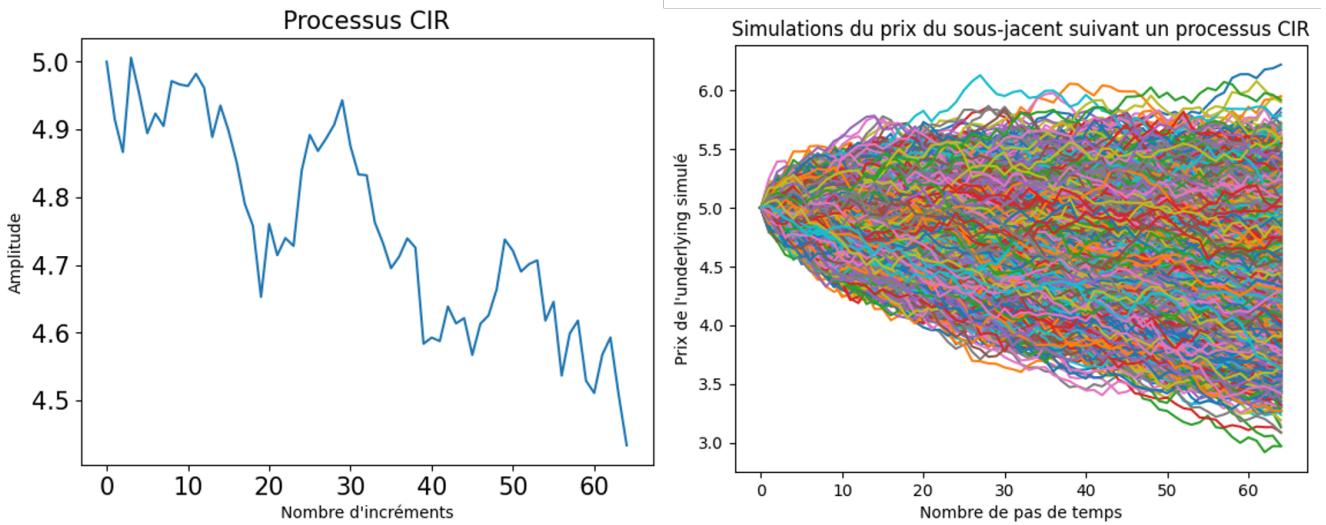


FIGURE 2.1 – Processus CIR et simulations du prix du sous-jacent suivant un CIR

Les fonctions `sim_CIR` et `sim_CIR_QMC` disponibles dans nos notebooks simulent un processus CIR. Les deux diffèrent car la simulation des incrément browniens n'est pas effectuée de la même manière, comme expliqué dans le paragraphe ci-dessous.

### 2.2 Simulation des incrément browniens

La simulation des incrément d'un mouvement brownien consiste à générer un mouvement aléatoire pour le prix du sous-jacent et à répéter cette simulation de nombreuses fois. On définit tout d'abord un pas de temps discret. Considérons le pas de temps  $\frac{T}{n}$ . À chaque étape de temps, si le prix du sous-jacent suit un mouvement brownien, alors il se déplace selon une distance de déplacement aléatoire générée, qui est calculée en utilisant une distribution de probabilité normale centrée sur zéro et de variance  $\frac{T}{n}$ .

Ainsi pour simuler  $N$  incrément d'un mouvement brownien avec la méthode de Monte-Carlo standard, il suffit de simuler  $N$  fois une  $\mathcal{N}(0, 1)$  et de multiplier chaque simulation par  $\sqrt{\frac{T}{n}}$  car :  $W_{t_{i+1}} - W_{t_i} \sim \mathcal{N}(0, \frac{T}{n})$

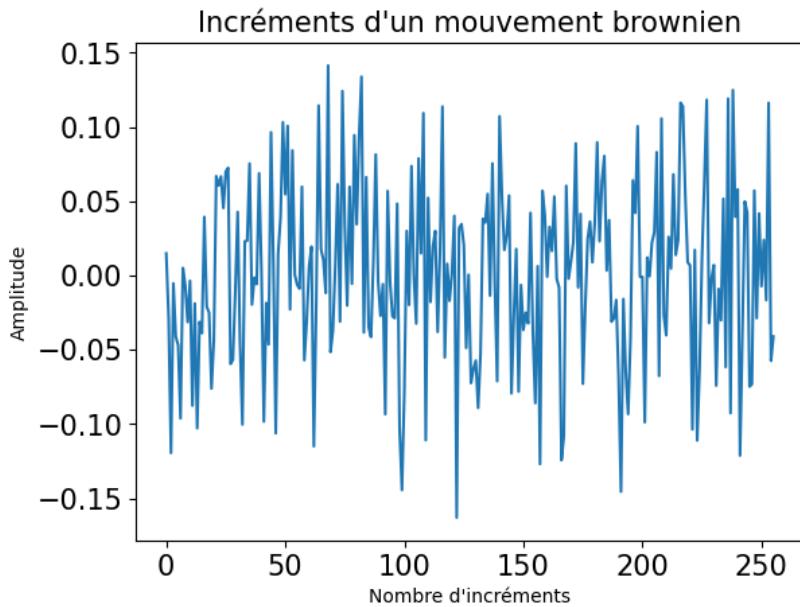


FIGURE 2.2 – Incréments d’un mouvement Brownien

Dans ce projet, nous avons simulé les incréments browniens de deux manières différentes. La première méthode consiste à utiliser des simulations de Monte-Carlo standard d’une  $\mathcal{N}(0, 1)$  alors que la deuxième utilise quant à elle du Randomized Quasi Monte-Carlo (RQMC).

### 2.2.1 Méthode Monte-Carlo standard

Cette méthode consiste à tirer  $N$  échantillons d’une  $\mathcal{N}(0, 1)$  avec `numpy.random.normal` par exemple et de multiplier chaque échantillon par  $\sqrt{\frac{T}{n}}$ . C’est ce que nous avons implémenté avec la fonction nommée `sim_dW`.

### 2.2.2 Méthode Randomized Quasi Monte-Carlo

La méthode Quasi Monte-Carlo est similaire à la méthode de Monte-Carlo standard, mais au lieu de générer des échantillons aléatoires, elle utilise des suites de points déterministes (une suite de Sobol dans notre cas) pour échantillonner l’espace de recherche. Le but de cette technique est de mieux recouvrir l’intervalle  $[0, 1]$  lors des simulations, et ainsi faire diminuer les variances des estimateurs.

On peut ensuite rendre cette suite de points aléatoire en utilisant la formule suivante :

$$\forall n \in \mathbb{N}, v_n = (u_n + w_n) \mod 1$$

avec  $(u_n)$  une suite d’uniformes simulées par méthode Quasi Monte-Carlo, une séquence Sobol par exemple, et  $w_n \sim \mathcal{U}(0, 1)$ . Cela permet d’ajouter un aspect aléatoire aux uniformes simulées.

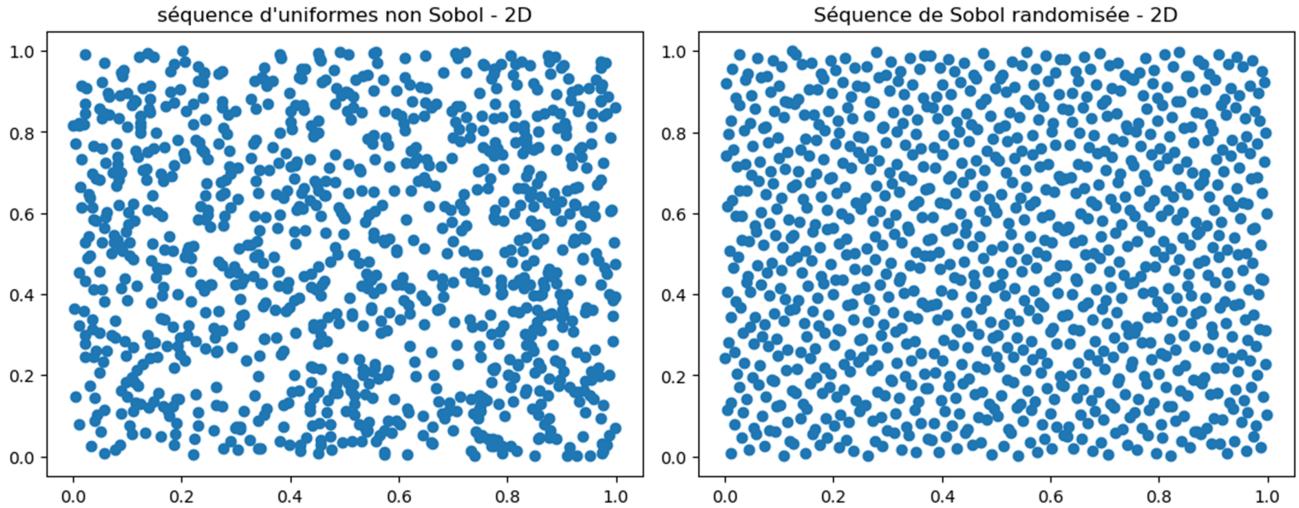


FIGURE 2.3 – Comparaison entre une séquence d'uniformes et une séquence de Sobol - 2D

Il existe plusieurs méthodes de Randomized Quasi Monte Carlo pour simuler des échantillons d'une  $\mathcal{N}(0, 1)$ . Dans notre projet, nous avons utilisé trois d'entre elles :

- la méthode acceptation rejet
- la méthode Box-Muller
- la méthode d'inversion

### Méthode acceptation-rejet

La méthode la plus intuitive (et la plus sûre mathématiquement) dans notre cas était l'acceptation-rejet. L'idée était d'utiliser la fonction densité de la loi de Laplace de moyenne nulle et de paramètre d'échelle 1, dont la formule est la suivante :

$$\forall x \in \mathbb{R}, g(x) = \frac{1}{2} \exp(-|x|)$$

On peut simuler des réalisations de cette loi en remarquant que  $X = \log(1 - 2|U|)$  suit notre loi de Laplace quand  $U \sim \mathcal{U}([- \frac{1}{2}, \frac{1}{2}])$ . On peut alors simuler des réalisations d'une loi  $\mathcal{N}(0, 1)$  de la manière suivante :

1. Simuler  $x$  suivant Laplace
2. Soit  $u_i$  une réalisation de  $\mathcal{U}([0, 1])$  simulée suivant une séquence Sobol randomisée
3. Si  $u_i \leq \frac{f(x)}{Mg(x)}$ , on accepte  $x$  comme simulation gaussienne
4. Sinon on refuse  $x$ , on considère  $i = i + 1$  et on recommence depuis 1

Dans cet algorithme,  $f$  est la densité de  $\mathcal{N}(0, 1)$  et  $M = \sqrt{\frac{2e}{\pi}}$  est la valeur optimale pour accepter les simulations. Le taux d'acceptation est  $\frac{1}{M} \approx 0.76$ . Pour simuler  $N$  réalisations normales, on doit alors établir une séquence de Sobol randomisée de  $N \times M$  échantillons uniformes. En pratique, on a établi une séquence de dimension  $1.1 \times N \times M$  pour être sûr d'en avoir assez. Cette méthode fonctionne parfaitement et permet bien de simuler les normales souhaitées, mais prend beaucoup de temps. En l'appliquant à notre algorithme de RQMC permettant de calculer le prix de l'option (fonction QMC\_call\_asian), on multiplie le temps d'exécution par  $10!$  Ainsi nous avons cherché d'autres méthodes pour réduire le temps de calcul.

## Méthode Box-Muller

La raison pour laquelle nous avons dans un premier temps écarté Box-Muller est le fait que cette méthode nécessite deux uniformes *indépendantes* pour simuler deux gaussiennes centrées réduites indépendantes. Mais du fait de choisir les uniformes dans une séquence Sobol, on perd l'indépendance mutuelle de ces réalisations.

Cependant, une autre méthode est de simuler une séquence de Sobol randomisée de  $N/2$  points en 2 dimensions. Ainsi, pour tout point  $u_i = \begin{pmatrix} u_{i_1} \\ u_{i_2} \end{pmatrix}$ , les points  $u_{i_1}$  et  $u_{i_2}$  sont indépendants. On peut alors utiliser la méthode de Box-Muller pour simuler nos gaussiennes :

$$\forall i \in \{1, \dots, N\}, \begin{cases} X = \sqrt{-2 \log(U_{i_1})} \cdot \cos(2\pi U_{i_2}) \\ Y = \sqrt{-2 \log(U_{i_2})} \cdot \cos(2\pi U_{i_1}) \end{cases}$$

Nous utilisons cette version de Box-Muller, et non la version améliorée, car nous avons besoin de connaître la taille de la séquence de Sobol randomisée à simuler au préalable, et donc nous souhaitons éviter les possibles rejets de la seconde méthode de Box-Muller.

## Méthode d'inversion

Si la fonction quantile d'une loi de probabilité  $X$  est connue, il est possible de simuler des réalisations de cette loi de la manière suivante :

$$X = F^{-1}(U), U \sim \mathcal{U}([0, 1])$$

Dans notre cas, la fonction quantile d'une loi gaussienne n'a pas d'expression analytique. Cependant, il est possible de l'approximer, comme cela a été fait dans le module *scipy.stats* sur Python. Cette méthode est moins fondée mathématiquement que les deux précédentes, du fait de cette approximation de la fonction quantile, mais elle est très rapide.

## Comparaison des différentes méthodes

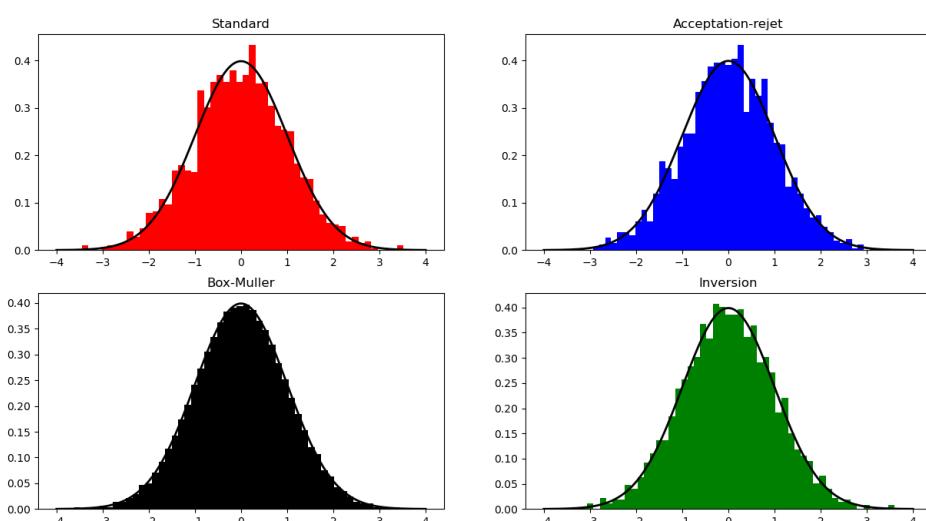


FIGURE 2.4 – Comparaison des méthodes Quasi Monte-Carlo pour les simulations de lois normales ; avec 1000 simulations

Dans la suite de notre étude, nous utilisons la méthode Box-Muller pour implémenter Quasi Monte-Carlo (RQMC) et Multi-Level Quasi Monte-Carlo (QMC MLMC).

# 3 Approximation du prix de l'option et calcul de la Mean Square Error

## 3.1 Méthode Monte-Carlo Standard

Cette méthode implique la simulation de plusieurs processus CIR afin de représenter différentes trajectoires du prix du sous-jacent. Les incrément du mouvement brownien nécessaires à cette simulation sont obtenus par méthode de Monte-Carlo standard, comme expliqué précédemment. Le nombre de simulations de trajectoires du sous-jacent et le pas adoptés seront explicités plus loin.

## 3.2 Méthode Multi-Level Monte-Carlo

### 3.2.1 Présentation de la méthode

La méthode Multi-Level Monte-Carlo (MLMC) est basée sur l'idée que l'on peut obtenir une estimation plus précise d'une quantité d'intérêt en utilisant des simulations à plusieurs niveaux de granularité. Dans le cas de l'option asiatique, la quantité d'intérêt est le prix de l'option, qui dépend du prix moyen de l'actif sous-jacent sur une certaine période.

Pour estimer le prix de l'option asiatique, la méthode MLMC utilise deux niveaux de granularité, déterminés par les valeurs des paramètres  $M$  et  $l$  dans le code. Le niveau de granularité « coarse » correspond à une période de temps plus longue, où l'actif sous-jacent est simulé avec une discrétisation plus grossière (c'est-à-dire avec un pas de temps plus grand). Le niveau de granularité « fine » correspond à une période de temps plus courte, où l'actif sous-jacent est simulé avec une discrétisation plus fine (c'est-à-dire avec un pas de temps plus petit).

Pour chaque niveau de granularité, la méthode MLMC effectue un grand nombre de simulations pour estimer le prix moyen de l'actif sous-jacent sur la période correspondante. Ensuite, elle utilise ces estimations pour calculer une estimation du prix de l'option asiatique. Plus précisément, l'estimation du prix de l'option est calculée comme la différence entre le prix moyen estimé à partir des simulations de niveau « fine » et le prix moyen estimé à partir des simulations de niveau « coarse ».

Soit  $C$  le prix de l'option et  $P_l$  son approximation au niveau  $l$  c'est à dire son approximation avec une discrétisation de pas  $h_l = \frac{1}{M^l}$ . Par somme telescopique :

$$\mathbb{E}[P_L] = \mathbb{E}[P_0] + \sum_{l=1}^L \mathbb{E}[P_l - P_{l-1}]$$

L'idée derrière MLMC est d'estimer indépendamment chacune des espérances du côté droit d'une manière qui minimise la variance globale pour un coût de calcul donné.

Soit  $\hat{Y}_0$  un estimateur de  $\mathbb{E}[P_0]$  utilisant  $N_0$  échantillons et  $\hat{Y}_l$  pour  $l > 0$  un estimateur de  $\mathbb{E}[P_l - P_{l-1}]$  utilisant  $N_l$  échantillons.

$$\hat{Y}_l = \frac{1}{N_l} \sum_{i=1}^{N_l} (P_l^i - P_{l-1}^i)$$

Soit  $\hat{Y}$  l'estimateur MLMC du prix de l'option.

$$\hat{Y} = \sum_{l=0}^L \hat{Y}_l = \sum_{l=0}^L \frac{1}{N_l} \sum_{i=1}^{N_l} (P_l^i - P_{l-1}^i)$$

$$\mathbb{V}[\hat{Y}] = \sum_{l=0}^L \mathbb{V}[\hat{Y}_l] \text{ par indépendance avec : } \mathbb{V}[\hat{Y}_l] = \frac{1}{N_l} V_l \text{ où } V_l = \mathbb{V}[P_l - P_{l-1}]$$

Si les  $N_l$ ,  $l \in \{0, \dots, L\}$ , sont calculés de manière optimale avec la formule (2) (évoquée dans la partie suivante), alors on peut montrer que pour un terme d'erreur  $\epsilon$  fixé :

$$\mathbb{V}[\hat{Y}] = \sum_{l=0}^L \mathbb{V}[\hat{Y}_l] \leq \frac{1}{2} \epsilon^{-2}$$

La méthode MLMC permet donc de faire diminuer la variance et permet de la fixer arbitrairement faible, en faisant tendre  $\epsilon$  vers 0. Cependant, cela nous place face à un arbitrage car le temps de calcul du MLMC augmente de manière quadratique lorsqu' $\epsilon$  diminue.

Malgré cela, on peut facilement prouver que l'estimateur MLMC reste sans biais. En effet :

$$\begin{aligned} \mathbb{E}[\hat{Y}] &= \sum_{l=1}^L \mathbb{E}[\hat{Y}_l] \\ &= \sum_{l=1}^L \mathbb{E}[P_l - P_{l-1}] \\ &= \mathbb{E}[P_L] - \mathbb{E}[P_0] \\ &= \mathbb{E}[P_L] \end{aligned}$$

Ainsi, l'estimateur MLMC est *sans biais*.

### 3.2.2 Choix des paramètres de MLMC

#### Paramètre $M$

Dans ce projet, nous avons choisi de fixer  $M$  égal à 4 dans la mesure où Michael B. Giles choisit dans son article cette valeur pour ses applications numériques. En effet, cette valeur de  $M$  donne la plupart des avantages d'une valeur plus élevée, mais en même temps  $M$  est suffisamment petit pour donner un nombre raisonnable de niveaux à partir desquels estimer le biais.

#### Paramètre $L$

Le paramètre  $L$  a été choisi de telle façon à prendre le plus petit  $L$  possible (pour diminuer la CPU time) tel que l'estimateur  $\hat{Y}_L$  soit convergent. Le critère utilisé (biais suffisamment petit) est le suivant :

$$\left| \hat{Y}_L - \frac{\hat{Y}_{L-1}}{M} \right| < \frac{1}{\sqrt{2}} (M^2 - 1) \epsilon \quad (1)$$

avec  $\epsilon$  le terme d'erreur choisi

## Paramètre $N_l$

La sélection de ce paramètre est cruciale car un choix optimal de  $N_l$  permet de réduire la variance.

Comme cela est expliqué dans l'article et dans le cours, il existe une méthode qui consiste simplement à choisir pour chaque valeur de  $l$ ,  $N_l$  proportionnel à  $\sqrt{V_l h_l} = \sqrt{\frac{V_l}{M^l}}$ . La démonstration de ce choix est réalisée en annexes.

Dans son article, Michael B. Giles propose une expression précise de  $N_l$  qui est également proportionnelle à  $\sqrt{V_l h_l}$ , mais qui fournit en plus l'expression du coefficient de proportionnalité. Nous avons donc choisi d'utiliser la formule suivante proposée par Michael B. Giles :

$$N_l = 2\epsilon^{-2} \sqrt{V_l \cdot h_l} \left( \sum_{l=0}^L \sqrt{\frac{V_l}{h_l}} \right) \quad (2)$$

Dans ce projet, les paramètres  $L$  et  $N_l$  ont été choisis optimalement à l'aide des fonctions `mlmc_adaptive` et `qmc_mlmc_adaptive` qui implémentent l'algorithme ci-dessous disponible dans l'article :

1. Fixer  $L = 0$
2. Estimer  $V_L$  avec  $N_L = 10^4$
3. Pour  $l \in [0, L]$  calculer le  $N_l$  optimal en utilisant l'équation (2)
4. Pour chaque  $N_l$  :
  - Si  $N_l^{nouveau} \geq N_l^{ancien}$  : simuler un nombre de nouvelles trajectoires du sous-jacent égal à  $N_l^{nouveau} - N_l^{ancien}$
  - Si  $N_l^{nouveau} < N_l^{ancien}$  : supprimer un nombre de trajectoires du sous-jacent égal à  $N_l^{ancien} - N_l^{nouveau}$
5. • Si  $L \geq 2$  : Tester la convergence en utilisant l'équation (1)
  - Si  $L < 2$  ou s'il n'y a pas convergence :  $L=L+1$  et aller à l'étape 2

## 3.3 Méthode Randomized Quasi Monte-Carlo

Cette méthode implique, de la même manière que la méthode Monte-Carlo standard, la simulation de plusieurs processus CIR afin de représenter différentes trajectoires du prix du sous-jacent. En revanche, les incrémentations du mouvement brownien nécessaires à cette simulation sont obtenus par méthode Randomized Quasi Monte Carlo utilisant l'acceptation-rejet, Box-Muller ou bien l'inversion. Comme expliqué précédemment, dans notre étude, nous avons choisi d'implémenter la méthode Box-Muller combinée à une suite de Sobol en 2D.

### 3.4 Calcul de la Mean Square Error

La Mean Square Error (MSE) est définie par :  $MSE = \sum_{i=1}^n (C - \hat{Y}_i)^2$  avec :

- $C$  la "vraie valeur" de l'option
- $\hat{Y}_i$  l'approximation du prix de l'option n°i
- n le nombre d'approximations du prix de l'option réalisées

Pour une option asiatique dont la date d'échéance n'est pas atteinte, il est impossible de connaître sa "vraie valeur" dans la mesure où le prix de l'option dépend de la trajectoire du prix du sous-jacent. Or cette trajectoire du prix du sous-jacent ne peut pas être connue avec certitude dans le futur. Ainsi pour calculer la MSE, il faut connaître cette "vraie valeur". Pour cela nous avons réalisé un Monte-Carlo standard avec un nombre de simulations très grand. En se fondant sur la loi des grands nombres, on peut considérer que l'approximation ainsi obtenue est la vraie valeur visée C. On utilise ensuite cette approximation calculée sur un très grand échantillon comme la vraie valeur pour estimer la MSE des différentes méthodes.

Ainsi nous avons fait tourner la fonction MC\_call\_asian pendant 6 heures qui a permis de réaliser  $10^6$  simulations de Monte-Carlo standard pour obtenir une valeur de  $C = 0.6797012714083033$ .

# 4 Comparaison des approximations du prix en termes de MSE et CPU time

Comme présenté dans l'article, le pas de discréétisation adopté pour la méthode Monte-Carlo standard correspond à celui le plus fin de la méthode Multi-Level Monte Carlo ( $l=L$ ) c'est à dire  $h_L = \frac{1}{M^L}$ . Pour effectuer nos comparaisons, le nombre de simulations  $N_{sim}$  du sous-jacent adopté pour la méthode Monte-Carlo standard est égal à la somme des simulations  $N_l$  pour  $l \in [0, L]$  de la méthode Multi-Level Monte-Carlo :  $N_{sim} = \sum_{i=1}^L N_l$

Les résultats sont obtenus avec les valeurs suivantes :  $K = 4, S_0 = 5, r = 0.05, \sigma = 0.2, a = 0.15, b = 1, M = 4$ .

## 4.1 Multiprocessing

Le calcul des différents estimateurs est coûteux, surtout lorsqu'il s'agit de réduire la tolérance d'erreur  $\epsilon$  ou d'augmenter le nombre de simulations nécessaires. Ainsi, pour nous permettre de tester plusieurs scenarii, nous avons introduit du multiprocessing - parallélisation en d'autres termes - dans notre code. Par exemple, si nous voulons calculer la variance des estimateurs de Monte-Carlo pour  $N_{sim}$  simulations, ce multiprocessing nous permet de calculer en parallèle  $N_{estimateurs}$  de Monte-Carlo avec le même nombre de simulations ; ce qui nous permet alors de calculer la variance empirique de ces estimateurs, la MSE etc.

## 4.2 Comparaison entre MC Standard et MLMC

Dans cette partie, nous effectuons des comparaisons entre la méthode Monte Carlo standard (MC) et Multi-Level Monte Carlo (MLMC) dont les incrément du mouvement brownien sont simulés par méthode Monte Carlo standard comme expliqué en 2.2.1.

### 4.2.1 MLMC avec paramètres non optimisés

Pour comparer ces deux méthodes, nous avons fait varier le nombre  $N_{sim}$  de simulations réalisées pour construire notre estimateur de Monte-Carlo. Pour l'estimateur de MLMC, nous avons repris ce nombre  $N_{sim}$  que nous avons divisé de manière uniforme pour construire chaque niveau, avec un niveau final  $L = 3$ .

En examinant les figures ci-dessous, on constate que les trajectoires suivent la même tendance. Plus le nombre de simulations augmente, plus la variance et la MSE diminuent, mais cela entraîne une augmentation du temps de calcul, ce qui n'est pas surprenant. Cependant, il est intéressant de se concentrer sur les valeurs présentées dans ces graphiques. Malgré les défis liés à la mise en œuvre de la méthode MLMC, nous constatons une MSE et une variance plus importantes pour la méthode MLMC que pour la méthode MC standard. Ainsi pour obtenir une performance optimale de la méthode MLMC, il est crucial d'utiliser les  $N_l$  optimaux. En outre, avec le même nombre de simulations  $N_{sim}$  que la méthode MC standard, la méthode MLMC est nettement plus rapide que la méthode MC standard. Cette différence de temps d'exécution s'explique par le fait que de nombreuses simulations sont réalisées avec un pas plus

gross dans MLMC, tandis que Monte-Carlo standard utilise le même pas fin pour toutes ses simulations.

Comparaison Monte-Carlo - Multi-level Monte-Carlo (non adaptatif)

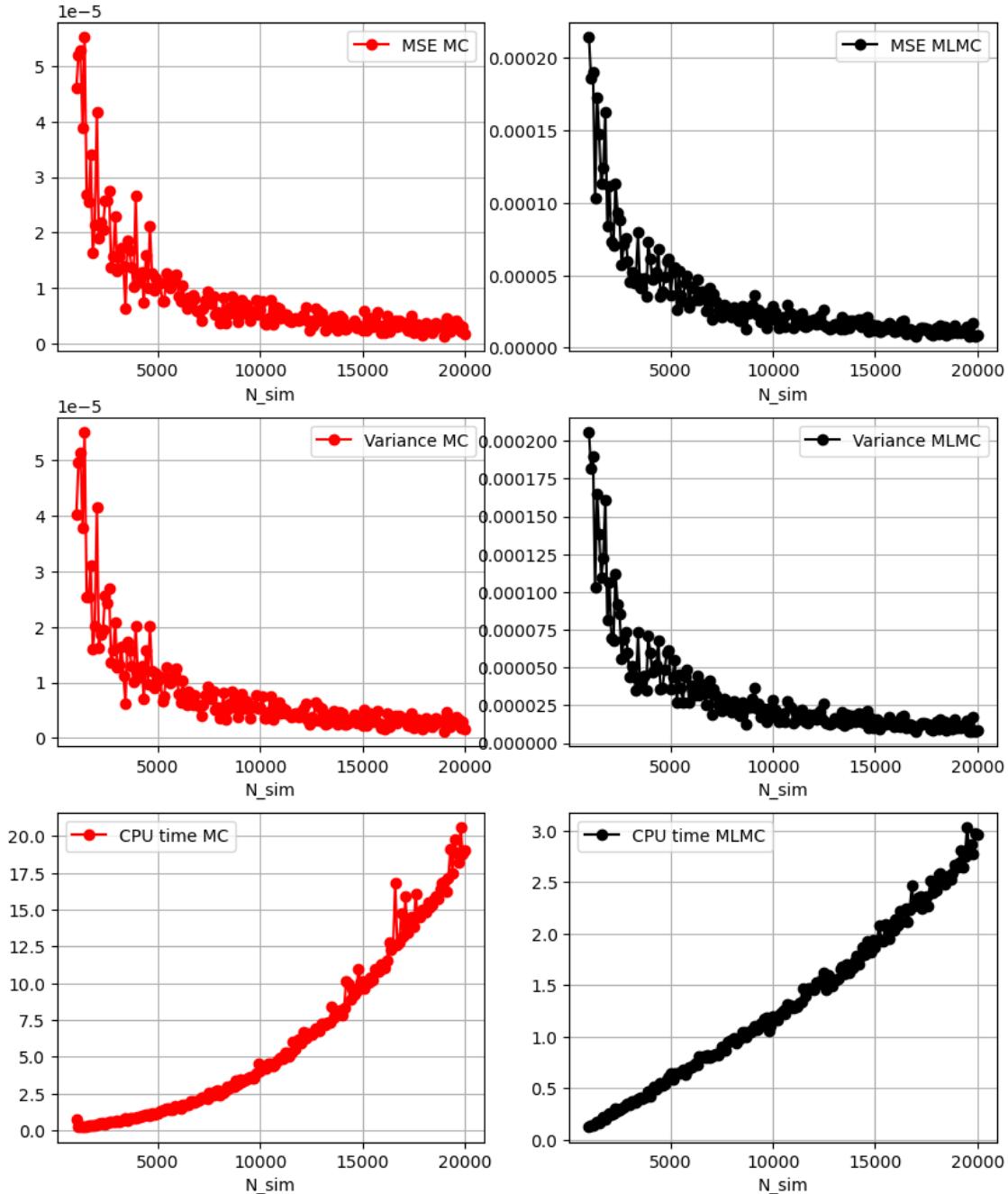


FIGURE 4.1 – Comparaison des performances entre MC et MLMC (non adaptatif)

### 4.2.2 MLMC avec paramètres optimisés

Si l'on s'intéresse à la comparaison des mêmes indicateurs mais en prenant le caractère optimal de MLMC, nous parvenons à des conclusions similaires.

Cependant, si l'on s'intéresse aux valeurs faibles de la tolérance  $\epsilon$  accordée, on remarque que l'algorithme MLMC réduit significativement la variance, démontrant ainsi la nécessité des valeurs optimales des  $N_l$  pour son bon fonctionnement.

Pour effectuer les comparaisons suivantes, différentes valeurs d' $\epsilon$  sont fixées afin de calculer la variance et la MSE de la méthode MLMC pour chaque valeur. Ensuite, l'estimateur Monte-Carlo standard est calculé avec un nombre de simulations égal à la somme des  $N_l$  nécessaires à l'estimateur MLMC pour atteindre une valeur  $\epsilon$  donnée.

Dans le tableau ci-dessous, les valeurs de gauche dans chaque cellule sont les valeurs obtenues pour Monte-Carlo standard et celles de droite sont celles de MLMC adaptatif.

| Monte-Carlo standard // MLMC Adaptatif |                               |                               |                  |
|--|-------------------------------|-------------------------------|------------------|
| $\epsilon$                             | MSE                           | Variance                      | CPU Time (s)     |
| $3.10^{-4}$                            | $1,2.10^{-4} // 7, 3.10^{-7}$ | $1,1.10^{-4} // 5, 1.10^{-8}$ | $8,1 // 19110,2$ |
| $4.10^{-3}$                            | $1,1.10^{-4} // 1, 2.10^{-5}$ | $1,2.10^{-4} // 1.2.10^{-6}$  | $4,9 // 16,7$    |
| $7.10^{-3}$                            | $2,0.10^{-4} // 1, 7.10^{-5}$ | $1,7.10^{-4} // 1, 7.10^{-5}$ | $3,9 // 6,9$     |

Il est notable que le temps d'exécution augmente considérablement pour le dernier niveau d' $\epsilon$  ( $3.10^{-4}$ ), mais cela permet une nette réduction de la variance et de la MSE. Néanmoins, afin d'obtenir des performances d'exécution similaires à celles de Monte-Carlo, tout en ayant une variance plus faible, il semble judicieux d'opter pour une valeur d' $\epsilon$  proche de  $10^{-3}$ .

### 4.3 Comparaison entre RQMC et QMC MLMC

Dans cette partie, nous effectuons des comparaisons entre la méthode Randomized Quasi Monte-Carlo (RQMC) et la méthode Multi-Level Quasi Monte-Carlo (QMC MLMC), ie Multi Level dont les incrémentations du mouvement brownien sont simulées par méthode Randomized Quasi Monte-Carlo comme expliqué en 2.2.2.

### 4.3.1 QMC MLMC avec paramètres non optimisés

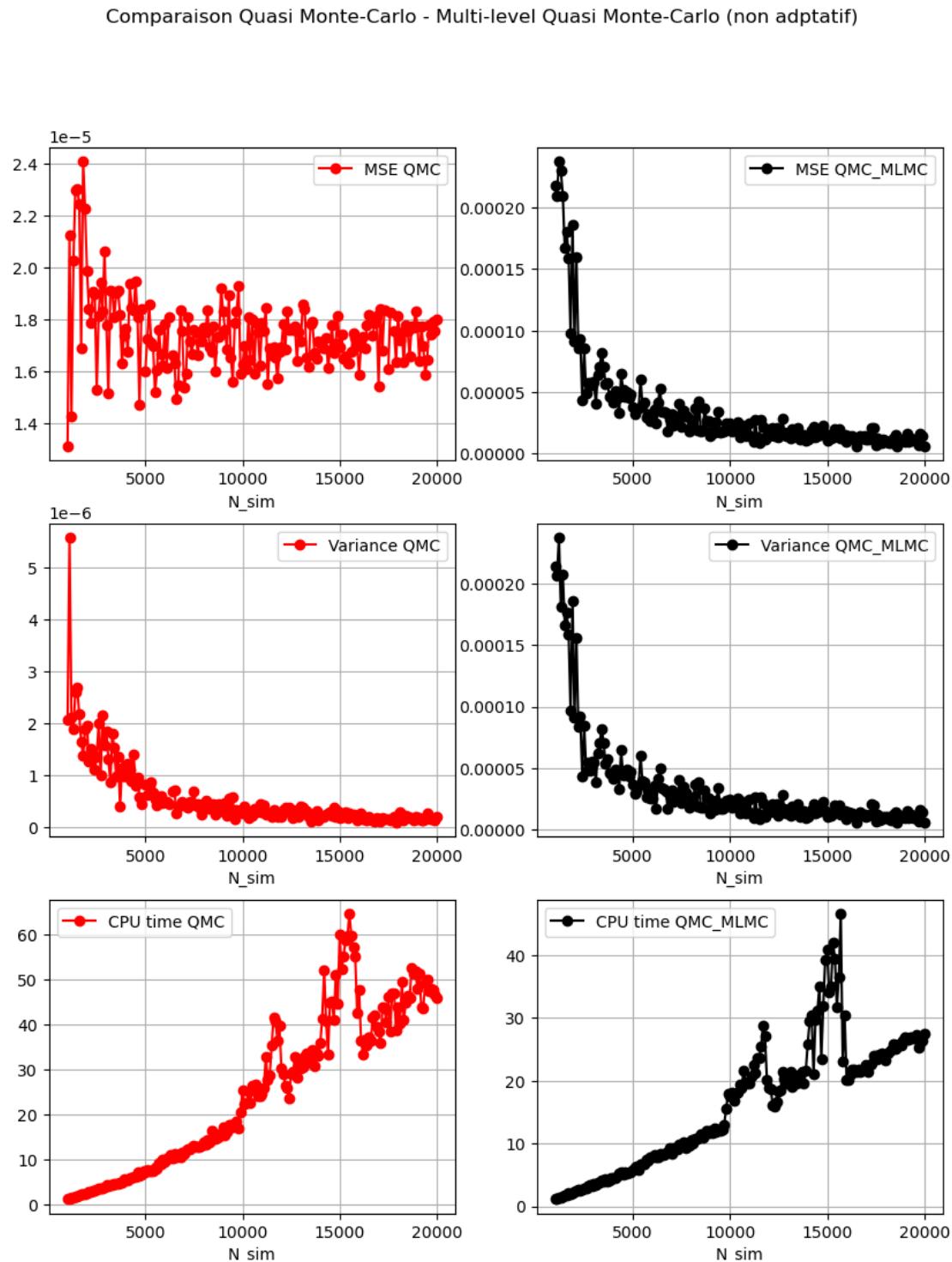


FIGURE 4.2 – Comparaison des performances entre QMC et QMC\_MLMC (non adaptatif)

Ce scénario est similaire au précédent, dans lequel nous avons comparé MC et MLMC. Toutefois, il convient de souligner que la variance de l'estimateur QMC est considérablement plus faible avec un nombre de simulations identique à celui de l'estimateur Monte-Carlo standard. Cependant, tout comme pour l'arbitrage à effectuer entre un faible  $\epsilon$  et un temps de calcul raisonnable pour MLMC, l'utilisation de QMC entraîne également une augmentation du temps de calcul nécessaire pour construire un estimateur.

### 4.3.2 QMC MLMC avec paramètres optimisés

Nous pouvons réaliser une étude comparable à celle faite dans la section 4.1.2.

| Quasi Monte-Carlo // QMC MLMC Adaptatif |                               |                               |                 |
|---|-------------------------------|-------------------------------|-----------------|
| $\epsilon$                              | MSE                           | Variance                      | CPU Time (s)    |
| $3.10^{-4}$                             | $4,3.10^{-6} // 7, 3.10^{-7}$ | $6,1.10^{-7} // 1,4.10^{-8}$  | 71,4 // 38678,6 |
| $4.10^{-3}$                             | $1,7.10^{-5} // 6, 5.10^{-6}$ | $5,1.10^{-6} // 2, 6.10^{-7}$ | 40,5 // 59,7    |
| $7.10^{-3}$                             | $1,8.10^{-5} // 1, 5.10^{-5}$ | $2,1.10^{-5} // 1, 0.10^{-6}$ | 7,7 // 43,1     |

Il est évident que l'utilisation de la méthode Quasi Monte-Carlo permet de réduire considérablement la variance dans les deux cas. Ainsi, la meilleure technique à appliquer pour faire baisser la variance est de combiner l'algorithme MLMC avec la méthode QMC, c'est à dire d'utiliser la méthode Multi-Level Quasi Monte Carlo (QMC MLMC).

## 4.4 Évolution de la MSE et de la CPU time

### 4.4.1 Variation de la MSE avec le nombre de simulations total

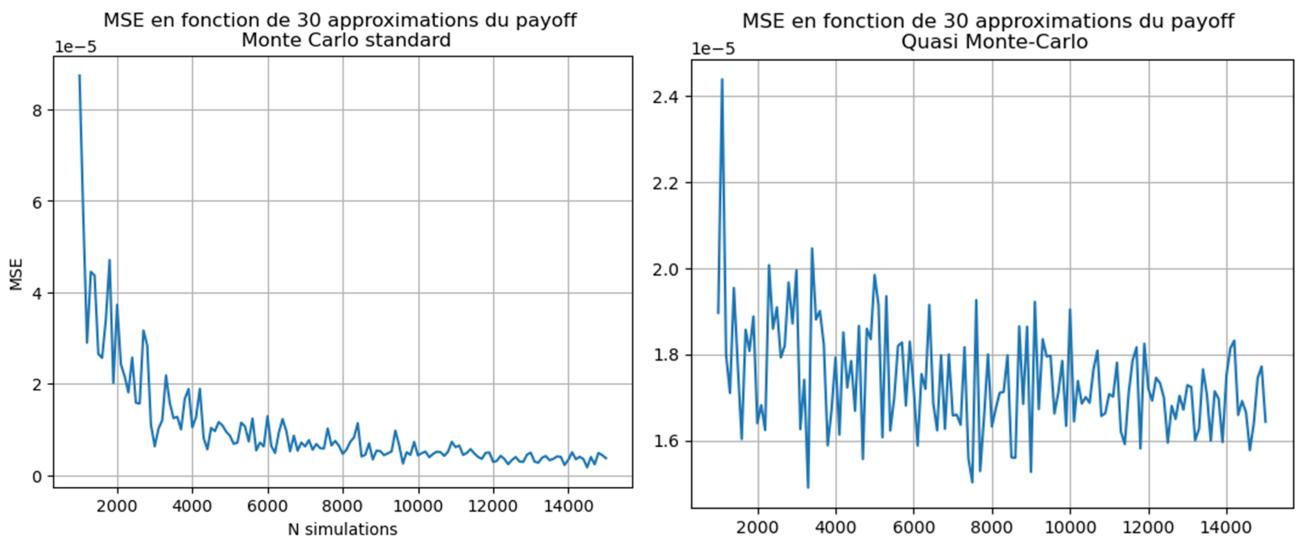


FIGURE 4.3 – Evolution de la MSE en fonction du nombre de simulations

La MSE diminue avec le nombre de simulations, ce qui est cohérent avec notre intuition.

#### 4.4.2 Variation de la MSE avec $\epsilon$

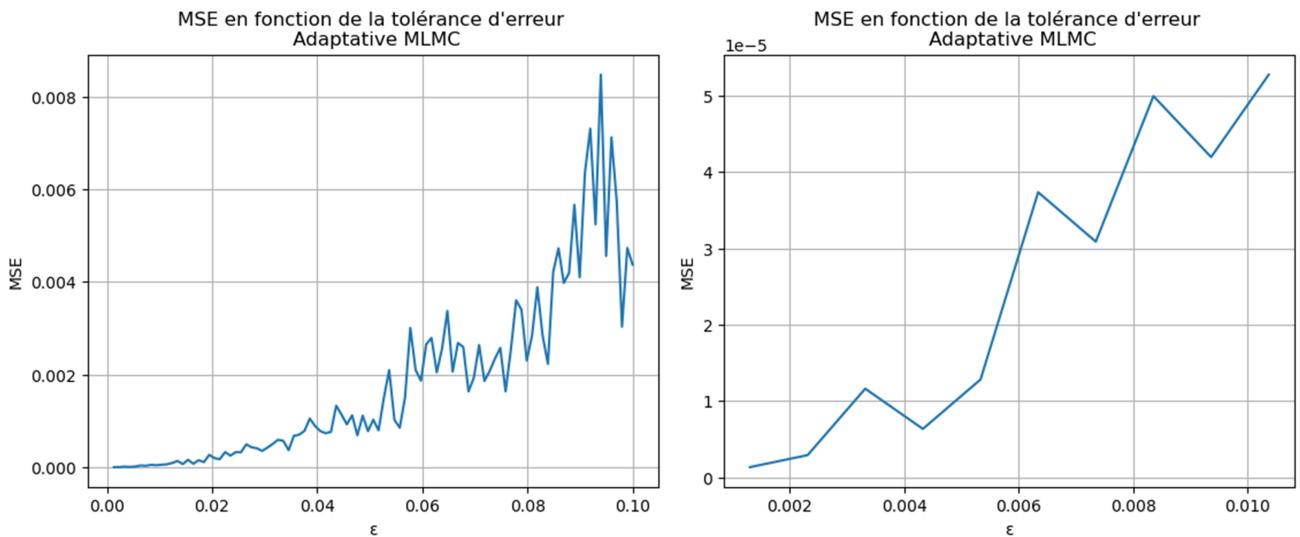


FIGURE 4.4 – Evolution de la MSE en fonction de la tolérance  $\epsilon$  – MLMC

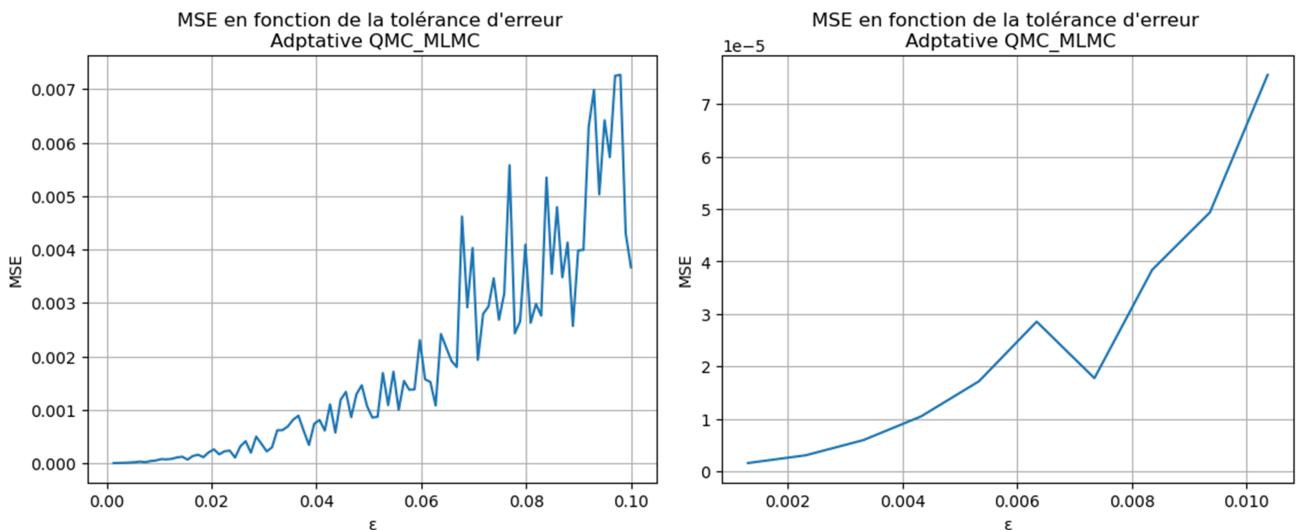


FIGURE 4.5 – Evolution de la MSE en fonction de la tolérance  $\epsilon$  – QMC MLMC

Bien que la variance soit plus stable que la MSE, cette dernière montre une tendance croissante avec l'augmentation d' $\epsilon$ . La méthode QMC MLMC, c'est à dire la méthode MLMC avec incrémentations du mouvement brownien simulés par méthode Randomized Quasi Monte Carlo, accentue cette croissance.

# 5 Utilisation d'un pont brownien

L'objectif est de simuler un mouvement brownien  $(W_{t_1}, W_{t_2}, \dots, W_{t_k})$  selon un pont brownien i.e.  $W_{t_i} | W_{t_{i-1}}, W_{t_{i+1}} \sim \mathcal{N}\left(\frac{(t_{i+1}-t_i)W_{t_{i-1}}+(t_i-t_{i-1})W_{t_{i+1}}}{t_{i+1}-t_{i-1}}, \frac{(t_{i+1}-t_i)(t_i-t_{i-1})}{t_{i+1}-t_{i-1}}\right)$  en suivant une séquence de Van der Corput de base 2.

## 5.1 Séquence de Van der Corput

Une séquence de Van der Corput est une séquence dont le principal avantage est de fournir une distribution de points plus uniforme dans un espace par rapport à une séquence aléatoire. Pour implémenter l'algorithme permettant d'obtenir une séquence de Van der Corput en base 2, la méthode est la suivante :

- Pour chaque indice  $i$  de la séquence, on calcule sa représentation en base 2
- On inverse les chiffres de la représentation en base 2 de l'indice  $i$  afin d'obtenir la représentation en base 2 de la séquence de Van der Corput pour l'indice  $i$ .
- On convertit la représentation en base 2 inversée en un nombre réel compris entre 0 et 1 en divisant chaque chiffre par  $2^z$  où  $z$  est la position du chiffre dans la représentation en base 2 inversée.
- On additionne ensuite ces termes pour obtenir la valeur de la séquence de Van der Corput pour l'indice  $i$

La fonction *vdc* effectue ces étapes. Cela nous permet d'ordonner les  $t_i$  comme notés dans le cours afin d'obtenir le pont brownien :  $t_k, t_{\frac{k}{2}}, t_{\frac{k}{4}}, \dots$

## 5.2 La méthode du pont brownien

Il faut désormais générer les  $W_{t_i} | W_{t_{i-1}}, W_{t_{i+1}} \sim \mathcal{N}\left(\frac{(t_{i+1}-t_i)W_{t_{i-1}}+(t_i-t_{i-1})W_{t_{i+1}}}{t_{i+1}-t_{i-1}}, \frac{(t_{i+1}-t_i)(t_i-t_{i-1})}{t_{i+1}-t_{i-1}}\right)$  suivant l'ordre d'indice donné par la séquence de Van der Corput obtenue avec la fonction *vdc*, soit  $W_{t_k}$  puis  $W_{t_{\frac{k}{2}}}, W_{t_{\frac{k}{4}}}, \text{etc.}$

C'est ce qu'effectue la fonction *simulate\_brownian\_motion* à travers ces différentes étapes :

- Appel de la *vdc* puis arrondi des éléments au plus proche supérieur
- On génère les variables aléatoires et on les trie suivant la séquence de Van der Corput
- On fixe les valeurs extrêmes du pont brownien
- On calcule les  $W_{t_i}$

Nous avons combiné cette méthode avec des méthodes de QMC de simulations de normales, notamment la méthode de Box-Muller 2D et une séquence de Sobol (expliquée en 2.2.2). Les résultats obtenus pour l'estimation Monte-Carlo standard avec ce pont brownien étaient très intéressants, car la variance des estimateurs a été réduite jusqu'à  $10^{-11}$  !

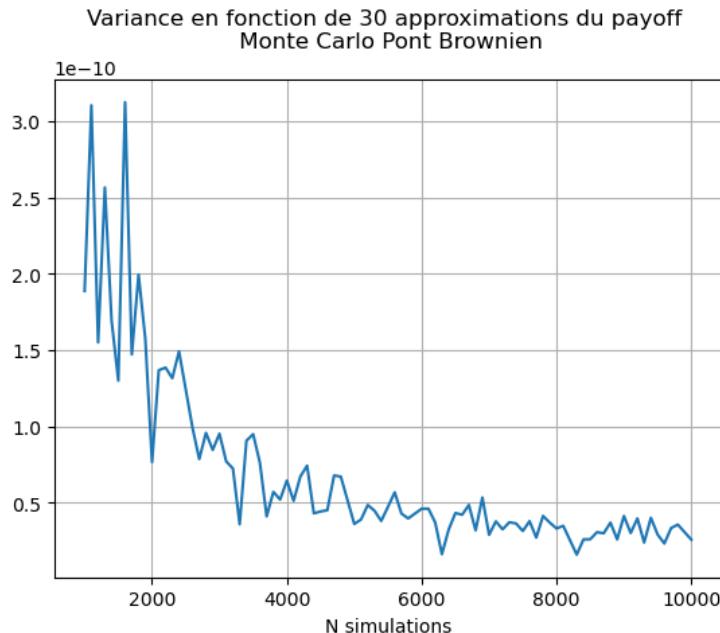


FIGURE 5.1 – Évolution de la variance des estimateurs QMC avec pont brownien en fonction du nombre de simulations

On arrive cependant à des résultats quelque peu décevants pour le pont brownien associé à l'algorithme MLMC car la variance et la MSE demeurent similaires à celles obtenues dans le cas QMC MLMC.

Si l'on se réfère à la conclusion du papier de Michael B. Giles, dans la section 7 "Remarques Finales", on peut comprendre que la mise en place du pont brownien devrait se faire *pendant* l'exécution de l'algorithme MLMC plutôt qu'avant, comme nous l'avons fait. En d'autres termes, à chaque étape jusqu'à convergence, l'algorithme utilise les incrémentations du pont brownien déjà calculés tout en en créant des nouveaux.

## 6 Conclusion

La méthode MLMC repose sur une idée mathématique simple, consistant à utiliser une somme télescopique d'estimateurs de plus en plus complexes pour approximer sans biais une espérance, tout en réduisant considérablement la variance de l'estimateur final. Cependant, pour que cette méthode fonctionne efficacement, il est impératif de calculer de manière optimale les valeurs de  $N_l$ , c'est-à-dire le nombre de simulations à effectuer à chaque niveau  $l$ .

De plus, comme cette méthode MLMC fonctionne comme un ensemble de méthodes de Monte-Carlo à la suite, il est possible de la combiner avec des méthodes classiques de réduction de variance de Monte-Carlo pour obtenir des résultats encore meilleurs - au prix d'un temps de calcul plus long cependant.

En théorie, la méthode MLMC permet de faire baisser de manière arbitraire la variance des estimateurs calculés ; mais en pratique, comme on peut le voir dans cette étude, le temps et la complexité de calcul deviennent très rapidement très élevés et nécessitent des puissances de calcul toujours plus importantes.

# 7 Annexes

## 7.1 Démonstration $N_l \propto \sqrt{V_l h_l}$

D'après la section 3.2.1 :

$$\mathbb{V}[\hat{Y}] = \sum_{l=1}^L \mathbb{V}[\hat{Y}_l] = \sum_{l=1}^L \frac{V_l}{N_l}$$

Nous souhaitons minimiser la variance. Le problème d'optimisation sous contrainte est le suivant :

$\min(\mathbb{V}[\hat{Y}])$  sous contrainte  $\sum_{l=1}^L C_l N_l = A$  avec  $C_l$  le cout CPU par échantillon et  $A$  une constante représentant le cout total

$$\mathcal{L} = \sum_{l=1}^L \frac{V_l}{N_l} + \lambda \left( \sum_{l=1}^L C_l N_l - A \right)$$

$$\frac{\partial \mathcal{L}}{\partial N_l} = -\frac{V_l}{N_l^2} + \lambda C_l$$

$$\text{Donc : } N_l = \sqrt{\frac{V_l}{\lambda C_l}}$$

$$C_l \text{ peut être approximé par } M^l \text{ donc } N_l = \sqrt{\frac{V_l}{\lambda M^l}} \propto \sqrt{V_l h_l}$$

## 7.2 Évolution des $N_l$

On peut aussi s'intéresser aux valeurs  $N_l$  donnée par l'algorithme de M. Giles. Ainsi il est intéressant de comparer ces valeurs en faisant varier  $\epsilon$ .

### 7.2.1 Multi-Level Monte-Carlo

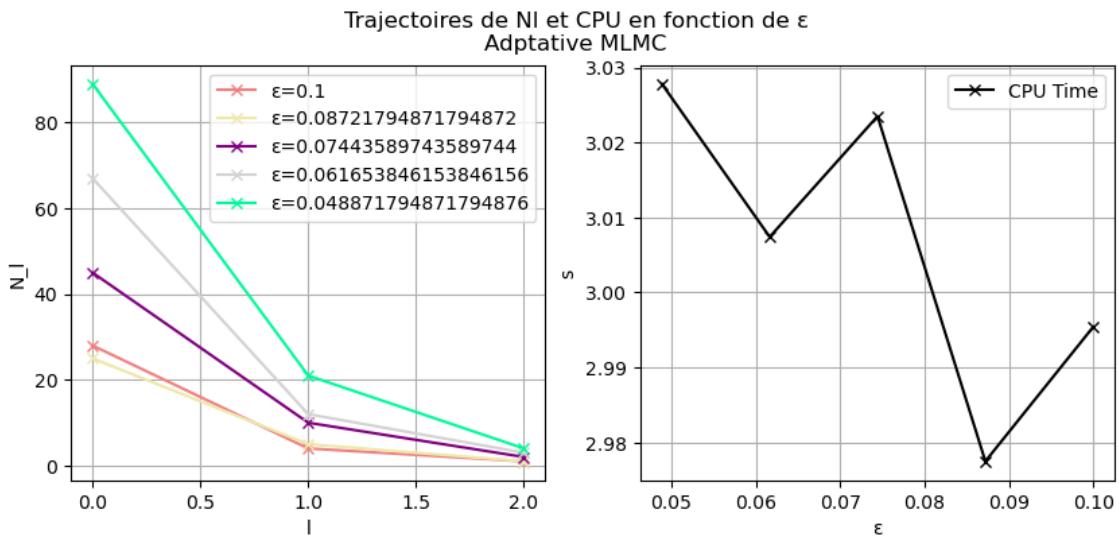


FIGURE 7.1 – Evolution des  $N_l$  et du temps CPU pour des  $\epsilon$  élevés - MLMC

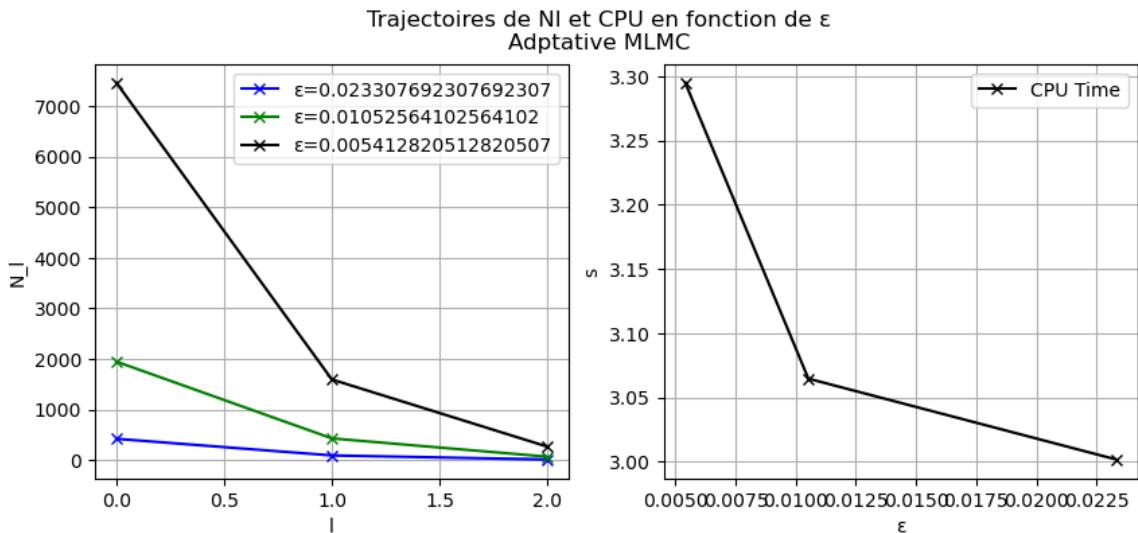


FIGURE 7.2 – Evolution des  $N_l$  et du temps CPU pour des  $\epsilon$  plus faibles - MLMC

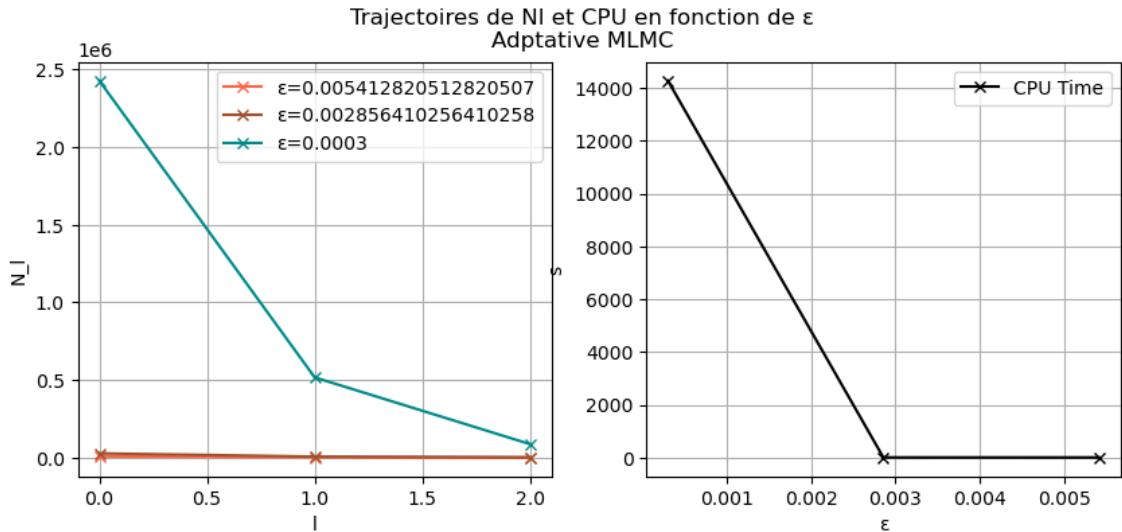


FIGURE 7.3 – Evolution des  $N_l$  et du temps CPU pour des  $\epsilon$  très faibles - MLMC

Il est notable que la valeur des  $N_l$  explose très vite lorsque la valeur de  $\epsilon$  devient inférieure à  $10^{-3}$ . Le nombre de  $N_l$  à calculer passe alors à des millions, entraînant un temps de calcul considérablement plus long que pour des  $\epsilon$  plus larges. Ainsi, on voit bien ici l’arbitrage entre la tolérance d’erreur et le temps de calcul. Bien que théoriquement il soit possible de réduire considérablement la variance, cela nécessiterait une puissance de calcul considérable et un temps de calcul prolongé en pratique.

## 7.2.2 Multi-Level Quasi Monte-Carlo

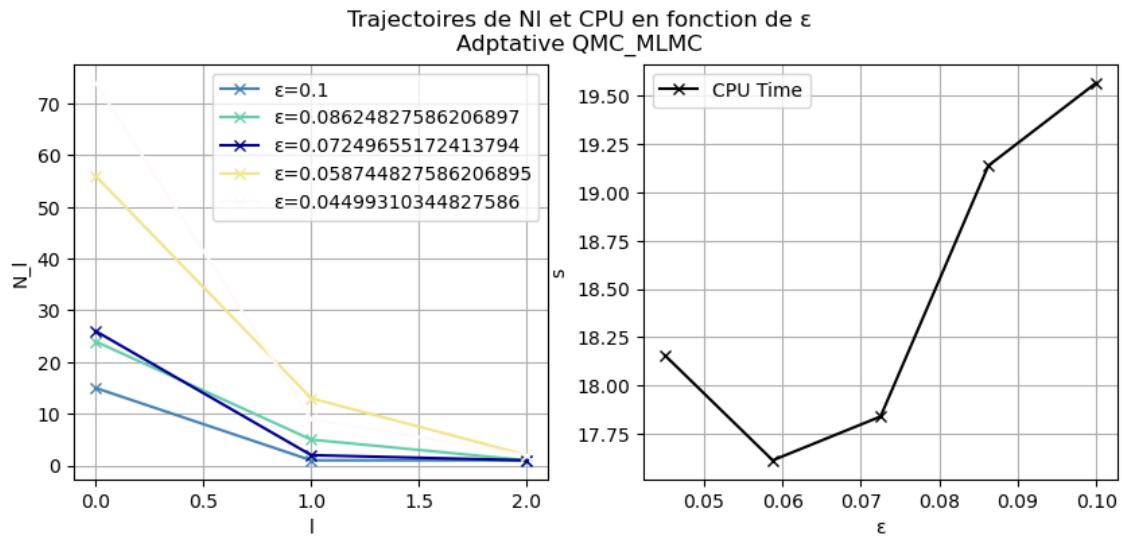


FIGURE 7.4 – Evolution des  $N_l$  et du temps CPU pour des  $\epsilon$  élevés - RQMC\_MLMC

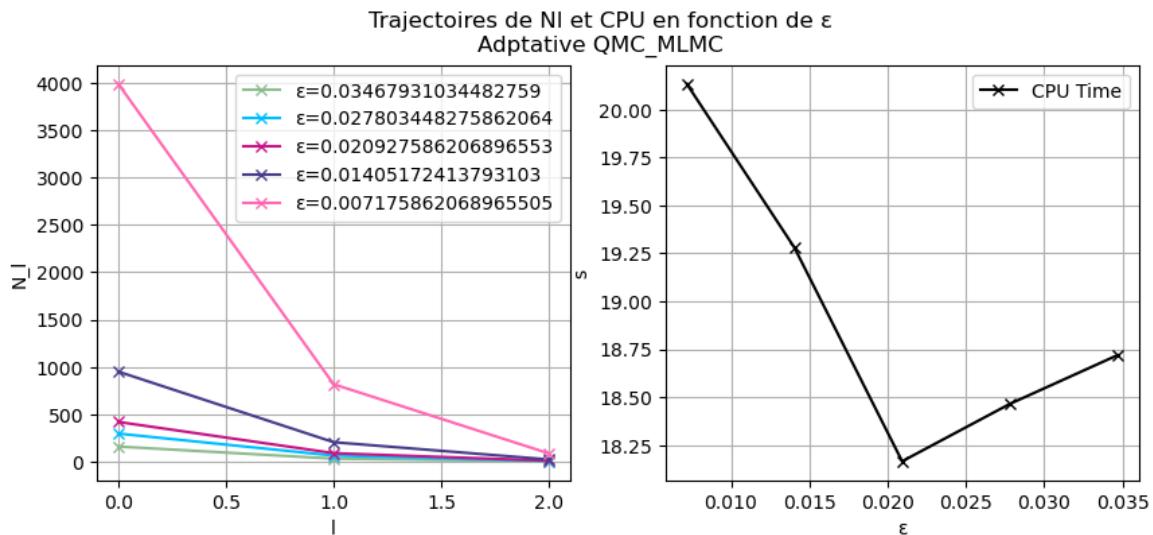


FIGURE 7.5 – Evolution des  $N_l$  et du temps CPU pour des  $\epsilon$  plus faibles - RQMC\_MLMC

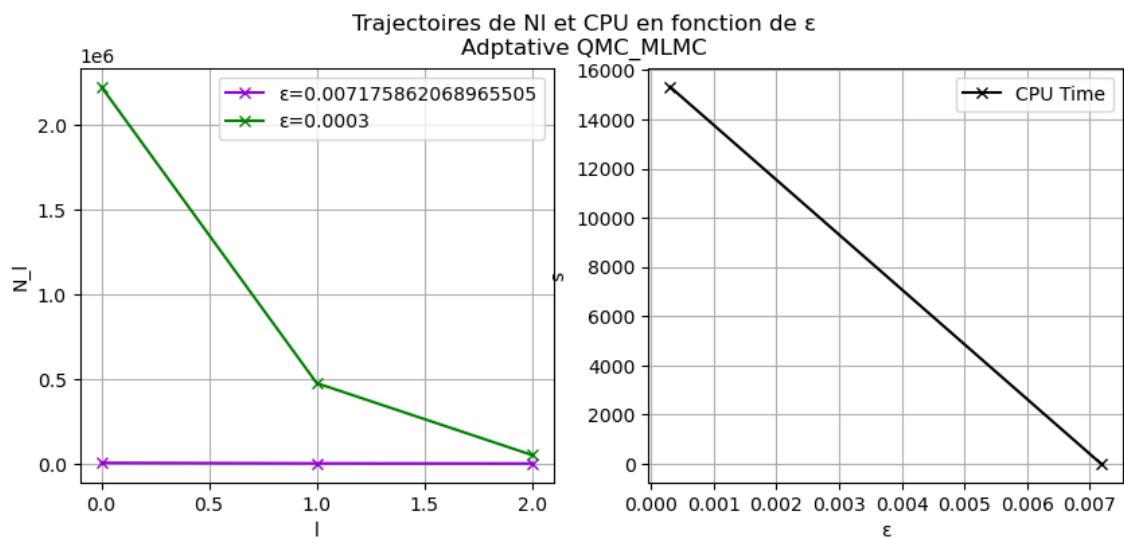


FIGURE 7.6 – Evolution des  $N_l$  et du temps CPU pour des  $\epsilon$  très faibles - RQMC\_MLMC

On peut voir que le même phénomène s'observe pour le cas du Multi-Level Quasi Monte-Carlo.