Parallel computing for linear, nonlinear and linear inverse problems in finance

Lokman Abbas-Turki

UNIVERSITÉ PARIS-EST

Ecole Doctorale : MATHÉMATIQUES ET SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THESE

présentée pour obtenir le grade de

DOCTEUR EN MATHÉMATIQUES

de l'Université Paris-Est

par

Lokman A. ABBAS-TURKI

Calcul parallèle pour les problèmes linéaires, non-linéaires et linéaires inverses en finance.

Parallel Computing for linear, nonlinear and linear inverse problems in finance.

Soutenue le 21 septembre 2012 devant le jury composé de :

Rapporteurs :	
Mireille BOSSY	D. Recherche INRIA, Sophia Antipolis
Paul GLASSERMAN	Professeur, Columbia University, USA
Examinateurs :	
Frédéric ABERGEL	Professeur, École Centrale Paris
Stéphane VIALLE	Professeur, Supélec
Directeurs de thèse :	
Damien LAMBERTON	Professeur, Université Paris-Est
Bernard LAPEYRE	Professeur, Ecole des Ponts ParisTech

Remerciements

Ce travail a été effectué au sein du Laboratoire d'Analyse et de Mathématiques Appliquées (LAMA) de l'Université Paris-Est. J'adresse mes remerciements au Professeur Marco CANNONE et au Directeur de Recherche CNRS François BOUCHUT, ancien et nouveau directeurs du LAMA, pour m'avoir accueilli et pour avoir mis à ma disposition tout ce qui a été nécessaire à la bonne réalisation de ce mémoire. Que tous les membres du LAMA, en particulier du GT modélisation stochastique et finance, trouvent ici l'expression de ma gratitude. Leurs gentillesse et disponibilité ont grandement contribué à rendre ma tâche facile et mon séjour agréable.

Je dois toute la reconnaissance au Professeur Damien LAMBERTON pour la confiance qu'il m'a accordée en me proposant ce sujet et pour son aide et les conseils très utiles qu'il n'a cessé de me prodiguer. Qu'il trouve ici mes plus respectueux remerciements.

Il m'est tout particulièrement agréable de pouvoir remercier le Professeur Bernard LAPEYRE qui a su m'écouter, me comprendre et m'encourager durant les trois années de thèse. Par son expérience dans le domaine numérique, il m'a aidé à faire évoluer très sensiblement ce travail.

Je remercie aussi très chaleureusement Professeur Vlad BALLY pour son aide et ses conseils avisés.

Je tiens également à remercier le Professeur Stéphane VIALLE de son aide et ses remarques pertinentes sur les aspects de parallélisation massive et aussi d'avoir accepté de juger ce travail.

Que le Professeur Fréderic ABERGEL reçoive toute ma reconnaissance et mon profond respect pour l'honneur qu'il m'a fait en acceptant d'examiner mon travail.

Je tiens à remercier Madame Mireille BOSSY, Directrice de Recherche, et le Professeur Paul GLASSERMAN d'avoir accepté la charge de rapporter mon travail. Leurs questions et commentaires pertinents m'ont permis de rendre plus claire ma rédaction et m'ont donné de nouvelles pistes de réflexion.

Qu'il me soit permis d'exprimer une pensée très particulière à mon oncle Omar pour son précieux soutien moral et financier durant mon cursus universitaire.

Je suis très reconnaissant à tous les enseignants que j'ai eus durant mes dix huit années d'étude et de formation, sans lesquels ce mémoire n'aurait pas eu son contenu actuel.

Enfin, on dit souvent que l'on ne choisit pas ses parents. En ce qui me concerne, si j'avais à le faire, j'aurais certainement retenu les miens. Qu'ils trouvent en moi l'enfant redevable toute sa vie.

Résumé

Traiter les problèmes paraboliques multidimensionnels linéaires, non-linéaires et linéaires inverses est l'objectif principal de ce travail. C'est le mot multidimensionnel qui rend pratiquement incontournable l'utilisation des méthodes de simulations fondées sur le Monte Carlo. Le mot multidimensionnel rend aussi indispensable l'utilisation des architectures parallèles. En effet, les problèmes manipulant un large nombre d'actifs sont de grands consommateurs en temps d'exécution, et il n'y a que la parallélisation pour faire chuter ce dernier.

De ce fait, le premier objectif de notre travail consiste à proposer des générateurs de nombres aléatoires appropriés pour des architectures parallèles et massivement parallèles de clusters de CPUs/GPUs. Nous testerons le gain en temps de calcul et l'énergie consommée lors de l'implémentation du cas linéaire du pricing européen. Le deuxième objectif est de reformuler le problème non-linéaire du pricing américain pour que l'on puisse avoir des gains de parallélisation semblables à ceux obtenus pour les problèmes linéaires. La méthode proposée fondée sur le calcul de Malliavin est aussi plus avantageuse du point de vue du praticien au delà même de l'intérêt intrinsèque lié à la possibilité d'une bonne parallélisation. Toujours dans l'objectif de proposer des algorithmes paralléles, le dernier point est l'étude de l'unicité de la solution de certains cas linéaires inverses en finance. Cette unicité aide en effet à avoir des algorithmes simples fondés sur Monte Carlo.

Mots clés : Contrat européen, contrat américain, calcul de Malliavin, réduction de variance, réduction de biais, régularité du flow, GPU, Monte Carlo, générateur de nombres aléatoires.

Abstract

Handling multidimensional parabolic linear, nonlinear and linear inverse problems is the main objective of this work. It is the multidimensional word that makes virtually inevitable the use of simulation methods based on Monte Carlo. This word also makes necessary the use of parallel architectures. Indeed, the problems dealing with a large number of assets are major resources consumers, and only parallelization is able to reduce their execution times.

Consequently, the first goal of our work is to propose "appropriate" random number generators to parallel and massively parallel architecture implemented on CPUs/GPUs cluster. We

quantify the speedup and the energy consumption of the parallel execution of a European pricing. The second objective is to reformulate the nonlinear problem of pricing American options in order to get the same parallelization gains as those obtained for linear problems. In addition to its parallelization suitability, the proposed method based on Malliavin calculus has other practical advantages. Continuing with parallel algorithms, the last point of this work is dedicated to the uniqueness of the solution of some linear inverse problems in finance. This theoretical study enables the use of simple methods based on Monte Carlo.

Key words : European contract, American contract, Malliavin calculus, variance reduction, bias reduction, flow regularity, GPU, Monte Carlo, random number generator.

Table des matières

Liste des tableaux

Table des figures

Chapitre 0

Introduction générale

> Most important of all was Fibonacci's introduction of Hindu-Arabic numerals. He not only gave Europe the decimal system, which makes all kinds of calculation far easier than with Roman numerals ; he also showed how it could be applied to commercial bookkeeping, to currency conversions and, crucially, to the calculation of interest. Significantly, many of the examples in the *Liber Abaci* are made more vivid by being expressed in terms of commodities like hides, peppers, cheese, oil and spices.
>
> Niall Ferguson "The Ascent of Money"

> La simulation numérique a aussi des liens forts avec la théorie et ça va dans les deux sens. Parfois un calcul théorique va vous donner des algorithmes beaucoup plus performants pour faire votre calcul. Parfois votre calcul va changer votre vision théorique d'un problème.
>
> Cédric Villani, durant la remise du prix Joseph Fourier 2011

0.1 Contexte du travail

La première citation ci-dessus mentionne la relation fondamentale en occident entre les mathématiques et la finance. En effet, même si la physique théorique est le premier domaine auquel on peut penser lorsqu'on parle de mathématiques appliquées, l'histoire nous enseigne que la première activité à utiliser les mathématiques est la finance. Il est aussi important de conditionner cette information au fait que les mathématiques en finance apparaissent beaucoup plus naturelles même pour les novices, alors que l'utilisation des mathématiques pour la physique est impossible sans la maîtrise de certains principes fondamentaux et relations physiques.

Cet argument est peut-être celui qui justifie le mieux ce fait historique et l'important nombre des mathématiciens qui s'intéressent sans discontinuer à ce domaine d'application . En effet, il est plus facile pour un mathématicien d'avoir une bonne intuition pour un problème financier que pour un phénomène gravitationnel ou quantique.

Le travail présenté dans ce manuscrit est une illustration du propos relaté précédemment. En effet, cette recherche résulte d'un intérêt croissant accordé à la simulation parallèle multi-coeurs et many-coeurs [1] et les problèmes abordés relèvent de la finance de marchés. En plus de la théorie et de l'expérience, la simulation numérique s'est imposée comme l'un des piliers de la connaissance scientifique. Dans cette étude, la simulation parallèle a pris une place importante. Une autre partie de ce travail consiste à formuler ou à reformuler un problème financier spécifique pour qu'il soit efficacement implémentable sur des architectures parallèles. Comme mentionné par Cédric Villani (seconde citation), ce double objectif est puissant et peut non seulement améliorer les résultats numériques, mais aussi nous procurer une meilleure vision du problème théorique.

Nous allons profiter de cette introduction pour dresser les évolutions historiques de l'architecture informatique depuis 1945 et détailler les tournures actuelles que prennent ces évolutions. A travers ce bref exposé, on essayera aussi de comprendre les changements progressifs qui ont eu lieu sur l'orientation des avancées théoriques. Après, nous reviendrons vers le domaine qui nous intéresse ici qui est les mathématiques appliquées à la finance de marchés. Nous allons analyser les causes de la montée fulgurante de ce domaine durant ces trente dernières années et voir l'action de la simulation sur les développements actuels et leurs orientations futures. Nous espérons ainsi concilier l'évolution des mathématiques appliquées, de l'outil informatique et des inventions financières. Nous précisons que l'objectif de notre propos n'est pas d'entrer dans les détails techniques mais de donner un aperçu de l'évolution de l'architecture informatique et des applications en mathématiques financières.

0.1.1 Du parallèle au séquentiel puis du séquentiel au parallèle

Malchanceux est l'être humain lorsqu'il cherche à se faire remplacer pour réduire ces efforts, car cette recherche en demande déjà beaucoup. Heureusement, cette quête n'a pas été vaine parce qu'elle a permis au moins de conforter la connaissance détaillée de certaines tâches, puisqu'on ne peut donner à une machine un travail à faire sans maîtriser les différentes parties

1. autrement dit, parallèle et massivement parallèle.

simples de cette tâche.

Certains retracent l'évolution des calculateurs à partir du boulier chinois (7 siècles avant J.-C.) mais, comme annoncé précédemment, on s'intéressera exclusivement aux machines électriques d'après guerre [2]. En supposant que l'on manipule une tâche qui se scinde en parties élémentaires (allocation mémoire d'une variable, initialisation d'une variable, addition de deux variables,...) qui peuvent, par moments, être entreprises sans communications entre elles (indépendantes au sens informatique), la définition d'une machine parallèle [3] et d'une machine séquentielle devient naturelle :

- Une machine est dite séquentielle si elle ne peut exécuter les parties élémentaires indépendantes que d'une manière séquentielle, c'est à dire l'une après l'autre.
- Une machine est dite parallèle si elle est capable d'exécuter au moins deux parties élémentaires indépendantes d'une manière parallèle, c'est à dire l'une en même temps que l'autre.

Définie ainsi, il est très difficile de trouver une machine séquentielle maintenant et pourtant ce n'était pas le cas, durant une longue période, lors de la commercialisation des premiers calculateurs à transistor. Aussi étonnant que cela puisse paraître, la première machine électronique polyvalente (Turing-complète [11]), l'ENIAC, était parallèle [28]. Plus tard, une version modifiée est devenue la toute première machine de Von Neumann [27], c'est à dire dans laquelle le programme est aussi stocké en mémoire. Cette révolution a été accompagnée par la contribution de Von Neumann au développement de la méthode de Monte Carlo. Tout ceci a fait de l'ENIAC le premier supercalculateur à exécuter durant une année les tests de la bombe à hydrogène.

Ensuite, on trouve la période qui commence par la commercialisation par Intel de l'un des premiers micro-ordinateurs en 1972. Entre la date de l'invention de la première machine à transistor (TRADIC 1954) et 1972, on avait toujours des machines parallèles parce qu'elles étaient destinées à être manipulées par des spécialistes et pour des applications généralement militaires. Démocratiser les calculateurs pour le grand public a posé de très grands challenges *hardware* et *software*. Les premiers ont été en grande partie résolus grâce à une maîtrise de la conception de mémoires et à leur hiérarchisation. Une fois que l'on a mis suffisamment de mémoire cache [4] et

2. La seconde guerre mondiale.
3. Cette définition est un exemple d'abus qu'on fait pour éviter des questions comme l'existence de plusieurs types de parallélisme donnant lieu par exemple à des architectures comme : Hyperthreading, FPGA,...
4. C'est une mémoire faisant partie de la CPU.

de mémoire vive [5] à la disposition [6] de l'unité de calcul, on pouvait beaucoup plus facilement exécuter des tâches assez complexes pour qu'elles puissent intéresser un éventuel acheteur amateur de nouvelles technologies.

Comme dans d'autres domaines, l'idée de l'amortissement des coûts de production en vendant au grand public est devenue incontournable en informatique. De plus, comme l'être humain raisonne séquentiellement, il se contente pour son premier achat d'une machine qui a un seul coeur de calcul mais assez de mémoire pour faire du traitement de texte, dessiner ... ou regarder une vidéo à basse résolution. Ainsi, la question de mettre plusieurs coeurs ne se posait même pas puisqu'on doublait [7], tous les 18 mois, le nombre de transistors sur une carte et la fréquence de fonctionnement. De ce fait, un coeur deux fois plus rapide peut être virtuellement considéré comme deux travaillant en même temps.

Pour faire de la parallélisation entre les années 1980 et 2000, l'utilisateur scientifique avait pratiquement pour unique choix l'achat de plusieurs machines séquentielles et de les connecter. Ceci ne s'est pas déroulé sans douleurs pour la communauté scientifique qui s'est vue très désavantagée par la multiplicité des plates-formes, par une communication inter-machine peu efficace et par des documentations insuffisantes. Devant de telles difficultés, c'est dans un monde de traitements séquentiels des EDPs [8] que les mathématiques appliquées ont eu leur essor, favorisant ainsi des méthodes itératives à fort caractère séquentiel. Un exemple standard saisissant de cette orientation séquentielle des recherches théoriques est celui de la multiplication matricielle : Dans une implémentation parallèle et avec une mémoire cache relativement faible pour chaque coeur, il est simple de séparer les multiplications faites par chaque coeur indépendamment des autres pour que le temps d'execution soit divisé par le nombre de coeurs disponibles. Alors que dans une implémentation EDPiste séquentielle, on essayera d'abord en amont d'avoir des matrices les plus creuses possibles et d'implémenter ensuite des algorithmes de multiplications qui utilisent intensément la grande quantité de cache disponible lorsqu'on travaille avec un seul coeur [56].

Cette orientation de l'architecture informatique et des développements théoriques a duré une vingtaine d'années, jusqu'au moment où on est arrivé à la limite en nombre de transistors que l'on pouvait mettre sur un même coeur. D'après la figure 1, on voit qu'à partir d'une

5. C'est une mémoire séparée de la CPU qui ne stocke des données que lorsque la machine est alimentée.
6. La mise à disposition regroupe la quantité de mémoire disponible et la rapidité d'accès à celle-ci.
7. Loi de Moore.
8. Équations aux dérivées partielles.

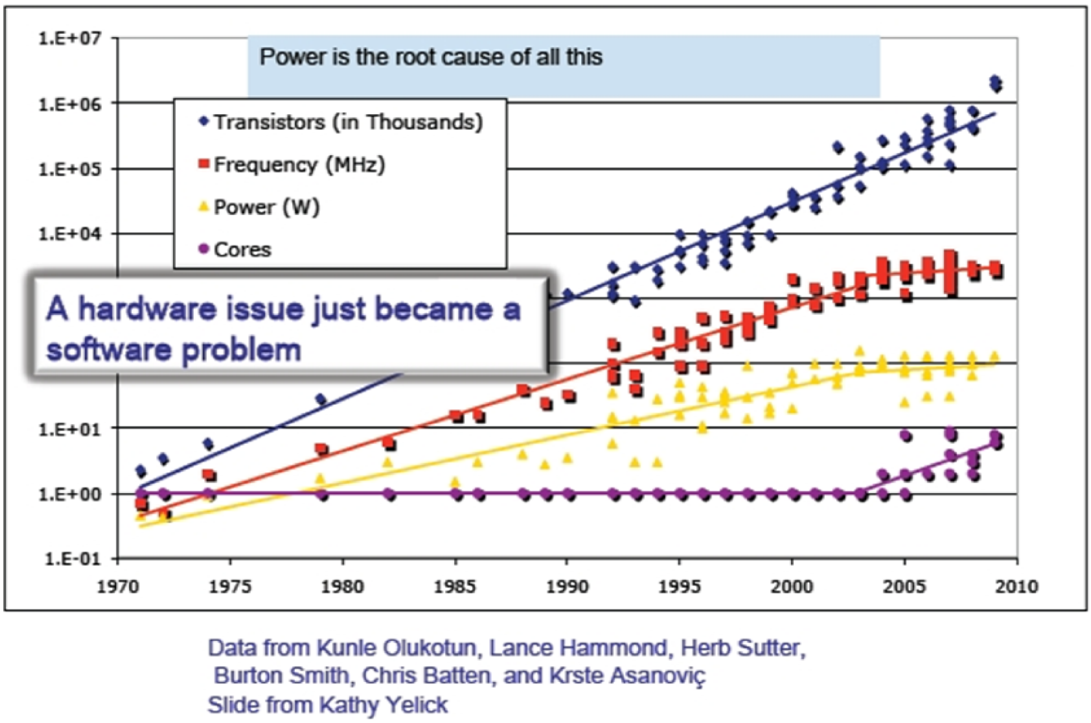

FIGURE 1 – Evolution historique sur une CPU de : nombre de coeurs, nombre de transistors, fréquence et puissance de fonctionnement.

certaine fréquence, la puissance électrique de fonctionnement est devenue inacceptable [9] pour justifier la conception mono-coeur des CPUs [10]. Les limites de cette architecture proviennent non seulement du fait que la puissance de fonctionnement est linéairement proportionnelle à la fréquence [11], mais aussi des difficultés rencontrées pour réaliser des gravures des circuits de plus en plus fines. Pour que la loi de Moore reste vérifiée, la solution directe consistait à augmenter le nombre de coeurs sur une même puce. Ceci a assez bien fonctionné pour deux coeurs puis pour quatre coeurs et a conduit à une ascension du nombre de coeurs dans les super-calculateurs (voir figure 2).

Puis, le même problème d'accès mémoire s'est invité de nouveau dans la liste des problèmes *hardware*. En fait, presque 80% de la CPU représente de la mémoire cache et augmenter le nombre de coeurs réduit considérablement le cache pour chaque coeur. C'est aussi vrai pour

9. La diffusion de chaleur n'étant plus supportable par les matériaux utilisés.
10. Central Processing Unit est le processeur principal d'une machine qui regoupe plusieurs unités de calcul et de la mémoire cache.
11. $P = CV^2 f$ où C est la capacité, V le potentiel et f la fréquence. Même si en général, la puissance totale de la carte mère est non-linéairement reliée à la fréquence.

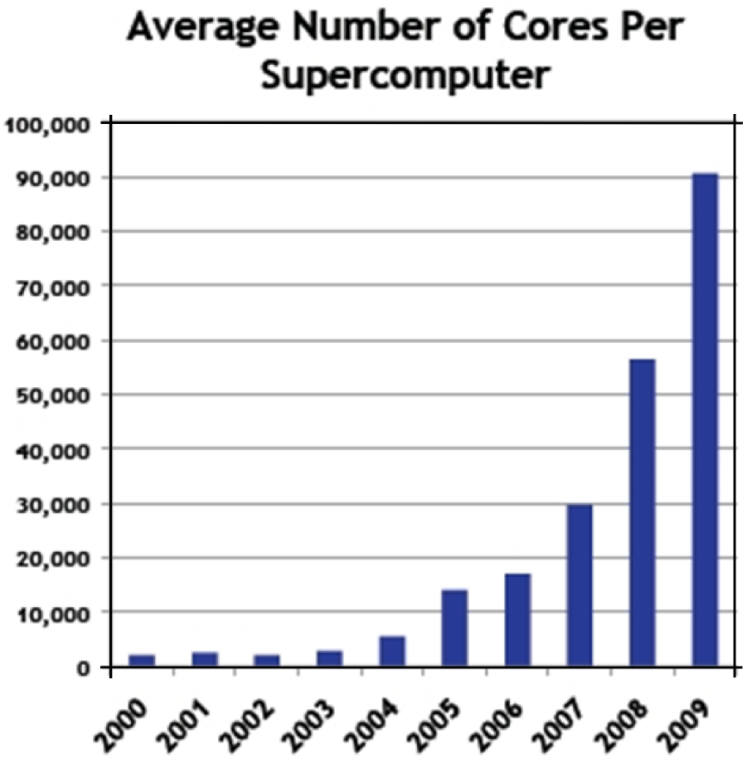

FIGURE 2 – Croissance exponentielle du nombre de coeurs due au besoin de parallélisation.

la RAM [12] qui est sollicitée par plusieurs coeurs. Par conséquent, il faut augmenter le nombre de bus [13] pour accéder à cette mémoire [14]. La solution à ces problèmes est fondée sur plus de localité de la mémoire et consiste à paralléliser le code sur les différents coeurs et à proposer des achitectures qui contiennent encore beaucoup plus de coeurs que deux ou quatre. C'est encore une fois la commercialisation au grand public qui mène la cadence, puisque ce sont les GPUs [15], processeurs des consoles vidéos, qui se sont imposées comme alternative sérieuse pour la parallélisation massive. C'est tellement vrai que l'introduction des GPUs dans les super-calculateurs a complètement modifié le classement des machines les plus puissantes dans le monde (voir le TOP500 sur http://www.top500.org/). La programmation de ce type d'architecture est devenue de plus en plus simple avec le temps et plusieurs solutions sont envisagées comme l'utilisation de :

12. Random Access Memory, ou mémoire vive en français.
13. Un système (essentiellement circuit + protocole de communication) de transfert des données.
14. pour ne pas réduire la bande passante pour chaque coeur.
15. Graphic processing unit est un coprocesseur vectoriel contenant beaucoup plus d'unités de calcul que le CPU mais son organisation mémoire ne permet pas de faire efficacement des opérations aussi complexes que sur le CPU.

– OpenCL qui est de bas niveau [16], et qui est proposé comme un langage à implémenter sur toutes les cartes.
– CUDA, langage de plus haut niveau qu'OpenCL, et qui a été conçu pour les cartes Nvidia mais qui commence à être implémenté sur d'autres.
– OpenACC (issu du OpenHMPP) est un langage à directives [17]. Son utilisation ne demande pas de réécrire le code CPU, mais seulement de chercher les parties du programme susceptibles de contenir de la parallélisation.

0.1.2 Simulation dans la finance

Dans son passionnant livre "The Big Short", Michael Lewis décrit ses derniers jours dans la finance, les années 1980, comme la période qui a vu la fin d'une espèce de financiers qui arrivaient difficilement à évaluer les contrats et les couvertures à prendre. Il reconnaît aussi, qu'à ce moment là, il n'avait pas saisi la montée en force d'une autre génération de scientifiques, les analystes quantitatifs ou Quants, capables de faire rapidement les calculs financiers nécessaires. Ceci a été le point de départ de la structuration de plusieurs types de contrats et de problèmes de plus en plus durs à résoudre. Dans une présentation à Zurich durant l'été 2011, René Carmona a expliqué qu'une partie de la complexité des problèmes qu'on traite provient des contrats susceptibles de simplifier notre positionnement dans le marché. Cette difficulté n'étant pas intrinsèque aux contrats, elle devient de plus en plus évidente avec le temps. Un exemple simple est celui de la diversification d'un portefeuille d'actions par l'achat ou la vente de *forward* sur les matières premières. La création de produits dérivés, faisant intervenir les actions et les matières premières, a créé une corrélation fictive [18] rendant plus délicat ce type de diversification.

Face à ce type de problème, les Quants qui pouvaient assez bien s'en sortir avec des formules fermées ont commencé à avoir un peu plus de difficultés à justifier leurs démarches et la simplicité de leurs modèles. Principalement, il restait la simulation par EDP car on pouvait avoir des temps de calcul raisonnables [19] pour les types de problèmes qu'on voulait traiter. Bien que peu courante, la simulation de Monte Carlo était justifiée pour certaines applications, surtout dans les marchés peu volatiles, qui permettaient une convergence assez rapide vers la solution [20]. L'utilisation de ces deux méthodes a commencé à créer une effervescence féconde

16. L'efficacité de l'implémentation dépend de notre bonne connaissance du hardware.
17. comme OpenMP.
18. La personne qui vend ce type de contrat doit se couvrir continuellement par des produits vanilles ou par l'achat ou la vente des produits sous-jacents.
19. En front office, on demande à avoir des calculs de l'ordre de la seconde.
20. Comme l'a été le marché fixed-income, dont la meilleure traduction est taux d'intérêt, avant la crise de 2008.

entre l'application et la théorie probabiliste et déterministe des EDPs. La parallélisation, quant à elle, était complètement inexistante car on n'attendait pas d'un Quant de savoir vectoriser un code mais plutôt de proposer un moyen d'émettre des prix sur des contrats, de plus en plus complexes, pour une meilleure plus-value.

Le "Time to Market" de la parallélisation en finance était assez long et il a fallu une crise financière globale pour prendre du recul par rapport à des pratiques de moins en moins justifiées. On devait non seulement avoir une couverture consistante des contrats standards mais surtout assurer une meilleure évaluation du risque. Dans un monde hautement interconnecté, ces deux activités sont devenues très difficiles et ne peuvent avoir de sens sans augmenter la dimension des problèmes traités. L'évaluation d'un contrat, sa couverture ou la détermination des risques qu'il peut engendrer est une tâche qui devient presque impossible avec des EDPs lorsque le nombre d'actifs excède 4. On revient donc à une utilisation massive des architectures parallèles à travers l'implémentation de Monte Carlo pour les problèmes financiers qui sont majoritairement paraboliques linéaires, non-linéaires, linéaires inverses et non-linéaires inverses.

La parallélisation d'une simulation de Monte Carlo pour des problèmes linéaires a été déjà traîtée sur les clusters de CPUs, par exemple [43, 41]. Cependant, sur cluster de GPUs, notre travail est parmi les premiers sur le sujet. Nous citons aussi des recherches contemporaines de l'entreprise NAG [21] et de [30]. Ensuite, en s'inspirant des travaux [19, 18] implémentés sur clusters de CPUs, nous avons initié l'étude en détail de la parallélisation massive des problèmes non-linéaires en finance. Les problèmes linéaires inverses, quant à eux, sont beaucoup plus larges. Nous nous sommes principalement investis sur une application multidimensionnelle très attractive d'un point de vue pratique qui met en évidence les difficultés que l'on peut rencontrer.

Nous ne nous sommes pas attardés sur les applications parallélisables que l'on trouve dans les banques parce que, d'une part, les méthodes spécifiques sont confidentielles en général et, d'autre part, ce que nous présentons comme problèmes englobe déjà l'essentiel de ce qu'on a en pratique.

21. Numerical Algorithms Group.

0.2 Objectif du travail

Traiter les problèmes paraboliques multidimensionnels linéaires, non-linéaires et linéaires inverses est l'objectif principal de ce travail. C'est le mot multidimensionnel qui rend pratiquement incontournable l'utilisation des méthodes de simulations fondées sur le Monte Carlo. Le mot multidimensionnel rend aussi indispensable l'utilisation des architectures parallèles. En effet, comme nous l'avons vu à la section précédente, les problèmes manipulant un large nombre d'actifs sont de grands consommateurs en temps d'exécution, et il n'y a que la parallélisation pour faire chuter ce dernier.

De ce fait, le premier objectif de notre travail consiste à proposer des générateurs de nombres aléatoires appropriés pour des architectures parallèles et massivement parallèles de clusters de CPUs/GPUs. Nous testerons le gain en temps de calcul et l'énergie consommée lors de l'implémentation du cas linéaire du pricing européen. Le deuxième objectif est de reformuler le problème non-linéaire du pricing américain pour que l'on puisse avoir des gains de parallélisation semblables à ceux obtenus pour les problèmes linéaires. La méthode proposée fondée sur le calcul de Malliavin est aussi plus avantageuse du point de vue du praticien au delà même de l'intérêt intrinsèque lié à la possibilité d'une bonne parallélisation. Toujours dans l'objectif de proposer des algorithmes paralléles, le dernier point est l'étude de l'unicité de la solution de certains cas linéaires inverses en finance. Cette unicité aide en effet à avoir des algorithmes simples fondés sur Monte Carlo.

0.3 Organisation du manuscrit

Une fois l'objectif de ce travail introduit, il est plus simple d'appréhender son organisation. Même si, d'un point de vue informatique, plusieurs recherches différentes ont été entamées en parallèle, les résultats sont eux apparus d'une manière séquentielle. Ce manuscrit est donc présenté d'une manière séquentielle, du plus simple vers le plus complexe. Les chapitres sont rédigés comme des articles de journaux. Les deux premiers vont être publiés dans "Concurrency and Computation : Practice and Experience" [3] et dans "SIAM Journal on Financial Mathematics" [1] alors que le troisième sera soumis prochainement et le dernier l'a été déjà. De ce fait, l'anglais s'est imposé naturellement comme langue de rédaction.

Chapitre 1 [3]

Cet article de journal fait la synthèse de deux articles de conférences [2, 4]. Nous détaillerons la parallélisation des générateurs de nombres aléatoires puis évaluerons le speedup et l'énergie consommée pour un problème de pricing européen sur un cluster de GPUs. Nous implémentons par la suite sur un seul GPU le pricing américain fondé sur l'algorithme de Longstaff-Schwartz. Nous évaluerons donc le speedup obtenu et nous donnerons quelques idées sur une éventuelle parallélisation de ce dernier algorithme sur un cluster de GPUs.

En plus de l'évaluation de l'avantage en temps et en énergie des architectures fondées sur GPUs, nous proposons une solution au problème de générer "efficacement" de "bons" nombres aléatoires malgré l'inexistence de mémoire cache. L'implémentation de l'algorithme de Longstaff-Schwartz utilisant les GPUs a une valeur pratique irréfutable parce qu'elle nous permet aussi de voir à quel point la partie régression détériore la parallélisation totale.

Chapitre 2 [1]

Soucieux d'avoir un meilleur speedup que lors de l'implémentation de l'algorithme de Longstaff-Schwartz [22], nous proposons ici une reformulation du problème de pricing américain en utilisant le calcul de Malliavin. L'algorithme obtenu ne fait plus appel à la communication CPU/GPU nécessaire lorsqu'on implémente une méthode à régression. En plus, la précision de la simulation n'est plus limitée par la base de régression mais seulement par le nombre de trajectoires simulées. Enfin, les résultats théoriques développés sont assez généraux et peuvent être appliqués sur une très large gamme de modèles.

Pour arriver à des résultats théoriques et numériques très intéressants, nous avons dû apporter des améliorations diverses au problème originel. La première consiste à redéployer l'outil de calcul de Malliavin pour réécrire complètement l'expression de l'espérance conditionnelle pour qu'elle puisse être généralisée à un plus grand nombre de modèles possible. Le deuxième apport est de montrer que la réduction de variance par conditionnement, de l'estimateur de l'espérance conditionnelle, est suffisante pour avoir de bons résultats de pricing. Le troisième point important repose sur l'observation que l'estimateur de l'espérance conditionnelle est biaisé [23]. Nous proposons alors une méthode pour réduire le biais total par une utilisation de nombres de trajectoires différents entre le numérateur et le dénominateur de l'estimateur. Nous nous sommes

22. ou d'autres méthodes fondées sur la régression.
23. même s'il est asymptotiquement sans biais.

aussi assurés que cette réduction de biais ne fait pas augmenter la variance de beaucoup. Enfin, nous avons donné une multitude de simulations pour conforter la précision des résultats et l'adaptabilité de la méthode sur des GPUs.

Chapitre 3

Le but de cette partie est double. D'une part, nous voulons détailler la parallélisation du Monte Carlo non-linéaire pour le pricing américain proposé au chapitre 2. D'autre part, puisque l'augmentation de la dimension rend le problème sensiblement plus complexe, nous présentons et testons une méthode de simplification qui permet de traîter le pricing américain à très haut nombre d'actifs [24]. Nous démontrons ainsi une autre dimension de la puissance de notre méthode à travers un algorithme qui passe à l'échelle [25] sur un cluster de GPUs. Nous évaluons le speedup et la réduction importante en consommation énergétique et nous justifions l'efficacité des simplifications faites à haute dimension.

Ce chapitre inclut une partie d'optimisation des différents paramètres de l'algorithme qui permettent d'avoir une implémenatation efficace. Ces derniers ne proviennent pas du Monte Carlo mais du fait que l'on traîte un problème non-linéaire qui conduit à une structure non-coalescente des données et à une limitation en nombre de threads [26] pour une partie de l'algorithme.

Chapitre 4

Dans cette partie, nous présentons une étude de la dépendance du prix de la structure de corrélation. Elle est certainement la moins concluante mais elle reflète les difficultés rencontrées lorsqu'on traîte des problèmes inverses avec des modèles autres que celui de Black et Scholes. En revanche, nous arrivons à donner des résultats asymptotiques très intéressants du point de vue du praticien et surtout à conforter la consistance de ces résultats asymptotiques par une multitude de simulations pour un très large choix de paramètres. Nous introduisons la notion de corrélation implicite, puis nous établissons nos principaux résultats de monotonicité sur un modèle de Heston bidimensionnel. Enfin, nous montrons que les résultats obtenus peuvent être généralisés pour des dimensions plus grandes et des modèles multidimensionnels dérivés du

24. Des tests présentés pour 5, 10 et 20 actifs.

25. Pour un nombre de trajectoires suffisamment grand, lorsqu'on multiplie par N le nombre de noeuds de calcul, on divise par N le temps d'exécution de notre algorithme.

26. Nombre de tâches lancées en parallèle.

Heston, comme le multidimensionnel double Heston.

En ce qui concerne le modèle de Heston ainsi que ceux qui en dérivent, des difficultés assez techniques sont à relever lorsque la condition de Feller n'est pas vérifiée. En effet, en notant κ l'intensité de retour à la moyenne, θ la volatilité long-terme, η la volatilité de la volatilité et y la volatilité initiale, la condition

$$y > 0, \quad 2\kappa\theta \geq \eta^2$$

simplifie le problème théorique, mais est très peu satisfaite en pratique. Pour garder l'intérêt appliqué du problème traîté, nous avons vérifié numériquement la monotonie et la précision de notre résultat d'approximation asymptotique lorsque

$$y > 0, \quad 4\kappa\theta > \eta^2$$

qui est assez bien vérifiée dans les marchés financiers à l'exception du marché du crédit (CDO, CDS...et autres produits).

Chapitre 0

General Introduction

> Most important of all was Fibonacci's introduction of Hindu-Arabic numerals. He not only gave Europe the decimal system, which makes all kinds of calculation far easier than with Roman numerals ; he also showed how it could be applied to commercial bookkeeping, to currency conversions and, crucially, to the calculation of interest. Significantly, many of the examples in the *Liber Abaci* are made more vivid by being expressed in terms of commodities like hides, peppers, cheese, oil and spices.
>
> Niall Ferguson "The Ascent of Money"

> La simulation numérique a aussi des liens forts avec la théorie et ça va dans les deux sens. Parfois un calcul théorique va vous donner des algorithmes beaucoup plus performants pour faire votre calcul. Parfois votre calcul va changer votre vision théorique d'un problème.
>
> Cédric Villani, durant la remise du prix Joseph Fourier 2011

0.1 The work context

The first quotation above mentions the fundamental relation in occident between mathematics and finance. Indeed, although theoretical physics is the first field to which we could think when talking about applied mathematics, history teaches us that finance is the first activity to use applied mathematics. This information has to be conditioned by the fact that mathematics in finance appears much more natural even for novices, in contrast with the use of mathematics in physics that is impossible without mastering some fundamental principles and relations. This argument is maybe the best one that justifies the historic fact and the important number

of mathematicians that are continuously interested by this application field. Indeed, for a mathematician, it is easier to have a good intuition for a financial problem than to have it for a gravitational or quantum phenomenon.

The work presented in this manuscript is an illustration of what we just said above. Indeed, this research results from a growing interest in the parallel multi-core and many-core [1] simulation and the addressed problems are taken from markets' finance. In addition to the theory and the experience, the numerical simulation has become a pillar of scientific knowledge. In this study, the parallel simulation takes an important place. The other goal of this work is to formulate or reformulate a specific financial problem so that it can be efficiently implemented on parallel architectures. As mentioned by Cédric Villani (second quotation), this double objective is powerful because it allows both to improve the numerical results and to have a better vision on the theoretical aspects.

We will use this introduction to raise the historical development of the computing architecture since 1945 and to detail the current directions that these developments take. Throughout this brief presentation, we will try also to understand the progressive changes that occurred on the theoretical advances. Afterwards, we return to the scientific application that interests us which is mathematics for markets' finance. We will analyze the causes of its fast growth during the last thirty years and see the action of simulation on the actual developments and their future orientations. We hope then to reconcile the evolution of applied mathematics, the computing tools and the financial inventions. We point out that our objective is not to give a technical presentation, but to provide a general survey on the evolution of computer architecture and the applications in financial mathematics.

0.1.1 From parallel to sequential then from sequential to parallel

Unlucky is the human being when he seeks to be replaced to reduce his efforts, because this research is already requiring a lot of it. Fortunately, this quest has not been in vain because it allowed at least to consolidate the detailed knowledge of some tasks, since no one can give a machine a job to do without mastering the different simple parts of it.

Some trace back the evolution of computers from the Chinese abacus (7 centuries BC) but,

1. said differently, parallel and massively parallel.

as announced before, we will focus exclusively on postwar electric machines [2]. Thinking of a task that can be split into elementary parts (ex: memory allocation of a variable, initialization of a variable, adding two variables) that enable non-communicative execution (independent within the computing meaning), the definition of both the parallel machine [3] and the sequential one becomes natural:

- A sequential machine is able to execute the different independent elementary parts only in a sequential fashion, ie one after the other.
- A parallel machine is able to execute at least two independent elementary parts in a parallel fashion, ie more than two at the same time.

By this definition, it is very difficult to find now a sequential machine although this was not the case, during a long period, after the commercialization of the first transistor computer. As astonishing as it could seem, the first electronic general-purpose machine (Turing-complete [11]), ENIAC, was parallel [28]. Later, a modified version of it became the first Von Neumann machine [27], that refers to a machine in which the program is also stored in the memory. In addition to this revolution, Von Neumann contributed to the development of Monte Carlo method. All this makes ENIAC the first supercomputer to execute during one year the tests of the hydrogen bomb.

Then, there is the period that begins with the Intel commercialization of the first microcomputer in 1972. Between the date of the invention of the first transistor machine (TRADIC 1954) and 1972, we had always had parallel machines because they were intended to be manipulated by specialists and generally for military applications. Democratizing computers for the general public has raised big hardware and software challenges. The hardware ones were resolved essentially thanks to a better conception of memories and to their hierarchization. Once that we provided sufficient [4] cache memory [5] and RAM [6] to the computing unit, we could execute much easier sufficiently complex tasks to interest the amateurs of new technologies.

Like in other areas, the idea of amortizing the production costs, by selling to the large public,

2. World War II.
3. This definition is an example of an abuse being done to avoid some technical issues like the existence of several types of parallelism resulting in architectures such as: Hyperthreading, FPGA
4. "Providing sufficient" includes the quantity of the memory and the speed of access to it.
5. this memory is a part of the CPU.
6. Random Access Memory: This memory is separated from the CPU.

has become inescapable in computer science. Moreover, as the human being reasons sequentially, for his first purchase, he is quite satisfied by a machine that has one core and sufficient memory for a text editor, drawing ... or watch a low resolution video. Thus, the question of increasing the number of cores on one chip was not justified because, each 18 months period, both the number of transistors on each chip and the operating frequency were doubled [7]. Subsequently, one core twice faster could be considered virtually as two cores working at the same time.

As far as the parallelization is concerned, between 1980 and 2000, the scientific user was almost constrained to use connected sequential machines. This has not been without pain for the scientific community that was very disadvantaged by the multiplicity of platforms, by an inefficient inter-machine communication and insufficient documentation. Faced to these difficulties, applied mathematics has seen its expansion in the world of sequential resolution of PDEs [8], promoting highly sequential iterative methods. A striking standard example of this sequential orientation of theoretical research is given by matrices multiplication: In a parallel implementation and with small cache memory for each core, it is simple to separate the multiplications done by each core independently from the others so that the execution time is divided by the total number of cores. However, in a sequential PDE-like implementation, first we try to manipulate matrices which are as sparse as possible, then implement a multiplication algorithm that intensely uses the big quantity of the cache available when we use only one core [56].

This orientation in computer architecture and theoretical developments lasted twenty years, until the moment that we reached the limit number of transistors that can be included in one core. According to Figure 1, one can see that from a certain frequency, the electric operating power has become unsustainable to justify a monocore conception of CPUs [9]. The limit of this architecture is due not only to the fact that the operating power is linearly proportional to the frequency [10], but also to the faced problems to realize thinner printed circuits. To ensure that the Moore's Law remains fulfilled, the direct solution was to increase the number of cores on a single chip. This worked well for two and four cores and has led to the ascent of the number of cores in supercomputers (see Figure 2).

7. Moore's Law.
8. Partial Differential Equations.
9. Central Processing Unit: The principal computer processor that includes some computing units and cache memory.
10. $P = CV^2 f$ where C is the capacity, V the potential and f the frequency. Even though the total power of the motherboard is no-linearly related to the frequency.

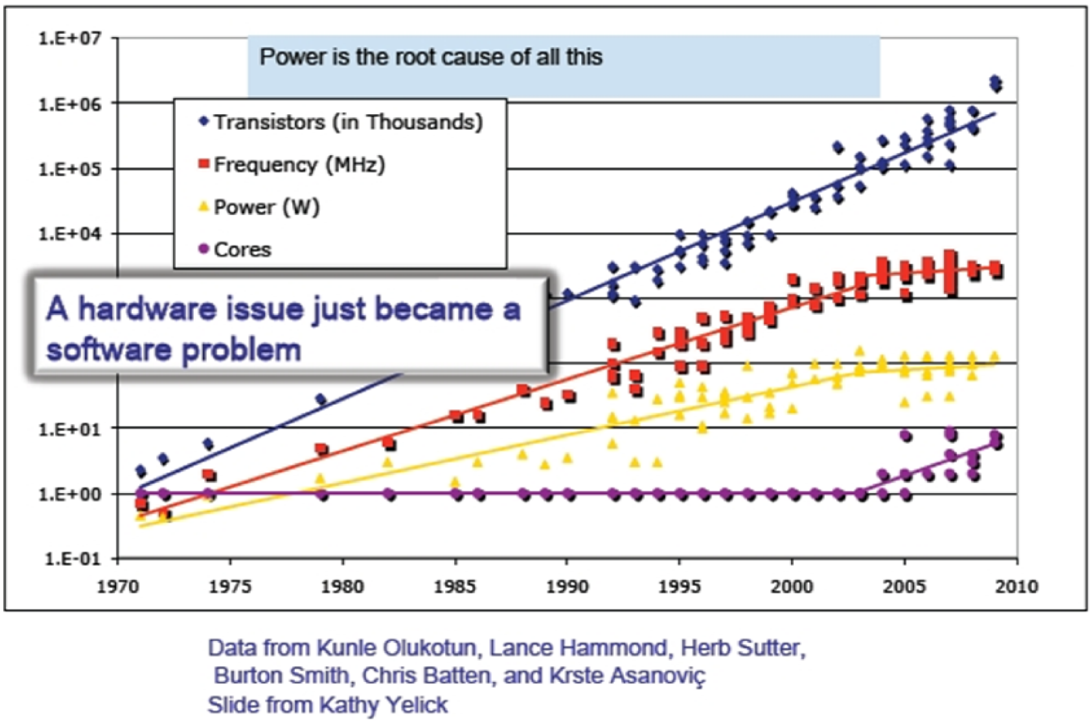

FIGURE 1 – Historic evolution on one CPU of: cores number, transistors number, operating frequency and operating power.

Then, the same memory access problem has become, one more time, an important hardware issue. In fact, the cache memory represents about 80% of the CPU and increasing the number of cores reduces considerably the amount of cache available for each core. This is also true for the RAM that is used by multiple cores. Consequently, we should also increase the number of buses [11] to access this memory [12]. The solution to these problems is based on more localized memory and uses both the code parallelization on different cores and proposing architectures that contain much more than two or four cores. Once more, the commercialization to the large public dictates the rhythm, because it is the GPUs [13], video game processors, that constitute a serious solution to massively parallel applications. So much true that the introduction of GPUs in supercomputers has completely changed the classification of the most powerful machines in the world (see TOP500 on http://www.top500.org/). Over the time, the programming of this

11. It is a system (essentially a printed circuit + communication protocol) to transfer data.
12. in order to keep a good bandwidth for each core.
13. Graphic processing unit: It is a vectorial coprocessor including much more computing units than the CPU, but whose memory organization does not allow to implement efficiently as complex operations as the one that can be performed by the CPU.

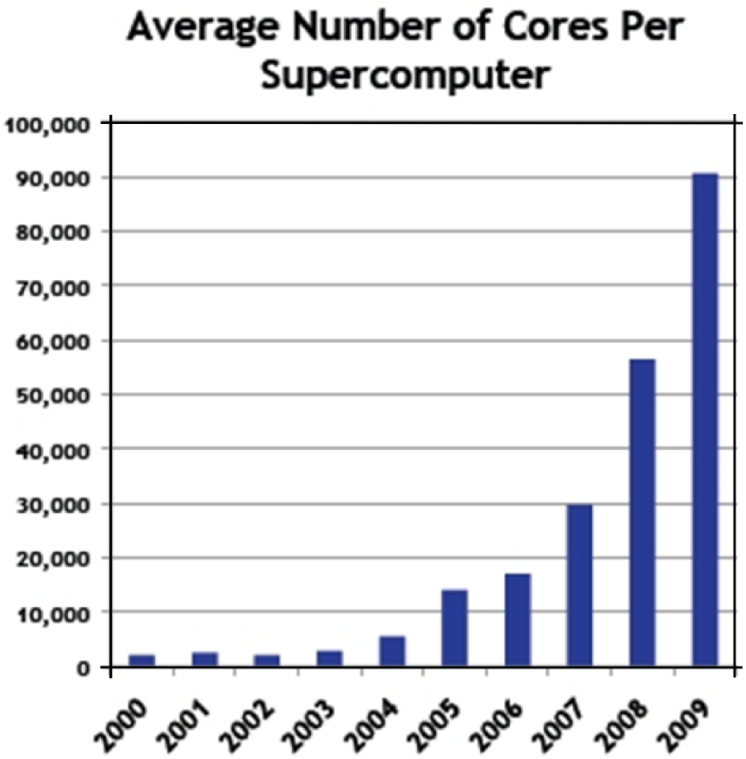

FIGURE 2 – Exponential growth of cores number due to a parallelization need.

type of architecture has become increasingly simple and several solutions can be considered, as the use of:

- OpenCL: A low level language [14], that is proposed as a language that can be implemented on all cards.
- CUDA: Less low level than OpenCL, which was dedicated to Nvidia cards but it started to be implemented on the others.
- OpenACC (came from OpenHMPP): Is a directives [15] language. Its use does not require to rewrite the CPU code, but only to localize the parts that are likely to contain parallelization.

0.1.2 Simulation in finance

In his amazing book "The Big Short", Michael Lewis describes his last days in finance, the 80s, as the period which saw the end of a generation of bankers that had difficulties in evaluating and hedging contracts. He also recognizes that he did not catch the strong ascent of another

14. The efficiency depends on our good knowledge of the hardware.
15. like OpenMP.

generation of scientists, the quantitative analysts or Quants, able to easily make the financial computations. This was the starting point of the structuring of various kinds of contracts which led to increasingly difficult problems. During a summer school in 2011, in his presentation in Zurich, René Carmona explained that a part of the financial problems complexity comes from the contracts that are supposed to simplify our market positioning. This difficulty is not intrinsic to the contracts, but becomes evident over the time. A simple example is the diversification of a portfolio of stocks by buying or selling forwards on commodities. Using derivative products, involving stocks and commodities, created fictitious correlation [16] leading to a more complex diversification.

Faced to this type of problems, Quants that were used to closed formulas started to confront bigger difficulties to justify their approach and the simplicity of the model used. For the problems that we wanted to solve, PDE simulation was a real alternative because it can be performed within a reasonable time [17]. Although less frequent, the Monte Carlo simulation was justified for some applications, mostly in the least volatile markets like the fixed-income, that allow sufficiently quick convergence to the solution. The use of these two methods created a fertile effervescence between the application and both the probabilistic and deterministic theories. Regarding parallelization, it was completely non-existent as we did not request from a Quant to vectorize a code but rather to find a way to give a price on increasingly complex contracts (which provide better gains).

The parallelization "Time to Market" in finance took a long period, a global financial crisis was necessary to have a better vision on some practices that are no more justified. We are not only obliged to have a good hedge of all the standard contracts but mostly ensure a better risk evaluation. In a highly interconnected world, the latter activities become extremely difficult and cannot have any sense if we do not increase the dimension of the problems. When the number of assets involved is bigger than 4, the evaluation of a contract, its hedge or the risks that it produces is an almost impossible task using PDEs. We return then to the Monte Carlo implementation using massively parallel architectures for financial problems that are generally parabolic either linear and nonlinear or linear inverse and nonlinear inverse.

Regarding linear problems, the Monte Carlo parallelization was already explored for CPU

16. The person who sells this kind of contract should continually hedge himself by vanilla products or by buying and selling the underlying assets.

17. In front office, we request simulations that do not take more than few seconds.

clusters, we refer the reader for example to [43, 41]. However, our work was among the first to do this parallel implementation on a GPU cluster, we cite also the NAG [18] contemporary research and the reference [30]. Then, inspired from the CPU cluster pricing in [19, 18], we have initiated the detailed study of the massively parallel implementation of nonlinear problems in finance. Besides, the category of inverse problems is very large. From a practitioner point of view, we have focused mainly on a very attractive multidimensional application that also shows the kind of theoretical difficulties that we can encounter.

We did not linger on the parallel implementation of banks' applications because, on the one hand, the specific methods are confidential. On the other hand, the presented problems already include the essential ideas found in practice.

0.2 The work objective

Handling multidimensional parabolic linear, nonlinear and linear inverse problems is the main objective of this work. It is the multidimensional word that makes virtually inevitable the use of simulation methods based on Monte Carlo. This word also makes necessary the use of parallel architectures. Indeed, as we have already seen in the previous section, the problems dealing with a large number of assets are major resources consumers, and only parallelization is able to reduce their execution times.

Consequently, the first goal of our work is to propose "appropriate" random number generators to parallel and massively parallel architecture implemented on CPUs/GPUs cluster. We quantify the speedup and the energy consumption of the parallel execution of a European pricing. The second objective is to reformulate the nonlinear problem of pricing American options in order to get the same parallelization gains as those obtained for linear problems. In addition to its parallelization suitability, the proposed method based on Malliavin calculus has other practical advantages. Continuing with parallel algorithms, the last point of this work is dedicated to the uniqueness of the solution of some linear inverse problems in finance. This theoretical study enables the use of simple methods based on Monte Carlo.

Once the objective introduced, it is easier to understand the organization of the manuscript. Although various research were conducted in parallel, the results were obtained sequentially.

18. Numerical Algorithms Group.

This dissertation is also presented sequentially from the simplest to the most complex problem. The chapters have been written like journal papers. Chapters 1 & 2 will be published in "Concurrency and Computation: Practice and Experience" [3] and in "SIAM Journal on Financial Mathematics" [1] while Chapter 4 has been already submitted and Chapter 3 will be submitted shortly. Thus, the English language was indispensable for the redaction. Moreover, in each chapter, the reader will find an extensive introduction for each studied subject.

Chapitre 1

From linear to nonlinear simulation on GPUs [3]

Abstract

This paper is about using the existing Monte Carlo approach for pricing European and American contracts on a state-of-art GPU architecture. First we adapt on a cluster of GPUs two different suitable paradigms of parallelizing random number generators which were developed for CPU clusters'. Since in financial applications we request results within seconds of simulation, the sufficiently large computations should be implemented on a cluster of machines. Thus, we make the European contract comparison between CPUs and GPUs using from 1 up to 16 nodes of a CPU/GPU cluster. We show that using GPUs for European contracts reduces the execution time by ~ 40 and diminishes the energy consumed by ~ 50 during the simulation. In the second set of experiments, we investigate the benefits of using GPUs' parallelization for pricing American options that requires solving an optimal stopping problem and which we implement using the Longstaff and Schwartz regression method. The speedup result obtained for American options varies between 2 and 10 according to the number of generated paths, the dimension and the time discretization.

1.1 Introduction and objectives

Monte Carlo (MC) simulation, the most widely used method in transport problems, owes its popularity in the scientific community to its three features: (1) the possibility to use MC for

complex transport problems that cannot be interpreted in deterministic language, (2) the ease of implementation and parallelization, and (3) contrary to deterministic methods such as finite element or finite difference methods, MC remains efficient in a dimension greater than 4 which is appropriate for systems requiring high degrees of freedom.

In this article, it is shown that although MC is theoretically very efficient for multi-core architectures, the methods based on Monte Carlo vary according to their effectiveness on these architectures. In this work, we will present the practical point of view of the pricing methods based on Monte Carlo and implemented on GPUs. This practical study will provide the comparison between CPUs and GPUs on pricing the two major derivative classes found in the financial field which are European contracts (ECs) and American contracts (ACs). As in practice, one multi-core card is generally insufficient for the execution of high-dimensional applications within seconds, we will compare on the same cluster multi-core GPUs with four core CPUs for pricing ECs. Moreover, we will explain how we can generalize this kind of cluster comparison for pricing ACs.

After the introduction of MC and its applications for pricing ECs and ACs in the second section, in the third section we will present two different methods of parallelizing random number generation that aim at the highest adaptability on GPUs and we will give an example of each method. In the fourth section, we give details on the implementation of a typical multidimensional EC on a multi-core CPU/GPU cluster. The fifth section will present a detailed study of the accuracy of the results, the speedups and the energy consumed during the simulation of ECs. Once the concept of pricing parallelization on ECs is understood through section four and five, we devote the final sections to pricing ACs which is known as one of the most challenging problems in financial applications. Thus, in section six and seven, we aim at reducing the running time of ACs simulation using GPUs and we will propose means of parallelizing it on a cluster of machines.

Before going into the detail of this work, the main specifications of the machines on which we implement our benchmark applications are as follows:

M1: is the XPS M1730 laptop composed of Intel Duo Core CPU with a clock rate of 2.50GHz and contains 2 nVIDIA 8800M GTX connected with SLI.

M2: is a cluster of 16 nodes. Each node is a PC composed of an Intel Nehalem CPU, with 4 hyperthreaded cores at 2.67GHz, and a nVIDIA GTX285 GPU with 1GB of memory. This cluster has a Gigabit Ethernet interconnection network built around a small DELL Power Object 5324 switch (with 24 ports). Energy consumption of each node is monitored by a Raritan DPXS20A-16 device that continuously measures the electric power consumption (in Watts) up to 20 nodes (in Watts). Then a Perl script samples these values and computes

the energy (Joules or WattHours) consumed by the computation on each node and on the complete cluster (including the interconnection switch).

1.2 Monte Carlo and multi-core programming

This section is divided into two parts: the first part goes over the general aspects of parallelizing MC simulations and the benchmark model used. The second part gives some details on pricing ECs and ACs. Indeed, in Markovian models, pricing ACs basically adds one step to the pricing algorithm. Thus, based on what is known on ECs, we will present the problem of pricing ACs.

1.2.1 An introduction to Monte Carlo methods

The general MC method is articulated by two theorems that constitute the two pillars of the probability theory [25]. The first one is the Strong Law of Large Numbers (SLLN) that announces the convergence of a certain series of independent random variables that have the same distribution to a value of an integral. The second one is the Central Limit Theorem (CLT) which determines the speed of the convergence revealed by SLLN. These two classic theorems can be found, for instance, in [60].

TABLE 1.1 – Contracts and associated payoffs

Name of contracts	Payoffs
Put	$(K - S_T(\varepsilon))_+$
Call	$(S_T(\varepsilon) - K)_+$
Lookback	$(max_T S_t(\varepsilon) - S_T(\varepsilon))$
Up and Out Barrier	$(f(S_T(\varepsilon))1_{max_T S_t(\varepsilon)<L})$
Floating Asian Put	$(mean_T S_t(\varepsilon) - S_T(\varepsilon))_+$

A video game processor, the graphic processing unit (GPU) becomes a serious programming device to massively parallel applications. The assumption that makes MC more attractive than other methods for GPUs is the independence of the random variables. The main concern of using MC on GPUs is how to spread this independence on the stream processor units. Unlike pricing ACs, pricing ECs with MC is no more than using the result of SLLN and CLT on random functions such as those presented in Table 1.1. In Table 1.1, $(x)_+ = max(x, 0)$ and max_T, $mean_T$ respectively stand for the maximum value and the average value on the trajectory of the stock $S_t(\varepsilon)$ on the time interval $[0, T]$. On the one hand, in this article, we suppose that

$\varepsilon = (\varepsilon^1, ..., \varepsilon^d)$ is a Gaussian vector and the coordinates $\varepsilon^1, ..., \varepsilon^d$ are independent. On the other hand, we denote by S_t the price of a basket of stocks $S_t^1, ..., S_t^d$ and to describe the behavior of each stock i, we will use the following standard geometric Brownian motion

$$S_t^i = S_0^i \exp\left[\left(r_i - d_i - \frac{\sigma_i^2}{2}\sum_{k=1}^{i}\rho_{ik}^2\right)t + \sigma_i \sum_{k=1}^{i}\rho_{ik}W_t^k\right] \tag{1.1}$$

which is **equal in density** to

$$S_t^i = S_0^i \exp\left[\left(r_i - d_i - \frac{\sigma_i^2}{2}\sum_{k=1}^{i}\rho_{ik}^2\right)t + \sigma_i \sum_{k=1}^{i}\rho_{ik}\sqrt{t}\varepsilon^k\right]$$

where:

S_0^i is the initial price of the asset i,
r_i is the rate of the asset i,
d_i is the dividend of the asset i,
σ_i is the volatility of the asset i,
$\sqrt{t}\varepsilon^k$ simulates the Brownian motion distribution W_t^k,
$(\rho_{ik})_{1\leq i,k\leq d}$ is a given matrix correlating the assets.

Thus, the first stage of using MC is to simulate the Gaussian distribution of ε through a set of samples ε_i. For a detailed presentation on MC in financial applications, we refer the reader to [25]. In order to parallelize pricing ECs, we implement algorithms that can be executed similarly on all the trajectories at the same time. With MC methods, the best way to perform this similarity task is to discretize the time interval then we run the same tasks sequentially for the whole current trajectories at the same step of the discretization. For instance, we can simulate the log-normal evolution of the stock $S_t^i(\varepsilon)$ at each time $t_k \in [0, T]$ using the two following steps:

1°) The simulation of normal distribution variable ε_q associated with the trajectory q.

2°) The actualization of the stock value using the recurrence relation

$$S_{t_k}^{iq} = S_{t_{k-1}}^{iq} \exp(f(\varepsilon_q))$$

where f is an affine function.

In the example above also demonstrated in Figure 1.1, we carry out the two steps sequentially. The parallelization takes part in performing the same step on different trajectories. Thus a subset of the whole set of trajectories can be associated with one processor unit and carry out each step independently from the other subsets. Moreover, parallelizing the simulation on a cluster of multi-cores CPUs/GPUs is no more than parallelizing or enlarging the set of trajectories to add all the contributions of the different machines.

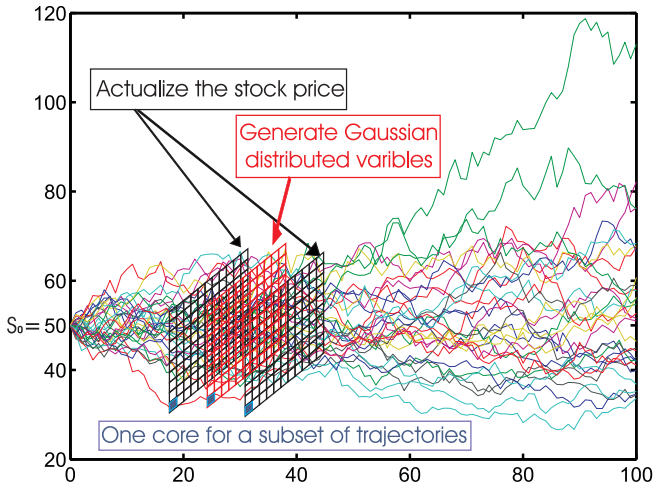

FIGURE 1.1 – Parallelizing the same task on different trajectories

1.2.2 Pricing European and American options

A European contract is one that can be exerciced only the maturity T, unlike American contract which can be exerciced anytime before the maturity T. Among ECs and ACs, the **"options"** contracts are ones that are the most studied. The option payoff is generally given using the function $(x)_+ = max(x, 0)$ which expresses the fact that options are contracts which **allow, without obligation,** the holder to buy or to sell an asset at a fixed price. For example, the put and call contracts given in Table 1.1 are options.

If r is the risk neutral rate and $\Phi(S_t)$ the payoff of a given contract, the price of a European version of this contract, at each time $t \in [0, T]$, is defined by the following expression

$$P_t^{Euro}(x) = \mathbb{E}_{t,x}\left(e^{-r(T-t)}\Phi(S_T)\right), \tag{1.2}$$

where $\mathbb{E}_{t,x}$ is the expectation associated to the risk neutral probability knowing that $S_t = x$.

Using the previous notations, American contracts can be exercised at any trading date until maturity and their prices are given, at each time t, by

$$P_t^{Amer}(x) = \sup_{\theta \in \mathcal{T}_{t,T}} \mathbb{E}_{t,x}\left(e^{-r(\theta-t)}\Phi(S_\theta)\right), \tag{1.3}$$

where $\mathcal{T}_{t,T}$ is the set of stopping times in the time interval $[t, T]$.

To simulate (1.3), we first need to approach stopping times in $\mathcal{T}_{t,T}$ with stopping times taking values in the finite set $t = t_0 < t_1 < ... < t_n = T$. When we do this approximation and use the dynamic programming principle [25], we obtain the following induction for each simulated

path

$$\begin{aligned} P_T^{Amer}(S_T) &= \Phi(S_T), \quad \forall k \in \{n-1, ..., 0\}, \\ P_{t_k}^{Amer}(S_{t_k}) &= \max\{\Phi(S_{t_k}), C(S_{t_k})\} \end{aligned} \tag{1.4}$$

$C(S_{t_k})$ in (1.4) represents the continuation value and is given by

$$C(S_{t_k}) = \mathbb{E}\left(e^{-r(t_{k+1}-t_k)} P_{t_{k+1}}(S_{t_{k+1}}) \Big| S_{t_k}\right). \tag{1.5}$$

Thus, to evaluate the price of (1.3), we need to estimate $C(S_{t_k})$. Longstaff and Schwartz consider the stopping times formulation of (1.4) which allows them to reduce the bias by using the actual realized cash flows. We refer the reader to [16] for a formal presentation of the LSR algorithm.

Algorithms devoted to American pricing and based on Monte Carlo, differ essentially on the way they estimate and use the continuation value (1.5). For example the authors of [57] perform a regression to estimate the continuation value, but unlike [42], they use $C(S_{t_k})$ instead of the actual realized cash flows to update the price. Other methods use Malliavin Calculus [7] or quantization methods [8] for $C(S_{t_k})$ estimation. In addition to these methods based on MC, there is a profusion of algorithms for American option pricing. However, the one that is gaining widespread adoption in the financial industry is the Longstaff and Schwartz Regression (LSR) method. This widespread adoption and the fact that LSR is based on Monte Carlo simulation leads us to choose LSR implementation on GPU.

The LSR method approximates the continuation value by projecting the cash flow $P_{t_{k+1}}(S_{t_{k+1}})$, generated at t_{k+1}, on a set of functions $\psi(S_{t_k})$ that depend only on the asset price at time t_k. However, contrary to an ordinary regression method, the LSR uses the drawings satisfying $\Phi(S_{t_k}) > 0$, the "in the money" drawings. Even if Longstaff and Schwartz give partial convergence results in their original paper [42], the authors of [16] proved the convergence and analyzed the speed of this convergence according to the number of simulated paths. This convergence analysis has been refined in [26] by studying the problems due to the degree of regression.

Strictly speaking, when $r = 0$, if we consider the regression vector A and we denote by $\overline{C}(S_{t_k}) = A^t \psi(S_{t_k})$ the estimated continuation value, one must find the vector A that minimizes the quadratic error

$$\left|\left| P_{t_{k+1}}(S_{t_{k+1}}) - \overline{C}(S_{t_k}) \right|\right|_{L^2}. \tag{1.6}$$

We can easily check that the regression vector that minimizes (1.6) is given by

$$A = \Psi^{-1}\mathbb{E}\left(P_{t_{k+1}}(S_{t_{k+1}})\psi(S_{t_k})\right), \tag{1.7}$$

where $\Psi = \mathbb{E}\left(\psi(S_{t_k})\psi^t(S_{t_k})\right)$.

Consequently, once we approximate the expectations in (1.7) by an arithmetic average using Monte Carlo, we must invert the matrix Ψ. One of the most used and most stable methods of inversion is the one based on a Singular Value Decomposition (SVD) [53]. However, this method and other methods of inversion are not efficient to parallelize on GPUs for relatively small and not sparse matrices. In the sixth section, we will explain how the GPU implementation can be used to slightly (x1.2 to x1.4) accelerate this part of the algorithm.

Without loss of generality, we use the basis $\psi(S_{t_k})$ of monomial functions to perform our regression. Also, in the case of geometric Brownian motion, the convergence study given in [26] shows that the number of polynomials $K = K_N$ for which accurate estimation is possible from N paths is $O(\sqrt{\log(N)})$. Consequently, we use monomials of degrees less than or equal to 2 for one dimension and we will use affine regression in the multidimensional simulation. Finally, we subdivide the algorithm of pricing ACs in three parts as in [15]:

1°) Paths Generation (PG) phase.

2°) Regression (REG) phase.

3°) Pricing (PRC) phase.

As the "Calibration phase" [15] can be a source of confusion with the model calibration activity in finance, we preferred to rename it by the "Regression phase".

1.3 Parallel RNG for SIMD architecture

The parallelization on the GPU of Random Number Generation (RNG) is essential in GPU implementation of MC. As a matter of fact, the GPU programmer must reduce the CPU/GPU communication if he aims at a good speedup. Indeed, although we can simulate random numbers on the CPU parallel to execute other tasks on the GPU, the communication time CPU/GPU makes this solution less efficient. We also have the same communication time problem when using True Random Number Generators (TRNG), this is why we adopt the traditional solution of using Pseudo Random Number Generators as RNGs instead of using TRNGs.

In RNG literature, we find considerable work on sequential RNGs but much less on parallel RNGs. The authors of [52] use the Mersenne Twister (MT) generator [44] even though this generator is relatively slow on GPUs. Indeed, as it is already mentioned in [30], because the

cache memory inexists on the GPU, MT presents problems due to the multiple accesses per generator and thus per thread to the global RAM in order to serially update the large state needed by MT. The two paradigms of generating random numbers that we are going to use are suited to the GPU architecture and generally to architectures that do not possess a large cache memory. The first one is based on period splitting and the second one is based on parametrization which is used in the SPRNG [43] library and which is also recommended by the authors of MT in [45]. For each parallelization method we will give an example and we will compare, at the end of the second subsection, these two examples to the optimized implementation of MT and the Niederreiter Quasirandom (NQ) generator given in [61]. Since our work was implemented on GPU cards that basically compute in single precision, the two examples of random generators that we are going to present in the next subsections are based on single precision. However the reader can easily extend our constructions to the double precision cards.

Our first goal is to have an efficient random number generator for GPU architecture which also provides sufficient good results. Thus, in the following, two methods that are proven to be sufficiently good on CPU clusters are adapted on GPU and GPU clusters.

1.3.1 Parallel-RNG from period splitting of one RNG

The simplest theoretical solution to parallelizing random number generator (RNG) is to split the period of a good sequential one into different random number streams. On the one hand, because we are going to split the whole period, we need to have a long one to split. For example, we cannot split a period of a standard $\sim 2^{31}$ LCG because it considerably reduces the period of each stream. On the other hand, because we use the RAM memory of the GPU, we should limit the parameters of the RNG in order to reduce the access of each thread to this memory. To explain the method of period splitting, we are going to take the example of an RNG whose random behavior had already been studied in [40], that has a long period to split and relatively few parameters. This RNG is the Combined Multiple Recursive Generator (CMRG) given in example 4 of [40] and it is obtained from a judicious combination of two MRGs and each MRG has the following general expression

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3}\ mod(M)$$

The main goal of combining two MRGs is to reduce the memory storage of the past values without really compromising the quality of the random numbers. To define the different streams of CMRG, we determine the number of these streams [1], then we compute the power of the

1. which is for instance equal to the number of trajectories simulated or to the number of processors involved in GPUs

companion matrices associated to the recurrence of CMRG which allows us to initialize the different streams at the different points of the period. Also the length of the streams should be chosen carefully so that a vector formed by the first number from each stream, for example, should have relatively independent coordinates. For further details, we refer the reader to [41].

Because splitting the period of CMRG implies the computation of huge [2] powers of 3×3 matrices, the operation of launching MC on an increasing number of machines can consume a considerable amount of time. As a result, even though computing the powers of matrices uses the efficient divide-and-conquer algorithm [36], we should precompute the jump-ahead matrices once and for all. Thus, the best way of implementing CMRG is:

- to fix the maximum of streams associated with the maximum of multi-cores used,
- then compute the matrices of transition between streams only once before launching the application.

For instance, let us consider the companion matrices of the CMRG given in example 4 of [40]. Each matrix is associated with one MRG from the combination:

$$A_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -183326 & 63308 & 0 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -539608 & 0 & 86098 \end{pmatrix}$$

The period of the matrix A_1 is $p_1 = m_1^3 - 1$ and the period of the matrix A_2 is $p_2 = m_2^3 - 1$, which means:

$$A_1^{p_1+1} \, mod \, m_1 = A_1 \, mod \, m_1, \quad A_2^{p_2+1} \, mod \, m_2 = A_2 \, mod \, m_2$$

For example, if we want:

- to associate an RNG stream with each trajectory,
- to perform the evolution of about 2^{18} trajectories by each multi-core GPU,
- to use a maximum of 16 GPUs.

We divide the total period of the CMRG $(p_1 \times p_2)/2 \sim 2^{205}$ by $2^{18} \times 16 = 2^{22}$ to obtain $InitPower = 2^{205}/2^{22} = 2^{183}$ and perform the powers:

$$A_1^{init} = A_1^{InitPower} \, mod \, m_1, \quad A_2^{init} = A_2^{InitPower} \, mod \, m_2$$

As shown in Figure 1.2, if we initialize the stream 0 with the seed vectors: $X_1^0 = (x_1^1, x_1^2, x_1^3)^T$ for the first MRG and $X_2^0 = (x_2^1, x_2^2, x_2^3)^T$ for the second MRG of the combination, the seed

2. proportional to the length of the period

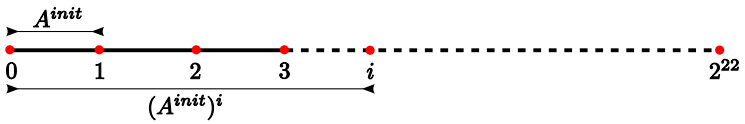

FIGURE 1.2 – Splitting the period of CMRG

TABLE 1.2 – Comparison of the effectiveness of RNGs on **M1**

Name of the RNG	PLCG	CMRG	MT	NQ
Mega Samples generated per second	16.33	3.30	1.86	1.21

values associated to the i^{th} stream are: $X_1^i = (A_1^{init})^i \times X_1^0 \, mod \, m_1$ for the first MRG and $X_2^i = (A_2^{init})^i \times X_2^0 \, mod \, m_2$ for the second MRG of the combination. Finally the algorithm of the CMRG on a single precision architecture is detailed in full in Figure 1, page 12 of [40].

1.3.2 Parallel-RNG from parameterization of RNGs

As mentioned by the authors of the Mersenne Twister RNG in [45], an acceptable way to parallelize RNGs is to parameterize them. Besides, if we consider the seeds of the RNGs as the parameters of RNGs, the method based on period splitting can also be regarded as a parametrization of these RNGs. In this article, we prefer to separate the two methods and to concentrate on the parametrizations given in [43]. One of the generators that is really efficient in implementing on double precision GPUs is the parameterized prime modulus LCG $(2^{61} - 1)$ that allows us to specify each RNG with only one parameter which is the multiplier "a" of (1.8). According to [43], this parameterization provides about 2^{58} streams. The prime modulus LCG $(2^{61} - 1)$ is based on the relation

$$x_n = a x_{n-1} \, mod (2^{61} - 1) \tag{1.8}$$

In [4] we use a parameterized prime modulus LCG $(2^{31} - 1)$ which is a single precision version of (1.8) and we implement it on single precision GPUs to compare two clusters of GPUs and CPUs. Because of its short period and its random behavior, the LCG $(2^{31} - 1)$ should be taken as a benchmark and not used for standardized applications. In Table 1.2 we compare the effectiveness of an optimized implementation of MT and NQ given in [61] with our sufficiently optimized implementation of the CMRG detailed in the previous subsection and the Parameterized LCG (PLCG) $(2^{31} - 1)$. The results presented in Table 1.2 are obtained by averaging on various simulations performed on the GPU of **M1**.

Even if NQ is not an RNG but a quasirandom generator which is based on a different theory from the RNG one, we consider it interesting to compare its effectiveness with RNGs. According to Table 1.2, we remark that CMRG is about 1.8 times faster than MT and that PLCG is

about 5 times faster than CMRG. Nevertheless, in order to be more confident about the quality of the random numbers, **we will use the CMRG in our next applications**.

1.4 Multi-paradigm parallel algorithm and implementation

1.4.1 Support application

In order to explore the effectiveness of pricing ECs on a cluster of GPUs, we are going to process a typical high dimensional EC whose price depends on the whole simulation of the trajectories of the stocks' prices (*path-dependent contracts*). Here, we take the example of a homogenous Asian option in 40 dimensions, this means that our contract is an Asian option on a homogenous weighting basket of 40 stocks. We can find this kind of contract, for instance, when managing the CAC 40 index. In the financial markets, we can find other contracts on high dimensional indices like S&P 500, DAX 30, FTSE 100. The procedure that we are going to illustrate can be easily generalized for all European path-dependent contracts like the lookback or barrier options whose payoffs are given in Table 1.1.

The Asian option is a contract whose price depends on the trajectory average. We compute the price of a floating Asian call option using:

$$E\left[e^{-rT}(S_T(\varepsilon) - \overline{S}_T)_+\right] \tag{1.9}$$

$$\overline{S}_T = mean_{0\leq t\leq T} S_t(\varepsilon) \tag{1.10}$$

In expressions (1.9) and (1.10), S_T represents the price of a homogenous weighting basket of 40 stocks at anytime T: $S_T = \frac{1}{40}.\sum_{i=1}^{40} S_T^i$. Each stock has the log-normal distribution given in (1.1). Besides, according to (1.10) $\overline{S}_T$ represents the average price of S during the life time of the contract. The third step of Algorithm 1 introduces a recursive method for computing this average price. In Algorithm 1 the exterior time loop is used for time discretization and in our application we take $\delta t = T/100$. Inside the time loop, we put another loop associated to the number of stocks S^i that take part in the pricing problem. The loops on trajectories are those that we parallelize on the different stream processor units.

The third step of Algorithm 1 uses the well known rectangle approximation of an integral. However in order to have a faster convergence, we use in the implemented version the trapezoidal approximation which is presented in [39] and characterized by the same implementation ease as the rectangular one.

Algorithm 1: 40 Dimension Floating Asian Call implemented on either CPU or GPU

Input: Model parameters and CMRG initialization
Output: $Call_{Asian} = \mathbb{E}\left(e^{-rT}(S_T(\varepsilon) - \overline{S}_T(\varepsilon))_+\right)$

for $t \in \{\delta t, 2\delta t, \ldots, T\}$ **do**
  for $i \in \{1, \ldots 40\}$ **do**
    for *each trajectory* $k \in \{1, 2, \ldots, N\}$ **do**
      /* First step: generating a normal distributed variable using CMRG and a distribution transformation as a Box-Muller one */
      $u_k \longleftarrow CMRG$;
      $\varepsilon_k \longleftarrow Box - Muller(u_k)$;
    end
    for *each trajectory* $k \in \{1, 2, \ldots, N\}$ **do**
      /* Second step: price actualization during the discretized time interval $[0, T]$ */

$$S_t^i(\varepsilon_k) = S_{t-\delta t}^i \exp\left[\left(r - d_i - \frac{\sigma_i^2}{2}\sum_{k=1}^{i}\rho_{ik}^2\right)\delta t + \sigma_i \sum_{k=1}^{i}\rho_{ik}\varepsilon_k\sqrt{\delta t}\right]$$

    end
    for *each trajectory* $k \in \{1, 2, \ldots, N\}$ **do**
      /* Third step: recursive implementation of the trajectory average using rectangle approximation */
      $\overline{S}_t^i(\varepsilon_k) \longleftarrow ((t - \delta t)/t)\overline{S}_{t-\delta t}^i(\varepsilon_k) + (1/t)S_t^i(\varepsilon_k)$;
    end
  end
end
for *each trajectory* $k \in \{1, 2, \ldots, N\}$ **do**
  /* Fourth step: a homogeneous weighing of 40 stocks */
  $\overline{S}_T(\varepsilon_k) = \frac{1}{40}.\sum_{i=1}^{40}\overline{S}_T^i(\varepsilon_k)$;
  $S_T(\varepsilon_k) = \frac{1}{40}.\sum_{i=1}^{40}S_T^i(\varepsilon_k)$;
end

$$Call_{Asian} \longleftarrow \frac{1}{N}\sum_{k=1}^{N}\left(e^{-rT}(S_T(\varepsilon_k) - \overline{S}_T(\varepsilon_k))_+\right)$$

In order to take advantage of various and heterogeneous architectures like multi-core CPUs, GPUs, CPUs cluster and GPUs cluster, we have designed a multi-paradigm parallelization of our option pricer. First, a coarse grained parallelization splits the problem in P_N big tasks (one per processing node), communicating by message-passing. Second, a fine grained parallelization splits each big task into some threads on a multi-core CPU, or in many light-threads on a GPU, communicating through a shared memory.

Input data files are read on processing node 0, and input data are *broadcast* to all other nodes. Then each node locally achieves its initializations, function of the common input data and its node number. Some of these initializations have been parallelized at fine-grained level, and the parallel CMRG RNG is initialized on each node according to the specifications of section 1.3.

Afterwards each node processes its subset of MC trajectories, using its fine grained level of parallelism. This is an *embarrassingly parallel* computing step, without any communications. Then each node computes the sum of its computed prices and the sum of its square prices. All nodes participate in a global *reduction* of these P_N pairs of results: at the end of this step the global sum of prices and global sum of square prices are available on node 0. Finally, node 0 computes the final price of the option and the associated error, and prints these results.

Broadcast and *reduction* are classic communication routines, efficiently implemented in the standard MPI communication library [49] that we used. Conversely, reading some input files concurrently from many nodes is not always supported by a file system. So, we prefer to read input files from node 0 and to broadcast data to other nodes using an MPI routine. This strategy is highly portable and scalable.

1.4.2 Fine grained parallelization on the CPU and the GPU

The implementation on multi-core CPU clusters **M2** has been achieved using both MPI, to create one process per node and to insure the few inter-node communications, and OpenMP to create several threads per core and take advantage of each available core. The OpenMP parallelization has been optimized to create the required threads (inside a large parallel region) only once, and to load balance the work among these threads. Inside each thread, data storage and data accesses are implemented in order to optimize cache memory usage. GPU implementation : Again, MPI is used to create one process per node, to distribute data and computations on the cluster and to collect results, while CUDA is used to send data and Monte Carlo trajectory computations on the GPU of each node. In order to avoid frequent data transfers between CPU and GPU, we have ported our RNG on the GPU: each CUDA thread computes random numbers and all node computations are executed on the GPU. Moreover, we have minimized the

TABLE 1.3 – Pricing results: CPU vs GPU

Number of	CPU pricing		GPU pricing	
trajectories	Value	ϵ	Value	ϵ
2^{18}	5.8614	0.0258	5.8616	0.0257
2^{19}	5.8835	0.0183	5.8836	0.0183
2^{20}	5.8761	0.0129	5.8763	0.0129

accesses to the global memory of the GPU, each GPU thread uses mainly fast GPU registers. This strategy leads to a very efficient usage of the GPUs, and achieves a high speedup on GPU clusters compared to a multicore CPU cluster.

To develop the CPU cluster version we used `g++ 4.1.2` compiler and its native and included `OpenMP` library, and the `OpenMPI 1.2.4` library. To develop the GPU cluster version we used the `nvcc 1.1 CUDA` compiler and the `OpenMPI 1.2.4` library. All these development environments appeared compatible.

Our GPU version is composed of `.h` and `.cu` files, compiled with the following commands :

```
nvcc --host-compilation C++
     -O3 -I/opt/openmpi/include
     -DOMPI_SKIP_MPICXX -c X.cu
nvcc -O3 -L/opt/openmpi/lib
     -o pricer X.o Y.o .... -lmpi -lm
```

On our machines the OpenMPI library is installed in the `/opt/openmpi/` directory. The `-DOMPI_SKIP_MPICXX` flag allows us to avoid the exception mechanisms implemented in the OpenMPI library (according to the MPI 2 standard), which are not supported by the `nvcc` compiler. The `-host-compilation C++` flag helps `nvcc` to understand the C++ code of the non-kernel routines.

1.5 Cluster comparison for pricing European contracts

The following subsections introduce results of three benchmark programs, implementing Algorithm 1 and computing ~ 0.25, ~ 0.5 and ~ 1 million MC trajectories (corresponding to different pricing accuracies).

The accuracy of the result

Table 1.3 represents the results of the same simulation using an increasing number of trajectories. Each one of these simulations is either done on a CPU cluster or a GPU cluster. 'Value'

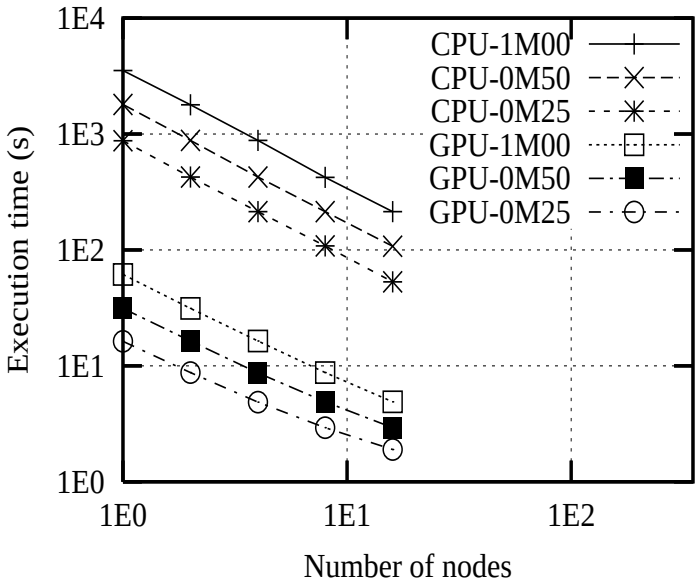

FIGURE 1.3 – The execution time of pricing European options

is the price of our Asian option and ϵ mesures the accuracy of the results using 95% confidence interval. ϵ is related to the standard deviation std of the simulation with the following relation : $\epsilon = 1.96 * std/\sqrt{Number\ of\ trajectories}$. We notice very slight differences between CPU and GPU simulations. The differences between GPU and CPU results are included in the 95% confidence interval. Although we repeated the experiments with different parameters we obtain the same similarity between GPU pricing and CPU pricing. This fact demonstrates that the single precision on GPUs does not affect the results of our simulations.

The parameters of the simulations are the following: Maturity $T = 1$, the time discretization $\delta t = 0.01$, $S_0^i = 100$, $r_i = r = 0.1$, $d_i = 0$, $\sigma_i = 0.2$ and the 40×40 correlating matrix $(\rho_{ik})_{1 \leq i,k \leq 40}$ is equal to the square root of a matrix (in the sense of Cholesky factorization) that is filled by 0.5 except on its diagonal which contains ones.

1.5.1 Computing efficiency

Effectiveness and speedup scaling: Figure 1.3 shows that the execution times of the three benchmarks on each testbed decrease very regularly: using 10 times more nodes divides the execution time by ~ 10. This result is due to the *embarrassingly parallel* feature of our algorithm, (communications are limited to input data broadcast and result reduction). So, Figure 1.3 shows our parallelization *scales* and efficiently uses the CPUs and GPUs of the cluster **M2**.

We process our largest benchmark (1 million MC trajectories) in $213.8s$ on 16 multi-core CPUs, while it requires $61.3s$ on one GPU and $4.9s$ on 16 GPUs. Figure 1.3 shows N GPUs

run about 45 times faster than N CPUs, so the speedup of our GPUs compared to our multi-core CPUs is close to 50. This speedup becomes close to 200 if we use only 1 CPU core, but using only 1 core of a CPU has no real sense. Also, according to Figure 1.3, when running the smallest benchmark of of 0.25 million trajectories on 16 GPUs the computation time is $1.9s$ where it takes $53.0s$ to run this simulation using 16 CPUs. The speedup on 16 GPUs of the 0.25 trajectories benchmark is small when compared to the 1 million trajectories benchmark, so the initialization time becomes significant and limits the global processing performance.

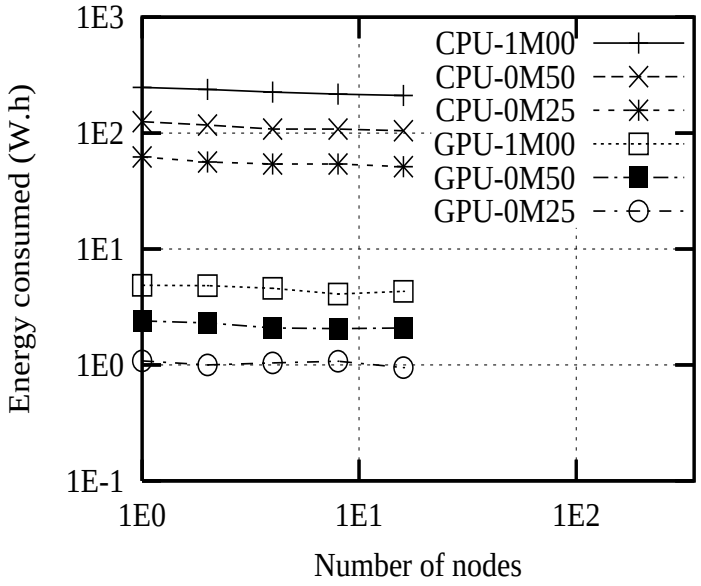

FIGURE 1.4 – Energetic consumption

1.5.2 Energy efficiency

We only consider the energy consumption of the nodes that are actually used during the computation, as it is easy to remotely switch off unused nodes of our GPU cluster. However it is not possible to switch off the GPU of one node when using only its CPU, and we have not yet tried to reduce the frequency and the energy consumption of the CPU when using mainly the GPU. Besides, we have not included the air conditioning energy consumption because the energy consumed depends on the type of air conditioning facility.

Effectiveness of computing energy: The cluster switch consumption of **M2** remains constant, independently of the number of nodes used. Figure 1.4 shows that the GPU computations of **M2** consume on average $0.0046\ kW.h$ to run our largest benchmark on 1 to 16 nodes, while the CPU computations consume on average $0.228\ kW.h$ to run the same option pricing on 1 to 16 nodes.

Complete balance sheet: Finally, using 16 GPU nodes we run our largest benchmark in $4.9s$ consuming $0.004\ kW.h$, in place of $213.8s$ and $0.211\ kW.h$ on 16 CPUs of the same **M2** cluster. It means we can perform our computation 43 times faster and consume 53 times less energy on our GPU cluster than on our CPU cluster. If we roughly consider the product of the speedup per the energy efficiency improvement, our GPU solution is globally $43 \times 53 \approx 2279$ times better than our CPU solution. As far as the smallest benchmark is concerned, we obtain a GPU solution which computes 27 times faster and consumes 53 times less energy.

1.6 Implementation of Longstaff-Schwartz algorithm on GPU

Although a lot of work has been done in variance reduction techniques, here we prefer the implementation of a basic LSR which will help a better understanding of the CPU/GPU comparison. In more standard applications, one can also implement the importance sampling method [46] or a European price as a control variable to accelerate the convergence. First of all, we detail the different steps of LSR in Algorithm 2. Afterwards, we are going to present the GPU version of Algorithm 2 in Algorithm 3. In Algorithm 2 and Algorithm 3, we use the parameter l as a path index and i as a dimension indix, we also denote n the number of simulated paths, m the total dimension and δt the time discretization. In addition, we use the set $\Gamma_l = \{e^{-r\delta t}\overline{C}(S_t^{(l)}) < \Phi(S_t^{(l)})\}$ that tests the continuation for each trajectory l using the indicator application 1.

1.6.1 Parallel Path Generation on GPU (PG)

This part of the algorithm depends on whether the random number generator can be parallelized or not. But, as presented in subsection 1.3.1, we use the CMRG that is parallellized by period splitting. Consequently, the PG phase is an embarrassingly parallel part of the simulation and we generate independently the random numbers for each path, then we use the Brownian bridge technique [25] to generate the Brownian motions and the asset prices at each time step according to (1.1).

1.6.2 The Regression Phase on GPU+CPU (REG)

As mentioned previously, the convergence study given in [26] shows that the number of trajectories needed to approximate the expectation (1.13) is more than exponentially proportional to the degree of regression. Thus, for the REG phase we use monomials of degrees less than or

Algorithm 2: LSR algorithm for an American put option

Input: Model parameters and CMRG initialization.
Output: $P_0(S_0)$
for $t \in \{T, \ldots, 2\delta t, \delta t\}$ **do**
  `/* Computations performed during the PG phase */`
  for $i \in \{1, \ldots m\}$ **do**
    for $l \in \{1, \ldots n\}$ **do**
      – Draw $W_t^{i,(l)}$ using CMRG and the Brownian bridge induction
      – Use (1.1) to update the asset price $S_t^{i,(l)}$
    end
  end
  if $(t < T)$ *and* $l \in \{\Phi(S_t^{(l)}) > 0\}$ **then**
    `/* Computations performed during the REG phase */`
    – Approach the expectations: (1.12) and (1.13)
    – $A = \Psi^{-1}\mathbb{E}\left(P_{t+\delta t}(S_{t+\delta t})\psi(S_t)\right)$
    `/* Computations performed during the PRC phase */`
    for $l \in \{1, \ldots n\}$ **do**
      – $\overline{C}(S_t^{(l)}) = A^t\psi(S_t^{(l)})$
      – Compute the payoff $\Phi(S_t^{(l)})$
      – $P_t(S_t^{(l)}) = 1_{\Gamma_l}\Phi(S_t^{(l)}) + 1_{\Gamma_l^c}e^{-r\delta t}P_{t+\delta t}(S_{t+\delta t}^{(l)})$
    end
    if $(t = \delta t)$ **then**
      `/* `$P_0(S_0)$` is the price of the option */`
      $P_0(S_0) = \max\left(\Phi(S_0), \frac{e^{-r\delta t}}{n}\sum_{l=1}^{l=n} P_{\delta t}(S_{\delta t}^{(l)})\right)$
    end
  end
  else
    `/* Computations performed during the PRC phase */`
    for $l \in \{1, \ldots n\}$ **do**
      $P_t(S_t^{(l)}) = 1_{t=T}\Phi(S_T^{(l)}) + 1_{\Phi(S_t^{(l)})\leq 0}e^{-r\delta t}P_{t+\delta t}(S_{t+\delta t}^{(l)})$
    end
    `/* We have, of course, `$\forall l \in \{1, \ldots n\}\ P_{T+\delta t}(S_{T+\delta t}^{(l)}) = 0$` */`
  end
end

equal to 2 for one dimension and we will use affine regression in the multidimensional simulation. We do not use monomials of degrees up to 2 in the multidimensional simulation because it does not improve significantly the numerical results either.

Besides, the inversion of matrices cannot be done in parallel. Subsequently we need to transfer all the following values for each path l, from GPU to CPU:

$$P_{t_{k+1}}(S^{(l)}_{t_{k+1}})\psi(S^{(l)}_{t_k}) \quad and \quad \psi(S^{(l)}_{t_k})\psi^t(S^{(l)}_{t_k}) \tag{1.11}$$

where l is the path index.

When the values of (1.11) are in the CPU memory, we approach expectations (1.12) and (1.13) with an arithmetic average then we perform the SVD method according to [53].

$$\mathbb{E}\left(P_{t_{k+1}}(S_{t_{k+1}})\psi(S_{t_k})\right) \approx \frac{1}{n}\sum_{l=1}^{n} P_{t_{k+1}}(S^{(l)}_{t_{k+1}})\psi(S^{(l)}_{t_k}) \tag{1.12}$$

$$\mathbb{E}\left(\psi(S_{t_k})\psi^t(S_{t_k})\right) \approx \frac{1}{n}\sum_{l=1}^{n} \psi(S^{(l)}_{t_k})\psi^t(S^{(l)}_{t_k}) \tag{1.13}$$

with n representing the number of paths.

In this simulation phase, the GPU plays a role in the computation of the different products. For example in the case of three assets $\psi(S^{(l)}_{t_k}) = (1, S^{1,(l)}_{t_k}, S^{2,(l)}_{t_k}, S^{3,(l)}_{t_k})$, one has to compute on GPU the following products associated to each path l:

$$\begin{gathered}\left((S^{1,(l)}_{t_k})^2, (S^{2,(l)}_{t_k})^2, (S^{3,(l)}_{t_k})^2, S^{1,(l)}_{t_k}S^{2,(l)}_{t_k}, S^{1,(l)}_{t_k}S^{3,(l)}_{t_k}, S^{2,(l)}_{t_k}S^{3,(l)}_{t_k}\right) \\ \left(P_{t_{k+1}}(S^{(l)}_{t_{k+1}})S^{1,(l)}_{t_k}, P_{t_{k+1}}(S^{(l)}_{t_{k+1}})S^{2,(l)}_{t_k}, P_{t_{k+1}}(S^{(l)}_{t_{k+1}})S^{3,(l)}_{t_k}, \right)\end{gathered} \tag{1.14}$$

As will be demonstrated in Section 1.7.1, performing the above computations on the GPU compensates for the loss due to the data transfer between GPU and CPU.

1.6.3 The Parallel Pricing on GPU (PRC)

Once we compute the regression vector A, the backward induction (1.4) can be done independently for each path of the simulation. At the final step time, we transfer the different price values from GPU to CPU and estimate the expectation using the arithmetic average of all prices.

Algorithm 3: GPU version of LSR algorithm for an American put option

Input: The same as in Algorithm 2
Output: $P_0(S_0)$
GPU initialization.
for $t \in \{T, \ldots, 2\delta t, \delta t\}$ **do**
  `/* Computations performed during the PG phase */`
  Distribute the n trajectories on stream processors + Perform the same operations as Algorithm 2.
  if $(t < T)$ *and* $l \in \{\Phi(S_t^{(l)}) > 0\}$ **then**
    `/* Computations performed during the REG phase */`
    – Perform the products (1.14) on GPU.
    – Transfer (1.11) from GPU to CPU.
    – Same operations as Algorithm 2.
    `/* Computations performed during the PRC phase */`
    Distribute + Perform the same operations as Algorithm 2.
    if $(t = \delta t)$ **then**
      – Transfer from GPU to CPU: $(P_{\delta t}(S_{\delta t}^{(l)}))_l$
      – Compute the price of the option $P_0(S_0)$ as in Algorithm 2.
    end
  end
  else
    `/* Computations performed during the PRC phase */`
    Distribute + Perform the same operations as Algorithm 2.
  end
end

TABLE 1.4 – Running time (seconds): CPU vs GPU

Simulation	50 time steps		100 time steps		300 time steps	
phases	CPU	GPU	CPU	GPU	CPU	GPU
PG	0.671	0.047	1.278	0.079	4.386	0.162
PRC	0.484	0.064	1.315	0.116	8.864	0.359
REG	0.266	0.222	0.557	0.447	1.919	1.324

1.7 Pricing American contracts using GPUs

In this work we were not able to directly compare our results to those presented in [15] as the authors do not give precise enough information on the digital procedure. Note that we also study here the running time of a multidimensional case which is more representative of real American option challenges. The results presented in this section are evaluated by computing an average value of the different simulation times

We divide this section in three parts. The first part includes the comparison between our GPU implementation on **M1** using the NVIDIA Cg Toolkit and the QuantLib open-source library [31] implementation on the same machine as the Longstaff and Schwartz algorithm. The second part studies the dependance of the running time on **M1** of a multidimensional American option according to the number of paths simulated and the dimension of the contract. Using the results of the previous subsections, in the last subsection, we discuss a possible parallelization of pricing ACs on a cluster and which introduces one prospective work related to this article.

1.7.1 The running time comparison between GPU and CPU

The QuantLib open-source library is a highly object-oriented library. In order to make a fair comparison between the GPU and the CPU, we need to avoid overheads which are unrelated to our algorithm. Thus, we only concentrate on the execution time of the main three phases of the simulation. Moreover, we only consider the original one-core implementation of QuantLib implementation and we do not parallelize the simulation on the two cores of **M1**. In Table 1.4, we compare the execution time between our GPU implementation and the QuantLib one-core CPU implementation of ACs for an increasing number of time steps. We perform the simulation of one-dimensional American put on $2^{14} = 16384$ trajectories. According to Table 1.4, the REG phase is faster on the GPU than it is on the CPU. The two other phases are significantly improved when using the GPU which reduces the total time of the simulation. It is also noticeable that when we increase the number of time steps, we make the simulation more complex and this provides a higher speedup.

TABLE 1.5 – Increasing dimensions and trajectories on GPU (time in seconds)

Simulation	1 asset (2^{14})	4 assets (2^{14})	4 assets (2^{16})	4 assets (2^{18})
Total	1.092	1.591	2.605	7,360
PG	0.047	0.146	0.159	0.171
PRC	0.064	0.114	0.303	1.090
REG	0.222	0.588	1.400	5.387

1.7.2 Multidimensional American option

In this part, we compare the running times of one-dimensional American put ($2^{14} = 16384$ trajectories and 50 time steps) with the running times of four assets American put (using 50 time steps) that has the following payoff $\Phi(S_T)$:

$$\Phi(S_T) = \left(K - \prod_{i=1}^{4} (S_T^i)^{1/4} \right)_+ \tag{1.15}$$

We study this multidimensional payoff, because it is easier to check the prices coherence. Indeed, the American put on a geometric average of stocks can be approximated very well when using the one-dimensional equivalence and a tree method. Besides, unlike [2], to reduce the complexity of the REG phase, we restrict ourselves to the constant and linear monomials regression for the multidimensional benchmark.

In Table 1.5, the first line provides the total running time that includes initialization and CPU/GPU data transfer [3]. The three columns on the right show the running times of the four assets American put associated to an increasing number of trajectories: 2^{14}, 2^{16}, 2^{18}.

According to Table 1.5, the running time of the PG phase increases linearly with the number of assets and is slightly modified when we increase the number of trajectories. This fact can be explained by the lightness of the operations performed associated to the chosen model. Conversely, the PRC phase is rather sensitive to the number of trajectories. Like the PRC phase, the running time of REG is approximately linear with the number of trajectories, and this is also the case when we increase the dimensionality of the problem [4]. Finally, even if pricing multidimensional ECs on GPUs allows better overall speedup, we obtain very short running times for a multi-asset American option pricing using a large number of trajectories.

When comparing the phases in Table 1.5, we see that the total running time on GPU is $\sim 70\%$ dominated by the running time of the REG phase. We will see, in the next subsection, a method that can reduce the execution time of the REG phase using a cluster of machines.

3. The initialization and the data transfer takes at most 0.8 seconds

4. Because in the one-dimensional benchmark we use $(1, S_1, S_1^2)$ as a regression basis and we use $(1, S_1, S_2, S_3, S_4)$ for the four-dimensional benchmark

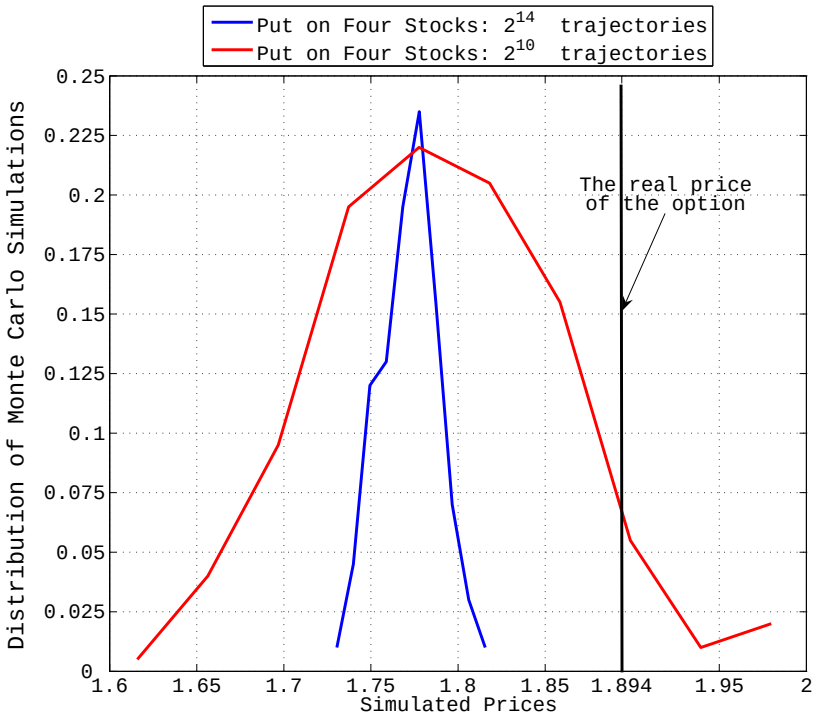

FIGURE 1.5 – The histogram of simulated prices according to the number of trajectories.

1.7.3 About pricing American contracts on GPU cluster

Pricing multidimensional ACs remains one of the most challenging problems in financial applications. The popularity of methods based on MC that use regression is due to the fact that they provide, in a sufficiently short time, relatively good solutions to dimensions included between one and three or one and five (It depends on the variance and the regression basis). Knowing the strengths and the weaknesses of these methods, we only tried to take advantage of the parallel architecture of the GPU to reduce the execution time of the Longstaff and Schwartz algorithm. As a result of the previous subsections, we show that we can efficiently execute on GPUs the phases PRC and PG. In this subsection we present how to reduce the execution time of the REG phase using a cluster of machines.

In Figure 1.5 we sketch the histogram of 200 simulated prices of the four dimensional American put whose payoff is given by (1.15). The parameters of the simulations are the following : Maturity $T = 1$, time discretization $\delta t = 0.02$, $S_0^i = 100$, $r_i = r = 0.0953$, $d_i = 0$, $\sigma_i = 0.2$ and the 4×4 correlating matrix $(\rho_{ik})_{1 \leq i,k \leq 4}$ is equal to the identity matrix. As said above, we choose this multidimensional benchmark because we can have a good approximation of the price of the option using the one-dimensional equivalence and a tree method. In Figure 1.5, we

give the real value [5] of this option and the prices resulting from the MC simulation of 2^{10} and 2^{14} trajectories.

According to Figure 1.5, the two histograms are centered approximately around the same value which is different from the real value of the option. This difference is due to regression errors and even if we use more trajectories (2^{14} instead of 2^{10}) the average of simulated values remains relatively unchanged. Nevertheless, when we increase the number of the simulated trajectories we shrink the distribution of simulated prices and thus we reduce the difference between the real value and the simulated value. We refer the reader to [16] for a CLT result of ACs.

Consequently, we can parallelize the AC pricing on a cluster of 16 machines using 2^{10} trajectories for each machine then averaging instead of simulating 2^{14} trajectories on only one machine. The former solution will improve the running time of the PG and the PRC phases. Also, according to the results of subsection 1.7.2, the decrease of the number of trajectories simulated per machine reduces almost linearly the execution time of the REG phase. The overall solution obtained would be more effective on a cluster of machines than it is on only one machine.

In the previous analysis, in order to parallelize our implementation on a cluster of machines, we use the fact that the reduction of the number of simulated trajectories does not affect the errors implied by the regression phase. However, there are limits to this result, indeed [26] recommends to have a number of polynomials $K = K_N \sim O(\sqrt{\log(N)})$ where N is the number of paths. Thus, for a fixed number of polynomials this determines approximately the minimum number of the simulated paths needed for a good regression.

1.8 Conclusion and future work

The main results of this research work are the following:

– We have analyzed two different methods of parallelizing RNGs for parallel and distributed architectures. The results of this study are two examples of RNGs which are most suited to the GPU architecture.
– When running MC simulations, the accuracy of the results obtained with a cluster of GPUs using single precision is similar to the one obtained with a cluster of CPUs using double precision.
– Mixed coarse and fine grain parallelization of MC simulations for pricing ECs, using

5. It is the value approximated using the tree method.

MPI and OpenMP on multi-core CPU cluster, or MPI and CUDA on GPU cluster, is an efficient strategy and scales.

– Execution time and energy consumption of MC simulations can be both efficiently reduced when using GPU clusters in place of pure CPU clusters.
– In the case of American options that depend on one asset, we compare our GPU implementation with the one given in QuantLib library. Even if the speedup is small compared to pricing ECs, we observe a 2 to 10 improvement of the execution time and the speedup increases with the complexity of the problem.
– We look into the multi-asset American option and how the execution time can change with the dimension and the number of trajectories. As a result, when using GPUs, the execution time is almost only dominated by the REG phase because it is the only phase that cannot be parallelized on the GPU. Consequently, we give a method that aims at reducing the running time of the REG phase and which is based on a cluster implementation.

Algorithms introduced in this paper remain adapted to the new multi-core CPUs and the new generation of graphic cards which computes in double precision.

Like for the European contracts, we are going to extend the ACs pricing on a CPU/GPU cluster using the method presented in the subsection 1.7.3. Subsequently, we will compare the speedup and the energy efficiency of the parallelization on GPUs and CPUs using the coarse grained and fine grained paradigms.

Besides, in order to improve the parallelization of the American options pricing, we are studying now the Malliavin Calculus-based algorithms [7] which completly avoid matrix regressions and allow an efficient computation of the hedge.

Acknowledgment: This research is part of the ANR-CIGC GCPMF project, and is supported both by ANR (French National Research Agency) and by Region Lorraine. The authors want to thank Professor Pierre L'Ecuyer for his valuable advices on the choice of random number generators.

Chapitre 2

American Options Based on Malliavin Calculus [1]

Abstract

This paper is devoted to pricing American options using Monte Carlo and Malliavin calculus. We develop this method on two types of models, the multidimensional exponential model with deterministic (non-constant) volatility and the multidimensional Heston model. To obtain good numerical results, we introduce a variance reduction technique based on conditioning and a bias reduction method that relies on an appropriate choice of the number of simulated paths in the computation of the quotient of two expectations. Since our techniques are well-suited to parallel implementation, our numerical experiments are performed using multi-core CPU and many-core GPU environments.

2.1 Introduction and objectives

In this paper, we explore a **M**onte **C**arlo (**MC**) method based on **M**alliavin calculus (**MCM**) for pricing **A**merican **O**ptions (**AO**). Unlike usual American option algorithms as **L**ongstaff-**S**chwartz (**LS**) [42] or Malliavin calculus techniques based on localization, the method presented here does not need any parametric regression and higher dimensional problems can be dealt with more easily as the accuracy of results depends only on the number of simulated trajectories.

Assuming that the asset S follows a Markovian model, American contracts can be exercised at any trading date until maturity and their prices are given, at each time t, by (see [25]) $P_t(S_t)$

with

$$P_t(x) = \sup_{\theta \in \mathcal{T}_{t,T}} E_{t,x}\left(e^{-r(\theta-t)}\Phi(S_\theta)\right), \tag{2.1}$$

where $\mathcal{T}_{t,T}$ is the set of stopping times in the time interval $[t,T]$, $E_{t,x}$ is the expectation associated to the risk neutral probability given that $S_t = x$ and r and $\Phi(S_t)$ are respectively the instantaneous interest rate and the payoff of the contract.

In order to evaluate numerically the price (2.1), we first need to approach continuous stopping times in $\mathcal{T}_{t,T}$ with discrete stopping times taking values in the finite set $t = t_0 < t_1 < ... < t_n = T$ (Bermudan approximation). When we do this approximation, pricing American options can be reduced to the implementation of a discrete time dynamic programming algorithm (see [25]). Like the LS algorithm [42], we implement the dynamic programming principle in terms of the optimal stopping times τ_k, for each path, as follows

$$\begin{aligned} \tau_n &= T, \\ \forall k \in \{n-1, ..., 0\}, \quad \tau_k &= t_k 1_{A_k} + \tau_{k+1} 1_{A_k^c}, \end{aligned} \tag{2.2}$$

where the set $A_k = \{\Phi(S_{t_k}) > C(S_{t_k})\}$ and $C(S_{t_k})$ is the continuation value, given by

$$C(S_{t_k}) = E\left(e^{-r\Delta t} P_{t_{k+1}}(S_{t_{k+1}}) \middle| S_{t_k}\right), \tag{2.3}$$

where $\Delta t = t_{k+1} - t_k$. Thus, to evaluate the price (2.1), we need to estimate $C(S_{t_k})$.

Algorithms devoted to American pricing and based on Monte Carlo, differ essentially in the way they estimate and use the conditional expectation (2.3). For example the authors of [57] perform a regression to estimate the continuation value, but unlike [42], they use $C(S_{t_k})$ instead of the actual realized cash flow $P_{t_{k+1}}(S_{t_{k+1}})$ to update the price in (2.2). We refer the reader to [16] for a presentation of the way this estimation is done for the LS algorithm and details on the convergence. Other methods use the Malliavin calculus with localization [7] or the quantization method [8] for $C(S_{t_k})$ computation.

In this work, we rewrite (2.3) using Malliavin calculus but unlike [7] we use the induction (2.2) for the implementation and we propose a nonparametric variance and bias reduction methods, without using localization. Formally speaking, for a constant r, we can rewrite the conditional expectation using the Dirac distribution $\varepsilon_x(\cdot)$ at point x then using the Malliavin calculus for a large class of diffusion models, we get

$$C(x) = \frac{E\left(e^{-r\Delta t} P_{t_{k+1}}(S_{t_{k+1}})\varepsilon_x(S_{t_k})\right)}{E\left(\varepsilon_x(S_{t_k})\right)} = \frac{E\left(e^{-r\Delta t} P_{t_{k+1}}(S_{t_{k+1}}) 1_{S_{t_k} \geq x} \Theta_{t_k, t_{k+1}}\right)}{E\left(1_{S_{t_k} \geq x} \Theta_{t_k, t_{k+1}}\right)}. \tag{2.4}$$

where $1_{S_{t_k} \geq x}$ is equal to the tensorial product $\prod_{i=1}^{d} 1_{S^i_{t_k} \geq x_i}$, when $d \geq 1$.

In this paper, we provide the value of $\Theta_{t_k,t_{k+1}}$ for two classes of models: **M**ultidimens-ional **E**xponential **D**iffusions with deterministic **C**oefficients (**MEDC**) and the **M**ulti-dimensional **H**eston (**MH**) model. In the MEDC models, the dynamics of the assets $\{S^i_t\}_{1\leq i\leq d}$ is given by

$$\frac{dS^i_t}{S^i_t} = r_i dt + \sum_{j=1}^{i} \sigma_{ij}(t) dW^j_t, \quad S^i_0 = z_i, \quad i = 1, .., d.$$

In the MH models, the dynamics of the assets $\{S^i_t\}_{1\leq i\leq d}$ is given by

$$\text{for } 1 \leq i \leq d \quad \begin{array}{c} d\nu^i_t = \kappa_i(\theta_i - \nu^i_t)dt + \eta_i\sqrt{\nu^i_t}d\widetilde{Z}^i_t, \quad \nu^i_0 = y^i, \\ dS^i_t = S^i_t\left(r_i dt + \sqrt{\nu^i_t}dZ^i_t\right), \quad S^i_0 = z^i. \end{array}$$

For the conditional expectation (2.4) simulation, a variance reduction method and a bias reduction method can be applied for MEDC and MH models as well as for other models. Without loss of generality, for the MEDC model, instead of simulating directly the last term in (2.4), we project $1_{S_{t_k}\geq x}(S_{t_k})\Theta_{t_k,t_{k+1}}$ using a conditioning as follows

$$C(x) = \frac{E\left(e^{-r\Delta t}P_{t_{k+1}}(S_{t_{k+1}})E\left[1_{S_{t_k}\geq x}\Theta_{t_k,t_{k+1}}\big|\{\int_0^{t_{k+1}}\sigma_{ij}(u)dW^j_u\}_{1\leq j\leq i\leq d}\right]\right)}{E\left(E\left[1_{S_{t_k}\geq x}\Theta_{t_k,t_{k+1}}\big|\{\int_0^{t_{k+1}}\sigma_{ij}(u)dW^j_u\}_{1\leq j\leq i\leq d}\right]\right)}. \tag{2.5}$$

Then, the bias reduction method is applied by setting the continuation value to the approximation below

$$C(x) \approx \frac{\frac{1}{N'}\sum_{l=1}^{N'} e^{-r\Delta t}P^l_{t_{k+1}}(S_{t_{k+1}})h(x, \{\int_0^{t_{k+1}}\sigma_{ij}(u)dW^j_u\}^l_{1\leq j\leq i\leq d})}{\frac{1}{N}\sum_{l=1}^{N} h(x, \{\int_0^{t_{k+1}}\sigma_{ij}(u)dW^j_u\}^l_{1\leq j\leq i\leq d})}, \tag{2.6}$$

with $h(x, \{w_{ij}\}_{j\leq i}) = E(1_{S_{t_k}\geq x}\Theta_{t_k,t_{k+1}}\big|\{\int_0^{t_{k+1}}\sigma_{ij}(u)dW^j_u\}_{1\leq j\leq i\leq d} = \{w_{ij}\}_{1\leq j\leq i\leq d})$ and $N \neq N'$. Thus, we improve the convergence to the real price of an American option by empirically using an appropriate relation between N and N' that does not increase much the variance of the estimator (2.6) (when compared to the case $N = N'$). Note that, even if one can also reduce the variance by an "appropriate" control variate, here we choose not to implement this kind of method because it is not standard for American options.

Regarding the numerical simulation, we test MCM on a multi-core CPU (Central Processing Unit) as well as a many-core GPU (Graphic Processing Unit). We will discuss the advantages

of the parallel implementation of MCM on a desktop computer that has the following specifications: Intel Core i7 Processor 920 with 9GB of tri-channel memory at frequency 1333MHz. It also contains one NVIDIA GeForce GTX 480.

The outline of this paper is as follows. In section 2.2 we establish the notations and the Malliavin calculus tools. We give in section 2.3 (see (2.14)) the value of $\Theta_{t_k,t_{k+1}}$ for the MEDC model and we extend it to the MH model in section 2.4. Section 2.5 is devoted to a variance reduction method based on conditioning and section 2.6 to the bias reduction method based on the appropriate relation between N and N' (2.6). In the last section, we show that the multi-dimensional MCM implementation on a many-core GPU is more than 60 times faster than its implementation on a multi-core CPU. We also provide the numerical comparison between LS and MCM. Finally, we study the results of using the two variance reduction methods (2.5) and (2.6) which allow to obtain accurate prices even when simulating only 2^{10} trajectories.

2.2 Notations, hypothesis and key tools

Let T be the maturity of the American contract, $(\Omega, \mathcal{F}, P)$ a probability space on which we define a d-dimensional standard Brownian motion $W = (W^1, ..., W^d)$ and $\mathbb{F} = \{\mathcal{F}_s\}_{s\leq T}$ the P-completion of the filtration generated by W until maturity. Moreover, we denote by $\{\mathcal{F}_s^{i,...,d}\}_{s\leq t}$ the P-completion of the filtration generated by $(W^i, ..., W^d)$ until the fixed time $t \in [0, T]$.

Throughout sections 2.3 and 2.4, we will use two operators: The Malliavin derivative D and the Skorohod integral δ and we define them in the same way as in [55]. For a fixed $m \in \mathbb{N}$, we define the subdivision $\{t_m^k\}_{k\leq 2^m}$ of the finite interval $[0, T]$ by: $t_m^k = kT/2^m$. Then we introduce $\mathcal{S}(\mathbb{R}^{d\times 2^m})$ the Schwartz space of infinitely differentiable and rapidly decreasing functions on $\mathbb{R}^{d\times 2^m}$. Let $f \in \mathcal{S}(\mathbb{R}^{d\times 2^m})$, we define the set $\mathfrak{S}^m$ of simple functionals by

$$F \in \mathfrak{S}^m \Leftrightarrow F = f\left(W_{t_m^1} - W_{t_m^0}, W_{t_m^2} - W_{t_m^1}, ..., W_{t_m^{2^m}} - W_{t_m^{2^m-1}}\right).$$

One can prove that $\mathfrak{S} = \bigcup_{m\in\mathbb{N}} \mathfrak{S}^m$ is a linear and dense subspace in $L^2(\Omega)$ and that the Malliavin derivatives $D^i F$ of $F \in \mathfrak{S}$ defined by

$$D_t^i F = \sum_{k=0}^{2^m-1} \frac{\partial f}{\partial x^{i,k}}\left(W_{t_m^1} - W_{t_m^0}, ..., W_{t_m^{2^m}} - W_{t_m^{2^m-1}}\right) \mathbf{1}_{[t_m^k, t_m^{k+1}[}(t)$$

is a process in $L^2(\Omega \times [0,T])$. We associate to $\mathfrak{S}$ the norm $||\cdot||_{1,2}$ defined by

$$||F||_{1,2}^2 = E|F|^2 + \sum_{i=1}^{d} E\int_0^T (D_t^i F)^2 dt.$$

Finally, the space $\mathbb{D}^{1,2}$ is the closure of $\mathfrak{S}$ with respect to this norm and we say that $F \in \mathbb{D}^{1,2}$ if there exists a sequence $F_m \in \mathfrak{S}$ that converges to F in $L^2(\Omega)$ and that $D_u F_m$ is a Cauchy sequence in $L^2(\Omega \times [0,T])$.

Now we use the duality property between δ and D to define the Skorohod integral δ. We say that the process $U \in Dom(\delta)$ if $\forall F \in \mathbb{D}^{1,2}$

$$\left| E\left(\int_0^T U_t \cdot D_t F dt \right) \right| \leq C(U)||F||_{1,2},$$

where $C(U)$ is a positive constant that depends on the process U. If $U \in Dom(\delta)$, we define the Skorohod integral $\delta(U) = \int U_t \delta W_t$ by

$$\forall F \in \mathbb{D}^{1,2}, \quad E\left(F\int_0^T U_t \cdot \delta W_t \right) = E\left(F\delta(U) \right) = E\left(\int_0^T U_t \cdot D_t F dt \right), \tag{2.7}$$

$(\cdot)$ is the inner scalar product on $\mathbb{R}^d$.

Below, we give some standard properties of the operators D and δ:

1. If the process U_t is adapted, $\delta(U) = \int U_t \delta W_t$ coincides with the Itô integral $\int U_t dW_t$.
2. *The Chain Rule:* Let $F = (F_1, F_2, ..., F_k) \in (\mathbb{D}^{1,2})^k$ and $\phi : \mathbb{R}^k \to \mathbb{R}$ a continuously differentiable function with bounded partial derivatives.
   Then $\phi(F_1, F_2, ..., F_k) \in \mathbb{D}^{1,2}$ and:

$$D_t\phi(F_1, F_2, ..., F_k) = \sum_{i=1}^{k} \frac{\partial \phi}{\partial x^i}(F_1, F_2, ..., F_k) D_t F_i.$$

3. *The Integration by Parts:* The IP formula will be extensively used in the next section on the time intervals $I = (0,s)$ and $I = (s,t)$ with $s < t \in]0,T]$: Assume $F \in \mathbb{D}^{1,2}$, U is an adapted process in $Dom(\delta)$ and $FU \in Dom(\delta)$. For each $1 \leq i \leq d$ we have the following equality

$$E\left(\int_I F U_u \delta^i W_u \right) = E\left(F\int_I U_u dW_u^i \right) - E\left(\int_I U_u D_u^i F du \right). \tag{2.8}$$

To simplify the notations, we denote $H_i(S_s^i) = H(S_s^i - x_i)$ for the Heaviside function of the difference between the i^{th} stock and the i^{th} coordinate of the positive vector x.

Throughout this article, we will suppose that $g \in \mathcal{E}_b(\mathbb{R}^d)$ is a measurable function with polynomial growth

$$\mathcal{E}_b(\mathbb{R}^d) = \left\{ f \in \mathcal{M}(\mathbb{R}^d) \ : \ \exists C > 0 \ \textit{and} \ m \in \mathbb{N}; \ |f(y)| \leq C(1 + |y|_d)^m) \right\}, \tag{2.9}$$

where $\mathcal{M}(\mathbb{R}^d)$ is the set of measurable functions on $\mathbb{R}^d$ and $|\cdot|_d$ is the euclidean norm. The elements of the set $\mathcal{E}_b(\mathbb{R}^d)$ satisfy the finiteness of the expectations computed in this article.

2.3 The continuation for a deterministic diffusion matrix

The process S_t models the price of a vector of assets $S_t^1, ..., S_t^d$ which constitute the solution of the following stochastic differential equation

$$\frac{dS_t^i}{S_t^i} = r_i dt + \sum_{j=1}^{i} \sigma_{ij}(t) dW_t^j, \quad S_0^i = z_i, \quad i = 1, .., d, \tag{2.10}$$

where r_i are constants and $\sigma(t) = \{\sigma_{ij}(t)\}_{1 \leq i,j \leq d}$ is a deterministic triangular matrix, thus $\{\sigma_{ij}(t)\}_{i<j} = 0$. We suppose that the matrix $\sigma(t)$ is invertible, bounded and uniformly elliptic which ensures the existence of the inverse matrix $\rho(t) = \sigma^{-1}(t)$ and its boundedness. Dynamics (2.10) is widely used for equity models, HJM interest rate models and variance swap models. Moreover, one should note that in the case where the dynamics of S is given by local volatility model, we can use a discretization scheme to reduce it to an SDE of type (2.10) on subintervals.

The first theorem of this section provides the expression of the continuation value (2.3) when using Malliavin calculus for MEDC models. This theorem can be considered as an extension of the results on the continuation value for the multidimensional model with constant parameters detailed in [7]. In Theorem 2.2, we provide a closed-form expression for $\Gamma_{s,t}^k$, introduced in Theorem 2.1. Corollary 2.1 treats the special case $\sigma_{ij}(t) = \sigma_i \delta(i - j)$ (σ_{ij} is a constant) that will be used, with other models, to test numerically our nonparametric variance reduction and bias reduction methods detailed in section 2.5 and section 2.6.

Theorem 2.1 *For any $s \in]0, t[$, $g \in \mathcal{E}_b(\mathbb{R}^d)$ and $x = (x_1, ..., x_d)$ with $x_i > 0$,*

$$E\left(g(S_t) \Big| S_s = x\right) = \frac{T_{s,t}[g](x)}{T_{s,t}[1](x)}, \tag{2.11}$$

where $T_{s,t}[f](x)$ is defined for every function[1] $f \in \mathcal{E}_b(\mathbb{R}^d)$ *by*

$$T_{s,t}[f](x) = E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^{d}\frac{H_k(S_s^k)}{S_s^k}\right), \tag{2.12}$$

$\Gamma_{s,t} = \Gamma_{s,t}^1$ *and* $\Gamma_{s,t}^1$ *can be computed by the following induction scheme*

$$\Gamma_{s,t}^d = \pi_{s,t}^{d,d},\ for\ k \leq d-1:\ \Gamma_{s,t}^k = \Gamma_{s,t}^{k+1}\pi_{s,t}^{k,d} - \sum_{j=k+1}^{d}\int_0^t D_u^j\Gamma_{s,t}^{k+1}D_u^j\pi_{s,t}^{k,d}du, \tag{2.13}$$

with

$$\pi_{s,t}^{k,d} = 1 + \sum_{j=k}^{d}\int_0^t \varphi_{jk}(u)dW_u^j, \quad \varphi_{jk}(u) = \frac{1}{s}\rho_{jk}(u)1_{u\in]0,s[} - \frac{1}{t-s}\rho_{jk}(u)1_{u\in]s,t[},$$

where ρ is the inverse matrix $\rho(u) = \sigma^{-1}(u)$.

$H_k(S_s^k)$ is the Heaviside function of the difference between the k^{th} stock and the k^{th} coordinate of the positive vector x, $\mathcal{E}_b(\mathbb{R}^d)$ is defined in (2.9).

From this theorem we obtain

$$E\left(g(S_t)\Big|S_s = x\right) = \frac{E\left(e^{-r\Delta t}P_{t_{k+1}}(S_{t_{k+1}})1_{S_{t_k}\geq x}\Theta_{t_k,t_{k+1}}\right)}{E\left(1_{S_{t_k}\geq x}\Theta_{t_k,t_{k+1}}\right)},\ \Theta_{t_k,t_{k+1}} = \frac{\Gamma_{t_k,t_{k+1}}}{\prod_{i=1}^d S_{t_k}^i}. \tag{2.14}$$

with $1_{S_{t_k}\geq x}$ equal to the tensorial product $\prod_{i=1}^{d}1_{S_{t_k}^i\geq x_i}$.

To prove Theorem 2.1, we need the following two lemmas which are proved in the appendix. It follows from Lemma 2.1 that the sum $\sum_{i=k}^d \rho_{ik}(u)D_u^i g(S_t)$ does not depend on u.

Lemma 2.1 *For any $u \in]0,t[$, $f \in \mathcal{C}^1(\mathbb{R}^d)$ and S given by the SDE (2.10), we have*

$$\sum_{i=k}^{d}\rho_{ik}(u)D_u^i f(S_t) = S_t^k\partial_{x_k}f(S_t). \tag{2.15}$$

The second lemma is based on the duality property of the Malliavin calculus.

1. In our case $f = g$ or $f = 1$

Lemma 2.2 *For any $I \subset]0,t[$, $h \in \mathcal{C}_b^\infty(\mathbb{R})$, $x \in \mathbb{R}_+^d$, $F \in \mathbb{D}^{1,2}$ and S given by the SDE (2.10), we have*

$$\begin{aligned} E\left(\int_I \frac{FD_u^k h(S_s^k)}{\sigma_{kk}(u)} du\right) &= E\left(h(S_s^k) F \sum_{i=k}^d \int_I \rho_{ik}(u) dW_u^i\right) \\ &- E\left(h(S_s^k) \sum_{i=k}^d \int_I \rho_{ik}(u) D_u^i F du\right). \end{aligned} \tag{2.16}$$

Proof of Theorem 2.1:

We will prove that for $h_i \in \mathcal{C}_b^\infty(\mathbb{R})$, $0 \le k \le d$ and $f \in \mathcal{E}_b(\mathbb{R}^d)$

$$E\left(f(S_t)\prod_{i=1}^d h_i'(S_s^i)\right) = E\left(f(S_t)\Gamma_{s,t}^{k+1}\prod_{i=1}^k h_i'(S_s^i)\prod_{i=k+1}^d \frac{h_i(S_s^i)}{S_s^i}\right) \tag{2.17}$$

and that Theorem 2.1 is obtained directly from (2.17) by setting $k = 0$.

Step 1: ((2.17) with $k = 0$) $\Rightarrow$ (2.11).

Heuristically $E\left(g(S_t)\middle|S_s = x\right)$ can be viewed as $E\left(g(S_t)\varepsilon_x(S_s)\right)/E\left(\varepsilon_x(S_s)\right)$ where ε_x is the Dirac distribution at x and we know that $\varepsilon_{x_i} = H_i'$. In order to make this reasoning rigorous we use $p_{s,t}(u,v)$, the distribution of the vector $(S_s^1, ..., S_s^d, S_t^1, ..., S_t^d)$. Indeed, according to our assumption, the distribution of this vector admits a log-normal joint density with respect to the Lebesgue measure on $\mathbb{R}_+^d \times \mathbb{R}_+^d$. Then

$$E\left(g(S_t)\middle|S_s = x\right) = \frac{\int_{\mathbb{R}}^d g(v) p_{s,t}(x,v) dv}{\int_{\mathbb{R}}^d p_{s,t}(x,v) dv}.$$

Let $\phi \in \mathcal{C}_c^\infty(\mathbb{R})$ be a mollifier function with support equal to $[-1,1]$ and such that $\int_{\mathbb{R}} \phi(u) du = 1$, then for any $u \in \mathbb{R}$ we define

$$h_{mk}(u) = (H_k * \phi_m)(u) \in \mathcal{C}_b^\infty(\mathbb{R}), \quad \phi_m(u) = m\phi(mu).$$

If the equality (2.17) is correct for any k, then it is correct for $k = 0$ which means

$$E\left(f(S_t)\prod_{k=1}^d h_{mk}'(S_s^k)\right) = E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^d \frac{h_{mk}(S_s^k)}{S_s^k}\right). \tag{2.18}$$

On the one hand, $h_{mk}(u)$ converges to $H_k(u)$ except at $u = x_k$ and the absolute continuity of the law of S_s^k ensures that $h_{mk}(S_s^k)$ converges almost surely to $H_k(S_s^k)$. Using the dominated

convergence theorem, we prove the convergence of $h_{mk}(S_s^k)$ to $H_k(S_s^k)$ in $L^p(\Omega)$ for $p \geq 1$. By Cauchy-Schwarz inequality, we prove the convergence

$$E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^{d}\frac{h_{mk}(S_s^k)}{S_s^k}\right) \longrightarrow E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^{d}\frac{H_k(S_s^k)}{S_s^k}\right).$$

On the other hand, $h'_{mk}(u_k) = \int_{\mathbb{R}} H_k(v_k)\phi'_m(u_k - v_k)dv_k = \phi_m(u_k - x_k)$. Moreover, using the log-normal joint density $p_{s,t}(u,v)$ of the vector $(S_s^1, ..., S_s^d, S_t^1, ..., S_t^d)$ with $u = (u_1, ..., u_d)$ and $v = (v_1, ..., v_d)$, we get

$$E\left(f(S_t)\prod_{k=1}^{d}h'_{mk}(S_s^k)\right) = \int_{\mathbb{R}^d} f(v)\left(\int_{\mathbb{R}^d}\prod_{k=1}^{d}\phi_m(u_k - x_k)p_{s,t}(u,v)du_1...du_d\right)dv_1...dv_d$$

Because $\int_{\mathbb{R}^d}\prod_{k=1}^{d}\phi_m(u_k - x_k)p_{s,t}(u,v)du_1...du_d$ converges to $p_{s,t}(x,v)$ and due to the regularity properties of the density and the growth condition on f, we easily have

$$E\left(f(S_t)\prod_{k=1}^{d}h'_{mk}(S_s^k)\right) \longrightarrow \int_{\mathbb{R}^d} f(v)p_{s,t}(x,v)dv_1...dv_d,$$

which concludes this step of the proof.

Step 2: We prove (2.17). Note that by a standard density argument of $\mathfrak{S}$ in $L^2(\Omega)$, we can assume $f \in \mathcal{C}^1(\mathbb{R}^d) \cap \mathcal{E}_b(\mathbb{R}^d)$.
We prove (2.17) by induction, we introduce the following notations:

$$\widehat{h}_k^d(x) = \prod_{i=k}^{d}\frac{h_i(x_i)}{x_i}, \quad \widehat{h'}_k(x) = \prod_{i=1}^{k}h'_i(x_i), \quad x = (x_1, ..., x_d).$$

When $k = d$, we have by the chain rule $h'_d(S_s^d) = \frac{D_u^d h_d(S_s^d)}{D_u^d S_s^d}$ and $D_u^d S_s^d = \sigma_{dd}(u)S_s^d$, thus

$$\begin{aligned} E\left(f(S_t)\widehat{h'}_d(S_s)\right) &= E\left(\frac{1}{s}\int_0^s f(S_t)\widehat{h'}_{d-1}(S_s)\frac{D_u^d h_d(S_s^d)}{D_u^d S_s^d}du\right) \\ &= E\left(\frac{1}{s}\int_0^s f(S_t)\widehat{h'}_{d-1}(S_s)\frac{D_u^d h_d(S_s^d)}{\sigma_{dd}(u)S_s^d}du\right). \end{aligned}$$

Using Lemma 2.2 with

$$F = \frac{f(S_t)}{S_s^d}\prod_{i=1}^{d-1}h'_i(S_s^i) = \frac{f(S_t)}{S_s^d}\widehat{h'}_{d-1}(S_s)$$

and the fact that $\widehat{h'}_{d-1}(S_s)$ does not depend on the d^{th} coordinate of the Brownian motion yields

$$\begin{aligned} & E\left(\frac{1}{s}\int_0^s f(S_t)\widehat{h'}_{d-1}(S_s)\frac{D_u^d h_d(S_s^d)du}{\sigma_{dd}(u)S_s^d}\right) \\ = \;& E\left(Fh_d(S_s^d)\frac{1}{s}\int_0^s \frac{dW_u^d}{\sigma_{dd}(u)}\right) - E\left(h_d(S_s^d)\frac{1}{s}\int_0^s D_u^d\frac{\widehat{h'}_{d-1}(S_s)f(S_t)}{S_s^d}\frac{du}{\sigma_{dd}(u)}\right) \quad (2.19) \\ = \;& E\left(Fh_d(S_s^d)\frac{1}{s}\int_0^s \frac{dW_u^d}{\sigma_{dd}(u)}\right) - E\left(\widehat{h'}_{d-1}(S_s)h_d(S_s^d)\frac{1}{s}\int_0^s D_u^d\frac{f(S_t)}{S_s^d}\frac{du}{\sigma_{dd}(u)}\right). \end{aligned}$$

Besides using Lemma 2.1 for the Malliavin derivative of $f(S_t)$, we get for $v \in]s,t[$

$$\frac{1}{\sigma_{dd}(u)}D_u^d\left[\frac{f(S_t)}{S_s^d}\right] = \frac{1}{S_s^d\sigma_{dd}(v)}D_v^d f(S_t) - \frac{f(S_t)}{S_s^d}.$$

Thus, the value of the last term of (2.19) is given by

$$\begin{aligned} E\left(\widehat{h'}_{d-1}(S_s)h_d(S_s^d)\frac{1}{s}\int_0^s D_u^d\frac{f(S_t)}{S_s^d}\frac{du}{\sigma_{dd}(u)}\right) &= -E\left(\widehat{h'}_{d-1}(S_s)h_d(S_s^d)\frac{f(S_t)}{S_s^d}\right) \\ &+E\left(\widehat{h'}_{d-1}(S_s)\frac{h_d(S_s^d)}{S_s^d}\frac{1}{t-s}\int_s^t D_v^d f(S_t)\frac{dv}{\sigma_{dd}(v)}\right). \end{aligned}$$

And by duality (2.7) we remove the Malliavin derivative of $f(S_t)$ in the last term of the previous equality

$$\begin{aligned} & E\left(\frac{\widehat{h'}_{d-1}(S_s)h_d(S_s^d)}{S_s^d}\frac{1}{t-s}\int_s^t\frac{D_v^d f(S_t)dv}{\sigma_{dd}(v)}\right) \\ & \quad = \quad E\left(\frac{\widehat{h'}_{d-1}(S_s)h_d(S_s^d)}{S_s^d}E\left\{\frac{1}{t-s}\int_s^t\frac{D_v^d f(S_t)dv}{\sigma_{dd}(v)}\Big|\mathcal{F}_s\right\}\right) \\ & \quad = \quad E\left(\frac{\widehat{h'}_{d-1}(S_s)h_d(S_s^d)}{S_s^d}E\left\{f(S_t)\frac{1}{t-s}\int_s^t\frac{dW_v^d}{\sigma_{dd}(v)}\Big|\mathcal{F}_s\right\}\right). \end{aligned}$$

Regrouping all terms together gives

$$E\left(f(S_t)\widehat{h'}_d(S_s)\right) = E\left(f(S_t)\Gamma_{s,t}^d\widehat{h'}_{d-1}(S_s)\widehat{h}_d^d(S_s)\right), \quad \Gamma_{s,t}^d = \pi_{s,t}^{d,d}.$$

Now, let us suppose that (2.17) is satisfied for k and prove it for $k-1$. We have by the chain rule $h_k'(S_s^k) = \frac{D_u^k h_k(S_s^k)}{D_u^k S_s^k}$ and $D_u^k S_s^k = \sigma_{kk}(u)S_s^k$, thus

$$\begin{aligned} E\left(f(S_t)\widehat{h}'_{d-1}(S_s)\right) &= E\left(f(S_t)\Gamma_{s,t}^{k+1}\widehat{h}_{k+1}^{d}(S_s)\widehat{h}'_{k}(S_s)\right) \\ &= E\left(\frac{1}{s}\int_0^s f(S_t)\Gamma_{s,t}^{k+1}\widehat{h}_{k+1}^{d}(S_s)\widehat{h}'_{k-1}(S_s)\frac{D_u^k h_k(S_s^k)}{\sigma_{kk}(u)S_s^k}du\right) \\ &= E\left(\frac{1}{s}\int_0^s \frac{f(S_t)\Gamma_{s,t}^{k+1}\widehat{h}_{k+1}^{d}(S_s)\widehat{h}'_{k-1}(S_s)}{S_s^k}\frac{D_u^k h_k(S_s^k)}{\sigma_{kk}(u)}du\right). \end{aligned}$$

Using Lemma 2.2 with

$$F = \frac{f(S_t)\Gamma_{s,t}^{k+1}\widehat{h}_{k+1}^{d}(S_s)\widehat{h}'_{k-1}(S_s)}{S_s^k}$$

and the fact that $\widehat{h}'_{k-1}(S_s)$ does not depend on the j^{th} coordinate ($j \geq k$) of the Brownian motion yields

$$\begin{aligned} E\left(\frac{1}{s}\int_0^s \frac{FD_u^k h_k(S_s^k)}{\sigma_{kk}(u)}du\right) &= \sum_{j=k}^{d} E\left(Fh_k(S_s^k)\frac{1}{s}\int_0^s \rho_{jk}(u)dW_u^j\right) \\ -\sum_{j=k}^{d} E\left(h_k(S_s^k)\widehat{h}'_{k-1}(S_s)\frac{1}{s}\int_0^s D_u^j\left[\frac{f(S_t)\widehat{h}_{k+1}^{d}(S_s)\Gamma_{s,t}^{k+1}}{S_s^k}\right]\rho_{jk}(u)du\right). \end{aligned} \tag{2.20}$$

Besides, if for $x = (x_1, ..., x_d)$ we denote $\Pi(x) = \frac{\widehat{h}_{k+1}^{d}(x)}{x_k}$, the Malliavin derivative of the last term of (2.20) provides

$$\begin{aligned} D_u^j\left[\Gamma_{s,t}^{k+1}\Pi(S_s)f(S_t)\right] = D_u^j\Gamma_{s,t}^{k+1}\Pi(S_s)f(S_t) &+ \Gamma_{s,t}^{k+1}D_u^j\Pi(S_s)f(S_t) \\ &+ \Gamma_{s,t}^{k+1}\Pi(S_s)D_u^j f(S_t). \end{aligned}$$

Using Lemma 2.1 for the Malliavin derivative in the two last terms, we get

$$\sum_{j=k}^{d}\rho_{jk}(u)D_u^j\Pi(S_s) = S_s^k\partial_{x_k}\Pi(S_s) = -\Pi(S_s), \tag{2.21}$$

$$\sum_{j=k}^{d}\rho_{jk}(u)D_u^j f(S_t) = S_t^k\partial_{x_k}f(S_t). \tag{2.22}$$

From (2.21), we deduce that

$$\widehat{h}'_{k-1}(S_s)h_k(S_s^k)f(S_t)\Gamma_{s,t}^{k+1}\frac{1}{s}\int_0^s\sum_{j=k}^{d}\rho_{jk}(u)D_u^j\Pi(S_s)du = -\frac{\widehat{h}'_{k-1}(S_s)\widehat{h}_k^{d}(S_s)f(S_t)\Gamma_{s,t}^{k+1}}{S_s^k}.$$

Thus, introducing the random variable $\widetilde{\pi}_{s,t}^{k,d} = 1 + \frac{1}{s}\sum_{j=k}^{d}\int_0^s \rho_{jk}(u)dW_u^j$ and using (2.20)

$$\begin{aligned} E\left(\frac{1}{s}\int_0^s \frac{FD_u^k h_k(S_s^k)}{\sigma_{kk}(u)}du\right) &= E\left(\frac{\widehat{h}_k^d(S_s)\widehat{h'}_{k-1}(S_s)f(S_t)\Gamma_{s,t}^{k+1}}{S_s^k}\widetilde{\pi}_{s,t}^{k,d}\right) \\ &- E\left(\frac{\widehat{h}_k^d(S_s)\widehat{h'}_{k-1}(S_s)f(S_t)}{S_s^k}\frac{1}{s}\int_0^s\sum_{j=k}^{d}\rho_{jk}(u)D_u^j\Gamma_{s,t}^{k+1}du\right) \quad (2.23) \\ &- E\left(\frac{\widehat{h}_k^d(S_s)\widehat{h'}_{k-1}(S_s)\Gamma_{s,t}^{k+1}}{S_s^k}\frac{1}{t-s}\int_s^t\sum_{j=k}^{d}\rho_{jk}(u)D_u^j f(S_t)du\right), \end{aligned}$$

where we used the fact that $\sum_{j=k}^{d}\rho_{jk}(u)D_u^j f(S_t)$ does not depend on u (see (2.22)). Let us develop the last term of (2.23)

$$\begin{aligned} &E\left(\frac{\widehat{h}_k^d(S_s)\widehat{h'}_{k-1}(S_s)\Gamma_{s,t}^{k+1}}{S_s^k}\frac{1}{t-s}\int_s^t\sum_{j=k}^{d}\rho_{jk}(u)D_u^j f(S_t)du\right) \\ &= E\left(\frac{\widehat{h}_k^d(S_s)\widehat{h'}_{k-1}(S_s)}{S_s^k}\sum_{j=k}^{d}E\left[\frac{1}{t-s}\int_s^t\Gamma_{s,t}^{k+1}\rho_{jk}(u)D_u^j f(S_t)du\Big|\mathcal{F}_s\right]\right) \\ &= E\left(\frac{\widehat{h}_k^d(S_s)\widehat{h'}_{k-1}(S_s)}{S_s^k}\sum_{j=k}^{d}E\left[f(S_t)\frac{1}{t-s}\int_s^t\Gamma_{s,t}^{k+1}\rho_{jk}(u)\delta W_u^j\Big|\mathcal{F}_s\right]\right) \\ &= \sum_{j=k}^{d}E\left(\frac{f(S_t)\widehat{h}_k^d(S_s)\widehat{h'}_{k-1}(S_s)\Gamma_{s,t}^{k+1}}{S_s^k}\frac{1}{t-s}\int_s^t\rho_{jk}(u)dW_u^j\right) \\ &- \sum_{j=k}^{d}E\left(\frac{f(S_t)\widehat{h}_k^d(S_s)\widehat{h'}_{k-1}(S_s)}{S_s^k}\frac{1}{t-s}\int_s^t\rho_{jk}(u)D_u^j\Gamma_{s,t}^{k+1}du\right). \end{aligned}$$

We applied (2.7) in the second equality to remove the Malliavin derivative of $f(S_t)$. We also used (2.8) in the last equality. To complete the proof, we should remark that

$$\frac{1}{s}\int_0^s D_u^j\Gamma_{s,t}^{k+1}\rho_{jk}(u)du - \frac{1}{t-s}\int_s^t D_v^j\Gamma_{s,t}^{k+1}\rho_{jk}(v)dv = -\int_0^t D_y^j\Gamma_{s,t}^{k+1}D_y^j\pi_{s,t}^{k,d}dy$$

and because $\Gamma_{s,t}^{k+1}$ is an $\mathcal{F}_t^{k+1,\ldots,d}$-measurable random variable ($\mathcal{F}_t^{i,\ldots,d}$ defined in section 2.2)

$$\Gamma_{s,t}^{k} = \Gamma_{s,t}^{k+1}\pi_{s,t}^{k,d} - \sum_{j=k}^{d}\int_0^t D_u^j\Gamma_{s,t}^{k+1}D_u^j\pi_{s,t}^{k,d}du = \Gamma_{s,t}^{k+1}\pi_{s,t}^{k,d} - \sum_{j=k+1}^{d}\int_0^t D_u^j\Gamma_{s,t}^{k+1}D_u^j\pi_{s,t}^{k,d}du.$$

■

Although $D_u^j \pi_{s,t}^{k,d} = \varphi_{jk}(u)$, note that the Malliavin derivative of $\Gamma_{s,t}^{k+1}$ intervenes in the induction (2.13) which is difficult to compute numerically. Consequently, we propose in Theorem 2.2 a new formula which enables us to get rid of the Malliavin derivatives and its computation can be easily done using (2.27).

We will use in Theorem 2.2 the set of the second order permutations $\overline{\mathcal{P}}_{k,d}$ defined as the following

$$\overline{\mathcal{P}}_{k,d} = \{p \in \mathcal{P}_{k,d};\ p \circ p = Id\}, \tag{2.24}$$

where $\mathcal{P}_{k,d}$ is the set of permutations on $\{k, ..., d\}$ and Id is the identity application. Because the second order permutations on $\{k, ..., d\}$ can be written as a combination of disjoint transpositions on this set, one can prove the following recursive relation between those permutations defined on $\{k, ..., d\}$ and the one defined on $\{k+1, ..., d\}$

$$\overline{\mathcal{P}}_{k,d} = \{\tau_k^k \circ p;\ p \in \overline{\mathcal{P}}_{k+1,d}\} \cup \{\tau_k^l \circ p;\ p \in \overline{\mathcal{P}}_{k+1,d},\ p(l) = l,\ l \in \{k+1, ..., d\}\}, \tag{2.25}$$

with the transposition application $\tau_i^j : i \leftrightarrow j$ defined on $\{k, ..., d\}$ as the application that swaps only i to j and j to i. We also denote by Δ the quasi-determinant that involves only the permutations of $\overline{\mathcal{P}}_{k,d}$, that is to say, the Δ associated to the matrix $C = \{C_{i,j}\}_{k \le i,j \le d}$ is given by

$$\Delta = \sum_{p \in \overline{\mathcal{P}}_{k,d}} \epsilon(p) \prod_{i=1}^{d} C_{i,p(i)}$$

where $\epsilon(p)$ is the signature of the permutation p.

$$\begin{vmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{vmatrix} = \epsilon(\tau_{11})\, C_{11} \Delta_{11} + \epsilon(\tau_{12})\, C_{12} C_{21} \Delta_{12} + \epsilon(\tau_{13})\, C_{13} C_{31} \Delta_{13}$$

$$= C_{11} \begin{vmatrix} \cancel{C_{11}} & \cancel{C_{12}} & \cancel{C_{13}} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{vmatrix} - C_{12} C_{21} \begin{vmatrix} \cancel{C_{11}} & \cancel{C_{12}} & \cancel{C_{13}} \\ \cancel{C_{21}} & \cancel{C_{22}} & \cancel{C_{23}} \\ C_{31} & C_{32} & C_{33} \end{vmatrix} + C_{13} C_{31} \begin{vmatrix} \cancel{C_{11}} & \cancel{C_{12}} & \cancel{C_{13}} \\ C_{21} & C_{22} & C_{23} \\ \cancel{C_{31}} & \cancel{C_{32}} & \cancel{C_{33}} \end{vmatrix}$$

$$= C_{11}(C_{22}C_{33} - C_{32}C_{23}) - C_{12}C_{21}C_{33} + C_{13}C_{31}C_{22}$$

FIGURE 2.1 – Illustration of the computation of Δ (2.27) for $d = 3$ and $k = 1$.

Theorem 2.2 *Based on the assumptions of Theorem 2.1, for $k \in \{1, ..., d\}$ the value of $\Gamma^k_{s,t}$ is given by*

$$\Gamma^k_{s,t} = \sum_{p \in \overline{\mathcal{P}}_{k,d}} \epsilon(p) \prod_{i=k}^{d} A_{i,p(i)}, \tag{2.26}$$

with $\epsilon(p)$ as the signature of the permutation $p \in \overline{\mathcal{P}}_{k,d}$, $\overline{\mathcal{P}}_{k,d}$ defined in (2.24) and

$$A = \begin{pmatrix} \pi^{1,d}_{s,t} & C_{1,2} & C_{1,3} & \cdots & C_{1,d} \\ 1 & \pi^{2,d}_{s,t} & C_{2,3} & \cdots & C_{2,d} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & \cdots & 1 & \pi^{d-1,d}_{s,t} & C_{d-1,d} \\ 1 & 1 & \cdots & 1 & \pi^{d,d}_{s,t} \end{pmatrix},$$

where $C_{k,l}$ is the covariance of $\pi^{k,d}_{s,t}$ and $\pi^{l,d}_{s,t}$.

Remark: In the theorem above, $C_{k,l}$ admits a closed-form expression because $\pi^{k,d}_{s,t}$ and $\pi^{l,d}_{s,t}$ are two correlated Gaussian variables whose general value is given in Theorem 2.1. Please remark that $\Gamma^1_{s,t} = \Gamma_{s,t}$ is a quasi-determinant that involves only the permutations of $\overline{\mathcal{P}}_{1,d}$ and for a quasi-determinant Δ, associated to an arbitrary $(d-k+1)\times(d-k+1)$ matrix $C = \{C_{i,j}\}_{k \le i,j \le d}$, whose permutations are in $\overline{\mathcal{P}}_{k,d}$, we use (2.25) to prove easily that

$$\Delta = C_{k,k}\Delta_{k,k} + \sum_{i=k+1}^{d} \epsilon(\tau^i_k) C_{i,k} C_{k,i} \Delta_{k,i}, \tag{2.27}$$

where $\Delta_{k,i}$ is the quasi-determinant associated to the $C^{i,k}$ obtained from C by suppressing the line and the column i as well as the line and the column k. Based on the development according to the first line, relation (2.27) provides a recursive formula which is even more efficient than the determinant formula. Of course, we can generalize the relation (2.27) to the one that involves the development according to a j^{th} line or a j^{th} column with $k \le j \le d$. In Figure 2.1, we provide an illustration of the computation of Δ when C is a 3×3 matrix.

Proof of Theorem 2.2:

We prove (2.26) by a decreasing induction. For $k = d$, the expression (2.26) is clearly satisfied. We suppose that (2.26) is satisfied for $k + 1$ and we prove it for k. According to Theorem 2.1,

$\Gamma^k_{s,t} = \Gamma^{k+1}_{s,t}\pi^{k,d}_{s,t} - \sum_{j=k+1}^d \int_0^t D^j_u \Gamma^{k+1}_{s,t} D^j_u \pi^{k,d}_{s,t} du$, but

$$\begin{aligned} D^j_u \Gamma^{k+1}_{s,t} &= \sum_{l=k+1}^d \sum_{p\in\overline{\mathcal{P}}_{k+1,d}} \epsilon(p) \prod_{i=k+1, i\neq l}^d A_{i,p(i)} D^j_u A_{l,p(l)} \\ &= \sum_{l=k+1}^d \sum_{p\in\overline{\mathcal{P}}_{k+1,d}, p(l)=l} \epsilon(p) \prod_{i=k+1, i\neq l}^d A_{i,p(i)} D^j_u A_{l,l}, \end{aligned}$$

the second equality is due to the fact that $A_{l,p(l)}$ is a constant except for $p(l) = l$. Subsequently

$$\begin{aligned} & -\sum_{j=k+1}^d \int_0^t D^j_u \Gamma^{k+1}_{s,t} D^j_u \pi^{k,d}_{s,t} du \\ = & -\sum_{l=k+1}^d \sum_{p\in\overline{\mathcal{P}}_{k+1,d}, p(l)=l} \epsilon(p) \prod_{i=k+1, i\neq l}^d A_{i,p(i)} \sum_{j=k+1}^d \int_0^t D^j_u A_{l,l} D^j_u \pi^{k,d}_{s,t} \\ = & -\sum_{l=k+1}^d \sum_{p\in\overline{\mathcal{P}}_{k+1,d}, p(l)=l} \epsilon(p) \prod_{i=k+1, i\neq l}^d A_{i,p(i)} C_{k,l}. \end{aligned}$$

Finally

$$\begin{aligned} \Gamma^k_{s,t} &= \Gamma^{k+1}_{s,t}\pi^{k,d}_{s,t} - \sum_{j=k+1}^d \int_0^t D^j_u \Gamma^{k+1}_{s,t} D^j_u \pi^{k,d}_{s,t} du \\ &= \pi^{k,d}_{s,t} \sum_{p\in\overline{\mathcal{P}}_{k+1,d}} \epsilon(p) \prod_{i=k+1}^d A_{i,p(i)} - \sum_{l=k+1}^d C_{k,l} \sum_{p\in\overline{\mathcal{P}}_{k+1,d}, p(l)=l} \epsilon(p) \prod_{i=k+1, i\neq l}^d A_{i,p(i)} \\ &= \sum_{p\in\overline{\mathcal{P}}_{k,d}} \epsilon(p) \prod_{i=k}^d A_{i,p(i)}. \end{aligned}$$

The last equality is due to the development of $\sum_{p\in\overline{\mathcal{P}}_{k,d}} \epsilon(p) \prod_{i=k}^d A_{i,p(i)}$ according to the k^{th} line of A which can be justified by (2.25).

■

As a corollary of Theorem 2.1 and Theorem 2.2, we obtain the following result for the multidimensional Black & Scholes model with independent coordinates

Corollary 2.1 *For any $s \in]0,t[$, $g \in \mathcal{E}_b(\mathbb{R}^d)$ and $x = (x_1, ..., x_d)$ with $x_i > 0$, if $\sigma_{ij}(t) = \sigma_i \delta(i-j)$ then*

$$E\left(g(S_t)\Big|S_s = x\right) = \frac{T_{s,t}[g](x)}{T_{s,t}[1](x)},$$

with

$$T_{s,t}[f](x) = E\left(f(S_t)\prod_{k=1}^{d}\frac{H_k(S_s^k)W_{s,t}^k}{\sigma_k s(t-s)S_s^k}\right), \tag{2.28}$$

and

$$W_{s,t}^k = (t-s)(W_s^k + \sigma_k s) - s(W_t^k - W_s^k), \quad k = 1, ..., d.$$

When compared to the previous results, the proof of this corollary is rather easy because the $\left\{\pi_{s,t}^{k,d}\right\}_{1\leq k\leq d}$ are independent and equal to $\pi_{s,t}^k = 1 + \frac{W_s^k}{s\sigma_k} - \frac{W_t^k - W_s^k}{(t-s)\sigma_k}$.

2.4 Extension to the multidimensional Heston model

In this section, we consider the multidimensional Heston model

$$\text{for } 1 \leq i \leq d \quad \begin{array}{c} d\nu_t^i = \kappa_i(\theta_i - \nu_t^i)dt + \eta_i\sqrt{\nu_t^i}d\widetilde{Z}_t^i, \quad \nu_0^i = y^i, \\ dS_t^i = S_t^i\left(r_i dt + \sqrt{\nu_t^i}dZ_t^i\right), \quad S_0^i = z^i, \end{array} \tag{2.29}$$

where $(Z^1, ..., Z^d, \widetilde{Z}^1, ..., \widetilde{Z}^d)$ is a vector of correlated Brownian motions with R as a non-singular constant correlation matrix. The first step is to rewrite (2.29) using independent Brownian motions by the Cholesky decomposition of $R = LL'$ where L is a lower triangular matrix which provides

$$\begin{pmatrix} d\nu_t^1 \\ \vdots \\ d\nu_t^d \\ dS_t^1 \\ \vdots \\ dS_t^d \end{pmatrix} = \begin{pmatrix} \kappa_1(\theta_1 - \nu_t^1) \\ \vdots \\ \kappa_d(\theta_d - \nu_t^d) \\ r_1 S_t^1 \\ \vdots \\ r_d S_t^d \end{pmatrix} dt + diag\begin{pmatrix} \eta_1\sqrt{\nu_t^1} \\ \vdots \\ \eta_d\sqrt{\nu_t^d} \\ \sqrt{\nu_t^1}S_t^1 \\ \vdots \\ \sqrt{\nu_t^d}S_t^d \end{pmatrix} L \begin{pmatrix} d\widetilde{W}_t^1 \\ \vdots \\ d\widetilde{W}_t^d \\ dW_t^1 \\ \vdots \\ dW_t^d \end{pmatrix}, \begin{array}{c} \nu_0^i = y^i \\ S_0^i = z^i \end{array}, \tag{2.30}$$

where $(\widetilde{W}^1, ..., \widetilde{W}^d, W^1, ..., W^d)$ is a vector of independent Brownian motions. Because the matrix L is a lower triangular matrix, conditionally to the Brownian motions $(\widetilde{W}^1, ..., \widetilde{W}^d)$, the dynamics of the asset vector $S = (S^1, ..., S^d)$ is similar to the one given in (2.10). This basic argument is the first we use to extend the results of the previous section to the multidimensional

Heston model and it can also be used with other stochastic volatility models. Indeed, it allows us to use the Malliavin calculus directly on $(W^1, ..., W^d)$ as in the previous section and to completely forget the dependence on $(\widetilde{W}^1, ..., \widetilde{W}^d)$. It is worth noting that the conditioning method is widely used for stochastic volatility models, we refer the reader for example to [13]. The second argument used in our extension is based on the following result, proved in [21].

Lemma 2.3 $E\left[\left(\int_0^t \nu_s^i ds\right)^r\right]$ *is finite for all* $r \in \mathbb{R}$ *and* $i \in \{1, ..., d\}$.

Before stating Theorem 2.3, we decompose the matrix L into three $d \times d$ blocks and we define the matrix $\sigma(u)$ using the constant third block σ.

$$L = \begin{pmatrix} \sigma' & 0 \\ \sigma'' & \sigma \end{pmatrix}, \quad \sigma(u) = \begin{pmatrix} \sqrt{\nu_u^1}\sigma_{11} & 0 & ... & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \sqrt{\nu_u^{d-1}}\sigma_{d-11} & ... & \sqrt{\nu_u^{d-1}}\sigma_{d-1d-1} & 0 \\ \sqrt{\nu_u^d}\sigma_{d1} & ... & \sqrt{\nu_u^d}\sigma_{dd-1} & \sqrt{\nu_u^d}\sigma_{dd} \end{pmatrix} \tag{2.31}$$

Theorem 2.3 *For any* $s \in]0, t[$ *let*

$$\Gamma_{s,t} = \sum_{p \in \overline{\mathcal{P}}_{1,d}} \epsilon(p) A_{1,p(1)} A_{2,p(2)} ... A_{d,p(d)} = \sum_{p \in \overline{\mathcal{P}}_{1,d}} \epsilon(p) \prod_{i=1}^{d} A_{i,p(i)}, \tag{2.32}$$

with $\epsilon(p)$ *as the signature of the permutation* $p \in \overline{\mathcal{P}}_{1,d}$, $\overline{\mathcal{P}}_{1,d}$ *defined in (2.24) and*

$$A = \begin{pmatrix} \pi_{s,t}^{1,d} & C_{1,2} & C_{1,3} & \cdots & C_{1,d} \\ 1 & \pi_{s,t}^{2,d} & C_{2,3} & \cdots & C_{2,d} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & \cdots & 1 & \pi_{s,t}^{d-1,d} & C_{d-1,d} \\ 1 & 1 & \cdots & 1 & \pi_{s,t}^{d,d} \end{pmatrix},$$

$$\pi_{s,t}^{k,d} = 1 + \sum_{j=k}^{d} \int_0^t \varphi_{jk}(u) dW_u^j, \quad \varphi_{jk}(u) = \frac{1}{s} \frac{\rho_{jk}}{\sqrt{\nu_u^k}} 1_{u \in]0,s[} - \frac{1}{t-s} \frac{\rho_{jk}}{\sqrt{\nu_u^k}} 1_{u \in]s,t[}, \tag{2.33}$$

and

$$C_{k,l} = \sum_{j=k}^{d} \frac{\rho_{jk}\rho_{jl}}{s^2} \int_0^s \frac{du}{\sqrt{\nu_u^k \nu_u^l}} + \sum_{j=k}^{d} \frac{\rho_{jk}\rho_{jl}}{(t-s)^2} \int_s^t \frac{du}{\sqrt{\nu_u^k \nu_u^l}}. \tag{2.34}$$

ρ is the inverse matrix $\rho = \sigma^{-1}$ and σ is the third-block matrix in the decomposition (2.31). If there is $1 < q < \infty$ such that $\Gamma_{s,t} \in L^q(\Omega)$ then, for $g \in \mathcal{E}_b(\mathbb{R}^d)$ and $x = (x_1, ..., x_d)$ with $x_i > 0$

$$E\left(g(S_t)\Big|S_s = x\right) = \frac{T_{s,t}[g](x)}{T_{s,t}[1](x)}, \tag{2.35}$$

where $T_{s,t}[f](x)$ is defined for every function $f \in \mathcal{E}_b(\mathbb{R}^d)$ by

$$T_{s,t}[f](x) = E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^{d}\frac{H_k(S_s^k)}{S_s^k}\right). \tag{2.36}$$

Proof of Theorem 2.3:

To prove Theorem 2.3, it is sufficient to prove the following recursive relation for $k = 0$, $h_i \in \mathcal{C}_b^\infty(\mathbb{R})$ and $f \in \mathcal{E}_b(\mathbb{R}^d)$

$$E\left(E\left[f(S_t)\prod_{i=1}^{d}h_i'(S_s^i)\Big|\widetilde{\mathcal{F}}_t\right]\right) = E\left(E\left[f(S_t)\Gamma_{s,t}^{k+1}\prod_{i=1}^{k}h_i'(S_s^i)\prod_{i=k+1}^{d}\frac{h_i(S_s^i)}{S_s^i}\Big|\widetilde{\mathcal{F}}_t\right]\right), \tag{2.37}$$

where $\widetilde{\mathcal{F}}_t$ is the completed filtration generated by $(\widetilde{W}^1, ..., \widetilde{W}^d)$ until t. If we subdivide this proof into two steps, *Step 2* is similar to the one in the proof of Theorem 2.1 because, as we said earlier, conditionally to $(\widetilde{W}^1, ..., \widetilde{W}^d)$, the processes $\{\nu_t^i\}_{1\le i\le d}$ can be considered as deterministic. Moreover the expression of $\Gamma_{s,t}$ can be found in the same fashion as for Theorem 2.2.

Step 1: ((2.37) with $k = 0$) $\Rightarrow$ (2.35).

Let $\phi \in \mathcal{C}_c^\infty(\mathbb{R})$ be a mollifier function with support equal to $[-1, 1]$ and such that $\int_{\mathbb{R}}\phi(u)du = 1$, then for any $u \in \mathbb{R}$ we define

$$h_{mk}(u) = (H_k * \phi_m)(u) \in \mathcal{C}_b^\infty(\mathbb{R}), \quad \phi_m(u) = m\phi(mu).$$

If the equality (2.37) is correct for any k, then it is correct for $k = 0$ which means

$$E\left(f(S_t)\prod_{k=1}^{d}h_{mk}'(S_s^k)\right) = E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^{d}\frac{h_{mk}(S_s^k)}{S_s^k}\right).$$

The proof of the convergence

$$E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^{d}\frac{h_{mk}(S_s^k)}{S_s^k}\right)\longrightarrow E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^{d}\frac{H_k(S_s^k)}{S_s^k}\right)$$

is also similar to the one in the proof of Theorem 2.1, the only difference is due to the replacement of the Cauchy-Schwartz inequality by the Hölder inequality that uses the L^q-boundedness of $\Gamma_{s,t}$.

Besides, $h'_{mk}(u_k) = \int_{\mathbb{R}} H_k(v_k)\phi'_m(u_k - v_k)dv_k = \phi_m(u_k - x_k)$ and the distribution of the vector $(S_s^1, ..., S_s^d, S_t^1, ..., S_t^d)$, conditionally to $(\widetilde{W}^1, ..., \widetilde{W}^d)$, admits a log-normal joint density with respect to the Lebesgue measure on $\mathbb{R}_+^d \times \mathbb{R}_+^d$, we denote it by $\widetilde{p}_{s,t}(u, v)$ with $u = (u_1, ..., u_d)$ and $v = (v_1, ..., v_d)$ where

$$\widetilde{p}_{s,t}(u, v) = \frac{1}{(\det(\Sigma_1)\det(\Sigma_2))^{\frac{1}{2}}}q(u, v)$$

and

$$q(u, v) = \frac{\prod_{i=1}^{d} 1_{u_i>0, v_i>0}}{(2\pi)^d \prod_{i=1}^{d} u_i \prod_{i=1}^{d} v_i}\exp\begin{pmatrix} -\frac{1}{2}(\ln u - d_1)'\Sigma_1^{-1}(\ln u - d_1) \\ -\frac{1}{2}(\ln v - \ln u - d_2)'\Sigma_2^{-1}(\ln v - \ln u - d_2)\end{pmatrix},$$

$d_1 = \left(sr_1 - \frac{1}{2}\Sigma_1^{11}, ..., sr_d - \frac{1}{2}\Sigma_1^{dd}\right)$, $d_2 = \left((t-s)r_1 - \frac{1}{2}\Sigma_2^{11}, ..., (t-s)r_d - \frac{1}{2}\Sigma_2^{dd}\right)$,
$\Sigma_1 = \int_0^s \sigma(w)\sigma'(w)dw$, $\Sigma_2 = \int_s^t \sigma(w)\sigma'(w)dw$ and $\sigma(w)$ is given in (2.31), thus

$$E\left(f(S_t)\prod_{k=1}^{d}h'_{mk}(S_s^k)\Big|\widetilde{\mathcal{F}}_t\right) = \frac{\int_{\mathbb{R}^d} f(v)\Big(\int_{\mathbb{R}^d}\prod_{k=1}^{d}\phi_m(u_k - x_k)q(u, v)du_1...du_d\Big)dv_1...dv_d}{(\det(\Sigma_1)\det(\Sigma_2))^{\frac{1}{2}}}$$

To prove the convergence

$$E\left(f(S_t)\prod_{k=1}^{d}h'_{mk}(S_s^k)\right)\longrightarrow E\left(\int_{\mathbb{R}^d} f(v)\widetilde{p}_{s,t}(x, v)dv_1...dv_d\right),$$

we should first remove the term $(\det(\Sigma_1)\det(\Sigma_2))^{\frac{1}{2}}$ using the Cauchy-Schwarz inequality thanks to Lemma 2.3, then we use the convergence of $\int_{\mathbb{R}^d}\prod_{k=1}^{d}\phi_m(u_k - x_k)q(u, v)du_1...du_d$ to $q(x, v)$ as in *Step 1* of the proof of Theorem 2.1.

■

In Theorem 2.3, we made the assumption that $\Gamma_{s,t} \in L^q(\Omega)$ and one should find the parameters κ_i, θ_i and η_i of ν_u^i that fulfill this condition. In this article, we test only the one-dimensional

Heston model for which the Feller conditions are sufficient to ensure that $\Gamma_{s,t} \in L^2(\Omega)$. Indeed, if $d = 1$, $\Gamma_{s,t} = \pi^{1,1}_{s,t}$ and it is sufficient to prove that $\int_0^t \frac{du}{\nu_u^1} \in L^1(\Omega)$ which is given in the following lemma. Because $d = 1$, in the lemma below, we remove the dimension index.

Lemma 2.4 *If $\kappa \geq 0$ and $2\kappa\theta \geq \eta^2$ then $E\left(\int_0^t \frac{du}{\nu_u}\right)$ is finite.*

Proof of Lemma 2.4:
According to Lemma A.2. in [9]

$$E \exp\left(\frac{\eta^2\left(\frac{2\kappa\theta}{\eta^2} - 1\right)^2}{8}\int_0^t \frac{du}{\nu_u}\right) < \infty$$

and the finiteness of $E\left(\int_0^t \frac{du}{\nu_u}\right)$ follows directly from the application of the Jensen's inequality on the logarithmic function.

■

2.5 Variance reduction method based on conditioning

As was said in the previous section, conditionally to the Brownian motions that generate the volatilities, studying the stochastic volatility model (2.30) is equivalent to studying the MEDC model (2.10). Thus, except for Theorem 2.6, in this section we suppose that the price of the asset S_t is given by (2.10) for which we show that one can reduce the variance by a projection on $L^2\left(\left\{\int_0^t \sigma_{ij}(u)dW_u^j\right\}_{i,j}\right)$ and by using the closed-form expression of $T_{s,t}[1](x)$.

We begin with $T_{s,t}[1](x)$, we can compute the explicit value of this function of x. The $T_{s,t}[1](x)$ closed formula can be got, for instance, from a change of probability. Indeed, we define the probability $\mathbb{P} = N_{coeff}(\prod_{k=1}^d S_0^k/S_s^k)P$ which yields

$$T_{s,t}[1](x) = \frac{1}{N_{coeff}}\mathbb{E}\left(\left[\prod_{k=1}^d H_k(S_s^k)\right]\Gamma_{s,t}\right),$$

N_{coeff} is a deterministic normalization coefficient such that $M_s = N_{coeff}(\prod_{k=1}^d S_0^k/S_s^k)$ is an exponential martingale with $E(M_s) = 1$. Under $\mathbb{P}$, $\Gamma_{s,t}$ has the same law as a polynomial of Gaussian variables which is sufficient to conduct the computations.

Let us now denote

$$h(x, w_{ij}) = E\left(\Gamma_{s,t} \prod_{k=1}^{d} \frac{H_k(S_s^k)}{S_s^k} | \left\{ \int_0^t \sigma_{ij}(u) dW_u^j \right\}_{1 \leq j \leq i \leq d} = \{w_{ij}\}_{1 \leq j \leq i \leq d}\right) \tag{2.38}$$

In what follows, we are going to prove that the function $h(x, \{w_{ij}\}_{1 \leq j \leq i \leq d})$ can be explicitly known if, for each j, the $(d-k)\times(d-k)$ matrix $\Sigma_{jt} = \left\{\Sigma_{jt}^{ik}\right\}_{j \leq i,k \leq d} = \left\{\int_0^t \sigma_{ij}(u)\sigma_{kj}(u)du\right\}_{j \leq i,k \leq d}$ is invertible. First, please note that according to our notations $i - j + 1$ and $k - j + 1$ are the indices of the element Σ_{jt}^{ik} in the matrix Σ_{jt} (we will use a similar convention for A^j, B^j, Ψ_{jt} and Φ_{jt}). Also, we notice that the invertibility condition of Σ_{jt} is not an important constraint, because one can choose a time discretization $\{t_m\}$ such that the matrices $\{\Sigma_{jt_m}\}_{k \leq d}$ fulfill this condition [2].

The computation of $h(x, \{w_{ij}\}_{1 \leq j \leq i \leq d})$ is based on a regression of Gaussian variables according to the Gaussian variables $Y_{ij} = \int_0^t \sigma_{ij}(u) dW_u^j$. First, we perform a linear regression of $\int_0^t \varphi_{jk}(u) dW_u^j$ according to Y_{ij}

$$\int_0^t \varphi_{jk}(u) dW_u^j = \sum_{i=j}^{d} a_{i,k}^j Y_{ij} + X_{jk}, \tag{2.39}$$

where $\{X_{jk}\}_{1 \leq k \leq j \leq d}$ is a Gaussian vector $\mathcal{N}(0, C_X)$ orthogonal to Y. Using Itô isometry twice and the orthogonality of Y and X, we obtain

$$E\left(\int_0^t \varphi_{jk}(u) dW_u^j Y_{lj}\right) = \int_0^t \varphi_{jk}(u)\sigma_{lj}(u)du = \sum_{i=j}^{d} \Sigma_{jt}^{li} a_{i,k}^j.$$

If we denote $A^j = \{a_{i,k}^j\}_{j \leq i,k \leq d}$ and $\Psi_{jt} = \left\{\int_0^t \varphi_{jk}(u)\sigma_{lj}(u)du\right\}_{k,l}$, we get

$$A^j = \Sigma_{jt}^{-1} \Psi_{jt}.$$

In the same way, we perform a linear regression of $\int_0^s \sigma_{kj}(u) dW_u^j$ according to Y_{ij}

$$\int_0^s \sigma_{kj}(u) dW_u^j = \sum_{i=j}^{d} b_{i,k}^j Y_{ij} + Z_{kj}, \tag{2.40}$$

where $\{Z_{kj}\}_{1 \leq j \leq k \leq d}$ is a Gaussian vector $\mathcal{N}(0, C_Z)$ orthogonal to Y. Using Itô isometry twice

2. Nevertheless, this is a difficult task when the dimension is sufficiently big.

and the orthogonality of Y and Z, we obtain

$$E\left(\int_0^s \sigma_{kj}(u)dW_u^j Y_{lj}\right) = \int_0^s \sigma_{kj}(u)\sigma_{lj}(u)du = \sum_{i=j}^{d} \Sigma_{jt}^{li} b_{i,k}^j.$$

If we denote $B^j = \{b_{i,k}^j\}_{j\leq i,k\leq d}$, we get

$$B^j = \Sigma_{jt}^{-1}\Sigma_{js}.$$

Now using (2.39), (2.40) and the value of A and B, the covariance matrices C_X, C_Z and $C_{XZ} = E(XZ)$ are given by ($\Phi_{jt}^{i,k} = \int_0^t \varphi_{ji}(u)\varphi_{jk}(u)du$)

$$[C_X]_{i,k}^j = E(X_{ji}X_{jk}) = \Phi_{jt}^{i,k} - (A_k^j)'\Psi_{jt}^i - (A_i^j)'\Psi_{jt}^k + (A_k^j)'\Sigma_{jt}A_i^j,$$

$$[C_Z]_{i,k}^j = E(Z_{ij}Z_{kj}) = \Sigma_{js}^{i,k} - (B_k^j)'\Sigma_{js}^i - (B_i^j)'\Sigma_{js}^k + (B_k^j)'\Sigma_{jt}B_i^j,$$

$$[C_{XZ}]_{i,k}^j = E(X_{ji}Z_{kj}) = \Psi_{js}^{i,k} - (A_k^j)'\Sigma_{js}^i - (B_i^j)'\Psi_{jt}^k + (A_k^j)'\Sigma_{jt}B_i^j.$$

Employing (2.39) and (2.40), we express $\Gamma_{s,t}$ and S_s^k according to Y_{ij}, Z_{ij} and X_{ji} then we conduct standard Gaussian computations to obtain the expression of $h(x, w_{ij})$ [3]. In Theorem 2.4, we give an explicit expression of $T_{s,t}[1](x)$ and $h(x, w_{ij})$ in the case of multidimensional B&S models with independent coordinates.

Regarding the model (2.10), we see that now that we know the explicit value of $T_{s,t}[1](x)$ and $h(x, \{w_{ij}\}_{1\leq j\leq i\leq d})$, subsequently, we should choose between the simulation of:

P1) N paths of $g(S_t)h\left(x, \{\int_0^t \sigma_{ij}(u)dW_u^j\}_{i,j}\right)$ then set the continuation value to

$$C(x) := \frac{\frac{1}{N}\sum_{l=1}^{N} e^{-r\Delta t} g^l(S_t) h\left(x, \{\int_0^t \sigma_{ij}(u)dW_u^j\}_{1\leq j\leq i\leq d}^l\right)}{T_{s,t}[1](x)}.$$

P2) N' paths of $g(S_t)h\left(x, \{\int_0^t \sigma_{ij}(u)dW_u^j\}_{i,j}\right)$ and N paths of $h\left(x, \{\int_0^t \sigma_{ij}(u)dW_u^j\}_{i,j}\right)$ then

3. One can use Mathematica to compute it formally.

set the continuation value to

$$C(x) := \frac{\frac{1}{N'}\sum_{l=1}^{N'} e^{-r\Delta t} g^l(S_t) h\left(x, \{\int_0^t \sigma_{ij}(u) dW_u^j\}_{1\le j\le i\le d}^l\right)}{\frac{1}{N}\sum_{l=1}^{N} h\left(x, \{\int_0^t \sigma_{ij}(u) dW_u^j\}_{1\le j\le i\le d}^l\right)}.$$

We will see in sections 2.7.2 and 2.7.3 that sometimes it is preferable to use **P2)**.

In the case of the multidimensional Heston model, please note that $T_{s,t}[1](x)$ is not given explicitly, however we explicitly know the value of the function

$$h(x, \{w_{ij}\}_{j\le i}) = E\left(1_{S_s\ge x}\Gamma_{s,t}\prod_{k=1}^{d}\frac{H_k(S_s^k)}{S_s^k}\Big|\widetilde{\mathcal{F}}_t \bigvee \{\int_0^t \sqrt{\nu_u^i} dW_u^j\}_{j\le i} = \{w_{ij}\}_{j\le i}\right), \quad (2.41)$$

for $1 \le j, i \le d$. Thus, we will exclusively use a **P2)** alike procedure, that is to say, simulate N' paths of $g(S_t)h\left(x, \{\int_0^t \sqrt{\nu_u^i} dW_u^j\}_{i,j}\right)$ and N paths of $h\left(x, \{\int_0^t \sqrt{\nu_u^i} dW_u^j\}_{i,j}\right)$ then set the continuation value to

$$C(x) := \frac{\frac{1}{N'}\sum_{l=1}^{N'} e^{-r\Delta t} g^l(S_t) h^l\left(x, \{\int_0^t \sqrt{\nu_u^i} dW_u^j\}_{1\le j\le i\le d}\right)}{\frac{1}{N}\sum_{l=1}^{N} h^l\left(x, \{\int_0^t \sqrt{\nu_u^i} dW_u^j\}_{1\le j\le i\le d}\right)}.$$

In this approximation of the continuation value, the trajectory index l is on the function h because it resulted from a conditioning according to $\widetilde{\mathcal{F}}_t$ and consequently h is not deterministic.

We provide in Theorem 2.4, Theorem 2.5 and Theorem 2.6 the expression of the conditioning for three cases that will be tested in section 2.7. The proofs of these theorems are given in the appendix. Unlike in the Theorem 2.4, in Theorem 2.5 and Theorem 2.6 we only give the expression of the function h because we will only use the procedure **P2)** for the simulation.

Theorem 2.4 *We suppose that S_t has the dynamics (2.10) and $\sigma_{ij}(t) = \sigma_{ij}\delta(i-j)$ then, by conditioning, the function h defined in (2.38) and the denominator $T_{s,t}[1](x)$ given in Theorem 2.1 have the following values*

$$T_{s,t}[1](x) = \prod_{k=1}^{d} \frac{e^{(\sigma_k^2 - r_k)s}}{\sigma_k S_0^k} \frac{1}{\sqrt{s2\pi}} e^{-\frac{\widetilde{d}_{x_k}^2}{2}}, \quad \widetilde{d}_{x_k} = \frac{\ln\left(\frac{x_k}{S_0^k}\right) - r_k s + \frac{3\sigma_k^2 s}{2}}{\sigma_k\sqrt{s}}$$

and

$$h(x, \{w_k\}_{1\le k\le d}) = \prod_{k=1}^{d} \frac{e^{(\sigma_k^2 - r_k)s}}{\sigma_k S_0^k} \sqrt{\frac{t}{s(t-s)2\pi}} \exp\left(\frac{-s\sigma_k}{t}\left(\frac{s\sigma_k}{2} + w_k\right) - \frac{(d_{x_k}(w_k))^2}{2}\right),$$

with

$$d_{x_k}(w_k) = \left(\ln\left(\frac{x_k}{S_0^k}\right) - r_k s + \frac{3\sigma_k^2 s}{2} - (s\sigma_k + w_k)\frac{s\sigma_k}{t}\right) \Big/ \left(\sigma_k\sqrt{s(t-s)/t}\right).$$

Theorem 2.5 *We suppose that S_t has the dynamics (2.10) and that $d = 2$ with ρ, σ_1 and σ_2 three constants such that*

$$\sigma(u) = \begin{pmatrix} \sigma_1 & 0 \\ \rho\sigma_2 & \sqrt{1-\rho^2}\sigma_2 \end{pmatrix}, \quad |\rho| < 1.$$

By conditioning, the expression of the function h defined in (2.38) is given by

$$\begin{aligned} h(x, w_1, w_2) &= E\left(\Gamma_{s,t}\prod_{k=1}^{2}\frac{H_k(S_s^k)}{S_s^k}\Big|W_t^1 = w_1, W_t^2 = w_2\right) \\ &= \frac{e^{\left(\frac{(\sigma_1^2+\sigma_2^2)}{2} - r_1 - r_2\right)s + \frac{s(t-s)(\sigma_1^2+2\rho\sigma_1\sigma_2+\sigma_2^2)}{2t} - \frac{(\sigma_1+\rho\sigma_2)sw_1 + \sigma_2\sqrt{1-\rho^2}sw_2}{t}}}{S_0^1 S_0^2}\Lambda_{x_1,x_2,w_1,w_2}, \end{aligned} \tag{2.42}$$

with

$$\begin{aligned} \Lambda_{x_1,x_2,w_1,w_2} &= \frac{t\rho(\sigma_2-\sigma_1)}{s(t-s)(1-\rho^2)\sigma_1^2\sigma_2^2}\Lambda^1_{x_1,x_2,w_1,w_2} + \frac{t\left((1-\rho^2)\sigma_2+\rho^2\sigma_1\right)}{s(t-s)\sigma_1\sigma_2^2 2\pi\sqrt{1-\rho^2}}\Lambda^2_{x_1,x_2,w_1,w_2} \\ &+ \sqrt{\frac{t}{s(t-s)}}\frac{\rho(\sigma_1-\sigma_2)}{\sigma_1\sigma_2}\left[1 + \frac{d_2\sqrt{1-\rho^2}}{\sigma_2}\sqrt{\frac{t}{s(t-s)}}\right]\frac{1}{\sqrt{2\pi}}\Lambda^3_{x_1,x_2,w_1,w_2}, \end{aligned}$$

and

$$\begin{aligned} \Lambda^1_{x_1,x_2,w_1,w_2} &= \frac{1}{2\pi\sqrt{1-\rho^2}}\int_{-\infty}^{d_1}\int_{-\infty}^{d_2\sqrt{1-\rho^2}} e^{-\frac{u_1^2+u_2^2-2\rho u_1 u_2}{2(1-\rho^2)}}du_1du_2, \\ \Lambda^3_{x_1,x_2,w_1,w_2} &= e^{-\frac{(1-\rho^2)d_2^2}{2}}\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{d_1}{\sqrt{1-\rho^2}}-\rho d_2} e^{-\frac{u^2}{2}}du, \ \Lambda^2_{x_1,x_2,w_1,w_2} = e^{-\frac{d_1^2}{2(1-\rho^2)}+\frac{\rho d_1 d_2}{\sqrt{1-\rho^2}}-\frac{d_2^2}{2}}, \end{aligned}$$

where d_1 and d_2 are functions of x_1,x_2,w_1 and w_2

$$\begin{aligned} d_1(x_1, w_1) &= \frac{\ln\left(\frac{S_0^1}{x_1}\right) + r_1 s + \frac{\sigma_1 sw_1}{t} - \frac{s(3t-2s)\sigma_1^2}{2t} - \frac{s(t-s)\rho\sigma_1\sigma_2}{t}}{\sigma_1\sqrt{\frac{s(t-s)}{t}}}, \\ d_2(x_2, w_1, w_2) &= \frac{\ln\left(\frac{S_0^2}{x_2}\right) + r_2 s + \frac{\sigma_2\rho sw_1}{t} + \frac{\sigma_2\sqrt{1-\rho^2}sw_2}{t} - \frac{s(3t-2s)\sigma_2^2}{2t} - \frac{s(t-s)\rho\sigma_1\sigma_2}{t}}{\sqrt{1-\rho^2}\sigma_2\sqrt{\frac{s(t-s)}{t}}}. \end{aligned}$$

Theorem 2.6 *For $d = 1$, we suppose that S_t satisfies (2.29), $\sigma(u) = \sqrt{1-\rho^2}\sqrt{\nu_u}$ with $|\rho| < 1$ and $\kappa > 0$, $2\kappa\theta > \eta^2$. By conditioning, the expression of the function h defined in (2.41) is given*

by

$$h(x,w)=\frac{F\left(\int_0^t\nu_u du,\int_0^s\nu_u du\right)}{S_0\sqrt{2\pi(1-\rho^2)}}\exp\left(\begin{array}{c}-d_x^2(w)-rs-\sqrt{1-\rho^2}\frac{\int_0^s\nu_u du}{\int_0^t\nu_u du}w-\rho\int_0^s\sqrt{\nu_u}d\widetilde{W}_u\\+\left(1-\frac{\rho^2}{2}\right)\int_0^s\nu_u du-\frac{1-\rho^2}{2}\frac{\left(\int_0^s\nu_u du\right)^2}{\int_0^t\nu_u du}\end{array}\right),$$

with

$$d_x(w)=\frac{\ln\left(\frac{S_0}{x}\right)+rs+\sqrt{1-\rho^2}\left(\frac{\int_0^s\nu_u du}{\int_0^t\nu_u du}\right)w+\rho\int_0^s\sqrt{\nu_u}d\widetilde{W}_u+\left(\rho^2-\frac{3}{2}\right)\int_0^s\nu_u du+\frac{(1-\rho^2)\left(\int_0^s\nu_u du\right)^2}{\int_0^t\nu_u du}}{\sqrt{1-\rho^2}\sqrt{\left(\int_0^t\nu_u du\right)\left(\int_0^s\nu_u du\right)-\left(\int_0^s\nu_u du\right)^2/\sqrt{\int_0^t\nu_u du}}},$$

and

$$F\left(\int_0^t\nu_u du,\int_0^s\nu_u du\right)=\frac{\sqrt{\int_0^t\nu_u du}}{\sqrt{\int_0^t\nu_u du\int_0^s\nu_u du-\int_0^s\nu_u du\int_0^s\nu_u du}}.$$

2.6 Bias reduction method

At each time step we estimate the conditional expectation by computing the quotient Q

$$\frac{E(X)}{E(Y)}\sim\frac{\frac{1}{N'}\sum_{i=1}^{N'}X_i}{\frac{1}{N}\sum_{i=1}^{N}Y_i}=Q, \tag{2.43}$$

where $\{X_i\}_{1\leq i\leq N'}$ and $\{Y_i\}_{1\leq i\leq N}$ are respectively independent copies of the square integrable random variables X,Y. Although with some models the expectation $E(Y)$ is known explicitly, the approximation of $E(X)$ by $\frac{1}{N'}\sum_{i=1}^{N'}X_i$ already produces a bias on the overall simulation of the price of the American option (we refer the reader to [25] for more details). In addition to that, if $E(Y)$ is not known explicitly, the estimator Q is also biased which can be easily checked when $\{X_i\}_{1\leq i\leq N'}$ and $\{Y_i\}_{1\leq i\leq N}$ are independent. The combination of the two bias produces an unpredictable bias on the price of the Bermudan option that approaches the American option price.

Using the notations

$$A=E(X),\ B=E(Y),\ \sigma_1^2=Var(X),\ \sigma_2^2=Var(Y)\ and\ \rho=Cov(X,Y)/(\sigma_1\sigma_2), \tag{2.44}$$

in section 2.7.2, we will observe on some experiments that

$$N' = \frac{N}{2} \ if \ A^2\sigma_2^2 \geq B^2\sigma_1^2 \ \ and \ \ N = \frac{N'}{2} \ if \ A^2\sigma_2^2 \leq B^2\sigma_1^2, \tag{2.45}$$

reduces significantly the bias of the overall simulation. Moreover, in this section, we are going to verify that the choice (2.45) does not increase significantly the variance of the estimator Q.

Indeed, if $|E(Y_i)| \geq \varepsilon > 0$, Q converges to $E(X_i)/E(Y_i)$ and in the following theorem we will study asymptotically the error of this estimator when acting on the relation between N and N' and we consider the two cases:

case 1: $N' = \lceil \lambda_1 N \rceil$ with $\lambda_1 \in]0,1[$, then (2.43) becomes

$$Q = \frac{\frac{1}{N'}\sum_{i=1}^{N'} X_i}{\frac{1}{N}\left(\frac{N'}{N'}\sum_{i=1}^{N'} Y_i + \frac{N-N'}{N-N'}\sum_{i=N'+1}^{N} Y_i\right)} = g_1(A_{N'}, B_{N'}, B_{N,N'}),$$

where

$$\begin{gathered} g_1(x,y,z) = x/(\lambda_1 y + (1-\lambda_1)z), \\ A_{N'} = \tfrac{1}{N'}\sum_{i=1}^{N'} X_i, \quad B_{N'} = \tfrac{1}{N'}\sum_{i=1}^{N'} Y_i, \quad B_{N,N'} = \tfrac{1}{N-N'}\sum_{i=N'+1}^{N} Y_i. \end{gathered} \tag{2.46}$$

case 2: $N = \lceil \lambda_2 N' \rceil$ with $\lambda_2 \in]0,1[$, then (2.43) becomes

$$Q = \frac{\frac{1}{N'}\left(\frac{N}{N}\sum_{i=1}^{N} X_i + \frac{N'-N}{N'-N}\sum_{i=N+1}^{N'} X_i\right)}{\frac{1}{N}\sum_{i=1}^{N} Y_i} = g_2(A_N, A_{N',N}, B_N),$$

where

$$\begin{gathered} g_2(x,y,z) = (\lambda_2 x + (1-\lambda_2)y)/z, \\ A_N = \tfrac{1}{N}\sum_{i=1}^{N} X_i, \quad A_{N',N} = \tfrac{1}{N'-N}\sum_{i=N+1}^{N'} X_i, \quad B_N = \tfrac{1}{N}\sum_{i=1}^{N} Y_i. \end{gathered} \tag{2.47}$$

Theorem 2.7 *Based on the notations (2.44) and the variables defined in (2.46) and (2.47), if* $|B| > 0$ *then as* $N \to \infty$ *and* $N' \to \infty$

$$g_1(A_{N'}, B_{N'}, B_{N,N'}) \overset{a.s.}{\longrightarrow} \frac{A}{B}, \quad \sqrt{N}\left(g_1(A_{N'}, B_{N'}, B_{N,N'}) - \frac{A}{B}\right) \overset{law}{\longrightarrow} \mathcal{N}(0, \Sigma_1(\lambda_1)),$$

$$\Sigma_1(\lambda_1) = \frac{1}{\lambda_1}\frac{\sigma_1^2}{B^2} + \frac{4A^2\sigma_2^2}{B^4}\left(\frac{1}{4} - \frac{B\sigma_1\rho}{2A\sigma_2}\right) \tag{2.48}$$

and

$$g_2(A_N, A_{N',N}, B_N) \overset{a.s.}{\longrightarrow} \frac{A}{B}, \quad \sqrt{N'}\left(g_2(A_N, A_{N',N}, B_N) - \frac{A}{B}\right) \overset{law}{\longrightarrow} \mathcal{N}(0, \Sigma_2(\lambda_2)),$$

$$\Sigma_2(\lambda_2) = \frac{1}{\lambda_2}\frac{A^2\sigma_2^2}{B^4} + \frac{4\sigma_1^2}{B^2}\left(\frac{1}{4} - \frac{A\sigma_2\rho}{2B\sigma_1}\right). \tag{2.49}$$

Proof of Theorem 2.7:

Since the computations are similar for the **case 2**, we give only the proof associated to the **case 1**. First, the variables $A_{N'}$, $B_{N'}$ and $B_{N,N'}$ are square integrable thanks to the fact that the variables X, Y involved in Q are square integrable. The almost sure convergence of $g_1(A_{N'}, B_{N'}, B_{N,N'})$ follows from the law of large numbers and from the continuity of g_1 at (A, B, B). For the same reasons, the gradient vector $\nabla g_1(A_{N'}, B_{N'}, B_{N,N'})$ converges a.s. to $\nabla g_1(A, B, B)$. Besides

$$\begin{aligned}
&\sqrt{N}(g_1(A_{N'}, B_{N'}, B_{N,N'}) - g_1(A, B, B)) \\
&\quad = \sqrt{\tfrac{N}{N'}}\sqrt{N'}(A_{N'} - A)\tfrac{\partial g_1}{\partial x}(A, B, B) + \sqrt{\tfrac{N}{N'}}\sqrt{N'}(B_{N'} - B)\tfrac{\partial g_1}{\partial y}(A, B, B) \\
&\quad + \sqrt{\tfrac{N}{N-N'}}\sqrt{N - N'}(B_{N,N'} - B)\tfrac{\partial g_1}{\partial z}(A, B, B) + \sqrt{N}||\Delta_N||\epsilon_N,
\end{aligned}$$

where $\Delta_N = (A_{N'} - A, B_{N'} - B, B_{N,N'} - B)$ and ϵ_N convergences a.s to 0. Using the Slutsky Theorem, with $G = (G_1, G_2, G_3) \sim \mathcal{N}(0, C)$ and the continuity of $(x, y) \mapsto xy$ and $(x, y) \mapsto x + y$ provide

$$\begin{aligned}
&\sqrt{N}(g_1(A_{N'}, B_{N'}, B_{N,N'}) - g_1(A, B, B)) \\
&\overset{law}{\longrightarrow} \sqrt{\frac{1}{\lambda_1}}\frac{\partial g_1}{\partial x}(A, B, B)G_1 + \sqrt{\frac{1}{\lambda_1}}\frac{\partial g_1}{\partial y}(A, B, B)G_2 + \sqrt{\frac{1}{1 - \lambda_1}}\frac{\partial g_1}{\partial z}(A, B, B)G_3,
\end{aligned}$$

with

$$C = \begin{pmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho & 0 \\ \sigma_1\sigma_2\rho & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_2^2 \end{pmatrix},$$

which allows us to compute $\Sigma_1(\lambda_1)$.

■

Theorem 2.7 tells us that one should use $\lambda_1 = 1$ and $\lambda_2 = 1$ to reduce the variance. But, as

we said previously, we prefer to use (2.45) because it reduces the bias of the overall simulation of the price and it also does not change a lot the variance of the estimator Q. Indeed, for instance, if $A^2\sigma_2^2 \gg B^2\sigma_1^2$ it does not really matter to multiply the term $\frac{\sigma_1^2}{B^2}$ in (2.48) by two.

Finally, when $E(Y)$ is known explicitly, we point out that even though the estimator Q is biased, it has sometimes smaller variance than the estimator $\frac{\frac{1}{N'}\sum_{i=1}^{N'} X_i}{E(Y)}$. Indeed, for $\lambda_1 = 1$ or $\lambda_2 = 1$ and from Theorem 2.7, the variance produced by Q is equal to $\frac{\sigma_1^2}{B^2} + \frac{A^2\sigma_2^2}{B^4} - \frac{2A\sigma_1\sigma_2\rho}{B^3}$ which can be lower than $\frac{\sigma_1^2}{B^2}$ for some values of ρ.

2.7 Simulation and numerical results

In this section, we perform three sets of tests that involve the results of Theorem 2.4, Theorem 2.5 and Theorem 2.6. But before that, we study the parallel adaptability of MCM as well as some considerations that one should respect when using GPUs to reduce the execution time.

2.7.1 Parallel considerations

To manage CPU (Central Processing Unit) power dissipation, the processor makers have oriented their architectures to multi-cores. This switch in technology led us to study the pricing algorithms based on Monte Carlo for multi-core and many-core architectures using CPUs and GPUs (Graphics Processing Units) in [4] and [2]. In the latter articles we basically studied the impact of using GPUs instead of CPUs for pricing European options using MC and American options using the Longstaff and Schwartz algorithm [42]. The results of this study prove that we can greatly decrease the execution time and the energy consumed during the simulation. Unlike the LS method that uses a regression phase which is difficult to parallelize according to [2], the MCM is a squared [4] Monte Carlo method which is more adapted to multi-core and many-core environments than the LS method. Moreover, since using MCM without localization does not involve any parametric regression, higher dimensional problems can be dealt with more easily and the accuracy of results depends only on the number of simulated trajectories.

Let us study the parallel adaptability of MCM for parallel architectures. In Figure 2.2, we present the speedup of parallelizing [5] MCM on the four cores of the CPU instead of implementing it on only one core. We notice that the speedup increases quickly according to the number of simulated trajectories and it reaches a saturation state for > 9000 trajectories. For a large

4. What we mean by squared Monte Carlo is not necessarily simulating a square number of trajectories, but a Monte Carlo simulation that requires a Monte Carlo estimation, for each path, of an intermediate value (here the continuation value) and this can be done by using the same set of trajectories as the first Monte Carlo simulation.

5. We use OpenMP directives.

dimensional problem, the maximum speedup obtained is greater than the number of physical cores [6] on the CPU which indicates that MCM is very appropriate for parallel architectures. We point out, however, that our parallelization of MCM is done on the trajectories [7], so the speedup is invariable according to dimensions and time steps.

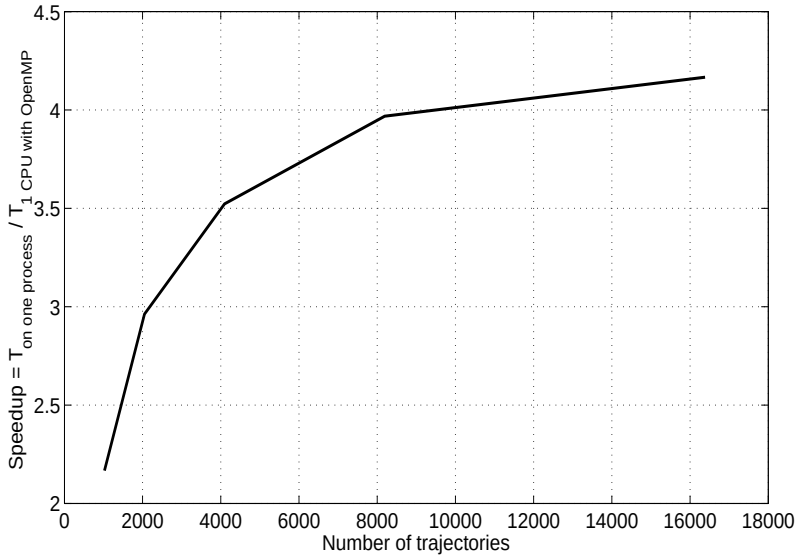

FIGURE 2.2 – The speedup of using all the CPU cores according to the number of trajectories.

Regarding GPU implementation, we also use a path parallelization of simulations. In Figure 2.3, we present the speedup of parallelizing [8] MCM on the GPU instead of implementing it on the four cores of the CPU. The speedup increases quickly not only according to the number of simulated trajectories, but also according to the dimension of the contract. The latter fact can be easily explained by the memory hierarchy of the GPU [48]. The speedups provided in Figure 2.3 prove, once again, the high adaptability of MCM on parallel architectures.

MCM is well suited to parallel architecture because it is completely based on Monte Carlo, unlike the Longstaff-Schwartz algorithm that performs a regression which cannot be efficiently parallelized. Indeed, for a regression that uses less than 10 polynomials, the Longstaff-Schwartz algorithm have almost the same behavior on the CPU as the one described for MCM in Figure 2.2. However, the many-core GPU implementation of the Longstaff-Schwartz algorithm is at most two times faster than its multi-core CPU implementation. For more details, we refer the reader to [2] which compares the GPU implementation of the Longstaff-Schwartz algorithm to its one-core CPU implementation.

6. which is due to hyper-threading.
7. which is the most natural procedure of parallelizing Monte Carlo.
8. We use CUDA language.

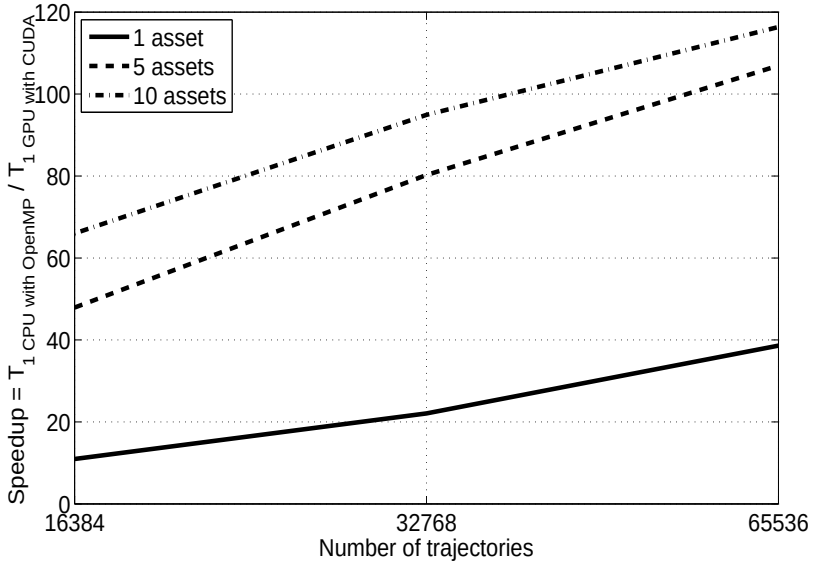

FIGURE 2.3 – The speedup of using the GPU instead of the CPU cores according to the number of trajectories.

Although MCM is based on Monte Carlo, one should, at least, respect the four points below when programming MCM on GPUs:

- Reduce the communication between CPU and GPU to its minimum, even more when implementing MCM whose trajectories are coupled, that is to say, one needs the other trajectories' data to simulate one value associated to one trajectory.
- Ensure the maximum coalescence of the data on the GPU because it affects greatly the execution time.
- Find the right compromise between the number of threads on each block and the number of blocks on the GPU when implementing MCM using less than 2^{16} trajectories.
- Saturate the GPU with as many instructions as possible thanks to the use of multi-streaming.

We refer the reader to [48] and [47] for more details on programming GPUs using CUDA.

2.7.2 Geometric average on independent B&S model

In this part, we simulate the prices associated to Theorem 2.4 and we test our simulations on a geometric average payoff that has the following expression

$$\Phi^d_{geo}(S_T) = \left(K - \prod_{i=1}^{d} (S_T^i)^{1/d} \right)_+ . \tag{2.50}$$

The parameters of the simulations are the following: The strike $K = 100$, the maturity $T = 1$, the risk neutral interest rate $r = \ln(1.1)$, the time discretization is defined using the time steps that is given as a parameter in each simulation, $S_0^i = 100$ and $\sigma_{ij}(t) = \sigma_{ij}(t)\delta(j-i)$ with $\sigma_{ii} = 0.2$. The true values, to which we compare our simulation results, are set using the one-dimensional equivalence and a tree method [12], available in Premia [32]. The values obtained by the latter procedure are very accurate and can be considered as the real prices of the American (not the Bermudan) contract.

As a continuation of what we began in section 2.6, we study the bias of the price of the American option when changing the values of $N(= \lceil \lambda_2 N' \rceil)$ or $N'(= \lceil \lambda_1 N \rceil)$ that intervene in the estimator (2.43). According to Figure 2.4, we see that the error of the overall simulation barely changes according to the value of λ in the one dimensional case. However, for five and ten dimensions the error is less important when we take $\lambda = 1/2$ as detailed in (2.45).

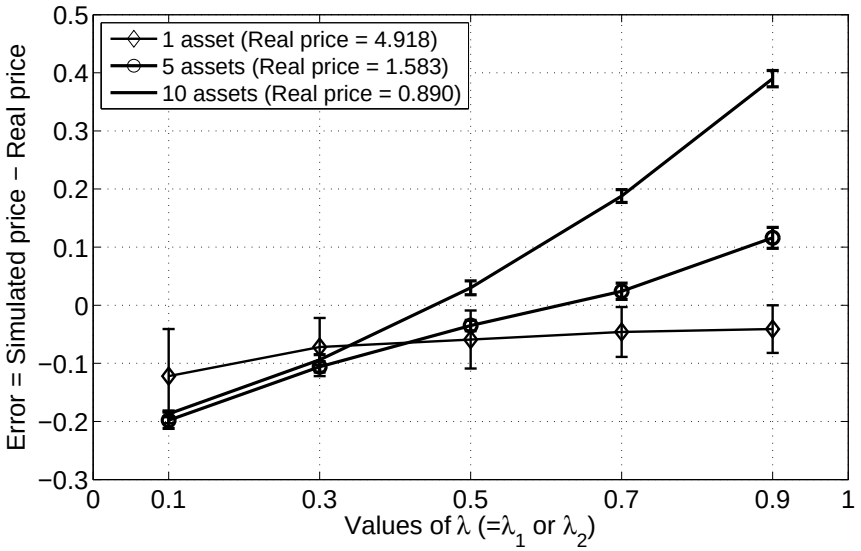

FIGURE 2.4 – *The bias of the simulation according to λ for thirty time steps and 2^{14} trajectories.*

In order to decide which value to use in (2.45), we should have a "sufficiently good" approximation of σ_1, σ_2, A and B. Consequently, we can implement one of the two methods below:

M0) If $B = T_{s,t}[1](x)$ and σ_2 are explicitly known, using all the simulated paths N_{max}, we approximate the values of σ_1 and A then we use either $\lambda_1 = 1/2$ or $\lambda_2 = 1/2$ to re-simulate Q.

M1) Using all the simulated paths N_{max}, we approximate the values of σ_1, σ_2, A and B then we use either $\lambda_1 = 1/2$ or $\lambda_2 = 1/2$ to re-simulate Q.

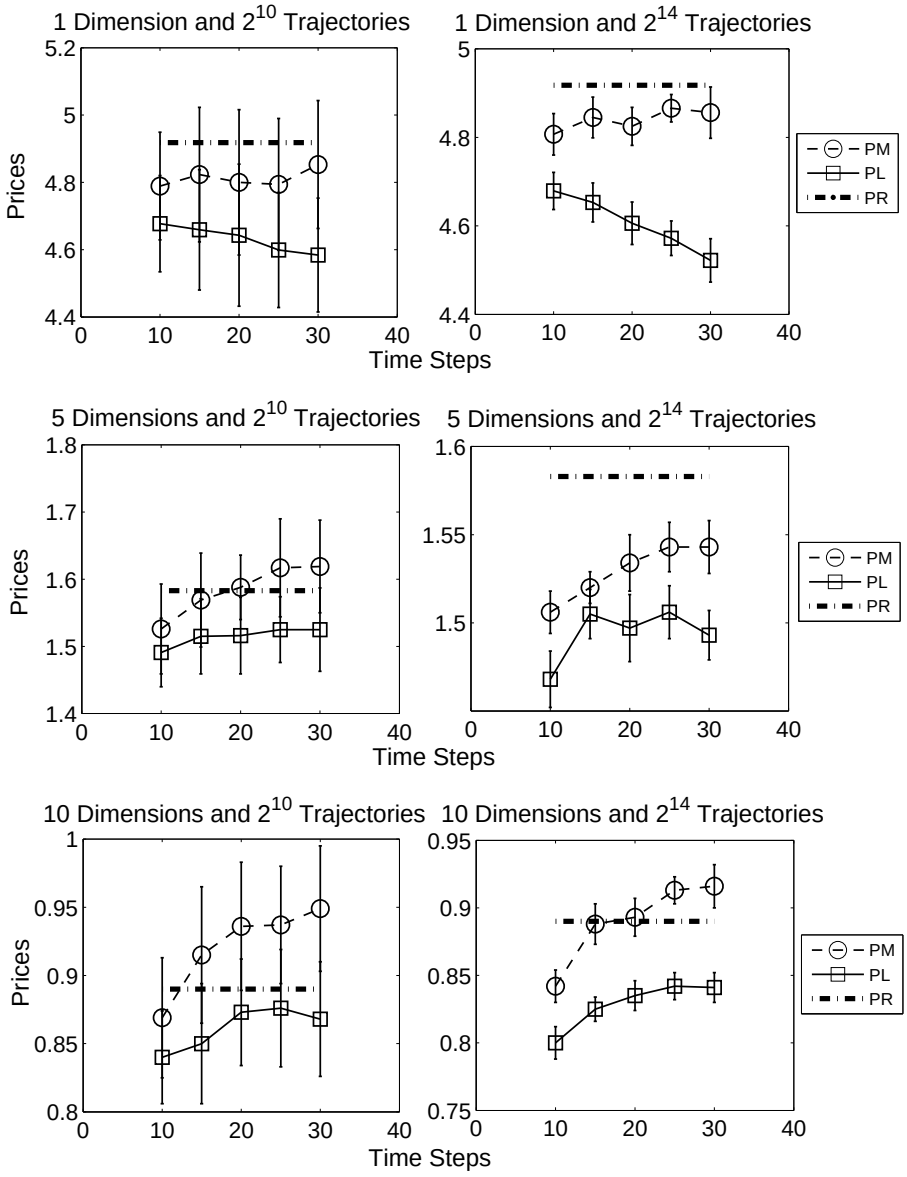

FIGURE 2.5 – *MCM Vs. LS for* $\Phi^d_{geo}(S_T)$*: PR is the real price. PM and PL are the prices obtained respectively by MCM and LS represented with their standard deviations.*

In Figure 2.5, we compare the **P2** ($\neq$) version of MCM with a standard LS algorithm. The LS is implemented using linear regression for multidimensional contracts and using up to three degree monomials for the one-dimensional contract. The reason behind the choice of linear regression in the multidimensional case is the fact that the regression phase of LS can really increase the execution time without a significant improvement of the prices tested.

In Figure 2.5, even if all the prices are sufficiently good, we see that MCM provides better prices than those of LS. Also when we increase the time steps, MCM is more stable than LS.

However, for $d = 10$ and time steps > 10, we remark that one should simulate 2^{14} trajectories to stablize MCM. This fact is expected due to the high variance of the ten dimension contract and that one should simulate more trajectories. The executions of MCM and LS with 2^{10} trajectories are carried out in less than one second. Moreover, using 2^{14} trajectories the LS and MCM are executed within seconds ($< 5s$). As a conclusion from this figure, MCM provides better results than LS in approximately the same execution time. When we increase the simulated trajectories to 2^{14}, the MCM prices are stabilized for high dimensions and are always better than LS prices.

In Table 2.1, we remain with the same payoff $\Phi^d_{geo}(S_T)$ but this time we compare the different nonparametric methods of implementing MCM. In **P2(**$=$**)** and **P2(**$\neq$**)**, we use the same **P2** method but with $N = N'$ for the first one and $N \neq N'$ for the second (the relation between N and N' is given in (2.45)). First, we notice that **P2(**$=$**)** is not stable in the multidimensional case and can give wrong results if the time steps > 10. This kind of bad results are also obtained for different values of the model parameters. However the **P2** method is stabilized when we implement the version $N \neq N'$ that reduces the bias. Also when we use 2^{10} trajectories, **P1** and **P2(**$\neq$**)** are almost similar. Nevertheless, with 2^{14} trajectories, **P2(**$\neq$**)** outperforms **P1**. As far as the execution time is concerned, the time consumed by **P2(**$\neq$**)** is not very different from **P1** when we use 2^{10} trajectories. In addition, using 2^{14} trajectories, to decide whether $N = N'/2$ or $N' = N/2$ is performed on the GPU independently on each trajectory and **P2(**$\neq$**)** is $< 5\%$ slower than **P1** for the tests that we have implemented.

2.7.3 Call on max and put on min on two-dimensional B&S model

In this part, we simulate the prices associated to Theorem 2.5 and we test our simulations on the American put on minimum and on the American call on maximum that have the following payoffs

$$\Phi_{min}(S_T) = \left(K - \min(S^1_T, S^2_T)\right)_+, \quad \Phi_{max}(S_T) = \left(\max(S^1_T, S^2_T) - K\right)_+. \tag{2.51}$$

The parameters of the simulations are the following: The strike $K = 100$, the maturity $T = 1$, the risk neutral interest rate $r = \ln(1.1)$, the time discretization is defined using the time steps that is given as a parameter in each simulation, $S^i_0 = 100$.

The true values, to which we compare our simulation results, are set using the Premia implementation of a finite difference algorithm [58] in two dimensions. Besides, we use the approximation presented in [20] for the bivariate cumulative distribution in the expression of

TABLE 2.1 – **P1** Vs. **P2** for $\Phi^d_{geo}(S_T)$: The real values are equal to 4.918, 1.583 and 0.890 for dimensions one, five and ten respectively

Simulated	Dim	Time		Price			Std Deviation	
Paths	d	Steps	**P1**	**P2**(=)	**P2**($\neq$)	**P1**	**P2**(=)	**P2**($\neq$)
2^{10}	1	10	4.750	4.826	4.789	0.213	0.167	0.160
2^{10}	1	20	4.729	4.880	4.800	0.270	0.226	0.216
2^{10}	1	30	4.679	4.909	4.853	0.270	0.179	0.190
2^{10}	5	10	1.548	1.681	1.526	0.071	0.073	0.067
2^{10}	5	20	1.632	> 2.0	1.588	0.070		0.048
2^{10}	5	30	1.650	> 2.3	1.619	0.074		0.069
2^{10}	10	10	0.900	1.112	0.869	0.039	0.045	0.044
2^{10}	10	20	0.921	> 1.3	0.936	0.043		0.047
2^{10}	10	30	0.908	> 1.5	0.949	0.035		0.046
2^{14}	1	10	4.738	4.812	4.807	0.057	0.046	0.047
2^{14}	1	20	4.675	4.869	4.825	0.047	0.044	0.043
2^{14}	1	30	4.638	4.876	4.856	0.072	0.059	0.058
2^{14}	5	10	1.487	1.526	1.506	0.057	0.012	0.012
2^{14}	5	20	1.504	1.639	1.534	0.047	0.021	0.016
2^{14}	5	30	1.508	> 1.8	1.543	0.072		0.015
2^{14}	10	10	0.845	0.938	0.842	0.013	0.015	0.012
2^{14}	10	20	0.901	> 1.2	0.893	0.012		0.014
2^{14}	10	30	0.923	> 1.3	0.916	0.015		0.016

$\Lambda^1_{x_1,x_2,w_1,w_2}$ (Theorem 2.5). For higher dimensions, we refer the reader to [24] for the approximation of the multivariate normal cumulative distribution.

Because of the bad results obtained previously with **P2**(=), we eliminate this method and we only consider **P2**($\neq$) and **P1**. In Table 2.2, we analyze the American put on minimum and the American call on maximum in two dimensions. As far as Φ_{min} is concerned, **P2**($\neq$) outperforms **P1** even when we use only 2^{10}. Regarding Φ_{max}, **P1** performs better than **P2**($\neq$) for 2^{10} trajectories which indicates that, because of the big variance produced by Φ_{max} relatively to Φ_{min}, the relation between N and N' is not well estimated. Simulating 2^{14} trajectories, we obtain similar results for **P1** and **P2**($\neq$) for Φ_{max}.

In Table 2.3, we show that our results are accurate even when $\rho \neq 0$ and when simulating only 2^{10} trajectories with **P2**($\neq$).

TABLE 2.2 – **P1** Vs. **P2** for Φ_{min} and Φ_{max}: Simulations for $\rho = 0$ and $\sigma_1 = \sigma_2 = 0.2$. The real values are equal to 8.262 and 21.15 respectively

Simulated Paths	The Payoff	Time Steps	Price **P1**	Price **P2**($\neq$)	Std Deviation **P1**	Std Deviation **P2**($\neq$)
2^{10}	Φ_{min}	10	7.734	7.986	0.190	0.248
2^{10}	Φ_{min}	20	7.618	7.895	0.257	0.270
2^{10}	Φ_{min}	30	7.564	7.920	0.224	0.263
2^{10}	Φ_{max}	10	21.03	20.33	0.66	0.86
2^{10}	Φ_{max}	20	20.46	19.38	0.61	0.73
2^{10}	Φ_{max}	30	19.73	18.13	0.73	0.93
2^{14}	Φ_{min}	10	7.755	8.088	0.058	0.067
2^{14}	Φ_{min}	20	7.584	8.098	0.098	0.052
2^{14}	Φ_{min}	30	7.467	8.087	0.082	0.043
2^{14}	Φ_{max}	10	20.96	20.91	0.09	0.24
2^{14}	Φ_{max}	20	20.58	20.56	0.16	0.16
2^{14}	Φ_{max}	30	20.36	20.05	0.15	0.22

TABLE 2.3 – Φ_{min} and Φ_{max}: Simulations for $\rho \neq 0$ and $\sigma_1 = \sigma_2 = 0.2$ using 2^{10} trajectories

Real values	The Payoff	Time Steps	Price $\rho = 0.5$	Price $\rho = -0.5$	Std Deviation $\rho = 0.5$	Std Deviation $\rho = -0.5$
$(+)7.23$	Φ_{min}	10	7.31	9.10	0.06	0.03
	Φ_{min}	20	7.47	9.29	0.07	0.06
$(-)9.05$	Φ_{min}	30	7.64	9.48	0.09	0.08
$(+)18.74$	Φ_{max}	10	18.78	23.23	0.53	0.35
	Φ_{max}	20	19.03	23.57	0.37	0.12
$(-)23.08$	Φ_{max}	30	19.29	23.94	0.19	0.20

TABLE 2.4 – Φ_{max}: Simulations for $\rho \neq 0$, $\sigma_1 = 0.1$ and $\sigma_2 = 0.2$ using 2^{10} trajectories

Time Steps	Price $\rho = 0.3$	Price -0.3	Price 0.7	Price -0.7	Std Deviation 0.3	Std Deviation -0.3	Std Deviation 0.7	Std Deviation -0.7
10	17.17	19.07	15.44	20.20	0.29	0.24	0.29	0.22
20	17.17	19.24	15.37	20.36	0.20	0.23	0.27	0.27
30	17.22	19.30	15.40	20.44	0.15	0.18	0.25	0.19
Real values	17.27	19.11	15.70	20.21				

When $\sigma_1 = \sigma_2$, the terms $\Lambda^1_{x_1,x_2,w_1,w_2}$ and $\Lambda^3_{x_1,x_2,w_1,w_2}$ of (2.42) do not intervene, thus in Table 2.4, we show that our results are also accurate even when $\sigma_1 \neq \sigma_2$ and when simulating

only 2^{10} trajectories with **P2($\neq$)**. We also considered, in Table 2.3, the case of highly correlated assets $|\rho| = 0.7$.

2.7.4 Put on Heston model

In this part, we simulate the prices associated to Theorem 2.6. Unlike the previous tests, in this part we do not know the real price of the American options. Thus, we will test the coherency of the results obtained with MCM to the results obtained using two different algorithms in Premia [32] (version 13). The methods to which we compare MCM are : The Longstaff-Schwartz(LS) method [42] implemented with a second order discretization scheme for the CIR process [5] and the Andersen-Broadie(AB) method [6] also implemented with a second order discretization scheme for the CIR process. We take the default parameters given in Premia for the two methods and we have tested various model parameters configurations including the one associated to Table 2.5: Dimension $d = 1$, maturity $T = 1$, strike $K = S_0 = 100$, $\nu_0 = 0.01$, $\kappa = 2$, $\theta = 0.01$, $\eta = 0.2$ and $r = \ln(1.1)$. Moreover, we have implemented MCM using the Milstein scheme for the CIR process and we used only 2^{10} trajectories with **P2($\neq$)** because we judged it sufficient for our simulations.

According to Table 2.5, the results obtained with MCM are coherent to the one obtained with LS and AB. In Table 2.5, we only present the results for the put option, but we obtained the same kind of coherence for the call option and even for high values of ν_0, θ and η but always under the Feller conditions.

TABLE 2.5 – Put option using 2^{10} trajectories and 50 time steps

Correlation(ρ)	Price(MCM)	Std Deviation(MCM)	LS	AB
-0.5	1.79	0.05	1.78	1.74
0.0	1.60	0.05	1.61	1.59
0.5	1.41	0.05	1.41	1.35

2.8 Conclusion and future work

In this article we provided, on the one hand, theoretical results that deal with the computation of the continuation value using the Malliavin calculus and how one can reduce the Monte Carlo variance only by conditioning. On the other hand, we presented numerical results related to a bias reduction method, to the accuracy of the prices obtained and the parallel adaptability of the MCM method on multi-core and many-core architectures.

As far as the theoretical results are concerned, based on the Malliavin calculus, we provided a generalization of the value of the continuation for the multidimensional models with deterministic and nonconstant triangular matrix $\sigma(t)$ as well as for the multidimensional Heston model. Moreover, we pointed out that one can judiciously reduce the variance by a simple conditioning method. Finally, we presented a very effective bias reduction method based on an appropriate choice of the number of trajectories used to approximate the quotient of two expectations.

Regarding the numerical part, we proved that instantaneous simulations on the CPU can be obtained using only 2^{10} trajectories and the results got with MCM are sufficient and better than with LS. Also, unlike LS, our nonparametric variance reduction implementation of MCM does not require parametric regression. Thus we improve the results of the simulation by only increasing the number of trajectories. Finally, increasing the number of trajectories is time consuming but MCM can be effectively parallelized on CPUs and GPUs. Indeed, for all the implemented tests, the MCM simulation of 2^{14} trajectories using the GTX 480 GPU can be performed within seconds ($< 5s$).

As future work, we plan to extend the results presented for the multidimensional Heston model to other stochastic volatility models. We will also look for a weaker and sufficient condition than the one presented in Lemma 2.4 for the Heston model and to extend it for the multidimensional case. Regarding the parallelization aspects, we are working on the parallelization of MCM on a CPU/GPU cluster using MPI+OpenMP+CUDA.

2.9 Appendix

Proof of Lemma 2.1:

The equality (2.15) can be easily proved. Indeed, using the chain rule

$$D_u^k f(S_t) = \sum_{p=k}^{d} \sigma_{pk}(u) S_t^p \partial_{x_p} f(S_t)$$

Besides, we assumed that $\rho(u) = \sigma^{-1}(u)$ which completes the proof.

■

Proof of Lemma 2.2:

Using duality (2.7) we have

$$E\left(h(S_s^k)F\sum_{i=k}^d\int_I\rho_{ik}(u)dW_u^i\right)=E\left(\sum_{i=k}^d\int_I D_u^i\left[h(S_s^k)F\right]\rho_{ik}(u)du\right)$$

$$=E\left(h(S_s^k)\sum_{i=k}^d\int_I D_u^iF\rho_{ik}(u)du\right)+E\left(F\sum_{i=k}^d\int_I h'(S_s^k)\sigma_{ki}(u)\rho_{ik}(u)S_s^kdu\right)$$

Moreover, the fact that $\sigma(u)$ and $\rho(u)$ are two triangular matrices such that $\rho_{kk}(u)=1/\sigma_{kk}(u)$ simplifies the last term which can be also rewritten using the Malliavin derivative

$$E\left(F\int_I h'(S_s^k)S_s^kdu\right)=E\left(F\int_I\frac{D_u^kh(S_s^k)}{\sigma_{kk}(u)}du\right)$$

This provides the required result.

■

Proof of Theorem 2.4:

Let us begin with $T_{s,t}[1](x)$, by independence of the coordinates we obtain

$$T_{s,t}[1](x)=E\left(\prod_{k=1}^d\frac{H_k(S_s^k)W_{s,t}^k}{\sigma_k s(t-s)S_s^k}\right)=\prod_{k=1}^d\frac{1}{\sigma_k s(t-s)}E\left(\frac{H_k(S_s^k)W_{s,t}^k}{S_s^k}\right).$$

Afterwards, we use the independence of the increments to obtain

$$\begin{aligned}E\left(\tfrac{H_k(S_s^k)}{S_s^k}W_{s,t}^k\right)&=E\left(\tfrac{H_k(S_s^k)}{S_s^k}[(t-s)(W_s^k+\sigma_k s)-s(W_t^k-W_s^k)]\right)\\&=(t-s)E\left(\tfrac{H_k(S_s^k)}{S_s^k}(W_s^k+\sigma_k s)\right)-sE\left(\tfrac{H_k(S_s^k)}{S_s^k}\right)E(W_t^k-W_s^k)\\&=(t-s)E\left(\tfrac{H_k(S_s^k)}{S_s^k}(\sqrt{s}G+\sigma_k s)\right),\end{aligned}$$

where the random variable G has a standard normal distribution. Moreover we have the following equality in distribution

$$S_s^k\doteq S_0^k\exp\left(\left(r_k-\frac{\sigma_k^2}{2}\right)s+\sigma_k\sqrt{s}G\right).$$

By computing the expectation, we obtain the requested result.

Regarding function h, we condition according to $W_t^k=w_k$ and we use the independence of

coordinates

$$E\left(g(S_t)\prod_{k=1}^{d}\frac{H_k(S_s^k)W_{s,t}^k}{\sigma_k s(t-s)S_s^k}\right)=E\left(g(S_t)\prod_{k=1}^{d}\frac{1}{\sigma_k s(t-s)}h_k(x_k,W_t^k)\right),$$

with

$$h_k(x_k,w_k)=E\left(H_k(S_s^k)W_{s,t}^k/S_s^k\Big|W_t^k=w_k\right).$$

Knowing $W_0^k=0$ and $W_t^k=w_k$, when we fix s the random variable $W_s^k \doteq \frac{sw_k}{t}+\sqrt{\frac{s(t-s)}{t}}G$ and G has a standard normal distribution. Also, we have the following equality in distribution for $W_{s,t}^k$:
$W_{s,t}^k \doteq \sigma_k s(t-s)+\sqrt{ts(t-s)}G$ and $S_s^k \doteq S_0^k \exp\left(\left(r_k-\frac{\sigma_k^2}{2}\right)s+\sigma_k\frac{sw_k}{t}+\sigma_k\sqrt{\frac{s(t-s)}{t}}G\right)$.
Then we compute $h_k(x_k,w_k)$ which yields:

$$h_k(x_k,w_k)=\frac{e^{(\sigma_k^2-r_k)s}}{S_0^k}\sqrt{\frac{ts(t-s)}{2\pi}}\exp\left(\frac{-s\sigma_k}{t}\left(\frac{s\sigma_k}{2}+w_k\right)-\frac{(d_{x_k}(w_k))^2}{2}\right).$$

■

Proof of Theorem 2.5:
For this model $\Gamma_{s,t}=\pi_{s,t}^{1,2}\pi_{s,t}^{2,2}-C_{1,2}$ with

$$\pi_{s,t}^{1,2}=1+\frac{\sqrt{1-\rho^2}W_s^1-\rho W_s^2}{s\sigma_1\sqrt{1-\rho^2}}-\frac{\sqrt{1-\rho^2}(W_t^1-W_s^1)-\rho(W_t^2-W_s^2)}{(t-s)\sigma_1\sqrt{1-\rho^2}},$$

$$\pi_{s,t}^{2,2}=1+\frac{W_s^2}{s\sigma_2\sqrt{1-\rho^2}}-\frac{(W_t^2-W_s^2)}{(t-s)\sigma_2\sqrt{1-\rho^2}},\quad C_{1,2}=\frac{-t\rho}{s(t-s)(1-\rho^2)\sigma_1\sigma_2}.$$

For $k=1,2$, knowing $W_0^k=0$ and $W_t^k=w_k$, when we fix s the random variable $W_s^k \doteq \frac{sw_k}{t}-\sqrt{\frac{s(t-s)}{t}}G_k$ where G_1 and G_2 are two independent with standard normal distribution. In addition, we obtain the following equalities in distribution

$$\frac{\prod_{i=1}^{2}S_s^i}{S_0^1S_0^2}\doteq e^{\left(\frac{(-\sigma_1^2-\sigma_2^2)}{2}+r_1+r_2\right)s+\frac{(\sigma_1+\rho\sigma_2)sw_1+\sigma_2\sqrt{1-\rho^2}sw_2}{t}-\sqrt{\frac{s(t-s)}{t}}\left((\sigma_1+\rho\sigma_2)G_1+\sigma_2\sqrt{1-\rho^2}G_2\right)}$$

$$\Gamma_{s,t}\doteq\left(1+\sqrt{\frac{t}{s(t-s)}}\left[\frac{\rho G_2}{\sigma_2\sqrt{1-\rho^2}}-\frac{G_1}{\sigma_1}\right]\right)\left(1-\sqrt{\frac{t}{s(t-s)}}\frac{G_2}{\sigma_2\sqrt{1-\rho^2}}\right)-C_{1,2},$$

which allows us to compute $h(x, w_1, w_2)$ and obtain the result of Theorem 2.5.

■

Proof of Theorem 2.6:

We perform the regression presented in pages 69-73 that provides the following equalities in distribution when we condition according to $\widetilde{\mathcal{F}}_t$ and $\int_0^t \sqrt{\nu_u} dW_u = w$

$$\frac{1}{s\sqrt{1-\rho^2}} \int_0^s \frac{dW_u}{\sqrt{\nu_u}} - \frac{1}{(t-s)\sqrt{1-\rho^2}} \int_s^t \frac{dW_u}{\sqrt{\nu_u}} \doteq \frac{-1}{s(t-s)\sqrt{1-\rho^2}} \sqrt{(t-s)^2 \int_0^s \frac{du}{\nu_u} + s^2 \int_s^t \frac{du}{\nu_u}} G_1,$$

$$\int_0^s \sqrt{\nu_u} dW_u \doteq \frac{\int_0^s \nu_u du}{\int_0^t \nu_u du} w - \frac{\sqrt{\left(\int_0^t \nu_u du\right)\left(\int_0^s \nu_u du\right) - \left(\int_0^s \nu_u du\right)^2}}{\sqrt{\int_0^t \nu_u du}} G_2,$$

where the Gaussian vector $(G_1, G_2) \sim \mathcal{N}(0, \Gamma)$ with

$$\Gamma = \begin{pmatrix} 1 & R \\ R & 1 \end{pmatrix}, \quad R = \frac{s(t-s)\sqrt{\int_0^t \nu_u du}}{\sqrt{\left(\int_0^t \nu_u du\right)\left(\int_0^s \nu_u du\right) - \left(\int_0^s \nu_u du\right)^2} \sqrt{(t-s)^2 \int_0^s \frac{du}{\nu_u} + s^2 \int_s^t \frac{du}{\nu_u}}}.$$

Thus $T_{s,t}[g](x) = E\left(E\left(g(S_t) h\left(x, \int_0^t \sqrt{\nu_u} dW_u\right) \middle| \widetilde{\mathcal{F}}_t\right)\right)$ with

$$h(x,w) = E\left(\frac{H(S_s)}{S_s}\left(1 - \frac{1}{s(t-s)\sqrt{1-\rho^2}} \sqrt{(t-s)^2 \int_0^s \frac{du}{\nu_u} + s^2 \int_s^t \frac{du}{\nu_u}} G_1\right) \middle| \widetilde{\mathcal{F}}_t \bigvee \int_0^t \sqrt{\nu_u} dW_u = w\right)$$

and after Gaussian computations we get the requested result.

■

Acknowledgment: This work was supported by CreditNext and BIG'MC projects. The authors want to thank Professor Damien Lamberton for his review of our work and Professor Vlad Bally for his valuable advice.

Chapitre 3

High-Dimensional American Pricing Algorithm on GPU Cluster

As it is widely known, the Feynman-Kac formula establishes a link between a linear parabolic PDE (Partial Differential Equation) and a stochastic problem that can be solved by linear **M**onte **C**arlo (**MC**) algorithms. The advantages of solving linear parabolic PDEs by linear MC, for pricing European contracts, were studied in Chapitre 1, especially the suitability of this method for massively parallel architectures based on a GPU cluster.

Some recent results, as the one provided in [14], establish a similar link between nonlinear parabolic PDEs and stochastic problems using BSDEs (Backward Stochastic Differential Equations) which can be solved by MC algorithms. We will call nonlinear MC algorithms the MC algorithms used to solve nonlinear parabolic PDEs. Unlike linear MC, the effectiveness of nonlinear MC algorithms is not obvious. A challenging nonlinear problem in mathematical finance is pricing American contracts. Like [1, 7, 10, 22], we use an MC method based on the **M**alliavin calculus (**MCM**) for pricing American contracts. In Chapitre 2, we proved that the formulation of pricing American options using the Malliavin calculus is very suited to parallel architectures. In this chapter, the latter fact will be explored more deeply: We will detail how to tune the algorithm parameters that act on the efficiency of the algorithm implemented on a GPU (Graphics Processing Unit) cluster. These parameters do not exist in a linear MC but come from the fact that we are dealing with a nonlinear problem. We also introduce a natural way of reducing the dimension [1] for problems that involve, at least, from five to ten assets. For this dimension reduction, we give a straightforward method that allows to check whether the accuracy of the result is sufficient. Finally, we study the execution time and the energy consumption of

1. As it will be explained later, we basically reduce the number of Brownian motions that drives the SDE, but only for the computation of the continuation and not for the payoff.

our MCM algorithm on a CPU/GPU cluster.

In section 3.1, we remind briefly the theoretical and algorithmic formulation of the problem of pricing American options using the Malliavin calculus. Afterwards, we detail in section 3.2 the dimension reduction method for the computation of the conditional expectation, then give the explicit formula obtained once this approximation is used. From a practitioner's point of view, the latter formula is important because it can be directly implemented regardless of all the theory and simplifications that justify it. In section 3.3, we present our multiparadigm procedure implemented on top of Cuda+MPI. In section 3.4, after a discussion on the accuracy of the results, we study the different parameters that help us to tune the heterogeneous CPU/GPU implementation. Finally, we compare, in section 3.5, the mono-node multi-threaded CPU and GPU implementations according to the execution time and the energy consumed. Also, on an increasing number of nodes, we test the scalability of our algorithm on a GPU cluster (16 GPUs NVIDIA Fermi architecture) for different trajectory size problems.

3.1 Theoretical and algorithmic presentation

3.1.1 State of art & brief theoretical considerations

In [50], the authors implemented an embarrassingly parallel algorithm for pricing American options thanks to a quantification that can be performed off-line. Although this method provides very accurate results for dimensions $d \leq 3$, it is limited for higher dimensions by the definition of a quantification grid. Other methods that give also very good results are the one presented in [33] & [51] and whose parallelization is studied in [19]. In addition to the dimension limitations, as described in [19], these two methods also have scalability limitations on a CPU cluster, even if some parts of the algorithms can be done off-line. The authors of [18] provide some interesting experiences in a grid computing environment that summarizes very well the kind of programming problems that one can face when dealing with American option pricing.

In Chapitre 1, we proposed the parallelization of the widely used **L**ongstaff-**S**chwartz (**LS**) algorithm. The advantage of this algorithm is its efficiency in low and medium dimensional problems $d < 5$, which means that we can simulate small number of trajectories and have a sufficiently good prices sometimes even for five dimensional problems. However, the drawback of LS is the bias introduced by the parametric regression and that cannot be removed by increasing the number of the simulated trajectories (see Figure 3.1), but it requires increasing the cardinal of the regression basis which can suppress completely the original efficiency of LS (see [26] for more details). Consequently, even for dimensions less than 5, the errors of regression methods

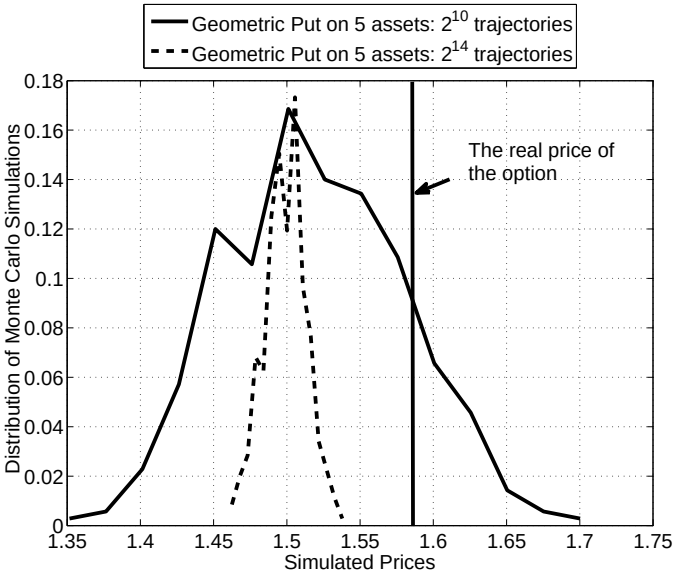

FIGURE 3.1 – Histogram of prices obtained by Monte Carlo using Longstaff-Schwartz algorithm with maturity $T = 1$ and thirty exercise dates.

lead to very important errors on the Greeks computation. Moreover, pricing American options with Monte Carlo using regression methods (see [2] or Chapitre 1) is intrinsically difficult to parallelize, because the regression part cannot be parallelized effectively on the GPU and it should be done on the CPU.

In this chapter, we study the parallel implementation and suitability of MCM presented in Chapitre 2. In contrast to regression and quantification algorithms, this approach aims at a non-parametric algorithm in which the accuracy can be improved by increasing only the number of the simulated trajectories. This fact is demonstrated on Figure 3.2 in which we can see that MCM performs better than LS when we increase the number of simulated trajectories. Moreover, according to Figure 3.2, MCM is as efficient as LS because it provides sufficiently good results even when simulating only 2^{10} trajectories which requires less than one second for execution.

Nevertheless, as we will see in section 3.2, for high-dimensional problems (dimension > 5) the expression of the conditional expectation using the Malliavin calculus becomes complex enough that one need to reduce the dimension using a PCA-like (Principal components analysis) method. Before introducing the simplifications in section 3.2, we want here to detail the part of the algorithm in which those simplifications are done.

As explained previously, for a multidimensional Markovian asset model S, the price of an American contract can be approximated for large number of time steps by the price of a

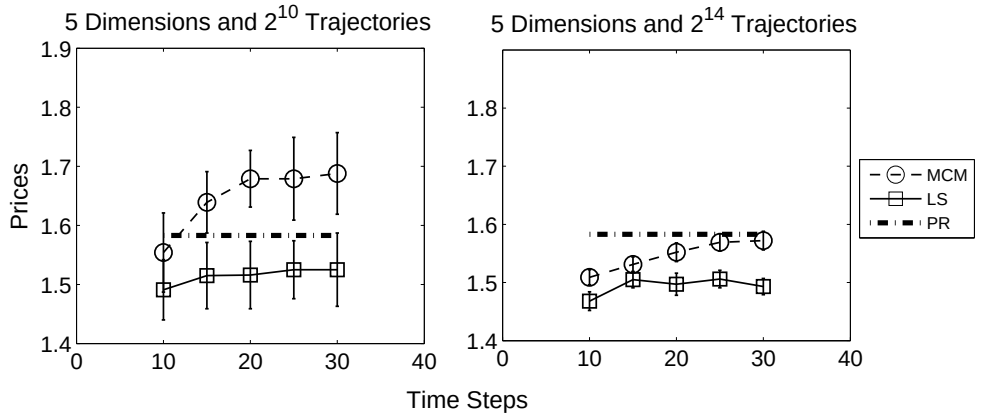

FIGURE 3.2 – MCM vs. LS for a 5 dimensional American option, the real price is given by PR.

Bermudan contract. The price of a Bermudan contract can be obtained thanks to the following induction that allows to get the optimal stopping time

$$\begin{aligned} \tau_N &= T, \\ \forall k \in \{N-1, ..., 0\}, \quad \tau_k &= t_k 1_{A_k} + \tau_{k+1} 1_{A_k^c}, \end{aligned} \tag{3.1}$$

where the set $A_k = \{\Phi(S_{t_k}) > C(S_{t_k})\}$, $\Phi(S_{t_k})$ is the payoff of a given contract on the time interval $t_k \in [t, T]$ and $C(S_{t_k})$ represents the continuation value and is given by

$$C(S_{t_k}) = E\left(e^{-r\delta t} P_{t_{k+1}}(S_{t_{k+1}}) \middle| S_{t_k}\right). \tag{3.2}$$

This formulation of the dynamic programming allows us to reduce the bias (we refer the reader to [25]). It also makes effective the dimension simplifications that we will perform in section 3.2: Reducing the number of Brownian motions that drive the SDE will be done only for the approximation of the continuation (3.2) which is involved in the determination of the optimal stopping time.

3.1.2 Algorithmic details

As it was done for LS in Chapitre 1, we subdivide the entire algorithm of MCM in three phases:

1) Paths Generation (PG) phase.
2) Approximation of the Continuation (AC) phase.
3) Pricing (PRC) phase.

Algorithm 4 is the global algorithm that includes the three phases of the simulation and Algorithm 5 is the one that details the operations performed in the AC phase (that replaces the REG phase of LS). In Algorithm 4 and Algorithm 5, we use the parameter l ($l = l_1, l_2$) as a path index and i as a dimension index, we also denote n the number of simulated paths, d the total dimension and δt the time discretization.

Algorithm 4: MCM algorithm for an American option

Input: Model parameters and random number generator initialization.
Output: $P_0(S_0)$
for $t \in \{T, \dots, 2\delta t, \delta t\}$ **do**
  `/* Computations performed during the PG phase */`
  for $i \in \{1, \dots d\}$ **do**
    for $l \in \{1, \dots n\}$ **do**
      – Draw $W_t^{i,l}$ using RNG and the Brownian bridge induction
      – Use (3.3) to update the asset price $S_t^{i,l}$
    end
  end
  if $(t < T)$ *and* $l \in \{\Phi(S_t^l) > 0\}$ **then**
    `/* Computations performed during the AC phase */`
    This part is detailed in Algorithm 5
    `/* Computations performed during the PRC phase */`
    for $l \in \{1, \dots n\}$ **do**
      – Compute the payoff $\Phi(S_t^l)$
      – $P_t(S_t^l) = 1_{\Lambda_l}\Phi(S_t^l) + 1_{\Lambda_l^c} e^{-r\delta t} P_{t+\delta t}(S_{t+\delta t}^l)$
    end
    if $(t = \delta t)$ **then**
      `/* P_0(S_0) is the price of the option */`
      $P_0(S_0) = \max\left(\Phi(S_0), \frac{e^{-r\delta t}}{n}\sum_{l=1}^{l=n} P_{\delta t}(S_{\delta t}^l)\right)$
    end
  end
  else
    `/* Computations performed during the PRC phase */`
    for $l \in \{1, \dots n\}$ **do**
      $P_t(S_t^l) = 1_{t=T}\Phi(S_T^l) + 1_{\Phi(S_t^l)\leq 0} e^{-r\delta t} P_{t+\delta t}(S_{t+\delta t}^l)$
    end
    `/* We have, of course, ∀l ∈ {1,...n} P_{T+δt}(S^l_{T+δt}) = 0 */`
  end
end

Algorithm 5: The AC phase

Input: $P_{t+\delta t}(S^l_{t+\delta t})$ and $\{W^i_{t+\delta t}\}^l_{1\leq i\leq d}$ with $l \in \{1, \dots n\}$
Output: $C(S^l_t)$ with $l \in \{1, \dots n\}$
for $l_1 \in \{1, \dots n\}$ **do**

AC1) For $l_2 \in \{1, \dots n\}$, evaluate

$$h\left(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right)$$

AC2) For either $(n', n'') = (n/2, n)$ or $(n'', n') = (n, n/2)$, we refer to Chapitre 2 for more details

$$C(S^{l_1}_t) \approx \frac{\frac{1}{n'}\sum_{l_2=1}^{n'} e^{-r\delta t} P_{t+\delta t}(S^{l_2}_{t+\delta t}) h\left(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right)}{\frac{1}{n''}\sum_{l_2=1}^{n''} h\left(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right)}$$

end

Path Generation phase (PG)

In this part of the algorithm, we simulate the paths of the multidimensional asset $S_t = (S^1_t, ..., S^d_t)$ by simulating the multidimensional Brownian motion $W_t = (W^1_t, ..., W^d_t)$ and the dynamic of S_t is given by

$$S^i_t = S^i_0 \exp\left[\left(r_i - \frac{1}{2}\sum_{k=1}^{i}\sigma^2_{ik}\right)t + \sum_{k=1}^{i}\sigma_{ik}W^k_t\right], \tag{3.3}$$

where:

S^i_0 is the initial price of the asset i,
r_i is the rate of the asset i,
σ_{ik} is an element of the diffusion matrix associated to the asset i and to the Brownian motion k.

Although, the theoretical results in section 3.2 are established for a very general model, the basic model given in (3.3) is sufficient to reflect the computational difficulties. Moreover, the Brownian bridge technique simulation (see [25]) of W^k_t does not reduce the generality of our algorithm because the GPU memory storage of the Brownian motion paths is negligible when compared to the memory storage needed for the AC phase. Moreover, the MPI implementation implies a communication between GPUs thanks to the CPU, subsequently the Brownian motion paths can be stored in the CPU memory until they are requested by the GPU. We simulate the Brownian motion paths thanks to the parallel CMRG Random Number Generator (RNG) [40, 41], the parallelization of this RNG on a GPU cluster is detailed in Chapitre 1. By setting an RNG for each trajectory, this phase can be executed independently on each streaming process.

Approximation of the Continuation phase(AC)

Using Malliavin calculus [1, 7, 10], for each path l_1, the continuation $C(S_t^{l_1})$ can be approximated in our model (3.3) by

$$C(S_t^{l_1}) \approx \frac{\frac{1}{n'}\sum_{l_2=1}^{n'} e^{-r\delta t} P_{t+\delta t}(S_{t+\delta t}^{l_2}) h\left(S_t^{l_1}, \{W_{t+\delta t}^i\}_{1\leq i\leq d}^{l_2}\right)}{\frac{1}{n''}\sum_{l_2=1}^{n''} h\left(S_t^{l_1}, \{W_{t+\delta t}^i\}_{1\leq i\leq d}^{l_2}\right)} \tag{3.4}$$

where:

$$h\left(x, \{y_i\}_{1\leq i\leq d}\right) = E\left(1_{S_t\geq x}\Theta_{t,t+\delta t}\middle|\{W_{t+\delta t}^i\}_{1\leq i\leq d} = \{y_i\}_{1\leq i\leq d}\right)$$

where $1_{S_t\geq x}$ is equal to the tensorial product $\prod_{i=1}^{d} 1_{S_t^i\geq x_i}$, when $d \geq 1$. Moreover, the value of $\Theta_{t,t+\delta t}$ was already given in Chapitre 2 as well as the relation between n' and n'' (either $(n', n'') = (n/2, n)$ or $(n'', n') = (n, n/2)$). After reminding in Theorem 3.1 the principal result of Chapitre 2, we give in (3.12) a detailed expression of the function h.

Due to the fact that we use a backward algorithm, we consider t_k as the actual time step and $t_{k+1} = t_k + \delta t$ as the previous time step. For each path l_1, to compute $C(S_{t_k}^{l_1})$ at the actual time step t_k, we need the whole path data of $P_{t_{k+1}}(S_{t_{k+1}}^{l_2})$ and $\{W_{t_{k+1}}^i\}_{1\leq i\leq d}^{l_2}$ at the previous time step t_{k+1}. Nevertheless, due to a shift of the time step, for each path l_2, $P_{t_{k+1}}(S_{t_{k+1}}^{l_2})$ is computed in the PRC phase and can be transferred from each GPU to the other GPUs during the PG phase. For the same reason, $\{W_{t_{k+1}}^i\}_{1\leq i\leq d}^{l_2}$ is computed in the PG phase and can be transferred during the AC and the PRC phase. Thanks to Algorithm 5, one can easily see that the complexity of the AC phase is equal to $O(n^2)$ and it is divided into:

AC1) the evaluation of the function $h\left(x, \{y_i\}_{1\leq i\leq d}\right)$,

AC2) the computation of the sum of the Monte Carlo approximation.

Pricing phase (PRC)

Once we compute the continuation $C(S_t^l)$ in the AC phase, this phase performs the backward induction (3.1) independently on each path of the simulation. The set $\Lambda_l = \{C(S_t^l) < \Phi(S_t^l)\}$ allows to test the continuation for each trajectory l using the indicator application 1.

3.2 The expression of function h after a dimension reduction

Like in section 2.3, we work here with a model that is more general than the one implemented in our tests and that satisfies the SDE (3.3). Thus, we suppose the assets $S_t^1, ..., S_t^d$ are

solution of the following stochastic differential equations

$$\frac{dS_t^i}{S_t^i} = r_i dt + \sum_{j=1}^{i} \sigma_{ij}(t) dW_t^j, \quad S_0^i = z_i, \quad i = 1, .., d, \tag{3.5}$$

where r_i are constants and $\sigma(t) = \{\sigma_{ij}(t)\}_{1 \leq i,j \leq d}$ is a deterministic triangular matrix ($\{\sigma_{ij}(t)\}_{i<j} = 0$). Also, we suppose that the matrix $\sigma(t)$ is invertible, bounded and uniformly elliptic.

3.2.1 Dimension reduction

Let us remind the principal results of section 2.3. Applying Theorem 2.1 and 2.2, given in Chapitre 2, to the model (3.5), we get the following theorem

Theorem 3.1 *We take $s \in]0, t[$, g an $\mathbb{R}^d$-measurable function with polynomial growth and $x = (x_1, ..., x_d)$ with $x_i > 0$. We denote $H_i(y_i) = H(y_i - x_i)$ the Heaviside function of the difference between y_i and x_i, then*

$$E\left(g(S_t)\Big|S_s = x\right) = \frac{T_{s,t}[g](x)}{T_{s,t}[1](x)}, \tag{3.6}$$

where $T_{s,t}[f](x)$ is defined for every function[2] *$f \in \mathcal{E}_b(\mathbb{R}^d)$ by*

$$T_{s,t}[f](x) = E\left(f(S_t)\Gamma_{s,t}\prod_{k=1}^{d}\frac{H_k(S_s^k)}{S_s^k}\right), \tag{3.7}$$

$\Gamma_{s,t}$ is given by

$$\Gamma_{s,t} = \sum_{p \in \overline{\mathcal{P}}_{1,d}} \epsilon(p) \prod_{i=1}^{d} A_{i,p(i)}, \tag{3.8}$$

with $\epsilon(p)$ as the signature of the permutation $p \in \overline{\mathcal{P}}_{1,d}$, $\overline{\mathcal{P}}_{1,d}$ is the set of the second order permutations i.e. $p \circ p = Id$ where Id is the identity application and

$$A = \begin{pmatrix} \pi_{s,t}^{1,d} & C_{1,2} & C_{1,3} & \cdots & C_{1,d} \\ 1 & \pi_{s,t}^{2,d} & C_{2,3} & \cdots & C_{2,d} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & \cdots & 1 & \pi_{s,t}^{d-1,d} & C_{d-1,d} \\ 1 & 1 & \cdots & 1 & \pi_{s,t}^{d,d} \end{pmatrix}, \tag{3.9}$$

2. In our case $f = g$ or $f = 1$.

where $C_{k,l}$ is the covariance of $\pi_{s,t}^{k,d}$ and $\pi_{s,t}^{l,d}$ with

$$\pi_{s,t}^{k,d} = 1 + \sum_{j=k}^{d} \int_0^t \varphi_{jk}(u) dW_u^j, \quad \varphi_{jk}(u) = \frac{1}{s}\rho_{jk}(u)1_{u\in]0,s[} - \frac{1}{t-s}\rho_{jk}(u)1_{u\in]s,t[},$$

ρ is the inverse matrix $\rho(u) = \sigma^{-1}(u)$.

As we have already seen, the expression (3.8) represents a quasi-determinant that can be computed in practice thanks to the development according to the first line as in Figure 2.1 of Chapitre 2 (In this figure, the computations are done for a general 3×3 matrix). In fact, the use of only the second order permutations $\overline{\mathcal{P}}_{1,d}$ implies a symmetry in the expression of $\Gamma_{s,t}$, thus when suppressing, for example, column 2 in the development we must also suppress line 2. Thanks to this symmetry, the number of the terms involved in the expression of $\Gamma_{s,t}$ is less than the number of terms involved in the determinant expression ($d!$). Unfortunately, according to the following proposition, the number of terms remains big for high-dimensional contracts $d > 5$. Before announcing this theorem, we provide a definition that explains all the needed notations.

Definition 3.1 *For a $d \times d$ matrix A of the form (3.9), we call n_d the number of non zero terms to sum in the computation of the quasi-determinant of A defined in (3.8). For $1 \leq i \leq d$, we call n_d^i the number of non zero terms to sum in the computation of the quasi-determinant of the i^{th} order simplification of the matrix A obtained when we put $\{C_{d-k,j}\}_{1\leq k\leq i, d-k+1\leq j\leq d} = 0$.*

For instance, when $d = 3$, a 1^{st} order simplification of a 3×3 matrix of the form (3.9) is given by

$$\begin{pmatrix} \pi_{s,t}^{1,d} & C_{1,2} & C_{1,3} \\ 1 & \pi_{s,t}^{2,d} & 0 \\ 1 & 1 & \pi_{s,t}^{3,3} \end{pmatrix}, \ with \ n_3^1 = 3,$$

when, as illustrated in Figure 2.1, $n_3 = 4$.

Proposition 3.1 *For $d \geq 3$ and $i \in \{1, ..., d-1\}$, n_d and n_d^i, introduced in Definition 3.1, satisfy the following inductions*

$$n_d = n_{d-1} + (d-1)n_{d-2} \tag{3.10}$$

and

$$n_d^i = n_{d-1}^{i-1} + (d-i-1)n_{d-2}^{i-1} \tag{3.11}$$

with: $n_1 = 1$, $n_2 = 2$ *and* $n_d^0 = n_d$.

Proposition 3.1 can be easily proved by induction using the relation

$$\overline{\mathcal{P}}_{1,d} = \{\tau_{d-1}^{d-1} \circ p;\ p \in \overline{\mathcal{P}}_{1,d-1}\} \cup \{\tau_{d-1}^{l} \circ p;\ p \in \overline{\mathcal{P}}_{1,d-1},\ p(l) = l,\ l \in \{1, ..., d-1\}\},$$

where $\tau_i^j : i \leftrightarrow j$ is the transposition application on $\{1, ..., d-1\}$ that swaps only i to j and j to i. This relation is another version of the relation (2.16) given in Chapitre 2. Although the induction (3.10) provides an n_d which is by far smaller than $d!$, for high dimensions n_d can be quite big. However, taking the $(d-2)^{th}$ and the $(d-3)^{th}$ order simplification, we reduce drastically the number of terms. Table 3.1 provides examples of values taken by n_d^i.

TABLE 3.1 – Values of n_d^i for some dimensions d and orders of simplification i

Dimension d	$i = 0$	$i = d-3$	$i = d-2$
5	26	14	5
6	76	22	6
7	232	32	7
8	764	44	8
9	2620	58	9
10	9496	74	10

In the special case of the Black & Scholes model (3.3), we point out that one can perform a change-of-variables method as explained in [7] and no approximation is needed. The proposed approximation is required for the general model (3.5) when σ is not constant.

Now, we provide two successive approximation strategies for the expression of the function h defined by

$$h(x, w_{ij}) = E\left(\Gamma_{s,t} \prod_{k=1}^{d} \frac{H_k(S_s^k)}{S_s^k} \Big| \left\{\int_0^t \sigma_{ij}(u) dW_u^j\right\}_{1 \le j \le i \le d} = \{w_{ij}\}_{1 \le j \le i \le d}\right), \quad s < t. \tag{3.12}$$

Approximation I

The first approximation consists in replacing the matrix given in (3.9) by a matrix that is sparse above its diagonal. A way of doing it is by replacing the diffusion matrix $\sigma(u)$ by $\widetilde{\sigma}(u)$ expressed in (3.13). Here, $\sigma(u)$ is replaced by $\widetilde{\sigma}(u)$ only [3] for the computation of h and this is

3. and not for the evaluation of the payoff.

done by placing in the first lines the assets that intervene the most in the value of the option [4]. Then we hold only the diagonal and the two Brownian motions (W^1, W^2) that are associated to the two biggest eigenvalues of the correlation matrix. Afterwards, using the parameter q, we suppress some dependencies with respect to W^2 of the assets that intervene the least in the value of the option [5]. To summarize, $\widetilde{\sigma}(u)$ comes from $\sigma(u)$ by holding the diagonal, the first column and $q-1$ terms ($q \geq 2$) of the second column.

$$\widetilde{\sigma}(u) = \begin{pmatrix} \sigma_{1,1}(u) & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \sigma_{2,1}(u) & \sigma_{2,2}(u) & 0 & & & & 0 & 0 \\ \sigma_{3,1}(u) & \sigma_{3,2}(u) & \sigma_{3,3}(u) & 0 & & & & 0 \\ \vdots & \vdots & 0 & \ddots & \ddots & & & \vdots \\ \vdots & \sigma_{q,2}(u) & 0 & 0 & \ddots & \ddots & & \vdots \\ \vdots & 0 & \vdots & & \ddots & \ddots & 0 & 0 \\ \sigma_{d-1,1}(u) & \vdots & 0 & & & 0 & \sigma_{d-1,d-1}(u) & 0 \\ \sigma_{d,1}(u) & 0 & 0 & \cdots & \cdots & 0 & 0 & \sigma_{d,d}(u) \end{pmatrix}. \quad (3.13)$$

With this simplification, the inverse matrix $\widetilde{\rho}(u) = \widetilde{\sigma}^{-1}(u)$ has elements equal to zero in the same places where the elements of $\widetilde{\sigma}$ are null. Replacing σ by $\widetilde{\sigma}$ also changes the matrix A of the expression (3.9) by the matrix A^d

$$A^d = \begin{pmatrix} \pi_{s,t}^{1,d} & C_{1,2} & C_{1,3} & C_{1,4} & \cdots & \cdots & C_{1,d-1} & C_{1,d} \\ 1 & \pi_{s,t}^{2,d} & C_{2,3} & \cdots & C_{2,q} & 0 & \cdots & 0 \\ 1 & 1 & \pi_{s,t}^{3,d} & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & \ddots & 0 & & & 0 \\ \vdots & & & \ddots & \ddots & \ddots & & \vdots \\ 1 & & & & 1 & \ddots & 0 & 0 \\ 1 & 1 & \cdots & \cdots & 1 & 1 & \pi_{s,t}^{d-1,d} & 0 \\ 1 & 1 & \cdots & \cdots & \cdots & 1 & 1 & \pi_{s,t}^{d,d} \end{pmatrix}, \text{ with} \quad (3.14)$$

4. Or the assets that have the biggest volatilities.
5. Or the assets that have the smallest volatilities.

$$\left\{\begin{array}{l} \pi_{s,t}^{1,d} = 1 + \sum_{j=1}^{d} \int_0^t \varphi_{j1}(u) dW_u^j, \quad \varphi_{j1}(u) = \frac{1}{s}\widetilde{\rho}_{j1}(u) 1_{u\in]0,s[} - \frac{1}{t-s}\widetilde{\rho}_{j1}(u) 1_{u\in]s,t[}, \\ \\ \pi_{s,t}^{2,d} = 1 + \sum_{j=2}^{q} \int_0^t \varphi_{j2}(u) dW_u^j, \quad \varphi_{j2}(u) = \frac{1}{s}\widetilde{\rho}_{j2}(u) 1_{u\in]0,s[} - \frac{1}{t-s}\widetilde{\rho}_{j2}(u) 1_{u\in]s,t[}, \\ \\ i \geq 3, \ \ \pi_{s,t}^{i,d} = 1 + \int_0^t \varphi_{ii}(u) dW_u^i, \quad \varphi_{ii}(u) = \frac{1}{s}\widetilde{\rho}_{ii}(u) 1_{u\in]0,s[} - \frac{1}{t-s}\widetilde{\rho}_{ii}(u) 1_{u\in]s,t[}. \end{array}\right. \tag{3.15}$$

where $C_{k,l}$ is the covariance of $\pi_{s,t}^{k,d}$ and $\pi_{s,t}^{l,d}$ whose general expression is given in (3.15) above.

By this approximation, the number of non zero terms to sum in the computation of the quasi-determinant of A^d is included between n_d^{d-3} and n_d^{d-2}, which remains computationally acceptable when $d > 5$ (see Table 3.1). In addition, this approximation is justified numerically by the fact that we can increase the value of q until we are satisfied by the accuracy of the result. This helps us also to quantify, more or less, the error committed by this kind of approximation.

If we use the notation $|\cdot|$ for a quasi-determinant that involves only the second order permutations $p \in \overline{\mathcal{P}}_{1,d}$, the following lemma provides the expression of $|A^d|$. The proof of this lemma is given in the appendix.

Lemma 3.1 *We consider A^d and $\pi_{s,t}^{i,d}$ defined as in (3.14) and (3.15), then*

$$\begin{array}{rcl} |A^d| & = & \sum_{p\in\overline{\mathcal{P}}_{1,d}} \epsilon(p) \prod_{i=1}^{d} A_{i,p(i)}^d \\ & = & \prod_{1\leq i\leq d} \pi_{s,t}^{i,d} - C_{1,2} \prod_{1\leq i\leq d}^{i\neq 1,2} \pi_{s,t}^{i,d} - C_{2,3} \prod_{1\leq i\leq d}^{i\neq 2,3} \pi_{s,t}^{i,d} \\ & + & \sum_{k=4}^{q} (-1)^{k+1} C_{1,k} \left[\prod_{2\leq i\leq d}^{i\neq k} \pi_{s,t}^{i,d} + \sum_{j=3}^{k-1} (-1)^j C_{2,j} \left(\prod_{3\leq i\leq d}^{i\neq j,k} \pi_{s,t}^{i,d} \right) \right] \\ & + & \sum_{k=4}^{q} (-1)^{k+2} C_{2,k} \left[\prod_{1\leq i\leq d}^{i\neq 2,k} \pi_{s,t}^{i,d} + \sum_{j=3}^{k-1} (-1)^j C_{1,j} \left(\prod_{3\leq i\leq d}^{i\neq j,k} \pi_{s,t}^{i,d} \right) \right] \\ & + & \sum_{k=q+1}^{d} (-1)^{k+1} C_{1,k} \left[\prod_{2\leq i\leq d}^{i\neq k} \pi_{s,t}^{i,d} + \sum_{j=3}^{q} (-1)^j C_{2,j} \left(\prod_{3\leq i\leq d}^{i\neq j,k} \pi_{s,t}^{i,d} \right) \right], \end{array} \tag{3.16}$$

with the convention that the last term of the previous equality is equal to zero when $q = d$.

As presented in Chapitre 2 (section 2.5), the proof of Lemma 3.2 is based on a regression argument according to $\int_0^t \sigma_{ij}(u) dW^j$ of the other Gaussian random variables. This Lemma defines also the regression parameters that intervene in the computation of the approximation of the function h.

Lemma 3.2 ***and Definition*** *There exist independent families of correlated Gaussian vectors, $Z_{11} \sim \mathcal{N}(0, V_1)$, $(Z_{12}, Z_{22}) \sim \mathcal{N}(0, V_2)$, $(Z_{13}, Z_{23}, Z_{33}) \sim \mathcal{N}(0, V_3)$, ..., $(Z_{1q}, Z_{2q}, Z_{qq}) \sim \mathcal{N}(0, V_q)$ then $(Z_{1(q+1)}, Z_{(q+1)(q+1)}) \sim \mathcal{N}(0, V_{q+1})$, ..., $(Z_{1d}, Z_{dd}) \sim \mathcal{N}(0, V_d)$ that are orthogonal to $\left\{\int_0^t \widetilde{\sigma}_{ij}(u) dW_u^j\right\}_{i,j\leq d}$ such that the random variables $\left\{\pi_{s,t}^{k,d}\right\}_{1\leq k\leq d}$, defined in (3.15), are*

also equal to

$$\left\{\begin{array}{c} \pi_{s,t}^{1,d} = 1 \quad + \quad \sum_{i=1}^{d} a_i^1 \int_0^t \sigma_{i1}(u)dW_u^1 + Z_{11} + \sum_{i=2}^{q} a_i^{12} \int_0^t \sigma_{i2}(u)dW_u^2 + Z_{12} \\ + \qquad\qquad \sum_{i=3}^{d} a_1^i \int_0^t \sigma_{ii}(u)dW_u^i + \sum_{i=3}^{d} Z_{1i}, \\ \pi_{s,t}^{2,d} = 1 + \sum_{i=2}^{q} a_i^{22} \int_0^t \sigma_{i2}(u)dW_u^2 + Z_{22} + \sum_{i=3}^{q} a_2^i \int_0^t \sigma_{ii}(u)dW_u^i + \sum_{i=3}^{q} Z_{2i}, \\ i \geq 3, \;\; \pi_{s,t}^{i,d} = 1 + a_i^i \int_0^t \sigma_{ii}(u)dW_u^i + Z_{ii}, \end{array}\right. \tag{3.17}$$

with $a^1 = \{a_i^1\}_{1\leq i\leq d} = \Sigma_{1t}^{-1}\Psi_{1t}$, $(a^{12}, a^{22}) = \{(a_i^{12}, a_i^{22})\}_{2\leq i\leq q} = \Sigma_{2t}^{-1}\Psi_{2t}$ *where*

$$\left\{\begin{array}{c} \Sigma_{1t} = \left\{\int_0^t \sigma_{i1}(u)\sigma_{k1}(u)du\right\}_{1\leq i,k\leq d}, \quad \Sigma_{2t} = \left\{\int_0^t \sigma_{i2}(u)\sigma_{k2}(u)du\right\}_{2\leq i,k\leq q}, \\ \Psi_{1t} = \left\{\int_0^t \varphi_{11}(u)\sigma_{k1}(u)du\right\}_{1\leq k\leq d}, \\ \Psi_{2t} = (\Psi_{2t}^1, \Psi_{2t}^2) = \left\{\left(\int_0^t \varphi_{21}(u)\sigma_{k2}(u)du, \int_0^t \varphi_{22}(u)\sigma_{k2}(u)du\right)\right\}_{2\leq k\leq q} \end{array}\right.$$

and $\left\{a_j^i\right\}_{i\geq 3,\; j=1,2,i} = \left\{\left(\int_0^t \varphi_{ij}(u)\sigma_{ii}(u)du\right) / \left(\int_0^t \sigma_{ii}^2(u)du\right)\right\}_{i\geq 3,\; j=1,2,i}$. *Moreover*

$$\left\{\begin{array}{l} V_1 = \int_0^t \varphi_{11}^2(u)du - {}^t\Psi_{1t}\,{}^t\Sigma_{1t}^{-1}\Psi_{1t}, \;\; V_2 = \left\{\int_0^t \varphi_{2i}(u)\varphi_{2j}(u)du\right\}_{i,j\in\{1,2\}} - {}^t\Psi_{2t}\,{}^t\Sigma_{2t}^{-1}\Psi_{2t}, \\ i \geq 3, \; V_i = \left\{\int_0^t \varphi_{ij}(u)\varphi_{ik}(u)du - \frac{1}{\int_0^t \sigma_{ii}^2(u)du} \int_0^t \varphi_{ij}(u)\sigma_{ii}(u)du \int_0^t \varphi_{ik}(u)\sigma_{ii}(u)du\right\}_{j,k\in\{1,2,i\}}. \end{array}\right.$$

Thanks to what was defined above, there exist three independent families $\{X_{1j}\}_{1\leq j\leq d}$, $\{X_{2j}\}_{2\leq j\leq d}$ *and* $\{X_{jj}\}_{3\leq j\leq d}$ *of correlated and centered Gaussian random variables orthogonal to both* $\left\{\int_0^t \widetilde{\sigma}_{ij}(u)dW_u^j\right\}_{i,j\leq d}$ *and the* Z*'s families, such that*

$$\left\{\begin{array}{c} 1 \leq j \leq d, \; \int_0^s \sigma_{j1}(u)dW_u^1 = \sum_{i=1}^{d} b_{ij}^1 \int_0^t \sigma_{i1}(u)dW_u^1 + c_j^{11}Z_{11} + X_{1j}, \\ 2 \leq j \leq q, \; \int_0^s \sigma_{j2}(u)dW_u^2 = \sum_{i=2}^{q} b_{ij}^2 \int_0^t \sigma_{i2}(u)dW_u^2 + c_j^{12}Z_{12} + c_j^{22}Z_{22} + X_{2j}, \\ 3 \leq j \leq d, \; \int_0^s \sigma_{jj}(u)dW_u^j = b_{jj} \int_0^t \sigma_{jj}(u)dW_u^j + c^{1j}Z_{1j} + c^{2j}Z_{2j} + c^{jj}Z_{jj} + X_{jj}, \end{array}\right. \tag{3.18}$$

where $b^1 = \Sigma_{1t}^{-1}\Sigma_{1s}$, $b^2 = \Sigma_{2t}^{-1}\Sigma_{2s}$ *and* $\{b_{jj}\}_{j\geq 3} = \left\{\left(\int_0^s \sigma_{jj}^2(u)du\right) / \left(\int_0^t \sigma_{jj}^2(u)du\right)\right\}_{j\geq 3}$. *Finally,* $c_j^{11} = V_1^{-1}U_{1j}$, $(c_j^{12}, c_j^{22}) = V_2^{-1}U_{2j}$ *and* $\{(c^{1j}, c^{2j}, c^{jj})\}_{j\geq 3} = \left\{V_j^{-1}U_{jj}\right\}_{j\geq 3}$ *with*

$$\left\{\begin{array}{c} \{U_{1j}\}_{1\leq j\leq d} = \left\{\int_0^s \varphi_{11}(u)\sigma_{j1}(u)du\right\}_{j\geq 1} - \Sigma_{1s}\Sigma_{1t}^{-1}\Psi_{1t}, \\ \{U_{2j}\}_{2\leq j\leq q} = \left\{\left(\int_0^s \varphi_{12}(u)\sigma_{j2}(u)du, \int_0^s \varphi_{22}(u)\sigma_{j2}(u)du\right)\right\}_{q\geq j\geq 2} - \Sigma_{2s}\Sigma_{2t}^{-1}\Psi_{2t}, \\ 3 \leq j, \; \left\{U_{jj}^i\right\}_{i=1,2,\overline{j}} = \left\{\int_0^s \varphi_{ji}(u)\sigma_{jj}(u)du - \frac{1}{\int_0^t \sigma_{jj}^2(u)du} \int_0^t \varphi_{ji}(u)\sigma_{jj}(u)du \int_0^s \sigma_{jj}^2(u)du\right\}_{i=1,2,j}. \end{array}\right.$$

For the previous lemma, we point out that the Z vectors have different sizes because of the structure of the matrix $\widetilde{\sigma}(u)$ given in (3.13). For instance, $(Z_{1(q+1)}, Z_{(q+1)(q+1)})$ has only two coordinates because $\widetilde{\sigma}_{q+1,2}(u) = 0$.

Approximation II

The second approximation strategy consists to set the Gaussian variables X, in Lemma 3.2, equal to zero. This is justified numerically, by the fact that the elements of the covariance matrix of the X's vectors are small enough that they can be neglected. The latter approximation is not needed in the Black & Scholes case (SDE (3.3)), because each Brownian motion $W_s^i = W_t^i - Z_{ii}$ with $Z_{ii} = (W_t^i - W_s^i)$.

3.2.2 The expression of the function h

First, we denote by the matrix L_i the Cholesky decomposition of the matrix V_i ($V_i = L_i L_i'$). Thanks to Lemma 3.2, we know that V_i is a 3×3 matrix when $3 \leq i \leq q$ and we will use the following notation for the elements of L_i

$$3 \leq i \leq q, \;\; L_i = \begin{pmatrix} l_{11}^i & 0 & 0 \\ l_{21}^i & l_{22}^i & 0 \\ l_{i1} & l_{i2} & l_{ii} \end{pmatrix}.$$

When $i > q$, V_i is a 2×2 matrix and we set $l_{21}^i = l_{22}^i = l_{i2} = 0$. Also, we define the vector κ_i as the product of the vector c^i, introduced in Lemma 3.2, and the matrix L_i, then

$$\begin{aligned} \text{if } 3 \leq i \leq q, \;\; \kappa_i = (\kappa_{i1}, \kappa_{i2}, \kappa_{ii}) = (c^{1i}, c^{2i}, c^{ii}) L_i \\ \text{if } i > q, \;\; \kappa_i = (\kappa_{i1}, \kappa_{ii}) = (c^{1i}, c^{ii}) L_i. \end{aligned}$$

Finally we define the variables λ_i, θ_i and μ_i as follows

$$\begin{cases} \lambda_1 & = & -\frac{1}{2}\int_0^s \sigma_{11}^2(u)du + \sum_{j=1}^d b_{j1}^1 \int_0^t \sigma_{j1}(u)dW_u^1 + c_1^{11}Z_{11}, \\ \lambda_2 & = & -\frac{1}{2}\int_0^s \left(\sigma_{21}^2(u)+\sigma_{22}^2(u)\right)du + \sum_{j=1}^d b_{j2}^1 \int_0^t \sigma_{j1}(u)dW_u^1 \\ & + & \sum_{j=2}^q b_{j2}^2 \int_0^t \sigma_{j2}(u)dW_u^2 + c_2^{11}Z_{11} + c_2^{12}Z_{12} + c_2^{22}Z_{22}, \\ \{\lambda_i\}_{3\le i\le q} & = & -\frac{1}{2}\int_0^s \left(\sum_{j=1,2,i}\sigma_{ij}^2(u)\right)du + \sum_{j=1}^d b_{ji}^1 \int_0^t \sigma_{j1}(u)dW_u^1 \\ & + & \sum_{j=2}^q b_{ji}^2 \int_0^t \sigma_{j2}(u)dW_u^2 + b_{ii}\int_0^t \sigma_{ii}(u)dW_u^i + c_i^{11}Z_{11} + c_i^{12}Z_{12} + c_i^{22}Z_{22}, \\ \{\lambda_i\}_{q<i\le d} & = & -\frac{1}{2}\int_0^s \left(\sum_{j=1,2,i}\sigma_{ij}^2(u)\right)du + \sum_{j=1}^d b_{ji}^1 \int_0^t \sigma_{j1}(u)dW_u^1 \\ & + & b_{ii}\int_0^t \sigma_{ii}(u)dW_u^i + c_i^{11}Z_{11}, \end{cases}$$

$$\theta_i = \ln\left(x_i/S_0^i\right) - \lambda_i, \tag{3.19}$$

$$\begin{cases} \mu_1 = 1 + \sum_{i=1}^d a_i^1 \int_0^t \sigma_{i1}(u)dW_u^1 + Z_{11} + \sum_{i=2}^q a_i^{12}\int_0^t \sigma_{i2}(u)dW_u^2 + Z_{12} \\ \qquad + \sum_{i=3}^d a_1^i \int_0^t \sigma_{ii}(u)dW_u^i, \\ \mu_2 = 1 + \sum_{i=2}^q a_i^{22}\int_0^t \sigma_{i2}(u)dW_u^2 + Z_{22} + \sum_{i=3}^q a_2^i \int_0^t \sigma_{ii}(u)dW_u^i, \\ i \ge 3,\ \ \mu_i = 1 + a_i^i \int_0^t \sigma_{ii}(u)dW_u^i, \end{cases}$$

Using the regression parameters defined in Lemma 3.2 and setting the X random variables that intervene in (3.18) to zero (see Approximation II), the following expressions are purely computational and the expressions were established and verified thanks to Mathematica.

If the asset vector $\widetilde{S}$ satisfies the SDE (3.5) in which the elements of σ are replaced by the elements of $\widetilde{\sigma}$, then

$$h\begin{pmatrix} x, \{w_{i1}\}_{1\le i\le d} \\ \{w_{i2}\}_{2\le i\le q}, \{w_{ii}\}_{3\le i\le d} \end{pmatrix} = E\left[\widetilde{h}\begin{pmatrix} x, \{w_{i1}\}_{1\le i\le d}, \{w_{i2}\}_{2\le i\le q} \\ \{w_{ii}\}_{3\le i\le d}, Z_{11}, Z_{12}, Z_{22} \end{pmatrix}\right] \tag{3.20}$$

and

$$\widetilde{h}\begin{pmatrix} x, \{w_{i1}\}_{1\le i\le d}, \{w_{i2}\}_{2\le i\le q} \\ \{w_{ii}\}_{3\le i\le d}, z_{11}, z_{12}, z_{22}\end{pmatrix} = E\left[|A^d|\prod_{k=1}^{d}\frac{H_k(\widetilde{S}_s^k)}{\widetilde{S}_s^k}\,\middle|\,\begin{matrix}\int_0^t \sigma_{i1}(u)du = w_{i1},\\ \int_0^t \sigma_{i2}(u)du = w_{i2},\\ \int_0^t \sigma_{ii}(u)du = w_{ii},\\ (Z_{11}, Z_{12}, Z_{22}) = (z_{11}, z_{12}, z_{22})\end{matrix}\right]$$

$$= E_z^w\left[|A^d|\prod_{k=1}^{d}\frac{H_k(\widetilde{S}_s^k)}{\widetilde{S}_s^k}\right], \quad w = (\{w_{i1}\}_{1\le i\le d}, \{w_{i2}\}_{2\le i\le q}\{w_{ii}\}_{3\le i\le d}),\ \ z = (z_{11}, z_{12}, z_{22})$$

whose expression, thanks to (3.16), is given by

$$\begin{aligned}&\mathcal{E}^{w,z} - C_{1,2}\mathcal{E}_{1,2}^{w,z} - C_{2,3}\mathcal{E}_{2,3}^{w,z} + \sum_{k=q+1}^{d}(-1)^{k+1}C_{1,k}\left[\mathcal{E}_{1,k}^{w,z} + \sum_{j=3}^{q}(-1)^j C_{2,j}\mathcal{E}_{1,2,j,k}^{w,z}\right]\\ &+\sum_{k=4}^{q}(-1)^{k+1}\left\{C_{1,k}\left[\mathcal{E}_{1,k}^{w,z} + \sum_{j=3}^{k-1}(-1)^j C_{2,j}\mathcal{E}_{1,2,j,k}^{w,z}\right] - C_{2,k}\left[\mathcal{E}_{2,k}^{w,z} + \sum_{j=3}^{k-1}(-1)^j C_{1,j}\mathcal{E}_{1,2,j,k}^{w,z}\right]\right\},\end{aligned} \tag{3.21}$$

where

$$\mathcal{E}_{1,2}^{w,z} = C_{st}\prod_{3\le i\le d}\left(\mu_i R^{i0} + l_{i1}R_1^{i1} + l_{i2}R_2^{i1} + l_{ii}R_i^{i1}\right),$$

$$\mathcal{E}_{1,2,j,k}^{w,z} = \frac{\mathcal{E}_{1,2}^{w,z}}{\left(\mu_j R^{j0} + l_{j1}R_1^{j1} + l_{j2}R_2^{j1} + l_{jj}R_j^{j1}\right)\left(\mu_k R^{k0} + l_{k1}R_1^{k1} + l_{k2}R_2^{k1} + l_{kk}R_k^{k1}\right)},$$

$$\mathcal{E}_1^{w,z} = \mathcal{E}_{1,2}^{w,z}\left[\mu_2 + \sum_{p=3}^{q}\frac{l_{21}^p\left(\mu_p R_1^{p1} + \sum_{j=1,2,p} l_{pj}R_{1j}^{p2}\right) + l_{22}^p\left(\mu_p R_2^{p1} + \sum_{j=1,2,p} l_{pj}R_{2j}^{p2}\right)}{\left(\mu_p R^{p0} + \sum_{j=1,2,p} l_{pj}R_j^{p1}\right)}\right],$$

$$\mathcal{E}_2^{w,z} = \mathcal{E}_{1,2}^{w,z}\left[\mu_1 + \sum_{p=3}^{d}\frac{l_{11}^p\left(\mu_p R_1^{p1} + \sum_{j=1,2,p} l_{pj}R_{1j}^{p2}\right)}{\left(\mu_p R^{p0} + \sum_{j=1,2,p} l_{pj}R_j^{p1}\right)}\right],$$

$$\mathcal{E}_{1,k}^{w,z} = \frac{R^{k0}\mathcal{E}_1^{w,z}}{\left(\mu_k R^{k0} + \sum_{j=1,2,k} l_{kj} R_j^{k1}\right)} + \frac{\mathcal{E}_{1,2}^{w,z} \sum_{j=1,2,k} l_{kj} \left(l_{21}^k \left[R_1^{k1} R_j^{k1} - R^{k0} R_{1j}^{k2}\right] + l_{22}^k \left[R_2^{k1} R_j^{k1} - R^{k0} R_{2j}^{k2}\right]\right)}{\left(\mu_k R^{k0} + \sum_{j=1,2,k} l_{kj} R_j^{k1}\right)^2},$$

$$\mathcal{E}_{2,k}^{w,z} = \frac{R^{k0}\mathcal{E}_2^{w,z}}{\left(\mu_k R^{k0} + \sum_{j=1,2,k} l_{kj} R_j^{k1}\right)} + \frac{\mathcal{E}_{1,2}^{w,z} \sum_{j=1,2,k} l_{kj} l_{11}^k \left[R_1^{k1} R_j^{k1} - R^{k0} R_{1j}^{k2}\right]}{\left(\mu_k R^{k0} + \sum_{j=1,2,k} l_{kj} R_j^{k1}\right)^2},$$

$$\mathcal{E}^{w,z} = \mu_1 \mathcal{E}_1 + \mu_2 \mathcal{E}_2 - \mu_1 \mu_2 \mathcal{E}_{1,2} + \sum_{e=3}^{d} \sum_{p=3}^{q} (l_{11}^e l_{21}^p Y_{ep}^{11} + l_{11}^e l_{22}^p Y_{ep}^{12}),$$

with

$$Y_{ep}^{11} = \frac{\delta_{e-p} \mathcal{E}_{1,2} \left(\mu_p R_{11}^{p2} + \sum_{j=1,2,p} l_{pj} R_{11j}^{p3}\right)}{\left(\mu_p R^{p0} + \sum_{j=1,2,p} l_{pj} R_j^{p1}\right)} + \frac{\overline{\delta}_{e-p} \mathcal{E}_{1,2} \prod_{k=e,p} \left(\mu_k R_1^{1k} + \sum_{j=1,2,p} l_{kj} R_{1j}^{2k}\right)}{\prod_{k=e,p} \left(\mu_k R^{k0} + \sum_{j=1,2,k} l_{kj} R_j^{k1}\right)}$$

$$Y_{ep}^{12} = \frac{\delta_{e-p} \mathcal{E}_{1,2} \left(\mu_p R_{12}^{p2} + \sum_{j=1,2,p} l_{pj} R_{12j}^{p3}\right)}{\left(\mu_p R^{p0} + \sum_{j=1,2,p} l_{pj} R_j^{p1}\right)} + \frac{\overline{\delta}_{e-p} \mathcal{E}_{1,2} \left(\mu_e R_1^{e1} + \sum_{j=1,2,p} l_{ej} R_{1j}^{e2}\right)\left(\mu_p R_2^{p1} + \sum_{j=1,2,p} l_{pj} R_{2j}^{p2}\right)}{\prod_{k=e,p} \left(\mu_k R^{k0} + \sum_{j=1,2,k} l_{kj} R_j^{k1}\right)}$$

and

$$C_{st} = \frac{e^{-\sum_{i=1}^{d} \lambda_i}}{\prod_{1 \le i \le d} S_0^i} 1_{\theta_1 < 0} 1_{\theta_2 < 0}, \quad R^{i0} = F_{i1}, \quad R_j^{i1} = -\kappa_j F_{ij} + \overline{F}_{ij},$$

$$R_{jj}^{i2} = \kappa_j^2 F_{ij} + \left(\frac{m_{ij}}{v_{ij}} - 2\kappa_j\right) \overline{F}_{ij}, \quad R_{jk}^{i2} = \kappa_j \kappa_k F_{ij} + \left(\frac{\theta_i \kappa_k}{\kappa_j^2 v_{ij}} - \kappa_k\right) \overline{F}_{ij},$$

$$R_{jjj}^{i3} = -\kappa_j (3 + \kappa_j^2) F_{ij} + \left(\frac{\theta_i^2}{\kappa_j^2 v_{ij}^2} - \frac{(1 + \theta_i)}{v_{ij}} + (3 + \kappa_j^2)\right) \overline{F}_{ij},$$

$$R^{i3}_{jkk} = -\kappa_j(1+\kappa_k^2)F_{ij} + \left(\frac{\theta_i^2\kappa_k^2}{\kappa_j^4 v_{ij}^2} - \frac{\kappa_k^2(1+\theta_i)}{\kappa_j^2 v_{ij}} + (1+\kappa_k^2)\right)\overline{F}_{ij},$$

$$R^{i3}_{jkl} = -\kappa_j\kappa_k\kappa_l F_{ij} + \frac{\kappa_k\kappa_l}{\kappa_j^2}\left(\frac{\theta_i^2}{\kappa_j^2 v_{ij}^2} - \frac{(1+\theta_i)}{v_{ij}} + \kappa_j^2\right)\overline{F}_{ij}.$$

where δ is the Kronecker delta and $\overline{\delta}_{e-p} = (1-\delta_{e-p})$, besides, for $i \leq q$ and $j \in \{1,2,i\}$

$$F_{ij} = \exp\left(\frac{\kappa_{i1}^2+\kappa_{i2}^2+\kappa_{ii}^2}{2}\right) * \begin{cases} \phi_{v_{ij}}(m_{ij}) \;\; if \;\; \kappa_{ij} \leq 0 \\ 1-\phi_{v_{ij}}(m_{ij}) \;\; if \;\; \kappa_{ij} \geq 0 \end{cases}, \quad \phi_v(x) = \int_{-\infty}^{x} \frac{e^{\frac{-u^2}{2v}}}{\sqrt{2\pi v}}du$$

$$\overline{F}_{ij} = \frac{-\kappa_{ij}}{|\kappa_{ij}|}\exp\left(\frac{\kappa_{i1}^2+\kappa_{i2}^2+\kappa_{ii}^2}{2}\right)\frac{\exp\left(\frac{-m_{ij}^2}{2v_{ij}}\right)}{\sqrt{2\pi v_{ij}}}$$

$$v_{ij} = \frac{\kappa_{i1}^2+\kappa_{i2}^2+\kappa_{ii}^2}{\kappa_{ij}^2}, \quad m_{ij} = \frac{\theta_i+\kappa_{i1}^2+\kappa_{i2}^2+\kappa_{ii}^2}{\kappa_{ij}}.$$

If $i > q$, then $j \in \{1,i\}$ and both F_{ij} and $\overline{F}_{ij}$ have the same expression but with $\kappa_{i2} = 0$.

Equality (3.20) tells us that one should perform a Monte Carlo simulation to get the expression of the function h. Using an importance sampling technique (we refer the reader to [25]), denoting $\omega = (\omega_{11}, \omega_{12}, \omega_{22})$ and $\omega \cdot Z = \omega_{11}Z_{11} + \omega_{12}Z_{12} + \omega_{22}Z_{22}$ then

$$h\begin{pmatrix} x, \{w_{i1}\}_{1\leq i\leq d} \\ \{w_{i2}\}_{2\leq i\leq q}, \{w_{ii}\}_{3\leq i\leq d}\end{pmatrix} = E\left[e^{\omega\cdot Z - \frac{|\omega|^2}{2}}\widetilde{h}\begin{pmatrix} x, \{w_{i1}\}_{1\leq i\leq d}, \{w_{i2}\}_{2\leq i\leq q}, \{w_{ii}\}_{3\leq i\leq d} \\ Z_{11}+\omega_{11}, Z_{12}+\omega_{12}, Z_{22}+\omega_{22}\end{pmatrix}\right] \quad (3.22)$$

Although this method is not mandatory to have good results, we will see in section 3.4.1 that one can choose ω that allows to perform very few drawings (< 32) of the random variables Z_{11}, Z_{12} and Z_{22} and obtain accurate simulations.

Finally, we conclude this theoretical parts by some thoughts on the positive aspects of this method. In fact, beyond its parallelization suitability and its easy extension to high dimensional contracts, this method allows to have a good idea on the accuracy of the result by changing the value of q. Moreover, the American pricing by Malliavin calculus can be easily applied to more complex diffusions than the one presented in (3.5). We have already presented the latter aspect in the previous chapter for the multidimensional Heston model, but this method can be also applied to a large class of multidimensional jump diffusion processes that are built by subordination [6]. In addition to all that, by the function h which is known thanks to (3.22) and

6. This can be done simply by conditioning according to the jump times which are independent from the

(3.21), one can price various contracts [7] at the same time with the same function h.

3.3 Multi-paradigm parallel program

Although the PG and the PRC phases can be efficiently parallelized, the AC phase is tricky and one needs to manage the CPU/GPU data transfer before launching this phase. In fact, even thought no communication is needed during computations in the one core implementation, the cluster version requests communications between all GPUs. Moreover, unlike the PG and the PRC phases which are linear according to the number of the simulated paths n, the complexity of the AC phase [8] is equal to $O(n^2)$. Indeed, during the AC phase, for each path $l \in \{1, \dots n\}$, the computation of the continuation $C(S_t^{(l)})$ is linear according to n. For all these reasons, the details of our parallelization will take into account the necessity to optimize as much as possible the AC phase. We will see some considerations of this optimization in this section and the rest of it in the section 3.4.

3.3.1 Development strategy

Clusters of multicore CPUs and manycore GPUs require three different grains of parallelism. Coarse grained parallelism is used to distribute computations across the cluster nodes and to achieve message-passing between nodes. Medium grain is used to run in parallel some CPU tasks on the different CPU cores of the processors. Finally, fine grained parallelism is required to parallelize computations on all hardware threads of each GPU, according to an SIMD-like programming paradigm.

In this version of implementation, we did not take the whole advantage of the use of medium grain parallelism based on the OpenMP multithreading library. However, we have only focused on the use of coarse and fine grained parallelism based on OpenMPI library and CUDA programming language.

We have identified four main difficulties and objectives in multi-paradigm developments. First, the *synchronization* of : Message-passing (on the cluster), data transfers between CPU and GPU (on each node), and GPU computations. A basic algorithm optimization consists in attempting to overlap communications, transfers and computations. Second, the design of data structures leading to fast data accesses in any part of the application, because optimized data

Brownian motions.

7. contracts that hold the Markov property and that differ only by their payoff.

8. This is the price to pay to have a pure Monte Carlo solution, since the continuation here is also computed by Monte Carlo.

access rules are different on CPUs and GPUs and can require different data storage. Third, minimizing all kinds of communications: inter-nodes cluster communications and data transfers between CPUs and GPUs. Fourth, the limited amount of GPU RAM (compared to the CPU RAM): When processing large problems, either several GPU cards are needed or one has to optimize the algorithm to allow a better use of the memory. The latter point is well explained in the section "Parameters tuning" and enabled us to reduce the CPU/GPU communications to their minimum.

3.3.2 Multi-paradigm parallel algorithm

Algorithm 6: Non-overlapping parallel algorithm run on each node of a GPU cluster.

Initialization on the CPU
Initialization on the GPU
Transfer initial values from the CPU to the GPU
Compute Gaussian random numbers on the GPU
Compute Brownian motion arrays at time step N on the GPU
Compute asset prices on the GPU
Compute the payoff on the GPU
Compute trajectory prices array at time step N on the GPU
for *Step = N-1 to 1* **do**
    Transfer Brownian motion arrays from the GPU to the CPU
    Transfer trajectory price arrays from the GPU memory to the CPU
    Route Brownian motion arrays across the cluster nodes (all-to-all)
    Route trajectory price arrays across the cluster nodes (all-to-all)
    Transfer received Brownian motion arrays from the CPU to the GPU
    Transfer received trajectory price arrays from the CPU to the GPU
    Compute Gaussian random numbers on the GPU
    Compute new Brownian motion arrays on the GPU
    Compute (update) asset prices on the GPU
    Compute the trajectory payoff arrays on the GPU
    Compute new trajectory price arrays on the GPU
end
Transfer trajectory price arrays from the GPU to the CPU
Compute payoff value on the CPU
Reduction of sums across the cluster on node 0
Compute final price and error values on node 0, on the CPU
Print results (price and error) and performances

Algorithm 6 shows the main steps of our non-overlapping algorithm. The size of the arrays will be specified later on Figure 3.3.The bold text lines concern the computations and the transfers that have to be done before launching the computations of the AC phase given in the mathematical Algorithm 5 on page 94.

All computations are carried out on the GPUs except some small ones performed after the computation loop. All data transfers between CPU and GPU in the computation loop are required to complete inter-node cluster communications and to participate to inter-GPU communications.

The inter-node communications of Brownian motion arrays (B) [9] and price arrays (Pr) [10] are expensive *all-to-all* communications. Then, each node stores in its CPU and GPU memories results of all nodes: it stores a large array indexed by all Monte Carlo trajectories computed in the cluster (see section 3.3.3). This solution *has not an infinite scaling* (it is not possible to process very large problems using a larger number of nodes). But considering the number of trajectories required by the American pricing problem and the current size of GPU memories, we will see in section 4 that our algorithmic solution is efficient and *scales* at least from 1 to 16 nodes on a 2^{19} trajectory problem (with 5 assets). Our first investigations prove it is possible to replace the one step *all-to-all* and *total storage* on each node by several substeps of a data circulation with limited storage. However, it would replace each large MPI communication and CPU/GPU transfer by many small ones, and should decrease efficiency. This alternative solution should be regarded for problems that require large number of trajectory simulation or dimension $d > 10$, some of these are multidimensional stochastic control problems formulated with BSDEs [14].

3.3.3 CPU and GPU data structures

Figure 3.3 introduces all arrays and variables implemented on each cluster node. All the computations are done in single precision using mainly float arrays and few integer arrays needed for the random number generation. On top of Figure 3.3 we can see four CPU arrays used to store model parameters, random number generator parameters and initial values of asset prices. These arrays are transferred just once to the GPU, at the beginning of the program. In the middle of Figure 3.3 are the Brownian motion and trajectory price output arrays of the node. They are double buffers, storing results of previous and current time steps. Results of the previous time step are transferred into CPU buffers and routed to all other nodes, in order to compute the

9. contains the values of $\{W^i_{t+\delta t}\}^l_{1\leq i\leq d}$, see Algorithm 5 on page 94.
10. contains the values of $P_{t+\delta t}(S^l_{t+\delta t})$, see Algorithm 5 on page 94.

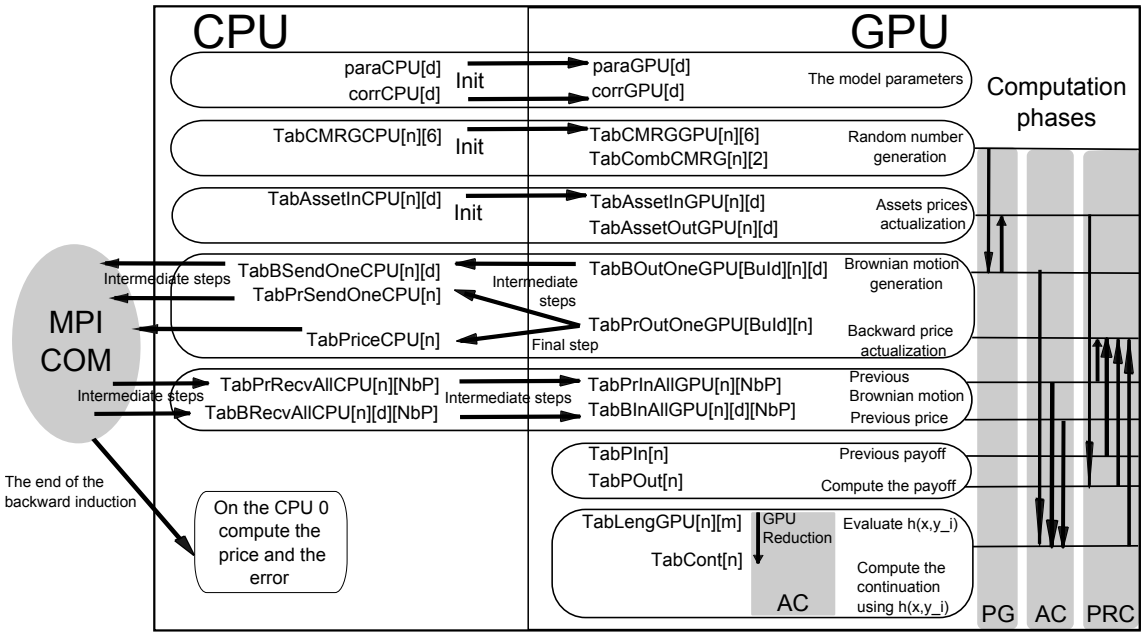

FIGURE 3.3 – Overview of all data structures implemented on each node (the parameter NbP is defined in section 3.4.2).

approximation (3.4) (see section 2). Then, some large 3-dimensional CPU arrays storing data of all nodes are transferred into symmetric GPU arrays on each GPU. On each node these large arrays store results of all trajectories computed on the cluster and impede our application to have an *infinite scalability*. But this solution allows to group data and to speedup data transfer and routing, as introduced in section 3.3.2. In section 4, we will see the application performance scales right from 1 up to 16 nodes on 2^{19} trajectories (which is the largest problem considered). Finally, on the bottom of Figure 3.3, the remaining GPU tables are used to compute intermediate results. Arrows on the right side of Figure 3.3 show the data arrays involved in computations of other data arrays, during the three phases of the algorithm detailed in Algorithm 4.

Besides, the different buffers of Pr and B can be allocated in the CPU memory in a standard way using *malloc* routine, or can be allocated by *cudaMallocHost* routine and compound of locked memory pages. Theoretically, this last solution leads to faster data transfers between CPU and GPU memories, but can not be used on too large buffers without disturbing the CPU memory management. During our numerous experiments, some unexpected bad termination of our application appeared when calling some MPI communication routines with locked memory. So, we have implemented both CPU buffer allocation mechanisms, controlled at runtime by an option on the command line.

3.3.4 Fine grained GPU computations

Our American option pricer includes 15 GPU kernels. Fourteen are classical intensive computing kernels, designed to carry out a maximum of coalescent data accesses. When calling these kernels we run one thread per trajectory processed by the node, and we run classical numbers of 128 or 256 threads per block (a CUDA block of threads is run on one SIMD multiprocessor of a NVIDIA GPU). The last kernel (the fifteenth) is the AC2 kernel of the AC phase that reduces some price arrays to mean prices.

Depending on the size of the problem, the most complex kernel (the biggest execution time) is either first or second in term of time consumption. On the whole it performs AC1 computations and some reductions (depending on the configuration) of AC2 of Algorithm 5 (see section 2): it computes an evaluation of the h function (defined (3.12), approximation discussed in 3.2.2), and a part of the sum of the Monte Carlo approximation (AC2 of algorithm 5). Because the data are coalescent, in kernel the AC2 reduction is efficiently achieved during the AC1 computations. As this kernel runs a large number of threads, we use 16 CUDA *streams* to asynchronously and concurrently run several medium grids of thread blocks (in place of a large one). This solution allows the GPU scheduler to optimize the kernel runs and to saturate the GPU that leads to an increase in performance close to 10% using 16 streams.

The fifteenth kernel completes the rest of this reduction, and aims to reduce a 2-dimension array [11] $TabLengGPU[n][m]$ (Figure 3.3) of prices into a 1-dimension array [12] $TabCont[n]$ (Figure 3.3) of average prices [13] (one per trajectory processed by the kernel). This important reduction has been implemented according to the strategy recommended by NVIDIA [29]. Each computation of one value of the final array is achieved running a set of blocks of threads, with optimized memory accesses, using the shared memory of each multiprocessor of the GPU, and optimizing the code at compilation time according to the problem size. To reduce the results of all these blocks to a unique value, we have chosen to use an `atomicAdd` on `float` values, available on FERMI architecture with CUDA 4.0. The overall strategy achieves good performances, however we will see in section 3.4.2 the global performance of our application can be increased with an over-splitting of the problem, in order to run several MPI processes on each node.

11. $n_x \times n_y \times m_x \times m_y$
12. $n_x \times n_y$
13. A conditional expectation averages and the continuation is considered as a price because it is the expected price of the future knowing the present.

Due to the limitation in the GPU memory, we process the trajectories of the node per sub-parts. This strategy reduces the size of some intermediate result arrays stored in the GPU memory. It also allows to process a large number of trajectories per node without requiring new data transfers between CPU and GPU. However, it appeared more efficient to process big sub-parts and section 3.4.2 introduces the impact of the sub-part size on the global efficiency.

3.4 Experimentation and optimization

3.4.1 Results accuracy

We test our simulations on a geometric average put option and a put option on minimum that both respectively have the following payoffs

$$\Phi^d_{geo}(S_t) = \left(K - \prod_{i=1}^{d} (S_t^i)^{1/d} \right)_+, \quad \Phi^d_{min}(S_t) = \left(K - \min_{1 \leq i \leq d} (S_t^i) \right)_+. \tag{3.23}$$

The parameters of the simulations are the following: The strike $K = 100$, the maturity $T = 1$, the risk neutral interest rate $r = \ln(1.1)$, the time discretization is defined using the number of exercise dates given as a parameter in each simulation, $S_0^i = 100$ and $\sigma = \{\sigma_{ij}\}_{1 \leq i,j \leq d} = 0.2\{\delta_{j-i} + cp(1-\delta_{j-i})\}_{1 \leq i,j \leq d}$ where δ is the Kronecker delta and cp is the correlation parameter $\in [0, 1]$ that we will fix for each simulation.

When $cp = 0$, the simulations using the d-dimensional payoff Φ^d_{geo} can be compared to the true values that are artificially set by the one-dimensional equivalence and a tree method [12], available in Premia [32]. For Φ^d_{min}, we only compared the coherence of the obtained results with the ones simulated by the Longstaff-Schwartz algorithm implemented on Premia that uses control variates and importance sampling techniques to reduce the variance.

First the importance sampling constants $\omega = (\omega_{11}, \omega_{12}, \omega_{22})$ in (3.22) are chosen in order to ensure a sufficient number of positive occurrence of the random variables θ_1 and θ_2 (defined in (3.19)). A good solution was found when taking ω_{11}, ω_{12} and ω_{22} that satisfy $c_2^{11}\omega_{11} = 2\delta t$, $c_2^{12}\omega_{12} = 2\delta t$ and $c_2^{22}\omega_{22} = 2\delta t$. For example, in the Black & Scholes model we have $c_2^{22} = 0$ and using the parameters of the model above, it was sufficient to take $\omega_{11} = \omega_{12} = \sqrt{\dfrac{k+1}{k * \delta t}}$ where k is the time step of the algorithm.

TABLE 3.2 – The accuracy of the results when changing the number of drawings N_ω of (Z_{11}, Z_{12}, Z_{22}). The number of simulated trajectories $n = 2^{10}$, we used 10 exercise dates and the standard deviation of the simulations is less than 5% of the price

Dimension d	$N_\omega = 4$	$N_\omega = 16$	Real price for Φ^d_{geo}
5	1.68	1.63	1.58
10	0.89	0.90	0.89

From Table 3.2, we see clearly that N_ω can be taken < 32. This is the case for other performed simulations, in particular, the one given in Table 3.3 that shows also the accurate results that we obtain even when we take $q = 2$.

TABLE 3.3 – The accuracy of the results when changing the value of q involved in (3.13) and (3.14). The number of simulated trajectories $n = 2^{14}$, $N_\omega = 4$, $d = 10$, we used 10 exercise dates and the standard deviation of the simulations is less than 2% of the price

Correlation parameter cp	$q = 2$	$q = 5$	$q = 10$	Premia version of LS Φ^d_{min}
0.2	15.35	15.66	16.01	16.63
0.5	13.01	13.17	13.24	13.17
0.8	9.04	9.02	9.10	9.31

3.4.2 Parameters tuning

Among the parameters that act on the efficiency of the simulations, we study the two parameters m and NbP that are highly influential on the execution time and their reaction is completely understandable from the AC phase of Algorithm 5 on page 94. The definition of m and NbP will be given during the explanations that follow.

In Algorithm 5 on page 94, we make n evaluations of $h\left(S_t^{l_1}, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right)$, for each $l_1 \in \{1, \dots n\}$, on n different points. If the memory space of the GPU were sufficient, we would make $n \times n$ evaluations of the function h according to l_1 and l_2. Nevertheless, we can only saturate the GPU memory size for an $l_1 \in \{1, \dots m\}$ and make $m \times n$ evaluations of the function h that yield to $m \times n$ data on the GPU which are stored in the table $TabLengGPU[n][m] = \left\{h\left(S_t^{(l_1)}, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right)\right\}_{1\leq l_1\leq m, 1\leq l_2\leq n}$ (Figure 3.3). Once we perform these $m \times n$ evaluations, we launch in parallel $2 \times m$ reductions [14] associated to the

14. one reduction for the numerator and one for the denominator. Here we present only the principal reductions

Algorithm 7: The AC phase when $NbP = 1$.

Input: $P_{t+\delta t}(S^l_{t+\delta t})$ and $\{W^i_{t+\delta t}\}^l_{1\leq i\leq d}$ with $l \in \{1,\dots,n\}$
Output: $C(S^l_t)$ with $l \in \{1,\dots,n\}$
for $l_3 \in \{0,\dots,n/m-1\}$ **do**

AC1) For $l_1 \in \{1+l_3*m,\dots,m+l_3*m\}$ and $l_2 \in \{1,\dots,n\}$, evaluate

$$h\left(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right)$$

AC2) For $l_1 \in \{1+l_3*m,\dots,m+l_3*m\}$ and either $(n',n'') = (n/2,n)$ or $(n'',n') = (n,n/2)$, set

$$C(S^{l_1}_t) \approx \frac{\frac{1}{n'}\sum_{l_2=1}^{n'} e^{-r\delta t} P_{t+\delta t}(S^{l_2}_{t+\delta t}) h\left(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right)}{\frac{1}{n''}\sum_{l_2=1}^{n''} h\left(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right)}$$

end

computation of the sum of the Monte Carlo approximation AC2. In our simulations, n can be divided by m because they are both equal to a power of two. Thus, we repeat n/m times the sequence of $m \times n$ evaluations as well as m parallel reductions and Algorithm 5 changes into Algorithm 7.

Although Algorithm 7 seems optimal when compared to Algorithm 5, we have found ourselves facing two problems:

- The non-coalescence of $TabLengGPU[n][m]$ data, indeed, during the AC2, we reduce the m sub-blocks of n prices arrays into m average prices stored in a sub-part of $TabCont[n]$ (thanks to an index in $\in \{1,...,n/m\}$).
- The number blocks of GPU threads used in the reduction phase AC2 is related to the number m. Figure 3.4 shows how the parameter m acts on the execution time for a given problem.

The number m is fixed by the memory space of the GPU, consequently we cannot increase, as much as we want, the number of blocks needed for the AC2 phase unless we transfer the data at each step to the CPU [15]. Fortunately, we reach a sufficient number of blocks with an $m \geq 512$ (see Figure 3.4) that allow to have good performances. Besides, we can use the parameter NbP to reduce the non-coalescence of the data and to perform some AC2 reductions during the AC1 operations. The parameter NbP is used to determine the number of processes used in our application on the cluster, that is to say

because in reality there are 4 reductions, performed in the same time, needed to fix also the value of n' and n'' according to what is detailed in the previous chapter.

15. All the tested solutions based on CPU/GPU transfers were inefficient.

$$NbP = \text{ nb of processes per node } \times \text{ nb of nodes.} \tag{3.24}$$

Algorithm 8: The AC phase launched by each process l_4 with $l_4 \in \{1, ..., NbP\}$.

Input: $P_{t+\delta t}(S^l_{t+\delta t})$ and $\{W^i_{t+\delta t}\}^l_{1\leq i\leq d}$ with $l \in \{1, \ldots, n\}$

`/* The output data are not the same and depend on` l_4`. */`

Output: $C(S^l_t)$ with $l \in \{1 + (l_4 - 1) * n/NbP, \ldots, n/NbP + (l_4 - 1) * n/NbP\}$

for $l_3 \in \{0, \ldots, n/(NbP * m) - 1\}$ **do**

AC1) For $l_1 \in \{1 + l_4 * l_3 * m, \ldots, m * NbP + l_4 * l_3 * m\}$ and $l_2 \in \{1 + (l_4 - 1) * n/NbP, \ldots, n/NbP + (l_4 - 1) * n/NbP\}$, evaluate

$$h\left(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2}_{1\leq i\leq d}\right).$$

Then for $l_2 \in \{1, \ldots, n/NbP\}$ compute Nu_{l_1,l_2} and De_{l_1,l_2}, such that

$$Nu_{l_1,l_2} = \sum_{l_5=1}^{NbP} P_{t+\delta t}(S^{l_2*l_5}_{t+\delta t}) h\big(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2*l_5}_{1\leq i\leq d}\big)\ ,\ De_{l_1,l_2} = \sum_{l_5=1}^{NbP} h\big(S^{l_1}_t, \{W^i_{t+\delta t}\}^{l_2*l_5}_{1\leq i\leq d}\big)$$

AC2) For $l_1 \in \{1 + l_4 * l_3 * m, \ldots, m * NbP + l_4 * l_3 * m\}$ and either $(n', n'') = (n/2, n)$ or $(n'', n') = (n, n/2)$, set

$$C(S^{l_1}_t) \approx \frac{\frac{1}{n'} \sum_{l_2=1}^{n'/NbP} e^{-r\delta t} Nu_{l_1,l_2}}{\frac{1}{n''} \sum_{l_2=1}^{n''/NbP} De_{l_1,l_2}}$$

end

To keep it simple, let us consider the operations performed on one node: If $NbP = 8$ instead of 1, 8 sums of AC2 reductions are done in the most complex kernel that executes the AC1 phase. Subsequently, $TabLengGPU[n][m]$ becomes $TabLengGPU[n/8][m]$ which decreases the number of non-coalescent reductions executed in the fifteenth kernel that performs the remaining reduction of AC2. Moreover, to perform AC1, the different processes $l_4 \in \{1, ..., 8\}$ will launch concurrently on the GPU the same task but with 8 different result data arrays $TabLengGPU[n/8][m]$ which provides $TabLengGPU[n/8][m \times 8]$. Subsequently, Algorithm 7, which was launched by only one process, becomes Algorithm 8 that is launched concurrently by NbP processes.

From Algorithm 8, we remark that the non-coalescent table of data used in the reduction AC2 is NbP times smaller. Moreover, the reduction performed in AC1 uses a coalescent structure of data produced by the NbP concurrent evaluations of h and this reduction is well defined

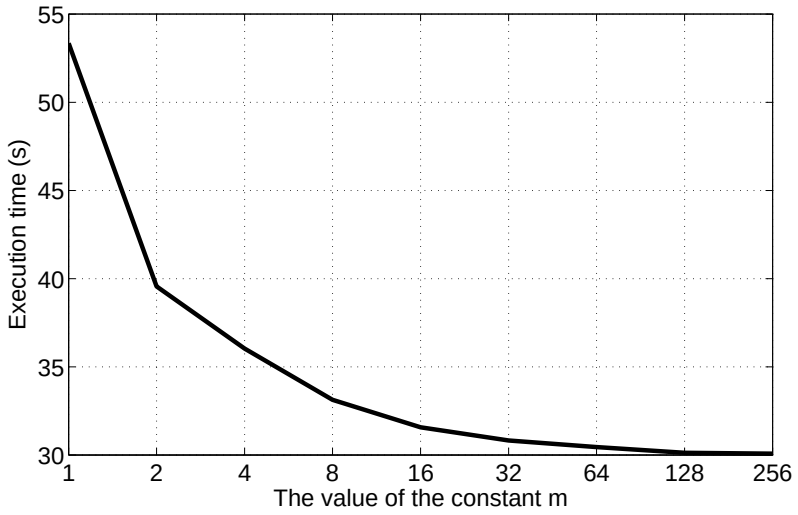

FIGURE 3.4 – The execution time according to the size of the sub-problems : 2^{16} trajectory simulation processing 5 asset problem. Here, the number of processes (NbP) is equal to one.

because it is performed once the data of the evaluation are available.

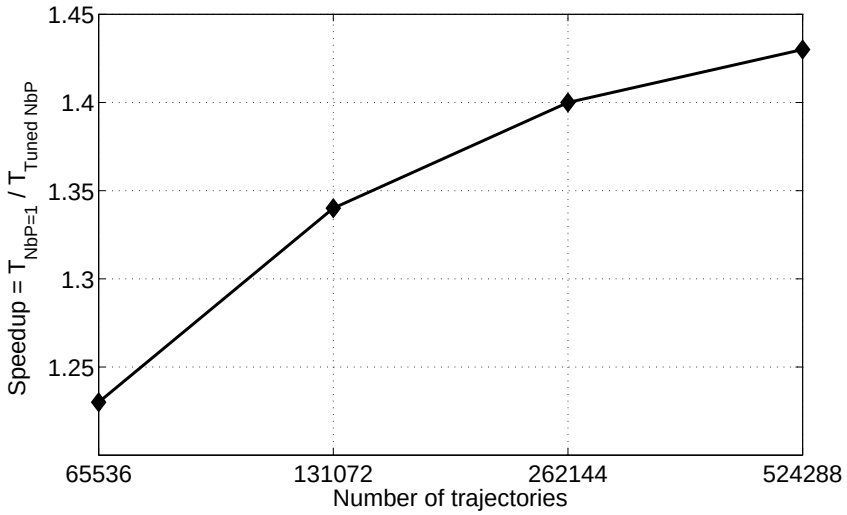

FIGURE 3.5 – On one node : The speedup obtained when using the appropriate number of processes (NbP) instead of only one process for pricing American Option on 5 assets.

In Figure 3.5, we quantify the benefits of using NbP on one node. It shows an increasing speedup when we increase the size of the problem which can be explained by the high non-coalescence of the data when we use a large number of trajectories. We should also point out that the value of the appropriate NbP varies according to the number of simulated trajectories as well as the dimension of the problem and the best choice of NbP does not exceed 8 for

the whole implemented simulations. According to our experimentations, for five assets and on one node, the best NbP appeared to be equal to 4 for 2^{16} trajectories and equal to 8 for 2^{17}, 2^{18} and 2^{19} trajectories. The solution based on using NbP to decrease the non-coalescence of the data is possible on Fermi architecture and it is interesting because it does not increase the complexity of our code and provides fair overall speedups when compared to the CPU OpenMP implementation (see section 3.5.2).

3.5 Mono-node comparison and scalability

3.5.1 Testbed introduction

Our testbed is a cluster of 16 nodes. Each node is a PC composed of an Intel Nehalem CPU with 4 hyperthreaded cores at 2.67GHz, 4GB of RAM, and a NVIDIA GTX480 GPU with 1.5GB of memory. This cluster has a Gigabit Ethernet interconnection network built around a small DELL Power Object 5324 switch (with 24 ports). The energy consumption of each node is monitored by two Raritan DPXS20A-16 device, that continuously measures the electric power dissipation (in Watts). Each of these device can monitor up to 20 nodes, but can deliver a maximum of 16A current power. It appeared we need to use two Raritan DPXS20A-16 devices to monitor our 16 node cluster. These devices host a SNMP server that a client can question to get the instantaneous power consumption of each node.

We developed SNMP clients to get these data with a sampling period of $1s$ and compute the consumed energy as a definite integral using the trapezoidal rule. This complete energy measurement system is a little bit complex, and can disturb the application execution. Moreover, the measurement resolution of our complete system appears to be close to $6Watts$ on each node. However, this mechanism allows to take into account the total energy consumed by the entire node: the PC hosting the CPU and the GPU, not just the energy consumed by the CPU and GPU cards.

During all our experiments we observe some systematic variations between two consecutive measurements. All time measures introduced in this work are means of 10, 20 or 50 successive executions. Moreover, we never take into account the first execution of a sequence, because the GPU increases its frequency and its performances during this first computation. All energy measurements are achieved on a sequence of 10 to 20 executions (avoiding the first execution) in order to measure significant amounts of energy. Then we deduce the energy consumed by only one execution.

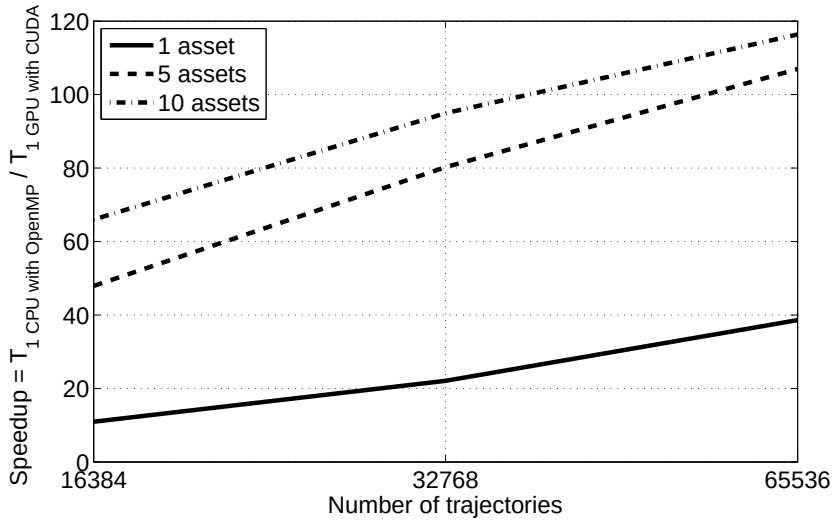

FIGURE 3.6 – Mono-node comparison and dimension: CPU OpenMP implementation *vs* GPU CUDA implementations

3.5.2 Mono-node GPU *vs* CPU comparison

In this subsection, the execution time and the energy consumption of using one GPU with CUDA are compared with the ones of using one CPU with OpenMP. Due to GPU memory architecture (shared memory), the effectiveness of the implementation on GPUs increases when we increase the dimension. This fact justifies the speedup results obtained in Figure 3.6 and thus we compare the energy efficiency only for an average dimensional problem (5 assets). According to Figure 3.7, the efficiency of using the GPU instead of the CPU increases when we increase the number of simulated paths which indicates that the GPU is under-exploited under 2^{16} trajectories. Thus, in subsection 3.5.3, we will study only the problem larger than 2^{16} trajectories. For 2^{16} trajectories problem, we obtain ~ 107 as a speedup and ~ 52 as an energy ratio which promotes clearly the GPU implementation.

3.5.3 GPU cluster performances

We only consider, in this section, the lowest [16] dimension (five assets) for which the other methods based on regression become insufficiently accurate especially for evaluating greeks. For dimensions $d > 5$, one can reasonably multiply the execution time and the energy consumed, provided in the figures below, by $d/5$ to obtain the ones associated to the dimension d. As we will see in Figure 3.11, the latter approximation becomes wrong for dimensions $d \gtrsim 20$ because of the communication time between CPUs in the cluster implementation.

16. Because benchmarking would take much more time to do for larger problems.

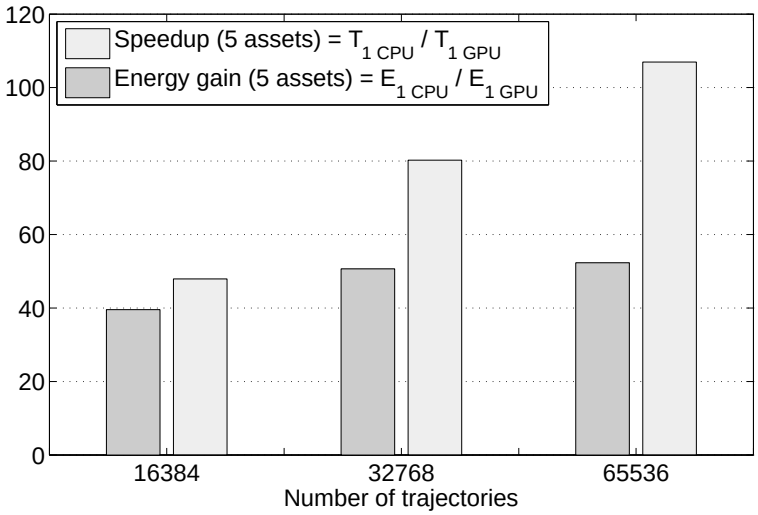

FIGURE 3.7 – Mono-node comparison (Energy + Speedup): CPU OpenMP implementation *vs* GPU CUDA implementations

Speedup and scalability

The curves of Figure 3.8 show a regular decrease of the execution time on our GPU cluster from 1 node up to 16 nodes, function of to the number of trajectories for 5 assets. The curves identified with ♦ represent the implementations that use a number of MPI processes equal to the number of nodes, when the other curves represent the simulations obtained by the appropriate choice of NbP. When we increase the number of nodes, the best choice of NbP tends clearly to be equal to the number of nodes (see equality (3.24)).

The curves of Figure 3.9 show the speedup associated to the best configuration of m and NbP on several nodes when compared to the best configuration of m and NbP on one node. When processing 2^{16} trajectories on the five assets problem, the speedup slows down on 16 nodes (Figure 3.9). The problem becomes too small to fully use 16 NVIDIA GTX480 GPUs. When processing 2^{17} or 2^{18} trajectories problem, the scalability of our parallelization is very good from 1 to 16 nodes. The 2^{17} size problem speedup is very close to the ideal speedup ($S(p) = p$), and the 2^{18} size problem speedup exhibits some hyper-accelerations. Indeed, our parameter tuning allows to use both more GPUs (using more nodes) and more GPU memories (increasing the size of the sub-problem we can process on each node).

Fixing the execution time, Figure 3.10 represents the different problem sizes that can be simulated when we increase the number of nodes. We remark that doubling the number of trajectories requires quadrupling the number of nodes if we want to keep the same execution time. This can be explained by the complexity of the AC phase that is equal to $O(n^2)$. This affirmation is a bit less verified for the average sized problems (2^{16}, 2^{17} and 2^{17}), for which

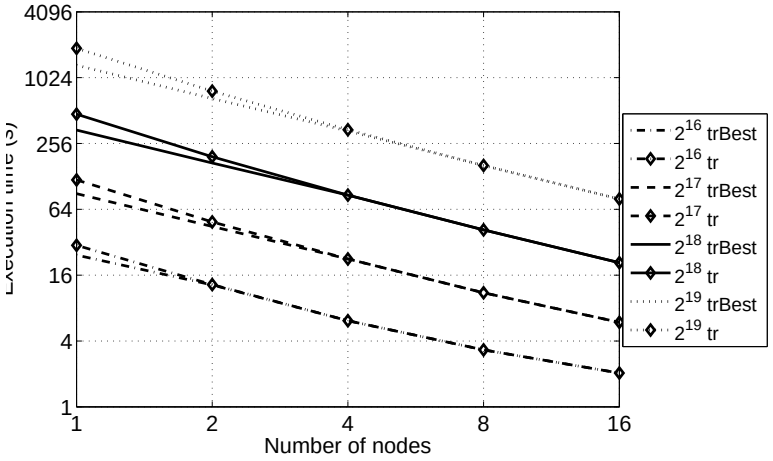

FIGURE 3.8 – Execution time on GPU cluster with 5 assets

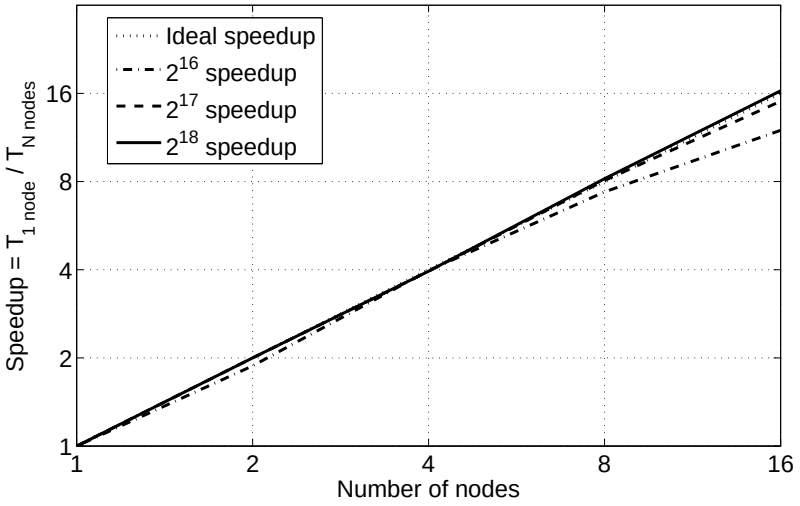

FIGURE 3.9 – Speedup on GPU cluster with 5 assets

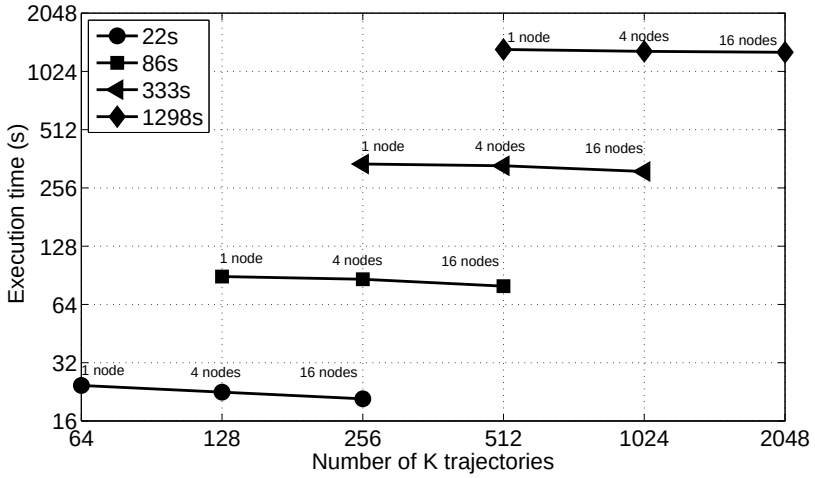

FIGURE 3.10 – Allocating more resources for 5 assets problem

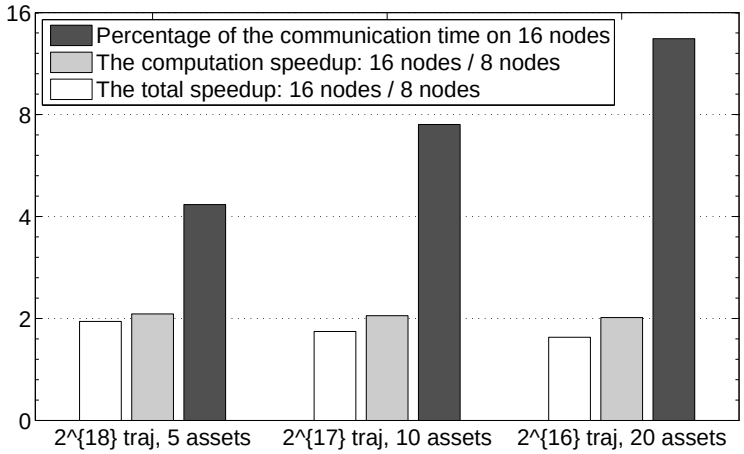

FIGURE 3.11 – When the communication time cannot be neglected

the PG and the PRC phases remain time consuming when compared to the overall execution time [17].

Finally, we point out that the excellent speedup of our simulations is largely due to the fact that the cluster communication time can be neglected when compared to the computation time. However this is not true for pricing contracts on a high number of assets. According to Figure 3.11, even for computations that provide a speedup equivalent to the number of nodes, the communication time becomes sufficiently big for $d = 20$ which deteriorates significantly the total speedup from 2.013 to 1.76. This leads us to explore, in future work, new ideas to overlap communications with computations.

17. The complexity of these two phases is equal to $O(n)$.

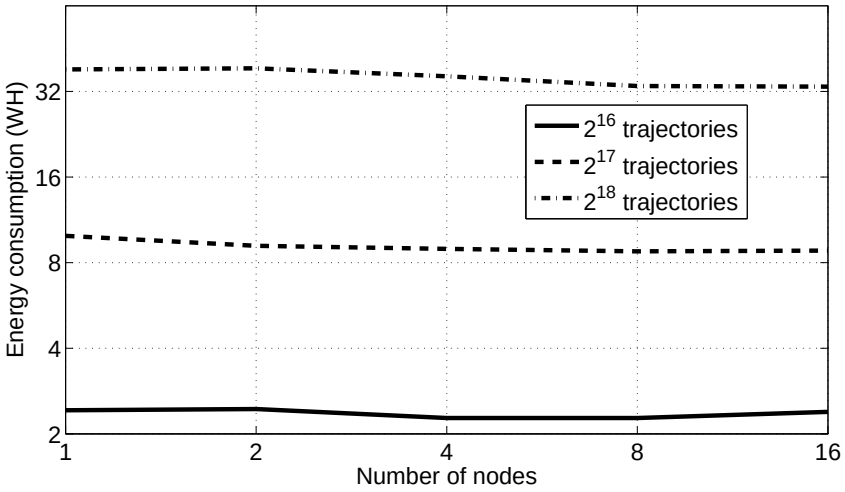

FIGURE 3.12 – Energy consumption on GPU cluster processing 5 assets problem

Energy consumption issues

Figure 3.12 shows the energy consumption of our application when processing 5 assets and different numbers of trajectories. This consumption is computed considering the energy consumed by the entire nodes (the complete PCs) and the fraction of the energy consumed by the switch of the cluster. We measured and summed the exact electrical power dissipated by each used node (we did not considered all nodes dissipate the same power). We considered a cluster is shared by several users, and each user allocates some nodes to run his applications (this is the management strategy of our clusters). We measured that the electrical power dissipated by our switch remains constant, independently of the network trafic, and was $P_0^{sw} = 34Watts$. So, we have taken into account the power dissipation $P^{sw} = P_0^{sw} \times NbUsedNodes/NbNodes$. Finally, we integrated all these measures during the execution time, and we obtained the energy consumption curves of Figure 3.12 (in $W.h$).

We observe that the energy consumed by our GPU cluster does not increase when we use more nodes, in fact, it tends to remain constant. Because of the relatively small size of the 2^{16} trajectories problem, we observed a significant slow down on 16 nodes which leads to higher energy consumption. When processing 2^{17} trajectories the speedup was close to the ideal speedup ($S(P) = P$) and the energy consumption remains approximately constant. When simulating 2^{18}, trajectories we achieved a small hyper-speedup up to 16 nodes that leads to a regular decrease of the energy consumption. This decrease and the associated hyper-speedup would stop if we could use more nodes.

3.6 Conclusion and future work

Pricing American contracts is one of the most challenging problems in mathematical finance. Our approach was to solve the problem of pricing multidimensional American contracts with a nonparametric method whose accuracy is only related to the number of the simulated paths. In this chapter we provided appropriate approximations for high dimensional problems and we proved the high adaptability of MCM on parallel architectures based on GPUs. Thanks to our experiments and results, we know that parallelism solves very efficiently the problem of increasing the number of simulated trajectories used in the nonlinear problem of pricing multidimensional American contracts. The pricing of an American option on ten assets using 2^{17} trajectories provided very accurate results and was performed within 11 seconds on our cluster of 16 GPUs.

To achieve these performances we designed some parallel algorithms cumulating 2 grains of parallelism and 2 parallel programming paradigms using MPI+CUDA, in order to run on clusters of CPU+GPU hybrid nodes. An optimal configuration was found for each number of nodes and problem size, on a specified GPU cluster. Speedups were close to the ideal speedup and sometimes better (small hyper-speedup) due to usage of both more GPUs and more GPU memories. Energy consumption tends to remain constant when increasing the number of nodes.

Finally, we obtain a very suited solution for GPU clusters that scales well on our benchmarks and allows very good speedup and energy consumption. However, our solution has a scalability limit replicating some data on each node, and requires a tuning to select best configurations. In addition, due to large communications when $d \geq 10$, we aim to design, in future work, a fully scalable variant to process very large problems and that cumulates 3 grains of parallelism implemented on top of MPI+OpenMP+CUDA which overlaps well the computations and the data transfers between GPUs.

3.7 Appendix

Proof of Lemma 3.1:

We use $A^{i,j}$ to denote a matrix, derived from A^d, that has $\pi^{j,d}$ as the last element of its diagonal and on which we suppress the line and the column of index i. For example, if $(i, j) = (1, q + 1)$

then

$$A^{1,q+1} = \begin{pmatrix} \pi_{s,t}^{2,d} & C_{2,3} & C_{2,4} & \cdots & C_{2,q} & 0 \\ 1 & \pi_{s,t}^{3,d} & 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 1 & \cdots & 1 & \pi_{s,t}^{q-1,d} & 0 & 0 \\ 1 & 1 & \cdots & 1 & \pi_{s,t}^{q,d} & 0 \\ 1 & 1 & 1 & \cdots & 1 & \pi_{s,t}^{q+1,d} \end{pmatrix}.$$

By induction, one can easily prove the following equalities

$$\left|A^{d}\right| = \left|A^{1,q}\right| \left(\sum_{l=q+1}^{d} (-1)^{l+1} C_{1,l} \left[\prod_{q+1 \le i \le d}^{i \neq l} \pi_{s,t}^{i,d} \right] \right) + \left[\prod_{q+1 \le i \le d} \pi_{s,t}^{i,d} \right] \left|A^{q}\right|,$$

with

$$\left|A^{1,q}\right| = \sum_{j=3}^{q} (-1)^{j} C_{2,j} \left[\prod_{3 \le i \le q}^{i \neq j} \pi_{s,t}^{i,d} \right] + \left[\prod_{2 \le i \le q} \pi_{s,t}^{i,d} \right]$$

and

$$|A^{q}| = \begin{vmatrix} \pi_{s,t}^{1,d} & C_{1,2} & C_{1,3} & \cdots & C_{1,q-1} & C_{1,q} \\ 1 & \pi_{s,t}^{2,d} & C_{2,3} & \cdots & C_{2,q-1} & C_{2,q} \\ 1 & 1 & \ddots & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 1 & & & 1 & \pi_{s,t}^{q-1,d} & 0 \\ 1 & 1 & \cdots & 1 & 1 & \pi_{s,t}^{q,d} \end{vmatrix} = \begin{array}{c} \pi_{s,t}^{q,d} \left|A^{q-1}\right| + (-1)^{q+2} C_{2,q} \left|A^{2,q-1}\right| \\ +(-1)^{q+1} C_{1,q} \left|A^{1,q-1}\right| \end{array},$$

then, by induction

$$\begin{aligned} |A^{q}| = & \prod_{1 \le i \le q} \pi_{s,t}^{i,d} - C_{1,2} \prod_{1 \le i \le q}^{i \neq 1,2} \pi_{s,t}^{i,d} - C_{2,3} \prod_{1 \le i \le q}^{i \neq 2,3} \pi_{s,t}^{i,d} \\ & + \sum_{l=4}^{q} (-1)^{l+1} C_{1,l} \left[\prod_{2 \le i \le q}^{i \neq l} \pi_{s,t}^{i,d} + \sum_{j=3}^{l-1} (-1)^{j} C_{2,j} \left(\prod_{3 \le i \le q}^{i \neq j,l} \pi_{s,t}^{i,d} \right) \right] \\ & + \sum_{l=4}^{q} (-1)^{l+2} C_{2,l} \left[\prod_{1 \le i \le q}^{i \neq 2,l} \pi_{s,t}^{i,d} + \sum_{j=3}^{l-1} (-1)^{j} C_{1,j} \left(\prod_{3 \le i \le q}^{i \neq j,l} \pi_{s,t}^{i,d} \right) \right]. \end{aligned}$$

Combining all the previous equalities, we get the required result.

■

Acknowledgment: This work was supported by CreditNext and Région Lorraine. The authors (L. A. Abbas-Turki & S. Vialle) want to thank Professor Bernard Lapeyre for his valuable advice and Patrick Mercier for his continuous technical management of the GPU clusters.

Chapitre 4

European Options Sensitivity with Respect to the Correlation for Multidimensional Heston Models

Abstract

This paper is devoted to the sensitivity study of the European option prices according to the correlation parameters when dealing with the multi-asset Heston model. When the Feller condition is not fulfilled, the CIR flow regularity is needed to prove the differentiability of the price according to the correlation. In the bidimensional case when the Feller condition is satisfied, the regularity of the volatility according to the correlation allows us to establish an asymptotic expression of the derivative of the price with respect to the correlation. This approximation provides the monotony for the exchange options then heuristically for spread option prices at short maturities. We also obtain this monotony for some restrictive choices of the products $\{\eta_i\rho_i\}_{i=1,2}$ and $\{\eta_i\sqrt{1-\rho_i^2}\}_{i=1,2}$ where η_i is the volatility of the volatility and ρ_i is the asset/volatility correlation coefficient. Then, we explain how to extend the overall study to options written on more than two assets and on models that are derived from Heston model, like the double Heston model. We conclude by a large number of simulations that comfort the theoretical results.

4.1 Introduction

For a convex payoff, the authors of [34] prove the monotony of the price of a European contract according to the volatility of the Black & Scholes (B&S) model. In the same fashion,

let us prove the monotony according to the correlation parameter for the bidimensional B&S model

$$\begin{aligned} dS_t^1 &= S_t^1\sigma_1 dW_t^1, \quad S_0^1 = x_0^1, \\ dS_t^2 &= S_t^2\sigma_2(\rho dW_t^1 + \sqrt{1-\rho^2}dW_t^2), \quad S_0^2 = x_0^2. \end{aligned} \tag{4.1}$$

Let f be the convex payoff

$$f(s) = (a_1 s_1 + a_2 s_2 \pm K)_+ = \max(a_1 s_1 + a_2 s_2 \pm K, 0), \quad (a_1, a_2) \in \mathbb{R}^2 \tag{4.2}$$

and $F(t,x)$ the price of the studied contract, given under the risk-neutral probability by

$$F(t,x) = E\left(f(S_T)\middle| S_t = x\right) = E_{t,x}\left(f(S_T)\right).$$

Associated to the model (4.1) and to the convex payoff (4.2), the price function $F(t,x) \in \mathcal{C}^{1,2}\left((0,T)\times\mathbb{R}_+^2\right)$. This can be justified by the fact that the asset vector S_T has a log-normal distribution which is sufficient to perform the wanted differentiations. Besides, $F(t,x)$ satisfies the Black & Scholes PDE

$$\frac{\partial F}{\partial t}(t,x) + \frac{\sigma_1^2}{2}\frac{\partial^2 F}{\partial x_1^2}(t,x)x_1^2 + \frac{\sigma_2^2}{2}\frac{\partial^2 F}{\partial x_2^2}(t,x)x_2^2 + \rho\sigma_1\sigma_2\frac{\partial^2 F}{\partial x_1\partial x_2}(t,x)x_1x_2 = 0,\ F(T,x) = f(x),$$

We suppose now that the misspecified asset vector has the dynamic (4.1) but with a misspecified correlation $\overline{\rho} \neq \rho$, that is to say

$$\begin{aligned} d\overline{S}_t^1 &= \overline{S}_t^1\sigma_1 dW_t^1, \quad \overline{S}_0^1 = x_0^1, \\ d\overline{S}_t^2 &= \overline{S}_t^2\sigma_2(\overline{\rho} dW_t^1 + \sqrt{1-\overline{\rho}^2}dW_t^2), \quad \overline{S}_0^2 = x_0^2. \end{aligned}$$

We have already seen that $F(t,x) \in \mathcal{C}^{1,2}\left((0,T)\times\mathbb{R}_+^2\right)$ and using Ito calculus

$$\begin{aligned} F(T,\overline{S}_T) = {} & F(0,\overline{S}_0) + \sum_{i=1}^{2}\int_0^T \frac{\partial F}{\partial x_i}(t,\overline{S}_t)d\overline{S}_t^i + \int_0^T \frac{\partial F}{\partial t}(t,\overline{S}_t)dt \\ & + \frac{1}{2}\int_0^T \left(\sigma_1^2\frac{\partial^2 F}{\partial x_1^2}(t,\overline{S}_t)\left[\overline{S}_t^1\right]^2 + \sigma_2^2\frac{\partial^2 F}{\partial x_2^2}(t,\overline{S}_t)\left[\overline{S}_t^2\right]^2 + 2\overline{\rho}\sigma_1\sigma_2\frac{\partial^2 F}{\partial x_1\partial x_2}(t,\overline{S}_t)\overline{S}_t^1\overline{S}_t^2\right)dt, \end{aligned}$$

Combining the previous equality with the Black & Scholes PDE we get

$$E(F(T,\overline{S}_T)) = F(0,S_0) + (\overline{\rho}-\rho)E\left\{\int_0^T \sigma_1\sigma_2\frac{\partial^2 F}{\partial x_1\partial x_2}(t,\overline{S}_t)\overline{S}_t^1\overline{S}_t^2 dt\right\}. \tag{4.3}$$

To compute the cross derivative, we consider the derivatives of S_T with respect to $S_t = x$ ($t < T$),

$$\partial_{x_1} S_T^1 = \frac{S_T^1}{S_t^1}, \quad \partial_{x_2} S_T^2 = \frac{S_T^2}{S_t^2}, \quad \partial_{x_2} S_T^1 = \partial_{x_1} S_T^2 = 0.$$

When the payoff (4.2) is used then

$$\frac{\partial^2 F}{\partial x_1 \partial x_2}(t,x) = a_1 a_2 E_{t,x}\left(\frac{S_T^1}{x_1}\frac{S_T^2}{x_2}\varepsilon(a_1 S_T^1 + a_2 S_T^2 \pm K)\right), \tag{4.4}$$

where ε is the Dirac distribution that can be justified thanks to the log-normal distribution g of $\left(\frac{S_T^1}{x_1}, \frac{S_T^2}{x_2}\right)$:

$$\begin{aligned}\frac{\partial^2 F}{\partial x_1 \partial x_2}(t,x) &= -\frac{a_2}{x_1}\int_{\mathbb{R}_+^2} 1_{a_1 x_1 v_1 + a_2 x_2 v_2 \geq \pm K}\partial_{v_1}\left[v_1 v_2 g(v_1,v_2)\right] dv_1 dv_2 \\ &= -\frac{a_1}{x_2}\int_{\mathbb{R}_+^2} 1_{a_1 x_1 v_1 + a_2 x_2 v_2 \geq \pm K}\partial_{v_2}\left[v_1 v_2 g(v_1,v_2)\right] dv_1 dv_2.\end{aligned}$$

From equality (4.4), $a_1 a_2 \frac{\partial^2 F}{\partial x_1 \partial x_2}(t,x)$ is clearly positive and the price is monotonous with respect to ρ. The direction of the latter monotony depends on the sign of the product $a_1 a_2$. As an analogue of the implied volatility, thanks to the uniqueness of ρ one can define it as the **implied correlation** obtained from the market calibration of two assets that has the bidimensional B&S dynamics. As we will see in section 4.3, this notion of implied correlation is difficult to prove theoretically when using more complex models, like the Heston model.

In this paper, the assumed bidimensional version of the Heston model presumes the following dynamic for the couples asset/volatility $(S_T^i, \nu_T^i)_{i=1,2}$

$$S_T^1 = x_1 \exp\left(\int_t^T \sqrt{\nu_s^1} dW_s^1 - \frac{1}{2}\int_t^T \nu_s^1 ds\right), \tag{4.5}$$

$$S_T^2 = x_2 \exp\left(\int_t^T \sqrt{\nu_s^2}\left(\rho dW_s^1 + \sqrt{1-\rho^2} dW_s^2\right) - \frac{1}{2}\int_t^T \nu_s^2 ds\right), \tag{4.6}$$

$$\begin{aligned}\nu_T^1 &= y_1 + \kappa_1 \textstyle\int_t^T (\theta_1 - \nu_s^1) ds + \eta_1 \int_t^T \sqrt{\nu_s^1} dB_s^1, \\ B_s^1 &= \rho_1 W_s^1 + \sqrt{1-\rho_1^2}\widetilde{W}_s^1,\end{aligned} \tag{4.7}$$

$$\begin{aligned} \nu_T^2 &= y_2 + \kappa_2 \int_t^T (\theta_2 - \nu_s^2) ds + \eta_2 \int_t^T \sqrt{\nu_s^2} dB_s^2, \\ B_s^2 &= \rho_2 \left(\rho W_s^1 + \sqrt{1-\rho^2} W_s^2 \right) + \sqrt{1-\rho_2^2} \widetilde{W}_s^2, \end{aligned} \tag{4.8}$$

where $(W^1, W^2, \widetilde{W}^1, \widetilde{W}^2)$ is a four-dimensional Brownian motion (these four Brownian motions are independent).

We point out that the model specified by the previous SDEs does not include all the bidimensional Heston models. Indeed, the choice of this correlation structure is justified from a practitioner's point of view because it allows to calibrate simply each asset to the one-dimensional put and call options, then add a correlation parameter ρ that can be calibrated from a spread option. Thus, the overall model will reproduce the prices of vanilla options and spread options. Although this model was already considered by various authors (see for example [17]) and widely used by practitioners, one of its drawbacks comes from constraining the correlation, between the Brownian motions of the two volatilities, to be equal to $\rho\rho_1\rho_2$.

Using the results of Bessel flow regularity in [59], we study in section 4.2 the regularity of the CIR flow related to the SDEs (4.7) and (4.8) then the volatility regularity with respect to the correlation of the Brownian motions. In section 4.3, we prove the differentiability of the price according to the correlation when the Feller condition is not fulfilled and we study some restrictive cases for which the price is monotonous with respect to the correlation. The derivative of ν^2 according to ρ is needed to establish in section 4.4 an asymptotic expression of the derivative of the price that works well for maturities $T \leq 0.3$. In sections 4.3 and 4.4, we present also the basic ideas that allow to generalize our results to the multi-asset Heston and to models that are derived from Heston model, like the double Heston model. Thanks to a parallel implementation on the GPU Nvidia 480GTX, section 4.5 shows several tests of the error of our asymptotic approximation and it provides various Monte Carlo simulations that illustrate the monotony.

4.2 CIR flow & volatility regularity according to the correlation

For a fixed $t \geq 0$ and for $s \geq t$, ν^1 and ν^2 share the same common CIR SDE given by

$$d\nu_s = \kappa(\theta - \nu_s) ds + \eta \sqrt{\nu_s} \left(r dW_s^1 + \sqrt{1-r^2} dW_s^2 \right), \quad \nu_t = y, \quad r \in [-1, 1], \tag{4.9}$$

where here the Brownian motions W^1 and W^2 are independent but are not the same as the ones used in the previous section. However, it is quite clear that studying the flow of ν in (4.9) is equivalent to studying the flow of ν^1 and ν^2 in (4.7) and (4.8). Moreover, the differentiability results of ν^2 with respect to ρ are similar to the differentiability results of ν with respect to r.

In this section, we use either the Feller condition

$$\textbf{(A0)} \quad y > 0, \quad 2\kappa\theta \geq \eta^2,$$

or the following weaker assumption

$$\textbf{(A1)} \quad y > 0, \quad 4\kappa\theta > \eta^2.$$

Introducing the process $(0,\infty) \ni y \mapsto \tau_0(y)$ defined by

$$\tau_0(y) = \inf\left\{s \geq t, \nu_s(y) = 0\right\}, \tag{4.10}$$

we refer for example to [38] for the proof of the finiteness of $\tau_0(y)$ once **(A0)** is not satisfied, which means for a fixed $y > 0$ we have $P\left(\tau_0(y) < \infty\right) = 1$.

The result of this section is summarized in the following theorem

Theorem 4.1 *Let ν be a CIR process driven by the SDE (4.9). Under the assumption **(A0)**, both applications $(0,\infty) \ni y \mapsto \nu_s$ and $(-1,1) \ni r \mapsto \nu_s$ are $\mathcal{C}^1$. When **(A0)** is not fulfilled but **(A1)** is satisfied, $(-1,1) \ni r \mapsto \nu_s$ remains continuous and there exists a modification $\widetilde{\nu}$ of ν such that $(0,\infty) \ni y \mapsto \widetilde{\nu}_s$ is $\mathcal{C}^1$ in probability sense. Moreover, the first derivative $\partial_y\widetilde{\nu}$ coincides with $\dot{\nu} := \partial_y\nu$ on $[t, \tau_0(y)[$ and the former derivative vanishes on $[\tau_0(y), \infty[$.*

*Besides, when either **(A0)** is fulfilled or **(A1)** is satisfied with $0 \leq t \leq s < \tau_0(y)$ then*

$$\frac{\dot{\nu}_s}{\sqrt{\nu_s}} = \frac{1}{\sqrt{\nu_t}} \exp\left(-\frac{\kappa(s-t)}{2} - \frac{1}{2}\left(\kappa\theta - \frac{\eta^2}{4}\right)\int_t^s \frac{du}{\nu_u}\right). \tag{4.11}$$

for these same assumptions and taking $t = 0$, $\partial_r\nu$ satisfies the following SDE

$$\begin{array}{rcl} \partial_r\nu_s & = & -\kappa\displaystyle\int_0^s \partial_r\nu_u du + \eta\int_0^s \frac{\partial_r\nu_u}{2\sqrt{\nu_u}}\left(rdW_u^1 + \sqrt{1-r^2}dW_u^2\right) \\ & + & \eta\displaystyle\int_0^s \sqrt{\nu_u}\left(dW_u^1 - \frac{r}{\sqrt{1-r^2}}dW_u^2\right) \end{array}, \quad \partial_r\nu_0 = 0, \tag{4.12}$$

that can be solved by a variation of constants method, to obtain

$$\partial_r \nu_s = \dot{\nu}_s \left(\eta r \int_0^s \frac{\sqrt{\nu_u}}{\dot{\nu}_u} \left(dW_u^1 - \frac{r}{\sqrt{1-r^2}} dW_u^2 \right) \right), \tag{4.13}$$

where in the latter equality, $\dot{\nu}$ is the flow derivative at $t = 0$ (replace t by 0 in (4.11)).

Note that (4.12) is only valid before time $\tau_0(y)$. Therefore, in order to prove the differentiability of the price with respect to the correlation under **(A1)**, we need additional work. This will be the main goal of section 4.3, in which we use the infinitesimal generator and the regularity of the flow. Unfortunately, the latter trick does not allow us to establish an asymptotic approximation and the only thing that we were able to do is to show that the asymptotic approximation, established when **(A0)** is fulfilled, works well numerically even for cases when only **(A1)** is satisfied.

Proof of Theorem 4.1:

We subdivide this proof into three steps

Step1 : Proving the regularity of the flow.

The solution of (4.9) is locally differentiable with respect to y, this means that we can differentiate with respect to y up to the time $\tau_0(y)$ which is the upper limit of $\tau_{1/n}^n(y) = \inf\{s \geq t : \nu_s^n \leq 1/n\}$, such that ν_s^n is the solution of the truncated SDE associated to (4.9) with $\nu_t^1 = y$ (we refer to [37] for more details). For $s \in [t, \tau_0(y)[$, we get

$$\dot{\nu}_s = 1 - \kappa \int_t^s \dot{\nu}_u du + \eta \int_t^s \frac{\dot{\nu}_u}{2\sqrt{\nu_u}} \left(r dW_u^1 + \sqrt{1-r^2} dW_u^2 \right). \tag{4.14}$$

By a change of variable using the logarithmic function, we obtain the solution of (4.14) for $s < \tau_0(y)$

$$\dot{\nu}_s = \exp\left(-\kappa(s-t) + \eta \int_t^s \frac{r dW_u^1 + \sqrt{1-r^2} dW_u^2}{2\sqrt{\nu_u}} - \frac{1}{2}\eta^2 \int_t^s \frac{du}{4\nu_u^i} \right). \tag{4.15}$$

Moreover, by another change of variable $X_s = \ln\left(e^{\kappa(s-t)}\nu_s\right)$, this time on the ν SDE, using Ito calculus we get

$$\exp\left(\eta \int_t^s \frac{r dW_u^1 + \sqrt{1-r^2} dW_u^2}{2\sqrt{\nu_u}} \right) = \sqrt{\frac{\nu_u}{y}} \exp\left(\frac{\kappa(s-t)}{2} - \frac{1}{2}\left(\kappa\theta - \frac{\eta^2}{2} \right) \int_t^s \frac{du}{\nu_u} \right),$$

which combined with (4.15) provides (4.11) for $s < \tau_0(y)$.

According to [59] (Proof of theorem 1.3), when $\delta \in]1, 2[$ the bessel flow $(0, \infty) \ni x \mapsto$

$\Theta(x,s)$, that satisfies (4.16) driven by the Brownian motion β

$$\Theta(x,s) = x + \beta_s + \frac{\delta-1}{2}\int_t^s \frac{du}{\Theta(x,u)}, \quad \Theta(x,t) = x, \tag{4.16}$$

has a modification that admits a continuous derivative in probability sense that vanishes when $s \geq \tau_0(y)$. Consequently, one can use the same modification for the CIR flow because they are both related by the following equalities

$$\nu_s = \exp\left[-\kappa(s-t)\right]\Theta^2\left(\sqrt{y}, \frac{\eta^2}{4\kappa}(e^{\kappa(s-t)}-1)\right), \quad \delta = \frac{4\kappa\theta}{\eta^2}. \tag{4.17}$$

To prove (4.17), we use Ito calculus on $Z_s = \exp\left[\kappa(s-t)\right]\nu_s$ and we employ the time change $l_s = \frac{1}{\kappa}\ln\left(\frac{4\kappa s}{\eta^2}\right) + t$ to get

$$Z_{l_s} = y + \frac{4\kappa\theta}{\eta^2}(s-t) + 2\int_t^s \sqrt{Z_{l_u}}d\beta_u.$$

Finally, defining $\Theta(\sqrt{y},s) = \sqrt{Z_{l_s}}$, we obtain (4.16) with $x = \sqrt{y}$.

Step2 : Proving the continuity of ν with respect to r.
We define two Brownian motions $B_s = rW_s^1 + \sqrt{1-r^2}W_s^2$ and $\overline{B}_s = \overline{r}W_s^1 + \sqrt{1-\overline{r}^2}W_s^2$ thanks to which we set

$$\begin{aligned} d\nu_s &= \kappa(\theta-\nu_s)ds + \eta\sqrt{\nu_s}dB_s, \quad \nu_0 = y, \\ d\overline{\nu}_s &= \kappa(\theta-\overline{\nu}_s)ds + \eta\sqrt{\overline{\nu}_s}d\overline{B}_s, \quad \overline{\nu}_0 = y \end{aligned} \tag{4.18}$$

and we will prove that $\lim_{\overline{r}\to r}\overline{\nu}_s = \nu_s$ a.s.

Let a_n be a positive decreasing sequence defined by $a_n = a_{n-1}e^{-n}$, that satisfies

$$\int_{a_n}^{a_{n-1}} \frac{dx}{x} = n. \tag{4.19}$$

Afterwards, we set $\phi_n \in \mathcal{C}_c^\infty(\mathbb{R})$ a mollifier function with support equal to $[a_n, a_{n-1}]$ such that $0 \leq \phi_n(x) \leq_{(*)} \frac{2}{nx}$ and (4.19) allows to have $\int_{\mathbb{R}}\phi_n(x)dx = 1$. Thanks to ϕ_n, we define an approximation

$$\psi_n(x) = \int_0^{|x|} dy \int_0^y \phi_n(z)dz \tag{4.20}$$

of the function absolute value $|\cdot|$. Indeed

$$\begin{aligned} |x| &= \int_0^{|x|} dy \left(\int_0^y \phi_n(z)dz + \int_y^\infty \phi_n(z)dz \right) \\ &= \psi_n(x) + \int_0^{|x|} dy \int_y^\infty \phi_n(z)dz \end{aligned}$$

thus, $|x| \geq \psi_n(x)$ and because $\int_y^\infty \phi_n(z) \leq 1_{[0,a_{n-1}]}(y)$, then $|x| - a_{n-1} \leq_{(**)} \psi_n(x)$. In addition, for $|x| \geq a_n$ (otherwise the first and the second derivative are equal to zero),

$$\psi_n'(x) = \text{Sgn}(x) \int_0^{|x|} \phi_n(z)dz \;\; with \;\; |\psi_n'(x)| \leq_{(***)} 1_{[0,a_{n-1}]}(|x|) \;\; \&$$

$$\psi_n''(x) = \phi_n(|x|) \in \left[0, \frac{2}{n|x|}\right].$$

Applying Ito calculus to $\psi_n(\Delta_s)$, with $\Delta_s = \overline{\nu}_s - \nu_s$, we obtain

$$\begin{aligned} \psi_n(\Delta_s) = & \int_0^s \psi_n'(\Delta_u)d\Delta_u + \frac{1}{2}\int_0^s \psi_n''(\Delta_u)d\left\langle \Delta \right\rangle_u \\ = & -\kappa \int_0^s \psi_n'(\Delta_u)\Delta_u du + L_s + \frac{1}{2}\int_0^s \psi_n''(\Delta_u)d\left\langle \Delta \right\rangle_u, \end{aligned}$$

where $L_s = \int_0^s \psi_n'(\Delta_u)d(M_u + N_u)$ is a square integrable martingale, because of the inequality $(***)$ and that both M and N are two square integrable martingales (we refer the reader to [21]) with

$$M_s = \eta \int_0^s \sqrt{\nu_u} d(\overline{B}_u - B_u), \quad N_s = \eta \int_0^s (\sqrt{\overline{\nu}_u} - \sqrt{\nu_u}) d\overline{B}_u.$$

Employing Doob's L^2-inequality on $L_s^* = \sup_{0\leq u\leq s} L_s$ and $(***)$ provide

$$\begin{aligned} E\left((L_s^*)^2\right) &\leq 4E\left(L_s^2\right) = 4E\left(\int_0^s (\psi_n')^2(\Delta_u)d\left\langle M+N\right\rangle_u\right) \\ &\leq 4E\left(\int_0^s 1_{[0,a_{n-1}]}(|\Delta_u|)d\left\langle M+N\right\rangle_u\right). \end{aligned} \tag{4.21}$$

Besides

$$\begin{aligned} d\left\langle \Delta, \Delta\right\rangle_u = d\left\langle M+N, M+N\right\rangle_u &\leq 2d\left\langle M, M\right\rangle_u + 2d\left\langle N, N\right\rangle_u \\ &= 2\eta^2\left(\nu_u d\left\langle \overline{B} - B\right\rangle_u + (\sqrt{\overline{\nu}_u} - \sqrt{\nu_u})^2 du\right). \end{aligned} \tag{4.22}$$

Using both inequalities $(**)$ and $(***)$

$$|\Delta_s| \leq a_{n-1} + \kappa \int_0^s |\Delta_u| du + L_s^* + \frac{1}{2} \int_0^s \psi_n''(\Delta_u) d \langle M+N, M+N \rangle_u .$$

Denoting the supremum $X_s = \sup_{0 \leq u \leq s} |\Delta_u|$ and using the inequality $(a+b+c+d)^2 \leq 4(a^2+b^2+c^2+d^2)$ we get

$$X_s^2 \leq 4a_{n-1}^2 + 4 \left(\kappa \int_0^s X_u du \right)^2 + 4(L_s^*)^2 + \left(\int_0^s \psi_n''(\Delta_u) d \langle M+N, M+N \rangle_u \right)^2 ,$$

afterwards, we take the expectation and we use (4.21) and Cauchy-Schwarz inequality for the first integral term $(s \leq T)$

$$\begin{aligned} E[X_s^2] \leq 4a_{n-1}^2 \quad &+ \quad 4T\kappa^2 E\left[\int_0^s X_u^2 du\right] \\ &+ \quad 16E\left[\int_0^s 1_{[0,a_{n-1}]}(|\Delta_u|) d \langle M+N \rangle_u\right] \\ &+ \quad E\left[\left(\int_0^s \psi_n''(\Delta_u) d \langle M+N \rangle_u\right)^2\right] \end{aligned}$$

then (4.22), $(*)$ and $(\sqrt{\overline{\nu}_u} - \sqrt{\nu_u})^2 \leq |\overline{\nu}_u - \nu_u|$ provide

$$\begin{aligned} E[X_s^2] \leq 4a_{n-1}^2 \quad &+ \quad 4\kappa^2 T E\left[\int_0^s X_u^2 du\right] \\ &+ \quad 16\eta^4 E\left[\int_0^s \nu_u d \left\langle \overline{B} - B \right\rangle_u + \int_0^s 1_{[0,a_{n-1}]}(|\Delta_u|) |\Delta_u| du\right] \\ &+ \quad \frac{16\eta^4}{n^2} E\left(s + \int_0^s 1_{[a_n, a_{n-1}]}(|\Delta_u|) \frac{\nu_u}{|\Delta_u|} d \left\langle \overline{B} - B \right\rangle_u\right)^2 , \end{aligned}$$

by continuing the computations, we obtain

$$\begin{aligned} E[X_s^2] \leq 4a_{n-1}^2(1+4s\eta^4) \quad &+ \quad 4\kappa^2 T E\left[\int_0^s X_u^2 du\right] \\ &+ \quad 16\eta^4 E\left[\int_0^s \nu_u d \left\langle \overline{B} - B \right\rangle_u\right] \\ &+ \quad \frac{16\eta^4}{n^2} E\left(s + \int_0^s \frac{\nu_u}{a_n} d \left\langle \overline{B} - B \right\rangle_u\right)^2 . \end{aligned}$$

Let us take a sequence $\overline{B} = B^k$ of Brownian motions that converges a.s. to B, once we apply Gronwall lemma

$$E\left[X_s^2\right] \leq \alpha(n,k) e^{4\kappa^2 T^2} \tag{4.23}$$

with

$$\begin{aligned}\alpha(n,k) &= 4a_{n-1}^2(1+4s\eta^4)+16\eta^4 s/n^2+(16\eta^4/n^2)E\left(\int_0^s \frac{\nu_u}{a_n}d\left\langle B^k-B\right\rangle_u\right)^2 \\ &+ \qquad\qquad 16\eta^4 E\left(\int_0^s \nu_u d\left\langle B^k-B\right\rangle_u\right).\end{aligned}$$

To conclude, we take n such that $4a_{n-1}^2(1+4s\eta^4)+\dfrac{16\eta^4 s}{n^2}<\epsilon/2$ then, for a fixed n, we choose k such that $\dfrac{16\eta^4}{n^2}E\left(\int_0^s \dfrac{\nu_u}{a_n}d\left\langle B^k-B\right\rangle_u\right)^2+16\eta^4 E\left(\int_0^s \nu_u d\left\langle B^k-B\right\rangle_u\right)<\epsilon/2$. The latter fact is possible because ν_u admits moments of all orders (see the reference [21]). Finally, we complete the proof of the continuity by using Fatou lemma on the left side of inequality (4.23).

Step3 : Proving the differentiability of ν with respect to r.

Taking $\epsilon\in]0,y[$, we define $\tau_\epsilon(y)$, $\overline{\tau}_\epsilon(y)$ and $\widetilde{\tau}_\epsilon(y)$ as

$$\tau_\epsilon(y)=\inf\{s>0:\nu_s(y)=\epsilon\},\ \ \overline{\tau}_\epsilon(y)=\inf\{s>0:\overline{\nu}_s(y)=\epsilon\},\ \ \widetilde{\tau}_\epsilon(y)=\tau_\epsilon(y)\wedge\overline{\tau}_\epsilon(y). \quad (4.24)$$

We introduce the stochastic processes $\Delta_s=(\overline{\nu}_s-\nu_s)/(\overline{r}-r)$ and Λ_s that satisfy the following SDEs

$$\begin{aligned}d\Delta_s &= -\kappa\Delta_s ds+\eta\frac{\sqrt{\overline{\nu}_s}-\sqrt{\nu_s}}{\overline{r}-r}d\overline{B}_s+\eta\sqrt{\nu_s}d\left(\frac{\overline{B}_s-B_s}{\overline{r}-r}\right) \\ &= -\kappa\Delta_s ds+\eta\frac{\Delta_s}{\sqrt{\overline{\nu}_s}+\sqrt{\nu_s}}d\overline{B}_s+\eta\sqrt{\nu_s}d\left(\frac{\overline{B}_s-B_s}{\overline{r}-r}\right)\end{aligned}, \quad \Delta_0=0,$$

$$d\Lambda_s=-\kappa\Lambda_s ds+\eta\frac{\Lambda_s}{\sqrt{\overline{\nu}_s}+\sqrt{\nu_s}}d\overline{B}_s,\quad \Lambda_0=1,$$

which provides, by a variation of constants technique $\Delta_{s\wedge\widetilde{\tau}_\epsilon(y)}=C_{s\wedge\widetilde{\tau}_\epsilon(y)}\Lambda_{s\wedge\widetilde{\tau}_\epsilon(y)}$ with

$$\Lambda_{s\wedge\widetilde{\tau}_\epsilon(y)}=\exp\left(-\kappa(s\wedge\widetilde{\tau}_\epsilon(y))+\eta\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)}\frac{d\overline{B}_u}{\sqrt{\overline{\nu}_u}+\sqrt{\nu_u}}-\frac{\eta^2}{2}\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)}\frac{du}{(\sqrt{\overline{\nu}_u}+\sqrt{\nu_u})^2}\right) \quad (4.25)$$

and

$$C_{s\wedge\widetilde{\tau}_\epsilon(y)}=\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)}\frac{\eta\sqrt{\nu_u}}{\Lambda_u}d\left(\frac{\overline{B}_u-B_u}{\overline{r}-r}\right)-\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)}\frac{\eta^2\sqrt{\nu_u}}{\Lambda_u(\sqrt{\overline{\nu}_u}+\sqrt{\nu_u})}d\left\langle\overline{B},\frac{\overline{B}-B}{\overline{r}-r}\right\rangle_u \quad (4.26)$$

where

$$\begin{aligned} \frac{\overline{B}_u - B_u}{\overline{r} - r} &= W_u^1 + \frac{\sqrt{1-\overline{r}^2} - \sqrt{1-r^2}}{\overline{r} - r} W_u^2 \\ and \quad \left\langle \overline{B}, \frac{\overline{B} - B}{\overline{r} - r} \right\rangle_u &= \left(\overline{r} + \sqrt{1-\overline{r}^2} \frac{\sqrt{1-\overline{r}^2} - \sqrt{1-r^2}}{\overline{r} - r} \right) u \end{aligned} \tag{4.27}$$

Now, we are going to take the limit as $\overline{r} \to r$ in equality (4.25). For this task, we use the continuity result established in Step2, the lower bounds $\{\overline{\nu}_u\}_{u<s\wedge\overline{\tau}_\epsilon(y)} \geq \epsilon$, $\{\nu_u\}_{u<s\wedge\tau_\epsilon(y)} \geq \epsilon$ and applying the dominated convergence theorem for the deterministic integral

$$\lim_{\overline{r}\to r} \Lambda_s 1_{s<\widetilde{\tau}_\epsilon(y)} = \exp\left(-\kappa s + \eta \int_0^s \frac{dB_u}{2\sqrt{\nu_u}} - \frac{\eta^2}{2} \int_0^s \frac{du}{4\nu_u} \right) 1_{s<\tau_\epsilon(y)} = \dot{\nu}_s 1_{s<\tau_\epsilon(y)}.$$

In the limit above, the proof of the convergence of the stochastic term comes from the equality $\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)} \frac{d\overline{B}_u}{\sqrt{\overline{\nu}_u} + \sqrt{\nu_u}} = \overline{r} \int_0^{s\wedge\widetilde{\tau}_\epsilon(y)} \frac{dW_u^1}{\sqrt{\overline{\nu}_u} + \sqrt{\nu_u}} + \sqrt{1-\overline{r}^2} \int_0^{s\wedge\widetilde{\tau}_\epsilon(y)} \frac{dW_u^2}{\sqrt{\overline{\nu}_u} + \sqrt{\nu_u}}$ and from the Doob's L^1-maximal inequality for the convergence of each term of this sum.

As for (4.26), let us first prove that the second term vanishes. Indeed, using (4.27), we have

$$\lim_{\overline{r}\to r} \left\langle \overline{B}, \frac{\overline{B} - B}{\overline{r} - r} \right\rangle_u = \lim_{\overline{r}\to r} \left(\overline{r} + \sqrt{1-\overline{r}^2} \frac{\sqrt{1-\overline{r}^2} - \sqrt{1-r^2}}{\overline{r} - r} \right) u = 0. \tag{4.28}$$

Thus

$$\begin{aligned} \lim_{\overline{r}\to r} C_{s\wedge\widetilde{\tau}_\epsilon(y)} = & \lim_{\overline{r}\to r} \int_0^{s\wedge\widetilde{\tau}_\epsilon(y)} \frac{\eta\sqrt{\nu_u}}{\Lambda_u} d\left(\frac{\overline{B}_u - B_u}{\overline{r} - r} \right) \\ = & \eta \lim_{\overline{r}\to r} \left[\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)} \frac{\sqrt{\nu_u}}{\Lambda_u} dW_u^\perp + \left(\frac{r}{\sqrt{1-r^2}} + \frac{\sqrt{1-\overline{r}^2} - \sqrt{1-r^2}}{\overline{r} - r} \right) \int_0^{s\wedge\widetilde{\tau}_\epsilon(y)} \frac{\sqrt{\nu_u}}{\Lambda_u} dW_u^2 \right], \end{aligned}$$

with $W^\perp = \left(W^1 - \frac{r}{\sqrt{1-r^2}} W^2 \right)$. The limit of $\left(\frac{r}{\sqrt{1-r^2}} + \frac{\sqrt{1-\overline{r}^2} - \sqrt{1-r^2}}{\overline{r} - r} \right)$ is null, consequently

$$\lim_{\overline{r}\to r} C_{s\wedge\widetilde{\tau}_\epsilon(y)} = \lim_{\overline{r}\to r} \int_0^{s\wedge\widetilde{\tau}_\epsilon(y)} \frac{\eta\sqrt{\nu_u}}{\Lambda_u} d\left(W_u^1 - \frac{r}{\sqrt{1-r^2}} W_u^2 \right)$$

and thanks to the independence of $\dot{\nu}$ and $W^\perp = W^1 - \frac{r}{\sqrt{1-r^2}} W^2$ (fact that can be seen

directly form (4.14)), we have

$$\lim_{\bar{r}\to r}\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)}\frac{\eta\sqrt{\nu_u}}{\Lambda_u}dW_u^\perp=\frac{1}{\dot{\nu}_{s\wedge\tau_\epsilon(y)}}\lim_{\bar{r}\to r}\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)}\frac{\eta\dot{\nu}_{s\wedge\widetilde{\tau}_\epsilon(y)}\sqrt{\nu_u}}{\Lambda_u}dW_u^\perp$$

Once more, we employ Doob's inequality on $M_T^{\bar{r}}=\sup_{0\le s\le T}\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)}\frac{\dot{\nu}_{s\wedge\widetilde{\tau}_\epsilon(y)}\sqrt{\nu_u}[\dot{\nu}_u-\Lambda_u]}{\dot{\nu}_u\Lambda_u}dW_u^\perp$ and for $\lambda>0$

$$\begin{aligned}P\big(M_T^{\bar{r}}\ge\lambda\big) &\le \frac{1}{\lambda^2}\sup_{0\le s\le T}E\left(\int_0^{s\wedge\widetilde{\tau}_\epsilon(y)}\frac{\dot{\nu}^2_{s\wedge\widetilde{\tau}_\epsilon(y)}\nu_u[\dot{\nu}_u-\Lambda_u]^2}{\dot{\nu}_u^2\Lambda_u^2}du\right)\\ &\le \frac{1}{\lambda^2}E\left(\int_0^{T}\frac{\dot{\nu}^2_{\widetilde{\tau}_\epsilon(y)}\nu_{u\wedge\widetilde{\tau}_\epsilon(y)}[\dot{\nu}_{u\wedge\widetilde{\tau}_\epsilon(y)}-\Lambda_{u\wedge\widetilde{\tau}_\epsilon(y)}]^2}{\dot{\nu}^2_{u\wedge\widetilde{\tau}_\epsilon(y)}\Lambda^2_{u\wedge\widetilde{\tau}_\epsilon(y)}}du\right)\\ &\le \frac{1}{\lambda^2}\int_0^T E\left(\nu_{u\wedge\widetilde{\tau}_\epsilon(y)}\left[\frac{\dot{\nu}_{u\wedge\widetilde{\tau}_\epsilon(y)}}{\Lambda_{u\wedge\widetilde{\tau}_\epsilon(y)}}-1\right]^2\right)du.\end{aligned}$$

Thus, one can choose a sequence r_k that tends to r such that : $\sum_{k\ge1}P(M_T^{r_k}\ge\lambda)<\infty$ and Borel-Cantelli Lemma allows us to conclude for the a.s. convergence

$$\lim_{\bar{r}\to r}C_{s\wedge\widetilde{\tau}_\epsilon(y)}=\eta\int_0^{s\wedge\tau_\epsilon(y)}\frac{\sqrt{\nu_u}}{\dot{\nu}_u}\left(dW_u^1-\frac{r}{\sqrt{1-r^2}}dW_u^2\right).$$

Finally, we have for $s<\tau_0(y)$

$$\partial_r\nu_s=\lim_{\epsilon\to0}\lim_{\bar{r}\to r}\Delta_s=\eta\dot{\nu}_s\int_0^s\frac{\sqrt{\nu_u}}{\dot{\nu}_u}\left(dW_u^1-\frac{r}{\sqrt{1-r^2}}dW_u^2\right).$$

■

4.3 Sensitivity using the infinitesimal generator

The presentation of this part is subdivided into two subparts : In section 4.3.1, we reuse the same operations performed in the introduction (section 4.1) but with stochastic volatility models. We also present the result of the formal computations to show the key tools that allow to extend the proven results obtained in section 4.3.2 for the bidimensional Heston model. Thus the last part of section 4.3.1 can be skipped for a first reading.

4.3.1 A general framework for stochastic volatility models

In this part, we suppose that the real price of the asset vector given by the market has the following multidimensional stochastic volatility dynamic

$$\begin{array}{c} dS_t^i = S_t^i \sqrt{\nu_t^i} dZ_t^i, \quad S_0^i = x_0^i, \\ d\nu_t^i = b_i(t, \nu_t^i)dt + \sigma_i(t, \nu_t^i)d\widetilde{Z}_t^i, \quad \nu_0^i = y_0^i, \end{array} \tag{4.29}$$

where $(Z_t^1, ..., Z_t^d, \widetilde{Z}_t^1, ..., \widetilde{Z}_t^d)$ is a vector of correlated Brownian motions.

Let f be a payoff of a multidimensional European contract on the considered asset vector, the price $F(t, x, y)$ of this contract is given by

$$F(t, x, y) = E\left(f(S_T)\Big| S_t = x, \nu_t = y\right) = E_{t,x,y}\left(f(S_T)\right).$$

Thus $F(t, x, y)$ satisfies the Black & Scholes PDE

$$\begin{array}{c} \dfrac{\partial F}{\partial t}(t, x, y) + \displaystyle\sum_i^d \int_0^T \frac{\partial F}{\partial y_i}(t, x, y) b_i(t, y) + \frac{1}{2} \sum_{i,j=1}^{2d} \frac{\partial^2 F}{\partial z_i \partial z_j}(t, x, y)\Gamma_{ij}(t, x, y) = 0, \\ F(T, x, y) = f(x), \end{array}$$

with $z_i = x_i$ if $i \leq d$ and $z_i = y_{i-d}$ if $i > d$ and $\Gamma(t, x, y)$ has the following expression

$$\Gamma(t, x, y) = \begin{pmatrix} \left(\rho'_{ij} x_i x_j \sqrt{y_i y_j}\right)_{1 \leq i,j \leq d} & \left(\rho_{ij} x_i \sqrt{y_i} \sigma_j(t, y_j)\right)_{1 \leq i,j \leq d} \\ \left(\rho_{ij} x_i \sqrt{y_i} \sigma_j(t, y_j)\right)_{1 \leq i,j \leq d} & \left(\rho''_{ij} \sigma_i(t, y_i)\sigma_j(t, y_j)\right)_{1 \leq i,j \leq d} \end{pmatrix},$$

with R is the $2d \times 2d$ correlation matrix of the vector $(Z_t^1, ..., Z_t^d, \widetilde{Z}_t^1, ..., \widetilde{Z}_t^d)$ of standard Brownian motions

$$R = \begin{pmatrix} \rho' & \rho \\ \rho & \rho'' \end{pmatrix}. \tag{4.30}$$

We suppose now that the misspecified price of the asset vector has the dynamic (4.29) but with $\overline{R} \neq R$ and different volatility of the volatility parameters $\overline{\sigma}_i \neq \sigma_i$, that is to say

$$\begin{array}{c} d\overline{S}_t^i = \overline{S}_t^i \sqrt{\overline{\nu}_t^i} dZ_t^i, \quad \overline{S}_0^i = x_0^i, \\ d\overline{\nu}_t^i = b_i(t, \overline{\nu}_t^i)dt + \overline{\sigma}_i(t, \overline{\nu}_t^i)d\widetilde{Z}_t^i, \quad \overline{\nu}_0^i = y_0^i. \end{array} \tag{4.31}$$

Using formally Ito calculus

$$\begin{aligned} F(T,\overline{S}_T,\overline{\nu}_T) &= F(0,S_0,\nu_0)+\sum_{i=1}^{d}\int_0^T \frac{\partial F}{\partial x_i}(t,\overline{S}_t,\overline{\nu}_t)d\overline{S}^i_t+\int_0^T \frac{\partial F}{\partial t}(t,\overline{S}_t,\overline{\nu}_t)dt \\ &+ \sum_{i=1}^{d}\int_0^T \frac{\partial F}{\partial y_i}(t,\overline{S}_t,\overline{\nu}_t)d\overline{\nu}^i_t+\frac{1}{2}\sum_{i,j=1}^{2d}\int_0^T \frac{\partial^2 F}{\partial z_i\partial z_j}(t,\overline{S}_t,\overline{\nu}_t)\overline{\Gamma}_{ij}(t,\overline{S}_t,\overline{\nu}_t)dt, \end{aligned}$$

where $z_i = x_i$ if $i \leq d$ and $z_i = y_{i-d}$ if $i > d$ and the matrix $\overline{\Gamma}(t,x,y)$ has the following expression

$$\overline{\Gamma}(t,x,y) = \begin{pmatrix} \left(\overline{\rho}'_{ij}x_ix_j\sqrt{y_iy_j}\right)_{1\leq i,j\leq d} & \left(\overline{\rho}_{ij}x_i\sqrt{y_i}\overline{\sigma}_j(t,y_j)\right)_{1\leq i,j\leq d} \\ \left(\overline{\rho}_{ij}x_i\sqrt{y_i}\overline{\sigma}_j(t,y_j)\right)_{1\leq i,j\leq d} & \left(\overline{\rho}''_{ij}\overline{\sigma}_i(t,y_i)\overline{\sigma}_j(t,y_j)\right)_{1\leq i,j\leq d} \end{pmatrix}, \tag{4.32}$$

and $\overline{R}$ is the $2d\times 2d$ correlation matrix of the vector $(Z^1_t,...,Z^d_t,\widetilde{Z}^1_t,...,\widetilde{Z}^d_t)$ of standard Brownian motions

$$\overline{R} = \begin{pmatrix} \overline{\rho}' & \overline{\rho} \\ \overline{\rho} & \overline{\rho}'' \end{pmatrix}. \tag{4.33}$$

Taking the expectation of the previous equality and using the localization for the local martingale term

$$\begin{aligned} E(F(T,\overline{S}_T,\overline{\nu}_T)) &= F(0,S_0,\nu_0)+\frac{1}{2}E\left\{\sum_{i,j=1}^{2d}\int_0^T \frac{\partial^2 F}{\partial z_i\partial z_j}(t,\overline{S}_t,\overline{\nu}_t)\overline{\Gamma}_{ij}(t,\overline{S}_t,\overline{\nu}_t)dt\right\} \\ &+ E\left\{\int_0^T \frac{\partial F}{\partial t}(t,\overline{S}_t,\overline{\nu}_t)dt+\sum_{i}^{d}\int_0^T \frac{\partial F}{\partial y_i}(t,\overline{S}_t,\overline{\nu}_t)b_i(t,\overline{\nu}_t)dt\right\}, \end{aligned}$$

where $z_i = x_i$ if $i \leq d$ and $z_i = y_{i-d}$ if $i > d$. Combining the previous equality with the Black & Scholes PDE we get

$$E(F(T,\overline{S}_T,\overline{\nu}_T)) = F(0,S_0,\nu_0)+\frac{1}{2}E\left\{\int_0^T\sum_{i,j=1}^{2d}\left[(\overline{\Gamma}_{ij}-\Gamma_{ij})\frac{\partial^2 F}{\partial z_i\partial z_j}\right](t,\overline{S}_t,\overline{\nu}_t)dt\right\}.$$

When $\overline{\sigma}_i = \sigma_i$ and the misspecified SDE (4.31) is different from (4.29) only through a

different correlation matrix $\overline{R}$, then the difference $(\overline{\Gamma} - \Gamma)(t, x, y)$ is given by the expression

$$\begin{aligned}(\overline{\Gamma} - \Gamma)(t, x, y) &= Q(t, x, y)\left(\overline{R} - R\right) Q(t, x, y), \quad with \\ Q(t, x, y) &= \begin{pmatrix} \left(\delta_{i-j} x_i \sqrt{y_i}\right)_{1 \leq i,j \leq n} & 0 \\ 0 & \left(\delta_{i-j} \sigma_i(t, y_i)\right)_{1 \leq i,j \leq n} \end{pmatrix}\end{aligned} \tag{4.34}$$

and using the trace operator tr

$$E(F(T, \overline{S}_T, \overline{\nu}_T)) - F(0, S_0, \nu_0) = \frac{1}{2} E\left\{ \int_0^T tr\left[Q\left(\overline{R} - R\right) Q \partial^2 F\right](t, \overline{S}_t, \overline{\nu}_t) dt \right\}. \tag{4.35}$$

We give now the result of the formal computation of the matrix $\frac{\partial^2 F}{\partial z_i \partial z_j}(t, x, y)$, with $z_i = x_i$ if $i \leq d$ and $z_i = y_{i-d}$ if $i > d$. An example of the mathematical justifications of the derivatives used and the permutation between the differentiation operator and the expectation depend on the model chosen and can be found in the section 4.3.2 for the bidimensional Heston model. The different terms of the Hessian matrix of the price $\frac{\partial^2 F}{\partial z_i \partial z_j}(t, x, y)$, are given by

$$\partial^2_{x_i, x_j} F(t, x, y) = E_{t,x,y}(\partial^2_{s_i, s_j} f(S_T) \partial_{x_i} S^i_T \partial_{x_j} S^j_T), \tag{4.36}$$

$$\partial^2_{y_i, y_j} F(t, x, y) = E_{t,x,y}(\partial^2_{s_i, s_j} f(S_T) \partial_{y_i} S^i_T \partial_{y_j} S^j_T) + E_{t,x,y}(\partial_{s_i} f(S_T) \partial^2_{y_i} S^i_T \delta_{i-j}), \tag{4.37}$$

$$\partial^2_{x_i, y_j} F(t, x, y) = E_{t,x,y}(\partial^2_{s_i, s_j} f(S_T) \partial_{x_i} S^i_T \partial_{y_j} S^j_T) + E_{t,x,y}(\partial_{s_i} f(S_T) \partial^2_{x_i, y_i} S^i_T \delta_{i-j}) \tag{4.38}$$

with the notation

$$f(s) = f(s_1, s_2, ..., s_d) \ and \ \partial_{s_{i+d}} f(s) = \frac{\partial f(s)}{\partial s_{i+d}} = \frac{\partial f(s)}{\partial s_i} = \partial_{s_i} f(s_1, ..., s_i, ..., s_d). \tag{4.39}$$

If the function f is convex, $M_{ij} = \left\{ E_{t,x,y}(\partial^2_{s_i, s_j} f(S_T) \partial_{z_j} S^j_T \partial_{z_i} S^i_T) \right\}_{i,j}$, with $z_i = x_i$ if $i \leq d$ and $z_i = y_{i-d}$ if $i > d$, is clearly a positive matrix. Consequently, if f is convex, we can rewrite the Hessian matrix of the price as a sum of a positive matrix M and a matrix N such that

$$\frac{\partial^2 F}{\partial z_i \partial z_j}(t, x, y) = M(t, x, y) + N(t, x, y), \ z_i = x_i \ if \ i \leq d \ and \ z_i = y_{i-d} \ if \ i > d$$

with

$$\begin{cases} M_{ij}(t,x,y) = E_{t,x,y}(\partial^2_{s_i,s_j} f(S_T)\partial_{z_i} S_T^i \partial_{z_j} S_T^j), \\ N_{ij}(t,x,y) = (\delta_{i-j-d} + \delta_{i-j} + \delta_{i-j+d}) E_{t,x,y}(\partial_{s_i} f(S_T)\partial^2_{z_i,z_j} S_T^i) \end{cases} \tag{4.40}$$

where δ represents the Kronecker delta.

Let us now focus on models based on the Heston model like the multidimensional Heston model (dimension$>$ 2) and the multidimensional double Heston model. The choice of these models is largely due to the fact that the results established in section 4.3.2 for the bidimensional Heston model can be easily extended to these models. However, the extension to a larger class of models is conceivable but will request other techniques to overcome some theoretical problems. For example, the assumption **(A1)** (in section 4.2) is an important point in the proofs given in sections 4.3.2 and 4.4.

As already mentioned, the correlation structure chosen for the bidimensional Heston model does not include all the configurations. The extension models considered here will have the same kind of correlation structure used for the bidimensional Heston model in (4.5), (4.6), (4.7) and (4.8), that is to say, we correlate each pair of stocks (S_T^i, S_T^j) independently by a coefficient ρ_{ij} and we propagate this correlation on the volatilities (ν_T^i, ν_T^j) thanks to ρ_i and ρ_j which are known from the one-dimensional calibration.

The idea here is first to check that $trace\left[(Q(t,S_t,\nu_t)\Delta R Q(t,S_t,\nu_t))\, N\right] = 0$. A sufficient condition is to have a matrix ΔR orthogonal to the matrix $O_{ij} = (\delta_{i-j-d} + \delta_{i-j} + \delta_{i-j+d})_{1\le i,j\le 2d}$ in the sense of the bilinear symmetric form $\Phi(A,B) = trace(AB)$. This condition is fulfilled by all symmetric matrices that have zeros on the diagonal of the four blocks

$$\Delta R = \begin{pmatrix} 0 & \times & ... & \times & \times & 0 & \times & ... & \times & \times \\ \times & 0 & \times & ... & \times & \times & 0 & \times & ... & \times \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \times & ... & \times & 0 & \times & \times & ... & \times & 0 & \times \\ \times & \times & ... & \times & 0 & \times & \times & ... & \times & 0 \\ 0 & \times & ... & \times & \times & 0 & \times & ... & \times & \times \\ \times & 0 & \times & ... & \times & \times & 0 & \times & ... & \times \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \times & ... & \times & 0 & \times & \times & ... & \times & 0 & \times \\ \times & \times & ... & \times & 0 & \times & \times & ... & \times & 0 \end{pmatrix}. \tag{4.41}$$

Regarding the multi-asset Heston model, as it will be done for the two-dimensional case, if

we fix the correlation between each asset and its volatility we easily obtain a matrix ΔR similar to (4.41). Consequently, if the misspecified asset vector $\overline{S}$ differs only from the market asset S by ρ_{ij}, the difference quotient (4.35) becomes

$$\frac{E(f(\overline{S}_T)) - E(f(S_T))}{\Delta\rho_{ij}} = E\left\{\int_0^T \overline{S}_t^i \overline{S}_t^j \sqrt{\overline{\nu}_t^i \overline{\nu}_t^j} E_{t,\overline{S}_t,\overline{\nu}_t}\Big[\partial^2_{s_i,s_j} f(S_T)\dot{S}_T^i \dot{S}_T^j \alpha_{t,T}^i \alpha_{t,T}^j\Big]dt\right\}, \quad (4.42)$$

where $\alpha_{t,T}^i$ and $\alpha_{t,T}^j$ have similar values as $\alpha_{t,T}^1$ and $\alpha_{t,T}^2$ given later in (4.48).

The same idea can be used for multi-asset models based on the double Heston model, indeed each stock i has the following dynamic

$$\begin{aligned}
dS_t^i &= S_t^i\left(\sqrt{\nu_t^{i1}}dZ_t^{i1} + \sqrt{\nu_t^{i2}}dZ_t^{i2}\right), \quad S_0^i = x_0^i,\\
d\nu_t^{i1} &= \kappa_1^i(\theta_1^i - \nu_t^{i1})dt + \eta_1^i\sqrt{\nu_t^{i1}}d\widetilde{Z}_t^{i1}, \quad \nu_0^{i1} = y_0^{i1},\\
d\nu_t^{i2} &= \kappa_2^i(\theta_2^i - \nu_t^{i2})dt + \eta_2^i\sqrt{\nu_t^{i2}}d\widetilde{Z}_t^{i2}, \quad \nu_0^{i2} = y_0^{i2},
\end{aligned}$$

$$d\left\langle Z^{i1}, Z^{i2}\right\rangle_t = d\left\langle \widetilde{Z}^{i1}, \widetilde{Z}^{i2}\right\rangle_t = 0,\ d\left\langle Z^{i1}, \widetilde{Z}^{i1}\right\rangle_t = \rho_{i1}dt,\ d\left\langle Z^{i2}, \widetilde{Z}^{i2}\right\rangle_t = \rho_{i2}dt.$$

Because of the decorrelation of $(Z^{1i}, \widetilde{Z}^{1i})$ and $(Z^{2i}, \widetilde{Z}^{2i})$, if (ρ_{i1}, ρ_{i2}) are already known using the one-dimensional calibration for each stock i, we obtain a matrix ΔR similar to (4.41) which allows to have a difference quotient analogous to (4.42).

4.3.2 Differentiability of the price and studying some specific cases

We suppose that the misspecified price of the asset vector is also given by (4.5), (4.6), (4.7) and (4.8) but with different inter-asset correlation $\overline{\rho}$, that is to say, the only misspecified parameter is the inter-asset correlation. Thus, the difference $(\overline{\Gamma} - \Gamma)(t, x, y)$ is given as in (4.34) with

$$\overline{R} - R = (\overline{\rho} - \rho)\begin{pmatrix} 0 & 1 & 0 & \rho_2 \\ 1 & 0 & \rho_1 & 0 \\ 0 & \rho_1 & 0 & \rho_1\rho_2 \\ \rho_2 & 0 & \rho_1\rho_2 & 0 \end{pmatrix}$$

and

$$Q(t,x,y) = \begin{pmatrix} x_1\sqrt{y_1} & 0 & 0 & 0 \\ 0 & x_2\sqrt{y_2} & 0 & 0 \\ 0 & 0 & \eta_1\sqrt{y_1} & 0 \\ 0 & 0 & 0 & \eta_2\sqrt{y_2} \end{pmatrix}, \quad \begin{matrix} x = (x_1, x_2) \\ y = (y_1, y_2) \end{matrix}.$$

The matrix N given in (4.40) is orthogonal to $\overline{R} - R$ by the trace operator and thus it is also orthogonal to $(\overline{\Gamma} - \Gamma)$. In fact

$$\frac{N(\overline{R} - R)}{\overline{\rho} - \rho} = \begin{pmatrix} 0 & \rho_1 D_1 & 0 & \rho_2\rho_1 D_1 \\ \rho_2 D_2 & 0 & \rho_1\rho_2 D_2 & 0 \\ 0 & D_1' & 0 & \rho_2 D_1' \\ D_2' & 0 & \rho_1 D_2' & 0 \end{pmatrix}$$

with $D_1 = E_{t,x,y}\left(\partial_{s_1} f(S_T)\partial^2_{x_1,y_1} S_T^1\right)$, $D_1' = E_{t,x,y}\left(\partial_{s_1} f(S_T)\left[\partial^2_{x_1,y_1} S_T^1 + \rho_1 \partial^2_{y_1,y_1} S_T^1\right]\right)$, $D_2 = E_{t,x,y}\left(\partial_{s_2} f(S_T)\partial^2_{x_2,y_2} S_T^2\right)$ and $D_2' = E_{t,x,y}\left(\partial_{s_2} f(S_T)\left[(\partial^2_{x_2,y_2} S_T^2 + \rho_2 \partial^2_{y_2,y_2} S_T^2)\right]\right)$.

Consequently, $tr\left[N(\overline{\Gamma} - \Gamma)(t,x,y)\right] = 0$ and, with this model, (4.35) is reduced to

$$E(F(T, \overline{S}_T, \overline{\nu}_T)) - F(0, S_0, \nu_0) = \frac{1}{2} E\left\{\int_0^T tr\left[(\overline{\Gamma} - \Gamma)M\right](t, \overline{S}_t, \overline{\nu}_t)dt\right\}, \tag{4.43}$$

where M given in (4.40).

Although we do get rid of the matrix N, we cannot obtain the uniqueness of ρ from (4.43). Indeed, even though we are happy that only the positive matrix M (positive when the payoff f is convex) remains in (4.43), the trace of the difference $(\overline{\Gamma} - \Gamma)$ is equal to zero which makes difficult the conclusion on the positivity of $E(F(T, \overline{S}_T, \overline{\nu}_T)) - F(0, S_0, \nu_0)$. This is why, in Proposition 4.2, we study only specific cases. The following proposition provides the difference quotient of the price according to ρ, here $\Delta\rho = \overline{\rho} - \rho$.

Proposition 4.1 *We consider the model specified by (4.5), (4.6), (4.7) and (4.8), we make also the assumption **(A1)**. Then, the flow derivatives*

$$\dot{S}_s^i = S_s^i / x_i, \tag{4.44}$$

$$\partial_{y_1} S_s^1 = S_s^1 \gamma_{t,s}^1, \;\; \gamma_{t,s}^1 = \int_t^s \frac{\dot{\nu}_u^1}{2\sqrt{\nu_u^1}} dW_u^1 - \frac{1}{2}\int_t^s \dot{\nu}_u^1 du, \tag{4.45}$$

$$\partial_{y_2} S_s^2 = S_s^2 \gamma_{t,s}^2, \;\; \gamma_{t,s}^2 = \int_t^s \frac{\dot{\nu}_u^2}{2\sqrt{\nu_u^2}} \left(\rho dW_u^1 + \sqrt{1-\rho^2} dW_u^2 \right) - \frac{1}{2} \int_t^s \dot{\nu}_u^2 du. \tag{4.46}$$

where the CIR flow derivative $\dot{\nu}_s^i$ is either given in (4.11) or replaced by its modification that vanishes once the volatility reaches zero.

Using these expressions, the difference quotient (4.35) becomes

$$\frac{E(f(\overline{S}_T)) - E(f(S_T))}{\Delta\rho} = E\left\{ \int_0^T S_t^1 \overline{S}_t^2 \sqrt{\nu_t^1 \overline{\nu}_t^2} E_{t,\overline{S}_t,\overline{\nu}_t} \left[\partial_{s_1,s_2}^2 f(S_T) \dot{S}_T^1 \dot{S}_T^2 \alpha_{t,T}^1 \alpha_{t,T}^2 \right] dt \right\} \tag{4.47}$$

with $\overline{S} = (S^1, \overline{S}^2)$, $\overline{\nu} = (\nu^1, \overline{\nu}^2)$, $\alpha_{t,T}^1$ and $\alpha_{t,T}^2$ provided by the equalities

$$\begin{aligned} \alpha_{t,T}^1 &= 1 + \eta_1 \rho_1 \gamma_{t,T}^1 = 1 + \eta_1 \rho_1 \left(\int_t^T \frac{\dot{\nu}_s^1}{2\sqrt{\nu_s^1}} dW_s^1 - \frac{1}{2} \int_t^T \dot{\nu}_s^1 ds \right), \\ \alpha_{t,T}^2 &= 1 + \eta_2 \rho_2 \gamma_{t,T}^2 = 1 + \eta_2 \rho_2 \left(\int_t^T \frac{\dot{\nu}_s^2}{2\sqrt{\nu_s^2}} \left(\rho dW_s^1 + \sqrt{1-\rho^2} dW_s^2 \right) - \frac{1}{2} \int_t^T \dot{\nu}_s^2 ds \right). \end{aligned} \tag{4.48}$$

Proof of Proposition 4.1:

The difference $(\overline{\Gamma} - \Gamma)(t,x,y)$ is equal to

$$\Delta\rho \sqrt{y_1 y_2} \begin{pmatrix} 0 & x_1 x_2 & 0 & \eta_2 \rho_2 x_1 \\ x_1 x_2 & 0 & \eta_1 \rho_1 x_2 & 0 \\ 0 & \eta_1 \rho_1 x_2 & 0 & \eta_1 \eta_2 \rho_1 \rho_2 \\ \eta_2 \rho_2 x_1 & 0 & \eta_1 \eta_2 \rho_1 \rho_2 & 0 \end{pmatrix}$$

with $x = (x_1, x_2)$ and $y = (y_1, y_2)$. Using this expression of $(\overline{\Gamma} - \Gamma)(t,x,y)$, the expression of M given in (4.40) and the value of the derivatives (4.44), (4.45) and (4.46) we get

$$\frac{tr\left[(\overline{\Gamma} - \Gamma)M\right](t,x,y)}{2\Delta\rho\sqrt{y_1 y_2}} = E_{t,x,y}\left[\partial_{s_1,s_2}^2 f(S_T) S_T^1 S_T^2 \left(1 + \eta_1 \rho_1 \gamma_{t,T}^1\right) \left(1 + \eta_2 \rho_2 \gamma_{t,T}^2\right) \right] \tag{4.49}$$

where the value of γ^1 and γ^2 are given in (4.45) and (4.46).

■

Based on the assumptions **(A1)** (section 4.2) and

$$\textbf{(A2)} \quad |\rho| < 1, \quad |\rho_1| < 1, \quad |\rho_2| < 1,$$

the following theorem gives a sense to the differentiation $\partial^2_{s_1,s_2} f(S_T)$ in (4.47) and it is based on the fact that the system of SDEs (4.7), (4.8), (4.5) and (4.6) driven by $A_1\ {}^t(W^1, \widetilde{W}^1, W^2, \widetilde{W}^2)$ with

$$A_1 = \begin{pmatrix} \rho_1 & \sqrt{1-\rho_1^2} & 0 & 0 \\ \rho\rho_2 & 0 & \rho_2\sqrt{1-\rho^2} & \sqrt{1-\rho_2^2} \\ 1 & 0 & 0 & 0 \\ \rho & 0 & \sqrt{1-\rho^2} & 0 \end{pmatrix},$$

can be rewritten thanks to the Brownian motion vector $(\beta^1, \beta^2, \beta^3, \beta^4)$ by setting the equality $A_1\ {}^t(W^1, \widetilde{W}^1, W^2, \widetilde{W}^2) = C_1\ {}^t(\beta^1, \beta^2, \beta^3, \beta^4)$ with

$$C_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \rho\rho_1\rho_2 & \sqrt{1-\rho^2\rho_1^2\rho_2^2} & 0 & 0 \\ \rho_1 & \frac{\rho\rho_2(1-\rho_1^2)}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & \frac{\sqrt{1-\rho_1^2}\sqrt{1-\rho^2\rho_2^2}}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & 0 \\ \rho\rho_1 & \frac{\rho_2(1-\rho^2\rho_1^2)}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & \frac{\rho(1-\rho_2^2)\sqrt{1-\rho_1^2}}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}\sqrt{1-\rho^2\rho_2^2}} & \frac{\sqrt{1-\rho^2}\sqrt{1-\rho_2^2}}{\sqrt{1-\rho^2\rho_2^2}} \end{pmatrix},$$

this also implies that

$${}^t(W^1, \widetilde{W}^1, W^2, \widetilde{W}^2) = A_1^{-1} C_1\ {}^t(\beta^1, \beta^2, \beta^3, \beta^4). \tag{4.50}$$

Theorem 4.2 *We suppose that the couples asset/volatility $(S_T^i, \nu_T^i)_{i=1,2}$ have the dynamic given by (4.5), (4.6), (4.7) and (4.8). We also assume* ***(A1)***, ***(A2)*** *and $f(s) = \max(a_1 s_1 + a_2 s_2 - K, 0)$ with $a_1, a_2 \in (\mathbb{R}^*)^2$. For a square integrable random variable X, the conditional expectation $E_{t,x,y,\beta_1,\beta_2}\left(\partial^2_{s_1,s_2} f(S_T) S_T^1 S_T^2 X\right) = E_{t,x,y}\left(\partial^2_{s_1,s_2} f(S_T) S_T^1 S_T^2 X \middle| (\beta_1, \beta_2)_{t \le w \le T}\right)$ is equal to the two following values*

$$E_{t,x,y,\beta_1,\beta_2}\left[\frac{a_2}{x_1} S_T^2 \left(\frac{K - a_2 S_T^2}{|a_1|}\right) g_1\left(\frac{K - a_2 S_T^2}{a_1 x_1} \middle| \frac{S_T^2}{x_2}\right) h\left(\frac{K - a_2 S_T^2}{a_1}, S_T^2\right)\right] \tag{4.51}$$

and

$$E_{t,x,y,\beta_1,\beta_2}\left[\frac{a_1}{x_2} S_T^1 \left(\frac{K - a_1 S_T^1}{|a_2|}\right) g_2\left(\frac{K - a_1 S_T^1}{a_2 x_2} \middle| \frac{S_T^1}{x_1}\right) h\left(S_T^1, \frac{K - a_1 S_T^1}{a_2}\right)\right], \tag{4.52}$$

with

$$h(s_1, s_2) = E_{t,x,y,\beta_1,\beta_2}\left(X \middle| S_T^1 = s_1, S_T^2 = s_2\right)$$

and

$$g_1(v_1|v_2) = \frac{\exp\left(-\frac{1}{2(1-\widetilde{\rho}^2)}\left[\frac{u_1(v_1)}{\sigma_1} - \widetilde{\rho}\frac{u_2(v_2)}{\sigma_2}\right]^2\right)}{\sqrt{2\pi}v_1\sigma_1\sqrt{1-\widetilde{\rho}^2}}1_{v_1>0}, \tag{4.53}$$

$$g_2(v_2|v_1) = \frac{\exp\left(-\frac{1}{2(1-\widetilde{\rho}^2)}\left[\frac{u_2(v_2)}{\sigma_2} - \widetilde{\rho}\frac{u_1(v_1)}{\sigma_1}\right]^2\right)}{\sqrt{2\pi}v_2\sigma_2\sqrt{1-\widetilde{\rho}^2}}1_{v_2>0}. \tag{4.54}$$

where

$$\sigma_1 = \sqrt{\tfrac{(1-\rho_1^2)(1-\rho^2\rho_2^2)}{1-\rho^2\rho_1^2\rho_2^2}}\sqrt{\textstyle\int_t^T \nu_s^1 ds}, \quad \sigma_2 = \sqrt{\tfrac{(1-\rho_2^2)(1-\rho^2\rho_1^2)}{1-\rho^2\rho_1^2\rho_2^2}}\sqrt{\textstyle\int_t^T \nu_s^2 ds},$$

$$u_1(v) = \ln(v) + \tfrac{1}{2}\textstyle\int_t^T \nu_s^1 ds - \rho_1 \int_t^T \sqrt{\nu_s^1}d\beta_s^1 - \frac{\rho\rho_2(1-\rho_1^2)}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}}\int_t^T \sqrt{\nu_s^1}d\beta_s^2,$$

$$u_2(v) = \ln(v) + \tfrac{1}{2}\textstyle\int_t^T \nu_s^2 ds - \rho\rho_1 \int_t^T \sqrt{\nu_s^2}d\beta_s^1 - \frac{\rho_2(1-\rho^2\rho_1^2)}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}}\int_t^T \sqrt{\nu_s^2}d\beta_s^2,$$

$$\widetilde{\rho} = \frac{\rho\sqrt{(1-\rho_1^2)}\sqrt{(1-\rho_2^2)}}{\sqrt{(1-\rho^2\rho_1^2)}\sqrt{(1-\rho^2\rho_2^2)}}\frac{\int_t^T \sqrt{\nu_s^1\nu_s^2}ds}{\sigma_1\sigma_2}.$$

The proof of this theorem is provided in the appendix.

Remark 4.1 *1) According to section 4.2 (equality (4.11)), for $i = 1, 2$ and $p \geq 1$, $\dot{\nu}_T^i$ or their modifications (once we reach $\tau_0(y)$, we replace them by their modifications that vanish) are L^p random variables and this is also the case for $\left(\int_t^T \nu_s^i ds\right)^{-1/2}$ thanks to the results developed in [21]. Thus $\alpha_{t,T}^1\alpha_{t,T}^2 \in L^2(\Omega)$ and Theorem 4.2 tells us that the equality (4.49), expressed formally thanks to some elements of the matrix M (see (4.40)), is equal to $E_{t,x,y}(\Lambda_{t,x,y,\beta_1,\beta_2})$ where $\Lambda_{t,x,y,\beta_1,\beta_2}$ is almost surely equal to both (4.51) and (4.52) with*

$$h(s_1, s_2) = E_{t,x,y,\beta_1,\beta_2}\left(\alpha_{t,T}^1\alpha_{t,T}^2\middle|S_T^1 = s_1, S_T^2 = s_2\right),$$

which provides the sense of our previous use of the Dirac distribution without justification for the model specified by (4.5), (4.6), (4.7) and (4.8).

2) The permutations of the differentiation and the expectation, that were done in the previous sections, are justified by the fact that for :
$X \in \left\{1, \sqrt{y_1}\gamma^1_{t,T}, \sqrt{y_2}\gamma^2_{t,T}, \sqrt{y_1 y_2}\gamma^1_{t,T}\gamma^2_{t,T}\right\}$ *either the expression :*
$\frac{a_2}{x_1} S^2_T \left(\frac{K - a_2 S^2_T}{|a_1|}\right) h\left(\frac{K - a_2 S^2_T}{a_1}, S^2_T\right) g_1\left(\frac{K - a_2 S^2_T}{a_1 x_1} \middle| \frac{S^2_T}{x_2}\right)$
or $\frac{a_1}{x_2} S^1_T \left(\frac{K - a_1 S^1_T}{|a_2|}\right) h\left(S^1_T, \frac{K - a_1 S^1_T}{a_2}\right) g_2\left(\frac{K - a_1 S^1_T}{a_2 x_2} \middle| \frac{S^1_T}{x_1}\right)$ *can be dominated according to* $x = (x_1, x_2)$ *and* $y = (y_1, y_2)$ *by an* L^1*-bounded random variable. Indeed, taking for example the first expression, we have first to get rid of* S^2_T *by a change of probability (*S^2_T *is a positive martingale and not only a local martingale, we refer the reader to [35] and [54]), afterwards,* $\left(\frac{K - a_2 S^2_T}{x_1 |a_1|}\right)$ *can be simplified with denominator of* g_1*, finally,* h *can be easily dominated using the previous remark.*

3) The assumption ***(A2)*** *is necessary to have the two expressions (4.51) and (4.52). Indeed, for instance if* $|\rho_1| = 1$*,* $|\rho_2| < 1$ *and* $|\rho| < 1$ *then the expression (4.52) still can be used but (4.51) cannot.*

4) Although Theorem 4.2 considers that $f(s) = \max(a_1 s_1 + a_2 s_2 - K, 0)$ *with* $a_1, a_2 \in (\mathbb{R}^*)^2$*, the result for* $f(s) = \max(a_1 s_1 + a_2 s_2 + K, 0)$ *with* $a_1, a_2 \in (\mathbb{R}^*)^2$ *can be easily derived in the same way. When dealing with* $f(s) = \max(a_1 s_1 + a_2 s_2 - K, 0)$*,* a_1 *and* a_2 *can be both positive and, subsequently, the result of Theorem 4.2 can be applied on contracts beyond the spread options.*

Now that we give a sense to all the formal expressions established previously, we provide the monotony result for some values of the products $\{\eta_i \rho_i\}_{i=1,2}$ and $\{\eta_i \sqrt{1 - \rho_i^2}\}_{i=1,2}$.

Proposition 4.2 *We suppose that the couples asset/volatility* $(S^i_T, \nu^i_T)_{i=1,2}$ *have the dynamic given by (4.5), (4.6), (4.7) and (4.8). Assuming* ***(A1)****,* ***(A2)*** *and a European option that has* $f(s) = \max(a_1 s_1 + a_2 s_2 \pm K, 0)$ *as payoff, then the price is differentiable according to* ρ *and if :*

c1) $\{\eta_i \rho_i\}_{i=1,2} = 0$ *or*

c2) $\eta_1 \rho_1 = 0$*,* $\eta_2 \sqrt{1 - \rho_2^2} = 0$ *and* $2\kappa_2 - \eta_2 \rho_2 > 0$ *or*

c3) $\eta_2 \rho_2 = 0$*,* $\eta_1 \sqrt{1 - \rho_1^2} = 0$ *and* $2\kappa_1 - \eta_1 \rho_1 > 0$*,*

then the price is monotonous with respect to ρ*. For these three cases, the price increases with respect to* ρ *if* $a_1 a_2 > 0$ *and decreases if* $a_1 a_2 < 0$*. Moreover, the prices of the one-dimensional calls and puts (*$a_1 a_2 = 0$*) do not depend on* ρ*.*

Remark 4.2 – *This result does not include the case* $\{\eta_i \sqrt{1 - \rho_i^2}\}_{i=1,2} = 0$ *because, as we pointed out previously in Remark 4.1 3), one should have, at least,* $|\rho_1| \neq 1$ *or* $|\rho_2| \neq 1$ *to be able to use (4.51) or (4.52).*

- *Even though these choices are restrictive, in some cases, practitioners can found themselves using this kind of assumptions on the parameters. We refer the reader for example to [23].*
- *Because the price is continuous according to η_i and ρ_i (because ν^i is continuous according to these parameters and the payoff is continuous with respect to ν^i), we can replace the zeros in this proposition by "small values". However, we preferred not to announce this more general result because its proof is heavier and it does not help to clarify all the situations for which we have the monotony.*
- *From a numerical point of view, remark also that the condition **(A1)** : $4\kappa_i\theta_i > \eta_i^2$ is generally sufficient to have $\eta_i \lesssim 2$. Indeed, θ_i represents the long term volatility and it is generally smaller than 0.4, also the mean reversion coefficient κ_i used in applications can be considered smaller than 3. Subsequently, $\eta_i|\rho_i| \lesssim 1$ or $\eta_i\sqrt{1-\rho_i^2} \lesssim 1$ is true.*

Proof of Proposition 4.2:

According to (4.47), the domination remark 4.1.2) of the term under the double integral and the continuity of $(-1,1) \ni r \mapsto \nu_s$ announced in Theorem 4.1 (here $r = \overline{\rho}$), we have

$$\begin{aligned}
\partial_\rho E(f(S_T)) &= \lim_{\overline{\rho}\to\rho} E\left\{\int_0^T \overline{S}_t^1\overline{S}_t^2\sqrt{\overline{\nu}_t^1\overline{\nu}_t^2}E_{t,\overline{S}_t,\overline{\nu}_t}\Big[\partial^2_{s_1,s_2}f(S_T)\dot{S}_T^1\dot{S}_T^2\alpha^1_{t,T}\alpha^2_{t,T}\Big]dt\right\} \\
&= E\left\{\int_0^T \lim_{\overline{\rho}\to\rho} S_t^1\overline{S}_t^2\sqrt{\nu_t^1\overline{\nu}_t^2}E_{t,\overline{S}_t,\overline{\nu}_t}\Big[\partial^2_{s_1,s_2}f(S_T)\dot{S}_T^1\dot{S}_T^2\alpha^1_{t,T}\alpha^2_{t,T}\Big]dt\right\} \qquad (4.55)\\
&= E\left\{\int_0^T S_t^1 S_t^2\sqrt{\nu_t^1\nu_t^2}E_{t,S_t,\nu_t}\Big[\partial^2_{s_1,s_2}f(S_T)\dot{S}_T^1\dot{S}_T^2\alpha^1_{t,T}\alpha^2_{t,T}\Big]dt\right\}.
\end{aligned}$$

This then prove the differentiability of the price according to ρ when only **(A1)** is fulfilled.

Using formally the derivative $\partial^2_{s_1,s_2}f(s) = a_1a_2\varepsilon(a_1s^1 + a_2s^2 \pm K)$ (ε is the Dirac distribution) in (4.47), it is sufficient to prove the positivity of $\alpha^i_{t,T}$. If $\eta_i\rho_i = 0$ then $\alpha^i_{t,T} = 1$ which is sufficient to prove *c1)*. Also if $\eta_1\rho_1 = 0$ and $\eta_2\sqrt{1-\rho_2^2} = 0$ then $\alpha^1_{t,T} = 1$ and thanks to (4.11), $\alpha^2_{t,T} = \dot{\nu}^2_T + \left(\kappa_1 - \frac{\eta_1\rho_1}{2}\right)\int_t^T \dot{\nu}^i_s ds$ and provided that $2\kappa_2 - \eta_2\rho_2 > 0$, $\alpha^2_{t,T} > 0$ which proves *c2)* and the proof of *c3)* is analogous.

■

4.4 Asymptotic approximation for short maturities

In this section, we remain working with the model specified by (4.5), (4.6), (4.7) and (4.8), we will establish, for short maturities, an asymptotic approximation of the derivative of the price

with respect to ρ. For the sake of simplicity, we consider the option that has the following payoff

$$f(s_1, s_2) = (s_1 - s_2)_+.$$

However, the general result for the payoff of the exchange option $f(s_1, s_2) = (a_1 s_1 - a_2 s_2)_+$, with $(a_1, a_2) \in (\mathbb{R}_+^*)^2$, is given in Theorem 4.3 and a numerically good approximation for the spread options is given in (4.68). Provided that we can commute the derivative with respect to ρ and the expectation, and that the expression under the expectation is differentiable with respect to ρ (see the proof of Theorem 4.3, Step2), the derivative of the price with respect to ρ is given by

$$\frac{\partial}{\partial \rho} E\left((S_T^1 - S_T^2)_+\right) = -E\left(\partial_\rho S_T^2 1_{S_T^1 \geq S_T^2}\right), \tag{4.56}$$

where 1 represents the indicator function. Provided that we can differentiate S^2 and ν^2 with respect to the correlation ρ (when the assumption **(A0)** of the section 4.2 is fulfilled)

$$\begin{aligned} \partial_\rho S_T^2 &= S_T^2 \left(\int_0^T \sqrt{\nu_s^2} \left(dW_s^1 - \frac{\rho}{\sqrt{1-\rho^2}} dW_s^2 \right) \right) \\ &+ S_T^2 \left(\int_0^T \frac{\partial_\rho \nu_s^2}{2\sqrt{\nu_s^2}} \left(\rho dW_s^1 + \sqrt{1-\rho^2} dW_s^2 \right) - \frac{1}{2} \int_0^T \partial_\rho \nu_s^2 ds \right). \end{aligned} \tag{4.57}$$

Replacing the value of $\partial_\rho S_T^2$ in (4.56), we get

$$\begin{aligned} \tfrac{\partial}{\partial \rho} E\left((S_T^1 - S_T^2)_+\right) = & E\left(1_{S_T^1 \geq S_T^2} S_T^2 \left(\frac{1}{2} \int_0^T \partial_\rho \nu_s^2 ds \right) \right) \\ & - E\left(1_{S_T^1 \geq S_T^2} S_T^2 \int_0^T \frac{\partial_\rho \nu_s^2}{2\sqrt{\nu_s^2}} \left(\rho dW_s^1 + \sqrt{1-\rho^2} dW_s^2 \right) \right) \\ & - E\left(1_{S_T^1 \geq S_T^2} S_T^2 \left(\int_0^T \sqrt{\nu_s^2} \left(dW_s^1 - \frac{\rho}{\sqrt{1-\rho^2}} dW_s^2 \right) \right) \right). \end{aligned} \tag{4.58}$$

According to various works like the one presented in [35] and [54], we know that S_T^2 is a real positive martingale and not only a local martingale. This allows us to define a new probability measure P^2 whose density is given by $\frac{dP^2}{dP} = \frac{S_T^2}{S_0^2}$. Under this new probability, Z^1 and Z^2 are

two independent Brownian motions related to W^1 and W^2 by

$$dZ_t^1 = dW_t^1 - \rho\sqrt{\nu_t^2}dt,$$

$$dZ_t^2 = dW_t^2 - \sqrt{1-\rho^2}\sqrt{\nu_t^2}dt.$$

Also, under the probability P^2, the value of S^1 and S^2 are given by

$$S_T^1 = x_1 \exp\left(\int_0^T \sqrt{\nu_s^1}dZ_s^1 + \rho\int_0^T \sqrt{\nu_s^1\nu_s^2}ds - \tfrac{1}{2}\int_0^T \nu_s^1 ds\right),$$

$$S_T^2 = x_2 \exp\left(\int_0^T \sqrt{\nu_s^2}\left(\rho dZ_s^1 + \sqrt{1-\rho^2}dZ_s^2\right) + \tfrac{1}{2}\int_0^T \nu_s^2 ds\right).$$

By this change of probability and using (4.58), we obtain

$$\begin{aligned} E\left(\partial_\rho S_T^2 1_{S_T^1 \geq S_T^2}\right) = \; & S_0^2 E^2\left(1_{S_T^1 \geq S_T^2}\int_0^T \sqrt{\nu_s^2}\left(dZ_s^1 - \frac{\rho}{\sqrt{1-\rho^2}}dZ_s^2\right)\right) \\ & + S_0^2 E^2\left(1_{S_T^1 \geq S_T^2}\int_0^T \frac{\partial_\rho \nu_s^2}{2\sqrt{\nu_s^2}}\left(\rho dZ_s^1 + \sqrt{1-\rho^2}dZ_s^2\right)\right). \end{aligned} \tag{4.59}$$

For short maturities and under the assumption

(A3) $\exists C \in \mathbb{R}^* \;\; such\; that \;\; \ln\left(\frac{a_2 x_2}{a_1 x_1}\right) = C\sqrt{T} + o(\sqrt{T}), \;\; (a_1, a_2) \in (\mathbb{R}_+^*)^2,$

we will see in the proof of Theorem 4.3 that the second term of (4.59) can be neglected because it tends to zero with respect to T faster than the first one. Also, in Theorem 4.3, the asymptotic derivative of the price with respect to ρ is established thanks to the following lemma obtained by Ito isometry.

Lemma 4.1 *On $\mathbb{R}^d$, we define a Brownian motion W_t and $[0,\infty) \ni t \mapsto H_t \in L^2(\mathbb{R}^d)$ an adapted random process such that $\lim_{t\downarrow 0} E\left(||H_t - H_0||^2\right) = 0$, where $||\cdot||$ is the Euclidean norm. Then*

$$\lim_{t\to 0}\frac{1}{t}E\left(\left\|\int_0^t H_s \cdot dW_s - H_0 \cdot W_t\right\|^2\right) = 0.$$

Theorem 4.3 *We suppose that the couples asset/volatility $(S_T^i, \nu_T^i)_{i=1,2}$ have the dynamic given by (4.5), (4.6), (4.7) and (4.8). We also make the assumptions **(A0)**, **(A2)** and **(A3)**. For short*

maturities, the derivative with respect to ρ of a European option that has $f(s_1, s_2) = (a_1 s_1 - a_2 s_2)_+$, with $(a_1, a_2) \in (\mathbb{R}_+^*)^2$ as payoff can be asymptotically approximated by

$$\frac{\partial}{\partial \rho} E\left((a_1 S_T^1 - a_2 S_T^2)_+\right) \underset{T \sim 0}{=} -a_2 x_2 \sqrt{\frac{T \nu_0^1 \nu_0^2}{2\pi\lambda}} \exp\left(-\frac{1}{2}\left[\frac{C}{\sqrt{\lambda}}\right]^2\right) + o(\sqrt{T}) \quad (4.60)$$

*with $\lambda = \nu_0^1 + \nu_0^2 - 2\rho\sqrt{\nu_0^1 \nu_0^2}$ and the constant C comes from **(A3)**.*

From Theorem 4.3 and because $a_2 > 0$, it is clear that the price of an exchange option is decreasing according to ρ for short maturities.

Proof of Theorem 4.3:

We divide the proof of this theorem into two steps : In the first step, we detail the computations of (4.60). In the second step, we show that the commutation of the derivative with respect to ρ and the expectation in (4.56) is correct.

Step1 :

The use of the constants a_1 and a_2 is not restrictive because they can be included in the spot prices $S_0^1 = x_1$ and $S_0^2 = x_2$. Defining the triplet of random variables (L_T^1, L_T^2, L_T^3) by

$$\begin{aligned} L_T^1 &= \frac{1}{\sqrt{T}} \int_0^T \sqrt{\nu_t^2} \left(dZ_t^1 - \frac{\rho}{\sqrt{1-\rho^2}} dZ_t^2\right), \quad L_T^2 = \frac{1}{\sqrt{T}} \int_0^T \sqrt{\nu_t^1} dZ_t^1, \\ L_T^3 &= \frac{1}{\sqrt{T}} \int_0^T \sqrt{\nu_t^2} \left(\rho dZ_t^1 - \sqrt{1-\rho^2} dZ_t^2\right) + \frac{\ln\left(S_0^2/S_0^1\right)}{\sqrt{T}} - \frac{1}{\sqrt{T}} \int_0^T \left(\rho\sqrt{\nu_t^1 \nu_t^2} - \frac{\nu_t^1 + \nu_t^2}{2}\right) dt, \end{aligned} \quad (4.61)$$

as $T \to 0$, Lemma 4.1 and the assumption **(A3)** allows us to have the convergence in probability of (L_T^1, L_T^2, L_T^3) to (L_0^1, L_0^2, L_0^3) with

$$\begin{aligned} L_0^1 &= \sqrt{\nu_0^2} \left(G_1 - \frac{\rho}{\sqrt{1-\rho^2}} G_2\right), \quad L_0^2 = \sqrt{\nu_0^1} G_1 \\ L_0^3 &= \sqrt{\nu_0^2} \left(\rho G_1 - \sqrt{1-\rho^2} G_2\right) + C. \end{aligned}$$

where G_1 and G_2 are two independent standard Normal random variables and C is the constant of the assumption **(A3)**.

Moreover the first term of (4.59) is equal to $S_0^2 \sqrt{T} E^2\left(L_T^1 1_{L_T^2 \geq L_T^3}\right)$ and thanks to both facts

$$P^2(L_0^2 = L_0^3) = 0 \quad \text{and} \quad E^2\left((L_T^1)^2\right) = E^2\left(\frac{1}{T}\int_0^T \nu_t^2 dt\right) < \infty,$$

we obtain the convergence

$$E^2\left(L_T^1 1_{L_T^2 \geq L_T^3}\right) \underset{T \to 0}{\longrightarrow} E^2\left(L_0^1 1_{L_0^2 \geq L_0^3}\right).$$

Let us compute $E^2\left(L_0^1 1_{L_0^2 \geq L_0^3}\right)$,

$$E^2\left(L_0^1 1_{L_0^2 \geq L_0^3}\right) = \sqrt{\nu_0^2} E^2\left(1_A\left(G_1 - \frac{\rho}{\sqrt{1-\rho^2}} G_2\right)\right)$$

with

$$A = \left\{ G_1 - \frac{\sqrt{\nu_0^2}(\rho G_1 + \sqrt{1-\rho^2} G_2)}{\sqrt{\nu_0^1}} \geq C \right\}.$$

By the decomposition of G_1 into two independent standard Normal random variables $\widetilde{G}$ and $\widehat{G}$:

$$\begin{gathered} G_1 = \rho \widetilde{G} + \sqrt{1-\rho^2}\widehat{G}, \text{ with} \\ \widetilde{G} = \rho G_1 + \sqrt{1-\rho^2} G_2, \quad \widehat{G} = \sqrt{1-\rho^2}\left(G_1 - \frac{\rho}{\sqrt{1-\rho^2}} G_2\right) \end{gathered} \tag{4.62}$$

and A becomes

$$A = \left\{ \widehat{G} \geq g(\widetilde{G}) \right\}, \text{ with } g(u) = \frac{(\sqrt{\nu_0^2} - \rho\sqrt{\nu_0^1})u}{\sqrt{1-\rho^2}\sqrt{\nu_0^1}} + C. \tag{4.63}$$

The computation of this expectation provides

$$\begin{aligned} E^2\left(1_A\left(G_1 - \frac{\rho}{\sqrt{1-\rho^2}} G_2\right)\right) &= E^2\left(E^2\left[1_A \frac{\widehat{G}}{\sqrt{1-\rho^2}} \middle| \widetilde{G}\right]\right) \\ &= \frac{1}{\sqrt{1-\rho^2}} E^2\left(\frac{1}{\sqrt{2\pi}} \int_{g(\widetilde{G})}^{\infty} u e^{-u^2/2} du\right) \\ &= \frac{1}{\sqrt{1-\rho^2}\sqrt{2\pi}} E^2\left(\exp\left\{-\frac{[g(\widetilde{G})]^2}{2}\right\}\right). \end{aligned} \tag{4.64}$$

By finishing the calculation of the expectation and multiplying it by $S_0^2\sqrt{T\nu_0^2}$, we obtain the expression given in (4.60).

To conclude that the derivative with respect to ρ is asymptotically given by (4.60), it is sufficient to prove that the second term of (4.59) divided by $\sqrt{T}$ vanishes as T tends to zero. By

Cauchy-Schwarz inequality

$$\frac{1}{\sqrt{T}}E^2\left|1_{S_T^1\geq S_T^2}\int_0^T\frac{\partial_\rho\nu_s^2}{2\sqrt{\nu_s^2}}\left(\rho dW_s^1+\sqrt{1-\rho^2}dW_s^2\right)\right|\leq\sqrt{I_T}\sqrt{E^2\left(1_{S_T^1\geq S_T^2}\right)} \tag{4.65}$$

with $I_T=\frac{1}{T}E^2\left(\int_0^T\frac{(\partial_\rho\nu_s^2)^2}{4\nu_s^2}ds\right)$. Thanks to a conditioning with respect to B^2 and using Ito isometry, we get

$$\begin{aligned}I_T=&\frac{1}{T}E^2\left(\int_0^T\frac{E^2\left[(\partial_\rho\nu_s^2)^2|B^2\right]}{4\nu_s^2}ds\right)\\=&\frac{\eta_2^2\rho_2^2}{4T(1-\rho^2)}\int_0^T\int_0^s E^2\left(e^{-\kappa_2(s-u)-\left(\kappa_2\theta_2-\frac{\eta_2^2}{4}\right)\int_u^s\frac{dr}{\nu_r^2}}\right)duds\leq\frac{\eta_2^2\rho_2^2T}{8(1-\rho^2)}.\end{aligned} \tag{4.66}$$

Step2 : The commutation of the derivative with respect to ρ and the expectation in (4.56) remains to be proven. When taking $|\rho|<1-\epsilon$ with $0<\epsilon\ll 1$, we can dominate the square of the random variables in the expectations E^2 of (4.59) by integrable random variables. The latter fact can be easily seen for the first term and regarding the second term, one should use the inequalities (4.65) and (4.66) to obtain it.

■

Remark 4.3 *1) First, we point out that assumption **(A0)** is necessary to have the differentiability of ν^2 according to ρ in the strong sense which was needed in Theorem 4.3. However, it is sufficient to have $4\kappa_2\theta_2>\eta_2^2$ to use the boundedness of (4.66) in Step1 and Step2 of the previous proof. Also because of the differentiability of the price according to ρ (see proposition 4.2), we conjecture the validity of the asymptotic approximation when the assumption **(A0)** is replaced by **(A1)**.*

*2) Rewriting the asymptotic approximation (4.60) without the constant C of the assumption **(A3)**, we get*

$$-a_2x_2\sqrt{\frac{T\nu_0^1\nu_0^2}{2\pi\lambda}}\exp\left(-\frac{1}{2}\left[\frac{\ln\left(\frac{a_2x_2}{a_1x_1}\right)}{\sqrt{T\lambda}}\right]^2\right),$$

with $\lambda=\nu_0^1+\nu_0^2-2\rho\sqrt{\nu_0^1\nu_0^2}$. Although this approximation works well for $T\leq 0.2$, we

find out numerically that the expression

$$-a_2x_2\sqrt{\frac{T\nu_0^1\nu_0^2}{2\pi\lambda}}\exp\left(-\frac{1}{2}\left[\frac{\ln\left(\frac{a_2x_2}{a_1x_1}\right)}{\sqrt{T\lambda}}+\frac{\sqrt{T\lambda}}{2}\right]^2\right) \tag{4.67}$$

allows us to have good results even when $T = 0.3$. *The term* $\frac{\sqrt{T\lambda}}{2}$ *comes from the finite variation process* $\frac{1}{\sqrt{T}}\int_0^T\left(\rho\sqrt{\nu_t^1\nu_t^2}-\frac{\nu_t^1+\nu_t^2}{2}\right)dt$ *in the expression of* L_T^3 *in (4.61).*

3) *As we will see in section 4.5.1, the expression (4.67) provides a good estimation of the derivative with respect to* ρ *for exchange options with maturities* $T \leq 0.3$. *For short maturity, using the following approximations*

$$E\left((a_1S_T^1-a_2S_T^2+K)_+\right)\simeq 0.5*E\left((a_1S_T^1-a_2\widetilde{S}_T^2)_+\right)+0.5*E\left((a_1\widetilde{S}_T^1-a_2S_T^2)_+\right)$$
$$with\ \ \widetilde{S}_T^1=(\widetilde{x}_1/x_1)S_T^1,\ \widetilde{S}_T^2=(\widetilde{x}_2/x_2)S_T^2\ \ and\ \ \widetilde{x}^1=x_1+\tfrac{K}{a_1},\ \widetilde{x}^2=x_2-\tfrac{K}{a_2}.$$

and applying (4.67) on these approximations, we obtain another good estimation of the derivative of the spread options with respect to ρ, *given by*

$$\begin{aligned}&\frac{-a_2\widetilde{x}_2}{2}\sqrt{\frac{T\nu_0^1\nu_0^2}{2\pi\lambda}}\exp\left(-\frac{1}{2}\left[\frac{\ln\left(a_2\widetilde{x}_2/a_1x_1\right)}{\sqrt{T\lambda}}+\frac{\sqrt{T\lambda}}{2}\right]^2\right)\\&+\frac{-a_2x_2}{2}\sqrt{\frac{T\nu_0^1\nu_0^2}{2\pi\lambda}}\exp\left(-\frac{1}{2}\left[\frac{\ln\left(a_2x_2/a_1\widetilde{x}_1\right)}{\sqrt{T\lambda}}+\frac{\sqrt{T\lambda}}{2}\right]^2\right),\end{aligned} \tag{4.68}$$

with $\lambda=\nu_0^1+\nu_0^2-2\rho\sqrt{\nu_0^1\nu_0^2}$.

4) *Finally, for short maturities we point out that using models based on Heston like the two-dimensional double Heston model and driving the same computations as the one done in this section, one can also obtain an approximation of the derivative of the price of an exchange option with respect to* ρ.

4.5 Numerical results

From a practitioner's point of view, it is interesting to figure out the interval of maturities for which the approximation (4.68) (or (4.67)) is acceptable and to estimate, thanks to a Monte Carlo simulation, the value of the errors produced by this approximation. In addition to that, because the monotony result is established for some values of η_i, ρ_i and $\sqrt{1-\rho_i^2}$, it is important to show, at least numerically, that the practical values of these parameters ensure the monotony.

When using Monte Carlo, in order to check the monotony of the price according to ρ, one has to decrease significantly the variance of the simulations by using as many trajectories as possible. The latter fact is even more true for the approximation of the derivative with respect to ρ using Monte Carlo. In all the implemented simulations we make sure that the obtained results are, at least, ten times bigger than the error induced by the 95% confidence interval [1]. To reach this high accuracy Monte Carlo simulation in an acceptable execution time, we simulated $M = 2^{22}$ trajectories on an Nvidia 480 GTX GPU (Graphics Processing Unit).

The reader may have noticed that the correlation structure, used in (4.5), (4.6), (4.7) and (4.8), does not allow the model to be affine. Consequently, we cannot use, for instance, the Alfonsi discretization scheme [5] for the Monte Carlo simulations. Nevertheless, for the volatilities, we implement the Milstein scheme because it is known to provide good results. Indeed, as already mentioned in [23], when the assumption $4\kappa_2\theta_2 \geq \eta_2^2$ is fulfilled, by setting

$$\nu_{t_{k+1}} = \left(\sqrt{\nu_{t_k}} + \frac{\eta}{2}\sqrt{\Delta t}G\right)^2 + \kappa(\theta - \nu_{t_k})\Delta t - \frac{\eta^2}{4}\Delta t, \quad \Delta t = t_{k+1} - t_k, \quad G \sim \mathcal{N}(0,1)$$

then $\nu_{t_{k+1}} > 0$ when $\nu_{t_k} = 0$ which reduces considerably the cases when $\nu_{t_{k+1}} < 0$. If the simulation provides $\nu_{t_{k+1}} < 0$, then it is sufficient to set $\nu_{t_{k+1}} = 0$ (for more details on the choice of discretization schemes, we refer the reader to [25]). Besides, the assets are simulated by an Euler scheme and the discretization time $\delta t = 0.01$. Consequently, in both sections 4.5.1 & 4.5.2, the parameters of the performed simulations fulfill the assumption **(A1)**.

4.5.1 Results for short maturities

This section is exclusively dedicated to testing the asymptotical derivative (4.68) thanks to a Monte Carlo simulation. We will consider spread options with maturities $T = 0.1, 0.2, 0.3$. We take the correlations $\rho_i \in \{-0.85, -0.8, ..., 0.8, 0.85\}$ such that $\Delta\rho = \rho_{i+1} - \rho_i = 0.05$ and we approach the derivative of the price with respect to ρ by the expression

$$\partial_\rho F(\rho_i) = \frac{F(\rho_{i+1}) - F(\rho_i)}{\Delta\rho} \tag{4.69}$$

where $F(\rho_{i+1})$ and $F(\rho_i)$ are the prices obtained by Monte Carlo. The resulted error between (4.68) and (4.69) will be quantified in percentage :

$$Error\ Percentage = 100 * \left|\frac{Expression(4.68) - Expression(4.69)}{Expression(4.69)}\right|. \tag{4.70}$$

1. The difference $F(\rho_{i+1}) - F(\rho_i)$ defined in (4.69) is at least, ten times bigger than the error induced by the 95% confidence interval.

We point out that the assumption $|\rho| < 1$, in Theorem 4.3, plays an important role in the precision of the approximation (4.69). In addition, because simulating $M = 2^{22}$ trajectories with a discretization time step $\delta t = 0.01$ is already time consuming, we have chosen to restrict ourselves to the values $\rho_i \in \{-0.85, -0.8, ..., 0.8, 0.85\}$. Besides, after a large number of simulations, we have decided to present only the most important numerical results related to the precision of the expression (4.68). For example, after a large set of simulations, we concluded that ρ_1 and ρ_2 do not intervene a lot in the accuracy of the approximation (4.68) and we took for all figures $\rho_1 = \rho_2 = -0.5$ that is also a reasonable choice in practice.

We first study the impact of the model parameters on the error. This allows us to derive the "worst" cases for which the error is big. We then examine the error behavior of the approximation (4.68) as a function of the maturity.

The parameters that deteriorate the most the asymptotic approximation

According to Figure 4.1, η_1 and η_2 change barely the error produced by (4.68). In fact, for short maturities, using small values of η_i creates bigger errors when the value of ρ is close to -1, but the average value of errors remains the same.

According to figures 4.2, 4.3 and 4.4, we notice that the precision of (4.68) is altered much more when κ_i is big and when θ_i is very different form ν_0^i. The latter fact can be explained heuristically by the mean reversion characteristic of the Heston model and because (4.68) does not include the action of θ_i which plays quickly an important role when κ_i is big.

The maturities for which the asymptotic approximation can be accepted

Now that we know the model parameters that reduce the most the accuracy of the approximation (4.68), we want to study the action of the payoff parameters a_1, a_2, $S_0^1 = x_1$, $S_0^2 = x_2$ and the strike K on the precision of the approximation (4.68). In figures 4.5, 4.6, 4.7 and 4.8, we have tested an extreme choice of model parameters in order to be pretty sure that the error obtained, more or less, dominates the errors gotten with standard market parameters.

From these figures, when the option is **I**n **T**he **M**oney (ITM) or **O**ut of **T**he **M**oney (OTM), we remark that the error increases quickly when ρ is close to 1. Although a small part [2] of the error is due to the approximation (4.69), the other part tells us that, when $T = 0.3$, $\rho > 0.8$ and the payoff is 20% ITM or OTM, one has to have small values of κ_i ($\kappa_i \leq 1$) or small difference between θ_i and ν_0^i ($\theta_i/\nu_0^i \geq 1/2$), otherwise the approximation (4.68) is strongly

2. When simulating $M = 2^{26}$ trajectories and using $\Delta\rho = 0.005$, we found out that the maximum error attained in Figure 4.5 is 24% instead of 28%.

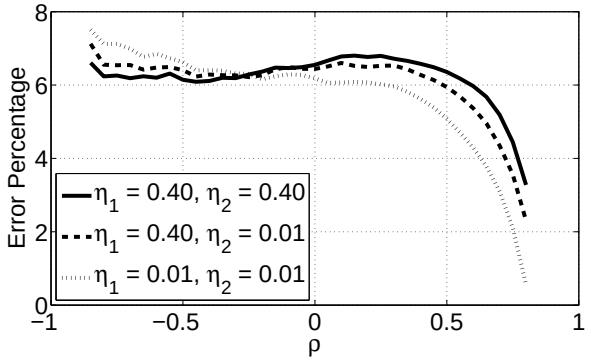

FIGURE 4.1 – The error according to η_1 and η_2, the other parameters used are : $\kappa_1 = \kappa_2 = 2.25$, $\theta_1 = \theta_2 = 0.1$, $\nu_0^1 = \nu_0^2 = 0.5$, $a_1x_1 = a_2x_2 = 100$, the maturity $T = 0.2$ and the strike $K = 0$.

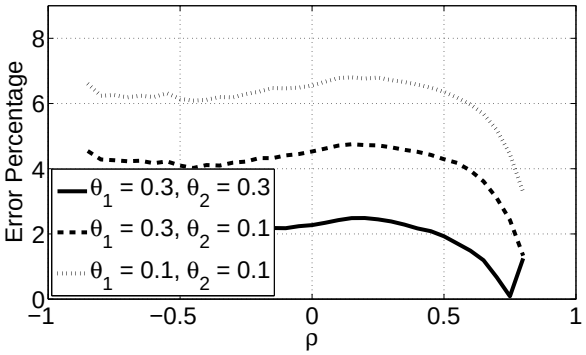

FIGURE 4.2 – The error according to θ_1 and θ_2, the other parameters used are : $\kappa_1 = \kappa_2 = 2.25$, $\eta_1 = \eta_2 = 0.4$, $\nu_0^1 = \nu_0^2 = 0.5$, $a_1x_1 = a_2x_2 = 100$, the maturity $T = 0.2$ and the strike $K = 0$.

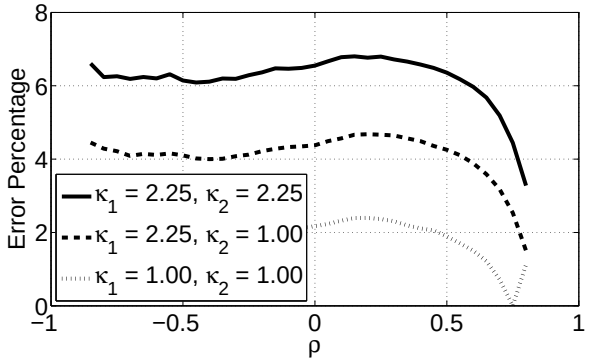

FIGURE 4.3 – The error according to κ_1 and κ_2, the other parameters used are : $\theta_1 = \theta_2 = 0.1$, $\eta_1 = \eta_2 = 0.4$, $\nu_0^1 = \nu_0^2 = 0.5$, $a_1x_1 = a_2x_2 = 100$, the maturity $T = 0.2$ and the strike $K = 0$.

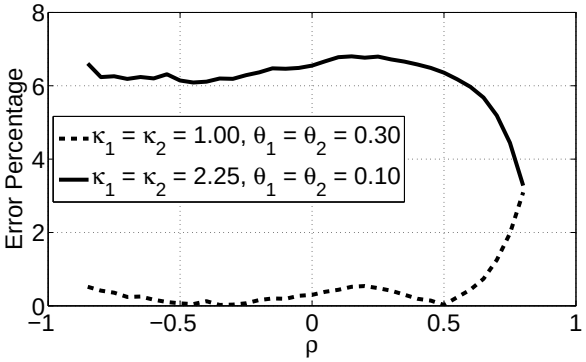

FIGURE 4.4 – The error according to $\{\theta_i\}_{i=1,2}$ and $\{\kappa_i\}_{i=1,2}$, the other parameters used are : $\eta_1 = \eta_2 = 0.4$, $\nu_0^1 = \nu_0^2 = 0.5$, $a_1x_1 = a_2x_2 = 100$, the maturity $T = 0.2$ and the strike $K = 0$.

wrong. When $T = 0.3$, we have found out that the error percentage is always lower than 18% when either $\kappa_i \leq 1.5$ and $\theta_i/\nu_0^i \geq 1/3$ or $\kappa_i \leq 2$ and $\theta_i/\nu_0^i \geq 1/2$. The maximum error percentage associated to all these cases is lower than 18% and the average error is lower than 10%.

To sum up, with $|\rho| \leq 0.9$ and $\nu_0^i \leq 0.5$, when

- $T \leq 0.1$ and the payoff is less than 20% ITM or OTM, the approximation (4.68) can be accepted when $\theta_i/\nu_0^i \geq 1/4$ and $\kappa_i \leq 3$.
- $T \leq 0.2$ and the payoff is less than 20% ITM or OTM, the approximation (4.68) can be accepted when $\theta_i/\nu_0^i \geq 1/4$ and $\kappa_i \leq 1.5$.

– $T \leq 0.3$ and the payoff is less than 10% ITM or OTM, the approximation (4.68) can be accepted when $\theta_i/\nu_0^i \geq 1/5$ and $\kappa_i \leq 3$.
– $\kappa_i \leq 1.5$ and $\theta_i/\nu_0^i \geq 1/3$ or $\kappa_i \leq 2$ and $\theta_i/\nu_0^i \geq 1/2$, the approximation (4.68) can be accepted for maturities $T \leq 0.3$ and payoffs less than 20% ITM or OTM.

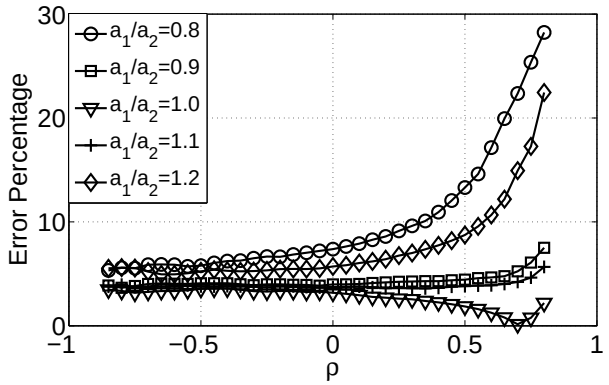

FIGURE 4.5 – The error percentage for a maturity $T = 0.1$ when changing a_1/a_2, the parameters used are : $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.1$, $\eta_1 = \eta_2 = 0.1$, $\nu_0^1 = \nu_0^2 = 0.5$, $x_1 = x_2 = 100$ and the strike $K = 0$.

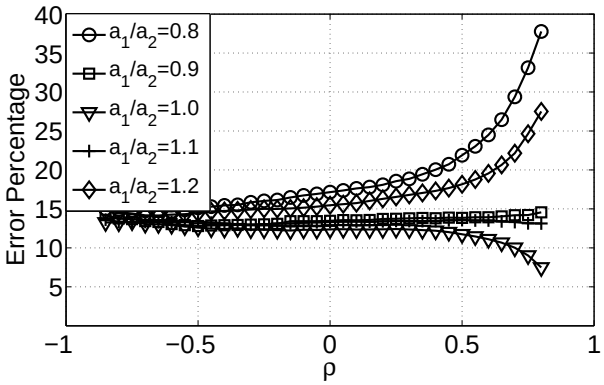

FIGURE 4.6 – The error percentage for a maturity $T = 0.3$ when changing a_1/a_2, the parameters used are : $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.1$, $\eta_1 = \eta_2 = 0.1$, $\nu_0^1 = \nu_0^2 = 0.5$, $x_1 = x_2 = 100$ and the strike $K = 0$.

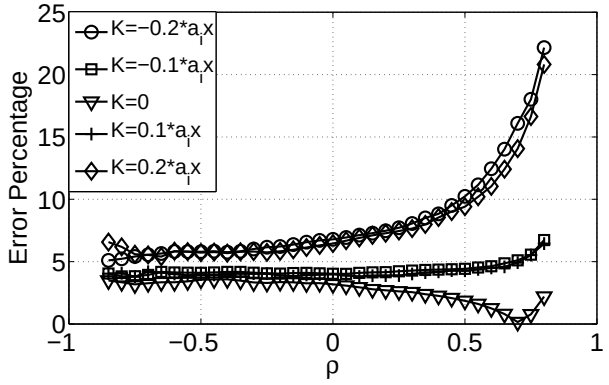

FIGURE 4.7 – The error percentage for a maturity $T = 0.1$ when changing the strike K, the parameters used are : $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.1$, $\eta_1 = \eta_2 = 0.1$, $\nu_0^1 = \nu_0^2 = 0.5$ and $a_1x_1 = a_2x_2 = 100$.

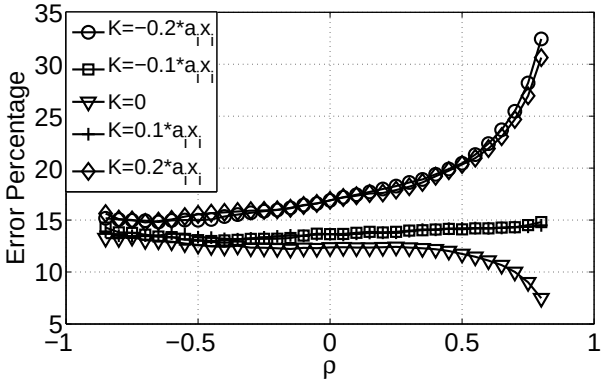

FIGURE 4.8 – The error percentage for a maturity $T = 0.3$ when changing the strike K, the parameters used are : $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.1$, $\eta_1 = \eta_2 = 0.1$, $\nu_0^1 = \nu_0^2 = 0.5$ and $a_1x_1 = a_2x_2 = 100$.

In Figure 4.9, we give an example of a standard choice of parameters when η_1 and η_2 do not fulfill the Feller assumption, but we remark that we still obtain good numerical results.

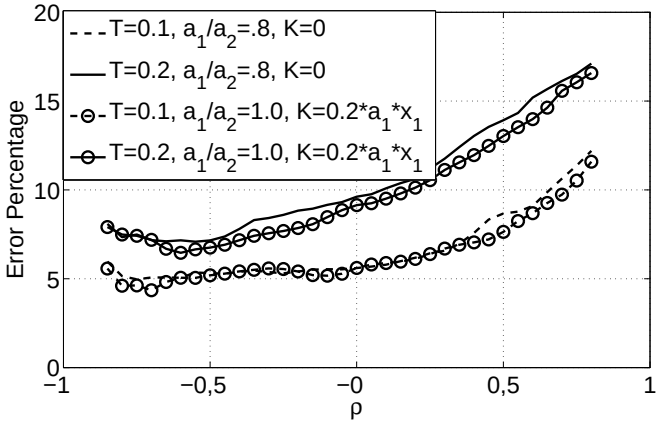

FIGURE 4.9 – The error percentage for 20% ITM or OTM contracts, the parameters used are : $\kappa_1 = \kappa_2 = 2.0$, $\theta_1 = \theta_2 = 0.2$, $\eta_1 = \eta_2 = 1.2$, $\nu_0^1 = \nu_0^2 = 0.4$ and $x_1 = x_2 = 100$.

4.5.2 Results for medium and large maturities

We have already seen, in section 4.3.2, that the monotony of the price according to ρ is fulfilled when η_i, ρ_i or $\sqrt{1-\rho_i^2}$ are sufficiently small. As far as ρ_i and $\sqrt{1-\rho_i^2}$ are concerned in our successive simulations, changing the value of ρ_1 and ρ_2 did not change much numerically the rate of the monotony of the price according to ρ. Consequently, we took for all figures $\rho_1 = \rho_2 = -0.5$. Nevertheless, comparing Figure 4.10 to Figure 4.11 and Figure 4.12 to Figure 4.13, we notice that the monotony is much stronger for small values of η_i than when η_i is close to $2\sqrt{\kappa_i\theta_i}$. What we call "Relative Increment %" in these figures is the quantity defined by

$$100 * \frac{F(\rho_i) - F(\rho_{i+1})}{F(\rho_i)}, \tag{4.71}$$

where $\rho_i \in \{-0.9, -0.8, ..., 0.8, 0.9\}$ and $F(\rho_i)$ is the price obtained by Monte Carlo.

We have preferred to simulate the value of (4.71), instead of the price or its derivative, for two reasons. The first one is due to the heaviness of the simulation of the derivative of the price. In fact, for $T \geq 5$, to have a good Monte Carlo approximation of the derivative of the price according to ρ, one should simulate $M = 2^{24}$ trajectories and preferably use $\Delta\rho = 0.05$ instead of 0.1.

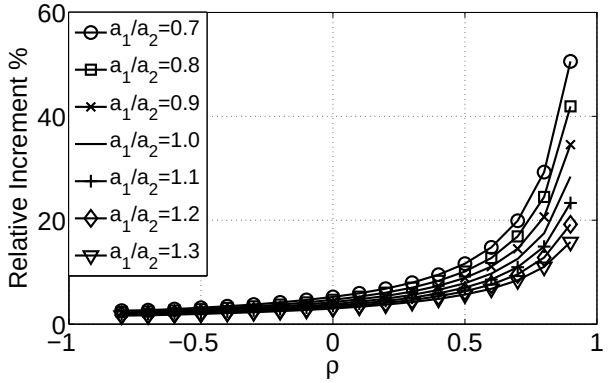

FIGURE 4.10 – The relative increment % for a maturity $T = 5$ when changing a_1/a_2, the parameters used are : $\eta_1 = \eta_2 = 0.1$, $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.2$, $\nu_0^1 = \nu_0^2 = 0.4$, $x_1 = x_2 = 100$ and the strike $K = 0$.

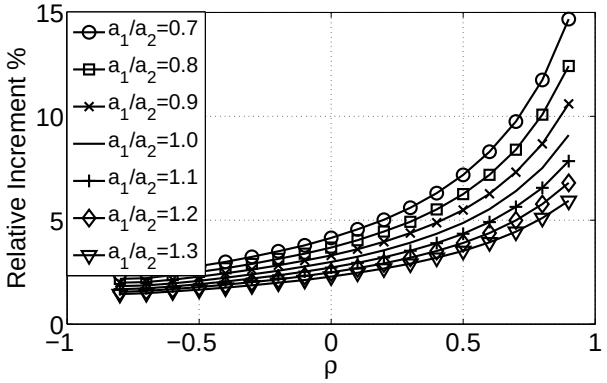

FIGURE 4.11 – The relative increment % for a maturity $T = 5$ when changing a_1/a_2, the parameters used are : $\eta_1 = \eta_2 = 1.5$, $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.2$, $\nu_0^1 = \nu_0^2 = 0.4$, $x_1 = x_2 = 100$ and the strike $K = 0$.

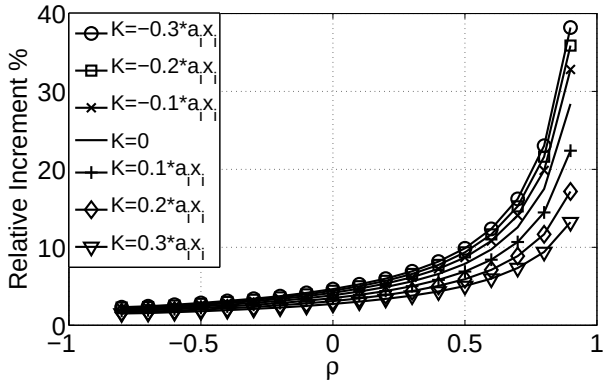

FIGURE 4.12 – The relative increment % for a maturity $T = 5$ when changing the strike K, the parameters used are : $\eta_1 = \eta_2 = 0.1$, $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.2$, $\nu_0^1 = \nu_0^2 = 0.4$ and $a_1x_1 = a_2x_2 = 100$.

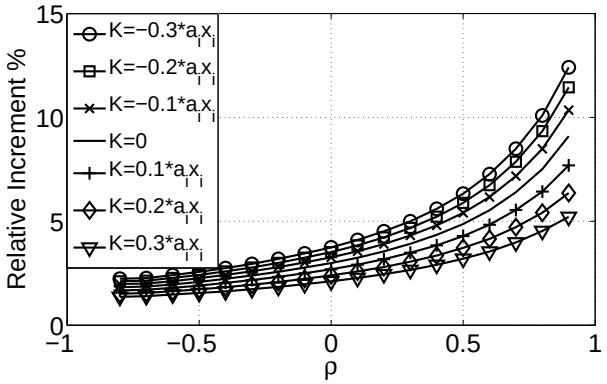

FIGURE 4.13 – The relative increment % for a maturity $T = 5$ when changing the strike K, the parameters used are : $\eta_1 = \eta_2 = 1.5$, $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.2$, $\nu_0^1 = \nu_0^2 = 0.4$ and $a_1x_1 = a_2x_2 = 100$.

In addition to a maturity $T \geq 5$ and a discretization $\delta t = 0.01$, the simulations would take an enormous time even on a GPU. The second reason comes from the fact that the monotony of the price when $\rho > 0.5$ is much bigger than for the other values of ρ. This behavior makes the curves almost flat when $\rho \leq -0.5$ which deteriorates the monotony information.

Because of what we said above, in figures 4.14, 4.15, 4.16 and 4.17, we restrict ourselves to study only the case for which the monotony is the least strong, that is to say, the case when

η_i is close to $2\sqrt{\kappa_i\theta_i}$. We remark that, not only, all the prices are monotonous, but also, the speed of this monotony decreases according to the maturity. Indeed, for maturities $T \geq 10$ and $\rho < -0.5$, the monotony can be barely seen from prices when simulating less than $M = 2^{20}$ trajectories.

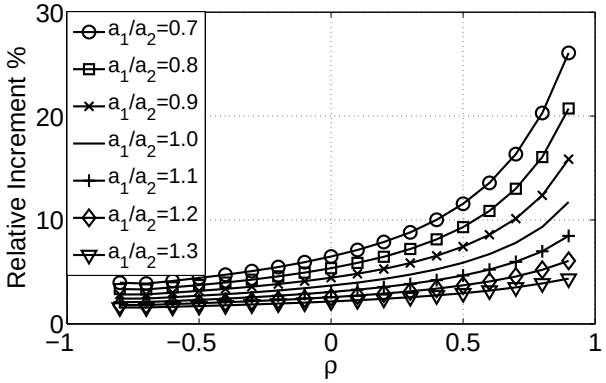

FIGURE 4.14 – The relative increment % for a maturity $T = 1$ when changing a_1/a_2, the parameters used are : $\eta_1 = \eta_2 = 1.5$, $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.2$, $\nu_0^1 = \nu_0^2 = 0.4$, $x_1 = x_2 = 100$ and the strike $K = 0$.

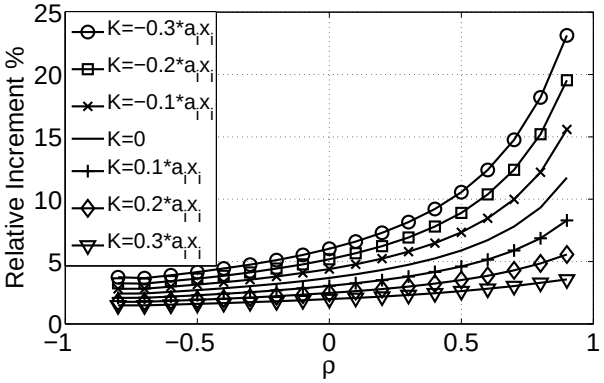

FIGURE 4.15 – The relative increment % for a maturity $T = 1$ when changing the strike K, the parameters used are : $\eta_1 = \eta_2 = 1.5$, $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.2$, $\nu_0^1 = \nu_0^2 = 0.4$ and $a_1x_1 = a_2x_2 = 100$.

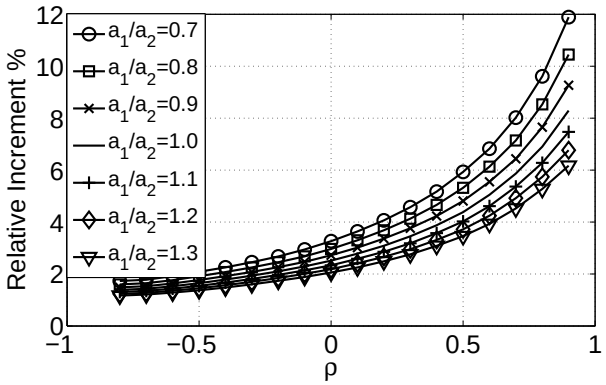

FIGURE 4.16 – The relative increment % for a maturity $T = 10$ when changing a_1/a_2, the parameters used are : $\eta_1 = \eta_2 = 1.5$, $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.2$, $\nu_0^1 = \nu_0^2 = 0.4$, $x_1 = x_2 = 100$ and the strike $K = 0$.

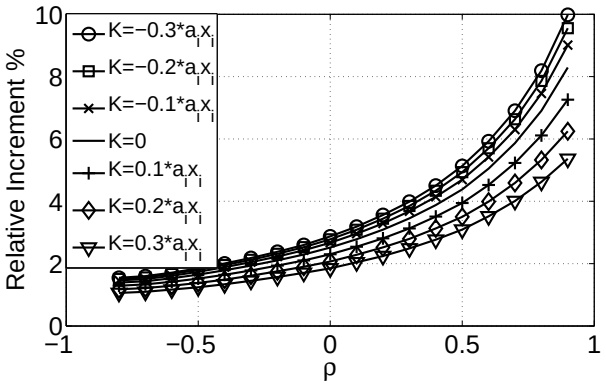

FIGURE 4.17 – The relative increment % for a maturity $T = 10$ when changing the strike K, the parameters used are : $\eta_1 = \eta_2 = 1.5$, $\kappa_1 = \kappa_2 = 3.0$, $\theta_1 = \theta_2 = 0.2$, $\nu_0^1 = \nu_0^2 = 0.4$ and $a_1x_1 = a_2x_2 = 100$.

We conclude that, even though the conditions of Proposition 4.2 can be considered as restrictive, the simulation results strengthen our faith in the global monotony result of the multidimensional Heston model.

4.6 Conclusion

In this work, we tried to present, as consistent as possible, the study of the price according to the correlation. We provided a good approximation of the derivative of the price with respect to ρ for short maturities. We also saw theoretically that the monotony is fulfilled for special choices of the parameters of the model. When compared to the simulation results, the theoretical ones are a bit frustrating because we remarked numerically the clear monotony of the price according to ρ. However, only from the proofs, one can identify the important difficulties that one can face when dealing with this kind of problem. In contrast to the simulation heaviness for which the parallel GPU implementation provides serious advantages that allowed us to have solid numerical study of the monotony of the price.

4.7 Appendix

Proof of Theorem 4.2:

All the computations will be done thanks to the fact that the couple $\left(\frac{S_T^1}{x_1}, \frac{S_T^2}{x_2}\right)$ has a log-normal density conditionally to the Brownian vector $(\beta_w^1, \beta_w^2)_{t\leq w\leq T}$ that drives the volatility SDEs. Indeed, this log-normality can be easily proven by rewriting the couple (W^1, W^2) in term of (β^1, β^2) and to (β^3, β^4) as described previously by (4.50) in which

$$A_1^{-1}C_1 = \begin{pmatrix} \rho_1 & \frac{\rho_2\rho(1-\rho_1^2)}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & \frac{\sqrt{1-\rho_1^2}\sqrt{1-\rho^2\rho_2^2}}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & 0 \\ \sqrt{1-\rho_1^2} & \frac{-\rho\rho_1\rho_2\sqrt{1-\rho_1^2}}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & \frac{-\rho_1\sqrt{1-\rho^2\rho_2^2}}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & 0 \\ 0 & \frac{\rho_2\sqrt{1-\rho^2}}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & \frac{-\rho\rho_2^2\sqrt{1-\rho^2}\sqrt{1-\rho_1^2}}{\sqrt{1-\rho^2\rho_2^2}\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & \frac{\sqrt{1-\rho_2^2}}{\sqrt{1-\rho^2\rho_2^2}} \\ 0 & \frac{\sqrt{1-\rho_2^2}}{\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & \frac{-\rho\rho_2\sqrt{1-\rho_1^2}\sqrt{1-\rho_2^2}}{\sqrt{1-\rho^2\rho_2^2}\sqrt{1-\rho^2\rho_1^2\rho_2^2}} & \frac{-\rho_2\sqrt{1-\rho^2}}{\sqrt{1-\rho^2\rho_2^2}} \end{pmatrix}.$$

If we denote by $g(v_1, v_2)$ the log-normal density of $\left(\frac{S_T^1}{x_1}, \frac{S_T^2}{x_2}\right)$ conditionally to $(\beta_w^1, \beta_w^2)_{t\leq w\leq T}$,

then

$$g(v_1,v_2)=\frac{\exp\left(-\frac{1}{2(1-\widetilde{\rho}^2)}\left[\frac{u_1^2(v_1)}{\sigma_1^2}+\frac{u_2^2(v_2)}{\sigma_2^2}-2\widetilde{\rho}\frac{u_1(v_1)u_2(v_2)}{\sigma_1\sigma_2}\right]\right)}{2\pi v_1v_2\sigma_1\sigma_2\sqrt{1-\widetilde{\rho}^2}}1_{v_1>0}1_{v_2>0}.$$

$g_1(v_1|v_2)$ and $g_2(v_2|v_1)$, given in (4.53) and (4.54), are the conditional densities respectively to $\frac{S_T^2}{x_2}=v_2$ and to $\frac{S_T^1}{x_1}=v_1$. Besides, if we denote

$$\Phi_{\beta^1,\beta^2}(s_1,s_2)=s_1s_2E\left(X\middle|S_T^1=s_1,S_T^2=s_2,\{\beta_w^1,\beta_w^2\}_{t\le w\le T}\right)$$

then, setting $v_i=s_i/x_i$, $E_{t,x,y,\beta_1,\beta_2}\left(\partial^2_{s_1,s_2}f(S_T)S_T^1S_T^2X\right)$ is equal to

$$\begin{array}{ll}
& E_{t,x,y}\left(\partial^2_{s_1,s_2}f(S_T)\Phi_{\beta^1,\beta^2}(S_T^1,S_T^2)|\{\beta_w^1,\beta_w^2\}_{t\le w\le T}\right)\\
= & \dfrac{1}{x_1x_2}\displaystyle\int_{\mathbb{R}_+^2}\partial^2_{v_1,v_2}f\left(x_1v_1,x_2v_2\right)\Phi_{\beta^1,\beta^2}\left(x_1v_1,x_2v_2\right)g(v_1,v_2)dv_1dv_2\\
\overset{(*)}{=} & -\dfrac{a_2}{x_1}\displaystyle\int_{\mathbb{R}_+^2}1_{a_1x_1v_1+a_2x_2v_2\ge K}\partial_{v_1}\left[\Phi_{\beta^1,\beta^2}\left(x_1v_1,x_2v_2\right)g(v_1,v_2)\right]dv_1dv_2\\
\overset{(**)}{=} & -\dfrac{a_1}{x_2}\displaystyle\int_{\mathbb{R}_+^2}1_{a_1x_1v_1+a_2x_2v_2\ge K}\partial_{v_2}\left[\Phi_{\beta^1,\beta^2}\left(x_1v_1,x_2v_2\right)g(v_1,v_2)\right]dv_2dv_1.
\end{array}$$

If $a_1>0$, equality $(*)$ provides

$$\begin{array}{ll}
& E_{t,x,y,\beta_1,\beta_2}\left(\partial^2_{s_1,s_2}f(S_T)\Phi_{\beta^1,\beta^2}(S_T^1,S_T^2)\right)\\
= & \dfrac{a_2}{x_1}\displaystyle\int_{\mathbb{R}_+}\Phi_{\beta^1,\beta^2}\left(\frac{K-a_2x_2v_2}{a_1},x_2v_2\right)g\left(\frac{K-a_2x_2v_2}{a_1x_1},v_2\right)dv_2.
\end{array}$$

Denoting $\overline{g}(v_2)=\int_{\mathbb{R}_+}g(v_1,v_2)dv_1$, we obtain

$$\begin{array}{ll}
& E_{t,x,y,\beta_1,\beta_2}\left(\partial^2_{s_1,s_2}f(S_T)\Phi_{\beta^1,\beta^2}(S_T^1,S_T^2)\right)\\
= & \dfrac{a_2}{x_1}\displaystyle\int_{\mathbb{R}_+}\Phi_{\beta^1,\beta^2}\left(\frac{K-a_2x_2v_2}{a_1},x_2v_2\right)g\left(\frac{K-a_2x_2v_2}{a_1x_1},v_2\right)\frac{\overline{g}(v_2)}{\overline{g}(v_2)}dv_2\\
= & E_{t,x,y,\beta_1,\beta_2}\left(E_{t,x,y,\beta_1,\beta_2}\left[\dfrac{a_2}{x_1}\Phi_{\beta^1,\beta^2}\left(\frac{K-a_2S_T^2}{a_1},S_T^2\right)g_1\left(\frac{K-a_2S_T^2}{a_1x_1}\middle|\frac{S_T^2}{x_2}\right)\middle|S_T^2\right]\right)\\
= & E_{t,x,y,\beta_1,\beta_2}\left(\dfrac{a_2}{x_1}S_T^2\left(\frac{K-a_2S_T^2}{a_1}\right)g_1\left(\frac{K-a_2S_T^2}{a_1x_1}\middle|\frac{S_T^2}{x_2}\right)h\left(\frac{K-a_2S_T^2}{a_1},S_T^2\right)\right).
\end{array}$$

If $a_1 < 0$, equality $(*)$ provides

$$
\begin{aligned}
& E_{t,x,y,\beta_1,\beta_2}\left(\partial^2_{s_1,s_2} f(S_T)\Phi_{\beta^1,\beta^2}(S^1_T, S^2_T)\right) \\
= & -\frac{a_2}{x_1}\int_{\mathbb{R}_+} \Phi_{\beta^1,\beta^2}\left(\frac{K-a_2x_2v_2}{a_1}, x_2v_2\right) g\left(\frac{K-a_2x_2v_2}{a_1x_1}, v_2\right) dv_2 \\
= & -\frac{a_2}{x_1}\int_{\mathbb{R}_+} \Phi_{\beta^1,\beta^2}\left(\frac{K-a_2x_2v_2}{a_1}, x_2v_2\right) g\left(\frac{K-a_2x_2v_2}{a_1x_1}, v_2\right) \frac{\overline{g}(v_2)}{\overline{g}(v_2)} dv_2 \\
= & -E_{t,x,y,\beta_1,\beta_2}\left(E_{t,x,y,\beta_1,\beta_2}\left[\frac{a_2}{x_1}\Phi_{\beta^1,\beta^2}\left(\frac{K-a_2S^2_T}{a_1}, S^2_T\right) g_1\left(\frac{K-a_2S^2_T}{a_1x_1}\middle|\frac{S^2_T}{x_2}\right)\middle| S^2_T\right]\right) \\
= & -E_{t,x,y,\beta_1,\beta_2}\left(\frac{a_2}{x_1}S^2_T\left(\frac{K-a_2S^2_T}{a_1}\right) g_1\left(\frac{K-a_2S^2_T}{a_1x_1}\middle|\frac{S^2_T}{x_2}\right) h\left(\frac{K-a_2S^2_T}{a_1}, S^2_T\right)\right).
\end{aligned}
$$

The expression (4.51) comes from a combination of this result with the one obtained when $a_1 > 0$.

In the same fashion, using equality $(**)$ and (4.54), we obtain (4.52).

■

Bibliographie

[1] L. A. Abbas-Turki and B. Lapeyre. American options based on Malliavin calculus and nonparametric variance & bias reduction methods. *To appear in SIAM Journal on Financial Mathematics*.

[2] L. A. Abbas-Turki and B. Lapeyre. American options pricing on multicore graphic cards. *IEEE The Second International Conference on Business Intelligence and Financial Engineering*, July 2009.

[3] L. A. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier. Pricing derivatives on graphics processing units using Monte Carlo simulation. *To appear in Concurrency and Computation : Practice and Experience*.

[4] L. A. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier. High dimensional pricing of exotic European contracts on a GPU cluster, and comparison to a CPU cluster. *Parallel and Distributed Computing in Finance, in IEEE International Parallel & Distributed Processing Symposium*, May 2009.

[5] A. Alfonsi. High order discretization schemes for the CIR process : Application to affine term structure and heston models. *Mathematics of Computation*, 79 :209–237, 2010.

[6] L. Andersen and M. Broadie. A primal-dual simulation algorithm for pricing multidimensional American options. *Management Science*, 50(9) :1222–1234, 2004.

[7] V. Bally, L. Caramellino, and A. Zanette. Pricing American options by Monte Carlo methods using a Malliavin calculus approach. *Monte Carlo Methods and Applications*, 11 :97–133, 2005.

[8] V. Bally and G. Pagès. A quantization algorithm for solving multidimensional discrete-time optimal stopping problems. *Bernoulli*, 9(6) :1003–1049, 2003.

[9] M. Bossy and A. Diop. An efficient discretisation scheme for one dimensional SDEs with a diffusion coefficient function of the form $|x|^\alpha$, $\alpha \in [1/2, 1)$. *Rapport de recherche INRIA*, (5396), July 2007.

[10] B. Bouchard, I. Ekeland, and N. Touzi. On the Malliavin approach to Monte Carlo approximation of conditional expectations. *Finance and Stochastics*, 8(1) :45–71, 2004.

[11] W. S. Brainerd and L. H. Landweber. *Theory of Computation*. John Wiley & Sons Inc, 1974.

[12] M. Broadie and J. Detemple. American option valuation : new bounds, approximations, and a comparison of existing methods securities using simulation. *The Review of Financial Studies*, 9 :1221–1250, 1996.

[13] M. Broadie and O. Kaya. Exact simulation of stochastic volatility and other affine jump diffusion processes. *OPERATIONS RESEARCH*, 54(3) :217–231, 2006.

[14] P. Cheridito, H. M. Soner, N. Touzi, and N. Victoir. Second order backward stochastic differential equations and fully non-linear parabolic PDEs. *Communications in Pure and Applied Mathematics*, 60(7), 2007.

[15] A. R. Choudhury, A. King, S. Kumar, and Y. Sabharwal. Optimizations in financial engineering : The least-squares Monte Carlo method of Longstaff and Schwartz. *IEEE International Parallel & Distributed Processing Symposium*, April 2008.

[16] E. Clément, D. Lamberton, and P. Protter. An analysis of a least squares regression algorithm for American option pricing. *Finance and Stochastics*, 17 :448–471, 2002.

[17] G. Dimitroff, S. Lorenz, and A. Szimayer. A parsimonious multi-asset Heston model : Calibration and derivatives pricing. *Working Paper*, September 2009.

[18] V.-D. Doan, A. Gaikwad, M. Bossy, and F. Baude. "Gridifying" classification-monte carlo algorithm for pricing high-dimensional American options. *High Performance Computational Finance, in IEEE ACM Supercomputing 2008*, November 2008.

[19] V.-D. Doan, A. Gaikwad, M. Bossy, F. Baude, and I. Stokes-Rees. Parallel pricing algorithms for multi-dimensional BermudanAmerican options using Monte Carlo methods. *INRIA Rapport de Recherche*, 6530, 2008.

[20] Z. Drezner and G. O. Wesolowsky. On the computation of the bivariate normal integral. *Journal of Statistical Computation and Simulation*, 35 :101–107, 1989.

[21] D. Dufresne. The integrated square-root process. *Research paper number 90, Center for Actuarial Studies, University of Melbourne*, Novembre 2001.

[22] E. Fournier, J.-M. Lasry, J. Lebuchoux, and P.-L. Lions. Applications of Malliavin calculus to Monte Carlo methods in finance ii. *Finance and Stochastics*, 5 :201–236, 2001.

[23] J. Gatheral. *The Volatility Surface*. John Wiley & Sons, Inc., 2006.

[24] A. Genz. A comparison of methods for numerical computation of multivariate normal probabilities. *Computing Science and Statistics*, 25 :400–405, 1993.

[25] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Applications of Mathematics. Springer, 2003.

[26] P. Glasserman and B. Yu. Number of paths versus number of basis functions in American option pricing. *The Annals of Applied Probability*, 14(4) :2090–2119, 2004.

[27] M. D. Godfrey and D. F. Hendry. The computer as von Neumann planned it. *IEEE Annals of the History of Computing*, 15(1) :11–21, 1993.

[28] H. H. Goldstine. *The Computer : from Pascal to von Neumann*. Princeton University Press, 1972.

[29] M Harris. *Optimizing Parallel Reduction in CUDA*. NVIDIA Developer Technology, 2.3 edition, 2008.

[30] L. Howes and D. Thomas. *GPU Gems 3*, chapter Efficient Random Number Generation and Application Using Cuda. Addison-Wesley, 2007.

[31] The Quantlib Library http ://quantlib.org/index.shtml.

[32] https ://www.rocq.inria.fr/mathfi/Premia/index.html.

[33] A. Ibanez and F. Zapatero. Monte carlo valuation of American options through computation of the optimal exercise frontier. *Journal of Financial and Quantitative Analysis*, 39(2) :239–273, 2004.

[34] N. El Karoui, M. Jeanblanc-Picqué, and S. E. Shreve. Robustness of the Black & Scholes formula. *Mathematical Finance*, 8(2) :93–126, 1998.

[35] M. Keller. Moment explosions and long-term behavior of affine stochastic volatility models. *Mathematical Finance*, 21(1) :73–98, 2011.

[36] D. E. Knuth. *The Art Of Computer Programming*. Addison-Wesley Publishing, 1998.

[37] H. Kunita. *Stochastic flows and stochastic differential equations*, volume 24. CAMBRIDGE UNIVERSITY PRESS, 1990.

[38] D. Lamberton and B. Lapeyre. *Introduction to Stochastic Calculus Applied to Finance*. Chapman & Hall/CRC Financial Mathematics Series, 2007.

[39] B. Lapeyre and E. Temam. Competitive Monte Carlo methods for the pricing of Asian options. *Journal of Computational Finance*, 5(1), 2001.

[40] P. L'Ecuyer. Combined multiple recursive random number generators. *Operations Research*, 44(5), 1996.

[41] P. L'Ecuyer, R. Simard, E. J. Chen, and W. D. Kelton. An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6), 2002.

[42] F. A. Longstaff and E. S. Schwartz. Valuing American options by simulation : A simple least-squares approach. *The Review of Financial Studies*, 14(1) :113–147, 2001.

[43] M. Mascagni and A. Srinivasan. Algorithm 806 : SPRNG : A scalable library for pseudo-random number generation. *ACM Transactions on Mathematical Software*, 28(3), 2001.

[44] M. Matsumoto and T. Nishimura. Mersenne twister : A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1) :3–30, 1998.

[45] M. Matsumoto, M. Saito, H. Haramoto, and T. Nishimura. Pseudorandom number generation : Impossibility and compromise. *Journal of Universal Computer Science*, 12(6), 2006.

[46] N. Moreni. A variance reduction technique for American option pricing. *Physica A : Statistical Mechanics and Its Applications*, 338 :292–295, 2004.

[47] Nvidia. *NVIDIA CUDA C Best Practices Guide*. 2010.

[48] Nvidia. *NVIDIA CUDA C Programming Guide*. 2010.

[49] P. S. Pacheco. *Parallel programming with MPI*. Morgan Kaufmann, 1997.

[50] G. Pagès and B. Wilbertz. GPGPUs in computational finance : Massive parallel computing for American style options. *Preprint*, 2011.

[51] J. A. Picazo. *Monte Carlo and Quasi-Monte Carlo Methods 2000 : Proceedings of a Conference Held at Hong Kong Baptist University*, chapter American Option Pricing : A Classification-Monte Carlo (CMC) Approach, pages 422–433. Springer-Verlag Berlin Heidelberg, 2002.

[52] V. Podlozhnyuk and M. Harris. Monte Carlo option pricing. *Part of CUDA SDK documentation, nVIDIA*, November 2007.

[53] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++ : The Art of Scientific Computing*. Cambridge University Press, second edition, 2002.

[54] S. Del Bano Rollin, A. Ferreiro-Castilla, and F. Utzet. A new look at the Heston characteristic function. *http ://arxiv.org/abs/0902.2154*, 21, 2009.

[55] V. Surkov. Parallel option pricing with Fourier space time-stepping method on graphics processing units. *First Workshop on Parallel and Distributed Computing in Finance, in IEEE International Parallel & Distributed Processing Symposium*, April 2008.

[56] R. P. Tewarson. *Sparse Matrices*. Academic Press, 1973.

[57] J. Tsitsiklis and B. Van Roy. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks*, 12(4) :694–703, 2001.

[58] S. Villeneuve and A. Zanette. Parabolic ADI methods for pricing American options on two stocks. *Mathematics of Operations Research*, 27, 2002.

[59] L. Vostrikova. On regularity properties of Bessel flow. *Stochastics*, 81(5) :431–453, 2009.

[60] D. Williams. *Probability With Martingales*. Cambridge University Press, 1991.

[61] www.nvidia.com/content/cudazone/cuda_sdk/Computational_Finance.html [13 December 2009].