Quelques algorithmes rapides pour la finance quantitative

Guillaume Sall

École Doctorale de Sciences Mathématiques de Paris Centre

Thèse de Doctorat

en vue de l'obtention du grade de

Docteur ès Sciences de l'Université Pierre et Marie Curie

Discipline: Mathématiques Appliquées

présentée par

Guillaume Sall

Quelques Algorithmes Rapides pour la Finance Quantitative

---

dirigée par Gilles Pagès et Olivier Pironneau

Rapportée par

Mike GILES	Université d'Oxford
Denis TALAY	INRIA

Soutenue le 21 Décembre 2017 devant le jury composé de

Youssef ALLAOUI	Global Market Solutions	Encadrant
Julien BERESTYCKI	Université Pierre et Marie Curie	Examinateur
Gilles PAGÈS	Université Pierre et Marie Curie	Directeur de thèse
Olivier PIRONNEAU	Université Pierre et Marie Curie	Co-Directeur de thèse
Denis TALAY	INRIA	Rapporteur

Quelques Algorithmes Rapides pour la Finance Quantitative

Contribution au Calcul du Risque de Contrepartie

Guillaume SALL

Laboratoire de Probabilités et
Modèles Aléatoires
4, Place de Jussieu
75005 Paris
France
Tél. +33 1 44 27 53 19

Global Market Solutions
R&D Center
7, Cité de l'Ameublement
75011 Paris
France
Tél. +33 4 91 37 06 38

Université Pierre et Marie Curie
4, Place de Jussieu
75005 Paris
France
Tél. +33 1 44 27 44 27

Laboratoire Jacques Louis-Lions
4, Place de Jussieu
75005 Paris
France
Tél. +33 1 44 27 42 98

...À ma mère et mes deux frères.
Merci pour votre amour indéfectible.

“La seule chose que je sais, c’est que je ne sais rien.”
“ἕν οἶδα ὅπ οὐδὲν οἶδα”

– Socrate, Philosophe, $(-469, -399)$.

Remerciements

Je tiens particulièrement à exprimer ma gratitude envers mes directeurs de thèse, Gilles Pagès et Olivier Pironneau, pour leur gentillesse et leur disponibilité ainsi que pour les conseils qu'ils m'ont prodigués tout au long de cette thèse. Ce fut un honneur pour moi que Gilles Pagès accepte de diriger cette thèse. Je le remercie pour son temps, le partage de son expérience et son engagement dans ce projet. Je ne remercierai jamais assez Olivier Pironneau pour m'avoir appuyé dans les démarches et sans qui ce projet n'aurait jamais abouti. J'ai énormément appris sur le plan professionnel et sur le plan humain à leurs côtés. Je les remercie pour m'avoir rassuré pendant les moments de doutes que j'ai traversé ces dernières années et pour leur grande pédagogie.

L'histoire de cette thèse a commencé chez Global Market Solutions et je tiens beaucoup à remercier Youssef Allaoui et Laurent Marcoux pour leur confiance. Ils ont toujours été présents pour mes questions et m'ont laissé une grande liberté dans ma recherche pour mes travaux. Je tiens à remercier Dominique Vignaux pour avoir accepté ce partenariat entre l'université et Global Market Solutions ainsi que Nathalie Aldini et Tania Corbay pour les démarches administratives. Je remercie également mes anciens collègues de bureau Pierre Weirich, Brahim Ait Haddou, Karim Tamarzist, Brahim Maatalla, Hervé Galicher, Olivier Lamourelle, Aleksey Larchenko et Yassine Sabbahi pour toutes les discussions autour des mathématiques et de la finance, leurs conseils, échanges et collaborations enrichissantes.

Je suis très honoré que Mike Giles et Denis Talay aient accepté de rapporter ma thèse. Je les remercie pour leur lecture attentive de ce manuscrit et l'intérêt porté à mon travail. Je remercie Julien Berestycki pour avoir accepté de présider le jury de ma soutenance de thèse.

Ce projet collaboratif a fait intervenir plusieurs entreprises. Je tiens à remercier les partenaires de chez Intel, Raphaêl Monten qui nous a très gentiment appuyé tout au long de ce projet et nous a mis à disposition le matériel adéquate et les ressources humaines nécessaires. Mais aussi Laurent Duhem, Ayal Zaks, Dahnken Christopher, Hans Pabst et Wolfgang Rosenberg pour leurs aides dans l'implémentation des différentes solutions. La réalisation d'une solution avec 6 cartes Intel Xeon Phi Coprocesseurs a été permise grâce à la collaboration de 2CRSI et de son équipe de développement: Karim Gasmi, Frédéric Mossman, Adrien Badina ainsi que Claire Chupin. Je remercie Massimiliano Fattica de chez Nvidia, pour son aide dans l'implémentation de la solution en CUDA. Je remercie également Arnaud Renard pour nous avoir laisser travailler sur le super calculateur de la région Champagne-Ardenne situé à l'université de Reims. Je souhaite particulièrement remercier Jacques Portès pour toutes les discussions autour de l'architecture des cartes accélératrices et graphiques, ainsi qu'à Alain Dominguez pour ces passionnantes sessions d'implémentation et d'échanges.

La vie au laboratoire est assez particulière et je tiens à remercier tous mes collègues doctorants et notamment ceux de mon bureau à qui j'ai rendu la vie un peu difficile! C'est-à-dire Nicolas Cagniart, Olivier Graf, Antonin Prunet et Guillaume Levy. Mais également ceux qui ont partagés ma vie au laboratoire Anouk Nicolopoulos, Jean François Abadie, Léo Girardin, Cécile Taing, Lilian Gaudin, Lucile Mégret, la liste étant non-exhaustive ...

Je tiens à remercier tout le personnel des secrétariats des deux laboratoires où j'ai intéragi et notamment Salima Lounici, Malika Larcher et Catherine Drouet au LJLL ainsi que Serena Benassu et Florence Deschamps au LPMA pour leur gentillesse et serviabilité.

Pour finir, une dernière pensée pour mes proches et amis qui ont toujours été présents pour moi. Et particulièrement ma mère Catherine pour sa confiance et son amour, mon petit-frère Thibault pour son soutien indéfectible et qui est aussi un modèle pour moi, mon grand frère Thomas pour avoir toujours cru en mes capacités.

Quelques Algorithmes Rapides pour la Finance Quantitative

Contribution au Calcul du Risque de Contrepartie

Résumé

Dans cette thèse, nous nous intéressons à des noeuds critiques du calcul du risque de contrepartie, la valorisation rapide des produits dérivées et de leurs sensibilités. Nous proposons plusieurs méthodes mathématiques et informatiques pour répondre à cette problématique. Nous contribuons à quatre domaines différents: une extension de la méthode Vibrato et l'application des méthodes multilevel Monte Carlo pour le calcul des grecques à ordre élevé ($n \geq 1$) avec une technique de différentiation automatique. La troisième contribution concerne l'évaluation des produits Américain, ici nous nous servons d'un schéma pararéel pour l'accélération du processus de valorisation et nous faisons également une application pour la résolution d'une équation différentielle stochastique rétrograde. La quatrième contribution est la conception d'un moteur de calcul performant à architecture parallèle.

Mots clés : Risque management, Mesures de risques, Sensibilités, Monte Carlo, Monte Carlo multilevel, Monte Carlo multilevel avec poids, Vibrato, Différentiation automatique, Pararéel, Intégration parallèle en temps, Option Américaine, Option barriére, Option exotique, Valorisation non-linéaire, Monte Carlo avec régression par Moindres carrés, Fonction discontinue, Régularisation, Parallélisation, Calcul sur GPU, Grille de noeuds de calcul.

Some Fast Algorithms for Quantitative Finance

Contribution to the Computation of Counterparty Credit Risk

Abstract

In this thesis, we will focus on the critical node of the computation of counterparty credit risk, the fast evaluation of financial derivatives and their sensitivities. We propose several mathematical and computer-based methods to address this issue. We have contributed to four areas: an extension of the Vibrato method and an application of the weighted multilevel Monte Carlo for the computation of the greeks for high order derivatives ($n \geq 1$) with automatic differentiation. The third contribution concerns the evaluation of American style option, here we use a parareal scheme to speed up the assessing process and we made an application for solving backward stochastic differential equations. The last contribution is the conception of an efficient computation engine for financial derivatives with a parallel architecture.

Keywords : Financial securities, Risk Assessment, Risk Measures, Greeks, Monte Carlo, Multilevel Monte Carlo, Weighted Multilevel Monte Carlo, Vibrato, Automatic Differentiation, Parallel-In-Time Integration, Parareal, American Option, Barrier Option, Exotic Option, Non-linear Pricing, Least Square Monte Carlo, Non-Smooth Function, Regularization, Parallelization, GPU computing, CPUs Cluster.

Contributions Issues de la Thèse

Nous mentionnons ici les contributions issues de la thèse, qui seront détaillées dans leur intégralité dans les chapitres 1, 2, 3 et 4.

- G. Sall. Approximation of a forward backward stochastic differential equation with a parareal method. *In preparation.*
- G. Pagès, G. Sall. Computing sensitivities with weighted multilevel Monte Carlo. *In preparation.*
- G. Pagès, O. Pironneau, G. Sall. 2017. Vibrato and automatic differentiation for high order derivative and sensitivities of financial options. Risk Magazine, *The Journal of Computational Finance.*
- G. Pagès, O. Pironneau, G. Sall. 2016. Second sensitivities in quantitative finance. ESAIM. *Proceedings and Surveys.* Volume 1, pages 1–10.
- G. Pagès, O. Pironneau, G. Sall. 2016. The parareal method for American options. Elsevier. *Comptes Rendus Mathématiques de l'Académie des Sciences.* Volume 354, Issue 11, pages 1132–1138.

Table des Matières

Liste des Figures

Liste des Tableaux

Liste des Algorithmes

Liste des Codes

Liste des Abréviations

AD	**A**utomatic **D**ifferentiation.
API	**A**pplication **P**rogramming **I**nterface.
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit.
AVX	**A**dvanced **V**ector e**X**tension.
BDP	**B**ackward **D**ynamic **P**rogramming.
BLAS	**B**asic **L**inear **A**lgebra **S**ubprograms.
BSDE	**B**ackward **S**tochastic **D**ifferential **E**quation.
BS	**B**lack **S**choles model.
CDS	**C**redit **D**efault **SWAP**.
CEV	**C**onstant **E**lasticity of **V**ariance model.
CPU	**C**entral **P**rocessing **U**nit.
CSA	**C**redit **S**upport **A**nnex.
CSV	**C**omma **S**eparated **V**alues.
CVA	**C**redit **V**aluation **A**djustment.
CollBA	**C**ollateral **V**enefit **A**djustment.
CollCA	**C**ollateral **C**ost **A**djustment.
CollVA	**C**ollateral **V**alue **A**djustment.
DF	**D**iscount **F**actor.
DP	**D**ouble **P**recision.
DVA	**D**ebt **V**alue **A**djustment.
EEPE	**E**ffective **E**xpected **P**ositive **E**xposure.
EE	**E**xpected **E**xposure.
EPE	**E**expected **P**ositive **E**xposure.
FBSDE	**F**orward **B**ackward **S**tochastic **D**ifferential **E**quation.
FD	**F**inite **D**ifference method.
FLOP	**FL**oating **OP**eration per seconde.
FPGA	**F**ield **P**rogrammable **G**ate **A**array.

FRTB	**F**undamental **R**eview of the **T**rading **B**ook.
FVA	**F**unding **V**alue **A**djustment.
GDDR5	**G**raphics **D**ouble **D**ada **R**ate type **5**.
GPU	**G**prahic **P**rocessing **U**nit.
HBA	**H**edging **B**enefit **A**djustment.
HCA	**H**edging **C**ost **A**djustment.
HVA	**H**edging **V**alue **A**djustment.
IM	**I**nitial **M**argin.
IRD	**I**nterest **R**ate **D**erivative.
IRIS	**I**nnovative **R**isk **I**ntegration **S**ystem.
IT	**I**nformation **T**echnologie.
KVA	Capital **V**alue **A**djustment.
LMM	**L**ibor **M**arket **M**odel.
LRM	**L**ikelihood **R**atio **M**ethod.
LRPW	**L**ikelihood **R**atio coupled to **P**ath-**W**ise method.
LSMC	**L**east **S**quare **M**onte **C**arlo.
LVA	**L**iquidity **V**alue **A**djustment.
MC	**M**onte **C**arlo method.
ML2R	**M**ulti**L**evel **R**richardson-**R**omberg extrapolation.
MLMC	**M**ulti**L**evel **M**onte **C**arlo.
MMX	**M**ultiple **M**ath e**X**tension.
MPI	**M**essage **P**assing **I**nterface.
MVA	**M**argin **V**alue **A**djustment.
MtM	**M**ark-to-**M**arket.
ODE	**O**rdinary **D**ifferential **E**quation.
OIS	**O**vernight **I**ndex **SWAP**.
OSS	**O**ne-**S**tep **S**urvival estimator.
OTC	**O**ver-**T**he-**C**ounter.
PCA	**P**rincipal **C**omponent **A**anlysis.
PCI	**P**eripheral **C**omponent **I**nterconnect.
PDE	**P**artial **D**ifferential **E**quation.
PFE	**P**otential **F**uture **E**xposure.
PW	**P**ath-**W**ise method.
P&L	**P**rofit and **L**oss profil.
RIX	**R**isk **I**ntelligence **E**xpression.
RMSE	**R**oot **M**ean **S**quare **E**rror.
SA-CCR	**S**tandardized **A**pproach for **C**ounterparty **C**redit **R**isk.
SDE	**S**tochastic **D**ifferential **E**quation.

SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata.
SIMM	**S**tandard **I**nitial **M**argin **M**odel.
SIMT	**S**ingle **I**nstruction **M**ultiple **T**hreads.
SISD	**S**ingle **I**nstruction **S**ingle **D**ata.
SP	**S**imple **P**recision.
SSE	**S**treaming **SIMD** e**X**tension.
TLPRAO	**T**wo **L**evels **P**ara**R**eal for **A**merican **O**ptions.
TLPR	**T**wo **L**evels **P**ara**R**eal.
UOC	**U**p–**O**ut **C**all barrier option.
VAD	**V**ibrato and **A**utomatic **D**ifferentiation (forward mode).
VM	**V**ariation **M**argin.
VRAD	**V**ibrato and **R**everse **A**utomatic **D**ifferentiation.
VaR	**V**alue-at-**R**isk.
xVA	all **V**aluation **A**djustments.

Avant-Propos

Cette thèse s'effectue dans le cadre d'une collaboration entre l'Université Pierre et Marie Curie[1] (UPMC) et Global Market Solutions (GMS) sous un contrat CIFRE géré par l'Association Nationale de la Recherche et de la Technologie (ANRT). La thèse s'est effectuée au sein du Laboratoire de Probabilités de Modèles Aléatoires (LPMA) sous la direction de Gilles Pagès et d'Olivier Pironneau, tous deux Professeurs d'université, et au sein du pôle Recherche et Développement de Global Market Solutions avec l'encadrement de Youssef Allaoui et Laurent Marcoux respectivement directeur technique et manager du pôle front office.

[1] Prochainement rebaptisée "Faculté de Science de la Sorbonne Université"

Préambule

"Le marché, à son insu, obéit à une loi qui le domine: la loi de la probabilité"
extract from Théorie de la Spéculation, [9].

– L. Bachelier, Mathématicien, (1870, 1946).

Global Market Solutions est une entreprise de consulting qui propose des implémentations et des solutions d'intégrations de progiciel financier en Europe, Asie et Amérique. L'entreprise s'intéresse particulièrement aux problématiques de front office et à l'intégration de modèle de valorisation d'actifs financiers; du management de risque qui inclus les risques de marché, les risque de crédit, et d'analyse de liquidité en temps réel; aux problématiques de back office.

En 2013, Global Market Solutions a été confronté au problème de valorisation de plusieurs dizaines de milliers de produits dérivés pendant les sessions de calculs de nuit (noeud de calcul pour le risque de contrepartie). Ils se sont tournés vers la parallélisation en réponse à ce problème, ce qui inclus des solutions et l'étude de carte graphique et de nouvelle architecture telle que le Xeon Phi Coprocessor. Dans cette thèse, nous étudions de nouveaux algorithmes rapides et performants pour la valorisation d'actifs financiers, produits dérivés et de leurs sensibilités en utilisant des machines parallèles.

Les méthodes de Monte Carlo sont naturellement parallèles à l'exception de leurs usages dans la programmation dynamique pour la valorisation des produits Américains dans l'algorithme de Longstaff Schwartz [124]. Des algorithmes de Monte Carlo rapides et efficaces ont déjà été proposé, indépendamment du parallélisme, comme les méthodes de Monte Carlo multilevel par Giles [61], la méthode de quantification optimale par Pagès et Printems [144], les méthodes de quasi Monte Carlo par Boyle et al. [105] et beaucoup d'autres... Pour l'évaluation des sensibilités ou des grecques, dans [60], Giles propose une méthode efficace et performante appelée Vibrato.

Dans cette thèse, nous contribuons dans trois domaines différents: 1/ Vibrato, 2/ Monte Carlo multilevel avec poids, 3/ Accélération pararéelle. Pour répondre à la problématique posée par Global Market Solutions, nous avons étudié les nouvelles architectures qui sont une extension ou vise à remplacer les architectures usuelles i.e. un PC ou une grille standard de CPUs.

Le chapitre 1 est consacré au calcul du second et des ordre supérieurs de dérivation des produits financiers. Nous avons combinés deux méthodes, Vibrato et la différentiation automatique et nous les comparons avec d'autres méthodes de calcul des sensibilités. Nous montrons que cette combinaison de technique est plus rapide et plus stable que la différentiation automatique pour la dérivée seconde ou l'approche par différences finies. Nous proposons un framework générique de calcul de grecques et présentons plusieurs applications à des produits financiers vanilles et

exotiques. Nous étendons également la différentiation automatique pour le calcul de la dérivée second pour la fonctions de payoffs qui ne sont pas dérivable deux fois.

Dans le chapitre 2, nous étudions l'approximation des sensibilités des produits dérivés financiers avec une méthode multilevel Monte Carlo avec poids. Le calcul des sensibilités utilise une régularisation de la fonction Heaviside pour les payoff financiers. Nous montrons comment calibrer la méthode multilevel pour les grecques en utilisant les paramètres optimaux proposés par Lemaire et Pagès [119] pour des schémas standards et antithétiques. Nous donnons également un framework qui calcule simultanément la valeur de produits financiers et de ses grecques sur une même grille en utilisant les paramètres optimaux. Nous proposons quelques applications sur des produits vanilles et exotiques.

Le chapitre 3 propose une description de la méthode pararéelle pour la valorisation des produits Américains et une section numérique pour évaluer ses performances. Nous étudions uniquement le cas scalaire. La méthode Monte Carlo avec régression par moindres carrés et la décomposition pararéelle en temps avec deux et plusieurs niveaux sont utilisées pour le développement d'une implémentation parallèle efficiente. Nous proposons également un argument de convergence pour l'algorithme à deux niveaux lorsque le pas de temps pour le schéma d'Euler à chaque niveau est approprié. Cet argument de convergence nous servira pour l'analyse du cas à plusieurs niveaux.

Le chapitre 4 est une extension du chapitre précédent. Nous faisons une application de la méthode pararéelle pour l'approximation de la solution d'une équation différentielle stochastique rétrograde (BSDE) pour la valorisation d'un produit financier i.e. Bid-Ask Spread Call option pour les taux d'intérêt. La méthode de Monte Carlo avec régression par moindres carrés est un choix populaire pour l'approximation ce type d'équation. Le schéma de discrétisation a une complexité assez élevée puisqu'il nécessite l'approximation d'au moins deux espérances conditionnelles. L'utilisation des schémas pararéels est donc une bonne méthode pour la résolution des équations différentielles stochastiques rétrogrades.

Le chapitre 5 est dédié au travail réalisé au sein de Global Market Solutions pour la conception d'un moteur de calcul efficient et rapide pour le calcul massif de produits dérivés de taux. Ce travail s'inscrit en réponse aux demandes croissantes d'indicateurs de risques, la valorisation rapide des portefeuilles de produits financiers est devenu un enjeu majeur pour les banques. Le moteur de calcul repose sur une architecture parallèle à mémoire partagée et distribuée. Nous verrons comment profiter de tous les types de granularité de parallélisation en vue de bénéficier de toutes les capacités des différentes architectures. Nous procédons à plusieurs tests numériques pour prouver l'efficacité de l'implémentation.

Dans l'annexe 6, nous présentons les librairies de différentiation automatique qui ont été implémentées dans le cadre de la collaboration avec Global Market Solutions et qui ont servies aux expérimentations faîtes dans les chapitres 1, 2 et 3. Ces librairies ont également été utilisées dans le cadre de test numérique interne et de la faisabilité de l'intégration de telles libraires dans IRIS (progiciel de calcul de risque développé par Global Market Solutions) au sein du département de recherche & développement de l'entreprise partenaire. Nous présentons succinctement les deux librairies implémentées i.e. une version C++ pour le calcul du premier et du second ordre de dérivation ainsi qu'une version CUDA destinée aux cartes graphiques pour le calcul du premier de dérivation. Nous faisons une rapide description de la régularisation utilisée. Quelques applications en finance sont présentées à titre illustratif sur des produits vanilles et exotiques dans plusieurs modèles stochastiques.

Introduction[2]

Global Market Solutions operates as a consulting company that engages implementation and integrates financial software projects in Europe, Asia, and the Americas. The company primarily focuses on front office and pricing model integrations; risk management that includes market risk, credit risk, and liquidity risk analysis, as well as real-time monitoring; back office and straight through processing; and accounting and regulatory monitoring.

In 2013, Global Market Solutions was confronted with the problem of evaluating several thousands of derivatives overnight. They turned to parallel computing for an answer, this includes GPU cards in a PC and new chips like the Xeon Phi Coprocessor. In this thesis, we investigate faster algorithms for pricing assets, derivatives and their sensitivities the so-called greeks, using parallel machines.

The Monte Carlo method is naturally parallel except when dynamic programming is used as in the Longstaff Schwartz [124] for American options. Fast Monte Carlo algorithms have been proposed independently of parallelism such as multilevel Monte Carlo Giles [61], optimal quantization Pagès et Printems [144], quasi Monte Carlo methods by Boyle et al. [105] and many more... For the computation of greeks, in [60], Giles proposed an acceleration which is called Vibrato.

In this thesis, we have contributed to four areas: 1/ Vibrato, 2/ Weighted multilevel, 3/ Parareal acceleration. To answer the challenge of global Market Solutions, we have investigated the new architectures to extend or replace single PC and PC clusters, this is our fourth contribution.

The chapter 1 deals with the computation of second or higher order Greeks of financial securities. It combines two methods, Vibrato and automatic differentiation and compares with other methods. We show that this combined technique is faster and more stable than automatic differentiation of second order derivatives or finite difference approximations. We present a generic framework to compute any Greeks and present several applications on different types of derivatives. We also extend automatic differentiation for second order derivatives of non-twice differentiable payoff.

In chapter 2, we study the approximation of the sensitivities of financial derivatives with a weighted multilevel Monte Carlo. The computation of sensitivities are made with an extended version of automatic differentiation for non-twice differentiable payoff using a regularization of the Heaviside function. We show how to calibrate the method for sensitivities using the optimal framework designed by Lemaire and Pagès in [119] for standard and antithetic discretization scheme. Also, we give a description of a framework which compute the value, Delta and Gamma

[2]This a translation of Préambule.

of a security using the optimal parameters for each function on the same grid. We give some numerical results on vanilla and exotic derivatives.

The chapter 3 gives a description of the parareal method for American contracts, a numerical section to assess its performance. The scalar case is investigated. Least Squares Monte Carlo (LSMC) and parareal time decomposition with two or more levels are used, leading to an efficient parallel implementation. It contains also a convergence argument for the two levels parareal Monte Carlo method when the time step used for the Euler scheme at each level is appropriate. This argument provides also a tool to analyze the multilevel case.

Chapter 4 is an extension of the previous chapter. We apply the parareal method to the approximation of a backward stochastic differential equation with an application to the pricing of a Bid-Ask Spread option for interest rates. The Least squares Monte Carlo method is a popular choice to approximate the solution for these equations. The discretization scheme is quite complex because there are at least two conditional expectations to evaluate. Hence, the parareal algorithm is a good method for solving backward stochastic differential equations.

The chapter 5 is dedicated to the work accomplished at Global Market Solutions for the design of an efficient calculation engine of rate derivatives. Fast portfolio assessing has become a major issue for financial institutions and banks, this work propose an answer to the growing demands in the number of risk measures. This engine calculation was written on a parallel architecture with distributed and shared memory. We will see how to use different type of parallelism granularity to benefit from every architectures have shown. We made several experiments to analyze and assess the performance.

In appendices 6, we present the different automatic differentiation libraries made during the collaboration with Global Market Solutions and which have been used in the chapters 1, 3 et 2. These libraries have been used for testing the possibility to integrate them inside IRIS (professional software to compute risk measures developed by Global Market Solutions). We briefly introduce the two libraries in C++ (up to 2nd order of derivation) and CUDA (1st order of derivation) on CPU and GPU. We make a short description of the implementation of the regularization for non smooth payoff. We make several applications to vanilla and exotic financial products in several simulation stochastic models.

Around Computation of Sensitivities

Chapitre 1

Vibrato and Automatic Differentiation for High Order Derivatives and Sensitivities of Financial Options

---

"La puissance de l'automatisation est telle que ses progrès mécaniques s'étendent aussi bien dans les contrées où la main d'œuvre est encore à vil prix, que dans les pays où le prix du travail manuel s'élève constamment"

– M. Alcan, Politician, (1810, 1877).

This chapter deals with the computation of second or higher order Greeks of financial securities. It combines two methods, Vibrato and automatic differentiation and compares with other methods. We show that this combined technique is faster and more stable than automatic differentiation of second order derivatives or finite difference approximations. We present a generic framework to compute any Greeks and present several applications on different types of derivatives. We also extend automatic differentiation for second order derivatives of non-twice differentiable payoff.

This chapter is an extended version of the article Pagès et al. [143] published in Risk Journal, *The Journal of Computational Finance*.

1.1 Introduction

Due to BASEL III regulations, banks are requested to evaluate the sensitivities of their portfolios every day (risk assessment). Some of these portfolios are huge and sensitivities are time consuming to compute accurately. Faced with the problem of building a software for this task and distrusting automatic differentiation for non-differentiable functions, we turned to an idea developed by Mike Giles called Vibrato.

Vibrato at core is a differentiation of a combination of likelihood ratio and pathwise evaluation. In Giles [60] and [62], it is shown that the computing time, stability and precision are enhanced compared with numerical differentiation of the full Monte Carlo path.

In many cases, gamma hedging being one, double sensitivities, i.e. second derivatives with respect to parameters, are needed. The purpose of this chapter is to investigate the feasibility of Vibrato for second and higher derivatives. We will first compare Vibrato applied twice with the analytic differentiation of Vibrato and show that it is equivalent; since the second is easier, we propose the best compromise for second derivatives: Automatic Differentiation of Vibrato.

In [40], Capriotti investigated the coupling of different mathematical methods – namely pathwise and likelihood ratio methods – with an Automatic differentiation technique for the computation of second order Greeks; here we follow the same idea but with Vibrato and also for the computation of higher order derivatives.

Automatic Differentiation (AD) of computer program as described by Griewank [81], Griewank and Walther [82], Naumann [137] and Hascœt [87] can be used in direct or reverse mode. In direct mode the computing cost is similar to finite difference approximation but with no roundoff errors: the method is exact because every line of the computer program which implements the financial option is differentiated exactly. The computing cost of a derivative of order n with respect to a single parameter is similar to running the program n times. Evaluation of all partial derivatives with respect of a vector valued parameter is substantially more expensive and requires the so-called reverse mode of AD.

For many financial products the first or the second sensitivities do not exist at some point, such is the case for the standard Digital option at $x = K$; even the payoff of a plain vanilla European option is not twice differentiable at $x = K$, yet the Gamma is well defined; in short, the end result is well defined but the intermediate steps of AD are not.

We tested `ADOL-C` of Griewank and Walther [83] and tried to compute the Hessian matrix for a standard European Call option in the Black Scholes model but the results were wrong. So we adapted our AD library based on operator overloading by including approximations of Dirac functions and obtained decent results; this is the second conclusion of the chapter: AD for second sensitivities can be made to work; it is simpler than Vibrato+AD (VAD) but it is slightly more computer intensive and risky because the precision depends on the right choice of the parameter which appears in the approximation of a Dirac function by an exponential.

More details on AD applied to quantitative finance can be found in Giles and Glasserman [58], Pironneau [154], Capriotti [39], Homescu [96] and the references therein.

An important constraint when designing professional software for risk assessment is to be compatible with the history of the company which contracts the software; most of the time, this rules out the use of partial differential equations to compute sensitivities (see Achdou and Pironneau [2]) as most quant companies use Monte Carlo algorithms for pricing their portfolios.

For security derivatives computed by a Monte Carlo method, the computation of their sensitivities with respect to a parameter is most easily approximated by finite difference (also known as the *shock method*) thus requiring the reevaluation of the security with an incremented parameter. There are two problems with this method: it is imprecise when generalized to higher order derivatives and expensive for multidimensional problems with multiple parameters. The n^{th} derivative of a security with p parameters requires $(n+1)p$ evaluations (and more if cross-derivatives are required); furthermore the choice of the perturbation parameter is delicate as it should be neither too large nor too small, as explained in section 1.3.1.

From a semi-analytical standpoint the most natural way to compute a sensitivity is the pathwise method described in Glasserman [71] which amounts to compute the derivative of the payoff for each simulation path. Unfortunately, this technique happens to be inefficient for certain

types of payoffs including some often used in quantitative finance like Digitals or Barrier options. For instance it is not possible to obtain the Delta of a Digital Call that way because the derivative of the expectation of a Digital is not equal to the expectation of the derivative. The pathwise derivative estimation is also called *infinitesimal perturbation* and there is a extensive literature on this subject; see for example Ho and Cao [94], in Suri and Zazanis [176] and in L'Ecuyer [118]. A general framework for some applications to option pricing is given in Glasserman [70].

There are also two well known mathematical methods to obtain sensitivities, the so-called log-likelihood ratio method and the Malliavin calculus. However, like the pathwise method, both have their own advantages and drawbacks. For the former, the method consists in differentiating the probability density of the underlying and clearly, it is not possible to compute Greeks if the probability density of the underlying is not known. Yet when the method applies, differentiating the probability density is easy because it is smooth while the payoff may not even be differentiable. More details can be found in Glynn [74], Reiman and Weiss [163], Rubinstein [167] and some applications to finance can be found in Broadie and Glasserman [35] and Glasserman and Zhao [73].

As for the Malliavin calculus, the computation of the Greeks consists in writing the expectation of the original payoff function times a specific factor i.e. the Malliavin weight. The main problem is that the computation of the Malliavin weight can be complex and/or computationally costly for a high dimensional problem. Several articles deal with the computation of Greeks via Malliavin calculus, Fournié et al. [55], Gobet and Munos [77] and Benhamou [19] to cite a few. The precision of the Malliavin formulas also degenerates for short maturities, especially for Δ-hedging.

Both the likelihood ratio and the Malliavin calculus are generally faster than the finite difference method when the volatility is constant. But implementations of these methods in the financial industry is limited by the specific analysis required for each new payoff.

The chapter is organized as follows; in section 1.2 we begin by recalling Vibrato for first order derivatives as in Giles [60] for the univariate and the multivariate case. We then generalize the method for the second and higher order derivatives with respect to one or several parameters and we describe how Vibrato of Vibrato is identical to analytical or Automatic differentiation of Vibrato.

In section 1.3, we recall briefly the different methods of Automatic differentiation. We describe the direct and the adjoint or reverse mode to differentiate a computer program. We also explain how it can be extended to some non differentiable functions.

Section 1.4 deals with several applications to different derivative securities. We show some results of second order derivatives (Gamma and Vanna) and third order derivatives in the case of a standard European Call option: the sensitivity of the Gamma with respect to changes in the underlying asset and cross-derivatives with respect to spot price and interest rate and others. Also, we compare different techniques of Automatic differentiation and we give some details about our computer implementations. We also illustrate the method on a multidimensional Basket option and on an Autocallable.

In section 1.5, we study the computing time for the evaluation of the Hessian matrix of a standard European Call Option in the Black Scholes model.

In section 1.6 we explain briefly how the computation of the Gamma for an American Put option computed with the Longstaff Schwartz algorithm in Longstaff and Schwartz [124] can be done much in the same fashion as for an European option and on a European Call with Heston's model in section 1.7.

Finally, In section 1.8 we compare VADs to Malliavin and to the likelihood ratio method in

the context of short maturities. We conclude with a summary and some perspectives for future work.

1.2 Vibrato

Vibrato was introduced by Giles in [60]; it is based on a reformulation of the payoff which is better suited to differentiation. The Monte Carlo path is split into the last time step and its past as represented on the figure 1.1. Let us explain the method on a plain vanilla multi-dimensional option.

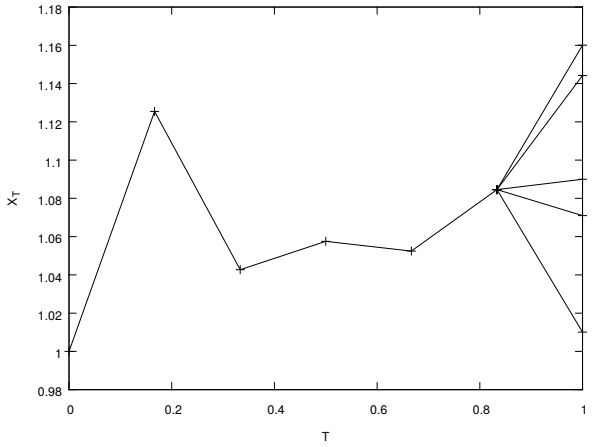

Figure 1.1: Representation of Vibrato decomposition on a Monte Carlo path.

First, let us recall the likelihood ratio method for derivatives.
Let the parameter set Θ be a subset of $\mathbb{R}^p$. Let $b : \Theta \times \mathbb{R}^d \to \mathbb{R}^d$, $\sigma : \Theta \times \mathbb{R}^d \to \mathbb{R}^{d\times q}$ be continuous functions, locally Lipschitz in the space variable, with linear growth, both uniformly in $\theta \in \Theta$. We omit time as variable in both b and σ only for simplicity. And let $(W_t)_{t\geq 0}$ be a q-dimensional standard Brownian motion defined on a probability space $(\Omega, \mathscr{A}, \mathbb{P})$.

Lemma 1. (Log-likelihood ratio)

Let $p(\cdot,\cdot) : \Theta \times \mathbb{R}^d \mapsto \mathbb{R}$,be the probability density of a random variable $X(\theta)$; given $V \in L^1(\mathbb{R}^d)$, consider

$$\mathbb{E}[V(X(\theta))] = \int_{\mathbb{R}^d} V(y)p(\theta, y)dy. \tag{1.1}$$

If $\theta \mapsto p(\theta,\cdot)$ is differentiable, then, under a standard domination or a uniform integrability assumption, one can interchange differentiation and integration : for $i = 1, .., p$,

$$\frac{\partial}{\partial\theta_i}\Big[\mathbb{E}[V(X(\theta))]\Big] = \int_{\mathbb{R}^d} V(y)\frac{\partial \ln p}{\partial\theta_i}(\theta, y)p(\theta, y)dy = \mathbb{E}\left[V(X(\theta))\frac{\partial \ln p}{\partial\theta_i}(\theta, X(\theta))\right]. \tag{1.2}$$

1.2.1 Vibrato for a European Contract

Let $X = (X_t)_{t\in[0,T]}$ be a diffusion process, the strong solution of the following Stochastic Differential Equation (SDE)

$$\begin{cases} dX_t = b(\theta, X_t)\,dt + \sigma(\theta, X_t)dW_t, \\ X_0 = x. \end{cases} \tag{1.3}$$

Obviously, X_t depends on θ; for clarity, we write $X_t(\theta)$ when the context requires it.

Given an integer $n > 0$, the Euler scheme with constant step $h = \frac{T}{n}$, defined below in (1.3), approximates X_t at time $t_k = kh$, i.e. $\bar{X}_k \approx X_{kh}$, and it is recursively defined by

$$\begin{cases} \bar{X}_k = \bar{X}_{k-1} + b(\theta, \bar{X}_{k-1})h + \sigma(\theta, \bar{X}_{k-1})\sqrt{h}Z_k, \quad k = 1, \ldots, n, \\ \bar{X}_0 = x, \end{cases} \tag{1.4}$$

where $\{Z_k\}_{k=1,..,n}$ are independent random Gaussian $\mathcal{N}(0, I_d)$ vectors. The relation between W and Z is

$$W_{t_k} - W_{t_{k-1}} = \sqrt{h}Z_k, \quad k = 1, .., n. \tag{1.5}$$

Vibrato is based on the fundamental remark that if

$$\begin{array}{rl} \bar{X}_n & = \mu_{n-1}(\theta) + \nu_{n-1}(\theta)Z_n \quad \text{with } Z_n \text{ independent of} \\ \mu_{n-1}(\theta) & = \bar{X}_{n-1}(\theta) + b(\theta, \bar{X}_{n-1}(\theta))h \text{ and} \\ \nu_{n-1}(\theta) & = \sigma(\theta, \bar{X}_{n-1}(\theta))\sqrt{h}, \end{array} \tag{1.6}$$

then,

$$\begin{aligned} \mathbb{E}\left[V(\bar{X}_n(\theta))\right] &= \mathbb{E}\left[\mathbb{E}\left[V(\bar{X}_n(\theta)) \mid (W_{t_k})_{k=0,\ldots,n-1}\right]\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[V(\bar{X}_n(\theta)) \mid \bar{X}_{n-1}(\theta)\right]\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[V(\mu_{n-1} + \nu_{n-1}Z_n) \mid \mu_{n-1}, \nu_{n-1}\right]\right]. \end{aligned} \tag{1.7}$$

This follows from the obvious fact that the Euler scheme defines a Markov chain $(\bar{X}_k)_{k=0,\ldots,n}$ with respect to the filtration $\mathcal{F}_k = \sigma(W_{t_\ell},\, \ell = 0, \ldots, k)$.

1.2.2 First Order Vibrato

Let us recall Giles' result:

Theorem 1. (Vibrato, multidimensional first order case)

$$\begin{aligned} \frac{\partial}{\partial\theta_i}\mathbb{E}[V(\bar{X}_n(\theta))] &= \mathbb{E}\left[\frac{\partial}{\partial\theta_i}\left\{\mathbb{E}[V(\mu_{n-1}(\theta) + \nu_{n-1}(\theta)Z_n) \mid \mu_{n-1}, \nu_{n-1}]\right\}\right] \\ &= \mathbb{E}\left[\left\{\frac{\partial\mu}{\partial\theta_i}\cdot\mathbb{E}\left[V(\mu + \nu Z_n)\Sigma^{-1}\nu Z_n\right] + \right.\right. \\ &\left.\left.\frac{\partial\Sigma}{\partial\theta_i} : \mathbb{E}\left[V(\mu + \nu Z_n)\left(-\frac{1}{2}\Sigma^{-1} + \frac{1}{2}(\Sigma^{-1}\nu Z_n)(\Sigma^{-1}\nu Z_n)^T\right)\right]\right\}\middle|\begin{array}{c}\mu = \mu_{n-1}(\theta) \\ \Sigma = \nu\nu^T \\ \nu = \nu_{n-1}(\theta)\end{array}\right]. \end{aligned} \tag{1.8}$$

where $\Sigma = h\sigma\sigma^T$, $\cdot$ denotes the scalar product and $:$ denotes the trace of the product of the matrices.

By $\frac{\partial \mu}{\partial \theta_i} \cdot \mathbb{E}\left[W(\mu,\nu,Z)\right]\Big|_{nu=\nu_{n-1}(\theta)}^{\mu=\mu_{n-1}(\theta)}$ we mean that inside $\mathbb{E}$ only Z is random, then the result and $\frac{\partial \mu}{\partial \theta_i}$ are evaluated at $\mu_{n-1}(\theta), \nu_{n-1}(\theta)$.

To prove the theorem we set some notations and prove two intermediate results.

We denote $\varphi(\mu,\nu) = \mathbb{E}\left[V(\mu+\nu Z_n)\right]$. From (1.7) and (1.3), for any $i \in \{1,\dots,p\}$,

$$\frac{\partial}{\partial \theta_i}\mathbb{E}[V(\bar{X}_n(\theta))] = \mathbb{E}\left[\frac{\partial \varphi}{\partial \theta_i}\right] = \mathbb{E}\left[\frac{\partial \mu_{n-1}}{\partial \theta_i}\cdot\frac{\partial \varphi}{\partial \mu} + \frac{\partial \nu_{n-1}}{\partial \theta_i} : \frac{\partial \varphi}{\partial \nu}\right] \tag{1.9}$$

Lemma 2. *The θ_i-tangent process to X, $Y_t = \dfrac{\partial X_t}{\partial \theta_i}$, is defined as the solution of the following SDE (see Kunita [115] for a proof)*

$$\begin{cases} dY_t = \left[b'_{\theta_i}(\theta,X_t) + b'_x(\theta,X_t)Y_t\right]dt + \left[\sigma'_{\theta_i}(\theta,X_t) + \sigma'_x(\theta,X_t)Y_t\right]dW_t, \\ Y_0 = \dfrac{\partial X_0}{\partial \theta_i} \end{cases} \tag{1.10}$$

where the primes denote standard partial derivatives.

As for $\bar{X}_k$ in (1.3), we may discretize (1.10) by

$$\begin{aligned} \bar{Y}_{k+1} &= \bar{Y}_k + \left[b'_{\theta_i}(\theta,\bar{X}_k) + b'_x(\theta,\bar{X}_k)\bar{Y}_k\right]h & (1.11)\\ &\quad + \left[\sigma'_{\theta_i}(\theta,\bar{X}_k) + \sigma'_x(\theta,\bar{X}_k)\bar{Y}_k\right]\sqrt{h}Z_{k+1}. & (1.12)\end{aligned}$$

Then from (1.6),

$$\begin{aligned} \frac{\partial \mu_{n-1}}{\partial \theta_i} &= \bar{Y}_{n-1}(\theta) + h\left[b'_{\theta_i}(\theta,\bar{X}_{n-1}(\theta)) + b'_x(\theta,\bar{X}_{n-1}(\theta))\bar{Y}_{n-1}(\theta)\right] \\ \frac{\partial \nu_{n-1}}{\partial \theta_i} &= \left(\sigma'_{\theta_i}(\theta,\bar{X}_{n-1}(\theta)) + \sigma'_x(\theta,\bar{X}_{n-1}(\theta))\bar{Y}_{n-1}(\theta)\right)\sqrt{h}. \end{aligned} \tag{1.13}$$

So far we have shown the following lemma.

Lemma 3. *When $\bar{X}_n(\theta)$ is the solution of (1.4), then $\dfrac{\partial}{\partial \theta_i}\mathbb{E}[V(\bar{X}_n(\theta))]$ is given by (1.9), (1.13) and (1.11).*

In (1.3) b and σ are constant in the time interval $(kh,(k+1)h)$, therefore the conditional probability of $\bar{X}_n$ given $\bar{X}_{n-1}$ is

$$p(x) = \frac{1}{(\sqrt{2\pi})^d\sqrt{\det(\Sigma)}}e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \tag{1.14}$$

where μ and Σ are evaluated at time $(n-1)h$ and given by (1.6). As in Dwyer and Macphail [52],

$$\begin{aligned} &\frac{\partial}{\partial \mu}\ln p(x) = \Sigma^{-1}(x-\mu), \qquad \frac{\partial}{\partial \Sigma}\ln p(x) = -\frac{1}{2}\Sigma^{-1} + \frac{1}{2}\Sigma^{-1}(x-\mu)(x-\mu)^T\Sigma^{-1} \Rightarrow \\ &\frac{\partial}{\partial \mu}\ln p(x)|_{x=\bar{X}_n} = \Sigma^{-1}\nu Z_n, \qquad \frac{\partial}{\partial \Sigma}\ln p(x)|_{x=\bar{X}_n} = -\frac{1}{2}\Sigma^{-1} + \frac{1}{2}(\Sigma^{-1}\sigma Z_n)(\Sigma^{-1}\sigma Z_n)^T \end{aligned}$$

Finally, applying Lemma 3 and Lemma 1 leads to (1.8).

1.2.3 Antithetic Vibrato

One can expect to improve the above formula – that is, reducing its variance – by the means of an antithetic transform. This can be done on all formulas obtained above. We do it only once, and in the case $q = d$, to illustrate the methodology. The following holds:

$$\mathbb{E}\left[V(\mu+\nu Z)\nu^{-T}Z\right] = \frac{1}{2}\mathbb{E}\left[\left(V(\mu+\nu Z) - V(\mu-\nu Z)\right)\nu^{-T}Z\right]. \tag{1.15}$$

where $\nu^{-T} := (\nu^T)^{-1}$; similarly, using $\mathbb{E}[ZZ^T - I] = 0$,

$$\begin{aligned}&\mathbb{E}\left[V(\mu+\nu Z)\nu^{-T}(ZZ^T - I)\nu^{-1}\right]\\ &= \frac{1}{2}\mathbb{E}\left[\left(V(\mu+\nu Z) - 2V(\mu) + V(\mu-\nu Z)\right)\nu^{-T}(ZZ^T - I)\nu^{-1}\right].\end{aligned} \tag{1.16}$$

Corollary 1. (One dimensional case, $d = 1$)

$$\begin{aligned}\frac{\partial}{\partial\theta}\quad \mathbb{E}[V(\bar{X}_n(\theta))] = \frac{1}{2}\mathbb{E}\Bigg[\Big\{&\frac{\partial\mu}{\partial\theta}\mathbb{E}\left[\left(V(\mu+\nu Z) - V(\mu-\nu Z)\right)\frac{Z}{\nu}\right]\\ &+ \frac{\partial\nu}{\partial\theta}\mathbb{E}\left[\left(V(\mu+\nu Z) - 2V(\mu) + V(\mu-\nu Z)\right)\frac{Z^2-1}{\nu}\right]\Big\}\Big|_{\substack{\mu=\mu_{n-1}(\theta)\\ \nu=\nu_{n-1}(\theta)}}\Bigg].\end{aligned} \tag{1.17}$$

Remark 1. *In the numerical tests, the expectation with respect to Z will be replaced by an average over M_Z paths. For simple cases such as the sensitivities of European options, a small M_Z suffices; this is because there is another average with respect to M in the outer loop, the number of paths to compute the expectation with respect to μ_{n-1} and ν_{n-1}.*

1.2.4 Second Order Derivatives

Assume that X_0, b and σ depend on two parameters $(\theta_1, \theta_2) \in \Theta^2$. There are two ways to compute second order derivatives. Either by differentiating the Vibrato (1.8) while using Lemma 1 or by applying the Vibrato idea to the second derivative.

1.2.4.1 Second Order Derivatives by Differentiation of Vibrato

Let us differentiate (1.8) with respect to a second parameter θ_j:

$$\begin{aligned}\frac{\partial^2}{\partial\theta_i\partial\theta_j}\mathbb{E}[V(X_T)] = \mathbb{E}\Big\{\Big[\Big(&\frac{\partial^2\mu}{\partial\theta_i\partial\theta_j}\cdot\mathbb{E}\left[V(\mu+\nu Z)\nu^{-T}Z\right]\\ &+\frac{\partial\mu}{\partial\theta_i}\cdot\frac{\partial}{\partial\theta_j}\mathbb{E}\left[V(\mu+\nu Z)\nu^{-T}Z\right]\Big)\\ &+\frac{1}{2}\Big(\frac{\partial^2\Sigma}{\partial\theta_i\partial\theta_j} : \mathbb{E}\left[V(\mu+\nu Z)\nu^{-T}(ZZ^T - I)\nu^{-1}\right]\\ &+\frac{\partial\Sigma}{\partial\theta_i} : \frac{\partial}{\partial\theta_j}\mathbb{E}\left[V(\mu+\nu Z)\nu^{-T}(ZZ^T - I)\nu^{-1}\right]\Big)\Big\}\Big|_{\substack{\mu=\mu_{n-1}(\theta)\\ \nu=\nu_{n-1}(\theta)}}\Big].\end{aligned} \tag{1.18}$$

The derivatives can be expanded further; for instance in the one dimensional case and after a tedious algebra one obtains:

Theorem 2. (Second Order by Differentiation of Vibrato, $d = 1$)

$$\frac{\partial^2}{\partial\theta^2}\mathbb{E}[V(X_T)] = \mathbb{E}\left[\left\{\frac{\partial^2\mu}{\partial\theta^2}\mathbb{E}\left[V(\mu+\nu Z)\frac{Z}{\nu}\right] + \left(\frac{\partial\mu}{\partial\theta}\right)^2\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^2-1}{\nu^2}\right]\right.\right.$$
$$+\left(\frac{\partial\nu}{\partial\theta}\right)^2\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^4-5Z^2+2}{\nu^2}\right] + \frac{\partial^2\nu}{\partial\theta^2}\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^2-1}{\nu}\right]$$
$$\left.\left.+2\frac{\partial\mu}{\partial\theta}\frac{\partial\sigma}{\partial\theta}\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^3-3Z}{\nu^2}\right]\right\}\Bigg|_{\substack{\mu=\mu_{n-1}(\theta)\\ \nu=\nu_{n-1}(\theta)}}\right]. \tag{1.19}$$

1.2.4.2 Second Order Derivatives by Second Order Vibrato

The same Vibrato strategy can be applied also directly to second derivatives.

As before the derivatives are transferred to the PDE p of X_T:

$$\begin{aligned}\frac{\partial^2}{\partial\theta_i\partial\theta_j}\mathbb{E}[V(X_T)] &= \int_{\mathbb{R}^d}\frac{V(x)}{p(x)}\frac{\partial^2 p}{\partial\theta_i\partial\theta_j}p(x)\mathrm{d}x = \int_{\mathbb{R}^d}V(x)[\frac{\partial^2\ln p}{\partial\theta_i\partial\theta_j}+\frac{\partial\ln p}{\partial\theta_i}\frac{\partial\ln p}{\partial\theta_j}]p(x)\mathrm{d}x\\ &= \mathbb{E}\left[V(x)\left(\frac{\partial^2\ln p}{\partial\theta_i\partial\theta_j}+\frac{\partial\ln p}{\partial\theta_i}\frac{\partial\ln p}{\partial\theta_j}\right)\right].\end{aligned} \tag{1.20}$$

Then

$$\begin{aligned}\frac{\partial^2}{\partial\theta_1\partial\theta_2}\mathbb{E}[V(\bar{X}_T(\theta_1,\theta_2))] &= \frac{\partial^2\varphi}{\partial\theta_1\partial\theta_2}(\mu,\nu)\\ &= \frac{\partial\mu}{\partial\theta_1}\frac{\partial\mu}{\partial\theta_2}\frac{\partial^2\varphi}{\partial\mu^2}(\mu,\nu)+\frac{\partial\nu}{\partial\theta_1}\frac{\partial\nu}{\partial\theta_2}\frac{\partial^2\varphi}{\partial\nu^2}(\mu,\nu)+\frac{\partial^2\mu}{\partial\theta_1\partial\theta_2}\frac{\partial\varphi}{\partial\mu}(\mu,\nu)\\ &\quad+\frac{\partial^2\nu}{\partial\theta_1\partial\theta_2}\frac{\partial\varphi}{\partial\nu}(\mu,\nu)+\left(\frac{\partial\mu}{\partial\theta_1}\frac{\partial\nu}{\partial\theta_2}+\frac{\partial\nu}{\partial\theta_1}\frac{\partial\mu}{\partial\theta_2}\right)\frac{\partial^2\varphi}{\partial\mu\partial\nu}(\mu,\nu),\end{aligned}$$

where μ and ν are evaluated at μ_{n-1}, ν_{n-1}. We need to calculate the two new terms $\dfrac{\partial^2}{\partial\theta_1\partial\theta_2}\mu_{n-1}(\theta_1,\theta_2)$ and $\dfrac{\partial^2}{\partial\theta_1\partial\theta_2}\nu_{n-1}(\theta_1,\theta_2)$. It requires the computation of the first derivative with respect to θ_i of the tangent process Y_t, that we denote $Y_t^{(2)}(\theta_1,\theta_2)$.

Then (1.13) is differentiated and an elementary though tedious computations yields the following proposition:

Proposition 1. *The θ_i-tangent process $Y_t^{(i)}$ defined above in Lemma 1.10 has a θ_j-tangent process $Y_t^{(ij)}$ defined by*

$$\begin{aligned}dY_t^{(ij)} &= \left[b''_{\theta_i\theta_j}+b''_{\theta_i,x}Y_t^{(j)}+b''_{\theta_j,x}Y_t^{(i)}+b''_{x^2}Y_t^{(i)}Y_t^{(j)}+b'_xY_t^{(ij)}\right]dt\\ &\quad+\left[\sigma''_{\theta_i\theta_j}+\sigma''_{\theta_i,x}Y_t^{(j)}+\sigma''_{\theta_j,x}Y_t^{(i)}+\sigma''_{x^2}Y_t^{(i)}Y_t^{(j)}+\sigma'_xY_t^{(ij)}\right]dW_t.\end{aligned}$$

Finally in the univariate case $\theta = \theta_1 = \theta_2$ this gives

Proposition 2. (Second Order Vibrato)

$$\frac{\partial^2}{\partial\theta^2}\mathbb{E}[V(X_T)] = \mathbb{E}\left[\frac{\partial^2\mu}{\partial\theta^2}\mathbb{E}\left[V(\mu+\nu Z)\frac{Z}{\nu}\right]+\left(\frac{\partial\mu}{\partial\theta}\right)^2\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^2-1}{\nu^2}\right]\right.$$

$$+\left(\frac{\partial\nu}{\partial\theta}\right)^2\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^4-5Z^2+2}{\nu^2}\right]+\frac{\partial^2\nu}{\partial\theta^2}\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^2-1}{\nu}\right]$$
$$+2\frac{\partial\mu}{\partial\theta}\frac{\partial\nu}{\partial\theta}\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^3-3Z}{\nu^2}\right]\Bigg]. \tag{1.21}$$

where μ and ν are evaluated at μ_{n-1},ν_{n-1} by 1.6.

Remark 2. *It is equivalent to Theorem 2, hence to the direct differentiation of Vibrato.*

1.2.4.3 Terms in Vibrato Second Order

Computation of the terms $\frac{\partial\varphi^2}{\partial\mu^2}$, $\frac{\partial\varphi^2}{\partial\nu^2}$ and $\frac{\partial\varphi^2}{\partial\mu\partial\sigma}$. For the second derivative with respect to μ, we start from

$$\frac{\partial\varphi}{\partial\mu}(\mu,\nu)=\mathbb{E}\left[V(\mu+\nu Z)\frac{Z}{\nu}\right]=\frac{1}{\nu^2}\left[\mathbb{E}[V(\mu+\nu Z)(\mu+\nu Z)]-\mu\mathbb{E}[V(\mu+\nu Z)]\right]. \tag{1.22}$$

We set

$$\tilde{V}(\mu+\nu Z)=V(\mu+\nu Z)(\mu+\nu Z). \tag{1.23}$$

Then

$$\begin{aligned}\frac{\partial^2}{\partial\mu^2}\mathbb{E}[V(\mu+\nu Z)] &= \frac{1}{\nu^2}\left[\mathbb{E}\left[\tilde{V}(\mu+\nu Z)\frac{Z}{\nu}\right]-\mathbb{E}[V(\mu+\nu Z)]\right.\\ &\quad\left.-\mu\mathbb{E}\left[V(\mu+\nu Z)\frac{Z}{\nu}\right]\right]\\ &= \frac{1}{\nu^2}\mathbb{E}\left[V(\mu+\nu Z)(\mu+\nu Z-\mu)\frac{Z}{\nu}-V(\mu+\nu Z)\right]\\ &= \mathbb{E}\left[V(\mu+\nu Z)\frac{Z^2-1}{\nu^2}\right].\end{aligned} \tag{1.24}$$

Now, for the second derivative with respect to σ, we have (with 1.13, 1.17 and 1.6)):

$$\frac{\partial\varphi}{\partial\sigma}(\mu,\nu)=\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^2-1}{\nu}\right]=\frac{1}{\nu^3}\mathbb{E}[V(\mu+\nu Z)(u-\mu)^2]-\frac{1}{\nu}\mathbb{E}[V(\mu+\nu Z)]. \tag{1.25}$$

With

$$\tilde{V}(u)=V(u)(u-\mu)^2. \tag{1.26}$$

Hence,

$$\begin{aligned}\frac{\partial^2}{\partial\nu^2}\mathbb{E}[V(\mu+\nu Z)] &= \frac{1}{\sigma^3 h}\mathbb{E}\left[\tilde{V}(\mu+\nu Z)\left(\frac{Z^2-1}{\nu}-\frac{3}{\nu}\right)\right]\\ &\quad-\frac{1}{\nu}\mathbb{E}\left[V(\mu+\nu Z)\left(\frac{Z^2-1}{\nu}-\frac{1}{\nu}\right)\right]\\ &= \mathbb{E}\left[\tilde{V}(\mu+\nu Z)\frac{Z^2-4}{\nu^2}\right]-\mathbb{E}\left[V(\mu+\nu Z)\frac{Z^2-2}{\nu^2}\right]\\ &= \mathbb{E}\left[V(\mu+\nu Z)\frac{Z^4-5Z^2+2}{\nu^2}\right].\end{aligned} \tag{1.27}$$

As for the mixed derivatives with respect to μ then to ν, we have (starting from (1.22) and with $\tilde{V}(\mu+\nu Z)=V(\mu+\nu Z)(\mu+\nu Z)$,1.13, 1.17 and 1.6)

$$\frac{\partial^2}{\partial\nu\partial\mu}\mathbb{E}[V(\mu+\nu Z)] = \frac{1}{\nu^2}\mathbb{E}\left[\bar{V}(\mu+\nu Z)\left(\frac{Z^2-1}{\nu}-\frac{2}{\nu}\right)\right]$$

$$
\begin{aligned}
& -\frac{1}{\nu^2}\mu\mathbb{E}\left[V(\mu+\nu Z)\left(\frac{Z^2-1}{\nu}\right)\right] \\
= \;& \mathbb{E}\left[V(\mu+\nu Z)\frac{Z(Z^2-1)}{\nu^2}\right]-\mathbb{E}\left[V(\mu+\nu Z)\frac{2Z}{\nu^2}\right] \\
= \;& \mathbb{E}\left[V(\mu+\nu Z)\frac{Z^3-3Z}{\nu^2}\right]. \qquad (1.28)
\end{aligned}
$$

1.2.5 Antithetic Transform, Regularity and Variance

In this section, we assume $d = q = 1$ for simplicity.

Starting from Vibrato $\varphi(\mu,\nu) = \mathbb{E}[f(\mu+\nu Z)]$ and assuming f Lipschitz continuous with Lipschitz coefficients $[f]_{\mathrm{Lip}}$, we have

$$
\frac{\partial\varphi}{\partial\mu}(\mu,\nu) = \mathbb{E}\left[f(\mu+\nu Z)\frac{Z}{\nu}\right] = \mathbb{E}\left[\left(f(\mu+\nu Z)-f(\mu-\nu Z)\right)\frac{Z}{2\nu}\right]. \qquad (1.29)
$$

Therefore the variance satisfies

$$
\begin{aligned}
\mathrm{Var}\left[\left(f(\mu+\nu Z)-f(\mu-\nu Z)\right)\frac{Z}{2\nu}\right] &\leq \mathbb{E}\left[\left|\left(f(\mu+\nu Z)-f(\mu-\nu Z)\right)\frac{Z}{2\nu}\right|^2\right] \\
&\leq [f]^2_{\mathrm{Lip}}\mathbb{E}\left[\frac{(2\nu Z)^2}{4\nu^2}Z^2\right] \\
&= [f]^2_{\mathrm{Lip}}\mathbb{E}[Z^4] \\
&= 3[f]^2_{\mathrm{Lip}}.
\end{aligned}
$$

As $\mathbb{E}[Z] = 0$, we also have

$$
\frac{\partial\varphi}{\partial\mu}(\mu,\nu) = \mathbb{E}\left[\left(f(\mu+\nu Z)-f(\mu)\right)\frac{Z}{\nu}\right]. \qquad (1.30)
$$

Then,

$$
\begin{aligned}
\mathrm{Var}\left[\left(f(\mu+\nu Z)-f(\mu)\right)\frac{Z}{\nu}\right] &\leq \mathbb{E}\left[\left|\left(f(\mu+\nu Z)-f(\mu)\right)\frac{Z}{\nu}\right|^2\right] \\
&\leq \frac{1}{\nu^2 h}[f]^2_{\mathrm{Lip}}\mathbb{E}\left[(\nu Z)^2 Z^2\right] = [f]^2_{\mathrm{Lip}}\mathbb{E}[Z^4] = 3[f]^2_{\mathrm{Lip}}
\end{aligned}
$$

Remark 3. *The variances of formulas (1.29) and (1.30) are equivalent but the latter is less expensive to compute.*

If f is differentiable and f' has polynomial growth, we also have

$$
\frac{\partial\varphi}{\partial\mu}(\mu,\nu) = \mathbb{E}[f'(\mu+\nu Z)]. \qquad (1.31)
$$

Thus,

$$
\mathrm{Var}\left[f'(\mu+\nu Z)\right] \leq \mathbb{E}\left[\left(f'(\mu+\nu Z)\right)^2\right] \leq \|f'\|^2_\infty.
$$

Remark 4. *Let $[f]_{Lip}$ denote the Lipschitz constant of f. If f' is bounded, we have $[f]_{Lip} = \|f'\|_\infty$ then the expression in (1.31) has a smaller variance than (1.29) and (1.30).*

Assume that f' is Lipschitz continuous with Lipschitz coefficients $[f']_{\mathrm{Lip}}$. We can improve the efficiency of (1.31) because

$$
\mathrm{Var}\left[f'(\mu+\nu Z)\right] = \mathrm{Var}\left[f'(\mu+\nu Z)-f'(\mu)\right]
$$

$$\begin{aligned} &\leq \mathbb{E}\left[\left|f'(\mu+\nu Z)-f'(\mu)\right|^2\right] \\ &\leq [f']^2_{\text{Lip}}\nu^2\mathbb{E}[Z^2] \\ &\leq [f']_{\text{Lip}}\nu^2 \end{aligned}$$

Remark 5. *Assuming that $f(x)=\mathbb{1}_{x\leq K}$, clearly we cannot differentiate inside the expectation and the estimation of the variance seen previously can not be applied.*

1.2.5.1 Indicator Function

Let us assume that $f(x)=\mathbb{1}_{x\leq K}$. To simplify assume that $K\leq\mu$, we have

$$|f(\mu+\nu Z)-f(\mu-\nu Z)| = \left|\mathbb{1}_{Z\leq\frac{K-\mu}{\nu}}-\mathbb{1}_{Z\geq\frac{\mu-K}{\nu}}\right| = \mathbb{1}_{Z\notin\left[\frac{K-\mu}{\nu},\frac{\mu-K}{\nu}\right]},$$

hence

$$\left|\left(f(\mu+\nu Z)-f(\mu-\nu Z)\right)\frac{Z}{\nu}\right| = \frac{1}{\nu}|Z|\mathbb{1}_{Z\notin\left[\frac{K-\mu}{\nu},\frac{\mu-K}{\nu}\right]}.$$

For the variance, we have

$$\text{Var}\left[\left(f(\mu+\nu Z)-f(\mu-\nu Z)\right)\frac{Z}{\nu}\right] \leq \mathbb{E}\left[\left|\left(f(\mu+\nu Z)-f(\mu-\nu Z)\right)\frac{Z}{\nu}\right|^2\right].$$

By Cauchy-Schwarz we can write

$$\begin{aligned} \mathbb{E}\left[\left|\left(f(\mu+\nu Z)-f(\mu-\nu Z)\right)\frac{Z}{\nu}\right|^2\right] &= \frac{1}{2\nu^2}\mathbb{E}\left[Z^2\left|f(\mu+\nu Z)-f(\mu-\nu Z)\right|^2\right] \\ &= \frac{1}{2\nu^2}\mathbb{E}\left[Z^2\mathbb{1}_{\left\{Z\notin\left[\frac{K-\mu}{\nu},\frac{\mu-K}{\nu}\right]\right\}}\right] \\ &\leq \frac{1}{2\nu^2}\left(\mathbb{E}[Z^4]\right)^{\frac{1}{2}}\left(\mathbb{P}\left(Z\notin\left[\frac{K-\mu}{\nu},\frac{\mu-K}{\nu}\right]\right)\right)^{\frac{1}{2}} \\ &\leq \frac{\sqrt{3}}{2\nu^2}\left(2\mathbb{P}\left(Z\geq\frac{\mu-K}{\nu}\right)\right)^{\frac{1}{2}}. \end{aligned}$$

Then

$$\frac{\sqrt{3}}{2\nu^2}\left(2\mathbb{P}\left(Z\geq\frac{\mu-K}{\nu}\right)\right)^{\frac{1}{2}} = \frac{\sqrt{6}}{2\nu^2}\left(\int_{\frac{\mu-K}{\nu}}^{+\infty}e^{-\frac{u^2}{2}}\frac{du}{\sqrt{2\pi}}\right)^{\frac{1}{2}}.$$

Now, $\forall\ a>0,\ \mathbb{P}(Z\geq a)\leq\dfrac{e^{-\frac{a^2}{2}}}{a\sqrt{2\pi}}$, so when $a\to+\infty$,

$$\begin{aligned} \text{Var}\left[\left(f(\mu+\nu Z)-f(\mu-\nu Z)\right)\frac{Z}{\nu}\right] &\leq \frac{1}{\nu^2}\sqrt{\frac{3}{2}}\frac{e^{-\frac{(\mu-K)^2}{4\nu^2}}}{(2\pi)^{\frac{1}{4}}\sqrt{\frac{\mu-K}{\nu}}} \\ &\leq \frac{1}{(2\pi)^{\frac{1}{4}}\nu^{\frac{3}{2}}}\sqrt{\frac{3}{2}}\frac{e^{-\frac{(\mu-K)^2}{4\nu^2}}}{\sqrt{\mu-K}} \underset{\nu\to0}{\longrightarrow} \begin{cases} 0 & \text{if } \mu\neq K \\ +\infty & \text{otherwise.} \end{cases} \end{aligned}$$

The fact that such estimate can be obtained with non differentiable f demonstrates the power of the Vibrato technique.

1.2.6 Higher Order Vibrato

The Vibrato-AD method can be generalized to higher order of differentiation of Vibrato with respect to the parameter θ with the help of the Faà di Bruno formula and its generalization to a composite function with a vector argument, as given in Mishkov [134].

We set:

$$\phi(\theta) = (\mu(\theta), \nu(\theta)) \tag{1.32}$$

Proposition 3. ***l*-th Order Vibrato (Unidimensional Case)** *We suppose that all necessary derivatives are defined, then*

$$\frac{d^l}{d\theta^l}\varphi(\phi(\theta)) = \sum_0 \sum_1 \sum_2 \cdots \sum_l \frac{l!}{\prod_{i=1}^n (i!)^{k_i} \prod_{i=1}^l q_{i,\mu}! q_{i,\nu}!} \frac{\partial^k}{\partial \mu^{p_\mu} \partial \nu^{p_\nu}} \times \prod_{i=1}^l (\mu^{(i)})^{q_{i,\mu}} (\nu^{(i)})^{q_{i,\nu}} \tag{1.33}$$

where the respective sums are over all non negative integer solutions of the Diophantine equations, as follows

$$\begin{aligned}
&\sum_0 \rightarrow k_1 + 2k_2 + \cdots + lk_l = l \\
&\sum_1 \rightarrow q_{1,\mu} + q_{1,\nu} = k_1 \\
&\sum_2 \rightarrow q_{2,\mu} + q_{2,\nu} = k_2 \\
&\vdots \\
&\sum_l \rightarrow q_{l,\mu} + q_{l,\nu} = k_l
\end{aligned} \tag{1.34}$$

and with p_μ and p_ν the order of the partial derivative with respect to μ and to ν, k denotes the order of the partial derivative with the following formulas

$$\begin{aligned}
p_\mu &= q_{1,\mu} + q_{2,\mu} + \cdots + q_{l,\mu} \\
p_\sigma &= q_{1,\sigma} + q_{2,\sigma} + \cdots + q_{l,\sigma} \\
k &= p_\mu + p_\sigma = k_1 + k_2 + \cdots + k_l
\end{aligned} \tag{1.35}$$

1.3 Second Derivatives by Vibrato plus Automatic Differentiation (VAD)

The differentiation that leads to formula (1.21) can be derived automatically by AD; then one has just to write a computer program that implements 1.8 and apply automatic differentiation to the computer program. We recall here the basis of AD.

1.3.1 Automatic Differentiation

Consider a function $z = f(u)$ implemented in `C` or `C++` on the code 1.1 by

```
double f(double u)
```

Code 1.1: Function implemented in C++.

To find an approximation of z'_u, one could call in `C` on the next code 1.2

```
double dxdu= (f(u + du)-f(u))/du
```

Code 1.2: Approximation of the derivative of a function implemented in C++.

because

$$z'_u = f'(u) = \frac{f(u+\mathrm{d}u) - f(u)}{\mathrm{d}u} + O(|\mathrm{d}u|).$$

A good precision ought to be reached by choosing $\mathrm{d}u$ small. However arithmetic truncation limits the accuracy (figure 1.2) and shows that it is not easy to choose $\mathrm{d}u$ appropriately because beyond a certain threshold, the accuracy of the finite difference formula degenerates due to an almost zero over almost zero ratio.

As described in Squire et al. [175], one simple remedy is to use complex imaginary increments because

$$Re\frac{f(u+\mathbf{i}\mathrm{d}u) - f(u)}{\mathbf{i}\mathrm{d}u} = Re\frac{f(u+\mathbf{i}\mathrm{d}u)}{\mathbf{i}\mathrm{d}u} = f'(u) - Ref'''(u+\mathbf{i}\theta\mathrm{d}u)\frac{\mathrm{d}u^2}{6}$$

leads to $f'(u) = Re[f(u+\mathbf{i}\mathrm{d}u)/(\mathbf{i}\mathrm{d}u)]$ where the numerator is no longer the result of a difference of two terms. Indeed tests show that the error does not deteriorate when $\mathrm{d}u \to 0$ (figure 1.3). Hence one can choose $\mathrm{d}u = 10^{-8}$ to render the last term with a $O(10^{-16})$ accuracy thus obtaining an essentially exact result.

The cost for using this formula is two evaluations of $f()$, and the programming requires to redefine all `double` as `std::complex` of the Standard Template Library in C++.

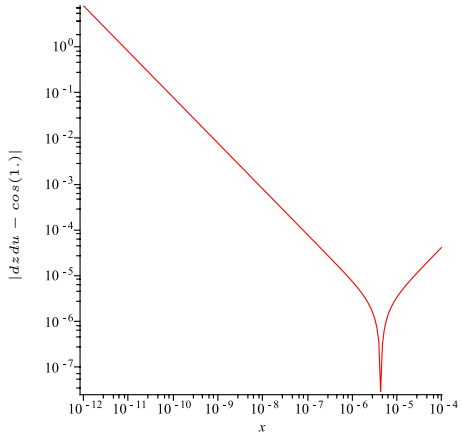

Figure 1.2: Precision (log-log plot of $|dzdu - cos(1.)|$ computed with the forward finite difference formula to evaluate $sin'(u)$ at $u = 1$.

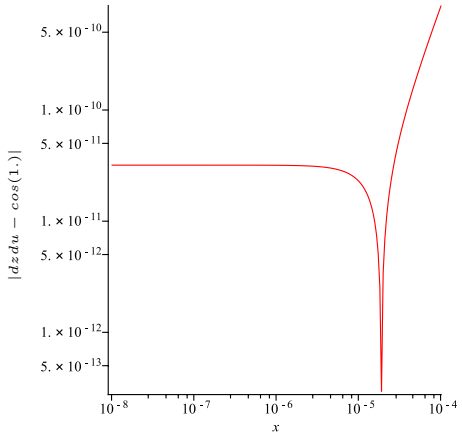

Figure 1.3: Same as Fig. 1.2 but with the finite difference which uses complex increments; both test have been done with Maple-14.

1.3.2 AD in Direct Mode

A conceptually better idea is based on the fact that each line of a computer program is differentiable except at switching points of branching statements like `if` and at zeros of the `sqrt` functions etc.

Denoting by `dx` the differential of a variable `x`, the differential of `a*b` is `da*b+a*db`, the differential of `sin(x)` is `cos(x)dx`, etc... By operator overloading, this algebra can be built into a C++ class, called `ddouble` here on the example 1.3:

```
class ddouble {
        public:
        double val[2];
        ddouble(double a=0, double b=0)
                {
                        val[1]=b;
                        val[0]=a;
                }
        ddouble operator=(const ddouble&a)
                {
                        val[1] = a.val[1];
                        val[0]=a.val[0];
                        return *this;
                }
        ddouble operator-(const ddouble&a, const ddouble&b)
                {
                return ddouble(a.val[0] - b.val[0],
                        a.val[1] - b.val[1]);
                }
        ...
```

```
};
```

Code 1.3: Example of forward class implementation in C++.

So all `ddouble` variables have a 2-array of data: `val[0]` contains the value of the variable and `val[1]` the value of its differential. Notice that the constructor of `ddouble` assigns zero by default to `val[1]`.

To understand how it works, consider the C++ example of figure 1.4 which calls a function $f(u, u_d) = (u - u_d)^2$ for $u = 2$ and $u_d = 0.1$ and where `double` has been changed to `ddouble` and the initialization of `u` implies that its differential is equal to 1. The printing statement displays now the differential of f which is also its derivative with respect to u if all parameters have their differential initialized to 0 except u for which has $du = 1$.

```
ddouble f(ddouble u, ddouble u_d)
{
        ddouble z = u-u_d;
        return z*(u-u_d);
}

int main()
{
        ddouble u=ddouble(2.,1.), u_d = 0.1;

        cout << f(u,u_d).val[0] << endl;
        cout << f(u,u_d).val[1] << endl;

        return 0;
}
```

Code 1.4: A tiny C++ program to compute $(u - u_d)^2$ at $u = 2, u_d = 0.1$ and $\frac{d}{du}(u - u_d)^2$ at $u = 2, u_d = 0.1$.

Writing the class double with all functions and common arithmetic operators is a little tedious but not difficult. An example can be downloaded from
`www.ann.jussieu.fr/pironneau`. The method can be extended to higher order derivatives easily. For second derivatives, for instance, `a.val[4]` will store a, its differentials with respected to the first and second parameter, d_1a, d_2a and the second differential $d_{12}a$ where the two parameters can be the same. The second differential of `a*b` is $a*d_{12}b+d_1a*d_2b+d_2a*d_1b+b*d_{12}a$, and so on.

Notice that $\frac{df}{du_d}$ can also be computed by the same program provided the first line in the `main()` is replaced by `ddouble u=2., u_d=ddouble(0.1,1.);`. However if both derivatives $\frac{df}{du}, \frac{df}{du_d}$ are needed, then, either the program must be run twice or the class ddouble must be modified to handle partial derivatives. In either case the cost of computing n partial derivatives will be approximately n times that of the original program; the reverse mode does not have this numerical complexity and must be used when, say, $n > 5$ if expression templates with traits are used in the direct mode and $n \leq 5$ otherwise Pironneau [154].

1.3.3 AD in Reverse Mode

Consider finding the derivative with respect to θ, $F'(u(\theta),\theta)$ where $F : \mathbb{R}^d \times \mathbb{R}^n \mapsto \mathbb{R}$. Assume that u is the solution of a well posed linear system $Au = B\theta + c$.

The direct differentiation mode applied to the C++ program which implements F will solve the linear system n times at the cost of d^2n operations at least.

The mathematical solution by calculus of variations starts with

$$F'\mathrm{d}\theta = (\partial_\theta F)\mathrm{d}\theta + (\partial_u F)\mathrm{d}u \text{ with } A\mathrm{d}u = B\mathrm{d}\theta,$$

then introduces $p \in \mathbb{R}^d$ solution of $A^T p = (\partial_u F)^T$ and writes

$$(\partial_u F)\mathrm{d}u = (A^T p)^T \mathrm{d}u = p^T B\mathrm{d}\theta \;\Rightarrow\; F' = \partial_\theta F + p^T B.$$

The linear system for p is solved only once, i.e. performing $O(d^2)$ operations at least. Thus, as the linear system is usually the costliest operation, this second method is the most advantageous when n is large.

A computer program only made of assignments can be seen as a triangular linear system for the variables. Loops can be unrolled and seen as assignments and tests, etc. Then, by the above method, the ith line of the program is multiplied by p_i and p is computed from the last line up; but the biggest difficulty is the book-keeping of the values of the variables, at the time p is computed.

In this study we have used the library `adept 1.0` by R.J. Hogan described in Hogan [95]. The nice part of this library is that the calling program for the reverse mode is quite similar to the direct mode presented above; all differentiable variables have to be declared as `ddouble` and the variable with respect to which things are differentiated is indicated at initialization, as above.

1.3.4 Non-Differentiable Functions

In finance, non-differentiability is everywhere. For instance, the second derivative in K of $(x-K)^+$ does not exist at $x = K$ as a function, yet the second derivative of $\int_0^\infty f(x)(x-K)^+dx$ is $f(K)$. Distribution theory extends the notion of derivative: the Heaviside function $H(x) = \mathbb{1}_{x\geq 0}$ has the Dirac mass at zero, $\delta(x)$, for derivative.

In Pagès et al. [142], Automatic differentiation can be extended to handle this difficulty to some degree by approximating the Dirac mass at 0 by

$$\delta^a(x) = \frac{1}{a\sqrt{\pi}} e^{-\frac{x^2}{a^2}}.$$

Now, suppose f is discontinuous at $x = z$ and smooth elsewhere; then

$$f(x) = f^+(x)H(x-z) + f^-(x)(1-H(x-z))$$

hence

$$f'_z(x) = (f^+)'_z(x)H(x-z) + (f^-)'_z(x)(1-H(x-z)) - (f^+(z) - f^-(z))\delta(x-z)$$

Unless this last term is added, the computation of the second order sensitivities will be wrong.

If in the AD library the ramp function x^+ is defined as $xH(x)$ with its derivative to be $H(x)$, if H is defined with its derivative equal to δ^a and if in the program which computes the financial

asset these new functions are used explicitly, like $(x-K)^+ = \text{ramp}(x-K)$ and not $\max(x-K,0)$, then the second derivative in K computed by the AD library will be $\delta^a(x-K)$. Moreover, it will also compute

$$\int_0^\infty f(x)(x-K)^+ dx \approx \frac{1}{N}\sum_{i=1}^{N} f(\xi_i)\delta^a(\xi_i - K)$$

where ξ_i are the N quadrature points of the integral or the Monte−Carlo points used by the programmer to approximate the integral.

However, this trick does not solve all problems and one must be cautious. Moreover, the precision is rather sensitive to the value of a.

Remark 6. *Notice that differentiation by finite difference (FD) is not plagued by this difficulty, which means that FD with complex increment is quite a decent method for first order sensitivities. For second order sensitivities the "very small over very small" indetermination is still present.*

1.4 VAD and the Black Scholes Model

In this section, we implement and test Vibrato+AD (called VAD) and give a conceptual algorithm to implement the method. Let us take the example of a standard European Call option in the Black Scholes model with a volatility σ and the interest rate r possibly function of the parameter θ but not of X. Compared with the previous sections, here $\mu = rX$ and σX was $\sigma(t, X_t)$ before.

1.4.1 Conceptual Algorithm for VAD

To generate M simulation paths with time step $h = \frac{T}{n}$ of the underlying asset X and its tangent process $Y = \dfrac{\partial X}{\partial \theta}$ we need M realizations of a the normal random variable Z; we need also M_Z realizations of Z_n^m, called $\mathbb{Z}_i$, for the last time step for each m. Then given $\bar{X}_0, \bar{Y}_0$, on the algorithm 1:

1.4.2 Greeks

In finance,

- $\Delta := \dfrac{\partial}{\partial X_0}\mathbb{E}[V(X_T)]$ is called the Delta,
- $\Gamma := \dfrac{\partial^2}{\partial X_0^2}\mathbb{E}[V(X_T)]$ is called the Gamma. The Gamma can be important for a Delta-hedging of a portfolio.
- The Vanna is $\dfrac{\partial^2}{\partial X_0 \partial \sigma}\mathbb{E}[V(X_T)]$.

1.4.3 Numerical Test

For the generation of random numbers, we chose the standard Mersenne−Twister generator available in the version 11 of the C++ STL (see detailed algorithm in Matsumoto and Nishimura [129]).

Algorithm 1 VAD for second derivatives using antithetic variance reduction

for $m = 1, .., M$ **do**
  for $k = 0, \dots, n-2$ **do**
    $\bar{X}^m_{k+1} = \bar{X}^m_k + rh\bar{X}^m_k + \bar{X}_k\sigma\sqrt{h}Z^m_{k+1}$
    $\bar{Y}^m_{k+1} = \bar{Y}^m_k + rh\bar{Y}^m_k + r'_\theta h\bar{X}^m_k + \left(\bar{Y}^m_k\sigma + \sigma'_\theta\bar{X}^m_k\right)\sqrt{h}Z^m_{k+1}$
  end for
  for $i = 1, .., M_Z$ **do**
    $X^m_{i,T_\pm} = \bar{X}^m_{n-1} + rh\bar{X}^m_{n-1} \pm \sigma^m\bar{X}_{n-1}\sqrt{h}\mathbb{Z}_i,\ \bar{X}_{i,T_o} = \bar{X}_{n-1} + rh\bar{X}_{n-1}$
    $V^m_{i,T_\pm,} = (\bar{X}^m_{i,T_\pm} - K)^+,\ V^m_{i,T_o} = (\bar{X}^m_{i,T_o} - K)^+.$
    $R = \bar{Y}^m_{n-1}(1+rh) + \bar{X}^m_{n-1}r'_\theta h,\quad S = \bar{Y}^m_{n-1}\sigma + \bar{X}^m_{n-1}\sigma'_\theta$
    $\left(\frac{\partial V_T}{\partial\theta}\right)^m_i = e^{-rT}\left(R\frac{(V^m_{i,T_+} - V^m_{i,T_-})\mathbb{Z}_i}{2\bar{X}^m_{n-1}\sigma\sqrt{h}} + S(V^m_{i,T_+} - 2V^m_{i,T_o} + V^m_{i,T_-})\frac{{\mathbb{Z}_i}^2 - 1}{2\bar{X}^m_{n-1}\sigma}\right)$
  end for
  Compute $\left(\frac{\partial^2 V_T}{\partial\theta^2}\right)^m_i$ by AD on the program that implements $\left(\frac{\partial V_T}{\partial\theta}\right)^m_i$
  $\left(\frac{\partial^2 V_T}{\partial\theta^2}\right)^m = \frac{1}{M_Z}\sum_{i=1}^{M_Z}\left(\frac{\partial^2 V_T}{\partial\theta^2}\right)^m_i$
end for
$\frac{\partial^2 V_T}{\partial\theta^2} = \frac{1}{M}\sum_{m=1}^{M}\left(\frac{\partial^2 V_T}{\partial\theta^2}\right)^m.$

By default, $M = 10^5$, $n = 25$ and $M_Z = 2$. i.e. we simulate only one extra last time step per path; as explained above, the mean over the number of paths allows such a small value for M_Z. For the European vanilla call we tested other values for M, n, M_Z and also a multiple time steps Euler scheme with or without a Brownian bridge.

The parameters considered in the following numerical experiments are $K = 100$, $\sigma = 20\%$ and $r = 5\%$, $T = 1$ year. The initial price X_0 varying from 1 to 200.

1.4.3.1 Preliminary Numerical Test

Here, we focus on the numerical precision of VAD on the Gamma of a standard European Call contract with constant volatility and drift for which there is an analytical Black Scholes formula. Since Vibrato of Vibrato is similar to Vibrato+AD (VAD 2 and Proposition 2) it is pointless to compare the two. However, the computation times are different and naturally Vibrato of Vibrato is faster.

On figure 1.4, the Gammas are compared at $X_0 = 120$; true value of the Gamma is $\Gamma_0 = 0.0075003$. Convergence with respect to the number of paths M is displayed for two values of M_Z.

Confidence intervals are plotted at some points. These are based on the Central Limit Theorem which gives a min and a max for the error as a function of the squares roots of the variance and the squares roots of the number of samples. In this chapter all confidence intervals correspond to 95% error namely

$$\text{Estimated means with N samples } \pm 1.96\sqrt{\frac{\text{estimated variance with N samples}}{\text{N}}}$$

Note however that these intervals do not include the bias.

The method shows a good precision and, of course, it is better to use $M_Z = 2$.

On figure 1.5, we compare the results without and with antithetic variance reduction (1.18). Precision versus M is displayed. Results show that variance reduction is efficient on that test case.

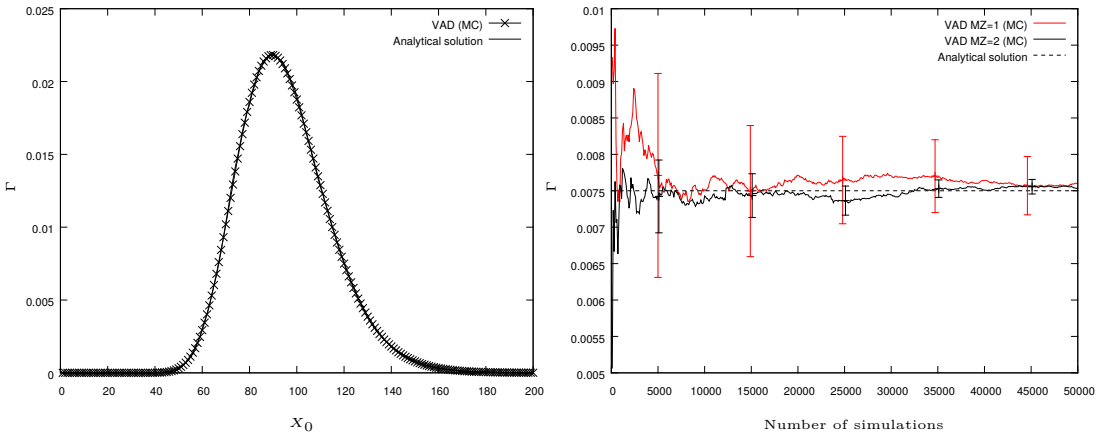

Figure 1.4: On the left the Gamma versus spot price X_0 is displayed when computed by VAD; the analytical exact Gamma is also displayed; both curves overlap. On the right, the Convergence of the Gamma at $X_0 = 120$ is displayed with respect to the number of Monte Carlo samples M. This is done for two values of M_Z (the number of the final time step), $M_Z = 1$ (low curve) and $M_Z = 2$ (upper curve).

The error is also displayed versus M with confidence intervals. Variance reduction is significantly better but requires to perform the M last times step twice. Convergence with respect to M, the number of MC sample, is slow beyond 10 000. Errors are smaller with $M_Z = 2$ compared to $M_Z = 1$. For $M_Z \geq 3$ the error does not decrease so fast. This is because a mean value is taken anyway by the fact that there are M MC samples. So even $M_Z = 1$ is quite acceptable.

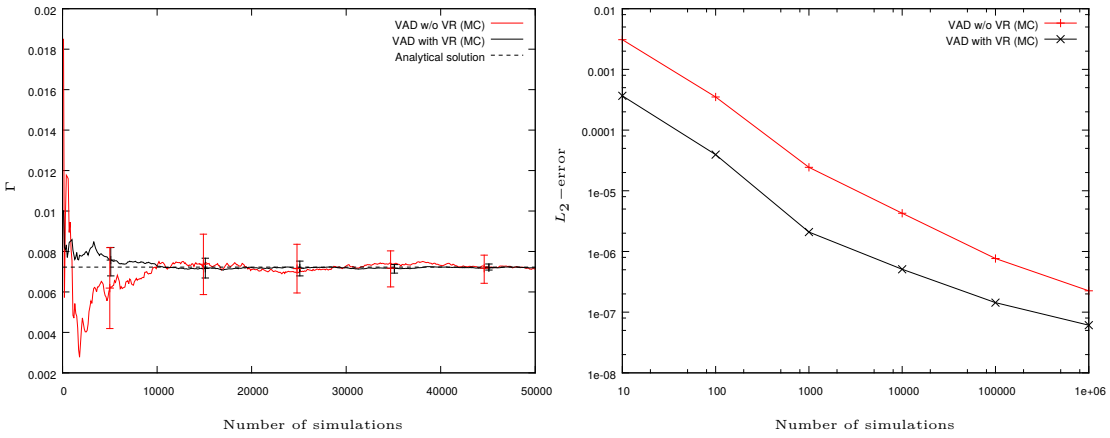

Figure 1.5: On the left the Gamma versus the number of simulation paths is displayed when computed by VAD with and without the variance reduction on Z for $X_0 = 120$; the straight line is the exact solution. On the right, the plot shows the mean error over 256 run of the program using each time a recomputed variance (macro-replication), versus the number of simulation paths M with and without variance reduction.

On figure 1.6 we display versus M the Vanna of an European Call option at $X_0 = 120$,

computed with VAD for $M_Z = 1$ and $M_Z = 2$; 50 time steps were used because the precision was poor with 25. Convergence is similar to the one observed for the Γ.

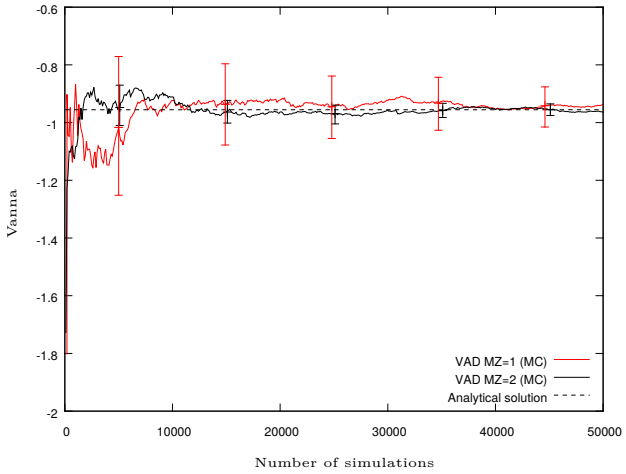

Figure 1.6: Convergence at $X_0 = 120$ of the Vanna versus M for $M_Z = 1$ (lower curve for M small) and $M_Z = 2$ (upper curve).

1.4.3.2 Third Order Derivatives

For third order derivatives, we compute second derivatives by Vibrato of Vibrato (Proposition 2) and differentiate by AD (VVAD). The sensitivity of the Gamma with respect to changes in X_0 is $\partial^3 V/\partial X_0^3$. The sensitivity of the Vanna with respect to changes in the interest rate is $\partial^3 V/\partial X_0 \partial\sigma\partial r$. The parameters of the European Call are the same but the Monte Carlo path number is 10^6 and 50 time steps. The results are displayed on figure 1.7. The convergence is slow; we could not eliminate the small differences between the analytical solution and the approximation by increasing the number of paths.

1.4.3.3 Ramp Function and High Order Derivatives

As explained in Section 1.3.4, it is possible to handle the non-differentiability of the function $(x-K)^+$ at $x = K$ by using approximated Dirac masses and a predefined ramp function in the AD library. We illustrate this technique with a standard European Call option in the Black Scholes model.

We computed the Gamma and the sixth derivative with respect to X_0 entirely by AD without using Vibrato. For the first derivative, the parameter a does not play an important role but, as we evaluate higher derivatives, a good choice of the parameter a becomes crucial; it requires a bigger M to catch the Dirac approximations with small a. Currently the choice of a is experimental and depends on X_0; in conclusion one can say that such a procedure could not be used in industry; 4^{th} derivative by AD over Vibrato of Vibrato was not tested.

The parameters are the same but the maturity is $T = 5$ for the Gamma and $T = 0.2$ for the sixth derivative with respect to X_0 and $M = 10^5$ with 25 time steps and $M_Z = 2$. The results are displayed on figure 1.8.

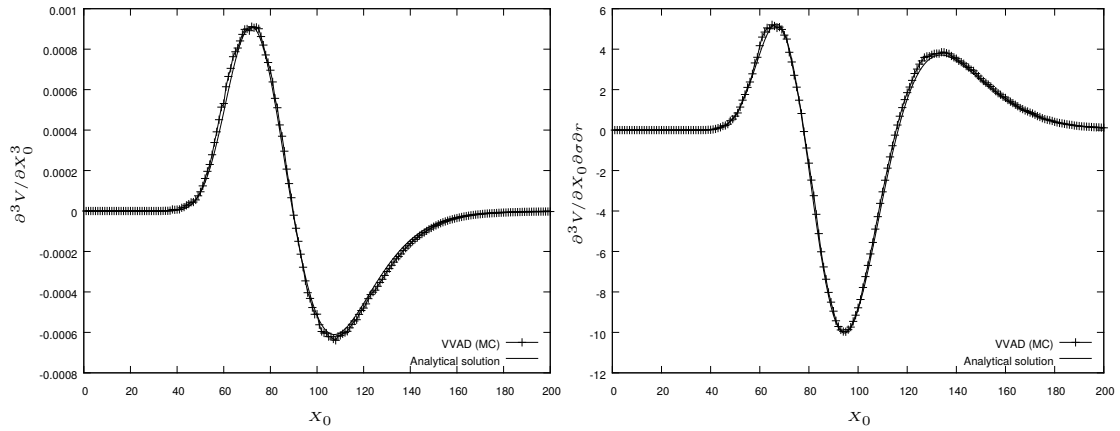

Figure 1.7: On the left $\partial^3 V/\partial X_0^3$ versus Price is displayed when computed by VVAD; the analytical exact curve is also displayed; both curves practically overlap. On the right, the plot shows the same for $\partial^3 V/\partial X_0 \partial \sigma \partial r$. Notice that errors are visible even though $M = 10^6$ and 50 time steps are used with $M_Z = 2$.

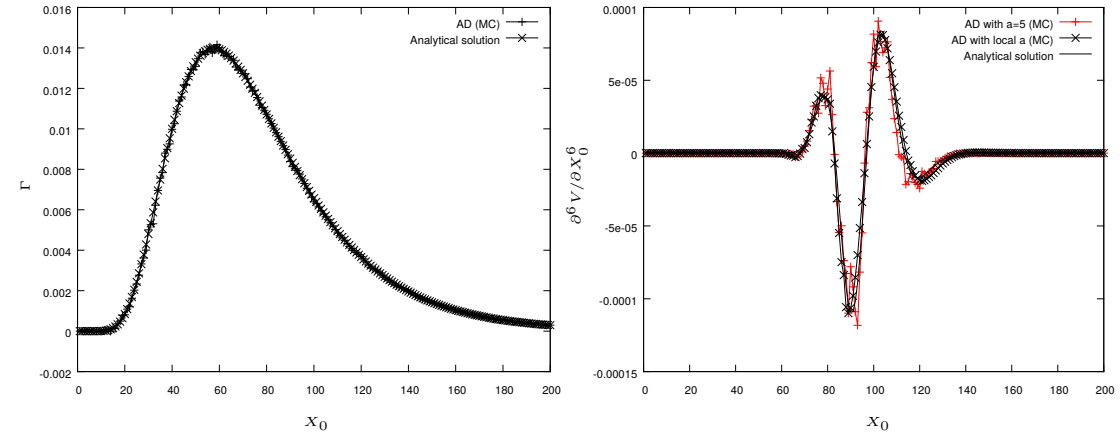

Figure 1.8: On the left the Gamma versus X_0 is displayed when computed by AD with the ramp function (with $a = 1$) and without Vibrato; the analytical exact Gamma is also displayed; both curves overlap. On the right, the sixth derivative versus X_0 is displayed when computed entirely by AD; the analytical solution is also displayed. An ad-hoc value for the parameter a was used function of X_0 : $a \in (1, 5)$.

For the Gamma, the exact and computed curves are overlapping but for the sixth derivative with respect to the parameter X_0, it did not work with a constant parameter a; so we chose an X_0 dependent parameter a; then the curves are practically overlapping. Admittedly this is a dishonest patch, but it shows that the method has the potential to give the right result.

1.4.4 Basket Call Option

A Basket option is a multidimensional derivative security whose payoff depends on the value of a weighted sum of several risky underlying assets.

As before, X_t is given by (1.3). But now $(W_t)_{t\in[0,T]}$ is a d-dimensional *correlated* Brownian motion with $\mathbb{E}[dW_t^i dW_t^j] = \rho_{i,j}dt$.

To simplify the presentation, we assume that r and $\sigma^*_{i,j}$ are real constants and the payoff is given by

$$V_T = e^{-rT}\mathbb{E}[(\sum_{i=1}^{d}\omega_i X_T^i - K)^+] \tag{1.36}$$

where $(\omega_i)_{i=1,\dots,d}$ are positive weights with $\sum_{i=1}^d \omega_i = 1$. Here, we chose to compare three different methods: 1/ the reference values comes from an approximated moment-matching dynamics (Levy [121] and in Brigo et al. [33]), 2/ VAD and 3/ second order finite difference (FD).

As the computing time would be too long, we make use of the fact that r and σ are constant and set $\bar{X}_T$ by

$$\begin{cases} \bar{X}_T^i = X_0^i e^{rT}\exp\left(-\dfrac{1}{2}\displaystyle\sum_{j=1}^{d}|\sigma^*_{ij}|^2 T\right) \\ \bar{X}_{T\pm}^i = X_0^i e^{rT}\exp\left(-\dfrac{1}{2}\displaystyle\sum_{j=1}^{d}|\sigma^*_{ij}|^2 T \pm \sum_{j=1}^{d}\sigma^*_{ij}\sqrt{T}Z_j\right), \quad i=1,\dots,d, \\ V_{T_\pm} = e^{-rT}(\displaystyle\sum_{i=1}^{d}\omega_i X_{T_\pm}^i - K)^+, \\ V_T = e^{-rT}(\displaystyle\sum_{i=1}^{d}\omega_i X_T^i - K)^+, \end{cases} \tag{1.37}$$

where Z denotes an $\mathcal{N}(0; I_d)$ random vector.

For each realization of Z (a Monte−Carlo path), with $C = \Sigma\Sigma^{\mathbf{T}}$, we compute the Delta by antithetic Vibrato:

$$\begin{aligned} \Delta &= \left(\frac{\partial\mu}{\partial X_0^i}\right)^T \frac{1}{2}(V_{T_+} - V_{T_-})C^{-T}Z \\ &\quad + \frac{1}{4}(V_{T_+} - 2V_T + V_{T_-})\frac{\partial\Sigma}{\partial X_0^i} : C^{-T}(ZZ^T - I_d)C^{-1} \end{aligned} \tag{1.38}$$

Then we compute the mean of the resulting vector over M paths. Finally we apply Automatic Differentiation to the above to obtain the Gamma.

1.4.4.1 Reference Solution by Approximated Moment-Matching Dynamics

From Brigo et al. [33], moment matching constructs a reduced order system by matching as many moments as possible of the original function to the moments of the associated transfer function. The average basket

$$Y_t = \omega \cdot X_t \tag{1.39}$$

is approximated by $\tilde{Y}_t$ solution of

$$d\tilde{Y}_t = r\tilde{Y}_t dt + \tilde{\sigma}\tilde{Y}_t dW_t, \quad \tilde{Y}_0 = \omega \cdot X_0. \tag{1.40}$$

Where $\tilde{\sigma}$ is adjusted in terms of σ, ρ and X. Consequently

$$\begin{cases} \tilde{Y}_t = Y_0 \exp\left((r - \frac{1}{2}\tilde{\sigma}^2)t + \tilde{\sigma}W_t\right), \\ \mathbb{E}[\tilde{Y}_t] = Y_0 e^{rt}, \\ \mathbb{E}[\tilde{Y}_t^2] = Y_0^2 \exp\big((2r + \tilde{\sigma}^2)t\big). \end{cases} \tag{1.41}$$

On the other hand, the second order moments of the basket are

$$\mathbb{E}[Y_t^2] = e^{2rt} X_0^T \tilde{C}_t X_0, \quad \text{where} \quad \tilde{C}_t^{\,ij} = \omega_i \omega_j \exp\left(\rho_{i,j}\sigma_i\sigma_j t\right). \tag{1.42}$$

Hence, $\tilde{\sigma}$ is adjusted so that

$$\mathbb{E}[Y_t^2] = \mathbb{E}[\tilde{Y}_t^2]. \tag{1.43}$$

Therefore,

$$\tilde{\sigma}^2 = \frac{1}{t}\left(\ln X_0^T \tilde{C}_t X_0 - \ln\left(\omega \cdot X_0\right)^2\right). \tag{1.44}$$

Accordingly

$$\tilde{V}_T = e^{-rT}\mathbb{E}\left[(\tilde{Y}_T - K)^+\right]. \tag{1.45}$$

As the process $(\tilde{Y}_t)_{t\in[0,T]}$ follows a log-normal distribution, we have an analytical expression for (1.45):

$$\begin{aligned} &\tilde{V}_T = Y_0\Phi(d_{\tilde{\Lambda}_+}) - e^{-rT}K\Phi(d_{\tilde{\Lambda}_-}) \text{ with } \Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{\frac{-t^2}{2}}dt, \\ &d_{\tilde{\Lambda}_+} = \frac{\ln\left(Y_0/K\right) + (r + \frac{1}{2}\tilde{\sigma}^2)T}{\tilde{\sigma}\sqrt{T}}, \quad d_{\tilde{\Lambda}_-} = d_{\tilde{\Lambda}_+} - \tilde{\sigma}\sqrt{T}. \end{aligned} \tag{1.46}$$

From this formula, we can compute the Delta and the Gamma with respect to Y_0 and assign the components of $(X_{i0})_{i=1,\dots,d}$ from the weighted sum. For instance

$$\tilde{\Delta}_{Y_0} = \Phi(d_{\tilde{\Lambda}_+}), \text{ hence } \Delta_{X_{i0}} = \omega_i\Phi(d_{\tilde{\Lambda}_+}), \quad i = 1,\dots,d. \tag{1.47}$$

For the Gamma,

$$\tilde{\Gamma}_{Y_0Y_0} = \frac{1}{Y_0\tilde{\sigma}\sqrt{T}}\phi(d_{\tilde{\Lambda}_+}), \text{ hence } \Gamma_{X_{i0}X_{i0}} = \omega_i\zeta_i\phi(d_{\tilde{\Lambda}_+}), \quad i = 1,\dots,d, \tag{1.48}$$

with $\phi(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$ and, for $i = 1,\dots,d$

$$\zeta_i = \frac{1}{(\tilde{\sigma}\sqrt{T})^2}\left(\left(\frac{w_i}{\omega \cdot X_0} + \frac{\beta_i}{2}T\right)\tilde{\sigma}\sqrt{T} - \frac{\beta_i}{2\sqrt{\tilde{\sigma}}}\sqrt{T}d_{\tilde{\Lambda}_+}\right), \quad \beta_i = \frac{1}{T}\frac{\alpha_i\left(\omega \cdot X_0\right)^2}{X_0^T\tilde{C}_T X_0} \tag{1.49}$$

and

$$\alpha_i = \frac{2(\tilde{C}_T X_0)_i}{\left(\omega \cdot X_0\right)^2} - 2\omega_i\frac{X_0^T\tilde{C}_T X_0}{\left(\omega \cdot X_0\right)^3}. \tag{1.50}$$

1.4.4.2 Procedure to Compute the Gamma of a Basket option

Here is a procedure to compute the Gamma of a Basket option (average of basket of underlying) in the multidimensional Black Scholes model. We make use of the fact that r and σ are constant.

1. Generate M simulation paths using a one time step scheme, for $1 < m < M$

$$\left(\bar{X}^i_{T\pm}\right)^m = X^i_0 e^{rT} \exp\left(-\frac{1}{2}\sum_{j=1}^{d}|\sigma^*_{ij}|^2 T \pm \sum_{j=1}^{d}\sigma^*_{ij}\sqrt{T}Z^m_j\right), \quad i = 1,\dots,d,$$

and

$$\left(\bar{X}^i_{T_o}\right)^m = X^i_0 e^{rT} \exp\left(-\frac{1}{2}\sum_{j=1}^{d}|\sigma^*_{ij}|^2 T\right),$$

where Z denotes an $\mathcal{N}(0; I_d)$-distributed random vector.

2. For each simulation path, $C = \Sigma\Sigma^{\mathbf{T}}$, compute for $i = 1,\dots,d$

$$\begin{aligned}\left(\frac{\partial V_T}{\partial X^i_0}\right)^m &= \left(\frac{\partial \mu}{\partial X^i_0}\right)^T \frac{1}{2}(V^m_{T_+} - V^m_{T_-})C^{-T}Z \\ &\quad + \frac{1}{4}(V^m_{T_+} - 2V^m_{T_o} + V^m_{T_-})\frac{\partial \Sigma}{\partial X^i_0} : C^{-T}(ZZ^T - I_d)C^{-1},\end{aligned}$$

with

$$V^m_{T_\pm} = e^{-rT}(\sum_{i=1}^{d}\omega_i(X^i_{T_\pm})^m - K)^+,$$

and

$$V^m_{T_o} = e^{-rT}(\sum_{i=1}^{d}\omega_i(X^i_{T_o})^m - K)^+.$$

3. Apply an Automatic Differentiation method on step (2) to compute the Gamma.

4. Compute the mean over M simulation paths.

$$\frac{\partial^2 V_T}{\partial X^2_0} = e^{-rT}\frac{1}{M}\sum_{m=1}^{M}\left(\frac{\partial^2 V_T}{\partial X^2_0}\right)^m.$$

1.4.4.3 Numerical Test

In this numerical test $d = 7$ and the underlying asset prices are:

$$X_0{}^T = (1840, 1160, 3120, 4330.71, 9659.78, 14843.24, 10045.40). \tag{1.51}$$

The volatility vector is:

$$\sigma^T = (0.1460, 0.1925, 0.1712, 0.1679, 0.1688, 0.2192, 0.2068). \tag{1.52}$$

The correlation matrix is

$$\begin{pmatrix} 1.0000 & 0.9477 & 0.8494 & 0.8548 & 0.8719 & 0.6169 & 0.7886 \\ 0.9477 & 1.0000 & 0.7558 & 0.7919 & 0.8209 & 0.6277 & 0.7354 \\ 0.8494 & 0.7558 & 1.0000 & 0.9820 & 0.9505 & 0.6131 & 0.9303 \\ 0.8548 & 0.7919 & 0.9820 & 1.0000 & 0.9378 & 0.6400 & 0.8902 \\ 0.8719 & 0.8209 & 0.9505 & 0.9378 & 1.0000 & 0.6417 & 0.8424 \\ 0.6169 & 0.6277 & 0.6131 & 0.6400 & 0.6417 & 1.0000 & 0.5927 \\ 0.7886 & 0.7354 & 0.9303 & 0.8902 & 0.8424 & 0.5927 & 1.0000 \end{pmatrix}. \tag{1.53}$$

The number of Monte Carlo paths varies from 1 to 10^6 using (1.37) for the time integration. Errors are calculated with reference to a solution computed by approximate moment matching.

On figures 1.9 , convergence is shown versus M for the computation of the Gamma of a Basket made of the first 4 assets (left) and then for the full 7 assets (right); VAD in direct mode is used and compared with brute force FD (1% shift of the initial price) applied the Monte Carlo algorithm which computes the option. Convergence speed of these methods is almost the same (with a slight advantage for the FD). Confidence intervals are also similar.

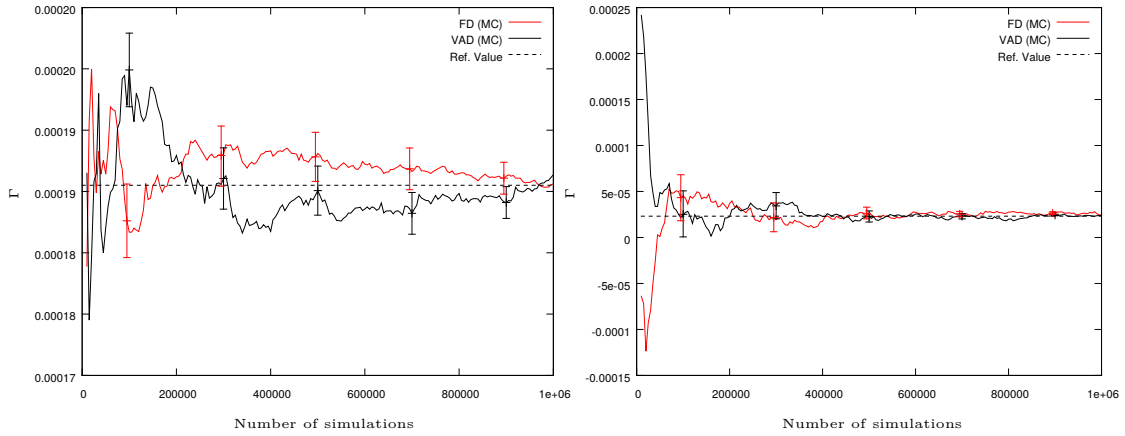

Figure 1.9: Convergence is shown versus M of the Gamma of a Basket option when $d = 4$ (on the left) and $d = 7$ (on the right) by VAD and via FDs. Maturity is $T = 0.1$.

More details are given in Tables 1.7 and 1.8 for the 7 assets Basket. Table 1.7 displays prices. Here too the FD has a slight advantage. Observe however that precision is always beyond 3 digits.

Table 1.8 displays the CPU time for an increasing number of assets in the basket ranging from 1 to 7. Beyond $d = 3$ the FD method is one third more expensive. Again, the method is slightly more accurate than VAD.

1.4.5 Autocallable

As defined in Alm et al. [6], an autocall (also called Autocallable) can be described as a set of prespecified observation dates (with a corresponding set of constant payoff), a barrier style and a redemption payment at the maturity of the contract. The Autocallable will be called automatically if the conditions of its barrier style are fulfilled on one of the predefined dates of

the set. More precisely, at each observation date it is checked whether the underlying asset price (or a basket of underlying asset prices) reached a certain barrier style (which is predefined at the time of signature of the contract). If the conditions are fulfilled, the holder of the Autocallable receives a prespecified constant cash-flow and it terminates the contract. Otherwise, the contract continues to exist until the next observation date and the process is restarting. In the case where the Autocallable survives until its maturity, a redemption payment which is depending on the performance of the underlying asset price (or a basket of underlying asset prices) at maturity is paid to the holder of the contract. On figure 1.10, a schematic representation of the payoff of an Autocallable.

By purchasing an Autocallable, the investor benefits from an attractive fixed return if the forecast view is correct and it is important to note that whatever happens, the investor always get a payout. For a prespecified set of observation dates $(t_i)_{i=1,\dots,n}$ with corresponding con-

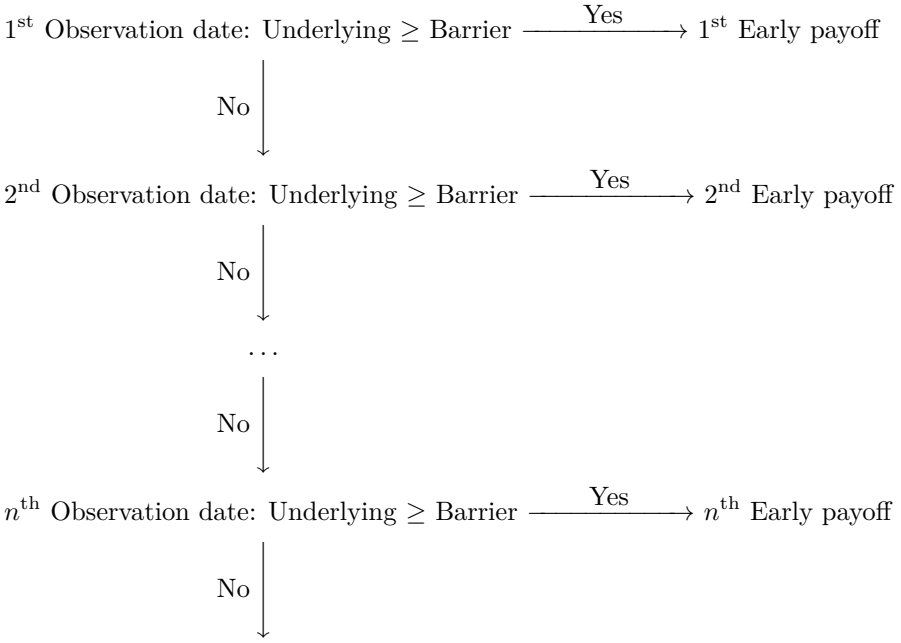

Figure 1.10: Schematic representation of the payoff of an Autocallable.

stant payoffs $(C_{t_i})_{i=1,\dots,n}$ and a redemption payment function R, the discounted payoff of an Autocallable can be expressed as:

$$V(X_{t_1},\dots,X_{t_n}) = \begin{cases} e^{-rt_j} C_{t_j}, & \text{For the first } j \text{ such that } \; X_{t_i} < B \leq X_{t_j} \quad \forall i < j, \\ e^{-rt_n} R\left(X_{t_n}\right), & \text{If } \; X_{t_j} < B \qquad\qquad\qquad\qquad\qquad\quad \forall j = 1,\dots,n. \end{cases} \tag{1.54}$$

Where X_{t_i} denotes the value of underlying asset price at each observation date and B denotes the value of the barrier. So the price of an Autocallable is:

$$V_{t_n} = \mathbb{E}[V(X_{t_1},\dots,X_{t_n})]. \tag{1.55}$$

And the standard Monte Carlo estimator for the price of an Autocallable is given by:

$$\frac{1}{M}\sum_{i=1}^{M} V(X^i_{t_1},\dots,X^i_{t_n}) \tag{1.56}$$

In this test case, the underlying is following a one dimension geometric Brownian motion as described in 1.3.

We apply our method to a simple Monte Carlo pricing algorithm (derived from the previous formulas) but the reference values has been obtained via a stable scheme based on a combination of Glasserman and Staums one-step survival method (OSS) as described in Glasserman and Staum [72], Delta's and Gamma's are calculated by finite differences. We recall briefly here the method: the idea is to sample only the paths which stay below the barrier for all observations dates. Obviously, it is not necessary to calculate $X_{t_{j+1}}$ if the Autocallable has been cancelled at the previous observation date i.e $X_{t_j} \geq B$. Therefore, for calculating each path $(X_{t_1},\dots,X_{t_n})$ using 1.3 requires at most m samples z_{t_j} from the standard normally distributed variable Z_{t_j}. These samples can be generated by $z_{t_j} = \Phi^{-1}(u_{t_j})$, and Φ is the cumulative standard normal distribution. The one-step survival technique is sampling only paths which stay below the barrier B for all observation dates $(t_i)_{i=1,\dots,n}$. In our case, this can be achieving by changing each step of the simulation path by, for every $j = 0,\dots,n-1$

$$z_{t_j} := \Phi^{-1}(p_{t_j}u_{t_j}), \quad p_{t_j} := \Phi\left(\frac{\ln\left(B/X_{t_j}\right) - \left(r - \frac{\sigma^2}{2}\right)(t_{j+1}-t_j)}{\sigma\sqrt{t_{j+1}-t_j}}\right), \quad u_{t_j} \sim \mathcal{U}(0,1) \tag{1.57}$$

Then we are only sampling the argument $p_{t_j}u_{t_j}$ from a uniform distribution on the restricted interval $(0, p_{t_j})$. We denotes $p_{t_j} = \mathbb{P}(X_{t_{j+1}} < B \mid X_{t_j} = X_{t_j})$ the probability that the underlying stays below the barrier B at the next step. Consequently, the sampling Z_{t_j} is given by a truncated standard normal distribution

$$Z_{t_j} < \frac{\ln\left(B/X_{t_j}\right) - \left(r - \frac{\sigma^2}{2}\right)(t_{j+1}-t_j)}{\sigma\sqrt{t_{j+1}-t_j}}. \tag{1.58}$$

But this will bias the result so we have to correct for the missing barrier hits which happen with probability $(1-p_j)$. We can offset this fact by adding a weighted $(j+1)-th$ premature payoff to the total payoff for each sample path and all further payoffs have to be weighted by p_{t_j}. Hence, the value of an Autocallable is

$$V_{t_n} = \mathbb{E}[\tilde{V}(X_{t_1},\dots,X_{t_n})]. \tag{1.59}$$

And the Monte Carlo estimator based on the one-step survival technique is

$$\frac{1}{M}\sum_{i=1}^{M} \tilde{V}(X^i_{t_1},\dots,X^i_{t_n}), \tag{1.60}$$

With the modified payoff established by

$$\begin{aligned}
&\tilde{V}(X_{t_1},\dots,X_{t_n}) \\
&\quad := (1-p_0)e^{-r(t_1-0)}C_{t_1} + p_0\Big[(1-p_{t_1})e^{-r(t_2-0)}C_{t_2} \\
&\quad + p_{t_1}\Big[(1-p_{t_2})e^{-r(t_3-0)}C_{t_3} + \dots + p_{n-2}\Big[(1-p_{t_{n-1}})e^{-r(t_n-0)}C_{t_n} \\
&\quad + \ p_{t_{m-1}}e^{-r(t_n-0)}R(X_{t_n}/B)\Big]\dots\Big]\Big]
\end{aligned} \tag{1.61}$$

So

$$\tilde{V}(X_{t_1},\dots,X_{t_n}) = G_{t_n} e^{-r(t_m-0)} R(X_{t_n}/B) + \sum_{j=0}^{n-1} G_{t_j}(1-p_{t_j}) e^{-r(t_{j+1}-0)} C_{t_{j+1}}, \tag{1.62}$$

With

$$G_{t_j} := \prod_{i=0}^{j-1} p_{t_i}. \tag{1.63}$$

For more details, the algorithm above is done explicitly in Alm et al. [6].

1.4.5.1 Procedure to Compute the Gamma of an Autocallable

We describe here a procedure to compute the Gamma of an Autocallable using a simple Monte Carlo pricing algorithm (with the same notation as in (1.4.1)) in the Black Scholes model.

1. Generate M simulation paths following the procedure given on the algorithm 1 but we only simulate the underlying asset price at the maturity.
2. Compute for each simulation path, for $1 < m < M$
   (a) Simulate all previous $n-1$ observations dates using recursive Brownian bridge construction (using the construction given in London [123]). Iterate from $k = n-1,\dots,1$ the following computation steps (with step size $h = \frac{T}{n}$ for the Euler scheme)
      i. Construct
      $$Z_{i+1}^{*,m} = Z_i^{*,m} + \sqrt{h} Z_i^m, \quad Z_0^{*,m} = 0, \quad i = 1,\dots,k-1$$
      where $(Z_i)_{1\le i\le k}$ denotes a sequence of $\mathcal{N}(0;1)$-distributed random variables given by $Z_i := \sqrt{\frac{1}{h}}(W_i^* - W_{i-1}^*)$

      Remark 7. *As we generate directly the underlying asset price $\bar{X}$ at the maturity, we do not need to compute the tangent process in the case of the computation of the Gamma.*
      ii. Then, compute from $k = n-1,\dots,1$,
      $$\bar{X}_k^m = X_0\left(1 + \frac{\bar{X}_{(k+1)}^m}{X_0}\frac{kh}{(k+1)h} + \sigma\left(Z_k^{*,m} - \frac{kh}{(k+1)h} Z_{(k+1)}^{*,m}\right)\right)$$
   (b) Compute the payoff of the Autocallable
   $$V_T^m = \begin{cases} e^{-rt_k} C_k, & \text{For the first } k \text{ such that } \bar{X}_j^m < B \le \bar{X}_k^m \quad \forall j < k, \\ e^{-rT} R\left(\bar{X}_T^m\right), & \text{If } \bar{X}_k^m < B \qquad\qquad \forall k = 1,\dots,m. \end{cases}$$
   (c) Compute the Delta of an Autocallable using the Vibrato method at maturity from 0 with the following formula, here $M_Z = 1$,
   $$\left(\frac{\partial V_T}{\partial X_0}\right)^m = (1+r\tau)\, V_T \frac{Z^m}{X_0\sigma\sqrt{\tau}} + \sigma\sqrt{\tau} V_T \frac{(Z^m)^2 - 1}{X_0\sigma\sqrt{\tau}},$$
   where τ corresponds to the maturity of exercised payment.

(d) Apply an Automatic Differentiation method from step (2c) to compute the Gamma of the Autocallable.

3. Compute the mean over the M simulation paths.

$$\frac{\partial^2 V_T}{\partial X_0^2} = \frac{1}{M}\sum_{m=1}^{M}\left(\frac{\partial^2 V_T}{\partial X_0^2}\right)^m.$$

1.4.5.2 Numerical Test

In this numerical test, the parameters for the Autocallable are: the value of the barrier $B = 4000$, the risk free rate $r = 5\%$, the volatility of the underlying $\sigma = 20\%$. We compare the value of the delta and the gamma of an Autocallable for different maturities $T \in [1, 3]$, number of observation dates $n \in [1, 3]$ and level of the underlying $X_0 \in [3000, 4000]$. The early payoffs are settled to

$$C_{t_k} = (1 + 0.1k)a, \quad a = 100, \quad k = 1, \ldots, 3. \tag{1.64}$$

And the redemption payoff which depends on the final performance of the underlying price is given to

$$R(x) = ax, \quad a = 100. \tag{1.65}$$

The Monte Carlo parameters are set to 1 000 000 simulation paths and the number of time steps are equal to the number of the observation dates.

On the table 1.9, we display the results for the approximation of the Gamma of an Autocallable versus the number of simulation paths for the coupling of Vibrato to a direct Automatic differentiation method; the Finite differences (1% shift of the initial price) on the optimized (OSS) and standard Monte Carlo algorithm are also displayed. The results obtained with the Finite differences on OSS method and our method are very close. The method provides a pretty good accuracy for the different set of parameters except when the underlying is too low.

1.5 Computation with Vibrato plus Reverse AD (VRAD)

If several Greeks are requested at once then it is better to use AD in reverse mode. To illustrate this point, we proceed to compute all second derivatives, sometimes referred as the Hessian matrix of the Call option:

$$\begin{pmatrix} \frac{\partial^2 V}{\partial X_0^2} & \frac{\partial^2 V}{\partial \sigma \partial X_0} & \frac{\partial^2 V}{\partial r \partial X_0} & \frac{\partial^2 V}{\partial T \partial X_0} \\ \frac{\partial^2 V}{\partial X_0 \partial \sigma} & \frac{\partial^2 V}{\partial \sigma^2} & \frac{\partial^2 V}{\partial \sigma \partial r} & \frac{\partial^2 V}{\partial T \partial v} \\ \frac{\partial^2 V}{\partial X_0 \partial r} & \frac{\partial^2 V}{\partial \sigma \partial r} & \frac{\partial^2 V}{\partial r^2} & \frac{\partial^2 V}{\partial T \partial r} \\ \frac{\partial^2 V}{\partial X_0 \partial T} & \frac{\partial^2 V}{\partial v \partial T} & \frac{\partial^2 V}{\partial r \partial T} & \frac{\partial^2 V}{\partial T^2} \end{pmatrix}. \tag{1.66}$$

It is easily seen that a FD procedure will require at least 33 evaluations of the original pricing function whereas we only call this function once if AD is used in reverse mode. Furthermore, we have to handle 4 different perturbation parameters. So in principle VRAD should be an order of magnitude faster.

The parameters are $X_0 = 90,\ K = 100,\ \sigma = 20\%,\ r = 5\%$ and $T = 1$. The number of Monte Carlo path is 200 000 with 50 time steps. We used the library `adept 1.0` for the reverse mode. On the computer programming side we have a single formula to compute all the Greeks.

The computing time is shown in the table 1.1, clearly the reverse automatic differentiation combined with Vibrato is almost 4 times faster than the FD procedures (to observe an order of magnitude speed-up we would need to compute more derivatives see Naumann [137])

Mode	FD (MC)	VRAD (MC)
Time (sec)	2.01	0.47

Table 1.1: CPU time (in seconds) to compute the Hessian matrix of a standard European Call option (considering $X_0,\ \sigma,\ r,\ T$ as variables) in the Black Scholes model.

In Tables 1.3,1.4,1.5 and 1.6 the analytical solutions are compared with the VAD solutions and also the FD solutions (with increment $\varepsilon = 0.01$). The computing time for each method is also given. In Table 1.3 all second order derivatives are given at maturity $T = 1$ for different values of M at spot price $X_0 = 90$ and in Table 1.5 for different values of X_0 with $M = 10^4$. The RMSE is given for these two sets of computations in Tables 1.4 and 1.6. The RMSE of the gamma is:

$$\text{RMSE} = \sqrt{\mathbb{E}\left[\left(\frac{\partial^2}{\partial\theta^2}V(X_T) - \frac{\partial^2}{\partial\theta^2}\mathbb{E}[V(X_T)]\right)^2\right]} \tag{1.67}$$

The standard error, used below is the squares of the RMSE. Notice that the convergence is slow both with respect to M and with respect to the time steps.

The purpose of Tables 1.5 and 1.6 is to show that the performances are identical at the money in the money and out of the money.

1.6 American Option

Recall that an American option is like an European option which can be exercised at any time before maturity. The value V_t of an American option requires the best exercise strategy. Let φ be the payoff, then

$$V_t := \operatorname*{ess\,sup}_{\tau\in\mathcal{T}_t} \mathbb{E}[e^{-r(\tau-t)}\varphi(X_\tau) \mid X_t] \tag{1.68}$$

where $\mathcal{T}_t$ denotes the set of $[t, T]$-valued stopping times (with respect to the (augmented) filtration of the process $(X_s)_{s\in[0,T]}$).

Consider a time grid $0 < t_1 < \cdots < t_n = T$ with time step h, i.e. $t_k = kh$. To discretize the problem we begin by assuming that the option can be exercised only at $t_k,\ k = 0, .., n$; its value is defined recursively by

$$\begin{cases} \bar{V}_{t_n} = e^{-rT}\varphi(\bar{X}_T) \\ \bar{V}_{t_k} = \max\limits_{0\le k\le n-1}\left(e^{-rt_k}\varphi(\bar{X}_{t_k}), \mathbb{E}[\bar{V}_{t_{k+1}} \mid \bar{X}_{t_k}]\right), \end{cases} \tag{1.69}$$

1.6.1 Longstaff Schwartz Algorithm

Following Longstaff and Schwartz [124] let the continuation value $C_{t_k} = \mathbb{E}[e^{-rh}\bar{V}_{t_{k+1}} \mid \bar{X}_{t_k}]$ as X is a Markov process. The holder of the contract exercises only if the payoff at t_k is higher than

the continuation value C_{t_k}. The continuation value is approximated by a linear combination of a finite set of R real basis functions ψ:

$$C_k \simeq \sum_{i=1}^{R} \alpha_{k,i} \psi_{k,i}(\bar{X}_{t_k}). \tag{1.70}$$

Typically, the $(\alpha_{k,i})_{i=1,\dots,R}$ are computed by least squares,

$$\min_{\alpha} \left\{ \mathbb{E}\left[\left(\mathbb{E}[e^{-rh} \bar{V}_{t_{k+1}} \mid \bar{X}_{t_k}] - \sum_{i=1}^{R} \alpha_{k,i} \psi_{k,i}(\bar{X}_{t_k}) \right)^2 \right] \right\}. \tag{1.71}$$

This leads to a Gram linear system

$$\sum_{j=1}^{R} \alpha_{k,i} \mathbf{Gram}\left\{ \psi_{k,i}(\bar{X}_{t_k}), \psi_{k,j}(\bar{X}_{t_k}) \right\} = \mathbb{E}[\mathbb{E}[e^{-rh} V_{k+1} \mid X_{t_k}] \psi_{k,i}(\bar{X}_{t_k})], \quad i = 1, \dots, R. \tag{1.72}$$

Remark 8. *Once the optimal stopping time is known, the differentiation with respect to θ of (1.69) can be done as for an European contract. The dependency of the τ^* on θ is neglected; arguably this dependency is second order but this point needs to be validated.*

1.6.2 Procedure to Compute the Gamma of an American option

Here, we describe a procedure to compute the Gamma of an American Put option in the Black Scholes model.

1. Generate M simulation paths of an Euler scheme with n time steps of size $h = \frac{T}{n}$. For $1 < m < M$ and $0 < k < n-1$

$$\begin{cases} \bar{X}^m_{k+1} = \bar{X}^m_k + rh\bar{X}^m_k + \bar{X}^m_k \sigma \sqrt{h} Z^m_{k+1}, & \bar{X}^m_0 = X_0, \\ \bar{Y}^m_{k+1} = \bar{Y}^m_k + rh\bar{Y}^m_k + \bar{Y}^m_k \sigma \sqrt{h} Z^m_{k+1}, & \bar{Y}^m_0 = 1. \end{cases}$$

2. For each simulation path

   (a) Simulate M_Z replications of the payoff, for $1 < i < M_Z$

$$(V_{T_\pm})^m_i = (\bar{X}^m_{i,T_\pm} - K)^+, \quad (V_{T_o})^m_i = (\bar{X}^m_{i,T_o} - K)^+$$

   with

$$\bar{X}^m_{i,T_\pm} = \bar{X}^m_{n-1} + rh\bar{X}^m_{n-1} \pm \bar{X}^m_{n-1} \sqrt{\bar{\mathcal{V}}^m_{n-1}} \sqrt{h} \mathbb{Z}_i, \quad \bar{X}^m_{i,T_o} = \bar{X}^m_{n-1} + rh\bar{X}^m_{n-1}.$$

   (b) Compute the Gamma of the terminal condition using (6) in section (1.4.1)

3. Iterate from $n-1$ to 1 and perform the following at the k-th time step.

   (a) Solve the Gram linear system (1.72).

   (b) Calculate the continuation value of each path.

$$C_{k+1}(\bar{X}^m_k) = \sum_{i=1}^{R} \alpha_{k,i} \psi_i(\bar{X}^m_k).$$

(c) Compute the Gamma by differentiating the Vibrato formula from the time step $k-1$ with respect to X_0

$$\begin{aligned}\tilde{\Gamma}_k^m &= \frac{1}{M_Z}\sum_{i=1}^{M_Z}\frac{\partial}{\partial X_0}\left(\bar{Y}_{k-1}^m(1+rh)\frac{1}{2}(V_{i,k_+}^m - V_{i,k_-}^m)\frac{\mathbb{Z}_i}{\bar{X}_{k-1}^m\sigma\sqrt{h}}\right.\\ &\left. + \bar{Y}_{k-1}^m\sigma\sqrt{h}\frac{1}{2}(V_{i,k_+}^m - 2V_{i,k_o}^m + V_{i,k_-}^m)\frac{\mathbb{Z}_i^2-1}{\bar{X}_{k-1}^m\sigma\sqrt{h}}\right).\end{aligned}$$

with

$$(V_{k_\pm})_i^m = (\bar{X}_{i,k_\pm}^m - K)^+, \quad (V_{k_o})_i^m = (\bar{X}_{i,k_o}^m - K)^+$$

and

$$\bar{X}_{i,k_\pm}^m = \bar{X}_{k-1}^m + rh\bar{X}_{k-1}^m \pm \bar{X}_{k-1}^m\sqrt{\bar{\mathcal{V}}_{k-1}^m}\sqrt{h}\mathbb{Z}_i, \quad \bar{X}_{i,k_o}^m = \bar{X}_{k-1}^m + rh\bar{X}_{k-1}^m.$$

(d) For $i = 1, \ldots, M$

$$\begin{cases} V_k^m = \tilde{V}_k^m, \quad \Gamma_k^m = \tilde{\Gamma}_k^m & \text{if } \tilde{V}_k^m \geq C_{k+1}(\bar{X}_k^{m,i}),\\ V_k^m = e^{-rh}V_{k+1}^m, \quad \Gamma_k^m = e^{-rh}\Gamma_{k+1}^m & \text{otherwise}\end{cases}$$

with $\tilde{V}_k^m = (K - \bar{X}_k^m)^+$.

4. Compute the mean of the Gamma in $t = 0$,

$$\left(\frac{\partial^2 V_0}{\partial X_0^2}\right) = \frac{1}{M}\sum_{m=1}^{M}\Gamma_0^m.$$

1.6.3 Numerical Test

We focus on an American Put (a Bermudan Put with 50 exercise dates) in the unidimensional Black Scholes model. We consider the following parameters : $\sigma = 20\%$ or $\sigma = 40\%$, X_0 varying from 36 to 44, $T = 1$ or $T = 2$, $K = 40$ and $r = 6\%$. The Monte Carlo parameters are: $M = 5\,10^4$ simulation paths and $T/h = 50$ time steps. The basis in the Longstaff Schwartz algorithm is $\psi_i(x) = x^{i-1}, i = 1, 2, 3$, i.e. $I = 3$.

We compare with the solution of the Black Scholes partial differential equation discretized by an implicit Euler scheme in time, finite element in space and semi-smooth Newton for the inequalities, see Achdou and Pironneau [2]. A large number of grid points are used, 10^4 to make it a reference solution.

A second order finite Difference approximation is also used to compute the Gamma for comparison.

Convergence versus M of the result of a Longstaff Schwartz plus Vibrato plus AD is shown on figure 1.11 and compared with FD (1% shift of the initial price) applied to the Monte Carlo algorithm; the perturbation parameter is 1% of the underlying asset price. The reference solution is also shown.

On table 1.10, the results are shown for different sets of parameters taken from Longstaff and Schwartz [124]. The method provides a good precision when variance reduction is used, except when the underlying asset price is small with a small volatility. As for the computation time, the method is faster than FD.

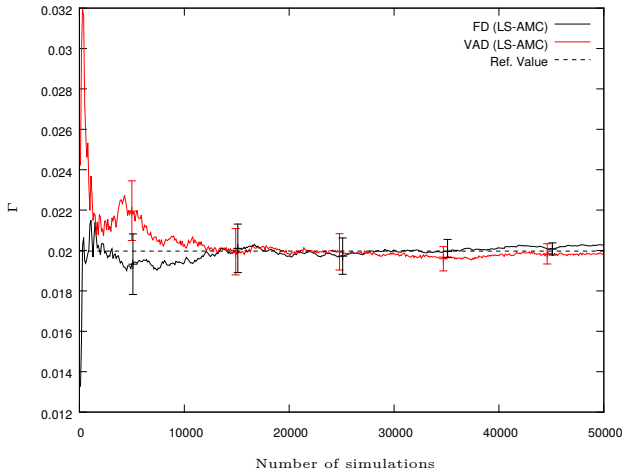

Figure 1.11: Convergence versus M of the Gamma of an American option computed by VAD, or by FD, on the Longstaff Schwartz algorithm. The parameters are $\sigma = 40\%$ and $X_0 = 40$.

1.7 Second Derivatives in a Stochastic Volatility Model

The Heston model [92] describes the evolution of an underlying asset $(X_t)_{t\in[0,T]}$ with a stochastic volatility $(\mathcal{V}_t)_{t\in[0,T]}$:

$$\begin{cases} dX_t = rX_t dt + \sqrt{\mathcal{V}_t} X_t dW_t^1, \\ X_0 = x \in \mathbb{R}, \\ d\mathcal{V}_t = \kappa(\eta - \mathcal{V}_t)dt + \xi\sqrt{\mathcal{V}_t} dW_t^2, \\ \mathcal{V}_0 = v \in \mathbb{R}. \end{cases} \tag{1.73}$$

Here ξ is the volatility of the volatility, η denotes the long-run mean of $\mathcal{V}_t$ and κ the mean reversion velocity. The standard Brownian process $(W_t^1)_{t\in[0,T]}$ and $(W_t^2)_{t\in[0,T]}$ are correlated: $\mathbb{E}[dW_t^1 W_t^2] = \rho dt$, $\rho \in (-1, 1)$. If $2\kappa\eta > \xi^2$, it can be shown that $\mathcal{V}_t > 0$ for every $t \in [0, T]$. We consider the evaluation of a standard European Call with payoff

$$V_T = \mathbb{E}[(X_T - K)^+]. \tag{1.74}$$

1.7.1 Procedure to Compute Second Derivatives in the Heston Model

Here, we describe a simple procedure to compute the Gamma of Call option in the Heston Model.

1. Generate M simulation paths for the underlying asset price $(\bar{X}, \bar{\mathcal{V}})$ and its tangent process $(\bar{Y}, \bar{\mathcal{U}}) = \frac{\partial(\bar{X},\bar{\mathcal{V}})}{\partial X_0}$ using an Euler scheme with n time steps of size $h = \frac{T}{n}$, but as the process $\mathcal{V}$ is not a function of X_0 the computation of its tangent process is not required. For $1 < m < M$ and $0 < k < n - 1$

$$\begin{cases} \bar{X}_{k+1}^m = \bar{X}_k^m + rh\bar{X}_k^m + \bar{X}_k^m\sqrt{\bar{\mathcal{V}}_k^m}\sqrt{h}\tilde{Z}_{k+1}^{1,m}, & \bar{X}_0^m = X_0, \\ \bar{Y}_{k+1}^m = \bar{Y}_k^m + rh\bar{Y}_k^m + \bar{Y}_k^m\sqrt{\bar{\mathcal{V}}_k^m}\sqrt{h}\tilde{Z}_{k+1}^{1,m}, & \bar{Y}_0^m = 1, \\ \bar{\mathcal{V}}_{k+1}^m = \bar{\mathcal{V}}_k^m + \kappa(\eta - \bar{\mathcal{V}}_k^m)h + \xi\sqrt{\bar{\mathcal{V}}_k^m}\sqrt{h}\tilde{Z}_{k+1}^{2,m}, & \bar{\mathcal{V}}_0^m = \mathcal{V}_0 \end{cases}$$

with

$$\begin{pmatrix} \tilde{Z}^1 \\ \tilde{Z}^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{pmatrix} \begin{pmatrix} Z^1 \\ Z^2 \end{pmatrix}$$

where $(Z_k^1, Z_k^2)_{1 \leq k \leq n}$ denotes a sequence of $\mathcal{N}(0; I_2)$-distributed random variables.

2. For each simulation path

   (a) Simulate M_Z replications of the payoff, for $1 < i < M_Z$

   $$(V_{T_\pm})_i^m = (\bar{X}_{i,T_\pm}^m - K)^+, \quad (V_{T_o})_i^m = (\bar{X}_{i,T_o}^m - K)^+$$

   with

   $$\bar{X}_{i,T_\pm}^m = \bar{X}_{n-1}^m + rh\bar{X}_{n-1}^m \pm \bar{X}_{n-1}^m \sqrt{\bar{\mathcal{V}}_{n-1}^m} \sqrt{h} \mathbb{Z}_i^1, \quad \bar{X}_{i,T_o}^m = \bar{X}_{n-1}^m + rh\bar{X}_{n-1}^m.$$

   (b) Compute the Delta using Vibrato at maturity with the

   $$\begin{aligned} \left(\frac{\partial V_T}{\partial X_0}\right)_i^m &= \bar{Y}_{n-1}^m (1+rh) \frac{1}{2} (V_{i,T_+}^m - V_{i,T_-}^m) \frac{\mathbb{Z}_i^1}{\bar{X}_{n-1}^n \sqrt{\bar{\mathcal{V}}_{n-1}^m} \sqrt{h}} \\ &\quad + \bar{Y}_{n-1}^m \sqrt{\bar{\mathcal{V}}_{n-1}^m} \sqrt{h} \frac{1}{2} (V_{i,T_+}^m - 2V_{i,T_o}^m + V_{i,T_-}^m) \frac{\mathbb{Z}_i^1 - 1}{\bar{X}_{n-1}^m \sqrt{\bar{\mathcal{V}}_{n-1}^m} \sqrt{h}} \end{aligned}$$

   (c) Apply an Automatic Differentiation method on step (2b) to compute the Gamma.

   (d) Compute the mean over M_Z replications

   $$\left(\frac{\partial^2 V_T}{\partial X_0^2}\right)^m = \frac{1}{M_Z} \sum_{i=1}^{M_Z} \left(\frac{\partial^2 V_T}{\partial X_0^2}\right)_i^m$$

3. Compute the mean over the M simulation paths of the result and discount it.

$$\frac{\partial^2 V_T}{\partial X_0^2} = e^{-rT} \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\partial^2 V_T}{\partial X_0^2}\right)^m.$$

1.7.1.1 Numerical Test

We have taken the following values: the underlying asset price $X_0 \in [60, 130]$, the strike is $K = 90$, the risk-free rate $r = 0.135\%$ and the maturity is $T = 1$.

The initial volatility is $\mathcal{V}_0 = 2.8087\%$, the volatility of volatility is $\xi = 10\%$, the mean reversion is $\kappa = 2.931465$ and the long-run mean is $\nu = 10.1\%$. The correlation between the two standard Brownian motions is $\rho = 50\%$. The number of Monte Carlo path is $500,000$ with 100 time steps each. The results are displayed on figures 1.12, 1.13 and 1.14. On figure 1.12 we compare the results obtained by Vibrato plus Automatic Differentiation (direct mode), with second order Finite Difference method (1% shift of the initial price) applied to a standard Monte Carlo simulation. On figures 1.13 and 1.14, we display respectively the Vomma and the Vanna of an European Call option in the Heston model. As for the Gamma, the method provides a good precision for the approximation of the Vomma and the Vanna. Both are computed at one point $(X_0, \mathcal{V}_0) = (85, 2.8087)$ with the same set of parameters as given above. In the case of the Vomma and the Gamma, VAD is 30% faster. For the Vanna Finite difference requires four times the evaluation of the pricing function so VAD is twice times faster.

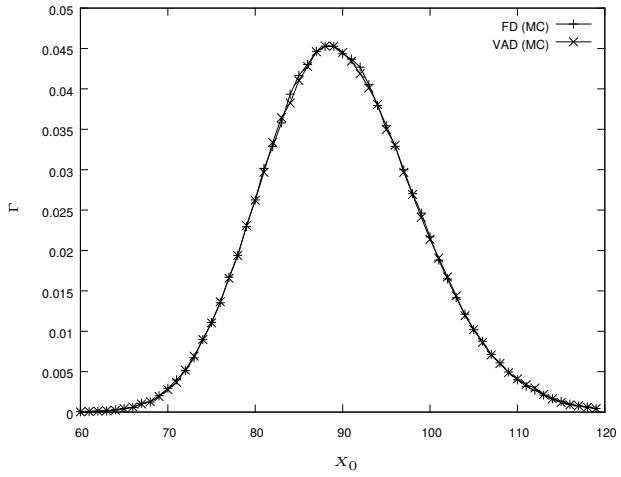

Figure 1.12: The Gamma versus Price is displayed when computed by VAD; the approximated Gamma via Finite Difference is also displayed; both curves overlap.

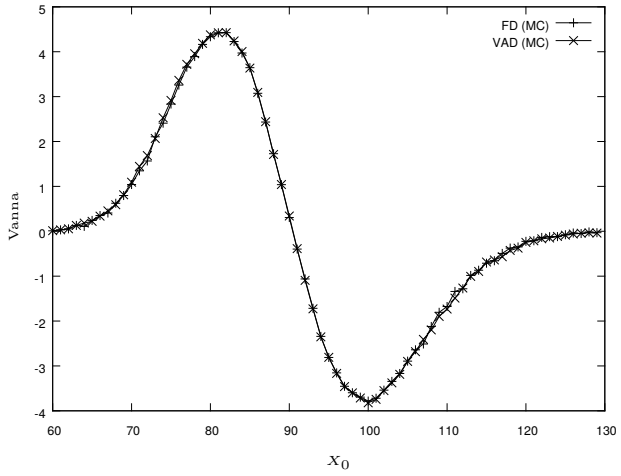

Figure 1.13: The Vanna versus Price is displayed when computed by VAD; the approximated Vanna via Finite Difference is also displayed; both curves overlap.

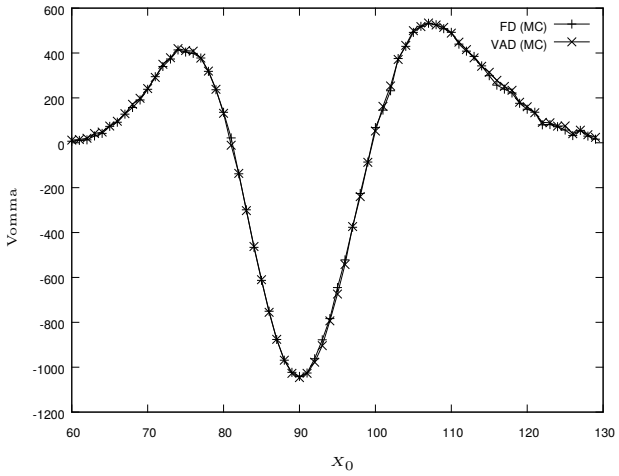

Figure 1.14: The Vomma versus Price is displayed when computed by VAD; the approximated Vomma via Finite Difference is also displayed; both curves overlap.

1.8 Malliavin Calculus and Likelihood Ratio Method

Here, we want to point out that Malliavin calculus and LRM are excellent methods but they have their own numerical issues especially with short maturities which may make VAD more attractive for a general purpose software.

Let us start by recalling briefly the foundations of Malliavin calculus (further details are available in Nualart [138], Fournié et al. [55] and in Gobet and Munos [77], for instance). We recall the Bismut-Elworthy-Li formula (see Bismut et al. [23], for example):

Proposition 4. (Bismut-Elworthy-Li formula) *Let X be a diffusion process given by (1.3) with $d = 1$, b and σ in $\mathcal{C}^1$. Let $f : \mathbb{R} \to \mathbb{R}$ be $\mathcal{C}^1$ with $\mathbb{E}[f(X_T)^2]$ and $\mathbb{E}[f'(X_T)^2]$ bounded. Let $(H_t)_{t\in[0,T]}$ an $\mathscr{F}$-progressively measurable process in $L^2([0,T]\times\Omega, dt\otimes d\mathbb{P})$ such that $\mathbb{E}\left[\int_0^T H_s^2 ds\right]$ is finite. Then*

$$\mathbb{E}\left[f(X_T)\int_0^T H_s dW_s\right] = \mathbb{E}\left[f'(X_T)Y_T\int_0^T \frac{\sigma(X_s)H_s}{Y_s}ds\right] \tag{1.75}$$

where $Y_t = \dfrac{\partial X_t}{\partial X_0}$ is the tangent process defined in (1.10).

By choosing $H_t = Y_t/\sigma(X_t)$ the above yields

$$\frac{\partial}{\partial X_0}\mathbb{E}\left[f(X_T)\right] = \mathbb{E}\left[f(X_T)\underbrace{\frac{1}{T}\int_0^T \frac{Y_s}{\sigma(X_s)}dW_s}_{\text{Malliavin weight}}\right] \tag{1.76}$$

provided f has polynomial growth and $\mathbb{E}\left[\int_0^T \left(\dfrac{Y_t}{\sigma(X_t)}\right)^2\right]$ is finite.

1.8.0.2 Second Derivative

In the context of the Black Scholes model, the Malliavin weights, π_Γ, for the Gamma is (see Benhamou [19]):

$$\pi_\Gamma = \frac{1}{X_0^2 \sigma T}\left(\frac{W_T^2}{\sigma T} - \frac{1}{\sigma} - W_T\right). \tag{1.77}$$

Hence

$$\Gamma_{\text{Mal}} = e^{-rT}\mathbb{E}\left[(X_T - K)^+ \frac{1}{X_0^2 \sigma T}\left(\frac{W_T^2}{\sigma T} - \frac{1}{\sigma} - W_T\right)\right]. \tag{1.78}$$

The pure likelihood ratio method gives a similar formula (see Lemma 1)

$$\Gamma_{\text{LR}} = e^{-rT}\mathbb{E}\left[(X_T - K)^+ \left(\frac{Z^2 - 1}{X_0^2 \sigma^2 T} - \frac{Z}{X_0^2 \sigma \sqrt{T}}\right)\right]. \tag{1.79}$$

LRPW is an improvement of LRM obtained by combining it with a pathwise method (see Glasserman [71]).

$$\Gamma_{\text{LRPW}} = \frac{\partial}{\partial X_0}\left(e^{-rT}\mathbb{E}\left[(X_T - K)^+ \frac{Z}{X_0 \sigma \sqrt{T}}\right]\right) = e^{-rT}\frac{K}{X_0^2 \sigma \sqrt{T}}\mathbb{E}[Z \mathbb{1}_{X_T > K}]. \tag{1.80}$$

LRPW is much cheaper than VAD, Malliavin or LRM and it is also less singular at $T = 0$. However all these methods require new analytically derivations for each new problem.

1.8.1 Numerical Tests

We compared VAD with LRPW, Malliavin calculus and FD (1% shift of the initial price). The results are shown on Table 1.2

T	VAD (MC)	FD (MC)	LRPW (MC)	Malliavin (MC)
1.000e+00	3.635e−05	1.768e−04	3.402e−04	9.197e−03
5.000e−01	8.556e−05	3.117e−04	7.791e−04	1.625e−02
1.000e−01	6.642e−04	1.504e−03	4.002e−03	6.543e−02
5.000e−02	1.494e−03	2.802e−03	7.571e−03	1.212e−01
1.000e−02	8.780e−03	1.843e−02	3.765e−02	5.443e−01
5.000e−03	1.862e−02	3.959e−02	7.553e−02	1.101e+00
1.000e−03	9.623e−02	1.776e−01	3.761e−01	5.743e+00
5.000e−04	1.851e−01	3.345e−01	7.562e−01	1.074e+01
1.000e−04	1.013e+00	1.632e+00	3.777e+00	5.265e+01
5.000e−05	1.988e+00	3.464e+00	7.543e+00	1.091e+02
1.000e−05	1.033e+01	1.781e+01	3.791e+01	5.404e+02

Table 1.2: Variance of the Gamma of a standard European Call with short maturities in the Black Scholes model. Gamma is computed with VAD, FD, LRPW and Malliavin. The computation are done on the same samples.

The Gamma is computed with the same parameters as in the section 1.4.3. The maturity is varying from $T = 1$ to 10^{-5} year. The Monte Carlo parameters are also set to 100 000 simulation paths and 25 time steps. Notice the inefficiency of LRPW, Malliavin Calculus and to a lesser degree of VAD and Finite Difference when T is small.

Note on CPU

Tests have been done on an Intel Core i5-3210M Processor @ 2,50 GHz. The processor has turbo speed of 3.1 GHz and two cores. We did not use parallel processing.

1.9 Conclusion

This article chapter the work of Mike Giles and investigates the Vibrato method for higher order derivatives in quantitative finance.

For a general purpose software for second derivatives, Vibrato of Vibrato is too complex, but we showed that it is essentially similar to the analytical differentiation of Vibrato (see 1.19 and 1.21) and hence also to Automatic Differentiation applied on Vibrato (VAD). The method applies also to baskets, stochastic volatilities and American contracts.

We have shown too that Automatic differentiation for higher derivatives without Vibrato can be enhanced to handle the singularities of the payoff functions used in finance. Yet, while AD for second derivatives is certainly the easiest solution, it is not the safest and it requires an appropriate choice of the parameter which is used to approximate Dirac functions by exponentials. Third derivatives can be computed as second derivative by AD of VAD. Here too a good choice of the parameter used to approximate the Dirac functions is needed.

The framework proposed, Vibrato combined with AD, is easy to implement, efficient, fast and stable and does not require an analytical derivation when the volatility, the payoff, etc, are changed. Furthermore AD in reverse mode allows for fast calculation of many second derivatives at once.

Developments are in progress to test the method with nested Monte Carlo and Multilevel-Multistep Richardson-Romberg extrapolation like in Burgos and Giles [37] and Lemaire and Pagès [119].

Acknowledgment

The author has received the support of ANRT and Global Market Solution inc. with special encouragements from Youssef Allaoui and Laurent Marcoux.

Table 1.3: Mean values of all second derivatives for an European Call option computed by VRAD or FD (with $\varepsilon = 0.01$), as a function of the number of MC samples M and of the number of time steps for the SDEs. The reference values are analytical solutions. Prices of each second derivatives and the computing time are displayed. The settings are $\sigma = 0.2$, $r = 0.05$, $K = 100$, $X_0 = 90$, $T = 1$.

M	T/h	Method	$\partial^2 V/\partial X_0^2$	$\partial^2 V/\partial X_0 \partial \sigma$	$\partial^2 V/\partial X_0 \partial r$	$\partial^2 V/\partial X_0 \partial T$	$\partial^2 V/\partial \sigma^2$	$\partial^2 V/\partial \sigma \partial rs$	$\partial^2 V/\partial \sigma \partial T$	$\partial^2 V/\partial r^2$	$\partial^2 V/\partial r \partial T$	$\partial^2 V/\partial T^2$	CPU time
–	–	Ref.	2.182e−02	7.400e−01	1.964e+00	1.722e−01	1.177e+01	3.125e+01	2.041e+01	1.431e+02	4.388e+01	9.793e−01	–
1000	25	VRAD	2.509e−02	8.605e−01	2.278e+00	2.013e−01	1.511e+01	3.831e+01	2.376e+01	1.713e+02	5.312e+01	1.054e+00	3.080e−03
1000	25	FD	2.244e−02	7.655e−01	2.005e+00	1.764e−01	1.027e+01	1.708e+01	2.164e+01	3.520e+01	5.197e+01	1.670e+00	3.769e−03
1000	50	VRAD	1.969e−02	7.642e−01	1.781e+00	1.635e−01	1.500e+01	3.163e+01	2.202e+01	1.268e+02	4.528e+01	1.107e+00	1.678e−03
1000	50	FD	1.832e−02	7.035e−01	1.781e+00	1.594e−01	3.239e+00	9.522e+00	1.811e+01	1.841e+01	4.417e+01	1.552e+00	7.007e−03
10000	25	VRAD	2.262e−02	7.599e−01	2.054e+00	1.811e−01	1.376e+01	3.334e+01	2.126e+01	1.513e+02	4.573e+01	9.657e−01	1.612e−02
10000	25	FD	2.110e−02	7.388e−01	1.904e+00	1.690e−01	1.229e+01	2.942e+01	2.066e+01	1.254e+02	5.176e+01	1.044e+00	3.496e−02
10000	50	VRAD	2.166e−02	7.388e−01	1.957e+00	1.701e−01	1.074e+01	3.392e+01	1.938e+01	1.425e+02	4.244e+01	8.775e−01	1.654e−02
10000	50	FD	1.996e−02	7.192e−01	1.868e+00	1.653e−01	1.207e+01	2.978e+01	1.984e+01	1.628e+02	5.095e+01	9.545e−01	6.509e−02
100000	25	VRAD	2.189e−02	7.431e−01	1.987e+00	1.762e−01	1.467e+01	3.219e+01	2.110e+01	1.451e+02	4.458e+01	9.640e−01	1.817e−01
100000	25	FD	2.166e−02	7.327e−01	1.909e+00	1.687e−01	1.197e+01	3.083e+01	2.049e+01	1.451e+02	5.201e+01	9.893e−01	3.530e−01
100000	50	VRAD	2.163e−02	7.484e−01	1.956e+00	1.729e−01	1.349e+01	3.227e+01	2.085e+01	1.424e+02	4.425e+01	9.758e−01	1.684e−01
100000	50	FD	2.212e−02	7.447e−01	1.978e+00	1.733e−01	1.229e+01	3.498e+01	2.070e+01	1.582e+02	5.354e+01	8.883e−01	6.581e−01

Table 1.4: RMSE of values in Table 1.3: i.e. mean value error of all second derivatives for a Call option the value of which are shown in Table 1.3. We compare VRAD with FD methods ($\varepsilon = 0.01$). The settings are $\sigma = 0.2$, $r = 0.05$, $K = 100$, $X_0 = 90$, $T = 1$.

M	T/h	Method	$\partial^2 V/\partial X_0^2$	$\partial^2 V/\partial X_0 \partial\sigma$	$\partial^2 V/\partial X_0 \partial r$	$\partial^2 V/\partial X_0 \partial T$	$\partial^2 V/\partial\sigma^2$	$\partial^2 V/\partial\sigma\partial r$	$\partial^2 V/\partial\sigma\partial T$	$\partial^2 V/\partial r^2$	$\partial^2 V/\partial r\partial T$	$\partial^2 V/\partial T^2$
1000	25	VRAD	2.289e−03	7.100e−02	2.066e−01	1.541e−02	6.247e+00	4.900e+00	2.441e+00	1.856e+01	4.313e+00	1.587e−01
1000	25	FD	4.130e−03	5.904e−02	3.162e−01	2.114e−02	4.326e+00	9.965e+00	1.727e+00	1.453e+00	5.211e+00	1.789e−01
1000	50	VRAD	2.613e−03	7.934e−02	2.354e−01	1.704e−02	7.486e+00	5.668e+00	2.646e+00	2.111e+01	4.075e+00	1.689e−01
1000	50	FD	3.604e−03	5.819e−02	2.989e−01	2.032e−02	3.337e+00	9.522e+00	1.433e+00	1.527e+01	3.926e+00	2.261e−01
10000	25	VRAD	6.922e−04	2.100e−02	6.244e−02	4.631e−03	1.655e+00	1.404e+00	7.627e−01	5.590e+00	1.255e+00	5.108e−02
10000	25	FD	1.270e−03	1.877e−02	9.770e−02	6.604e−03	2.026e+00	6.067e+00	5.619e−01	4.526e+01	2.080e+00	1.807e−01
10000	50	VRAD	8.258e−04	2.215e−02	7.438e−02	5.275e−03	1.508e+00	1.601e+00	6.822e−01	6.672e+00	1.205e+00	5.174e−02
10000	50	FD	1.222e−03	1.851e−02	9.667e−02	6.524e−03	2.205e+00	6.430e+00	5.462e−01	5.146e+01	2.103e+00	1.980e−01
100000	25	VRAD	2.168e−04	6.865e−03	1.955e−02	1.456e−03	5.745e−01	4.655e−01	2.475e−01	1.751e+00	3.937e−01	1.597e−02
100000	25	FD	4.076e−04	5.867e−03	3.092e−02	2.084e−03	6.595e−01	1.999e+00	1.770e−01	1.513e+01	6.722e−01	6.001e−02
100000	50	VRAD	2.578e−04	7.025e−03	2.322e−02	1.668e−03	6.747e−01	4.710e−01	2.618e−01	2.082e+00	4.040e−01	1.718e−02
100000	50	FD	4.108e−04	5.985e−03	3.146e−02	2.122e−03	6.727e−01	2.105e+00	1.830e−01	1.552e+01	7.112e−01	6.231e−02

Table 1.5: Price of second derivatives of a Call option computed by VAD or FD. Same as Table 1.3 but when X_0 varies and M is fixed at $M = 100\ 000$ and the time step is $T/h = 25$. The reference values are analytical solutions.

X_0	Method	$\partial^2 V/\partial X_0^2$	$\partial^2 V/\partial X_0 \partial \sigma$	$\partial^2 V/\partial X_0 \partial r$	$\partial^2 V/\partial X_0 \partial T$	$\partial^2 V/\partial \sigma^2$	$\partial^2 V/\partial \sigma \partial r$	$\partial^2 V/\partial \sigma \partial T$	$\partial^2 V/\partial r^2$	$\partial^2 V/\partial r \partial T$	$\partial^2 V/\partial T^2$	CPU time
80	Ref.	1.860e−02	1.437e+00	1.488e+00	2.181e−01	8.802e+01	9.114e+01	2.526e+01	1.031e+02	3.017e+01	−8.591e−01	–
80	VRAD	1.892e−02	1.462e+00	1.523e+00	2.237e−01	9.119e+01	9.368e+01	2.591e+01	1.059e+02	3.092e+01	−9.054e−01	1.537e−01
80	FD	1.835e−02	1.415e+00	1.453e+00	2.142e−01	8.478e+01	9.215e+01	2.534e+01	8.350e+01	3.420e+01	−7.235e−01	3.453e−01
90	Ref.	2.182e−02	7.400e−01	1.964e+00	1.722e−01	1.177e+01	3.125e+01	2.041e+01	1.431e+02	4.388e+01	9.793e−01	–
90	VRAD	2.200e−02	7.449e−01	1.997e+00	1.766e−01	1.360e+01	3.259e+01	2.089e+01	1.460e+02	4.470e+01	9.592e−01	1.633e−01
90	FD	2.137e−02	7.351e−01	1.921e+00	1.695e−01	1.213e+01	3.033e+01	2.055e+01	1.418e+02	5.202e+01	1.008e+00	3.479e−01
100	Ref.	1.876e−02	−2.814e−01	1.876e+00	6.567e−02	9.850e+00	−6.567e+01	1.646e+01	1.344e+02	5.339e+01	2.098e+00	–
100	VRAD	1.863e−02	−2.732e−01	1.889e+00	6.941e−02	1.240e+01	−6.404e+01	1.712e+01	1.354e+02	5.438e+01	2.079e+00	1.595e−01
100	FD	1.868e−02	−2.745e−01	1.854e+00	6.528e−02	1.112e+01	−6.775e+01	1.638e+01	1.520e+02	6.729e+01	2.082e+00	3.503e−01
110	Ref.	1.289e−02	−8.881e−01	1.418e+00	−1.794e−02	8.075e+01	−1.289e+02	1.722e+01	8.606e+01	6.128e+01	1.825e+00	–
110	VRAD	1.293e−02	−8.797e−01	1.455e+00	−1.377e−02	8.422e+01	−1.278e+02	1.816e+01	8.996e+01	6.280e+01	1.818e+00	1.626e−01
110	FD	1.268e−02	−8.804e−01	1.413e+00	−1.766e−02	7.191e+01	−1.185e+02	1.691e+01	8.368e+01	8.168e+01	1.819e+00	3.506e−01
120	Ref.	7.500e−03	−9.555e−01	9.000e−01	−5.055e−02	1.447e+02	−1.363e+02	1.845e+01	2.660e+01	6.911e+01	9.296e−01	–
120	VRAD	7.462e−03	−9.511e−01	9.312e−01	−4.719e−02	1.470e+02	−1.347e+02	1.865e+01	3.016e+01	6.962e+01	8.660e−01	1.629e−01
120	FD	7.903e−03	−9.936e−01	9.268e−01	−5.290e−02	1.642e+02	−1.534e+02	1.953e+01	2.916e+01	9.543e+01	8.894e−01	3.527e−01

Table 1.6: RMSE for each second derivatives for several values of X_0. The settings are $\sigma = 0.2$, $r = 0.05$, $K = 100$, $T = 1$ and the Monte Carlo settings are $M = 100\ 000$ and the number of time steps is $T/h = 25$. The price of these items are shown in Table 1.5.

X_0	Method	$\partial^2 V/\partial X_0^2$	$\partial^2 V/\partial X_0 \partial \sigma$	$\partial^2 V/\partial X_0 \partial r$	$\partial^2 V/\partial X_0 \partial T$	$\partial^2 V/\partial \sigma^2$	$\partial^2 V/\partial \sigma \partial r$	$\partial^2 V/\partial \sigma \partial T$	$\partial^2 V/\partial r^2$	$\partial^2 V/\partial r \partial T$	$\partial^2 V/\partial T^2$
80	VRAD	2.312e−04	1.633e−02	1.854e−02	2.516e−03	1.029e+00	1.183e+00	2.825e−01	1.460e+00	3.554e−01	2.295e−02
80	FD	4.234e−04	2.191e−02	2.878e−02	3.617e−03	3.923e+00	4.394e+00	5.106e−01	1.072e+01	7.951e−01	1.162e−01
90	VRAD	2.174e−04	6.822e−03	1.961e−02	1.456e−03	5.326e−01	4.662e−01	2.395e−01	1.756e+00	3.932e−01	1.592e−02
90	FD	4.037e−04	5.872e−03	3.101e−02	2.087e−03	6.474e−01	1.981e+00	1.768e−01	1.490e+01	6.693e−01	5.934e−02
100	VRAD	1.779e−04	8.760e−03	1.782e−02	5.271e−04	7.203e−01	7.154e−01	2.858e−01	1.794e+00	4.357e−01	1.668e−02
100	FD	3.399e−04	1.070e−02	2.889e−02	4.732e−04	1.718e+00	3.511e+00	1.650e−01	1.652e+01	2.688e−01	1.242e−02
110	VRAD	1.359e−04	1.441e−02	1.498e−02	7.333e−04	1.310e+00	1.402e+00	3.101e−01	1.673e+00	4.910e−01	1.921e−02
110	FD	2.535e−04	2.002e−02	2.409e−02	8.666e−04	5.093e+00	5.002e+00	2.418e−01	1.417e+01	3.930e−01	2.236e−02
120	VRAD	9.581e−05	1.655e−02	1.151e−02	1.087e−03	2.357e+00	1.770e+00	3.438e−01	1.412e+00	5.468e−01	2.569e−02
120	FD	1.867e−04	2.337e−02	1.878e−02	1.484e−03	8.710e+00	5.818e+00	4.735e−01	1.201e+01	6.003e−01	5.843e−02

Table 1.7: Results for the price, the Delta and the Gamma of a Basket Option priced with the moment-matching approximation (reference values), FD on Monte Carlo and Vibrato plus Automatic Differentiation on Monte Carlo. The settings of Monte Carlo simulation are 1 time step and 1 000 000 simulation paths.

d	T	Price MMA	Price (MC)	Delta MMA	Delta Vibrato (MC)	Delta FD (MC)	Gamma MMA	Gamma VAD (MC)	Gamma FD (MC)
1	0.1	3.842e+01	3.738e+01	5.527e−01	5.515e−01	5.540e−01	4.655e−3	4.661e−3	4.649e−3
2	0.1	3.444e+01	3.412e+01	2.746e−01	2.721e−01	2.848e−01	1.289e−3	1.349e−3	1.281e−3
3	0.1	4.607e+01	4.598e+01	1.830e−01	1.828e−01	1.862e−01	4.291e−4	4.285e−4	4.210e−4
4	0.1	5.967e+01	5.878e+01	1.377e−01	1.363e−01	1.411e−01	1.861e−4	1.932e−4	1.790e−4
5	0.1	9.284e+01	9.090e+01	1.095e−01	1.084e−01	1.096e−01	7.645e−5	7.796e−5	7.599e−5
6	0.1	1.392e+02	1.417e+02	9.192e−01	9.021e−02	9.050e−02	3.542e−5	3.718e−5	3.411e−5
7	0.1	1.554e+02	1.533e+02	7.816e−01	7.752e−02	7.731e−02	2.316e−5	2.090e−5	2.183e−5
1	1	1.553e+02	1.547e+02	6.618e−01	6.600e−01	6.721e−01	1.308e−3	1.300e−3	1.328e−3
2	1	1.354e+02	1.331e+02	3.258e−01	3.219e−01	3.273e−01	3.806e−4	3.869e−4	3.838e−4
3	1	1.819e+02	1.826e+02	2.170e−01	2.144e−01	2.169e−01	1.265e−4	1.344e−4	1.249e−4
4	1	2.349e+02	2.320e+02	1.633e−01	1.607e−01	1.653e−01	5.491e−5	5.629e−5	5.509e−5
5	1	3.646e+02	3.633e+02	1.306e−01	1.271e−01	1.281e−01	2.258e−5	2.382e−5	2.192e−5
6	1	5.436e+02	5.408e+02	1.074e−01	1.043e−01	1.044e−01	1.041e−5	8.998e−6	1.138e−5
7	1	6.038e+02	6.072e+02	9.241e−02	8.924e−02	8.990e−02	6.870e−6	7.703e−6	7.228e−6

Table 1.8: Time computing (in seconds) for the Gamma with FD on Monte Carlo and with Vibrato plus Automatic Differentiation on Monte Carlo simulation, dimension of the problem are varying. The settings of Monte Carlo algorithm are the same as above.

Method (Computing Gamma)	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$
FD (MC)	0.49	0.95	1.33	1.82	2.26	2.91	3.36
VAD (MC)	0.54	0.77	0.92	1.21	1.50	1.86	2.31

Table 1.9: Results for the price, the Delta and the Gamma of an Autocallable. The reference values are the results obtained with the One-step survival Monte Carlo techniques plus Finite differences for the Greeks, they are compared to Vibrato plus Automatic Differentiation on Monte Carlo.

S	T	Observations Dates	Price (MC)	Price (OSS MC)	Delta FD (OSS MC)	Delta Vibrato (MC)	Gamma FD (OSS MC)	Gamma Vibrato + AD (MC)
3000	1	1	7.105e+02	7.104e+02	1.753e−02	1.811e−02	−9.435e−06	−1.012e−05
3000	2	2	7.455e+02	7.456e+02	2.192e−02	2.140e−02	−8.104e−06	−1.056e−05
3000	3	3	7.724e+02	7.719e+02	2.433e−02	2.396e−02	−8.704e−06	−1.105e−05
3250	1	1	7.415e+02	7.415e+02	1.512e−02	1.570e−02	−8.973e−06	−9.713e−06
3250	2	2	7.981e+02	7.977e+02	1.894e−02	1.894e−02	−9.892e−06	−1.046e−05
3250	3	3	8.309e+02	8.310e+02	2.187e−02	2.187e−02	−1.133e−05	−1.122e−05
3500	1	1	7.870e+02	7.872e+02	1.313e−02	1.432e−02	−8.416e−06	−8.836e−06
3500	2	2	8.450e+02	8.446e+02	1.694e−02	1.743e−02	−1.054e−05	−9.998e−06
3500	3	3	8.833e+02	8.832e+02	1.782e−02	1.862e−02	−1.359e−05	−1.433e−05
3750	1	1	8.172e+02	8.173e+02	1.102e−02	1.185e−02	−7.828e−06	−7.949e−06
3750	2	2	8.848e+02	8.851e+02	1.367e−02	1.398e−02	−1.133e−05	−1.109e−05
3750	3	3	9.271e+02	9.272e+02	1.523e−02	1.446e−02	−1.418e−05	−1.430e−05
4000	1	1	9.624e+02	9.612e+02	9.020e−02	9.080e−3	−6.947e−06	−7.142e−06
4000	2	2	8.450e+02	8.446e+02	1.083e−02	1.076e−02	−1.115e−05	−1.082e−05
4000	3	3	8.833e+02	8.832e+02	1.237e−02	1.215e−02	−1.361e−05	−1.320e−05

Table 1.10: Results of the price, the Delta and the Gamma of an American option. The reference values are obtained via the Semi-Newton method plus FD, they are compared to Vibrato plus Automatic Differentiation on the Longstaff Schwartz algorithm. We compute the standard error for each American Monte Carlo results. The settings of the American Monte Carlo are 50 time steps and $50,000$ simulation paths.

S	σ	T	Price Ref. Value	Price (AMC)	Standard Error	Delta Ref. Value	Delta Vibrato (AMC)	Standard Error	Gamma Ref. Value	Gamma VAD (AMC)	Standard Error
36	0.2	1	4.479e+00	4.462e+00	1.303e−02	6.850e−01	6.812e−01	1.820e−03	8.787e−02	6.795e−02	6.947e−05
36	0.2	2	4.838e+00	4.815e+00	1.699e−02	6.184e−01	5.996e−01	1.813e−03	7.311e−02	6.489e−02	6.846e−05
36	0.4	1	7.071e+00	7.079e+00	1.635e−02	5.103e−01	5.113e−01	1.674e−03	3.320e−02	3.534e−02	4.852e−05
36	0.4	2	8.441e+00	8.456e+00	2.314e−02	4.459e−01	4.418e−01	1.488e−03	2.594e−02	2.660e−02	5.023e−05
38	0.2	1	3.241e+00	3.233e+00	1.379e−02	5.370e−01	5.302e−01	1.821e−03	7.318e−02	7.261e−02	1.198e−04
38	0.2	2	3.740e+00	3.727e+00	1.508e−02	4.869e−01	4.673e−01	1.669e−03	5.943e−02	5.860e−02	1.111e−04
38	0.4	1	6.115e+00	6.112e+00	1.655e−02	4.477e−01	4.507e−01	1.453e−03	3.099e−02	3.189e−02	5.465e−05
38	0.4	2	7.599e+00	7.610e+00	2.560e−02	3.972e−01	3.957e−01	1.922e−03	2.210e−02	2.313e−02	4.827e−05
40	0.2	1	2.310e+00	2.305e+00	1.247e−02	4.110e−01	4.076e−01	1.880e−03	6.001e−02	5.901e−02	1.213e−04
40	0.2	2	2.878e+00	2.860e+00	1.496e−02	3.808e−01	3.924e−01	1.747e−03	4.765e−02	4.697e−02	5.175e−04
40	0.4	1	5.279e+00	5.287e+00	1.945e−02	3.903e−01	3.942e−01	1.629e−03	2.759e−02	2.746e−02	1.249e−05
40	0.4	2	6.847e+00	6.858e+00	2.612e−02	3.556e−01	3.541e−01	1.416e−03	2.021e−02	2.153e−02	3.989e−05
42	0.2	1	1.613e+00	1.607e+00	1.107e−02	3.063e−01	2.970e−01	1.734e−03	4.747e−02	4.532e−02	4.797e−05
42	0.2	2	2.206e+00	2.190e+00	1.417e−02	2.950e−01	2.814e−01	1.601e−03	3.790e−02	3.619e−02	5.560e−05
42	0.4	1	4.550e+00	4.571e+00	1.587e−02	3.394e−01	3.437e−01	1.517e−03	2.444e−02	2.418e−02	3.194e−05
42	0.4	2	6.174e+00	6.184e+00	2.353e−02	3.186e−01	2.999e−01	1.347e−03	1.771e−02	1.776e−02	2.961e−05
44	0.2	1	1.108e+00	1.096e+00	9.009e−03	2.133e−01	2.052e−01	1.503e−03	3.679e−02	3.557e−02	1.486e−04
44	0.2	2	1.685e+00	1.669e+00	1.213e−02	2.287e−01	2.192e−01	1.487e−03	2.984e−02	2.891e−02	2.363e−04
44	0.4	1	3.917e+00	3.908e+00	1.573e−02	2.942e−01	2.973e−01	1.403e−03	2.129e−02	2.168e−02	1.274e−04
44	0.4	2	5.572e+00	5.582e+00	2.834e−02	2.841e−01	2.841e−01	1.325e−03	1.548e−02	1.510e−02	2.162e−04

Chapitre 2

Computing Sensitivities of Financial Derivatives with Weighted Multilevel Monte Carlo

---

"La probabilité est relative, en partie, à cette ignorance et en partie, à nos connaissances."
extract from Théorie Analytique des Probabilités, 1812.

– P. S. Laplace, Mathematician, (1749, 1827).

This chapter deals with the approximation of the sensitivities of financial derivatives with a weighted multilevel Monte Carlo. The computation of sensitivities are made with an extended version of automatic differentiation for non-twice differentiable payoff using a regularization of the Heaviside function. We show how to calibrate the method for sensitivities using the optimal framework designed by Lemaire and Pagès in [119] for standard and antithetic discretization scheme. Also, we give a description of a framework which compute the value, the Delta and the Gamma of a security using the optimal parameters for each function on the same grid. We give some numerical results on vanilla and exotic derivatives.

This chapter is a part of an article in preparation.

2.1 Introduction

In order to avoid new financial crisis, the global regulations are constantly refined and adding new ones. These regulations concern the entire business of a financial institution as trading, accounting, capital requirements and reporting. It is changing their infrastructures and they struggle to keep up with challenge.

The recent capital regulations as the Fundamental Review of the Trading Book (FTRB)[1] from the Basel agreement and the Standardized Approach for measuring exposure at default

[1] See `www.bis.org/publ/bcbs265.pdf`

for Counterparty Credit Risk (SA-CCR)[2] have the highest impact on computational complexity. These regulations imply to have a market risk approach based on sensitivities to several risk factors and requires calculating sensitivities of credit value adjustment.

In September 2016, the International SWAPs and Derivatives Associations (ISDA) delivered a new regulation entitled Standard Initial Margin Model (SIMM)[3] for non-cleared derivatives which is a mandatory methodology to calculate initial margin. It requires computing sensitivities of the trades for each risk factors for many Monte Carlo scenarios at multiple dates, which causes a computational challenge to banks.

These new challenges require changes in the pricing infrastructure and in a response to these issues, new efficient Monte Carlo methods are being developed such as multilevel Monte Carlo and weighted multilevel Monte Carlo. There is a growing interest around these two methods and the research around this topic is a today's challenge.

The multilevel Monte Carlo methods may have a significant impact on reducing the computational cost by combining a set of estimators with a decreasing number of path simulations and increasing accuracy (number of time steps). The first one provides a rough estimation of a quantity of interest and the other ones are correcting this estimation.

In Giles [60] and [61], it is shown that the multilevel technique greatly improves performance compared to a standard Monte Carlo method. More recently, Lemaire and Pagès in [119] developed a new framework based on the multilevel method by adding weights for each estimators. The results show still a better convergence at a smaller cost when the parameter $\beta \leq 1$ for an optimal setting and they provided a set of applications to illustrate this case.

Historically, the multilevel approach has been developed on parametric integrations by Heinrich in [89] and [88]. The original multilevel Monte Carlo method was first introduced by Heinrich in [90] for the same problem. Later, it has been popularized by Giles in [60] for financial applications (evaluation of securities, greeks, VaR...) and extended to other domains of studies such as weather model ling, biological systems model ling etc...

There is wide literature on multilevel and applications to finance, we give some important developments but the list is non exhaustive[4]. For the pricing of derivatives Alaya and Kebaier [5], Altmayer and Neuenkirch [7], Giogi [67], Giles and Szpruch [64], Belomestny et al. [15] and [14]. Around the Brownian SDE studies and multilevel Monte Carlo, Giorgi et al. [68], Mbaye et al. [130], Speight [174], Giles and Waterhouse [65], Giles et al. [63] and Giles [59]. We, also, refer the interested reader to the key developments on multilevel by Giles [60] and [61], Lemaire and Pagès [119], Hahi-Ali et al. [85], Rhee and Glynn [165].

Concerning the weighted multilevel Monte Carlo (and the regular one), we refer the reader to the application developed by Daphné Giorgi `http://simulations.lpma-paris.fr/multilevel/`. It allows the user to test and analyze the results obtained for a various classes of derivatives[5] with the optimal parameters for the weighted and regular multilevel Monte Carlo methods. For now, there is only one discretization scheme available, i.e. Euler Maruyama scheme and one diffusion model i.e. Black Scholes model.

The computation of greeks with multilevel path simulation has been investigated in Burgos [36] and Burgos and Giles [37]. They combined Vibrato to the multilevel technique to approximate the Delta and the Vega of several exotic derivatives such as European Lookback Call,

[2]See `https://www2.isda.org/attachment/OTQzNg==/ISDA%20SA-CCR%20Briefing%20Paper%20-%20Final1.pdf`.

[3]See `https://www2.isda.org/attachment/NjE2Ng==/SIMM%20for%20Non-cleared%2020131210.pdf`.

[4]Retrieve the whole literature on Mike Giles multilevel Monte Carlo community page: `https://people.maths.ox.ac.uk/gilesm/mlmc_community.html`.

[5]Vanilla, Barrier, Lookback and Compound options.

Digital Call and European Barrier Call options. Here, we follow the same idea but we compute the second order sensitivities of vanilla/exotic derivatives without any changes in the payoff and with automatic differentiation.

In terms of coding, we just rewrite the payoff as a simple expression. Taking the example of the positive function x^+, we rewrite this function as $x \times H(x)$ where $H(\cdot)$ is the Heaviside function which takes the value 1 if $x > 0$ and 0 otherwise. The Heaviside function is implemented in the library of Automatic Differentiation and overloaded to compute the first and second order derivative by using a smoothing technique. We use a parametrized Gauss error function to smooth the discontinuity and make it $\mathcal{C}^\infty$.

The main focus here is the conception of a framework that computes the premium, first and second derivatives of a contingent claim with only one run using the same trajectories or at least a part of them for the sensitivities. Indeed, we will see that the sensitivities do not require the same number of path simulation than the value but more time steps. This framework are able to handle several set of optimal (or at least optimized) parameters for the value and its sensitivities at the same time.

This chapter is organized as follows; in sections 2.3 and 2.4 we begin by recalling the multilevel Monte Carlo method and the weighted multilevel Monte Carlo method. We give the optimal settings for both frameworks.

Following the work of Giorgi [67], we made some applications for the computations of sensitivities using antithetic scheme. In section 2.5, we introduce briefly Giles Szpruch antithetic meta-scheme, Milstein scheme and the corresponding framework with optimal settings for the regular and weighted multilevel Monte Carlo method. We make an application to some smooth payoff function in the Clark Cameron model, already tested in [66].

In section 2.6, we describe our framework to compute the price, first and second derivative of a financial security with the two methods. We give a description of the algorithm implemented and some details about code optimizations.

The section 2.7 is dedicated to numerical analysis of the computation of the Delta and the Gamma of vanilla/exotic option in the Heston model. Also, we provide some numerical results for the framework with the two methods for the approximation of the premium, the Delta and the Gamma of an European Call option using the same trajectories and a part of them for the different objectives. This last section deals with the numerical results of the computation of the Delta and the Gamma of a smooth payoff function for both method (regular and weighted multilevel Monte Carlo) with an antithetic scheme in the Clark Cameron model.

We conclude with a summary and some perspectives for future work.

2.2 Preliminaries

The main idea is to use a telescopic summation to compute a quantity of interest at a minimal cost for a fixed precision $\varepsilon > 0$. This quantity of interest can be represented as an expectation of a non-simulatable random variable Y_0. However, it takes advantage of the existence of a family of simulatable random variables Y_h, $h > 0$, both strongly and weakly approximates Y_0 as $h \to 0$. Let us explain and present a short description of the method. We adopt the same notations as the article of Lemaire and Pagès in [119]. Representation of the multilevel method on a simulation process (geometric Brownian motion) on figure 2.1.

Consider a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ and on which is defined a family $(Y_h)_{h\in\mathcal{H}}$ square integrable of real-valued random variables associated to a non degenerated $Y_0 \in L^2(\mathbb{P})$ where

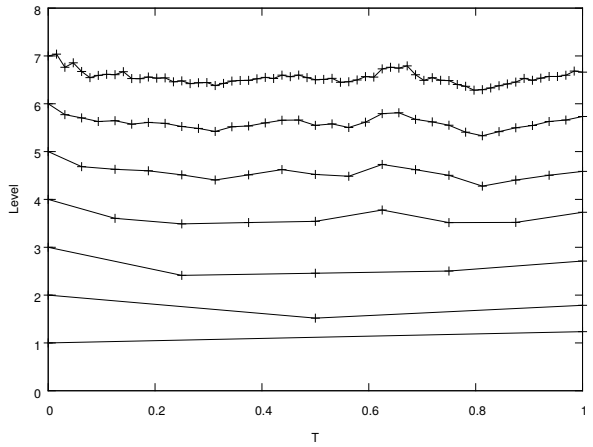

Figure 2.1: Representation of the multilevel simulation path.

$\mathcal{H} \in \left\{\frac{\mathbf{h}}{n}, n \geq 1\right\}$, $\mathbf{h} \in (0, +\infty)$ and satisfying

$$\lim_{h \to \mathcal{H}} \mathbb{E}\left[(Y_h - Y_0)^2\right] = 0.$$

The random variables Y_h often, but not always (think to nested Monte Carlo) come from a time discretization scheme with time step h and is called the bias parameter. We make the assumption that Y_h is easily simulatable (see further on for its precise complexity) whereas Y_0 is not, at least at a reasonable cost. We assume that the quadratic error is controlled by h^β, $\beta > 0$

$$(SE)_\beta \equiv \|Y_h - Y_0\|_2^2 = \mathbb{E}\left[|Y_h - Y_0|^2\right] \leq V_1 h^\beta$$

where V_1 is a positive constant real-valued. And it approximates Y_0 not only in *strong* quadratic sense but also in a *weak* sense. To be more precise, we assume that $\mathbb{E}[Y_h] - \mathbb{E}[Y_0]$ can be expanded when $h \to 0$ in $\mathcal{H}$ in the basis $(h^\alpha)_{k \geq 1}$, $\alpha > 0$, namely

$$(WE)_\alpha^L \equiv \mathbb{E}[Y_h] - \mathbb{E}[Y_0] = \sum_{k=1}^{L} c_k h^{\alpha k} + o(h^{\alpha L}), \quad L \geq 2, \tag{2.1}$$

where L denotes the depth of the multilevel framework as stated in Lemaire and Pagès [119] and c_k are real coefficient.

The starting point is the *bias-variance decomposition* of the error induced by using independent copies $(Y_h^k)_{k \geq 1}$ of Y_h to approximate the quantity of interest $I_0 = \mathbb{E}\, Y_0$ by the Monte Carlo estimator

$$\widehat{I}_N = \frac{1}{N} \sum_{k=1}^{N} Y_h^k,$$

namely

$$\left\| \mathbb{E}\left[Y_0\right] - \widehat{I}_N \right\|_2^2 = \underbrace{\left(\mathbb{E}\left[Y_0\right] - \mathbb{E}\left[Y_h\right]\right)^2}_{\text{squared bias}} + \underbrace{\frac{\mathrm{Var}(Y_h)}{N}}_{\text{Monte Carlo variance}}. \tag{2.2}$$

This decomposition is a straightforward consequence of the identity

$$\mathbb{E}\left[Y_0\right] - \frac{1}{N} \sum_{k=1}^{N} Y_h^k = \left(\mathbb{E}\left[Y_0\right] - \mathbb{E}\left[Y_h\right]\right) \overset{\perp}{+} \left(\mathbb{E}\left[Y_h\right] - \frac{1}{N} \sum_{k=1}^{N} Y_h^k\right).$$

The two terms are orthogonal in L^2 since $\mathbb{E}\left(\mathbb{E}\left[Y_h\right] - \frac{1}{N}\sum_{k=1}^{N} Y_h^k\right) = 0$.

Definition 3. *The squared quadratic error* $\|I_0 - \widehat{I}_N\|_2^2 = \mathbb{E}\left[(I_0 - \widehat{I}_N)^2\right]$ *is called the* Mean Squared Error *(MSE) of the estimator whereas the quadratic error itself* $\|I_0 - \widehat{I}_N\|_2$ *is called the* Rooted Mean Squared Error *(RMSE).*

In the following, the refiners have a geometric structure, namely

$$n_i = M^{i-1}, \quad i = 1, \dots, L. \tag{2.3}$$

Nevertheless, we will keep use the synthetic notation n_i and we note the refiners vector as the following.

Definition 4. *An increasing L-tuple* $\underline{n} = (n_1, \dots, n_R)$ *of positive integers satisfying* $1 = n_1 < n_2 < \cdots < n_R$ *is called a vector of* refiners. *For every* $h \in \mathcal{H}$*, the resulting sub-family of approximators is denoted* $Y_{\underline{n},h} := \left(Y_{\frac{h}{n_i}}\right)_{i=1,\dots,R}$.

2.3 Multilevel Monte Carlo with Optimal Parameters

2.3.1 Multilevel Setting

The basic principle of the multilevel paradigm is to split the Monte Carlo simulation into two parts: a first part made of *coarse* level based on an estimator of $Y_h = Y_{\frac{h}{n_1}}$ with a not too small $h \in \mathcal{H}$, that will achieve most part of the job to compute $I_0 = \mathbb{E}[Y_0]$ with a bias $\mathbb{E}[Y_h]$, and a second part made of several correcting *refined* levels relying on increments $Y_{\frac{h}{n_i}} - Y_{\frac{h}{n_{i-1}}}$ to correct the former bias. These increments being small – since they are close to Y_0 – consequently have small variances and need smaller simulated samples to accurately estimate their expectations i.e. the bias correction. By combining in an appropriate way these increments, one can almost "kill" the bias while keeping under control the variance and the complexity of the simulation. The penultimate goal is to minimize the cost of the method for a prescribed RMSE $\varepsilon > 0$. By making optimal balance between the bias term and the variance term thanks to appropriate choice of the parameter h and the size N of the simulation, one achieves this prescribed error bound at a minimal cost.

Let us assume that the weak error expansion holds simply at the first order

$$\mathbb{E}[Y_{\frac{h}{n_L}}] - \mathbb{E}[Y_0] = c_1 \left(\frac{h}{n_L}\right)^{\alpha} + o\left(\left(\frac{h}{n_L}\right)^{\alpha}\right),$$

Now, introducing artificially a telescopic sum, we get

$$\begin{aligned}
\mathbb{E}[Y_{\frac{h}{n_L}}] &= \mathbb{E}[Y_h] + \left(\mathbb{E}[Y_{\frac{h}{2}}] - \mathbb{E}\left[Y_h\right]\right) + \cdots + \left(\mathbb{E}[Y_{\frac{h}{n_L}}] - \mathbb{E}[Y_{\frac{h}{n_{L-1}}}]\right) \\
&= \mathbb{E}[Y_h] + \sum_{i=2}^{L} \mathbb{E}\left[Y_{\frac{h}{n_i}} - Y_{\frac{h}{n_{i-1}}}\right] \\
&= \mathbb{E}\left[\underbrace{Y_h}_{\text{coarse level}} + \sum_{i=2}^{L} \underbrace{Y_{\frac{h}{n_i}} - Y_{\frac{h}{n_{i-1}}}}_{\text{refined level}}\right].
\end{aligned}$$

We will assume again that the $Y_{\frac{h}{n_i}} - Y_{\frac{h}{n_{i-1}}}$ variables at each level i are sampled *independently i.e.*,

$$\mathbb{E}[Y_0] = \mathbb{E}\left[\underbrace{Y_h^{(1)}}_{\text{coarse level}} + \sum_{i=2}^{L} \underbrace{Y_{\frac{h}{n_i}}^{(i)} - Y_{\frac{h}{n_{i-1}}}^{(i)}}_{\text{refined level}}\right] - c_1\Big(\frac{h}{n_L}\Big)^{\alpha} + o\Big(\Big(\frac{h}{n_L}\Big)^{\alpha}\Big),$$

where the families $Y_{\underline{n},h}^{(i)}$, $i = 1, \ldots, L$, are independent. $Y_h^{(1)}$ has no reason to be small since it is close to a copy of Y_0, whereas, by contrast, $Y_{\frac{h}{n_i}}^{(i)} - Y_{\frac{h}{n_{i-1}}}^{(i)} \approx 0$ as the difference of two variables approximating Y_0. The seminal idea of the multilevel paradigm, introduced by Giles in [60], is to dispatch the L different simulation sizes N_i of each level $i = 1, \ldots, L$ so that $N_1 + \cdots + N_L = N$. This leads to the following formal definition.

Definition 5. *The family of (regular) multilevel estimators attached to $(Y_h)_{h\in\mathcal{H}}$ and the refiners vector $\underline{n}$ is defined by*

$$\widehat{I}_N^{MLMC} := \frac{1}{N_1}\sum_{k=1}^{N_1} Y_h^{(1),k} + \sum_{i=2}^{L}\frac{1}{N_i}\sum_{k=1}^{N_i}\Big(Y_{\frac{h}{n_i}}^{(i),k} - Y_{\frac{h}{n_{i-1}}}^{(i),k}\Big),$$

where $N_1, \ldots, N_L \in \mathbb{N}^*$, $N_1 + \cdots + N_L = N$, $\big(Y_{\underline{n},h}^{(i),k}\big)_{i=1,\ldots,L,k\geq 1}$ are i.i.d. copies of $Y_{\underline{n},h} := \big(Y_{\frac{h}{n_i}}\big)_{i=1,\ldots,L}$.

It is often convenient to re-write the above multilevel estimator as a stratified estimator *i.e.* set $q_i = \frac{N_i}{N}$, $i = 1, \ldots, L$ so that

$$\widehat{I}_N^{MLMC} = \widehat{I}_N^{MLMC}(L, q, h) = \frac{1}{N}\left[\sum_{k=1}^{N_1}\frac{Y_h^{(1),k}}{q_1} + \sum_{i=2}^{L}\sum_{k=1}^{N_i}\frac{1}{q_i}\big(Y_{\frac{h}{n_i}}^{(i),k} - Y_{\frac{h}{n_{i-1}}}^{(i),k}\big)\right], \quad q = (q_i)_{i=1,\ldots,L} \in \mathcal{S}_L.$$

with

$$q = (q_i)_{1\leq i\leq L} \in \mathcal{S}_L = \Big\{(u_i)_{1\leq i\leq L} \in (0,1)^L : \sum_i u_i = 1\Big\},$$

(so $\mathcal{S}_L$ denotes here the "open" simplex of $[0,1]^L$). If N is large enough we will adopt the reverse strategy: first select $q \in \mathcal{S}_L$ and then set $N_i = \lfloor q_i N \rfloor$, $i = 1, \ldots, L$.

Remark 9. *The estimator $\widehat{I}_N^{MLMC}$ does not only depend on N but also on the set of parameters $(h, q, \underline{n}, R)$ (coarse step, vector of paths allocation across the levels, vector of refiners and depth).*

▷ **Bias:** The telescopic/domino structure of the estimator implies (regardless of the sizes $N_i \geq 1$) that

$$\mathrm{Bias}\big(\widehat{I}_N^{MLMC}\big) = c_1\Big(\frac{h}{n_L}\Big)^{\alpha} + o\left(\Big(\frac{h}{n_L}\Big)^{\alpha}\right). \tag{2.4}$$

▷ **Variance:** Taking advantage of the independence of the levels, one straightforwardly compute the variance of $\widehat{I}_M^{MLMC}$

$$\mathrm{Var}(\widehat{I}_N^{MLMC}) \quad = \quad \frac{\mathrm{Var}(Y_h^1)}{N_1} + \sum_{i=2}^{L}\frac{\mathrm{Var}(Y_{\frac{h}{n_i}}^i - Y_{\frac{h}{n_{i-1}}}^i)}{N_i}.$$

▷ **Complexity:** The complexity, or simulation cost, of the MLMC estimator is clearly

$$\mathrm{Cost}(\widehat{I}_{N}^{MLMC}) = \frac{\bar{\kappa}N}{h}\sum_{i=1}^{L}\kappa_i q_i. \tag{2.5}$$

where $\kappa_1 = 1$, $\kappa_i = \left(1 + M^{-1}\right)n_i$, $i = 2, \ldots, L$.

Definition 6. *The effort of an estimator is given by the product of its complexity and its variance, namely*

$$\mathrm{Effort}(\widehat{I}^{MLMC}) = \frac{\bar{\kappa}}{h}\left(\sum_{i=1}^{L}\frac{\sigma^2(i,h)}{q_i}\right)\left(\sum_{i=1}^{L}\kappa_i q_i\right) = \kappa(\widehat{I}_{N}^{MLMC})\mathrm{Var}(\widehat{I}_{N}^{MLMC}), \quad M \geq 1. \tag{2.6}$$

Remark 10. *Note that linear Monte Carlo estimator, the effort does not depend on the size of N.*

Minimizing the effort for a fixed h (fixed L and a prescribed RMSE $\varepsilon > 0$) as a function of the vector $q \in \mathcal{S}_{L}$ and then minimizing the denominator of the above ratio cost in the bias parameter $h \in H$. It follows from the elementary lemma below, straightforward application of the Schwarz Inequality and its equality case (see Briane and Pagès [29]).

Lemme 1. *Let $a_i > 0$, $b_i > 0$ and $q \in \mathcal{S}_{R}$. Then*

$$\left[\sum_{i=1}^{R}\frac{a_i}{q_i}\right]\left[\sum_{i=1}^{R}b_i q_i\right] \geq \left[\sum_{i=1}^{R}\sqrt{a_i b_i}\right]^2,$$

and equality holds if and only if $q_i = q^{\dagger}\sqrt{a_i b_i^{-1}}$, $i = 1, \ldots, R$, with $q^{\dagger} = q^{\dagger}(a,b) = \left(\sum_{i=1}^{R}\sqrt{a_i b_i^{-1}}\right)^{-1}$.

Applying this to 2.6, we get the solution to the effort minimization problem:

$$\mathrm{argmin}_{q\in\mathcal{S}_R}\mathrm{Effort}(\widehat{I}_{N}^{MLMC}) = q^{*} = q^{*,\dagger}(h)\left(\frac{\sigma(i,h)}{\sqrt{\kappa_i}}\right)_{i=1,\ldots,R},$$

with $q^{*,\dagger}(h) = \left(\sum_{1\leq i\leq L}\frac{\sigma(i,h)}{\sqrt{\kappa_i}}\right)^{-1}$ and where we set $\sigma^2(1,h) = \mathrm{Var}(Y_h)$ and $\sigma^2(i,h) = \mathrm{Var}\left(Y_{\frac{h}{n_i}} - Y_{\frac{h}{n_{i-1}}}\right)$, $i = 2, \ldots, L$ with a resulting minimal effort:

$$\min_{q\in\mathcal{S}_R}\mathrm{Effort}(\widehat{I}_{N}^{MLMC}) = \frac{\bar{\kappa}}{h}\left(\sum_{i=1}^{R}\sqrt{\kappa_i}\,\sigma(i,h)\right)^2.$$

Minimizing the resulting cost and compiling the above results, we can show that

$$\inf_{RMSE(\widehat{I}_{N}^{MLMC})\leq\varepsilon}\mathrm{Cost}(\widehat{I}_{N}^{MLMC}) \leq \bar{\kappa}\frac{(1+2\alpha)^{1+\frac{1}{2\alpha}}}{2\alpha}\frac{|c_1|^{\frac{1}{\alpha}}}{n_{L}}\frac{\left(\sum_{1\leq i\leq L}\sqrt{\kappa_i}\sigma(i,h^{*}(\varepsilon))\right)^2}{\varepsilon^{2+\frac{1}{\alpha}}}, \tag{2.7}$$

when $\varepsilon \to 0$, $h_{\max}(\varepsilon) \to 0$ and $h^{*}(\varepsilon) \sim h_{\max}(\varepsilon)$. Moreover,

$$h_{\max}(\varepsilon) = \left(\frac{\varepsilon}{|c_1|}\right)^{\frac{1}{\alpha}}\frac{n_L}{(1+2\alpha)^{\frac{1}{2\alpha}}},$$

so that we are led to set

$$h^*(\varepsilon) = \text{lower nearest neighbor of } \; h_{\max}(\varepsilon) \text{ in } \mathcal{H} = \mathbf{h}\lceil \mathbf{h}/h_{\max}(\varepsilon)\rceil^{-1}. \tag{2.8}$$

The minimization of the upper-bound (2.7) with respect to the depth L under the additional assumption ($Var(Y_h - Y_0) \leq V_1 h^\beta$, $\forall h \in \mathcal{H}$) which provides a non-asymptotic control of the variance of the Y_h. It is obvious that, when $L \uparrow +\infty$ for a fixed $\varepsilon > 0$, the upper bound of the complexity in (2.26) goes to 0 since $n_{_L} = M^{L-1} \uparrow +\infty$. For technical reason detailed in Lemaire and Pagès [119], A natural way to specify the depth L is as high as possible, as long as $h^*(\varepsilon)$ lies in $\mathcal{H}$. This is equivalent to $h_{\max}(\varepsilon) \leq \mathbf{h}$ which in turn reads, owing to (2.8),

$$L^*(\varepsilon) = 1 + \left\lceil \frac{\log\left(|c_1|^{\frac{1}{\alpha}}\mathbf{h}\right)}{\log(M)} + \frac{\log\left((1+2\alpha)^{\frac{1}{2}}/\varepsilon\right)}{\alpha\log(M)} \right\rceil.$$

It remains to inspect three cases, depending on the parameter β ($\beta > 1$, $\beta = 1$ and $\beta \in (0,1)$) to get the following theorem.

Theorem 1 (MLMC estimator, see Giles [60]). *Assume $n_i = M^{i-1}$, $i = 1, \ldots, L$ and $(WE)_1^\alpha$. The MLMC estimator $(\widehat{I}_{_N}^{MLMC})_{N\geq 1}$ satisfies*

$$\inf_{\substack{h\in\mathcal{H},\, q\in\mathcal{S}_L,\, L\geq 1 \\ \|\bar{Y}^{N,q}_{h,\underline{n}} - I_0\|_2^2 \leq \varepsilon^2}} \mathrm{Cost}\big(\widehat{I}_{_N}^{MLMC}\big) \;\lesssim K^{MLMC}_{\alpha,\beta,M,\theta} \frac{\mathrm{Var}(Y_0)}{\varepsilon^2} \times \begin{cases} 1 & \text{if } \beta > 1, \\ \big(\log(1/\varepsilon)\big)^2 & \text{if } \beta = 1, \\ \varepsilon^{-\frac{1-\beta}{\alpha}} & \text{if } \beta < 1. \end{cases}$$

Closed form for the depth $L^*(\varepsilon)$, the bias parameter $h^* = h^*(\varepsilon, L(\varepsilon)) \in \mathcal{H}$, the optimal allocation vector $q(\varepsilon)$ are given in table 2.2.

The $*$ in subscript are "dropped" to alleviate notations in the next Table:

2.4 Weighted Multilevel Monte Carlo Method with Optimal Parameters

2.4.1 Richardson-Romberg Extrapolation

Now, we assume that the weak error expansion of $\mathbb{E}[Y_h]$ in the $(h^{\alpha\ell})_{\ell\geq 1}$ scale holds at the second order *i.e.*

$$(WE)_2^\alpha \;\equiv\; \mathbb{E}[Y_h] = \mathbb{E}[Y_0] + c_1 h^\alpha + c_2 h^{2\alpha} + o\big(h^{2\alpha}\big), \tag{2.9}$$

and that $\mathrm{Var}(Y_h) \to \mathrm{Var}(Y_0)$ holds as well as $\bar{\sigma}^2 = \sup_{h\in\mathcal{H}} \mathrm{Var}(Y_h) < +\infty$.

One derives by linearly combining (2.9) with h and $\frac{h}{2}$ that

$$2^\alpha\, \mathbb{E}[Y_{\frac{h}{2}}] - \mathbb{E}[Y_h] = \mathbb{E}[Y_0] + (2^{-\alpha} - 1) c_2 h^{2\alpha} + o\big(h^{2\alpha}\big). \tag{2.10}$$

$L(\varepsilon)$	$\left\lceil 1 + \dfrac{\log\left(\lvert c_1\rvert^{\frac{1}{\alpha}}\mathbf{h}\right)}{\log(M)} + \dfrac{\log\left((1+2\alpha)^{\frac{1}{2}}/\varepsilon\right)}{\alpha\log(M)} \right\rceil$
$h(\varepsilon)$	$\mathbf{h}/\left\lceil \mathbf{h}(1+2\alpha)^{\frac{1}{2\alpha}}\varepsilon^{-\frac{1}{\alpha}}\lvert c_1\rvert^{\frac{1}{\alpha}}M^{-(L-1)}\right\rceil^{-1}$
$q(\varepsilon)$	$q_1(\varepsilon) = q^\dagger(1+\theta h^{\frac{\beta}{2}}), \quad \theta = \sqrt{\dfrac{V_1}{\mathrm{Var}(Y_0)}},$ $q_j(\varepsilon) = q^\dagger\theta h^{\frac{\beta}{2}}\dfrac{n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}}{\sqrt{n_{j-1}+n_j}},\ j=2,\dots,L,\ q^\dagger \text{ s.t. } \sum_{j=1}^{L} q_j(\varepsilon) = 1$
$N(\varepsilon)$	$\left\lceil \left(1+\dfrac{1}{2\alpha}\right)\dfrac{\mathrm{var}(Y_0)\left(1+\theta h^{\frac{\beta}{2}}\sum_{j=1}^{L}\left(n_{j-1}^{-\frac{\beta}{2}}+n_j^{-\frac{\beta}{2}}\right)\sqrt{n_{j-1}+n_j}\right)^2}{\varepsilon^2 q^\dagger} \right\rceil$

Figure 2.2: Parameters of the MLMC estimator (standard framework).

This suggests a natural way to modify the estimator of $I_0 = \mathbb{E}\,Y_0$ to reduce the bias: set

$$\widetilde{Y}_h = \frac{2^\alpha Y_{\frac{h}{2}} - Y_h}{2^\alpha - 1}, \quad h \in \mathcal{H}.$$

It is clear from (2.10) that $(\widetilde{Y}_h)_{h\in\mathcal{H}}$ satisfies $(WE)_1^{2\alpha}$. Without additional assumptions $\widetilde{Y}_h$ does not satisfy Assumption that the $\mathrm{Var}(Y_h)$ converges towards $\mathrm{Var}(Y_0)$ as $h \to 0$, however its variance remains bounded if that of Y_h does since

$$\begin{aligned}\mathrm{Var}(\widetilde{Y}_h) = \sigma(\widetilde{Y}_h)^2 &\leq \frac{1}{(2^\alpha-1)^2}\left(2^\alpha\sigma(Y_{\frac{h}{2}}) + \sigma(Y_h)\right)^2 \\ &\leq \left(\frac{2^\alpha+1}{2^\alpha-1}\right)^2\bar{\sigma}^2\end{aligned}$$

Also note that the simulation cost $\tilde{\kappa}(h)$ of $\widetilde{Y}_h$ satisfies $\tilde{\kappa}(h) = \kappa(h) + 2\kappa(h) = \frac{3\bar{\kappa}}{h}$. This leads to introduce, for every $h\in\mathcal{H}$, the *Richardson-Romberg estimator*

$$\widetilde{I}_N^{RR} = \widetilde{I}_N^{RR}(h,N) = \frac{1}{N}\sum_{k=1}^{N}\widetilde{Y}_h^k = \frac{1}{N(2^\alpha-1)}\sum_{k=1}^{N} 2^\alpha Y_{\frac{h}{2}}^k - Y_h^k, \quad N \geq 1,$$

where $(Y_h^k, Y_{\frac{h}{2}}^k)_{k\geq 1}$ are independent copies of $(Y_h, Y_{\frac{h}{2}})$. Applying the results of the former section

to the family $(\widetilde{Y}_h)_{h\in\mathcal{H}}$ with $\tilde{\alpha}=2\alpha$, $\tilde{\kappa}(h)$, $\tilde{c}_1=-(1-2^{-\alpha})c_2$, we can prove that

$$\inf_{RMSE(\widetilde{I}_N^{RR})\le\varepsilon}\mathrm{Cost}(\widetilde{I}_N^{RR})\le\frac{(1+4\alpha)^{1+\frac{1}{4\alpha}}}{4\alpha}\frac{3(2^\alpha+1)^2[(1-2^{-\alpha}|c_2|]^{\frac{1}{2\alpha}}}{(2^\alpha-1)^2}\frac{\bar{\kappa}\,\bar{\sigma}^2}{\varepsilon^{2+\frac{1}{2\alpha}}}.$$

2.4.2 The Multistep Approach

In this section, we extend the above Richardson-Romberg method as in Pagès [139] by introducing a multistep approach based on a higher order extension of the weak error $\mathbb{E}[Y_h]-\mathbb{E}[Y_0]$. Let us assume that the following assumption holds for some integer $R\ge 2$

$$(WE)_R^\alpha\ \equiv\ \mathbb{E}[Y_h]=\mathbb{E}[Y_0]+\sum_{r=1}^{R}c_r h^{\alpha r}+o\big(h^{\alpha R}\big). \tag{2.11}$$

Like for all Taylor like expansions of this type, the coefficients c_r are uniquely defined and do not depend on $R\ge r$. We also make the quadratic convergence assumption ($Y_h\to Y_0$ in L^2) as well, though it is not mandatory in this setting. The driving idea is, a vector $\underline{n}$ of refiners being fixed, to extend the Richardson-Romberg estimator by searching a linear combination of the components of $\mathbb{E}[Y_{\frac{h}{n_r}}]$ which kills the resulting bias up to order $R-1$, relying on the expansion $(WE)_R^\alpha$, then, derive the multistep estimator accordingly. To determine this linear combination, whose coefficients hopefully will not depend on $h\in\mathcal{H}$, we consider a generic R-tuple of *weights* $w=(w_1,\ldots,w_R)\in\mathbb{R}^R$. Then, owing to $(WE)_R^\alpha$,

$$\sum_{j=1}^{R}w_j\mathbb{E}\,Y_{\frac{h}{n_j}}=\Big(\sum_{j=1}^{R}w_j\Big)\mathbb{E}\,Y_0+\sum_{r=1}^{R}c_r h^{\alpha r}\left[\sum_{j=1}^{R}\frac{w_j}{n_j^{\alpha r}}\right]+o\Big(h^{\alpha R}\Big)$$

where we interchanged the two sums in the second line. If w is solution to

$$\sum_{j=1}^{R}\frac{w_j}{n_j^{\alpha(r-1)}}=0^{r-1},\ r=1,\ldots,R, \tag{2.12}$$

then (2.12) reads

$$\sum_{j=1}^{R}w_j\mathbb{E}\,Y_{\frac{h}{n_j}}=\widetilde{W}_{R+1}c_R h^{\alpha R}+o\Big(h^{\alpha R}\Big),\quad \widetilde{W}_{R+1}=\sum_{j=1}^{R}\frac{w_j}{n_j^{\alpha R}}.$$

The linear system (2.12) is obviously of Vandermonde type, namely $\mathrm{Vand}(x_1,\ldots,x_{_R})w=[c^{i-1}]_{i=1:R}$ where the Vandermonde matrix attached to a R-tuple $(x_1,\ldots,x_{_R})\in(\mathbb{R}^+)^R$ is defined by $\mathrm{Vand}(x_1,$
$\ldots,x_R)=[x_j^{i-1}]_{i,j=1:R}$ and $c\in\mathbb{R}$. In our case, $x_i=n_i^{-\alpha}$, $i=1,\ldots,R$ and $c=0$. As a consequence, Cramer's formulas yield

$$w_i=\frac{\det[\mathrm{Vand}(n_1^{-\alpha},\ldots,n_{i-1}^{-\alpha},0,n_{i+1}^{-\alpha},\ldots,n_{_R}^{-\alpha})]}{\det[\mathrm{Vand}[(n_1^{-\alpha},\ldots,n_i^{-\alpha},\ldots,n_R^{-\alpha})]},\quad i=1,\ldots,R.$$

As a consequence, the weight vector solution to (2.12) has a closed form since it is classical background that

$$\forall\,x_1,\ldots,x_{_R}>0,\quad \det[\mathrm{Vand}(x_1,\ldots,x_i,\ldots,x_R)]=\prod_{1\le i<j\le R}(x_j-x_i).$$

Synthetic formulas are given in the following proposition for w and $\widetilde{W}_{R+1}$.

Proposition 1. (*a*) *For every fixed integer $R \geq 2$, the weight vector w admits a closed form given by*

$$\forall i \in \{1, \dots, R\}, \quad w_i = \frac{n_i^{\alpha(R-1)}}{\prod_{j \neq i} n_i^\alpha - n_j^\alpha} = (-1)^{R-i} \frac{1}{\prod_{j \neq i} |1 - (n_j/n_i)^\alpha|}. \tag{2.13}$$

(*b*) *Furthermore*

$$\widetilde{W}_{R+1} = \frac{(-1)^{R-1}}{\underline{n}!^\alpha} \quad \text{where} \quad \underline{n}! = \prod_{i=1}^R n_i. \tag{2.14}$$

As a straightforward consequence of the former proposition, we get with the weights given by (2.13),

$$\sum_{i=1}^R w_i \, \mathbb{E}\, Y_{\frac{h}{n_i}} = \mathbb{E}\, Y_0 + (-1)^{R-1} \frac{c_R}{\underline{n}!^\alpha} h^{\alpha R} + o\big(h^{\alpha R}\big). \tag{2.15}$$

Remark 11. *The main point to be noticed is the universality of these weights which do not depend on $h \in \mathcal{H}$ nor the refiners $\underline{n}$. Not either on the Y_h themselves, except through the exponent α of the weak error expansion.*

These computations naturally suggest the following definition for a multistep estimator at order R.

Definition 7 (Multistep estimator). *The multistep estimators at order R associated to the vector of refiners $\underline{n}$ is defined for every $h \in \mathcal{H}$ and every simulation size $N \geq 1$ by*

$$\widehat{I}_N^{RR} = \widehat{I}_N^{RR}(R, \underline{n}, h, N) \quad = \quad \frac{1}{N} \sum_{i=1}^R w_i \sum_{k=1}^N Y_{\frac{h}{n_i}}^k \tag{2.16}$$

where $Y_{\underline{n},h}^k$ are independent copies of $Y_{\underline{n},h} = \big(Y_{\frac{h}{n_i}}\big)_{i=1,\dots,R}$ and the weight vector w is given by (2.13)*. The parameter R is called the* depth *of the estimator.*

2.4.3 Weighted Multilevel Setting

Let us assume

$$\mathbb{E}\,[Y_h] = \mathbb{E}\,[Y_0] + \sum_{r=1}^R c_r h^{\alpha r} + o\big(h^{\alpha R}\big). \tag{2.17}$$

Then, starting again from the weighted combination (2.12) of the elements of $Y_{\underline{n},h}$ with the weight vector w given by (2.13), one gets by an Abel transform

$$\begin{aligned}
\sum_{i=1}^R w_i \mathbb{E}\,[Y_{\frac{h}{n_i}}] &= W_1 \mathbb{E}\,[Y_h] + \sum_{i=2}^R W_i \Big(\mathbb{E}\,[Y_{\frac{h}{n_i}}] - \mathbb{E}\,[Y_{\frac{h}{n_{i-1}}}] \Big) \\
&= W_1 \mathbb{E}\,[Y_h] + \sum_{i=2}^R W_i \mathbb{E}\Big[Y_{\frac{h}{n_i}} - Y_{\frac{h}{n_{i-1}}} \Big] \\
&= \mathbb{E}\left[\underbrace{Y_h^{(1)}}_{\text{coarse level}} + \sum_{i=2}^R W_i \underbrace{\Big(Y_{\frac{h}{n_i}}^{(i)} - Y_{\frac{h}{n_{i-1}}}^{(i)} \Big)}_{\text{refined level}} \right]
\end{aligned} \tag{2.18}$$

with

$$W_i = w_i + \cdots + w_{_R}, \quad i = 1, \ldots, R. \tag{2.19}$$

Note that $W_1 = w_1 + \cdots + w_{_R} = 1$. In the last line, the families $(Y^{(i)}_{\underline{n},h})_{i=1,\ldots,R}$, are independent copies of $Y_{\underline{n},h}$. As the weights satisfy the Vandermonde system (2.12), we derive from (2.18) that

$$\mathbb{E}[Y_h^{(1)}] + \sum_{i=2}^{R} W_i\, \mathbb{E}\left[Y^{(i)}_{\frac{h}{n_i}} - Y^{(ji)}_{\frac{h}{n_{i-1}}}\right] = \mathbb{E}[Y_0] + \widetilde{W}_{R+1} c_R h^{\alpha R} + o\Big(h^{\alpha R}\Big) \tag{2.20}$$

where $\widetilde{W}_{_{R+1}} = (-1)^{R-1} M^{-\frac{R(R-1)}{2}\alpha}$ since $n_i = M^{i-1}$. As already noted $\mathbb{E}[Y_h^{(1)}] \approx I_0$ and $Y_h^{(1)}$ is close to (a copy of) Y_0 so that it has no reason to be small whereas $Y^{(i)}_{\frac{h}{n_i}} - Y^{(i)}_{\frac{h}{n_{i-1}}} \approx 0$ since both quantities are supposed to be close to (copies of) Y_0. The underlying idea to devise the regular Multilevel estimator is to calibrate the R different sizes N_i of sample paths assigned to each level $i = 1, \ldots, R$ so that $N_1 + \cdots + N_{_R} = N$. This leads to the following general definition of a *Multilevel Richardson-Romberg* estimator.

Definition 8. *The family of weighted Multilevel estimator or Multi-Level Richardson-Romberg (ML2R) estimators attached to $(Y_h)_{h\in\mathcal{H}}$ is defined, for every $h \in \mathcal{H}$, by*

$$\widehat{I}^{ML2R}_{_N} = \widehat{I}^{ML2R}_{_N}(R, q, h) := \frac{1}{N_1}\sum_{k=1}^{N_1} Y_h^{(1),k} + \sum_{i=2}^{R} \frac{W_i}{N_i} \sum_{k=1}^{M_i} \Big(Y^{(i),k}_{\frac{h}{n_i}} - Y^{(i),k}_{\frac{h}{n_{i-1}}}\Big),$$

where $N_1, \ldots, N_{_R} \in \mathbb{N}^*$ and $N_1 + \cdots + N_{_R} = N$, $\big(Y^{(i),k}_{\underline{n},h}\big)$, $i = 1, \ldots, R$, $k \geq 1$, are i.i.d.copies of $Y_{\underline{n},h} := \big(Y_{\frac{h}{n_i}}\big)_{i=1,\ldots,R}$.

At this stage, it is convenient to re-write the above multilevel estimator as a stratified estimator *i.e.* set $q_i = \frac{N_i}{N}$, $i = 1, \ldots, R$ so that

$$\widehat{I}^{ML2R}_{_N} := \frac{1}{N}\left[\sum_{k=1}^{N_1} \frac{Y_h^{(1),k}}{q_1} + \sum_{i=2}^{R}\sum_{k=1}^{N_i} \frac{W_i}{q_i}\Big(Y^{(i),k}_{\frac{h}{n_i}} - Y^{(i),k}_{\frac{h}{n_{i-1}}}\Big)\right], \quad N \geq R,$$

If N is large enough it is possible to adopt the reverse convention and to consider $q = (q_i)_{i=1,\ldots,R}$ as a $\mathcal{S}_R$-valued vector and set $N_i = \lfloor q_i N \rfloor$, $i = 1, \ldots, R$.

Remark 12. *Like the multilevel estimator, the estimator $\widehat{I}^{ML2R}_{_N}$ not only depend on N but also on the set of parameters $(h, q, \underline{n}, R)$.*

We are now in position to evaluate the basic characteristics of this estimator.

▷ **Bias:** The estimator satisfies (regardless of the simulation sizes $N_i \geq 1$ of the levels...)

$$\mathrm{Bias}\big(\widehat{I}^{ML2R}_{_N}\big) = \mathrm{Bias}\big(\widehat{I}^{RR}_{_N}\big) = (-1)^{R-1} c_{_R}\Big(\frac{h^R}{\underline{n}!}\Big)^{\alpha} + o\left(\Big(\frac{h^R}{\underline{n}!}\Big)^{\alpha}\right). \tag{2.21}$$

▷ **Variance:** Taking advantage of the mutual independence of $(Y^{(j)}_{\underline{n},h})_{i=1,\ldots,R}$ across the levels, one straightforwardly compute the variance of $\widehat{I}^{ML2R}_{_N}$

$$\mathrm{Var}(\widehat{I}^{ML2R}_{_N}) = \frac{\mathrm{Var}(Y_h^1)}{N_1} + \sum_{i=2}^{R} W_i^2 \frac{\mathrm{Var}\big(Y^i_{\frac{h}{n_i}} - Y^i_{\frac{h}{n_{i-1}}}\big)}{N_i} = \frac{1}{N}\sum_{i=1}^{R} \frac{\sigma^2_{_W}(i,h)}{q_i}, \tag{2.22}$$

where we set $\sigma^2_{_W}(1,h) = \mathrm{Var}(Y_h)$ and $\sigma^2_{_W}(i,h) = W_i^2 \mathrm{Var}\Big(Y_{\frac{h}{n_i}} - Y_{\frac{h}{n_{i-1}}}\Big)$, $i = 2, \ldots, R$.

▷ **Complexity:** The complexity of such an estimator is *a priori* given — or at least dominated – by

$$\mathrm{Cost}(\widehat{I}_N^{ML2R}) = \bar{\kappa}\left(\frac{N_1}{h} + \sum_{i=2}^{R} N_i\Big(\frac{n_i}{h} + \frac{n_{i-1}}{h}\Big)\right) = \frac{\bar{\kappa}N}{h}\sum_{i=1}^{R}\kappa_i q_i, \quad n_i = M^{i-1},$$

with $\kappa_1 = 1,\ \kappa_i = \big(1 + N^{-1}\big)n_i,\ i = 2, \dots, R$. As defined for the multilevel estimator, the effort for the weighted multilevel estimator is given by:

$$\mathrm{Effort}(\widehat{I}^{ML2R}) = \mathrm{Cost}(\widehat{I}_N^{ML2R})\mathrm{Var}(\widehat{I}_N^{ML2R}) = \mathrm{Cost}(\widehat{I}_1^{ML2R})\mathrm{Var}(\widehat{I}_1^{ML2R}), \quad N \geq 1. \tag{2.23}$$

The bias-variance decomposition of the weighted multilevel estimator $\widehat{I}_N^{ML2R}$ and the fact that the $RMSE$ constraint should be saturated to maximize the denominator of the ratio in

$$\mathrm{Cost}(\tilde{Y}_M^{ML2R}) = \frac{\mathrm{Cost}(\tilde{Y}_M^{ML2R})\mathrm{Var}(\tilde{Y}_M^{ML2R})}{\mathrm{Var}(\tilde{Y}_M^{ML2R})} \tag{2.24}$$

which allows to reformulate our minimization problem as

$$\inf_{\|\widehat{I}_N^{ML2R} - I_0\|_2 \leq \varepsilon} \mathrm{Cost}(\widehat{I}_N^{ML2R}) = \inf_{|\mathrm{Bias}(\widehat{I}_N^{ML2R})| < \varepsilon} \left[\frac{\mathrm{Effort}(\widehat{I}^{ML2R})}{\varepsilon^2 - \mathrm{Bias}\big(\widehat{I}_N^{ML2R}\big)^2}\right]. \tag{2.25}$$

For following the same step for minimizing the effort in the section 2.3.1, we get the solution to the effort minimization problem:

$$\mathrm{argmin}_{q \in \mathcal{S}_R}\mathrm{Effort}(\widehat{I}_N^{ML2R}) = q^* = q^{*,\dagger}(h)\left(\frac{\sigma_{_W}(i,h)}{\sqrt{\kappa_i}}\right)_{i=1,\dots,R}$$

with $q^{*,\dagger}(h) = \left(\sum_{1 \leq i \leq R} \frac{\sigma(i,h)}{\sqrt{\kappa_i}}\right)^{-1}$ with a resulting minimal effort:

$$\min_{q \in \mathcal{S}_R} \mathrm{Effort}(\widehat{I}_N^{ML2R}) = \frac{\bar{\kappa}}{h}\left(\sum_{i=1}^{R}\sqrt{\kappa_i}\,\sigma_{_W}(i,h)\right)^2.$$

Minimizing the resulting cost and compiling the above results (effort minimization, expansion of the bias, et the global optimization problem 2.25), we can show that

$$\inf_{RMSE(\widehat{I}_N^{ML2R}) \leq \varepsilon} \mathrm{Cost}^{ML2R}(h, N, q) \leq \bar{\kappa}\frac{(1+2\alpha R)^{1+\frac{1}{2\alpha R}}}{2\alpha R}\frac{|c_{_R}|^{\frac{1}{\alpha R}}}{\underline{n}!^{\frac{1}{R}}}\frac{\left(\sum_{1 \leq i \leq R}\sqrt{\kappa_i}\sigma_{_W}(i, h^*(\varepsilon))\right)^2}{\varepsilon^{2+\frac{1}{\alpha R}}}, \tag{2.26}$$

$$h_{\max}(\varepsilon) = \left(\frac{\varepsilon}{|c_{_R}|}\right)^{\frac{1}{\alpha R}}\frac{\underline{n}!^{\frac{1}{R}}}{(1+2\alpha R)^{\frac{1}{2\alpha R}}}, \tag{2.27}$$

so that we are led to set

$$h^*(\varepsilon) = \text{lower nearest neighbor of } \ h_{\max}(\varepsilon) \text{ in } \mathcal{H} = \mathbf{h}\lceil \mathbf{h}/h_{\max}(\varepsilon)\rceil^{-1}.$$

Our aim here is to optimize the depth R of the estimator, once the above optimization of both the effort and the step, have been performed. This will lead to an unexpected result, in particular we will see that the parameter $h^*(\varepsilon)$ will no longer fade as $\varepsilon \to 0$. To achieve this optimization

we need to have precise non-asymptotic bounds on the optimized effort. First, we need to make one last additional assumption, namely that the weak error expansion holds true at any order and that coefficients c_r do not grow too fast at infinity when $r \to \infty$, namely

$$(WE)_{\infty}^{\alpha} \equiv (WE)_{R}^{\alpha} \text{ holds for every depth } R \geq 2 \text{ and } \tilde{c}_{\infty} = \overline{\lim_{R \to +\infty}} |c_R|^{\frac{1}{R}} \in (0, +\infty). \quad (2.28)$$

It is obvious that, when $R \to +\infty$ for a fixed $\varepsilon > 0$, the upper bound of the complexity in (2.26) goes to 0 since $\frac{(1+2\alpha R)^{1+\frac{1}{2\alpha R}}}{2\alpha R} \to 1$, $|c_R|^{\frac{1}{\alpha R}}$ remains bounded whereas $\underline{n}!^{\frac{1}{R}} = M^{\frac{R-1}{2}} \uparrow +\infty$. On the other hand $\varepsilon^{2+\frac{1}{\alpha R}} \to \varepsilon^2$. It appears clearly that the best choice is to consider R as high as possible so that $h^*(\varepsilon)$ lies in $\mathcal{H}$ which is equivalent to $h_{\max}(\varepsilon) \leq \mathbf{h}$. For technical reasons detailed in Lemaire and Pagès [119], the best choice for $R^*(\varepsilon)$ is to adopt the upper integer value of the above bound.

$$R^*(\varepsilon) = \left\lceil \frac{1}{2} + \frac{\log(\mathbf{h}\,\tilde{c}_{\infty}^{\frac{1}{\alpha}})}{\log(M)} + \sqrt{\Big(\frac{1}{2} + \frac{\log(\mathbf{h}\,\tilde{c}_{\infty}^{\frac{1}{\alpha}})}{\log(M)}\Big)^2 + \frac{2\log(\frac{\sqrt{1+4\alpha}}{\varepsilon})}{\alpha \log(M)}} \right\rceil. \quad (2.29)$$

The first quantity to be analyzed is of course the family of weights $W^{(R)} = (W_i^{(R)})_{i=1,\dots,R}$ as R grows. The lemma thereafter shows that they remain uniformly bounded as R grows.

Lemme 2. *Let $\alpha > 0$ and the associated weights $(W_j^{(R)})_{i=1,\dots,R}$ defined by 2.19 with $n_i = N^{i-1}$.*

(a) Let $a_\ell = \prod_{1 \leq k \leq \ell-1} (1 - N^{-k\alpha})^{-1}$, $\ell \geq 1$ and $b_\ell = (-1)^\ell M^{-\frac{\alpha}{2}\ell(\ell+1)}$.

$$W_j^{(R)} = \sum_{\ell=j}^{R} a_\ell a_{R-\ell+1} b_{R-\ell} = \sum_{\ell=0}^{R-j} a_{R-\ell} a_{\ell+1} b_\ell$$

(b) The sequence $\alpha_\ell \uparrow a_\infty = \prod_{1 \leq k \leq \ell-1}(1 - M^{-k\alpha})^{-1}$ as $\ell \uparrow +\infty$ and $\widetilde{B}_\infty = \sum_{\ell=0}^{+\infty} |b_\ell| < +\infty$ so that the weights $W_j^{(R)}$ are uniformly bounded and $\forall R \in \mathbb{N}^*, \forall j \in \{1, \dots, R\}$, $|W_j^{(R)}| \leq a_\infty^2 \widetilde{B}_\infty < +\infty$.

Now, we have to inspect the behavior of $\sum_{i=1}^{R} \kappa_i \sigma_W(i, h^*(\varepsilon))$ depending on the value of β. Given 2.23, one has to inspect three different settings:

- $\beta > 1$: *fast strong approximation* (corresponding *e.g.* to the use of Milstein scheme for vanilla payoffs in local volatility model),
- $\beta = 1$: *regular strong approximation* (corresponding *e.g.* to the use of the Euler scheme for vanilla or lookback payoffs in local volatility model),
- $\beta < 1$: *slow strong approximation* (corresponding *e.g.* to the use of the Euler scheme for payoffs including barriers in local volatility model).

Combining all what precedes, elementary though technical computations (see Lemaire and Pagès [119] for details), lead to the following theorem.

Theorem 2 (ML2R estimator, Lemaire and Pagès [119]). *Let $n_i = M^{i-1}$, $i \geq 1$. Assume $(WE)^{\alpha}_{\infty}$ (see (2.28)). Assume $\lim_{R\to+\infty} |c_R|^{1/R} = \widetilde{c}_{\infty} \in (0, +\infty)$.*

(a) The ML2R estimator satisfies, as $\varepsilon \to 0$,

$$\inf_{\substack{h\in\mathcal{H},q\in\mathcal{S}_R,R\geq 1,\\ \|\widehat{I}_N^{ML2R}-I_0\|_2\leq\varepsilon}} \mathrm{Cost}(\widehat{I}_N^{ML2R}) \precsim K^{ML2R}_{\alpha,\beta,M,\theta} \frac{\mathrm{Var}(Y_0)}{\varepsilon^2} \times \begin{cases} 1 & \text{if } \beta > 1, \\ \log(1/\varepsilon) & \text{if } \beta = 1, \\ e^{\frac{1-\beta}{\sqrt{\alpha}}\sqrt{2\log(1/\varepsilon)\log(M)}} & \text{if } \beta < 1. \end{cases}$$

(b) When $\beta < 1$ the optimal rate is achieved with $M = 2$.

(c) Closed forms are available for $R^*(\varepsilon)$, $h^* = h^*(\varepsilon, R^*(\varepsilon)) \in \mathcal{H}$, $N(\varepsilon)$ and are available in table 2.3

Remark 13.

- *The MLMC estimator achieves the rate ε^{-2} of an unbiased simulation as soon as $\beta > 1$ (fast strong rate) under lighter conditions than ML2R since it only requires a first order weak error expansion.*
- *When $\beta = 1$, the cost of the MLMC estimator for a prescribed $RMSE$ is $o(\varepsilon^{-2+\eta})$ for every $\eta > 0$ but it is slower by a factor $\log\left(\frac{1}{\varepsilon}\right)$ than the weighted ML2R Multilevel estimator. Keep in mind than for $\varepsilon = 1\mathrm{bp} = 10^{-4}$, $\log\left(\frac{1}{\varepsilon}\right) \approx 6.9$.*
- *When $\beta \in (0, 1)$, the simulation cost is no longer close to unbiased simulation since $\varepsilon^{-\frac{1-\beta}{\alpha}}$ goes to infinity as $\varepsilon \to 0$ at polynomial rate while the weighted ML2R estimator remains close to the unbiased framework. In that setting, which is great interest in derivative pricing since it corresponds e.g. to barrier options or γ-hedging, the weighted ML2R clearly outperforms the regular MLMC estimator.*

2.4.3.0.1 ▷ *Numerical estimation of* $\mathrm{Var}(Y_0)$, V_1, θ *and* $\widetilde{c}_{\infty}$.

Such an estimation is crucial since $\mathrm{Var}(Y_0)$ and $\theta = \sqrt{\frac{V_1}{\mathrm{Var}(Y_0)}}$ appear in the simulation parameters reported in the above table. A (possibly rough) statistical pre-processing is subsequently necessary.

– *Estimation of* $\mathrm{Var}(Y_0)$. One can without damage perform a rough estimation of $\mathrm{Var}(Y_h)$ for some "medium" $h \in \mathcal{H}$ *i.e.* smaller than $\mathbf{h}$ but not too small either (say $\frac{\mathbf{h}}{N}$). Namely

$$\mathrm{Var}(Y_0) \approx \frac{1}{m}\sum_{k=1}^{m}(Y_h^k - \overline{Y}_{h,m})^2 \quad \text{where} \quad \overline{Y}_{h,m} = \frac{1}{m}\sum_{k=1}^{m} Y_h^k.$$

– *Estimation of* V_1. The parameter V_1 comes from the strong approximation rate assumption (SE_β) and it naturally comes that $V_1 \simeq \lim_{h\to 0} \sup h^{-\beta}\|Y_h - Y_0\|_2^2$. As Y_0 is not simulatable, we should proceed as follow for a good estimation of V_1. Assuming that $\|Y_h - Y_0\| \sim V_1 h^\beta$ as $h \to 0$ is holding only when h is not too small. So, deriving from Minkowski's inequality, we have

$$\|Y_h - Y_{\frac{h}{M}}\|_2 \leq \|Y_h - Y_0\|_2 + \|Y_0 - Y_{\frac{h}{M}}\|_2 \tag{2.30}$$

$R(\varepsilon)$	$\left\lceil \frac{1}{2} + \frac{\log(\widetilde{c}_\infty^{\frac{1}{\alpha}}\mathbf{h})}{\log(N)} + \sqrt{\left(\frac{1}{2} + \frac{\log(\widetilde{c}_\infty^{\frac{1}{\alpha}}\mathbf{h})}{\log(N)}\right)^2 + 2\frac{\log\left(\frac{\sqrt{1+4\alpha}}{\varepsilon}\right)}{\alpha\log(N)}} \right\rceil$
$h(\varepsilon)$	$\mathbf{h}/\left\lceil \mathbf{h}(1+2\alpha R)^{\frac{1}{2\alpha R}} \widetilde{c}_\infty^{\frac{1}{\alpha}} \varepsilon^{-\frac{1}{\alpha R}} N^{-\frac{R-1}{2}} \right\rceil$
$q(\varepsilon)$	$q_1(\varepsilon) = q^\dagger(1+\theta h^{\frac{\beta}{2}}), \quad \theta = \sqrt{\frac{V_1}{\mathrm{Var}(Y_0)}},$ $q_j(\varepsilon) = q^\dagger \theta h^{\frac{\beta}{2}} \lvert W_j^{(R)}(N)\rvert \frac{n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}}{\sqrt{n_{j-1}+n_j}} \;\; j = 2,\dots,R,\; q^\dagger \text{ s.t. } \sum_{j=1}^{R} q_j(\varepsilon) = 1$
$M(\varepsilon)$	$\left\lceil \left(1+\frac{1}{2\alpha R}\right) \frac{\mathrm{Var}(Y_0)\left(1+\theta h^{\frac{\beta}{2}} \sum_{j=1}^{R} \lvert W_j^{(R)}(N)\rvert \left(n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}\right)\sqrt{n_{j-1}+n_j}\right)^2}{\varepsilon^2 q^\dagger} \right\rceil$

Figure 2.3: Parameters of the ML2R estimator (standard framework).

So

$$V_1 \gtrsim (1+M^{-\frac{\beta}{2}})^{-2} h^{-\beta} \|Y_h - Y_{\frac{h}{M}}\|_2^2. \tag{2.31}$$

Choosing $M = M_{\max}$ large enough, we have the following formula for the estimation of V_1

$$\hat{V}_1(h) = (1+M_{\max}^{-\frac{\beta}{2}})^{-2} \|Y_h - Y\frac{h}{M_{\max}}\|_2^2. \tag{2.32}$$

– *Calibration of* $\widetilde{c}_\infty$. First it is not possible to include an estimation of $\widetilde{c}_\infty$ in a preliminary simulation, beyond the fact that $R^*(\varepsilon)$ does depend on $\widetilde{c}_\infty$ since the natural statistical estimator are far too unstable. On the other hand, to our best knowledge, under $(WE)_\infty^\alpha$, there is no significant result about the behavior of the coefficients c_r as $r \to \infty$ whatever the assumptions on the model are. What can be noted nevertheless is that these coefficients c_r have polynomial growth *i.e.* $|c_r| = O(r^a)$ as $r \to \infty$, then $\widetilde{c}_\infty = 1$ and, from a practical point of view, $|c_R|^{\frac{1}{R}} \approx 1$ if R grows.

Numerical experiments (see Giorgi [67]) show that the ML2R estimator, implemented with $|c_R|^{\frac{1}{R}} \approx 1$ is quite robust to variation of the coefficients c_r.

A possible improvement, that can however be discussed, is to proceed as follows: the magnitude of the coefficients c_r depends linearly on that of Y_0, in the sense that one switches from Y_0 to λY_0, then $(WE)_\infty^\alpha$ will hold with coefficients λc_r. Consequently an heuristics can be to approximate $\widetilde{c}_\infty$ by $|c_1|^{\frac{1}{R}}$ since this first coefficient can be estimated by considering a (biased!)

estimator of the form

$$\left[\widehat{c}_1\right]_N = \frac{h^{-\alpha}}{(1-2^{-\alpha})}\frac{1}{N}\sum_{k=1}^{N} Y_h^k - Y_{\frac{h}{2}}^k$$

for some medium $h \in H$ (*e.g.* $\frac{\mathbf{h}}{M}$, where $(Y_h^k, Y_{\frac{h}{2}}^k)_{k\geq 1}$ are i.i.d. copies of $(Y_h, Y_{\frac{h}{2}})$ and $M \ll N$.

2.5 Antithetic Schemes

In this section, we introduce briefly the antithetic (meta-)schemes devised from the Milstein scheme introduced in [66] by Giles and Szpruch. We give a short description to understand how the multilevel technique can be applied to that kind of schemes. Finally, we give the optimal parameters for both framework and make an application to a smooth payoff function and for the computation of its derivatives.

Consider the following multidimensional stochastic differential equation of the form

$$\begin{cases} dX_t = b(X_t)dt + \sigma(X_t)dW_t, \quad t \in [0,T] \\ X_0 = x, \quad x \in \mathbb{R}. \end{cases} \tag{2.33}$$

Where $b : \mathbb{R}^d \mapsto \mathbb{R}^d$ is the drift coefficient, $\sigma = [\sigma_{i,j}] : \mathbb{R}^d \mapsto \mathcal{M}(d,q,\mathbb{R})$ is a (differentiable) diffusion coefficient and $W = (W^1, \ldots, W^q)$ is a $q-$dimensional standard Brownian motion. The components are uncorrelated in what follows.

We have the following definition for the Milstein scheme with time step $\frac{T}{n}$, $T \in \mathbb{R}^+$, $k = 0, \ldots, n-1$,

$$\begin{cases} \tilde{X}^n_{t^n_{k+1}} = \tilde{X}^n_{t^n_k} + b(\tilde{X}^n_{t^n_k})\dfrac{T}{n} + \sigma(\tilde{X}^n_{t^n_k})\Delta W_{t^n_{k+1}} + \displaystyle\sum_{1\leq i,j\leq q} \partial\sigma_{.i}\sigma_{.j}(\tilde{X}^n_{t^n_k}) \times \int_{t^n_k}^{t^n_{k+1}} (W^i_s - W^i_{t^n_k})dW^j_s, \\ \tilde{X}_0 = X_0, \end{cases} \tag{2.34}$$

where $t^n_k = k\frac{T}{n}$, $k = 0, \ldots, n$ and $\sigma_{.k}(x)$ denotes the k^{th} column of the matrix σ and

$$\forall x = (x_1, \ldots, x_d) \in \mathbb{R}^d, \quad \bar{\sigma} = \partial\sigma_{.i}\sigma_{.j}(x) := \frac{1}{2}\sum_{l=1}^{d} \frac{\partial\sigma_{.i}}{\partial x_l}\sigma_{lj}(x)(x) \in \mathbb{R}^d. \tag{2.35}$$

One particularity on that scheme is the arise of the Lévy areas in 2.34 when $i \neq j$, $\int_{t^n_k}^{t^n_{k+1}} W^i_s dW^j_s$. The ability of simulating such a scheme relies entirely on the exact simulations of

$$\left(W^1_{t^n_{k+1}} - W^1_{t^n_k}, \ldots, W^q_{t^n_{k+1}} - W^q_{t^n_k}, \int_{t^n_k}^{t^n_{k+1}} (W^i_s - W^i_{t^n_k})dW^j_s i, \quad j = 1, \ldots, q, \quad i \neq j \right), \quad k = 0, \ldots, n-1.$$

Unfortunately, there is no known efficient method to simulate when $q \geq 2$ the Milstein scheme.

2.5.1 Truncated Milstein Scheme

The get rid of the Lévy areas, the idea is to force symmetrization by taking advantage of the identity

$$\int_{t^n_k}^{t^n_{k+1}} (W^i_s - W^i_{t^n_k})dW^j_s + \int_{t^n_k}^{t^n_{k+1}} (W^j_s - W^j_{t^n_k})dW^i_s = (W^i_{t^n_{k+1}} - W^i_{t^n_k})(W^j_{t^n_{k+1}} - W^j_{t^n_k}), \tag{2.36}$$

so that we can replace both Lévy's areas by their half-sum

$$\frac{1}{2}(W^i_{t^n_{k+1}} - W^i_{t^n_k})(W^j_{t^n_{k+1}} - W^j_{t^n_k}). \tag{2.37}$$

This leads to the truncated Milstein scheme with time step where the Lévy's areas are replaced by their half-sum

$$\begin{cases} \bar{X}^n_{t^n_{k+1}} = \bar{X}^n_{t^n_k} + b(\bar{X}^n_{t^n_k})\frac{T}{n} + \sigma(\bar{X}^n_{t^n_k})\Delta W_{t^n_{k+1}} + \frac{1}{2}\sum_{1\le i,j\le q} \partial\sigma_i\sigma_j(\bar{X}^n_{t^n_k})\left(\Delta W^i_{t^n_{k+1}}\Delta W^j_{t^n_{k+1}} - \mathbb{1}_{i=j}\frac{T}{n}\right), \\ \bar{X}_0 = X_0, \end{cases} \tag{2.38}$$

with $\Delta W_{t^n_{k+1}} = W_{t^n_{k+1}} - W_{t^n_k}$, $k = 0, \dots, n-1$. This discrete scheme can be successfully implemented but the analysis of its strong convergence show a behavior similar to the Euler Mayurama.Under smooth assumptions on the drift and diffusion coefficients, the weak error behaves like the Euler scheme and it satisfies a first order expansion in h. Under smoothness assumptions, it is expected that a higher order weak error expansion holds true

The idea introduced by Giles and Szpruch in [66] is to combine several truncated Milstein schemes at different scales in order to make the fine level behave as if the scheme was satisfying the strong convergence $(SE)_\beta$ for a $\beta > 1$. In order to achieve that construction, the fine scheme $i \ge 2$ is split into two schemes with the same step but based on swapped Brownian increments. Let's take the Markov representation of the truncated Milstein scheme,

$$\begin{cases} \bar{X}^n_{t^n_{k+1}} = \tilde{\mathcal{M}}(\bar{X}^n_{t^n_k}, \frac{T}{n}, \Delta W_{t^n_{k+1}}), \quad k = 0, \dots, n-1, \\ \bar{X}_0 = X_0. \end{cases} \tag{2.39}$$

We consider a first scheme $(\bar{X}^{2n}_{t^{2n,[1]}_{k+1}})_{k=0,\dots,2n}$ with time step $\frac{T}{2n}$:

$$\begin{cases} \bar{X}^{2n}_{t^{2n,[1]}_{2k+1}} = \tilde{\mathcal{M}}(\bar{X}^{2n}_{t^{2n,[1]}_{2k}}, \frac{T}{2n}, \Delta W_{t^{2n}_{2k+1}}), \\ \bar{X}^{2n}_{t^{2n,[1]}_{2(k+1)}} = \tilde{\mathcal{M}}(\bar{X}^{2n}_{t^{2n,[1]}_{2k+1}}, \frac{T}{2n}, \Delta W_{t^{2n}_{2(k+1)}}), \quad k = 0, \dots, n-1, \end{cases} \tag{2.40}$$

and a second scheme $(\bar{X}^{2n}_{t^{2n,[2]}_{k+1}})_{k=0,\dots,2n}$ with pairwise swapped Brownian increments $\Delta W_{t^{2n}_{2k+1}}$ and $\Delta W_{t^{2n}_{2(k+1)}}$,

$$\begin{cases} \bar{X}^{2n}_{t^{2n,[2]}_{2k+1}} = \tilde{\mathcal{M}}(\bar{X}^{2n}_{t^{2n,[2]}_{2k}}, \frac{T}{2n}, \Delta W_{t^{2n}_{2(k+1)}}), \\ \bar{X}^{2n}_{t^{2n,[2]}_{2(k+1)}} = \tilde{\mathcal{M}}(\bar{X}^{2n}_{t^{2n,[2]}_{2k+1}}, \frac{T}{2n}, \Delta W_{t^{2n}_{2k+1}}),, \quad k = 0, \dots, n-1, \end{cases} \tag{2.41}$$

Note that $\Delta W_{t^{2n}_{2(k+1)}} + \Delta W_{t^{2n}_{2k+1}} = \Delta W_{t^n_k}$ so that both schemes are consistent with the coarse scheme with time step $\frac{T}{n}$.

In what follows, the root $M = 2$ and we will focus on the marginal case $I_0 = \mathbb{E}[f(X_T)]$, assuming that f is such that $(SE)_\beta$ is satisfied for some $\beta \in (1, 2]$. This leads to a new truncated Milstein pseudo-scheme with time step h for the refined level i defined as follows:

- Coarse level ($i = 1$), Simply set $Y^{(1)}_h = f(X^{n,(1)}_T)$ where X is driven by a $q-$dimensional standard Brownian motion $W^{(1)}$.

- Fine level ($i \geq 2$), The regular difference $Y^{(i)}_{\frac{h}{n_i}} - Y^{(i)}_{\frac{h}{n_{i-1}}}$ (with $n_i = 2^{i-1}$) is replaced by

$$\frac{1}{2}\left(Y^{[1],(i)}_{\frac{h}{n_i}} + Y^{[2],(i)}_{\frac{h}{n_i}}\right) - Y^{(i)}_{\frac{h}{n_{i-1}}} \tag{2.42}$$

with $Y^{(i)}_{\frac{h}{n_{i-1}}} = f(\bar{X}_T^{nM^{i-2},(i)})$ and $Y^{[r],(i)}_{\frac{h}{n_i}} = f(\bar{X}_T^{nM^{i-1},[r],(i)})$, $r = 1, 2$. All the three schemes at level i are driven by the same $q-$dimensional standard Brownian Motion $W^{(i)}$, $i = 1, \ldots, R$,

Giles and Szpruch established the following theorem

Theorem 3. *Assume $b \in \mathcal{C}^2(\mathbb{R}^d, \mathbb{R}^d)$, $\sigma \in \mathcal{C}^2(\mathbb{R}^d, \mathcal{M}(d, q, \mathbb{R})$ with bounded existing partial derivatives. Idem for $\bar{\sigma}$ (up to order 2) are all bounded.*

(a) For smooth payoffs. Let $f \in \mathcal{C}^2(\mathbb{R}^d, \mathbb{R}^d)$. Then,

$$\left\| f(\bar{X}_T^n) - \frac{1}{2}\left(f(\bar{X}_T^{2n,[1]}) + f(\bar{X}_T^{2n,[2]})\right)\right\|_2 \leq C_{b,\sigma,T}\frac{T}{n}(1 + \|X_0\|_2). \tag{2.43}$$

i.e. (SE_β) is satisfied with $\beta = 2$.

(b) For almost smooth payoffs, Assume that f is Lipschitz continuous on $\mathbb{R}^d$ and that's its first two order partial derivatives exist outside a Lebesgue negligible set $\mathcal{M}_0$ of $\mathbb{R}^d$ and are uniformly bounded. Assume that the diffusion $(X_t)_{t\in[0,T]}$ satisfies that

$$\overline{\lim_{\varepsilon\to 0}}\varepsilon^{-1}\mathbb{P}(\inf_{z\in\mathcal{N}_0}|X_T - z| \leq \varepsilon) < +\infty \tag{2.44}$$

then, $(SE)_\beta$ is satisfied for every $\beta \in (0, \frac{3}{2})$, since

$$\left\| f(\bar{X}_T^n) - \frac{1}{2}\left(f(\bar{X}_T^{2n,[1]}) + f(\bar{X}_T^{2n,[2]})\right)\right\|_2 \leq C_{b,\sigma,T}\left(\frac{T}{n}\right)^{\frac{3}{4}-\eta}(1 + \|X_0\|_2). \tag{2.45}$$

where every $\eta \in (0, \frac{3}{4})$.

For the proof Owing to the theorem 3, it is clear that , under appropriate assumptions, we have

$$\left\|\frac{1}{2}\left(Y^{[1],(i)}_{\frac{h}{n_i}} + Y^{[2],(i)}_{\frac{h}{n_i}}\right) - Y^{(i)}_{\frac{h}{n_{i-1}}}\right\|_2^2 \leq V_1\left(\frac{h}{n_i}\right)^\beta. \tag{2.46}$$

with $\beta = (\frac{3}{2})^-$ or 2.

The time discretization scheme with two successive half time steps to obtain strong convergence with $\beta = 2$ and based on the Brownian increments swaps, is defined as follows:

$$\begin{cases} \bar{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}} = \bar{X}^{2n}_{t^{2n}_k} + b(\bar{X}^{2n}_{t^{2n}_k})\frac{T}{2n} + \sigma(\bar{X}^{2n}_{t^{2n}_k})\Delta W_{t^n_{k+\frac{1}{2}}} + \sum_{1\leq i\leq j\leq q}\bar{\sigma}_{ij}(\bar{X}^{2n}_{t^{2n}_k})\left(\Delta W^i_{t^n_{k+\frac{1}{2}}}\Delta W^j_{t^{2n}_{k+\frac{1}{2}}} - \mathbb{1}_{i=j}\frac{T}{2n}\right), \\ \tilde{X}^{2n}_{t^{2n}_{k+1}} = \tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}} + b(\tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}})\frac{T}{2n} + \sigma(\tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}})\Delta W_{t^{2n}_{+1}} + \sum_{1\leq i\leq j\leq q}\bar{\sigma}_{ij}(\tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}})\left(\Delta W^i_{t^{2n}_{k+1}}\Delta W^j_{t^{2n}_{k+1}} - \mathbb{1}_{i=j}\frac{T}{2n}\right), \end{cases} \tag{2.47}$$

with $\Delta W^i_{t^{2n}_{k+\frac{1}{2}}} = W^i_{t^{2n}_{k+\frac{1}{2}}} - W^i_{t^{2n}_k}$ and $\Delta W^i_{t^{2n}_{k+1}} = W^i_{t^{2n}_{k+1}} - W^i_{t^{2n}_{k+\frac{1}{2}}}$, and

$$\bar{\sigma}_{ij}(x) = \frac{1}{2}(\partial\sigma_{.i}\sigma_{.j} + \partial\sigma.j\sigma.i), \quad 1 \leq i < j \leq q, \quad \bar{\sigma}_{ii}(x) = \frac{1}{2}\sigma_{.i}\sigma_{.i}(x).$$

with the tensor $\partial\sigma_{.i}\sigma_{.j}$ defined by

$$\forall x = (x_1, \ldots, x_d) \in \mathbb{R}^d, \quad \partial\sigma_{.i}\sigma_{.j}(x) := \sum_{l=1}^{d} \frac{\partial\sigma_{.i}}{\partial x_l}(x)\sigma_{lj}(x) \in \mathbb{R}^d.$$

The antithetic scheme results from the same formulas using Brownian increments swaps as given as follows

$$\begin{cases} \tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}} = \bar{X}^{2n}_{t^{2n}_k} + b(\bar{X}^{2n}_{t^{2n}_k})\dfrac{T}{2n} + \sigma(\bar{X}^{2n}_{t^{2n}_k})\Delta W_{t^n_{k+\frac{1}{2}}} + \displaystyle\sum_{1\le i\le j\le q} \bar{\sigma}_{ij}(\bar{X}^{2n}_{t^{2n}_k})\left(\Delta W^i_{t^{2n}_{k+1}}\Delta W^j_{t^{2n}_{k+1}} - \mathbb{1}_{i=j}\dfrac{T}{2n}\right), \\ \tilde{X}^{2n}_{t^{2n}_{k+1}} = \tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}} + b(\tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}})\dfrac{T}{2n} + \sigma(\tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}})\Delta W_{t^{2n}_{+1}} + \displaystyle\sum_{1\le i\le j\le q} \bar{\sigma}_{ij}(\tilde{X}^{2n}_{t^{2n}_{k+\frac{1}{2}}})\left(\Delta W^i_{t^n_{k+\frac{1}{2}}}\Delta W^j_{t^{2n}_{k+\frac{1}{2}}} - \mathbb{1}_{i=j}\dfrac{T}{2n}\right). \end{cases} \tag{2.48}$$

2.5.2 Optimal Settings

Following Al Gerbi et al. [57], the optimal settings for the regular multilevel Monte Carlo are given in 2.4 and in 2.5 for the weighted method.

$L(\varepsilon)$	$\left\lceil 1 + \log_2\left(\lvert c_1\rvert^{\frac{1}{\alpha}}\mathbf{h}\right) + \dfrac{1}{\alpha}\log_2\left(\dfrac{\sqrt{(1+2\alpha)}}{\varepsilon}\right)\right\rceil$
$h(\varepsilon)$	$\mathbf{h}/\left\lceil \mathbf{h}(1+2\alpha)^{\frac{1}{2\alpha}}\left(\dfrac{\varepsilon}{\lvert c_1\rvert}\right)^{-\frac{1}{\alpha}} 2^{-(L-1)}\right\rceil^{-1}$
$q(\varepsilon)$	$q_1(\varepsilon) = q^\dagger \mathrm{Var}(Y_0), \quad \theta = \sqrt{\dfrac{V_1}{\mathrm{Var}(Y_0)}},$ $q_j(\varepsilon) = q^\dagger \theta h^{\frac{\beta}{2}} \dfrac{n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}}{\sqrt{n_{j-1}+n_j}},\ j = 2, \ldots, L,\ q^\dagger \text{ s.t. } \displaystyle\sum_{j=1}^{L} q_j(\varepsilon) = 1$
$N(\varepsilon)$	$\left\lceil \left(1+\dfrac{1}{2\alpha}\right)\dfrac{\mathrm{var}(Y_0)\left(1+\theta h^{\frac{\beta}{2}}\displaystyle\sum_{j=1}^{L}\left(n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}\right)\sqrt{n_{j-1}+n_j}\right)^2}{\varepsilon^2 q^\dagger}\right\rceil$

Figure 2.4: Parameters of the MLMC estimator (antithetic scheme).

$R(\varepsilon)$	$\left\lceil \frac{1}{2} + \log_2(\widetilde{c}_\infty^{\frac{1}{\alpha}}\mathbf{h}) + \sqrt{\left(\frac{1}{2} + \log_2(\widetilde{c}_\infty^{\frac{1}{\alpha}}\mathbf{h})\right)^2 + \frac{2}{\alpha}\log_2\left(\frac{\sqrt{1+4\alpha}}{\varepsilon}\right)} \right\rceil$
$h(\varepsilon)$	$\mathbf{h}/\left\lceil \mathbf{h}(1+2\alpha R)^{\frac{1}{2\alpha R}} \widetilde{c}_\infty^{\frac{1}{\alpha}} \varepsilon^{-\frac{1}{\alpha R}} 2^{-\frac{R-1}{2}} \right\rceil$
$q(\varepsilon)$	$q_1(\varepsilon) = q^\dagger \mathrm{Var}(Y_0), \quad \theta = \sqrt{\frac{V_1}{\mathrm{Var}(Y_0)}},$ $q_j(\varepsilon) = q^\dagger \theta h^{\frac{\beta}{2}} \lvert W_j^{(R)}(2)\rvert \frac{n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}}{\sqrt{n_{j-1}+n_j}} \;\; j = 2, \ldots, R, \; q^\dagger \text{ s.t. } \sum_{j=1}^{R} q_j(\varepsilon) = 1$
$N(\varepsilon)$	$\left\lceil \left(1 + \frac{1}{2\alpha R}\right) \frac{\mathrm{Var}(Y_0)\left(1 + \theta h^{\frac{\beta}{2}} \sum_{j=1}^{R} \lvert W_j^{(R)}(2)\rvert \left(n_{j-1}^{-\frac{\beta}{2}} + n_j^{-\frac{\beta}{2}}\right)\sqrt{n_{j-1}+n_j}\right)^2}{\varepsilon^2 q^\dagger} \right\rceil$

Figure 2.5: Parameters of the ML2R estimator (antithetic scheme).

2.6 Generic Procedure for Option Pricing and Sensitivities

In this section, we use the following notations:

- the subscript $\cdot_P$ designed a measure/parameter of the price (α_P, M_P, $\mathrm{Var}(Y_P)$, ...)
- the subscript $\cdot_\Delta$ designed of measure/parameter of the Delta (α_P, M_Δ, $\mathrm{Var}(Y_\Delta)$, ...)
- the subscript $\cdot_\Gamma$ designed a measure/parameter of the Gamma (α_P, M_Γ, $\mathrm{Var}(Y_\Gamma)$, ...)

Here, we give the description of the algorithm to compute the premium, the Delta and the Gamma of a financial derivatives at the same time using the optimal multilevel framework. We build an estimator for these three values, the structure parameters for the sensitivities are computed based on the root M_P of the price (i.e. they are build on the same discretization grid). We take the same root M for the premium, the Delta and the Gamma to compute their structure parameters. For the Delta and the Gamma we choose the settings which have the closest numerical cost to the price one.

As the sensitivities functions are not equal to the price function, we have different values for the associated β and α. Therefore their structure parameters are different. It seems that, when $\beta < 1$ one needs more levels but less simulations. During our experiments, we observed that the Delta and the Gamma required more levels than the price but less simulations. So the estimators of the Delta and the Gamma are computed with only a part of the simulations required for the price but the simulations have more time steps.

STEP 1: *Select parameters.* The only parameters entered by the user are the prescribed RMSE ε_P for the approximation of the price of a contingent claim, $M_{\max}$ and a minimum prescribed $\varepsilon_{\min}$ (see further). The structure parameters of the sensitivities are computed according to the numerical cost of the price (we take the structure parameters which have the closest numerical cost to the price one). All others parameters are computed or approximated via the frameworks described in tables 2.2 and 2.3, the only difference concern the parameter β which are specified empirically (we could proceed following the same procedure for the parameters α but the study is restricted to the analysis of the parameter β_P with respect to the payoff function).

STEP 2: *Initialization.* pre-processing evaluation of the initial specifications β_P, β_Δ, β_Γ $\mathrm{Var}(Y_0)$, V_1 and θ required to compute the structural estimator parameters (table 2.2 and 2.3). As it does not depend on ε_P, one can compute those quantities offline.

– Compute estimates for β_P, β_Δ and β_Γ. Assuming that the variance are decreasing with h, for a fixed $L \in \mathbb{N}^*$, $L \geq 1$ and $M = 2$ we would expect that:

$$\mathrm{Var}(Y_i^P) = 2^{-\beta_P}\mathrm{Var}(Y_{i-1}^P), \quad i = 1, \ldots, L, \tag{2.49}$$

hence, by taking the $\log_2$

$$\beta_P = \log_2\left(\mathrm{Var}(Y_{i-1}^P)\right) - \log_2\left(\mathrm{Var}(Y_i^P)\right). \tag{2.50}$$

We proceed the same way for the parameters β_Δ and β_Γ.

– *(Optional)* If they are not provided, estimate the parameters α_P, α_Δ and α_Γ:

$$\alpha_P = \log_2(\mathbb{E}(Y_{\frac{T}{2^{n-1}}}^{(n-1),P})) - \log_2(\mathbb{E}(Y_{\frac{T}{2^n}}^{(n),P})). \tag{2.51}$$

(resp. for Δ and Γ).

– Compute the coefficient $c_1 \sim \hat{c}_1$:

$$\left[\widehat{c}_1\right]_I = \frac{h^{-\alpha}}{(1-2^{-\alpha})}\frac{1}{I}\sum_{k=1}^{I} Y_h^k - Y_{\frac{h}{2}}^k, \quad I \in \mathbb{N}^* \tag{2.52}$$

– Estimate the other initials specifications of the framework estimators MLMC and ML2R.

– Approximate roughly $\mathrm{Var}(Y_0^P)$, $\mathrm{Var}(Y_0^\Delta)$ and $\mathrm{Var}(Y_0^\Gamma)$ (i.e. the true estimators) with a sample of size 100,000 – Compute the rate of convergence V_1^P of Y_h^P towards Y_0^P by:

$$V_1^P = (1 + M_{\max}^{-\frac{\beta_P}{2}})^{-2} h^{-\beta_P} \|Y_h^P - Y^P_{\frac{h}{M_{\max}}}\|_2^2 \tag{2.53}$$

Proceed likewise for V_1^Δ and V_1^Γ.

– Deduce the parameter θ^P (the quadratic rate of convergence of Y_h^P towards Y_0^P and the variance of Y_0^P) with from following equation:

$$\theta^P = \sqrt{\frac{V_1^P}{\mathrm{Var}(Y_0^P)}}. \tag{2.54}$$

Proceed likewise for θ^Δ and θ^Γ with the same equation.

Step 3: *Compute the asymptotic structural parameters for the price.* For ML2R estimator (resp. MLMC) compute the structural parameters $R(\varepsilon_P)$, $h(\varepsilon_P)$, $q(\varepsilon_P)$ and $N(\varepsilon_P)$ (resp. $L(\varepsilon_P)$, $h(\varepsilon_P)$, $q(\varepsilon_P)$ and $N(\varepsilon_P)$) for the price using table 2.2 et 2.3 for a set of values for $M^P = 2, \ldots, M_{\max}$.

– Retrieve the optimal parameter $M^{P,*}$ which corresponds to the minimum cost for a set of values for M^P

$$M^{P,*} = \underset{M^P \in \{2,\ldots,M^P_{\max}\}}{\operatorname{argmin}} C_{\varepsilon^P}(M^P) \tag{2.55}$$

where $C_{\varepsilon^P}(M^P)$ is the numerical complexity cost function.

– Select the asymptotic structural parameters corresponding to the value of $M^{P,*}$.

Step 4: *Compute the optimal structural parameters for the Delta and the Gamma.* To reduce the difficulty of building this framework, we choose the root M^Δ of the Delta and M^Γ of the Gamma equivalent to the root M^P of the price. In others words, we construct all estimators on the same discretization grid.

$$M^{\Delta,*} = M^{P,*}, \quad M^{\Gamma,*} = M^{P,*}. \tag{2.56}$$

– As done for the price, we compute the asymptotic structural parameters for the Delta and the Gamma (either for ML2R or MLMC estimator) for a set of values for ε_Δ and ε_Γ.

– Handling the same discretization grid means that the estimation of the price, the Delta and the Gamma use the same simulation path (but one may not use the same number of levels). Therefore, it has the same cost and in order to avoid too much calculation for one of the estimators, as the root M^Δ and M^Γ are fixed, we have to find the prescribed RMSE ε^*_Δ and ε^*_Γ for both estimators with the closest numerical cost than for the price. Hence,

$$\begin{cases} \varepsilon^*_\Delta = \underset{\varepsilon_\Delta \in \{\varepsilon_1,\ldots,\varepsilon_{\min}\}}{\operatorname{argmin}} \left\{ \mid C_{\varepsilon_P}(M^{P,*}) - C_{\varepsilon_\Delta}(M^{\Delta,*}) \mid \right\}, \\ \varepsilon^*_\Gamma = \underset{\varepsilon_\Gamma \in \{\varepsilon_1,\ldots,\varepsilon_{\min}\}}{\operatorname{argmin}} \left\{ \mid C_{\varepsilon_P}(M^{P,*}) - C_{\varepsilon_\Gamma}(M^{\Gamma,*}) \right\} \end{cases} \tag{2.57}$$

with $\varepsilon_1 \geq \ldots \geq \varepsilon_{\min}$.

– Select the optimal structural parameters corresponding to ε^*_Δ and ε^*_Γ.

Step 5: *Construct the M2LR or the MLMC estimators on each level.* Compute the number of path simulations per level:

$$\tilde{N}_j = \max\left(\left\lceil N^P q_j^P \right\rceil \mathbb{1}_{\{j \leq R^P\}}, \left\lceil N^\Delta q_j^\Delta \right\rceil \mathbb{1}_{\{j \leq R^\Delta\}}, \left\lceil N^\Gamma q_j^\Gamma \right\rceil \mathbb{1}_{\{j \leq R^\Gamma\}} \right), \quad j = 1, \ldots, R_{\max} \tag{2.58}$$

where

$$R_{\max} = \max\left(R^P, R^\Delta, R^\Gamma\right). \tag{2.59}$$

– Compute the ML2R estimator $\hat{\mathbf{I}}^{ML2R} = (\hat{I}_P^{ML2R}, \hat{I}_\Delta^{ML2R}, \hat{I}_\Gamma^{ML2R})$, defined as

$$\hat{\mathbf{I}}^{ML2R} := \mathbf{D}^{(1)} \sum_{i=1}^{\tilde{N}_1} \mathbf{Q}^{(1),i} \mathbf{Y}_h^{(1),i} + \sum_{i=2}^{R_{\max}} \mathbf{D}^{(i)} \mathbf{H}^{(i)} \sum_{j=1}^{\tilde{N}_i} \mathbf{Q}^{(i),j} \left(\mathbf{Y}^{(i),j}_{\frac{h}{n_i}} - \mathbf{Y}^{(i),j}_{\frac{h}{n_{i-1}}} \right) \tag{2.60}$$

where the matrices $\mathbf{D}$, $\mathbf{Q}$ and $\mathbf{H}$ are diagonal and are defined as:

$$\begin{aligned}
\mathbf{D}^{(j)} &= \text{diag}\left(\frac{1}{N_j^P}\mathbb{1}_{\{j\leq R^P\}}, \frac{1}{N_j^\Delta}\mathbb{1}_{\{j\leq R^\Delta\}}, \frac{1}{N_j^\Gamma}\mathbb{1}_{\{j\leq R^\Gamma\}}\right), \quad j=1,\dots,R_{\max} \\
\mathbf{Q}^{(i),j} &= \text{diag}\left(\mathbb{1}_{\{i\leq N_j^P\}}, \mathbb{1}_{\{i\leq N_j^\Delta\}}, \mathbb{1}_{\{i\leq N_j^\Gamma\}}\right), \quad i=1,\dots,\tilde{N}_j, \quad j=1,\dots,R_{\max} \\
\mathbf{H}^{(j)} &= \text{diag}\left(\hat{W}_j^P, \hat{W}_j^\Delta, \hat{W}_j^\Gamma\right), \quad j=1,\dots,R_{\max}
\end{aligned}$$

with

$$(N_j^P, N_j^\Delta, N_j^\Gamma) = \left(\lceil N^P q_j^P\rceil, \lceil N^\Delta q_j^\Delta\rceil, \lceil N^\Gamma q_j^\Gamma\rceil\right), \quad j=1,\dots,R_{\max}, \tag{2.61}$$

and the vector $\mathbf{Y}$ defined as:

$$\mathbf{Y}_h = \left(Y_h^P, Y_h^\Delta, Y_h^\Gamma\right). \tag{2.62}$$

More explicitly,

$$\hat{I}_P^{ML2R} = \frac{1}{N_1^P}\mathbb{1}_{\{1\leq R^P\}}\sum_{i=1}^{\tilde{N}_1}\mathbb{1}_{\{i\leq N_1^P\}}Y_h^{(1),i,P} + \sum_{i=2}^{R_{\max}}\frac{1}{N_i^P}\mathbb{1}_{\{i\leq R^P\}}\hat{W}_i^P\sum_{j=1}^{\tilde{N}_i}\mathbb{1}_{\{j\leq N_i^P\}}\left(Y_{\frac{h}{n_i}}^{(i),j,P} - Y_{\frac{h}{n_{i-1}}}^{(i),j,P}\right)$$

(resp. for Δ and Γ).

– If we denote by $\hat{\mathbf{\Sigma}}^{ML2R}$ the variance of the ML2R estimator $\hat{\mathbf{I}}^{ML2R}$, we just have to compute the diagonal of the variance-covariance matrix $\hat{\mathbf{\Sigma}}^{ML2R}$. Hence,

$$\begin{aligned}
\left(\text{Var}(\hat{I}_P^{ML2R}), \text{Var}(\hat{I}_\Delta^{ML2R}), \text{Var}(\hat{I}_\Gamma^{ML2R})\right) &= \text{diag}\left(\hat{\mathbf{\Sigma}}^{ML2R}\right)v \\
&= \left(\hat{\Sigma}_{11}^{ML2R}, \hat{\Sigma}_{22}^{ML2R}, \hat{\Sigma}_{33}^{ML2R}\right)
\end{aligned} \tag{2.63}$$

where $v=(1,\dots,1)$. Let us denote by $\hat{\mathbf{V}}^{ML2R} = \left(\text{Var}(\hat{I}_P^{ML2R}), \text{Var}(\hat{I}_\Delta^{ML2R}), \text{Var}(\hat{I}_\Gamma^{ML2R})\right)$, we can rewrite the variance of each estimators as:

$$\hat{\mathbf{V}}^{ML2R} := \mathbf{D}^{(1)}\mathbf{V}_h^{(1)} + \sum_{i=2}^{R_{\max}}\mathbf{D}^{(i)}(\mathbf{H}^{(i)})^2\mathbf{V}_h^{(i)} \tag{2.64}$$

where

$$\mathbf{V}_h^{(i)} = \begin{cases} \left(\text{Var}(Y_h^{(1),P}), \text{Var}(Y_h^{(1),\Delta}), \text{Var}(Y_h^{(1),\Gamma})\right) & \text{if } i=1, \\ \left(\text{Var}(Y_{\frac{h}{n_i}}^{(i),P} - Y_{\frac{h}{n_{i-1}}}^{(i),P}), \text{Var}(Y_{\frac{h}{n_i}}^{(i),\Delta} - Y_{\frac{h}{n_{i-1}}}^{(i),\Delta}), \text{Var}(Y_{\frac{h}{n_i}}^{(i),\Gamma} - Y_{\frac{h}{n_{i-1}}}^{(i),\Gamma})\right) & \text{otherwise} \end{cases}$$

Therefore, each component of the vector $\hat{\mathbf{V}}^{ML2R}$ are given by:

$$\text{Var}(\hat{I}_P^{ML2R}) = \frac{1}{N_1^P}\mathbb{1}_{\{1\leq R^P\}}\text{Var}(Y_h^{(1),P}) + \sum_{i=2}^{R_{\max}}\frac{1}{N_i^P}\mathbb{1}_{\{i\leq R^P\}}(\hat{W}_i^P)^2\text{Var}(Y_{\frac{h}{n_i}}^{(i),P} - Y_{\frac{h}{n_{i-1}}}^{(i),P})$$

(resp. for Δ and Γ).

– Compute the bias over J independent replications of the ML2R estimator:

$$\text{Bias}(\hat{\mathbf{I}}^{ML2R}) := \frac{1}{J}\sum_{j=1}^{J}(\hat{\mathbf{I}}^{ML2R})^{(j)} - \mathbf{I}_0, \tag{2.65}$$

where

$$\text{Bias}(\hat{\mathbf{I}}^{ML2R}) = \left(\text{Bias}(\hat{I}_P^{ML2R}), \text{Bias}(\hat{I}_\Delta^{ML2R}), \text{Bias}(\hat{I}_\Gamma^{ML2R})\right), \tag{2.66}$$

and $\mathbf{I}_0 = (P_0, \Delta_0, \Gamma_0)$ is a vector of the true values (or the reference values is there is no "true") values of the price, the Delta and the Gamma.

– Compute the empirical L^2-error or empirical root mean squared error (RMSE) of each estimators

$$\text{RMSE}(\hat{\mathbf{I}}^{ML2R}) \quad := \quad \mathbf{F}\left(\frac{1}{J}\sum_{j=1}^{J}\left((\hat{\mathbf{I}}^{ML2R})^{(j)} - \mathbf{I}_0\right)\circ\left((\hat{\mathbf{I}}^{ML2R})^{(j)} - \mathbf{I}_0\right)\right)$$

where $\mathbf{F}(\mathbf{X}) = \left(\sqrt{x_i}, \ldots, \sqrt{x_n}\right), \quad \mathbf{X} \in \mathbb{R}^n, \quad n \in \mathbb{N}$ and $\circ$ is the Hadamard product.

– For the MLMC estimator, just replace $R_{\max}$ by $L_{\max}$ (with the other corresponding structural parameters) and replace the matrix $\mathbf{H}$ by the identity matrix in the equations 2.60 and 2.64.

This procedure is summarized in the algorithm 2 and a detailed part of the simulation process for the Heston model in appendix 3.

2.7 Numerical Results

In this section we consider three different test cases, the computation of the Price, the Delta and the Gamma independently with the optimized regular and weighted Monte Carlo framework with the smoothing technique for an exotic product in the Heston model. The second test case is the computation of both the price, the Delta and the Gamma at the same time of an European Call option in the Black Scholes model with the framework described in section 2.6. The third case is dedicated to the analysis of numerical experiments of the antithetic scheme with regular and weighted Monte Carlo.

We compare the two estimators i.e. MLMC and M2LR through several indicators namely the time of computation, numerical cost, empirical bias error, empirical L_2-error and the empirical variance. The empirical bias error $\tilde{\mu}_L$ of the estimator $\bar{Y}_{h,\underline{n}}^{N,R}$ is computed using $L = 256$ replications

$$\tilde{\mu}_L = \frac{1}{L}\sum_{l=1}^{L}(\bar{Y}_{h,\underline{n}}^{N,R})^{(l)} - I_0, \tag{2.67}$$

where $I_0 = \mathbb{E}[Y_0]$ is true value.

The empirical L_2-error (RMSE) $\tilde{\varepsilon}_L$ of the estimator $\bar{Y}_{h,\underline{n}}^{N,R}$ is obtained following

$$\tilde{\varepsilon}_L = \sqrt{\frac{1}{L}\sum_{l=1}^{L}\left((\bar{Y}_{h,\underline{n}}^{N,R})^{(l)} - I_0\right)^2}. \tag{2.68}$$

The empirical variance of the previous estimator is given by the following formula

$$\tilde{\nu}_L = \frac{1}{L-1}\left(\sum_{i=1}^{L}\left((\bar{Y}^{N,R}_{h,\underline{n}})^{(l)}\right)^2 - \frac{1}{L}\left(\sum_{l=1}^{L}(\bar{Y}^{N,R}_{h,\underline{n}})^{(l)}\right)^2\right). \tag{2.69}$$

The computation were performed on a supercomputer with 16 Intel Xeon CPUs E5-4650 v2 @ 2.40GHz (octo-core processors so 128 processors in total). The code of one estimator runs on a single processor, we used the MPI protocol/C++ for the parallelism and we tuned the code with vectorization to get great performances in term of time computing.

2.7.1 Computation of Sensitivities of Up and Out Call Option in the Heston model

Consider the Heston model with initial conditions $X_0 = x \in \mathbb{R}$ and $\mathcal{V}_0 = \nu \in \mathbb{R}^+$.

$$\begin{cases} dX_t = rX_t dt + \sqrt{\mathcal{V}_t} X_t dW_t^1, \\ d\mathcal{V}_t = \kappa(\eta - \mathcal{V}_t)dt + \xi\sqrt{\mathcal{V}_t} dW_t^2, \end{cases} \tag{2.70}$$

where ξ denotes the volatility of volatility, η the long-run mean of $\mathcal{V}_t$ and κ the mean reversion velocity. $W = (W^1, W^2)$ is a $2-$dimensional Brownian motion, the components are $\rho-$correlated: $\mathbb{E}[W_t^1 W_t^2] = \rho t$, $\rho \in (-1, 1)$. If $2\kappa\eta > \xi^2$, it can be shown that $\mathcal{V}_t > 0$ for every $t \in [0, T]$. We have the following definition for the Euler scheme with time step $\frac{T}{n}$, $T \in \mathbb{R}^+$, $n \geq 1$, for $k = 0, \ldots, n-1$

$$\begin{cases} \bar{X}^n_{t^n_{k+1}} = \bar{X}^n_{t^n_k} + r\bar{X}^n_{t^n_k}\dfrac{T}{n} + \sqrt{|\bar{\mathcal{V}}^n_{t^n_k}|}\bar{X}^n_{t^n_k}\Delta W^1_{t^n_{k+1}}, \\ \bar{X}_0 = X_0, \\ \bar{\mathcal{V}}^n_{t^n_{k+1}} = \bar{\mathcal{V}}^n_{t^n_k} + \kappa(\eta - \bar{\mathcal{V}}^n_{t^n_k})\dfrac{T}{n} + \xi\sqrt{|\bar{\mathcal{V}}^n_{t^n_k})|}\Delta W^2_{t^n_{k+1}}, \\ \bar{\mathcal{V}}_0 = \mathcal{V}_0. \end{cases} \tag{2.71}$$

with $\frac{T}{n} = t^n_{k+1} - t^n_k$ and $\Delta W_{t^n_{k+1}} = W_{t^n_{k+1}} - W_{t^n_k}$.

We consider the payoff of a Up–Out Call Barrier option, the price is given by

$$V_T = e^{-rT}\mathbb{E}[\mathbb{1}_{\max\limits_{0\leq t\leq T} X_t < B}(X_T - K)^+]. \tag{2.72}$$

We implemented the previous payoff with the smoothing technique used in the section 1.3.4. The parameters of the Heston model are $X_0 = 100$ and $r = 3\%$ for the dynamic of the underlying and $\mathcal{V}_0 = 10\%$, $\xi = 20\%$, $\kappa = 2$ and $\eta = 10\%$ for the volatility process. The correlation between the two Brownian motions is $\rho = -0.5$. The parameters for the Up–Out Call option are $K = 95$, $B = 120$ and $T = 1$. As there is no closed form formula when the correlation is not set to zero, we took the reference solution from a PDE solver and we used the finite difference method (1% shift of the initial price) to compute the derivatives. Therefore, the reference solution are for the Delta $\Delta_0 = -0.01216316$ and for the Gamma $\Gamma_0 = -0.00135199446$. The reference value for the price is $P_0 = 0.533071$.

The parameters are fixed to $\alpha = 0.5$ and $\beta = 0.5$ for the price. We display the results in tables 2.1 and 2.2 respectively for the weighted and regular multilevel Monte Carlo. Clearly, the weighted framework shows a L_2-error always below the prescribed RMSE ε but the MLMC does not. ML2R is up to 6 times faster than MLMC. On figure 2.6, the time of computation

of the regular multilevel Monte Carlo and the weighted one against the prescribed RMSE ε are displayed and the empirical L_2-error $\tilde{\varepsilon}_L$, ML2R shows better results.

For the Delta, the parameters are fixed to $\alpha = 0.5$ and $\beta = 0.5$. The results are displayed on table 2.3 for the regular multilevel Monte Carlo and 2.4 for the weighted one. Both of the methods respect the constraint on the prescribed RMSE ε but the M2LR has better results in term of numerical cost and time of computation. The weighted multilevel Monte Carlo shows a speed-up up to 5 times faster than the regular one. On figure 2.7, we display the CPU time of standard multilevel Monte Carlo and the weighted one against the prescribed RMSE ε and the empirical L_2-error $\tilde{\varepsilon}_L$, the multilevel Richardson Romberg estimator has the best results.

For the Gamma, we fixed the parameter to $\alpha = 0.5$ and $\beta = 0.75$; actually, it comes from a priori estimation of the β for that derivative in the Heston model. The beta is higher than for the Delta because of the smoothing technique. On table 2.5, the results of the weighted multilevel Monte Carlo and the results for the standard framework are displayed on the table 2.6. As for the Delta, the ML2R shows a better convergence at a lesser numerical cost and time of computation than for the MLMC. It is still faster but the difference is smaller than for the Delta case. On the figure 2.8, we displayed the results of the MLMC and M2LR of the CPU time against the empirical RMSE $\tilde{\varepsilon}_L$ and the prescribed precision ε.

2.7.2 Using the Framework 2.6 for the Value, the Delta and the Gamma of a Call Option

Consider the Black Scholes model

$$\begin{cases} dX_t = rX_t + \sigma X_t dW_t, \\ X_0 = x \in \mathbb{R}, \end{cases} \tag{2.73}$$

with r the risk free rate, σ the volatility and $(W_t)_{t\in[0,T]}$ is a standard Brownian motion. Its corresponding Euler scheme with time step $\frac{T}{n}$, $T \in \mathbb{R}^+$, $n \geq 1$,

$$\begin{cases} \bar{X}^n_{t^n_{k+1}} = \bar{X}^n_{t^n_k} + r\bar{X}^n_{t^n_{k+1}} \dfrac{T}{n} + \sigma \bar{X}^n_{t^n_{k+1}} \Delta W_{t^n_{k+1}}, \quad k = 0, \ldots, n-1, \\ \bar{X}_0 = X_0, \end{cases} \tag{2.74}$$

with $\frac{T}{n} = t^n_{k+1} - t^n_k$ and $\Delta W_{t^n_k} = W_{t^n_{k+1}} - W_{t^n_k}$ for $k = 0, \ldots, n-1$.

We consider the payoff of an European Call option in the Black Scholes model, its price is given by

$$V_T = e^{-rT}\mathbb{E}[(X_T - K)^+]. \tag{2.75}$$

The parameters of the Black Scholes model are $X_0 = 100$, $r = 6\%$ and $\sigma = 40\%$. As for the European Call option, the parameters are set to $T = 1$ and $K = 80$. We compare the results obtained with the framework in section 2.6 for the regular and weighted multilevel Monte Carlo. The parameter α and β are approximated by the framework and given as $(\alpha_P, \beta_P) = (1, 1)$ for the price, the delta is estimated as $(\alpha_\Delta, \beta_\Delta) = (1, 0.5)$ and as for the Gamma $(\alpha_\Gamma, \beta_\Gamma) = (1, 1)$ due to the smoothing technique described in section 1.3.4. The true value for the price, the Delta and the Gamma are given by the closed formula and are given as $P_0 = V_T = 29.4987$, $\Delta_0 = \partial V_T/\partial X_0 = 0.818024$ and $\Gamma_0 = \partial^2 V_T/\partial X_0^2 = 0.00660504$.

The results are displayed on tables 2.7 for the weighted multilevel Monte Carlo and 2.8 for the regular method. The ML2R shows a L_2-error always below the prescribed RMSE ε which is not the case for the MLMC, at it does at a lower cost and can see that numerically,

$$\max\left(\text{Cost}(\hat{\mathbf{I}}^{ML2R})\right) \leq \max\left(\text{Cost}(\hat{\mathbf{I}}^{MLMC})\right). \tag{2.76}$$

Hence, the over all time of computation is always lower for the weighted multilevel Monte Carlo. The same conclusion can barely be made for the others indicators, the empirical bias and variance are smaller for almost all test cases. One can observe that for the same required precision on the price, the ML2R is looking for a lower precision on the the Delta and the Gamma. Taking the example of $k = 8$ for the price, if we look at the results for the Gamma when the ML2R handles a prescribed RMSE $\varepsilon_\Gamma = 3.05e-05$ the MLMC only manages a prescribed RMSE $\varepsilon_\Gamma = 6.10e-05$.

2.7.3 Antithetic Scheme

Let us consider the $2-$dimensional stochastic differential equation (Clark Cameron model), with the initial conditions $X_0 = S_0 = 0$.

$$\begin{cases} X_t = \mu t + \displaystyle\int_0^t dW_t^1, \\ S_t = \sigma \displaystyle\int_0^t X_t dW_t^2, \end{cases} \quad t \in [0,T], \quad \sigma > 0, \quad \mu \in \mathbb{R}. \tag{2.77}$$

where $W = (W^1, W^2)$ is a $2-$dimensional standard Brownian motion, the components are uncorrelated. The Giles Szpruch scheme with time step $\frac{T}{n}$, $T \in \mathbb{R}^+$ and $n \geq 1$ for this model is given by, $k = 1, \ldots, n-1$

$$\begin{cases} \bar{X}^n_{t^n_{k+1}} = \bar{X}^n_{t^n_k} + \mu \dfrac{T}{n} + \Delta W^1_{t^n_{k+1}} \\ \bar{S}^n_{t^n_{k+1}} = \bar{S}^n_{t^n_k} + \sigma \bar{X}^n_{t^n_k} \Delta W^2_{t^n_{k+1}} + \dfrac{1}{2}\sigma \Delta W^1_{t^n_{k+1}} \Delta W^2_{t^n_{k+1}} \end{cases} \tag{2.78}$$

For a smooth payoff function, following Giles Szpruch [66] and Al Gerbi et al. [57], we have the settings $\alpha = 1$ and $\beta = 2$ for the antithetic Giles Szpruch model.

Following the work of Giorgi [67], consider the smooth payoff function $f(x,s) = 10\cos(\sigma s)$. Hence, the price is given by

$$V_T = 10\mathbb{E}[\cos(\sigma S_T)]. \tag{2.79}$$

The closed formula (deduced from Pitman and Yor [155]) for this payoff is given by

$$\mathbb{E}[\cos(\sigma S_t)] = \frac{1}{\sqrt{\cosh(\sigma t)}} \exp\left(-\frac{\mu^2 t}{2}\left(1 - \frac{\tanh(\sigma t)}{\sigma t}\right)\right). \tag{2.80}$$

With initial conditions $X_0 = S_0 = 0$, the maturity $T = 1$, $\mu = 1$ and the volatility $\sigma = 1$, the closed formula gives the true value $P_0 = 7.14556$, the Vega is $\mathfrak{V}_0 = \partial V_T/\partial\sigma = -3.94154$ and the Vomma is $\mathscr{V}_0 = \partial^2 V_2/\partial\sigma^2 = 0.829263$.

We compare the regular and weighted multilevel Monte Carlo for the approximation of the Vega and Vomma of the smooth payoff function. For the Vega, the results are displayed in 2.9 for the ML2R and in 2.10 for the MLMC. The time of computation are from 10% to 100% better for the weighted framework than for the regular one (same for the numerical cost). For the same root $M = 2$, MLMC requires several more levels than ML2R (up to twice) and with more simulation paths. On figure 2.9, we displayed the results of MLMC and ML2R of the CPU time against the empirical RMSE ε_L and the prescribed RMSE $\tilde{\varepsilon}_L$. On both figures, one can analyze the superiority of the ML2R on MLMC. For the Vomma, the results are displayed on the figure 2.10, the tables 2.11 and 2.9 respectively for the ML2R and MLMC. The same conclusion are holding.

2.8 Conclusion

In this chapter, we investigated the computation of the sensitivities with a weighted multilevel Monte Carlo.

We, also, proposed an efficient implementation of a global framework which computes the price, the Delta and the Gamma of a financial security using a simple regularization for the approximation of the sensitivities. The numerical results on standard and antithetic schemes shown the superiority of the weighted multilevel Monte Carlo on the regular method. One question remains about the consistency between the value and its sensitivities as we do not use the same number of simulation paths for each of them.

Future directions for furthering the development of the scheme will concern the American case.

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
1	5.00e−01	1.48e−01	1.13e−03	−9.58e−02	1.28e−02	3	4	1	5.68e+03	3.15e+04
2	2.50e−01	1.32e−01	4.48e−03	−1.19e−01	3.21e−03	3	4	1	2.27e+04	1.26e+05
3	1.25e−01	4.16e−02	1.83e−02	−3.33e−02	6.20e−04	3	7	1	8.30e+04	5.64e+05
4	6.25e−02	1.04e−02	1.85e−01	−4.00e−03	9.18e−05	3	10	2	4.44e+05	3.73e+06
5	3.12e−02	5.12e−03	6.15e−01	1.29e−03	2.46e−05	4	5	1	2.18e+06	2.26e+07
6	1.56e−02	2.40e−03	2.69e+00	−2.06e−05	5.74e−06	4	6	1	8.84e+06	1.01e+08
7	7.81e−03	1.17e−03	1.18e+01	1.55e−04	1.36e−06	4	7	1	3.62e+07	4.52e+08

Table 2.1: Price of a UOC ($\alpha = 0.5$, $\beta = 0.5$): Parameters and results of the ML2R estimator in the Heston model.

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
1	5.00e−01	6.59e−01	3.47e−04	6.46e−01	1.80e−02	2	8	1	2.48e+03	6.32e+03
2	2.50e−01	2.82e−01	3.61e−03	2.78e−01	2.37e−03	3	6	1	2.06e+04	9.17e+04
3	1.25e−01	1.10e−01	3.57e−02	1.08e−01	3.29e−04	4	6	1	1.50e+05	1.18e+06
4	6.25e−02	7.25e−02	1.88e−01	7.20e−02	7.01e−05	4	8	1	6.80e+05	6.73e+06
5	3.12e−02	3.16e−02	1.70e+00	3.15e−02	1.08e−05	5	7	1	4.49e+06	6.96e+07
6	1.56e−02	1.49e−02	1.18e+01	1.48e−02	2.04e−06	5	10	1	2.29e+07	5.09e+08
7	7.81e−03	7.72e−03	9.16e+01	7.70e−03	3.49e−07	6	8	1	1.37e+08	4.24e+09

Table 2.2: Price of a UOC ($\alpha = 0.5$, $\beta = 0.5$): Parameters and results of the MLMC estimator in the Heston model.

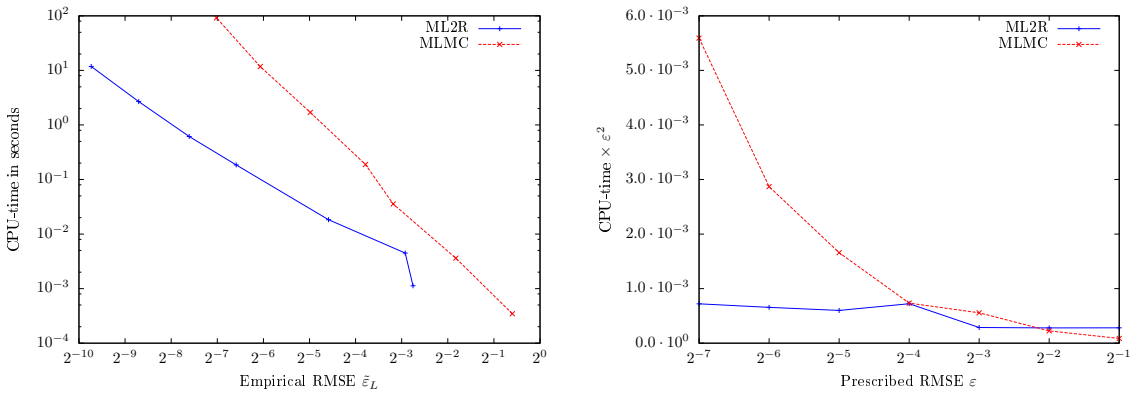

Figure 2.6: Price of a UOC in the Heston Model (MLMC vs ML2R). On the left, CPU-time (y-axis, log scale) as a function of $\tilde{\varepsilon}_L$ (x-axis, $\log_2$ scale). On the right, CPU-time $\times\ \varepsilon^2$ (y-axis) as a function of ε (x-axis, $\log_2$ scale).

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
3	1.25e−01	3.22e−02	1.22e−03	9.34e−03	9.50e−04	3	7	1	5.91e+03	3.46e+04
4	6.25e−02	1.21e−02	1.31e−02	4.20e−03	1.30e−04	3	10	2	3.54e+04	2.53e+05
5	3.12e−02	6.75e−03	3.30e−02	−1.96e−03	4.17e−05	4	5	1	1.38e+05	1.10e+06
6	1.56e−02	3.90e−03	1.36e−01	2.27e−03	1.01e−05	4	6	1	5.42e+05	4.60e+06
7	7.81e−03	1.60e−03	5.66e−01	3.51e−04	2.44e−06	4	7	1	2.17e+06	1.95e+07
8	3.91e−03	1.36e−03	2.50e+00	1.13e−03	5.78e−07	4	9	1	8.86e+06	8.86e+07
9	1.95e−03	5.03e−04	1.37e+01	3.65e−04	1.20e−07	5	5	1	4.68e+07	5.15e+08

Table 2.3: Delta of a UOC ($\alpha = 0.5$, $\beta = 0.5$): Parameters and results of the ML2R estimator in the Heston model.

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
3	1.25e−01	2.36e−02	2.09e−03	−7.63e−03	5.42e−04	4	6	1	9.92e+03	6.20e+04
4	6.25e−02	1.12e−02	9.79e−03	5.08e−03	1.22e−04	4	8	1	4.24e+04	3.13e+05
5	3.12e−02	4.69e−03	6.74e−02	−5.11e−03	2.18e−05	5	7	1	2.42e+05	2.43e+06
6	1.56e−02	2.31e−03	3.67e−01	8.69e−03	4.59e−06	5	10	1	1.10e+06	1.41e+07
7	7.81e−03	1.50e−03	2.22e+00	3.29e−03	8.96e−07	6	8	1	5.77e+06	9.07e+07
8	3.91e−03	1.13e−03	1.60e+01	1.69e−03	1.61e−07	7	8	1	3.21e+07	6.99e+08
9	1.95e−03	6.37e−04	7.48e+01	8.53e−03	3.73e−08	7	9	1	1.37e+08	3.30e+09

Table 2.4: Delta of a UOC ($\alpha = 0.5$, $\beta = 0.5$): Parameters and results of the MLMC estimator in the Heston model.

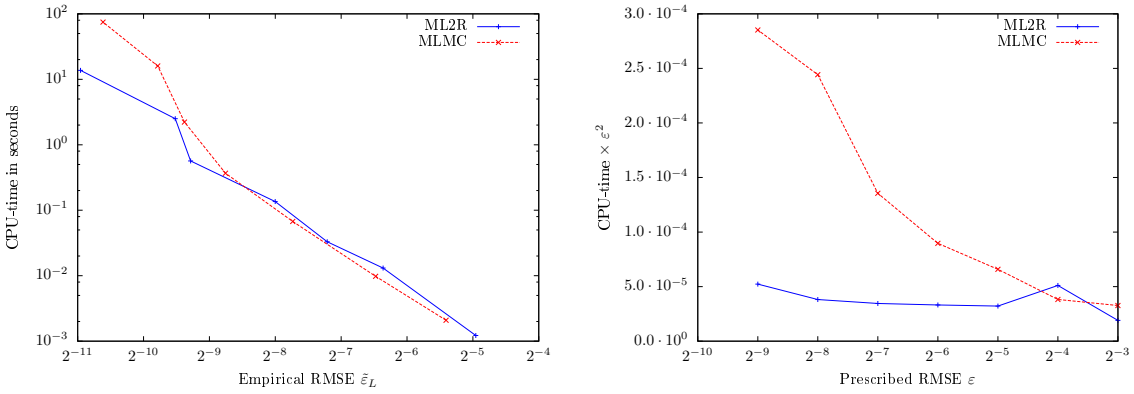

Figure 2.7: Delta of a UOC in the Heston Model (MLMC vs ML2R). On the left, CPU-time (y-axis, log scale) as a function of $\tilde{\varepsilon}_L$ (x-axis, $\log_2$ scale). On the right, CPU-time $\times\ \varepsilon^2$ (y-axis) as a function of ε (x-axis, $\log_2$ scale).

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
4	6.25e−02	1.25e−02	1.81e−02	1.09e−03	1.55e−04	3	10	2	4.86e+04	3.49e+05
5	3.12e−02	7.60e−03	4.56e−02	−2.90e−03	4.94e−05	4	5	1	1.89e+05	1.51e+06
6	1.56e−02	3.76e−03	1.88e−01	1.49e−03	1.19e−05	4	6	1	7.44e+05	6.36e+06
7	7.81e−03	1.71e−03	7.84e−01	1.56e−04	2.89e−06	4	7	1	2.98e+06	2.70e+07
8	3.91e−03	9.97e−04	3.45e+00	5.58e−04	6.83e−07	4	9	1	1.22e+07	1.22e+08
9	1.95e−03	3.80e−04	1.90e+01	−5.61e−05	1.42e−07	5	5	1	6.42e+07	7.12e+08
10	9.77e−04	2.80e−04	7.60e+01	−2.07e−04	3.54e−08	5	5	1	2.57e+08	2.85e+09

Table 2.5: Gamma of a UOC ($\alpha = 0.5$, $\beta = 0.75$): Parameters and results of the ML2R estimator in the Heston model.

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
4	6.25e−02	1.56e−02	1.35e−02	9.98e−03	1.44e−04	4	8	1	5.81e+04	4.33e+05
5	3.12e−02	9.87e−03	9.32e−02	7.01e−03	2.57e−05	5	7	1	3.32e+05	3.36e+06
6	1.56e−02	6.41e−03	5.09e−01	−5.74e−03	5.43e−06	5	10	1	1.51e+06	1.95e+07
7	7.81e−03	4.51e−03	3.07e+00	−2.83e−03	1.06e−06	6	8	1	7.92e+06	1.25e+08
8	3.91e−03	1.81e−03	2.21e+01	1.03e−03	1.91e−07	7	8	1	4.40e+07	9.66e+08
9	1.95e−03	6.04e−04	1.04e+02	−7.10e−04	4.40e−08	7	9	1	1.88e+08	4.57e+09
10	9.77e−04	3.91e−04	6.24e+02	−5.91e−04	8.74e−09	8	8	1	9.59e+08	2.87e+10

Table 2.6: Gamma of a UOC ($\alpha = 0.5$, $\beta = 0.75$): Parameters and results of the MLMC estimator in the Heston model.

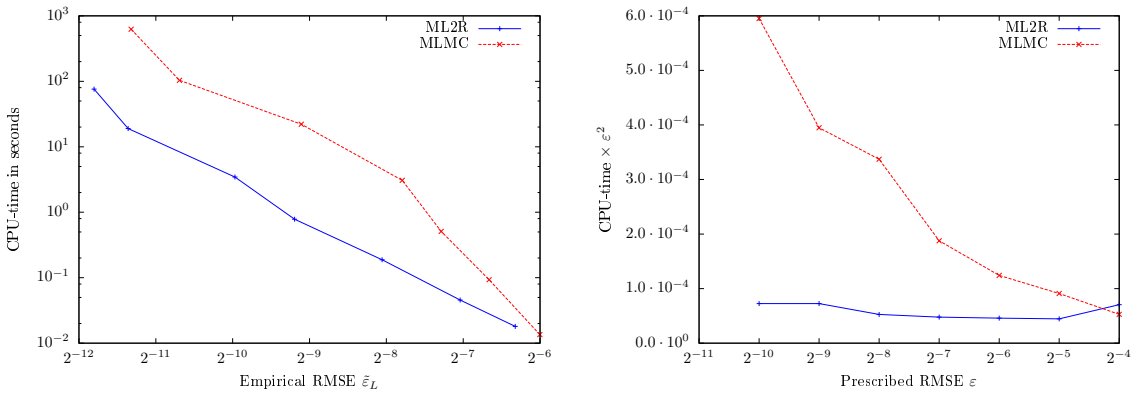

Figure 2.8: Gamma of a UOC in the Heston Model (MLMC vs ML2R). On the left, CPU-time (y-axis, log scale) as a function of $\tilde{\varepsilon}_L$ (x-axis, $\log_2$ scale). On the right, CPU-time $\times\ \varepsilon^2$ (y-axis) as a function of ε (x-axis, $\log_2$ scale).

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
2	2.50e−01	2.02e−01	5.67e−03	9.15e−02	3.23e−02	2	9	1	5.89e+04	1.05e+05
–	7.81e−03	4.51e−03	–	4.87e−05	2.03e−05	3	–	1	3.10e+04	8.22e+04
–	3.91e−03	1.02e−03	–	−3.62e−04	9.17e−07	3	–	1	8.44e+03	3.92e+04
5	3.12e−02	2.01e−02	5.90e−01	−3.66e−03	3.92e−04	3	5	1	4.98e+06	1.14e+07
–	9.77e−04	5.72e−04	–	2.23e−04	2.77e−07	4	–	1	2.41e+06	7.26e+06
–	2.44e−04	5.50e−05	–	5.91e−06	2.99e−09	4	–	1	2.89e+06	1.51e+07
8	3.91e−03	2.36e−03	4.23e+01	1.48e−04	5.53e−06	3	9	1	3.24e+08	8.36e+08
–	1.22e−04	7.04e−05	–	3.24e−05	3.91e−09	4	–	1	1.60e+08	5.56e+08
–	3.05e−05	8.20e−06	–	−5.14e−06	4.08e−11	4	–	1	1.85e+08	1.19e+09

Table 2.7: Results for framework 2.6 in the Black Scholes model (ML2R).

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
2	2.50e−01	2.84e−01	7.41e−03	2.34e−01	2.58e−02	2	7	1	6.46e+04	1.05e+05
–	1.56e−02	6.04e−03	–	−7.74e−04	3.59e−05	4	–	1	1.80e+04	8.67e+04
–	3.91e−03	8.58e−04	–	5.50e−04	4.35e−07	5	–	1	1.81e+04	1.30e+05
5	3.12e−02	2.98e−02	8.19e−01	2.45e−02	2.89e−04	3	8	1	6.14e+06	1.47e+07
–	1.95e−03	5.93e−04	–	2.17e−04	3.04e−07	5	–	1	2.11e+06	1.90e+07
–	4.88e−04	2.34e−04	–	2.20e−04	6.53e−09	5	–	1	1.18e+06	8.95e+06
8	3.91e−03	4.32e−03	6.22e+01	3.89e−03	3.47e−06	4	8	1	5.15e+08	1.62e+09
–	4.88e−04	2.15e−04	–	1.65e−04	1.90e−08	5	–	1	3.38e+07	3.04e+08
–	6.10e−05	2.61e−05	–	2.44e−05	8.35e−11	6	–	1	9.22e+07	8.58e+08

Table 2.8: Results for framework 2.6 in the Black Scholes model (MLMC).

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
3	1.25e−01	9.54e−02	1.03e−02	−6.47e−03	9.07e−03	4	2	1	1.89e+04	3.16e+04
4	6.25e−02	4.83e−02	4.17e−02	−7.40e−03	2.27e−03	4	2	1	7.59e+04	1.26e+05
5	3.12e−02	1.95e−02	1.80e−01	2.61e−03	3.73e−04	5	2	1	3.34e+05	5.77e+05
6	1.56e−02	9.67e−03	7.60e−01	6.73e−04	9.32e−05	5	2	1	1.33e+06	2.30e+06
7	7.81e−03	4.83e−03	3.04e+00	3.19e−04	2.33e−05	5	2	1	5.34e+06	9.23e+06
8	3.91e−03	2.41e−03	1.21e+01	−1.46e−04	5.82e−06	5	2	1	2.13e+07	3.69e+07
9	1.95e−03	1.08e−03	4.91e+01	−2.219e−04	1.12e−06	6	2	1	9.07e+07	1.60e+08

Table 2.9: Vega of a smooth function ($\alpha = 1$, $\beta = 2$): Parameters and results of the ML2R estimator in the Clark Cameron model.

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
3	1.25e−01	8.59e−02	1.38e−02	6.25e−02	3.34e−03	6	2	1	2.49e+04	5.28e+04
4	6.25e−02	4.89e−02	6.44e−02	4.01e−02	7.84e−04	7	2	1	1.11e+05	2.42e+05
5	3.12e−02	1.97e−02	2.85e−01	1.44e−02	1.81e−04	8	2	1	4.82e+05	1.09e+06
6	1.56e−02	1.02e−02	1.22e+00	7.83e−03	4.31e−05	9	2	1	2.03e+06	4.70e+06
7	7.81e−03	7.05e−03	5.14e+00	6.48e−03	1.04e−05	10	2	1	8.45e+06	1.97e+07
8	3.91e−03	2.98e−03	2.12e+01	2.47e−03	2.53e−06	11	2	1	3.46e+07	8.18e+07
9	1.95e−03	1.41e−03	8.85e+01	1.17e−03	6.22e−07	12	2	1	1.41e+08	3.35e+08

Table 2.10: Vega of a smooth function ($\alpha = 1$, $\beta = 2$): Parameters and results of the MLMC estimator in the Clark Cameron model.

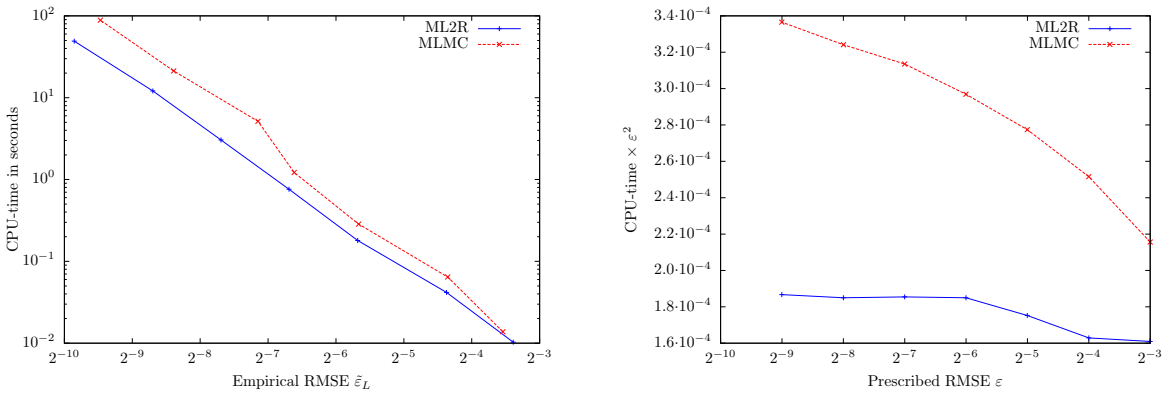

Figure 2.9: Vega in the Clark Cameron Model (MLMC vs ML2R). On the left, CPU-time (y-axis, log scale) as a function of $\tilde{\varepsilon}_L$ (x-axis, $\log_2$ scale). On the right, CPU-time $\times\ \varepsilon^2$ (y-axis) as a function of ε (x-axis, $\log_2$ scale).

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
3	1.25e−01	1.04e−01	2.65e−02	−2.72e−02	1.90e−02	4	2	1	4.82e+04	7.48e+04
4	6.25e−02	6.95e−02	1.06e−01	−5.47e−03	4.81e−03	4	2	1	1.92e+05	2.99e+05
5	3.12e−02	2.70e−02	4.60e−01	−2.65e−03	7.25e−04	5	2	1	8.50e+05	1.36e+06
6	1.56e−02	1.36e−02	1.94e+00	2.19e−03	1.81e−04	5	2	1	3.40e+06	5.44e+06
7	7.81e−03	6.73e−03	7.77e+00	−7.20e−05	4.53e−05	5	2	1	1.36e+07	2.17e+07
8	3.91e−03	3.39e−03	3.11e+01	−4.51e−04	1.13e−05	5	2	1	5.44e+07	8.71e+07
9	1.95e−03	1.44e−03	1.26e+02	2.27e−04	2.04e−06	6	2	1	2.31e+08	3.77e+08

Table 2.11: Vomma of a smooth function ($\alpha = 1$, $\beta = 2$): Parameters and results of the ML2R estimator in the Clark Cameron model.

k	$\varepsilon = 2^{-k}$	L^2-error	time (s)	bias	variance	R	M	h^{-1}	N	Cost
3	1.25e−01	1.11e−01	3.51e−02	−8.18e−02	5.81e−03	6	2	1	6.32e+04	1.21e+05
4	6.25e−02	5.84e−02	1.63e−01	−4.58e−02	1.31e−03	7	2	1	2.84e+05	5.67e+05
5	3.12e−02	2.75e−02	7.27e−01	−2.12e−02	3.05e−04	8	2	1	1.23e+06	2.51e+06
6	1.56e−02	1.17e−02	3.12e+00	−8.09e−03	7.22e−05	9	2	1	5.19e+06	1.08e+07
7	7.81e−03	5.37e−03	1.31e+01	−3.39e−03	1.74e−05	10	2	1	2.15e+07	4.54e+07
8	3.91e−03	2.38e−03	5.44e+01	−1.20e−03	4.24e−06	11	2	1	8.86e+07	1.88e+08
9	1.95e−03	1.37e−03	2.23e+02	−9.14e−04	1.04e−06	12	2	1	3.60e+08	7.70e+08

Table 2.12: Vomma of a smooth function ($\alpha = 1$, $\beta = 2$): Parameters and results of the MLMC estimator in the Clark Cameron model.

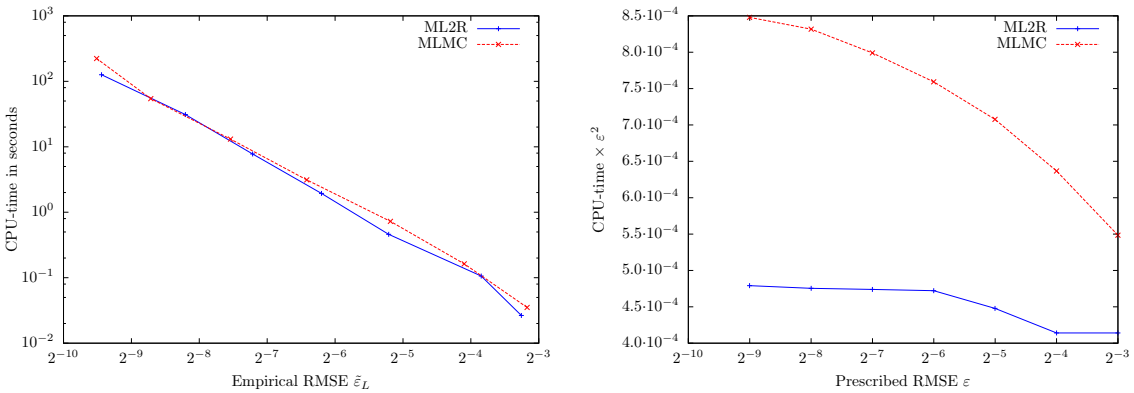

Figure 2.10: Vomma of a smooth function in the Clark Cameron Model (MLMC vs ML2R). On the left, CPU-time (y-axis, log scale) as a function of $\tilde{\varepsilon}_L$ (x-axis, $\log_2$ scale). On the right, CPU-time $\times\ \varepsilon^2$ (y-axis) as a function of ε (x-axis, $\log_2$ scale).

Algorithm 2 Framework for the evaluation of sensitivities and price with optimal settings.

Require: aimed precision ε_P; bias parameter $M^P_{\max}$; minimum precision $\varepsilon_{\min}$; J number of replications; *(optional)*: α_P, α_Δ and α_Γ

Ensure: mean, variance, bias and L^2-error for the price, the Delta and the Gamma

compute asymptotically the parameter $\beta_P \sim \lim_{n\to+\infty} \log_2(\mathrm{Var}(Y^{(n-1),P}_{\frac{T}{2^{n-1}}})) - \log_2(\mathrm{Var}(Y^{(n),P}_{\frac{T}{2^n}}))$, (resp. for Δ and Γ)

(optional): approximate asymptotically the parameter $\alpha_P \sim \lim_{n\to+\infty} \log_2(\mathbb{E}(Y^{(n-1),P}_{\frac{T}{2^{n-1}}})) - \log_2(\mathbb{E}(Y^{(n),P}_{\frac{T}{2^n}}))$, (resp. for Δ and Γ) if they are not provided.

compute the structural parameter dependencies $\mathrm{Var}(Y^P_0)$, $V^P_1 = (1 + M_{\max}^{-\frac{\beta_P}{2}})^{-2} h^{-\beta_P} \|Y^P_h - Y^P_{\frac{h}{M_{\max}}}\|_2^2$ and $\theta^P = \sqrt{\frac{V^P_1}{\mathrm{Var}(Y^P_0)}}$ (resp. for Δ and Γ)

compute the coefficient $c_1 \sim \hat{c}_1 = \frac{h^{-\alpha}}{(1-2^{-\alpha})} \frac{1}{I} \sum_{k=1}^{I} Y^k_h - Y^k_{\frac{h}{2}}$

find the optimal bias parameter $M^{P*} = \underset{M^P \in \{2,\dots,M^P_{\max}\}}{\arg\min} C_{\varepsilon^P}(M^P)$

select the set of structural parameters corresponding to M^{P*}

set $M^{\Delta*} = M^{P*}$ and $M^{\Gamma*} = M^{P*}$

find the optimal precisions $\varepsilon^*_\Delta = \underset{\substack{\varepsilon_\Delta \in \{\varepsilon_1,\dots,\varepsilon_{\min}\} \\ \text{s.t. } h(\varepsilon_\Delta)=h(\varepsilon_P)}}{\arg\min} \left| C_{\varepsilon_\Delta}(M^{\Delta*}) - C_{\varepsilon_P}(M^{P*}) \right|$ (resp. for the Γ)

select the set of structural parameters corresponding to $M^{\Delta*}$ and $M^{\Gamma*}$

compute the maximum level of the framework $R_{\max} = \max(R^P, R^\Delta, R^\Gamma)$

compute the maximum number of trajectories per level $\tilde{N}_j = \max(N^P_j, N^\Delta_j, N^\Gamma_j)$, $j = 1, \dots, R_{\max}$

construct the matrices $\mathbf{D}^{(j)}$, $\mathbf{H}^{(j)}$ and $\mathbf{Q}^{(j)}$ for $j = 1, \dots, R_{\max}$

for j=1 to J **do**

  compute the mean of the first level $\hat{\mathbf{I}}^{ML2R}_j = \mathbf{D}^{(1)} \sum_{r=1}^{\tilde{N}_1} \mathbf{Q}^{(1),r} \mathbf{Y}^{(1),r,j}_h$

  compute the variance of the first level $\hat{\mathbf{V}}^{ML2R}_j = \mathbf{D}^{(1)} \mathbf{V}^{(1),j}_h$

  for i = 2 to $R_{\max}$ **do**

    compute the other levels $\hat{\mathbf{I}}^{ML2R}_j = \hat{\mathbf{I}}^{ML2R}_j + \mathbf{D}^{(i)} \mathbf{H}^{(i)} \sum_{r=1}^{\tilde{N}_i} \mathbf{Q}^{(i),r} \left(\mathbf{Y}^{(i),r,j}_{\frac{h}{n_i}} - \mathbf{Y}^{(i),r,j}_{\frac{h}{n_{i-1}}} \right)$

    compute the rest of the variance $\hat{\mathbf{V}}^{ML2R}_j = \hat{\mathbf{V}}^{ML2R}_j + \mathbf{D}^{(i)} (\mathbf{H}^{(i)})^2 \mathbf{V}^{(i),j}_h$

  end for

  Update the mean $\mathrm{Esp}(\hat{\mathbf{I}}^{ML2R}) = \mathrm{Esp}(\hat{\mathbf{I}}^{ML2R}) + \hat{\mathbf{I}}^{ML2R}_j$

  Update the variance $\mathrm{Var}(\hat{\mathbf{I}}^{ML2R}) = \mathrm{Var}(\hat{\mathbf{I}}^{ML2R}) + \hat{\mathbf{V}}^{ML2R}_j$

  Update the bias $\mathrm{Bias}(\hat{\mathbf{I}}^{ML2R}) = \mathrm{Bias}(\hat{\mathbf{I}}^{ML2R}) + \left(\hat{\mathbf{I}}^{ML2R}_j - \mathbf{I}_0 \right)$

  Update the L^2-error $\mathrm{RMSE}(\hat{\mathbf{I}}^{ML2R}) = \mathrm{RMSE}(\hat{\mathbf{I}}^{ML2R}) + \left(\hat{\mathbf{I}}^{ML2R}_j - \mathbf{I}_0 \right) \circ \left(\hat{\mathbf{I}}^{ML2R}_j - \mathbf{I}_0 \right)$

end for

compute the average of the mean, the variance, the bias and the RMSE

compute the $\mathrm{RMSE}(\hat{\mathbf{I}}^{ML2R}) = \mathbf{F}\left(\mathrm{RMSE}(\hat{\mathbf{I}}^{ML2R}) \right)$

Algorithm 3 Detailed part of steps 16:20 of Algorithm 2 (Heston model, Call option).

for r = 2 to $R_{\max}$ **do**
  for i= 1 to $\tilde{N}_r$ **do**
    initiate $X^{(r),i}_{\frac{h}{n_r}} = X_0$, $X^{(r),i}_{\frac{h}{n_{r-1}}} = X_0$, $\mathcal{V}^{(r),i}_{\frac{h}{n_r}} = \mathcal{V}_0$, $\mathcal{V}^{(r),i}_{\frac{h}{n_{r-1}}} = \mathcal{V}_0$, $X^{(r)}_{\min} = X_0$ and $X^{(r-1)}_{\min} = X_0$
    for j = 1 to $h(\varepsilon_P)^{-1}$ **do**
      for k = 1 to n_{r-1} **do**
        initiate $W^{(r),i,X}_{\frac{h}{n_{r-1}}} = 0$ and $W^{(r),i,\mathcal{V}}_{\frac{h}{n_{r-1}}} = 0$
        for l = 1 to M^{P*} **do**
          generate $Z^{(r),i,\mathcal{V}}_{j,k,l}$ and $Z^{(r),i,X}_{j,k,l}$, $Z \sim \mathcal{N}(0,1)$
          set $W^{(r),i,X}_{\frac{h}{n_r}} \sim \sqrt{\frac{h}{n_r}} Z^{(r),i,X}_{j,k,l}$, $W^{(r),i,\mathcal{V}}_{\frac{h}{n_r}} \sim \sqrt{\frac{h}{n_r}} \left(\rho Z^{(r),iX}_{j,k,l} + \sqrt{1-\rho^2} Z^{(r),i,\mathcal{V}}_{j,k,l}\right)$
          compute $W^{(r),i,X}_{\frac{h}{n_r}} = W^{(r),i,X}_{\frac{h}{n_{r-1}}} + W^{(r),i,X}_{\frac{h}{n_r}}$, $W^{(r),i,\mathcal{V}}_{\frac{h}{n_r}} = W^{(r),i,\mathcal{V}}_{\frac{h}{n_{r-1}}} + W^{(r),i,\mathcal{V}}_{\frac{h}{n_r}}$
          compute $X^{(r),i}_{\frac{h}{n_r}} = X^{(r),i}_{\frac{h}{n_r}} \left(1 + \mu \frac{h}{n_r} + \sqrt{|\mathcal{V}^{(r),i}_{\frac{h}{n_r}}|} W^{(r),i,X}_{\frac{h}{n_r}}\right)$
          compute $\mathcal{V}^{(r),i}_{\frac{h}{n_r}} = \mathcal{V}^{(r),i}_{\frac{h}{n_r}} + \kappa \left(\theta - \mathcal{V}^{(r),i}_{\frac{h}{n_r}}\right) \frac{h}{n_r} + \xi \sqrt{|\mathcal{V}^{(r),i}_{\frac{h}{n_r}}|} W^{(r),i,\mathcal{V}}_{\frac{h}{n_r}}$
        end for
        compute $X^{(r),i}_{\frac{h}{n_{r-1}}} = X^{(r),i}_{\frac{h}{n_{r-1}}} \left(1 + \mu \frac{h}{n_{r-1}} + \sqrt{|\mathcal{V}^{(r),i}_{\frac{h}{n_{r-1}}}|} W^{(r),i}_{\frac{h}{n_{r-1}}}\right)$
        compute $\mathcal{V}^{(r),i}_{\frac{h}{n_{r-1}}} = \mathcal{V}^{(r),i}_{\frac{h}{n_{r-1}}} + \kappa \left(\theta - \mathcal{V}^{(r),i}_{\frac{h}{n_{r-1}}}\right) \frac{h}{n_r} + \xi \sqrt{|\mathcal{V}^{(r),i}_{\frac{h}{n_{r-1}}}|} W^{(r),i,\mathcal{V}}_{\frac{h}{n_{r-1}}}$
      end for
    end for
    if $r \leq R^P$ **and** $i \leq N^P_r$ **then**
      compute $Y^{(r),i,P}_{\frac{h}{n_r}} = \left(X^{(r),i}_{\frac{h}{n_r}} - K\right)_+$, $Y^{(r),i,P}_{\frac{h}{n_{r-1}}} = \left(X^{(r),i}_{\frac{h}{n_{r-1}}} - K\right)_+$
      discount $Y^{(r),i,P}_{\frac{h}{n_r}}$ and $Y^{(r),i,P}_{\frac{h}{n_{r-1}}}$
      update $\hat{I}^{ML2R}_P = \hat{I}^{ML2R}_P + \hat{W}^{(r)}_P \left(Y^{(r),i,P}_{\frac{h}{n_r}} - Y^{(r-1),i,P}_{\frac{h}{n_{r-1}}}\right)$
      update $\mathrm{Var}(\hat{I}^{ML2R}_P) = \mathrm{Var}(\hat{I}^{ML2R}_P) + \left(\hat{W}^{(r)}_P \left(Y^{(r),i,P}_{\frac{h}{n_r}} - Y^{(r),i,P}_{\frac{h}{n_{r-1}}}\right)\right)^2$
    end if
    for the Δ follow steps 18:23 of Algorithm 2, replace P by Δ then compute with automatic differentiation $Y^{(r),i,\Delta}_{\frac{h}{n_r}} = \frac{\partial}{\partial X_0}\left(Y^{(r),i,P}_{\frac{h}{n_r}}\right)$ and $Y^{(r),i,\Delta}_{\frac{h}{n_{r-1}}} = \frac{\partial}{\partial X_0}\left(Y^{(r),i,P}_{\frac{h}{n_{r-1}}}\right)$
    for the Γ follow steps 18:23 of Algorithm 2, replace P by Γ and compute with automatic differentiation $Y^{(r),i,\Gamma}_{\frac{h}{n_r}} = \frac{\partial^2}{\partial X_0^2}\left(Y^{(r),i,P}_{\frac{h}{n_r}}\right)$ then $Y^{(r),i,\Gamma}_{\frac{h}{n_{r-1}}} = \frac{\partial^2}{\partial X_0^2}\left(Y^{(r),i,P}_{\frac{h}{n_{r-1}}}\right)$
  end for
end for
average the mean $\hat{I}^{ML2R}_P$ and the variance $\mathrm{Var}(\hat{I}^{ML2R}_P)$ (resp. Δ and Γ)
compute $\mathrm{Var}(\hat{I}^{ML2R}_P) = \mathrm{Var}(\hat{I}^{ML2R}_P) - (\hat{I}^{ML2R}_P)^2$ (resp. Δ and Γ)

Around Pricing Method

Chapitre 3

The Parareal Algorithm for American Options

"Progress imposes not only new possibilities for the future but new restrictions"
extract from The Human Use of Human Beings: Cybernetics and Society, [182].

– N. Wiener, Mathematician, (1894, 1964).

With parallelism in mind we investigate the parareal method for American contracts both theoretically and numerically. Least-Square Monte-Carlo (LSMC) and parareal time decomposition with two or more levels are used, leading to an efficient parallel implementation which scales linearly with the number of processors and is appropriate to any multiprocessor-memory architecture in its multilevel version. We prove L^2 superlinear convergence for an LSMC backward in time computation of American contracts, when the conditional expectations are known, i.e. before Monte-Carlo discretization. The argument provides also a tool to analyze the multi-level parareal algorithm; in all cases the computing time is increased only by a constant, compared to the sequential algorithm on the finest grid, and speed-up is guaranteed when the number of processors is larger than that constant. A numerical implementation confirms the theoretical error estimates.

This chapter is an extended version of the article Pagès et al. [141] published in *Les Comptes Rendus Mathématiques de l'Académie des Sciences.*

3.1 Introduction

American contracts are not easy to compute on a parallel computer; even if a large number of them have to be computed at once, an embarrassingly parallel problem, still the cost of the transfer of data, makes parallelism at the level of one contract attractive.

The importance of the problem is reflected by the large literature on parallelism and GPU implementation of numerical methods for option pricing see Abbas-Turki and Lapeyre [1], Benguigui [17], Benguigui and Baude [18], Herrera and Paulot [91], Fatica and Phillips [53], Girard and Toke [69], Hu et al. [97], Khodja et al. [112], Pagès and Wilbertz [145] and Dang et al. [49]. For instance, careful implementations of LSMC have been proposed by M. Fatica and E.

Phillips [53] and J. A. Varela et al. [179] by optimizing the matrix-vector multiply and random numbers generators for a GPU; such optimization can reduce the computing time by an order of magnitude. As LSMC is not modified, but only optimized, these are not in competition with parareal but rather complementary in the sense that they can be combined with the parareal approach proposed here.

A parallel method shown to scale linearly with the number of processors is not easy to find for American contracts. Furthermore it is important that such method to be applicable also when the number of underlying assets is large (see Aitsahlia and Carr [3] and Wan et al. [181]). Note that even sparse grid for partial differential equations Achdou and Pironneau [2] is limited in practice.

The most popular sequential algorithm, the one which we adopt here, is the Least Square Monte-Carlo (LSMC) method of Tsitsiklis and van Roy [178] (incorrectly called Longstaff Schwarz [124] in the finance industry). Exploiting parallelism by allocating blocks of Monte-Carlo paths to different processors is not convincingly efficient Abbas-Turki and Baude [18] because the backward regression is essentially sequential and needs all Monte-Carlo paths in the same processor.

In this chapter, we investigate the parareal method, introduced in Lions et al. [122], for the task. The results presented here have been announced in Pagès et al. [141] without proofs; here all details are given. An earlier study by Bal and Maday [10] has paved the way but it is restricted to Stochastic Differential Equations (SDE) without LSMC. Yet it contains a convergence proof for the two levels method in the restricted case where the solution is computed exactly at the lowest level Bal and Maday [10].

After stating the problem and recalling LSMC in Section 1, we briefly present the parareal method for ODE, the results established in [122] and the extension of Bal and Maday for SDE [10]. Next, new convergence rates are proved for the two level method in the general case involving both a coarse and and a refined schemes and a discussion is given for the multi-levels case.

This chapter is organized was follows; in section 3.2 we introduce and briefly recall the American contract assessing problem. In section 3.3, we begin by describing the two level parareal algorithm for an ordinary differential equation and give an illustration of a schematic representation of the method.

In section 3.4, we give a modified version of the previous algorithm given in section 3.3 by correcting the *Markovian deficiency*.

In section 3.5, we give the algorithm for the American problem i.e. the parareal method for backward dynamic programming principle. We establish a theorem about the convergence of the scheme with the iterations and give the proof of the theorem and its corollary.

The section 3.6 is dedicated to numerical results and applications, we give convergence with respect to the number of iterations for the simulation of a geometric Brownian motion and for an American Put option. We, also, give a third algorithm i.e. the so-called multilevel parareal which is based on L level.

In section 3.7, we make an application for the computation of the Delta of an American Put option using the Automatic Differentiation. Then, we conclude with a summary of the work accomplished and some perspective for future work.

3.2 The Problem

With the usual notations in Kloeden et al. [114] consider a probability space $(\Omega, \mathscr{A}, \mathbb{P})$, and functions $b, \sigma, f : [0, T] \times \mathbb{R} \mapsto \mathbb{R}$, uniformly Lipschitz continuous in x, t.

Let $W = (W_t)_{t\in[0,T]}$ be a standard scalar Brownian motion on $(\Omega, \mathcal{A}, \mathbb{P})$. Let $(X_t)_{t\in[0,T]}$, be a diffusion process, strong solution of the SDE

$$\begin{cases} dX_t = b\,(t, X_t)\,dt + \sigma(t, X_t)dW_t, \\ X_0 \in \mathbb{R} \text{ given.} \end{cases} \tag{3.1}$$

A (vanilla) European contract on X is defined by its maturity T and its payoff $\mathbb{E}[f(T, X_T)]$; typically, for a put of strike price κ and interest rate r, $f(t,x) = e^{r(T-t)}(\kappa - x)^+$, where $\varphi^+(x) = \max\{\varphi(x), 0\}$. An American style contract allows the owner to claim the payoff $f(t, X_t)$ at any time $\tau \in [0, T]$. So a rational strategy to maximize the average profit $\mathcal{V}$ at time t is to find the $[t, T]$-valued $\mathscr{F}$-stopping time τ_t solution of the *Snell envelope* problem:

$$\begin{aligned} \mathcal{V}(t, X_t) &:= \mathbb{E}[e^{-r(\tau_t - t)} f(\tau_t, X_{\tau_t})|\mathscr{F}_t] \\ &= \mathbb{P}\text{-ess sup}_{\tau\in\mathcal{T}_t^{\mathscr{F}}} \mathbb{E}[e^{-r(\tau-t)} f(\tau, X_\tau)|\mathscr{F}_t] \end{aligned} \tag{3.2}$$

where $\mathscr{F} = (\mathscr{F}_t)_{t\in(0,T)}$ is the (augmented) filtration of W and $\mathcal{T}_t^{\mathscr{F}}$ denotes the set of $[t, T]$-valued $\mathscr{F}$-stopping times. Such an optimal stopping times exists (see Caverhill and Webber [41]).We do not specify b, σ or f to stay in a general Optimal Stopping framework. In practice American style options are replaced by so-called *Bermuda* options where the exercise instants are restricted to a time grid $t_k = kh$, $k = 0 : K$ where $h = \frac{T}{K}$ ($K \in \mathbb{N}^*$). Owing to the Markov property of $\{X_{t_k}\}_{k=0}^K$, the corresponding Snell envelope reads $(V(t_k, X_{t_k}))_{k=0:K}$ and satisfies a Backward Dynamic Programming (BDP) recursion on $k = K - 1 : 0$:

$$\begin{cases} V(t_K, X_{t_K}) = f(T, X_T), \\ V(t_k, X_{t_k}) = \max\{f(t_k, X_{t_k}), e^{-rh}\mathbb{E}[V(t_{k+1}, X_{t_{k+1}})|X_{t_k}]\}. \end{cases} \tag{3.3}$$

The optimal stopping times τ_k (from time t_k) are given by a similar backward recursion: $k = K - 1 : 0$,

$$\begin{cases} \tau_K = T, \\ \tau_k = t_k \text{ if } f(t_k, X_{t_k}) > e^{-rh}\mathbb{E}[V(t_{k+1}, X_{t_{k+1}})|X_{t_k}], \\ \tau_k = \tau_{k+1} \text{ otherwise.} \end{cases} \tag{3.4}$$

When $(X_{t_k})_{k=0:K}$ cannot be estimated at a reasonable computational cost, it can be approximated by the Euler scheme with step h, denoted $(\bar{X}^h_{t_k})_{k=0:K}$, which is a simulatable Markov chain recursively defined by

$$\begin{cases} \bar{X}^h_{t_{k+1}} = \bar{X}^h_{t_k} + b(t_k, \bar{X}^h_{t_k})h + \sigma(t_k, \bar{X}^h_{t_k})\Delta W_{k+1}, \\ \bar{X}^h_0 = X_0, \ k = 0 : K - 1, \end{cases} \tag{3.5}$$

where $\Delta W_{k+1} := W_{t_{k+1}} - W_{t_k} = \sqrt{h}\, Z_{k+1}$ so that $\{Z_k\}_{k=1}^K$ are i.i.d. $\mathcal{N}(0, 1)$-distributed random variables. From now on we switch to the Euler scheme and its Snell envelope computed by the following

In the Longstaff-Schwartz algorithm for each k, $\mathbb{E}[V(t_{k+1}, \bar{X}^h_{t_{k+1}})|\bar{X}^h_{t_k}]$ as a function of $x = \bar{X}^h_{t_k}$, is approximated by its L^2-projection on the linear space spanned by the monomials $\{x^p\}_{p=0}^P$; more precisely, let $\mathfrak{P}$ denotes this projection operator:

$$\mathfrak{P}\mathbb{E}[V(t_{k+1}, \bar{X}^h_{t_{k+1}})|\bar{X}^h_{t_k}] = \sum_{p=0}^{P} \bar{a}^p_k \left(\bar{X}^h_{t_k}\right)^p$$

Algorithm 4 BDP-Algorithm

1: Starting from $\bar{X}_0^h$,
2: **for** $k = 0 : K-1$ **do**
3: Compute $\bar{X}_{t_k}^h$ by (1.4).
4: **end for**
5: Evaluate $V(t_K, \bar{X}_{t_K}^h) = f(T, X_T)$;
6: **for** $k = K-1 : 0$ **do**
7: Compute $V(t_k, \bar{X}_{t_k}^h) = \max\{f(t_k, \bar{X}_{t_k}^h), e^{-rh}\mathbb{E}[V(t_{k+1}, \bar{X}_{t_{k+1}}^h)|\bar{X}_{t_k}^h]\}$.
8: **end for**

where

$$\{\bar{a}_k^o, \dots, \bar{a}_k^P\} = \arg \min_{\{(a_k^0,\dots,a_k^P)\in\mathbb{R}^{P+1}\}} \mathbb{E}\Big(\mathbb{E}[V(t_{k+1}, \bar{X}_{t_{k+1}}^h)|\bar{X}_{t_k}^h] - \sum_{p=0}^{P} a_k^p \left(\bar{X}_{t_k}^h\right)^p\Big)^2. \tag{3.6}$$

In practice a Monte-Carlo method is used, hence to generate $\{\bar{X}_{t_{k+1}}^{h,(m)})\}_{m=1}^M$ using (1.4) and then,

$$\{\bar{a}_k^o, \dots, \bar{a}_k^P\} = \arg \min_{\{(a_k^0,\dots,a_k^P)\in\mathbb{R}^{P+1}\}} \sum_{m=1}^{M} \left(V_{k+1}^{(m)} - \sum_{p=0}^{P} a_k^p \left(\bar{X}_{t_{k+1}}^{h,(m)}\right)^p\right)^2. \tag{3.7}$$

and $V_{k+1}^{(m)} = V\big(t_{k+1}, \bar{X}_{t_{k+1}}^{h,(m)}\big)$.

Then each path has its own optimal stopping time $\tau_k^{(m)}$ at each $k \in \{0, :, K-1\}$ computed backward in time from its predecessor by

$$\begin{cases} \tau_K^{(m)} = T, \\ \tau_k^{(m)} = t_k \text{ if } f(t_k, \bar{X}_{t_{k+1}}^{h,(m)}) > e^{-rh} \sum_{p=0}^{P} \bar{a}_k^p \ (\bar{X}_{t_{k+1}}^{h,(m)})^p, \\ \tau_k^{(m)} = \tau_{k+1}^{(m)} \text{ otherwise.} \end{cases} \tag{3.8}$$

Finally the price of the American contract is

$$V(0, X_0) = \max\{f(0, X_0), \frac{1}{M} \sum_{m=1}^{M} e^{-r\tau_1^{(m)}} f(\tau_1^{(m)}, \bar{X}_{\tau_1^{(m)}}^h)\}. \tag{3.9}$$

We summarize the algorithm below:

Remark 14. *Note the computing unit which performs (3.7) requires access to all the paths values $X_{tk}^{(m)}$. It cannot be parallelized efficiently as such.*

Algorithm 5 LSMC-algorithm

Starting from $\bar{X}_0^h$;
for $m = 1 : M$ **do**
  for $k = 0 : K - 1$ **do**
    Compute $\bar{X}_{t_{k+1}}^{h,(m)}$ by (1.4).
  end for
  Evaluate $V(t_K, \bar{X}_{t_K}^{h,(m)}) = f(T, \bar{X}_{t_K}^{h,(m)})$;
end for
for $k = K - 1 : 0$ **do**
  Compute $\{\bar{a}_k^o, \ldots, \bar{a}_k^P\}$ by (3.7)
  for $m = 1 : M$ **do**
    Set $V(t_k, \bar{X}_{t_{k+1}}^{h,(m)}) = \max\{f(t_k, \bar{X}_{t_{k+1}}^{h,(m)}), e^{-rh} \sum_0^P \bar{a}_k^p \, (\bar{X}_{t_{k+1}}^{h,(m)})^p\}$
  end for
end for
Define $V(0, X_0)$ by (3.9).

3.3 A Two Level Parareal Algorithm

3.3.1 The Parareal Method for ODE

The method was introduced by J.L. Lions, Y. Maday and G. Turinici [122] to improve parallel computations of time dependent partial differential equations. M. Gander [56] established the connection between parareal and multi-level / multigrid methods and G. Bal and Y. Maday[10] studied the method for SDE. The name come from a contraction of parallelism and real time systems.

3.3.1.1 Two Levels Parareal

Consider an ODE

$$\dot{x} = f(x,t),\ x(0) = x_0,\ t \in [t_0, t_K] = \cup_0^{K-1}[t_k, t_{k+1}],$$

(keeping in mind that $t_k = k\frac{T}{K}$, $k = 0 : K$).

Assume that $G_\delta(x_k, t_k)$ is a high precision integrator which computes x at t_{k+1} from x_k at t_k. Assume G_Δ is a similar integrator but of low precision. The parareal algorithm is an iterative process with $n = 0 : N - 1$ above a forward loop in time, $k = 0 : K - 1$

$$x_{k+1}^{n+1} = G_\Delta(x_k^{n+1}, t_k) + G_\delta(x_k^n, t_k) - G_\Delta(x_k^n, t_k). \tag{3.10}$$

So the coarse grid solution is corrected by the difference between the fine grid prediction computed from the old value on that interval and the coarse grid old solution. On figure 3.1, a schematic representation of the two level parareal algorithm.

In this analysis G_δ and G_Δ are Euler explicit schemes with time step $\delta t = \frac{\Delta t}{J}\frac{T}{KJ}$, $J \in \mathbb{N}^*$, respectively:

Initialization: From $\hat{x}_0^0 = x_0$, compute on the coarse mesh:

$$\hat{x}_{t_{k+1}}^0 = \hat{x}_{t_k}^0 + f(t_k, \hat{x}_{t_k}^0)\Delta t,\ k = 0 : K-1$$

for n=0: N-1 (parareal iterations)

for k=0: K-1

1. Compute the fine grid solution $\{\tilde{x}_{t_{k,j}}^n\}_{j=0}^J$ of the ODE on (t_k, t_{k+1}), started at $t_{k,0} = t_k$ with $\tilde{x}_{t_k,0}^n = \hat{x}_{t_k}^n$:

$$\tilde{x}_{t_k,j+1}^n = \tilde{x}_{t_k,j}^n + f(t_k, \tilde{x}_{t_k,j}^n)\delta t,\ j = 0 : J-1 \tag{3.11}$$

2. Compute the coarse grid solution at t_{k+1} by one Euler step:
$\bar{x}_{t_{k+1}}^\Delta = \hat{x}_{t_k}^{n+1} + f(t_k, \hat{x}_{t_k}^{n+1})\Delta t$.
3. Set $\hat{x}_{t_{k+1}}^{n+1} = \bar{x}_{t_{k+1}}^\Delta + \tilde{x}_{t_{k,J}}^n - \hat{x}_{t_{k+1}}^n$.

end k-loop

end n-loop .

Convergence rates have been proved for linear systems of ODE when the fine integrator is exact Gander and Vandewalle [56], i.e. when (3.11) is replaced by an exact solver, giving for $\tilde{x}_{t_{k;J}}^n$ the exact solution of the ODE started at $t_{k,0} = t_k$ with $\tilde{x}_{t_k,0}^n = \hat{x}_{t_k}^n$. Then, if ΔT is not too large,

$$\max_{k=0\cdot K} |x(t_k) - \hat{x}_{t_k}^n| \leq \frac{(C\Delta T)^n}{n!} \max_{k=0\cdot K} |x(t_k) - \hat{x}_{t_k}^0| \tag{3.12}$$

3.3.1.2 Multi Levels Parareal

Evidently the process is recursive and one can replace $G_\delta(x_k, t_k)$ by an operator which is the result of a number of parareal steps on each interval $[t_{k,j}, t_{k,j+1}]$.

The estimate (3.12) can also be used recursively and shows that multi-level parareal algorithm converges superlinearly.

3.3.2 The Parareal Method for SDE

The same method can be applied to an SDE in the context of the Monte-Carlo method provided the random variables defining ΔW_k in (1.4) are sampled once and for all in the initial phase of the algorithm and reused for all n.

3.3.2.1 Notations and Algorithm

If Y is a variable computed by a parareal algorithm we shall denote

- $\bar{Y}$ values computed on the coarse grid with time step Δt,
- $\tilde{Y}$ values computed on the fine grid with time step $\delta t = \frac{\Delta t}{J}$,
- $\hat{Y}$ the final result.

Let K is the number of coarse time intervals and J the number of fine time intervals, equal for of each coarse time interval.

Let $[t_k, t_{k+1}] = \cup_{j=0}^{J-1}[t_{k,j}, t_{k,j+1}]$ with $t_{k,j+1} = t_{k,j} + \delta t$, $t_k = t_{k,0} = t_{k-1,J}$.

Let $\mathbf{Z}_k = \{Z_{k,j}\}_{j=1}^{J}$ a J-dimensional white noise with an $\mathcal{N}(0, I_J)$ distribution and $Z_{k+1} = \frac{1}{\sqrt{J}}\sum_{j=1}^{J} Z_{k,j} = W_{t_{k+1}} - W_{t_k}$.

Algorithm 6 TLPR: Two levels Parareal for SDE

1: **Initialization**:

Solve recursively forward from $\hat{X}_0^0 = X_0$, on the coarse mesh:

$\hat{X}_{t_{k+1}}^0 = \hat{X}_{t_k}^0 + b(t_k, \hat{X}_{t_k}^0)\Delta t + \sigma(t_k, \hat{X}_{t_k}^0)\sqrt{\Delta t} Z_{k+1}$.

Set $\bar{X}_{t_k}^{0,\Delta} = \hat{X}_{t_k}^0$, $k = 1 : K$.

2: **for** $n = 0 : N-1$ **do** (parareal iterations):

3: **for** $k = 0 : K-1$ **do** (forward time loop):

4: Compute the fine grid solution $\{\tilde{X}_{t_{k,j}}^n\}_{j=1}^{J}$ of the SDE on (t_k, t_{k+1}), started at $t_{k,0} = t_k$ with $\tilde{X}_{t_{k,0}}^n = \hat{X}_{t_k}^n$:

for j=0:J-1 $\tilde{X}_{t_{k,j+1}}^n = \tilde{X}_{t_{k,j}}^n + b(t_k, \tilde{X}_{t_{k,j}}^n)\delta t + \sigma(t_k, \tilde{X}_{t_{k,j}}^n)\sqrt{\delta t} Z_{k,j+1}$.

5: Compute the coarse grid solution at t_{k+1} by (one Euler step)

$\bar{X}_{t_{k+1}}^{n+1,\Delta} = \hat{X}_{t_k}^{n+1} + b(t_k, \hat{X}_{t_k}^{n+1})\Delta t + \sigma(t_k, \hat{X}_{t_k}^{n+1})\sqrt{\Delta t} Z_{k+1}$.

6: Set $\hat{X}_{t_{k+1}}^{n+1} = \bar{X}_{t_{k+1}}^{n+1,\Delta} + \tilde{X}_{t_{k,J}}^n - \bar{X}_{t_{k+1}}^{n,\Delta}$.

7: **end for**

8: **end for**

In other words the iterative process (3.10) is applied on each sample path with G_Δ a single step of (1.4) with $h = \Delta t$ and G_δ the result of J steps of (1.4) with $h = \delta t$. An error analysis is available in Bal and Maday [10] for the stochastic case in the limit case $\delta t = 0$, i.e. when the fine integrator is the exact solution. We extend the result to the case $0 < \delta t < \Delta t$ with a bound between the TLPR solution and the standard solution of the Euler explicit scheme (1.4) with step $h = \delta t$. It will be usually be denoted $(\bar{X}_{t_{k,j}}^{\delta})_{k=0:K, j=0:J}$ in what follows (however we will denote $\bar{X}_{t_k}^{\delta} = \bar{X}_{t_{k,j}}^{\delta}$ since $t_{k,0} = t_k$)

Theorem 9. *Let $(\hat{X}_{t_k}^n)_{k=0}^K$ be the TLPR solution after n iterations; let $\bar{X}_{t_k}^{\Delta}$, $\bar{X}_{t_k}^{\delta}$ be the solutions at time t_k of (1.4) with $h = \Delta t$ and $h = \delta t$ respectively. Assume $b, \sigma : [0, T] \times \mathbb{R}$ as well as their*

first two spatial derivatives are Lipschitz in (t,x)*. Then there exists* C*, independent of* $k, \Delta t$ *and* n*, such that for* $k = 0:K$*,* $n = 0:N$*,*

$$\|\hat{X}^n_{t_k} - \bar{X}^\delta_{t_k}\|_{L^2(\mathbb{P})} \le (C\Delta t)^n \sqrt{\binom{k}{n}} \max_{k=1:K} \|\bar{X}^\Delta_{t_k} - \bar{X}^\delta_{t_k}\|_{L^2(\mathbb{P})} \le (C\Delta t)^{n+\frac{1}{2}} \sqrt{\binom{k}{n}} \tag{3.13}$$

and $\hat{X}^n_{t_k} = \bar{X}^\delta_{t_k}$ *for all* $k \le n$*.*

3.3.2.1.1 Proof For the sake of simplicity we detail the proof in the homogeneous case $b(t,x) \equiv b(x)$ and $\sigma(t,x) = \sigma(x)$ and we assume that b and σ are $\mathbb{C}^2$ and $b^{(m)}, \sigma^{(m)},\ m = 0,1,2$ Lipschitz.

Step 1 (*Toward a discrete time Gronwall inequality*). Consider the Euler scheme with coarse time steps, started at $t_0 = 0$ by X_0:

$$\bar{X}^\Delta_{t_{k+1}} = \bar{X}^\Delta_{t_k} + \Delta t\, b(\bar{X}^\Delta_{t_k}) + \sigma(\bar{X}^\Delta_{t_k})\sqrt{\Delta t} Z_{k+1}, \quad k = 0:K-1. \tag{3.14}$$

Consider also the Euler schemes with fine time steps, started at t_0 by X_0:

$$\bar{X}^\delta_{t_{k,j+1}} = \bar{X}^\delta_{t_{k,j}} + \delta t\, b(\bar{X}^\delta_{t_{k,j}}) + \sigma(\bar{X}^\delta_{t_{k,j}})\sqrt{\delta t} Z_{k,j+1},\ j = 0:J-1,\ k = 1:K. \tag{3.15}$$

The parareal scheme starts with $\hat{X}^0_{t_k} = \bar{X}^\Delta_{t_k}, \quad k = 1:K$, and sets

$$\hat{X}^{n+1}_{t_{k+1}} = G_\Delta\left(\hat{X}^{n+1}_{t_k}, Z_{k+1}\right) + G^{(J)}_\delta\left(\hat{X}^n_{t_k}, \mathbf{Z}_{k+1}\right) - G_\Delta\left(\hat{X}^n_{t_k}, Z_{k+1}\right),\ k = 0:K-1,$$

where $G_\Delta(x,z) = x + b(x)\Delta t + \sigma(x)\sqrt{\Delta t}\, z$, $G^{(0)}_\delta(x,z_1) = x + b(x)\delta t + \sigma(x)\sqrt{\delta t}\, z_1$ and

$$G^{(j+1)}_\delta(x,(z_1,\dots z_{j+1})) = x + b\left(G^{(j)}_\delta\left(x,(z_0,\dots z_j)\right)\right)\delta t + \sigma\left(G^{(j)}_\delta\left(x,(z_1,\dots z_j)\right)\right)\sqrt{\delta t}\, z_{j+1}.$$

We have

$$\begin{aligned} \hat{X}^{n+1}_{t_{k+1}} - \quad \bar{X}^\delta_{t_{k+1}} &= G_\Delta(\hat{X}^{n+1}_{t_k}, Z_{k+1}) - G_\Delta(\bar{X}^\delta_{t_k}, Z_{k+1}) + G^{(J)}_\delta(\hat{X}^n_{t_k}, \mathbf{Z}_{k+1}) - G^{(J)}_\delta(\bar{X}^\delta_{t_k}, \mathbf{Z}_{k+1}) \\ &= G_\Delta(\hat{X}^{n+1}_{t_k}, Z_{k+1}) - G_\Delta(\bar{X}^\delta_{t_k}, Z_{k+1}) + \phi_{\Delta,\delta}(\hat{X}^n_{t_k}, \mathbf{Z}_{k+1}) - \phi_{\Delta,\delta}(\bar{X}^\delta_{t_k}, \mathbf{Z}_{k+1}) \end{aligned} \tag{3.16}$$

with

$$\phi_{\Delta,\delta}(x, \mathbf{Z}_{k+1}) = G^{(J)}_\delta(x, \mathbf{Z}_{k+1}) - G_\Delta(x, Z_{k+1}) = \bar{X}^{\delta,x}_{\Delta t} - \bar{X}^{\Delta,x}_{\Delta t} \tag{3.17}$$

where $\bar{X}^{\delta,x}_{\Delta t}$ denotes the solution at $t + \Delta t$ of the δ-Euler scheme starting from x at t, and similarly with $\bar{X}^\Delta$. Note that

$$\bar{X}^{\delta,x}_{\Delta t} = x + \int_0^{\Delta t} b(\bar{X}^{\delta,x}_{\underline{s}_\delta})ds + \int_0^{\Delta t} \sigma(\bar{X}^{\delta,x}_{\underline{s}_\delta})dW_s$$

where $\bar{X}^{\delta,x}_{\underline{t}_\delta}$ is the continuous time Euler scheme solution of

$$d\bar{X}^\delta_t = b(\bar{X}^\delta_{\underline{t}_\delta})dt + \sigma(\bar{X}^\delta_{\underline{t}_\delta})dW_t, \quad \underline{t}_\delta = \lfloor \frac{t}{\delta t} \rfloor \delta t, \quad \bar{X}^\delta_0 = x. \tag{3.18}$$

Our aim is to establish an induction property for $\|\hat{X}^{n+1}_{t_{k+1}} - \bar{X}^{\delta}_{t_{k+1}}\|^2_{L^2(\mathbb{P})}$. To this end we first deal with the last two terms of (3.16). First we note that

$$\begin{aligned}
\phi_{\triangle,\delta}(x,\mathbf{Z}_{k+1}) \quad & - \phi_{\triangle,\delta}(y,\mathbf{Z}_{k+1}) = \bar{X}^{\delta,x}_{\Delta t} - \bar{X}^{\triangle,x}_{\Delta t} - (\bar{X}^{\delta,y}_{\Delta t} - \bar{X}^{\triangle,y}_{\Delta t}) \\
&= -\big(b(x)\Delta t + \sigma(x)\Delta W\big) + \int_0^{\Delta t} b(\bar{X}^{\delta,x}_{\underline{s}_\delta})ds + \int_0^{\Delta t} \sigma(\bar{X}^{\delta,x}_{\underline{s}_\delta})dW_s \\
&\quad + \big(b(y)\Delta t + \sigma(y)\Delta W\big) - \int_0^{\Delta t} b(\bar{X}^{\delta,y}_{\underline{s}_\delta})ds - \int_0^{\Delta t} \sigma(\bar{X}^{\delta,y}_{\underline{s}_\delta})dW_s \\
&= -\int_0^{\Delta t} \Big(b(x) - b(y) - \big(b(\bar{X}^{\delta,x}_{\underline{s}_\delta} - b(\bar{X}^{\delta,y}_{\underline{s}_\delta})\big)\Big)ds \\
&\quad - \int_0^{\Delta t} \Big(\sigma(x) - \sigma(y) - \big(\sigma(\bar{X}^{\delta,x}_{\underline{s}_\delta} - \sigma(\bar{X}^{\delta,y}_{\underline{s}_\delta})\big)\Big)dW_s. \qquad (3.19)
\end{aligned}$$

The first integral will be called A. The last integral, B, can be bounded as follows

$$\mathbb{E}[B^2] = \int_0^{\Delta t} \mathbb{E}\Big[\Big(\sigma(x) - \sigma(y) - \big(\sigma(\bar{X}^{\delta,x}_{\underline{s}_\delta} - \sigma(\bar{X}^{\delta,y}_{\underline{s}_\delta})\big)\Big)^2\Big]ds. \qquad (3.20)$$

Applying Itô's formula to $\bar{X}^{\delta,x}$ and σ yields for any $s \in [0,\Delta t]$

$$\begin{aligned}
\sigma(\bar{X}^{\delta,x}_s) \quad &= \sigma(x) + \int_0^s \sigma'(\bar{X}^{\delta,x}_u)\sigma(\bar{X}^{\delta,x}_{\underline{u}_\delta})dW_u \\
&\quad + \int_0^s \Big(\sigma'(\bar{X}^{\delta,x}_u)b(X^{\delta,x}_{\underline{u}_\delta}) + \frac{1}{2}\sigma"(X^{\delta,x}_u)\sigma^2(X^{\delta,x}_{\underline{u}_\delta})\Big)du. \qquad (3.21)
\end{aligned}$$

The same holds with y instead of x and so

$$\sigma(\bar{X}^{\delta,x}_s) - \sigma(\bar{X}^{\delta,y}_s) - (\sigma(x) - \sigma(y)) = \int_0^s S^{x,y}_u du + \int_0^s H^{x,y}_u dW_u \qquad (3.22)$$

with

$$\begin{aligned}
S^{x,y}_u &= \frac{1}{2}\Big[\sigma''(X^{\delta,x}_u)\sigma^2(X^{\delta,x}_{\underline{u}_\delta}) - \sigma''(X^{\delta,y}_u)\sigma^2(X^{\delta,y}_{\underline{u}_\delta})\Big] + \sigma'(\bar{X}^{\delta,x}_u)b(X^{\delta,x}_{\underline{u}_\delta}) - \sigma'(\bar{X}^{\delta,y}_u)b(X^{\delta,y}_{\underline{u}_\delta}) \\
H^{x,y}_u &= \sigma'(X^{\delta,x}_u)\sigma(X^{\delta,x}_{\underline{u}_\delta}) - \sigma'(X^{\delta,y}_u)\sigma(X^{\delta,y}_{\underline{u}_\delta}). \qquad (3.23)
\end{aligned}$$

Hence, denoting $\|Y\|_2 := \|Y\|_{L^2(\mathbb{P})}$ and using the general Minkowski and Doob inequalities, leads to

$$\begin{aligned}
\|\sigma(\bar{X}^{\delta,x}_s) - \sigma(\bar{X}^{\delta,y}_s) - (\sigma(x) - \sigma(y))\|_2 \quad &\le \quad \int_0^s \|S^{x,y}_u\|_2 du + \Big\| \int_0^s H^{x,y}_u dW_u \Big\|_2 \\
&\le \quad \int_0^s \|S^{x,y}_u\|_2 du + \Big[\mathbb{E}[\int_0^s (H^{x,y}_u)^2 du]\Big]^{\frac{1}{2}} \\
&\le \quad \int_0^s \|S^{x,y}_u\|_2 du + \Big[\int_0^s (\|H^{x,y}_u)\|_2^2 du]\Big]^{\frac{1}{2}}. \qquad (3.24)
\end{aligned}$$

Now σ' bounded and σ Lipschitz leads to

$$\|H^{x,y}_u\|_2 \le \|\sigma'\|_\infty [\sigma]_{Lip} \|\bar{X}^{\delta,x}_{\underline{u}_\delta} - \bar{X}^{\delta,y}_{\underline{u}_\delta}\|_2 + \|\sigma\|_\infty [\sigma']_{Lip}) \|\bar{X}^{\delta,x}_u - \bar{X}^{\delta,y}_u\|_2. \qquad (3.25)$$

As b and σ are Lipschitz continuous, a classical result (see *e.g.* Pagès [140]) on the Euler scheme says that for all $v \in [0,T]$, uniformly in δ,

$$\|\bar{X}^{\delta,x}_v - \bar{X}^{\delta,y}_v\|_2 \le \sup_{t\in[0,T]} \|\bar{X}^{\delta,x}_t - \bar{X}^{\delta,y}_t\|_2 \le C_{b,\sigma}|x-y|. \qquad (3.26)$$

Consequently $\sup_{u\in[0,T]}\|H_u^{x,y}\|_2 \le C_{b,\sigma,\sigma'}|x-y|$. As for $S_u^{x,y}$, assuming σ'' Lipschitz, a similar computation leads to $\sup_{u\in[0,T]}\|S_u^{x,y}\|_2 \le C_{b,\sigma,\sigma',\sigma''}|x-y|$. Note by the way that all these terms vanish if σ is constant.

Plugging these bounds in (3.24) leads to

$$\begin{aligned}\|\sigma(\bar X_s^{\delta,x}) &- \sigma(\bar X_s^{\delta,y}) - (\sigma(x)-\sigma(y))\|_2 \\ &\le \max\{C_{b,\sigma,\sigma'}, C_{b,\sigma,\sigma',\sigma''}\}(\sqrt{s}+s)|x-y] \le \widetilde{C}\sqrt{s}|x-y| \end{aligned} \tag{3.27}$$

which implies, in turn,

$$\mathbb{E}[B]^2 \le \int_0^{\Delta t} \widetilde{C}^2 s|x-y|^2 ds = \frac{1}{2}\widetilde{C}^2(\Delta t)^2|x-y|^2. \tag{3.28}$$

The term A in (3.19) can be treated likewise:

$$\begin{aligned} b(\bar X_s^{\delta,x}) - b(x) &= \int_0^s b'(\bar X_u^{\delta,x})\sigma(\bar X_{\underline{u}_\delta}^{\delta,x})dW_u \\ &+ \int_0^s \Big[b'(\bar X_u^{\delta,x})b(\bar X_{\underline{u}_\delta}^{\delta,x}) + \frac{1}{2}b''(\bar X_u^{\delta,x})\sigma^2((\bar X_{\underline{u}_\delta}^{\delta,x})\Big]du. \end{aligned} \tag{3.29}$$

In the end, we note that

$$\mathbb{E}[A^2] \le \Delta t \int_0^{\Delta t} \big(\mathbb{E}[b(x)-b(y)-[b(\bar X_{\underline{u}_\delta}^{\delta,x}) - b(\bar X_{\underline{u}_\delta}^{\delta,y})]\big)^2 du$$

owing to Schwartz Inequality. Then, as b'' is Lipschitz continuous, we obtain like for the similar term involving σ that

$$\int_0^{\Delta t} \big(\mathbb{E}[b(x)-b(y)-[b(\bar X_{\underline{u}_\delta}^{\delta,x}) - b(\bar X_{\underline{u}_\delta}^{\delta,y})]\big)^2 du \le \frac{1}{2}\widetilde{C}_{b,b',b'',\sigma}(\Delta t)^2|x-y|^2.$$

so that

$$\mathbb{E}[A^2] \le \frac{1}{2}\widetilde{C}_{b,b',b'',\sigma}(\Delta t)^3|x-y|^2.$$

So we have proved that

$$\|\phi_{\Delta,\delta}(x,\mathbf{Z}_{k+1}) - \phi_{\Delta,\delta}(y,\mathbf{Z}_{k+1})\|_2^2 \le C(\Delta t)^2|x-y|^2. \tag{3.30}$$

Let us bound the two other terms in (3.16). First using that $Z=0$ and $\mathbb{E}\,Z^2 2 = 1$, one easily derives that

$$\begin{aligned}\|G_\Delta(x,Z) - G_\Delta(y,Z)\|_2^2 &\le \Big(1 + 2\Delta t[b]_{Lip} + \Delta t[\sigma]_{Lip}^2 + \Delta t^2[b]_{Lip}^2\Big)|x-y|^2 \\ &\le \big(1 + C'_{b,\sigma,T}\Delta t\big)^2|x-y|^2 \end{aligned} \tag{3.31}$$

where $C'_{b,\sigma,T} = [b]_{Lip} + \frac{1}{2}([\sigma]_{Lip}^2 + T[b]_{Lip}^2)$. As one must raise (3.16) to the square, a cross term appears,

$$D := \mathbb{E}\Big[\Big(G_\Delta(\hat X_{t_k}^{n+1}, Z_{k+1}) - G_\Delta(\bar X_{t_k}^{\delta}, Z_{k+1})\Big)\Big(\phi_{\Delta,\delta}(\hat X_{t_k}^{n}, \mathbf{Z}_{k+1}) - \phi_{\Delta,\delta}(\bar X_{t_k}^{\delta}, \mathbf{Z}_{k+1})\Big)\Big]$$

As Z_{k+1} and $\mathbb{Z}_{k+1}$ are independent of $(\hat X_{t_k}^{n+1}, \hat X_{t_k}^{n}, \bar X_{t_k}^{\delta})$, it follows that

$$D = \mathbb{E}\,\Phi(\hat X_{t_k}^{n+1}, \hat X_{t_k}^{n}, \bar X_{t_k}^{\delta})$$

where

$$\Phi(x,x',y) := \mathbb{E}\Big[\Big((x-y)+\Delta t(b(x)-b(y))+(\sigma(x)-\sigma(y))\Delta W\Big)\Big(A'+B'\Big)\Big]$$

with
$$A' = \int_0^{\Delta t}\Big(b(x')-b(y)-\big(b(\bar X^{\delta,x'}_{\underline{s}_\delta})-b(\bar X^{\delta,y}_{\underline{s}_\delta})\big)\Big)ds$$
$$B' = \int_0^{\Delta t}\Big(\sigma(x')-\sigma(y)-\big(\sigma(\bar X^{\delta,x'}_{\underline{s}_\delta})-\sigma(\bar X^{\delta,y}_{\underline{s}_\delta})\big)\Big)dW_s$$

(these are the last two integrals from (3.19) except that x is changed to x').

Note that $\mathbb{E}[B']=0$ because it is a stochastic integral, hence

$$D = [(x-y)+\Delta t(b(x)-b(y))]\mathbb{E}[A'] + (\sigma(x)-\sigma(y))\mathbb{E}[(A'+B')\Delta W].$$

Now, using Schwartz' inequality,

$$\begin{aligned}|\mathbb{E}[D]| &\le (1+\Delta t[b]_{Lip})\|A'\|_2|x-y| + [\sigma]_{Lip}|x-y|\|\Delta W\|_2\|A'+B'\|_2\\ &\le (1+\Delta t[b]_{Lip})\|A'\|_2|x-y| + [\sigma]_{Lip}|x-y|\sqrt{\Delta t}\big(\|A'\|_2+\|B'\|_2\big).\end{aligned} \tag{3.32}$$

We recall our previous bounds on A and B,

$$\|B'\|_2 \le \tilde C\Delta t|x'-y|, \quad \|A'\|_2 \le \tilde C(\Delta t)^{\frac{3}{2}}|x'-y|. \tag{3.33}$$

Consequently, with $2\bar C = \tilde C(1+\Delta t[b]_{Lip}+[\sigma]_{Lip})$,

$$|\mathbb{E}[D]| \le 2\bar C(\Delta t)^{\frac{3}{2}}|x'-y||x-y|. \tag{3.34}$$

We are now in a position to patch the pieces together; C denotes a generic constant:

$$\begin{aligned}&\mathbb{E}\Big[\Big(G_\Delta(\hat X^{n+1}_{t_k},Z_{k+1})-G_\Delta(\bar X^\delta_{t_k},Z_{k+1})+\phi_{\triangle,\delta}(\hat X^n_{t_k},\mathbf{Z}_{k+1})-\phi_{\triangle,\delta}(\bar X^\delta_{t_k},\mathbf{Z}_{k+1})\Big)^2\Big]\\ &= \mathbb{E}\Big[\Big(G_\Delta(\hat X^{n+1}_{t_k},Z_{k+1})-G_\Delta(\bar X^\delta_{t_k},Z_{k+1})\Big)^2\Big] + \mathbb{E}\Big[\Big(\phi_{\triangle,\delta}(\hat X^n_{t_k},\mathbf{Z}_{k+1})-\phi_{\triangle,\delta}(\bar X^\delta_{t_k},\mathbf{Z}_{k+1})\Big)^2\Big]\\ &\quad +2\mathbb{E}\Big[\Big(G_\Delta(\hat X^{n+1}_{t_k},Z_{k+1})-G_\Delta(\bar X^\delta_{t_k},Z_{k+1})\Big)\Big(\phi_{\triangle,\delta}(\hat X^n_{t_k},\mathbf{Z}_{k+1})-\phi_{\triangle,\delta}(\bar X^\delta_{t_k},\mathbf{Z}_{k+1})\Big)\Big]\\ &\le (1+C\Delta t)^2|x-y|^2 + C\Delta t^2|x'-y|^2 + 2C(\Delta t)^{\frac{3}{2}}|x-y||x'-y|\\ &\le (1+C\Delta t)^2|x-y|^2 + C\Delta t^2|x'-y|^2,\end{aligned} \tag{3.35}$$

since $2(\Delta t)^{\frac{3}{2}}|x-y||x'-y| \le \Delta t|x-y|^2 + \Delta t^2|x'-y|^2$.

As $x=\hat X^{n+1}_{t_k}, x'=\hat X^n_{t_k}, y=\bar X^\delta_{t_k}$ are independent of $\mathbf{Z}_{k+1}$, integrating (3.35) with respect to the distribution of the triplet $\mathbb{P}_{(\hat X^{n+1}_{t_k},\hat X^n_{t_k},\bar X^\delta_{t_k})}(dx,dx',dy)$ yields by Fubini's theorem

$$\|\hat X^{n+1}_{t_{k+1}} - \bar X^\delta_{t_{k+1}}\|_2^2 \le (1+C'\Delta t)\|\hat X^{n+1}_{t_k} - \bar X^\delta_{t_k}\|_2^2 + C'\Delta t^2\|\hat X^n_{t_k} - \bar X^\delta_{t_k}\|_2^2. \tag{3.36}$$

With $\epsilon^n_k := \|\hat X^n_{t_k} - \bar X^\delta_{t_k}\|_2^2$, it is rewritten as

$$\epsilon^{n+1}_{k+1} \le (1+C'\Delta t)\epsilon^{n+1}_k + C'\Delta t^2\epsilon^n_k. \tag{3.37}$$

STEP 2 (*Solving the discrete time Gronwall inequality*). We will now show that the recurrence (3.37) started with $\epsilon_0^n = 0,\ n = 0 : N$, implies on the one hand

$$\epsilon_k^n \leq \max_{k=1,\dots K}\{\epsilon_k^0\}\ C'^n \binom{k}{n} (\Delta t)^{2n}, \quad k = 0 : K,\ n = 0 : N. \tag{3.38}$$

and on the other hand also that $\epsilon_k^n = 0$ for every $k = 1 : n$.

To this end introduce $\tilde{\epsilon}_k^n = (1+C'\Delta t)^{n-k} C'^{-n} (\Delta t)^{-2n} \epsilon_k^n$. It is easy to check that $\tilde{\epsilon}_k^n$ satisfies, for $k = 1 : K-1$ and $n = 0 : N-1$,

$$\tilde{\epsilon}_{k+1}^{n+1} \leq \tilde{\epsilon}_k^{n+1} + \tilde{\epsilon}_k^n. \tag{3.39}$$

It implies that

$$\tilde{\epsilon}_k^n = 0,\ \forall k \in \{0, \dots, n\}. \tag{3.40}$$

Indeed, if (3.40) is true for n, then by (3.39)

$$\tilde{\epsilon}_{k+1}^{n+1} \leq \tilde{\epsilon}_k^{n+1} \leq \dots \leq \tilde{\epsilon}_0^{n+1} = 0.$$

Which proves that (3.40) is true for $n+1$.

Now if (3.38) holds for k, n, it holds for $k+1, n+1$ because by (3.37) and (3.42)

$$\begin{aligned} \tilde{\epsilon}_{k+1}^{n+1} &\leq \max_{k=1,\dots K}\{\epsilon_k^0\} \left[\binom{k}{n+1} + \binom{k}{n} \right] \\ &= \max_{k=1,\dots K}\{\epsilon_k^0\} \binom{k+1}{n+1}. \end{aligned} \tag{3.41}$$

Finally notice that the error bounds for the Euler scheme,

$$\begin{aligned} \tilde{\epsilon}_k^0 \leq \epsilon_k^0 &= \|\hat{X}_{t_k}^{\Delta} - \bar{X}_{t_k}^{\delta}\|_2^2 \\ &\leq 2\Big(\|\hat{X}_{t_k}^{\Delta} - X_{t_k}\|_2^2 + \|X_{t_k} - \bar{X}_{t_k}^{\delta}\|_2^2\Big) \\ &\leq 2(\sqrt{\Delta t} + \sqrt{\delta t})^2 (1 + |X_0|)^2 C_{b,\sigma,T} \\ &\leq C_1 \Delta t. \end{aligned} \tag{3.42}$$

Corollary 2. *Let $K \in \mathbb{N}$, $K > n$*

$$\max_{k=0:K} \|\hat{X}_{t_k}^n - \bar{X}_{t_k}^{\delta}\|_{L^2(\mathbb{P})} \leq \frac{(C_T \Delta t)^{\frac{n+1}{2}}}{\sqrt{n!}} e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}$$

where C_T depends only on the Lipschitz constants and norms of b, b', b'', $\sigma, \sigma', \sigma''$ and on T.

This estimate shows that, when Δt is smaller than C^{-1}, the method converges exponentially in n and geometrically in Δt.

3.3.2.1.2 Proof Let $K > n$. Keeping in mind that $\Delta t = \frac{T}{K}$, we have

$$\begin{aligned}(\Delta t)^{n+\frac{1}{2}}\sqrt{\binom{K}{n}} &= \left(\frac{T}{K}\right)^{n+\frac{1}{2}}\sqrt{\frac{K!}{n!(K-n)!}} \\ &= \frac{1}{\sqrt{n!}}\left(\frac{T}{K}\right)^{\frac{n+1}{2}}\left(\left(\frac{T}{K}\right)^{n}\frac{K!}{(K-n)!}\right)^{\frac{1}{2}}. \end{aligned} \tag{3.43}$$

As

$$\begin{aligned}K^{-n}K!/(K-n)! &= \frac{K(K-1)\cdots(K-(n-1))}{K^n} \\ &= \prod_{\ell=1}^{n-1}\left(1-\frac{\ell}{k}\right) \\ &\leq e^{-\frac{n(n-1)}{2}\frac{\Delta t}{T}}, \end{aligned} \tag{3.44}$$

since $1 - x \leq e^x$. Then, we derive that

$$(C\Delta t)^{n+\frac{1}{2}}\sqrt{\binom{K}{n}} \leq \frac{C^{n+\frac{1}{2}}T^{\frac{n}{2}}}{\sqrt{n!}}e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}(\Delta t)^{\frac{n+1}{2}} = \frac{\left(C^2T\Delta t\right)^{\frac{n+1}{2}}e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}}{\sqrt{TC\,n!}}.$$

From Theorem 9, since $\binom{k}{n} \leq \binom{K}{n}$ whenever $K \geq k > n$,

$$\begin{aligned}\max_{k=1:K}\|\hat{X}^n_{t_k} - \bar{X}^\delta_{t_k}\|_{L^2(\mathbb{P})} &\leq (C\Delta t)^{n+\frac{1}{2}}\sqrt{\binom{K}{n}} \\ &\leq \frac{\left(C^2T\Delta t\right)^{\frac{n+1}{2}}}{\sqrt{TC\,n!}}e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}. \end{aligned} \tag{3.45}$$

Corollary 3. *Let $K \in \mathbb{N}$, $K > n$.*

$$\Big\|\max_{k=0:K}|\hat{X}^n_{t_k} - \bar{X}^\delta_{t_k}|\Big\|_{L^2(\mathbb{P})} \leq \frac{(C_T\Delta t)^{\frac{n}{2}}}{\sqrt{(n+1)!}}\Big(1+\frac{\Delta t}{T}\Big)^{\frac{1}{2}}e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}$$

where C_T depends only on the Lipschitz constants and norms of b, b', b'', $\sigma, \sigma', \sigma''$ and on T.

3.3.2.1.3 Proof We first note that

$$\begin{aligned}\mathbb{E}\max_{k0=:K}|\hat{X}^n_{t_k} - \bar{X}^\delta_{t_k}|^2 &\leq \sum_{k=n+1}^{K}\mathbb{E}\,|\hat{X}^n_{t_k} - \bar{X}^\delta_{t_k}|^2 \\ &\leq \sum_{k=n+1}^{K}(C\Delta t)^{2n+1}\binom{k}{n} \\ &= (C\Delta t)^{2n+1}\sum_{k=n+1}^{K}\binom{k}{n}. \end{aligned}$$

First, one checks by an easy induction on K that

$$\sum_{k=n+1}^{K}\binom{k}{n} = -1+\sum_{k=n}^{K}\binom{k}{n} = \binom{K+1}{n+1}-1 \leq \binom{K+1}{n+1}. \tag{3.46}$$

Now

$$(\Delta t)^{2n+1}\binom{K+1}{n+1} = \frac{T^{2n+1}}{K^n}\Big(1+\frac{1}{K}\Big)\times 1\times\prod_{\ell=1}^{n-1}\Big(1-\frac{\ell}{K}\Big) \leq T^{n+1}(\Delta t)^n\Big(1+\frac{\Delta t}{T}\Big)e^{-\frac{n(n-1)}{2}\frac{\Delta t}{T}} \tag{3.47}$$

and one concludes as above to define the real constant C_T.

3.4 Multi-level Parareal for SDE

The estimate (3.13) indicates that a recursive use of parareal with each sub-interval redivided into $J = O(\Delta t^{-1})$ smaller intervals, the so-called multilevels parareal is better than many iterations at the second level only.

Indeed, assume for clarity that $J = K$ and that each interval of size $\delta t = T/K^2$ is sub-divided in K intervals of size $\frac{\delta t}{K} = T/K^3$ and so on for each of the L levels ($L = 1$ for the 2-level case). Then as the error decreases proportionally to $(\Delta t)^{\frac{n}{2}}$ at each level after L levels the error is decreased by $(K^{-L})^{\frac{n}{2}}$.

Compared with the 2-level case, the same level of accuracy is attained when $K^{-\frac{n}{2}} = K^{-\frac{Ln'}{2}}$, where n' is the number of parareal iterations at each level for the L-level case and n is the number of iteration for the 2-level method. Hence $n = n'L$ will give comparable precisions.

On the other hand the total number of operations being proportional to $(Kn')^L$ it makes sense to minimize $(Kn')^L$ with $n'L$ fixed by the desired precision. Consequently multiple levels does not pay for L small but pays when L is larger than some critical number. Of course parallelism is in favor of multi-levels.

3.5 American Options

We denote by V_k a realization of $V(t_k, X_{t_k})$ given by the BDP algorithm (4), for $k = 0 : K = \frac{T}{\Delta t}$.

As before each interval $(t_k, t_{t_{k+1}})$ has a uniform sub-partition of time step $\delta t = \frac{\Delta t}{J}$, for some integer $J > 1$ and $n = 0 : N-1$ is the iteration index of the parareal algorithm.

Consider the following

Remark 15. *Note that all fine grid computations are local and can be allocated to a separate processor for each k, for parallelization;*

Algorithm 7 TLPRAO(A): Two levels Parareal for American Options

1: **Initialization**:

From $\hat{V}_K^0(\omega^m) = e^{-rT} f(\hat{X}_T^0(\omega^m))$, $m = 1 : M$ Compute $\hat{V}_k^0 = \max\{f(\hat{X}_{t_k}^0), \mathfrak{P}\,\mathbb{E}[\hat{V}_{k+1}^0 | \hat{X}_{t_k}^0]\}$, , $k = K-1:0$;

2: **for** $n = 0 : N-1$ **do** (parareal iterations)

3: Evaluate $\bar{V}_K^{n+1} = \hat{V}_K^{n+1} = f(\hat{X}_T^{n+1})$;

4: **for** $k = K-1:0$ **do**

5: On each (t_k, t_{k+1}), from $\tilde{V}_{k,J}^n = \mathfrak{P}\,\mathbb{E}(\hat{V}_{k+1}^n \,|\, \tilde{X}_{k,J}^n)$, (alternatively $\tilde{V}_{k,J}^n = \mathfrak{P}\,\mathbb{E}(\bar{V}_{k+1}^n \,|\, \tilde{X}_{k,J}^n)$) compute

$$\tilde{V}_{k,j}^n = \max\left\{f(\tilde{X}_{t_{k,j}}^n), e^{-r\delta t}\mathfrak{P}\,\mathbb{E}[\tilde{V}_{k,j+1}^n | \tilde{X}_{t_{k,j}}^n]\right\},\ j = J-1:0.$$

6: Compute $\bar{V}_k^{n+1} = \max\left\{f(\hat{X}_{t_k}^{n+1}), e^{-r\Delta t}\mathfrak{P}\,\mathbb{E}[\bar{V}_{k+1}^{n+1} | \hat{X}_{t_k}^{n+1}]\right\}$.

7: Set $\hat{V}_k^{n+1} = \bar{V}_k^{n+1} + \tilde{V}_{k,0}^n - \hat{V}_k^n$ (alt. $\hat{V}_k^{n+1} = \bar{V}_k^{n+1} + \tilde{V}_{k,0}^n - \bar{V}_k^n$).

8: **end for**

9: **end for**

Remark 16. *The above describes an algorithm called TLPRAO and an alternative version which we call TLPRAOA. The tests show that the alternative version converges somewhat slower, but we are able to prove the following results, when the conditional expectations are defined with respect to the filtration $\mathcal{F}$ instead of the process X and then for the algorithm as described. There is a difference because $\hat{X}_{t_k}^{n+1}$ is not a Markov process as it depends on all the past of the algorithm.*

Remark 17. *In the Longstaff-Schwartz algorithm, BDP formulas like*

$$V_k = \max\left\{f(X_{t_k}), e^{-r\Delta t}\mathfrak{P}\,\mathbb{E}[V_{k+1} | X_{t_k}]\right\}$$

are replaced by $V_k = \mathbb{E}[e^{-r\tau_k} f(X_{\tau_k}) \,|\, X_{t_k}]$ with τ initialized by $\tau_K = T$ and then recursively computed by

$$\tau_k = k\Delta t \text{ if } f(X_{t_k}) > \mathfrak{P}\mathbb{E}[e^{r(t_k - \tau_{k+1})} f(X_{\tau_{k+1}}) | X_{t_k}], \text{ else } \tau_k = \tau_{k+1}.$$

Thus, in the TLPRAOA (without the projection $\mathfrak{P}$), the dynamics of $(\widetilde{V}_k^{n+1})_{k=0:K}$ satisfies a standard BDP since $\hat{X}_{t_k}^n$ is replaced by the whole σ-field $\mathcal{F}_{t_k}$. Hence

$$e^{-rt_k}\bar{V}_{t_k}^{n+1} = \max\left(f(\hat{X}_{t_k}^{n+1}), e^{-rt_{k+1}}\mathbb{E}[\bar{V}_{t_{k+1}}^{n+1} | \mathcal{F}_{t_k}]\right),\ k = 0, \ldots, K-1, \tag{3.48}$$

so that $(e^{-rt_k}\bar{V}_{t_k}^{n+1})$ is the $(\mathcal{F}_{t_k}, \mathbb{P})$-Snell envelope of $\left(e^{-rt_k} f(\hat{X}_{t_k}^{n+1})\right)_{k=0:K}$. As such, it can be equivalently characterized as

$$\bar{V}_{t_k}^n = \mathbb{P}\text{-esssup}_{\tau \in \mathcal{T}_{t_k}^{\mathcal{F}}} \mathbb{E}[e^{-r(\tau - t_k)} f(\tau, \hat{X}_\tau^n) | \mathcal{F}_{t_k}],\ k = 0 : K \tag{3.49}$$

where $\mathcal{T}_{t_k}^{\mathcal{F}}$ denotes the set of $\{t_k, t_{k+1} : t_K\}$-valued $\mathcal{F}$-stopping times.

Proposition 5. *Consider TLPRAOA. Let*

$$\bar{V}^{\Delta,\delta}_{t_k} = \mathbb{P}\text{-esssup}_{\tau\in\mathcal{T}^{\mathcal{F}}_{t_k}}\mathbb{E}[e^{-r(\tau-t_k)}f(\tau,\bar{X}^{\delta}_{\tau})|\mathcal{F}_{t_k}]., \ k=0,\dots,K. \tag{3.50}$$

Then $\left(e^{-rt_k}\bar{V}^{\Delta,\delta}_{t_k}\right)_{k=0:K}$ *is the* $(\mathcal{F}_{t_k},\mathbb{P})$*-Snell envelope of* $\left(e^{-rt_k}f(t_k,\bar{X}^{\delta}_{t_k})\right)_{k=0:K}$ *where* $(\bar{X}^{\delta}_{t_{k,j}})_{k=0:K,j=0:J}$ *denotes the standard Euler scheme of the diffusion with step* δt *and* $\bar{X}^{\delta t}_{t_k}=\bar{X}^{\delta t}_{t_{k,0}}$*. This Snell envelope is related to the fine Euler scheme, only observed at coarse epochs.*

(a) Conditioning by $\mathcal{F}_{t_k}$*. Then, for some constant* $C=C_{b,\sigma,T}$*,*

$$\left\|\max_{k=0:K}\left|\bar{V}^{n}_{t_k}-\bar{V}^{\Delta,\delta}_{t_k}\right|\right\|_{L^2(\mathbb{P})} \leq [f]_{\mathrm{Lip}}\frac{(C\Delta t)^{\frac{n}{2}}}{\sqrt{(n+1)!}}\Big(1+\frac{\Delta t}{T}\Big)^{\frac{1}{2}}e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}. \tag{3.51}$$

At a fixed time t_k*, we have the better estimate*

$$\left\|\bar{V}^{n}_{t_k}-\bar{V}^{\Delta,\delta}_{t_k}\right\|_{L^2(\mathbb{P})} \leq [f]_{Lip}(C\Delta t)^{n+\frac{1}{2}}\sqrt{\binom{K+1}{n+1}-\binom{k}{n+1}}. \tag{3.52}$$

(b) Conditioning by $\hat{X}^{n}_{t_k}$*. At the cost of losing a* $\sqrt{\Delta t}$ *in the error estimate, sa result similar to* (a) *holds for the "variante"* $(\breve{V}^{n}_{t_k})_{k=0:K}$ *of* $(\bar{V}^{n}_{t_k})_{k=0:K}$ *defined by a (pseudo-) BDP mimicking* (3.48) *where the conditioning is taken given* $\hat{X}^{n}_{t_k}$ *in place of* $\mathcal{F}_{t_k}$*, namely*

$$\breve{V}^{n}_{t_k} = \max\left\{f(\hat{X}^{n}_{t_k}),e^{-r\delta t}\mathbb{E}[\breve{V}^{n}{}_{t_{k+1}}|\mathcal{F}_{t_k}]\right\}, \ k=0,\dots,K-1. \tag{3.53}$$

Then, one has

$$\left\|\breve{V}^{n}_{t_k}-\bar{V}^{\Delta,\delta}_{t_k}\right\|_{L^2(\mathbb{P})} \leq \frac{(C''\Delta t)^{\frac{n-1}{2}}}{\sqrt{(n+1)!}}\Big(1+\frac{\Delta t}{T}\Big)e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}. \tag{3.54}$$

3.5.0.1.4 Proof (a) For every $k\in\{0,\dots,K\}$,

$$\begin{aligned}
\left|\bar{V}^{n}_{t_k}-\bar{V}^{\Delta,\delta}_{t_k}\right| &= \left|\mathbb{P}\text{-esssup}_{\tau\in\mathcal{T}^{\mathcal{F}}_{t_k}}\mathbb{E}\big[e^{-r(\tau-t_k)}f(\tau,\hat{X}^{n}_{\tau})|\mathcal{F}_{t_k}\big]-\mathbb{P}\text{-esssup}_{\tau\in\mathcal{T}^{\mathcal{F}}_{t_k}}\mathbb{E}\big[e^{-r(\tau-t_k)}f(\tau,\bar{X}^{\delta}_{\tau})\big|\mathcal{F}_{t_k}\big]\right| \\
&\leq \mathbb{P}\text{-esssup}_{\tau\in\mathcal{T}^{\mathcal{F}}_{t_k}}\left|\mathbb{E}\Big[e^{-r(\tau-t_k)}\big(f(\tau,\hat{X}^{n}_{\tau})-f(\tau,\bar{X}^{\delta}_{\tau})\big)\,\Big|\,\mathcal{F}_{t_k}\Big]\right| \\
&\leq \mathbb{E}\left[\mathbb{P}\text{-esssup}_{\tau\in\mathcal{T}^{\mathcal{F}}_{t_k}}\left|f(\tau,\hat{X}^{n}_{\tau})-f(\tau,\bar{X}^{\delta}_{\tau})\right|\,\middle|\,\mathcal{F}_{t_k}\right] \\
&\leq [f]_{\mathrm{Lip}}\mathbb{E}\left[\max_{l=k:K}\left|\hat{X}^{n}_{t_l}-\bar{X}^{\delta}_{t_l}\right|\,\middle|\,\mathcal{F}_{t_k}\right].
\end{aligned} \tag{3.55}$$

Consequently, owing to Doob's inequality,

$$\begin{aligned}
\left\|\max_{k=0:K}\left|\bar{V}^{n}_{t_k}-\bar{V}^{\Delta,\delta}_{t_k}\right|\right\|_2^2 &\leq 4[f]^2_{Lip}\left\|\max_{l=0:K}\left|\hat{X}^{n}_{t_l}-\bar{X}^{\delta}_{t_l}\right|\right\|_2^2 \leq 4[f]^2_{Lip}\sum_{l=0}^{K}\left\|\hat{X}^{n}_{t_l}-\bar{X}^{\delta}_{t_l}\right\|_2^2 \\
&\leq 4[f]^2_{Lip}(C\Delta t)^{2n+1}\sum_{l=n}^{K}\binom{l}{n} \\
&= 4[f]^2_{Lip}(C\Delta t)^{2n+1}\binom{K+1}{n+1} \\
&\leq \left[2[f]_{Lip}\frac{(C'\Delta t)^{\frac{n}{2}}}{\sqrt{(n+1)!}}\Big(1+\frac{\Delta t}{T}\Big)^{\frac{1}{2}}e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}\right]^2
\end{aligned} \tag{3.56}$$

where we used in the second line that $\hat{X}^n_{t_l} = \bar{X}^\delta_{t_l}$ for $l \le n-1$ and (3.47) to obtain the last inequality.

If we are only interested in $\big\|\widetilde{V}^{\Delta,n}_{t_k} - \bar{V}^{\Delta,\delta}_{t_k}\big\|_2^2$, we obtain starting again from (3.55)

$$\begin{aligned}\big\|\bar{V}^n_{t_k} - \bar{V}^{\Delta,\delta}_{t_k}\big\|_2^2 &\le [f]^2_{Lip}\sum_{l=k\vee n}^{K}\big\|\hat{X}^n_{t_l} - \bar{X}^\delta_{t_l}\big\|_2^2 \le [f]^2_{Lip}(C\Delta t)^{2n+1}\sum_{l=k\vee n}^{K}\binom{l}{n}\\ &= [f]^2_{Lip}(C\Delta t)^{2n+1}\left[\binom{K+1}{n+1} - \binom{k}{n+1}\right] \end{aligned} \tag{3.57}$$

with the usual convention on the binomial coefficient $\binom{k}{n}$ when $k \ge n$.

(b) The proof of (b) is postponed at the end of the next section. □

3.5.1 Correcting Markovian deficiency

We define the sequence of random variables $(\check{V}_{t_k})_{k=0:K}$ by the following (pseudo-)BDP:

$$\check{V}^n_T = f(T, \hat{X}^n_T),\ \check{V}^n_{t_k} = \max\Big(f(t_k, \hat{X}^n_{t_k}), \mathbb{E}\big(\check{V}^n_{t_k}|\hat{X}^n_{t_k}\big)\Big),\ k = 0:K-1.$$

What is the error induced by considering $\check{V} - n_{t_k}$ rather than $\bar{V}^n_{t_k}$?

Proposition 6. *There exists a real constant $C = C_{b,\sigma,f,T} > 0$ only depending on b, σ, $[f]_{\mathrm{Lip}}$ and T such that*

$$\max_{k=0:K}\big\|\check{V}^n_{t_k} - \bar{V}^n_{t_k}\big\|_2 \le \frac{(C\Delta t)^{\frac{n-1}{2}}}{\sqrt{(n+1)!}}\Big(1+\frac{\Delta t}{T}\Big)^{\frac12} e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}. \tag{3.58}$$

Proof. Starting from both backward definitions for $(\check{V}_{t_k})$ and $(\bar{V}_{t_K})_k$ we get

$$|\check{V}_{t_k} - \bar{V}^n_{t_k}| \le \Big|\mathbb{E}\big(\check{V}^n_{t_{k+1}}\big|\hat{X}^n_{t_k}\big) - \mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\Big|.$$

As $\hat{X}^n_{t_k}$ is $\mathcal{F}_{t_k}$-measurable, one has

$$\mathbb{E}\big(\check{V}^n_{t_{k+1}}\big|\hat{X}^n_{t_k}\big) - \mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big) = \mathbb{E}\big(\check{V}^n_{t_{k+1}} - \bar{V}^n_{t_{k+1}}\big|\hat{X}^n_{t_k}\big) \overset{\perp}{+} \mathbb{E}\big(\mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\big|\hat{X}^n_{t_k}\big)\big) - \mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)$$

where $\perp$ denotes orthogonality in $L^2(\mathbb{P})$. Consequently, also using that conditional expectations is an L^2-contraction, we obtain

$$\big\|\check{V}_{t_k} - \bar{V}^n_{t_k}\big\|_2^2 \le \big\|\check{V}^n_{t_{k+1}} - \bar{V}^n_{t_{k+1}}\big\|_2^2 + \big\|\mathbb{E}\big(\mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\big|\hat{X}^n_{t_k}\big)\big) - \mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\big\|_2^2. \tag{3.59}$$

Conditional expectation being an orthogonal projector, for every Borel function $\varphi : \mathbb{R} \to \mathbb{R}$ such that $\varphi(\hat{X}^n_{t_k}) \in L^2(\mathbb{P})$

$$\big\|\mathbb{E}\big(\mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\big|\hat{X}^n_{t_k}\big)\big) - \mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\big\|_2^2 \le \big\|\varphi(\hat{X}^n_{t_k}) - \mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\big\|_2^2. \tag{3.60}$$

Now, let us consider the sequence $(\bar{V}^\delta_{t_{k,j}})_{k=0:k,j=0:J}$ such that $(e^{-rt_k}\bar{V}^\delta_{t_{k,j}})_{k=0:k,j=0:J}$ is the $(\mathcal{F}_{t_{k,j}},\mathbb{P})$-Snell envelope of the discounted payoff $\big(e^{-rt_k}f(t_{k,j},\bar{X}^\delta_{t_{k,j}})\big)_{k,j}$ associated to the global refined Euler scheme $(\bar{X}^\delta_{t_{k,j}})_{k,j}$ with step δt..

It is defined by

$$\bar{V}^{\delta}_{t_{k,j}} = \mathbb{P}\text{-esssup}_{\tau\in\mathcal{T}^{\mathcal{F},\delta}_{t_{k,j}}}\mathbb{E}\big[e^{-r(\tau-t_{k,j})}f(\tau,\bar{X}^{\delta}_{\tau})\,|\,\mathcal{F}_{t_{k,j}}\big] \tag{3.61}$$

where, for every $k\in\{0,\dots,K\}$, $\mathcal{T}^{\mathcal{F},\delta}_{t_{k,j}}$ denotes the set of $\mathcal{F}$-stopping times taking values in $\big\{t_{k',j'},\, k'\in\{k:K\},\, j'\in\{j:J\}\big\}$.
Then, as the Euler scheme is a Markov chain, $(\bar{V}^{\delta}_{t_{k,j}})_{k,j}$ satisfies the BDP

$$\bar{V}^{\delta}_{t_{k,j}} = \max\big\{f(t_{k,j},\bar{X}^{\delta}_{t_{k,j}}), e^{-r\delta t}\mathbb{E}(\bar{V}^{\delta}_{t_{k,j+1}}\,|\,\bar{X}^{\delta}_{t_{k,j}})\big\},\ j=0:J-1,\ k=0:K-1 \tag{3.62}$$

having in mind that $t_{k,J}=t_{k+1,0}$. Hence,

$$\big\|\varphi(\hat{X}^n_{t_k})-\mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\big\|_2^2 \le \Big(\big\|\bar{V}^n_{t_{k+1}}-\bar{V}^{\delta}_{t_{k+1}}\,\big\|_2+\big\|\varphi(\hat{X}^n_{t_k})-\mathbb{E}\big(\bar{V}^{\delta}_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)\big\|_2\Big)^2. \tag{3.63}$$

It classically follows from a backward induction, based on the above BDP that there exist Borel functions $v_{k,j}:\mathbb{R}^d\to\mathbb{R}_+$ such that

$$\bar{V}^{\delta}_{t_{k,j}} = v_{k,j}(\bar{X}^{\delta}_{t_{k,j}}),\ j=0:J,\ k=0:K$$

with $v_{K,J}=f(T,.)$ (and $v_{k,J}=v_{k+1,0}$). The propagation of Lipschitz continuity by the Euler transition kernel (see [11]) implies t that these functions are in fact Lipschitz continuous and

$$[v_{k,j}]_{\mathrm{Lip}}\le C=C_{b,\sigma,T}[f]_{\mathrm{Lip}},\ k=0:K,\ j=0:J$$

where $C_{b,\sigma,T}$ only depends on $[b]_{\mathrm{Lip}}$, $[\sigma]_{\mathrm{Lip}}$ and T. Consequently

$$\mathbb{E}\big(\bar{V}^n_{t_{k+1}}\big|\mathcal{F}_{t_k}\big)=\mathbb{E}\big(v_{t_{k+1,0}}(\bar{X}^{\delta}_{t_{k+1}})\big|\mathcal{F}_{t_k}\big)=\mathbb{E}\big(v_{t_{k+1,0}}\circ G^{(J)}_{\delta}(\bar{X}^{\delta}_{t_k},\mathbf{Z}_{k+1})\big|\bar{X}^{\delta}_{t_k}\big)=w_k(\bar{X}^{\delta}_{t_k})$$

with

$$[w_k]_{\mathrm{Lip}}\le(1+C'\delta t)^J[v_{k+1,0}]_{\mathrm{Lip}}\le e^{C'\Delta t}C[f]_{\mathrm{Lip}}\le(1+C'\Delta t)C[f]_{\mathrm{Lip}}$$

(where, once again, C' and C'' only depend on $[b]_{\mathrm{Lip}}$, $[\sigma]_{\mathrm{Lip}}$ and T).

Setting $\varphi=w_k$ in (3.60) and (3.63), plugging these inequalities in (3.59) and then using Theorem 9 and Equation (3.52), yields the the recursion inequality

$$\begin{aligned}
\big\|\breve{V}_{t_k}-\bar{V}^n_{t_k}\big\|_2^2 &\le \big\|\breve{V}^n_{t_{k+1}}-\bar{V}^n_{t_{k+1}}\big\|_2^2\\
&\quad+2\left[[f]^2_{\mathrm{Lip}}(C\Delta t)^{2n+1}\left[\binom{K+1}{n+1}-\binom{k}{n+1}\right]+[w_k]^2_{\mathrm{Lip}}C^{2n}\binom{k}{n}(\Delta t)^{2n+1}\right]\\
&\le \big\|\breve{V}^n_{t_{k+1}}-\bar{V}^n_{t_{k+1}}\big\|_2^2+C'[f]^2_{\mathrm{Lip}}(C\Delta t)^{2n+1}\left[\binom{K+1}{n+1}-\binom{k}{n+1}+\binom{k}{n}\right]\\
&\le \big\|\breve{V}^n_{t_{k+1}}-\bar{V}^n_{t_{k+1}}\big\|_2^2+C'[f]^2_{\mathrm{Lip}}(C\Delta t)^{2n+1}\binom{K+1}{n+1}.
\end{aligned}$$

Having in mind that $\breve{V}_T=\bar{V}^n_T=f(T,\hat{X}^n_T)$, we derive

$$\begin{aligned}
\big\|\breve{V}_{t_k}-\bar{V}^n_{t_k}\big\|_2^2 &\le C'f]^2_{\mathrm{Lip}}(C\Delta t)^{2n+1}(K-k)\binom{K+1}{n+1}\\
&\le C'[f]^2_{\mathrm{Lip}}T\,(C\Delta t)^{2n}\binom{K+1}{n+1}\\
&\le C'[f]^2_{\mathrm{Lip}}\frac{(C\Delta t)^{n-1}}{(n+1)!}\Big(1+\frac{\Delta t}{T}\Big)e^{-\frac{n(n-1)}{2}\frac{\Delta t}{T}}
\end{aligned} \tag{3.64}$$

where we used (3.47) in the last line. □

Proof of claim (b) **of Proposition 5**. Triangle inequality, etc...

3.5.2 Some Considerations on TLPRAO

Let us consider TLPRAO but without least-square projections, i.e. with $\mathfrak{P} = \mathbf{Id}$. We recall it below for clarity.

for k=K-1:0 (backward loop):

1. On each (t_k, t_{k+1}), from $\tilde{V}^{\delta,n}_{k,J} = \mathbb{E}(\bar{V}^n_{k+1} \,|\, \tilde{X}^{\delta,n}_{k,J})$, compute

$$\tilde{V}^{\delta,n}_{k,j} = \max\big\{f(t_{k,j}, \tilde{X}^{\delta,n}_{t_{k,j}}), e^{-r\delta t}\mathbb{E}[\tilde{V}^{\delta,n}_{k,j+1}|\tilde{X}^{\delta,n}_{t_{k,j}}]\big\},\ j = J-1:0. \tag{3.65}$$

2. Compute $\bar{V}^{n+1}_k = \max\big\{f(t_k, \hat{X}^{n+1}_{t_k}), e^{-r\Delta t}\mathbb{E}[\bar{V}_{k+1}|\hat{X}^{n+1}_{t_k}]\big\}$.

3. Set $\hat{V}^{n+1}_k = \bar{V}^{n+1}_k + \tilde{V}^{\delta,n}_{k,0} - \bar{V}^n_k$.

In this modified algorithm, our first aim is to compare $\widetilde{V}^{\delta,n}_{t_{k,0}}$ and $\bar{V}^{\delta}_{t_{k,0}}$ and, more generally, $\widetilde{V}^{\delta,n}_{t_{k,j}}$ and $\bar{V}^{\delta}_{t_{k,j}}$ where $(e^{-rt_{k,j}}\bar{V}^{\delta}_{t_{k,j}})_{j=0:J}$ is the $(\mathcal{F}_{t_{k,j}}, \mathbb{P})_{k=0:K, j=0:J}$-Snell envelope of $\big(e^{-rt_{k,j}} f(t_{k,j}, \bar{X}^{\delta}_{t_{k,j}})\big)_{k,j}$, $(\bar{X}^{\delta}_{t_{k,j}})_{k=0:K, j=0:J}$ being the global Euler scheme with step δt (precisely defined in (3.15)).

The two characterizations of $(\bar{V}^{\delta}_{t_{k,j}})_{j=0:J}$ as a discounted the Snell envelope are recalled in (3.61) and (3.62) respectively. In particular, k being fixed, $(\bar{X}^{\delta}_{t_{k,j}})_{j=0:J}$ is a Markov chain so that

$$\bar{V}^{\delta}_{t_{k,j}} = \mathbb{P}\text{-esssup}_{\theta \in \mathcal{T}^{\mathcal{F},\delta}_{t_{k,j}}}\big[e^{-r(\theta - t_{k,j})}\bar{f}_\theta \,|\, \mathcal{F}^{\delta}_{t_{k,j}}\big],\ j = 0, \ldots, J.$$

with $\bar{f}_{t_{k,j}} = f(t_{k,j}, \bar{X}^{\delta}_{t_{k,j}})$, $j = 0, \ldots, J$ and

$$\bar{V}^{\delta}_{t_{k,j}} = \bar{v}_{k,j}(\bar{X}^{\delta}_{t_{k,j}}),\ j = 0, \ldots, J \quad \text{with} \quad \bar{v}_{k,j} = \max\big\{f(t_{k,j}, .), e^{-r\delta t}\bar{v}_{k;j+1}\big(G_\delta(., \zeta)\big)\big\}$$

(where ζ is $\mathcal{N}(0; I_q)$-distributed).

Let us focus now on $(\widetilde{V}^{\delta,n}_{t_{k,j}})_{j=0:J}$ as defined in (3.65). Having in mind that, $k \in \{0 : K-1\}$ being fixed, $(\widetilde{X}^{\delta,n}_{t_{k,j}})_{j=0:J}$ is the Euler scheme with step δt starting from $\widehat{X}^n_{t_k}$ at time $t_k = t_{k,0}$, and, as such, an $\big((\mathcal{F}_{t_{k,j}}), \mathbb{P}\big)_{j=0:J}$-Markov chain, it is clear that $\big(\widetilde{V}^{\delta,n}_{t_{k,j}}\big)_{j=0:J}$ as defined above in (3.65) is the $\big((\mathcal{F}_{t_{k,j}})_{j=0:J}, \mathbb{P}\big)$-Snell envelope (with horizon Δt) of the obstacle process

$$\widetilde{f}_{t_{k,J}} = \widetilde{V}^{\delta,n}_{t_{k,J}} \quad \text{and} \quad \widetilde{f}_{t_{k,j}} = f(t_{k,j}, \widetilde{X}^{\delta,n}_{t_{k,j}}),\ j = 0 : J-1.$$

Consequently, on the one hand,

$$\widetilde{V}^{\delta,n}_{t_{k,j}} = \mathbb{P}\text{-esssup}_{\theta \in \mathcal{T}^{\mathcal{F},\delta}_{t_{k,j}}}\big[e^{-r(\theta - t_{k,j})}\widetilde{f}_\theta \,|\, \mathcal{F}^{\delta}_{t_{k,j}}\big],\ j = 0, \ldots, J.$$

and, on the other hand,

$$\widetilde{V}^{\delta,n}_{k,j} = \widetilde{v}_{k,j}(\widetilde{X}^{\delta,n}_{t_{k,j}}),\ j = 0, \ldots, J,$$

where $\widetilde{v}_{k,J} : \mathbb{R}^d \to \mathbb{R}_+$ is defined from the identity $\widetilde{V}^{\delta,n}_{k,J} = \mathbb{E}(\bar{V}^n_{t_{k+1}} \,|\, \widetilde{X}^{\delta,n}_{k,J}) = \widetilde{v}_{k,J}(\widetilde{X}^{\delta,n}_{k,J})$ and, for every $j = 0 : J-1$, $\widetilde{v}_{k,j} = \max\big\{f(t_{k,j}), e^{-r\delta t}\mathbb{E}\, v_{k,j+1}\big(G_\delta(., \zeta)\big)\big\}$ (ζ is $\mathcal{N}(0; I_q)$-distributed).

Proposition 7. *There exists a real constant $C = C_{b,\sigma,T}$ such that, for every $k = 0 : K-1$,*

$$\Big\| \max_{j=0:J} \big|\bar V^\delta_{t_{k,j}} - \widetilde V^{\delta,n}_{t_{k,j}}\big| \Big\|_2 \le [f]_{\mathrm{Lip}} C\sqrt{\Delta t}.$$

In particular for every $k = 0 : K$, $\big\|\bar V^\delta_{t_k} - \bar V^n_{t_k}\big\|_2 \le [f]_{\mathrm{Lip}} C\sqrt{\Delta t}$.

Proof of Proposition 7: Here also we assume $r = 0$ to improve readability. As seen in the proof of the former proposition, one immediately derives from the definitions of $\bar V$ and $\widetilde V^\delta$ as $(\mathbb{P}, \mathcal{F}^\delta_{t_{k,j}})$-Snell envelopes that

$$\big|\bar V^\delta_{t_{k,j}} - \widetilde V^{\delta,n}_{t_{k,j}}\big| \le \mathbb{E}\left[\max_{\ell=j:J} \big|\widetilde f_{t_{k,\ell}} - \bar f_{t_{k,\ell}}\big| \,\Big|\, \mathcal{F}_{t_{k,j}} \right]$$

so that, owing to conditional Jensen's Inequality

$$\big|\bar V^\delta_{t_{k,j}} - \widetilde V^{\delta,n}_{t_{k,j}}\big|^2 \le \mathbb{E}\left[\max_{\ell=0:J} \big|\widetilde f_{t_{k,\ell}} - \bar f_{t_{k,\ell}}\big|^2 \,\Big|\, \mathcal{F}_{t_{k,j}} \right],\ j = 0 : J.$$

It follows from the conditional Doob's Inequality that

$$\mathbb{E}\left[\max_{j=0:J} \big|\bar V^\delta_{t_{k,j}} - \widetilde V^{\delta,n}_{t_{k,j}}\big|^2 \,\Big|\, \mathcal{F}_{t_{k,0}} \right] \le 4\mathbb{E}\left[\max_{\ell=0:J} \big|\widetilde f_{t_{k,\ell}} - \bar f_{t_{k,\ell}}\big|^2 \,\Big|\, \mathcal{F}_{t_{k,0}} \right].$$

Now, f being Lipschitz continuous in x, uniformly in $t \in [0,T]$, we get

$$\begin{aligned} \max_{\ell=0:J} \big|\widetilde f_{t_{k,\ell}} - \bar f_{t_{k,\ell}}\big|^2 &\le |\mathbb{E}(\bar V^n_{t_{k+1}} \,|\, \widetilde X^n_{t_{k,J}}) - \bar V^\delta_{t_{k,J}}|^2 \vee \left[[f]^2_{\mathrm{Lip}} \max_{j=0:J-1} |\widetilde X^n_{t_{k,j}} - \bar X^\delta_{t_{k,j}}|^2 \right] \\ &\le \big|\mathbb{E}(\bar V^n_{t_{k+1}} \,|\, \widetilde X^n_{t_{k,J}}) - \bar V^\delta_{t_{k,J}}\big|^2 + [f]^2_{\mathrm{Lip}} \max_{j=0:J-1} \big|\widetilde X^n_{t_{k,j}} - \bar X^\delta_{t_{k,j}}\big|^2. \quad (3.66) \end{aligned}$$

Conditional expectation given $\widetilde X^n_{t_{k,J}}$ being an orthogonal projector on $L^2(\sigma(\widetilde X^n_{t_{k,J}}), \mathbb{P})$, one has by the Pythagoras Theorem

$$\big\|\mathbb{E}\big(\bar V^n_{t_{k+1}} \,|\, \widetilde X^n_{t_{k,J}}\big) - \bar V^\delta_{t_{k,J}}\big\|^2 = \big\|\mathbb{E}\big(\bar V^n_{t_{k+1}} - \bar V^\delta_{t_{k,J}} \,|\, \widetilde X^n_{t_{k,J}}\big)\big\|^2 + \big\|\mathbb{E}\big(\bar V^\delta_{t_{k,J}} \,|\, \widetilde X^n_{t_{k,J}}\big) - \bar V^\delta_{t_{k,J}}\big\|^2.$$

The functions $f(t_{k,j},.)$ being $[f]_{\mathrm{Lip}}$-Lipschitz, the functions $\bar v_{k,j}$ are uniformly Lipschitz (see Bally and Pagès [11]). In particular, for every $k \in \{0,\dots,K\}$, $\bar V^\delta_{t_{k,J}} = \bar v_{k,J}(\bar X^\delta_{t_{k,J}})$ where the functions $\bar v_{k,J}$ are Lipschitz continuous with $[v_{k,J}]_{\mathrm{Lip}} \le C = C_{b,\sigma,T}$. Conditional expectation given $\widetilde X^n_{t_{k,J}}$ being the best quadratic approximation by a function of $\widetilde X^n_{t_{k,J}}$, one has

$$\big\|\bar V^\delta_{t_{k,J}} - \mathbb{E}\big(\bar V^\delta_{t_{k,J}} \,|\, \widetilde X^n_{t_{k,J}}\big)\big\|_2^2 \le [\bar v_{k,J}]^2_{\mathrm{Lip}} \big\|\bar X^\delta_{t_{k,J}} - \widetilde X^n_{t_{k,J}}\big\|_2^2 \le C\big\|\bar X^\delta_{t_{k,J}} - \widetilde X^n_{t_{k,J}}\big\|_2^2$$

whereas, as an L^2-contraction,

$$\big\|\mathbb{E}\big(\bar V^n_{t_{k+1}} - \bar V^\delta_{t_{k,J}} \,|\, \widetilde X^n_{t_{k,J}}\big)\big\|_2^2 \le \big\|\bar V^n_{t_{k+1}} - \bar V^\delta_{t_{k,J}}\big\|_2^2 = \big\|\bar V^n_{t_{k+1}} - \bar V^\delta_{t_{k+1}}\big\|_2^2.$$

On the other hand, the Euler schemes yield

$$\mathbb{E}\left[\max_{j=0:J} \big|\widetilde X^n_{t_{k,j}} - \bar X^\delta_{t_{k,j}}\big|^2 \,\Big|\, \mathcal{F}_{t_{k,0}} \right] \le C\big|\hat X^n_{t_k} - \bar X^\delta_{t_k}\big|^2,$$

for a real constant $C = C_{b,\sigma,T}$. Taking the expectation in (3.66) and in the above equation and then using the bounds established in (3.13) of Theorem 9 and in (3.52), we derive that

$$\begin{aligned}\mathbb{E}\left[\max_{j=0:J}\left|\bar{V}^{\delta}_{t_{k,j}}-\widetilde{V}^{\delta,n}_{t_{k,j}}\right|^2\right] &\leq 4\left[[f]^2_{\mathrm{Lip}}C_{b,\sigma,T}\mathbb{E}\left[\left|\hat{X}^n_{t_k}-\bar{X}^{\delta}_{t_k}\right|^2\right]+\mathbb{E}\left[\left|\bar{V}^{\delta}_{t_{k+1}}-\bar{V}^n_{t_{k+1}}\right|^2\right]\right]\\ &\leq 4\left[[f]^2_{\mathrm{Lip}}C^{2n+1}\binom{k}{n}(\Delta t)^{2n+1}+\mathbb{E}\left[\left|\bar{V}^{\delta}_{t_{k+1}}-\bar{V}^n_{t_{k+1}}\right|^2\right]\right] \quad (3.67)\end{aligned}$$

by Theorem 9. Now, if we denote by $(\bar{V}^{\delta}_{t_k})_{k=0:K}$ the $(\mathbb{P},\mathcal{F}_{t_k})$-Snell envelope of $f(t_k,\bar{X}^{\delta}_{t_k})_{k=0:K}$ (where exercise is possible only at times t_k), then

$$\left\|\bar{V}^{\delta}_{t_{k+1}}-\bar{V}^n_{t_{k+1}}\right\|_2\leq\left\|\bar{V}^{\delta}_{t_{k+1}}-\bar{V}^{\Delta,n}_{k+1}\right\|_2+\left\|\bar{V}^{\Delta,n}_{k+1}-\bar{V}^n_{t_{k+1}}\right\|_2.$$

Now we will show, using the very definition of the (discounted) Snell envelope, the uniform Lipschitz continuity of the functions $f(t_k,.)$ and the fact that the Euler scheme is an Itô process that

First note that, by its very definition

$$\bar{V}^{\delta}_{t_k}=\mathbb{P}\text{-ess sup}\Big\{\mathbb{E}\big[e^{-r(\tau-t_k)}f(\tau,\bar{X}^{\delta}_{\tau})\,|\mathcal{F}_{t_k}\big],\ \tau\in\mathcal{T}^{\mathcal{F},\delta}_{t_{k,0},t_{K,J}}\Big\}$$

whereas (see (3.50))

$$\bar{V}^{\Delta,n}_{t_k}=\mathbb{P}\text{-esssup}\Big\{\mathbb{E}\big[e^{-r(\tau-t_k)}f(\tau,\bar{X}^{\delta}_{\tau})|\mathcal{F}_{t_k}\big],\ \tau\in\mathcal{T}^{\mathcal{F}}_{t_k}\Big\}$$

keeping in mind that $\mathcal{T}^{\mathcal{F},\delta}_{t_{k,0},t_{K,J}}$ denotes the set of $\mathcal{F}_{t_{\ell,j}}$-stopping times having values in $\{t_{\ell,j},\,\ell=k:K,,\,j=0:J\}$ whereas $\mathcal{T}^{\mathcal{F}}_{t_k}$ is the set of $\mathcal{F}_{t_k}$-stopping times having values in the coarse mesh $\{t_\ell,\,\ell=k:K\}$. It is clear that $\bar{V}^{\delta,\Delta}_{t_k}\leq\bar{V}^{\delta}_{t_k}$ since $\mathcal{T}^{\mathcal{F}}_{t_k}\subset\mathcal{T}^{\mathcal{F},\delta}_{t_{k,0},t_{K,J}}$.

Then, to every stopping $\tau\in\mathcal{T}^{\mathcal{F},\delta}_{t_{k,0},t_{K,J}}$, we associate its left best approximation in $\tau^{\Delta}\in\mathcal{T}^{\mathcal{F}}_{t_k}$, namely $\tau^{\Delta}=\sum_{\ell=k}^{K-1}t_\ell\mathbf{1}_{\{t_{\ell,0}\leq\tau\leq t_{\ell,J-1}\}}+T\mathbf{1}_{\{\tau=T\}}$. It is clear that $\tau^{\Delta}\in\mathcal{T}^{\mathcal{F}}_{t_k}$ and $0\leq\tau-\tau^{\Delta}\leq\Delta t$. Consequently

$$\bar{V}^{\Delta,\delta}_{t_k}\geq\mathbb{P}\text{-esssup}\big\{\mathbb{E}[f(\tau^{\Delta},\bar{X}^{\delta}_{\tau^{\Delta}})|\mathcal{F}_{t_k}],\ \tau\in\mathcal{T}^{\mathcal{F},\delta}_{t_{k,0},t_{K,J}}\big\}$$

so that

$$0\leq\bar{V}^{\delta}_{t_k}-\bar{V}^{\Delta,\delta}_{t_k}\leq\mathbb{P}\text{-esssup}\Big\{\mathbb{E}\left[\left|f(\tau,\bar{X}^{\delta}_{\tau})-f(\tau^{\Delta},\bar{X}^{\delta}_{\tau^{\Delta}})\right|\left|\mathcal{F}_{t_k}\right.\right],\ \tau\in\mathcal{T}^{\mathcal{F},\delta}_{t_{k,0},t_{K,J}}\Big\}$$

which implies, owing to conditional Jensen's inequality and elementary properties of $\mathbb{P}$- essential supremum,

$$\begin{aligned}0\leq(\bar{V}^{\delta}_{t_k}-\bar{V}^{\Delta,\delta}_{t_k})^2 &\leq \mathbb{P}\text{-esssup}\Big\{\mathbb{E}\left[\left|f(\tau,\bar{X}^{\delta}_{\tau})-f(\tau^{\Delta},\bar{X}^{\delta}_{\tau^{\Delta}})\right|^2\left|\mathcal{F}_{t_k}\right.\right],\ \tau\in\mathcal{T}^{\mathcal{F},\delta}_{t_{k,0},t_{K,J}}\Big\}\\ &\leq \mathbb{E}\Big[\sup_{s,t\in[0,T],\,0\leq t-s\leq\Delta t}\left|f(t,\bar{X}^{\delta}_t)-f(s,\bar{X}^{\delta}_s)\right|^2\ \left|\mathcal{F}_{t_k}\right.\Big].\end{aligned}$$

Taking expectations and unsung that f is globally Lipschitz continuous in (t,x) yields

$$\begin{aligned}0\leq\mathbb{E}\,(\bar{V}^{\delta}_{t_k}-\bar{V}^{\Delta,\delta}_{t_k})^2 &\leq \mathbb{E}\Big[\sup_{s,t\in[0,T],\,0\leq t-s\leq\Delta t}\left|f(t,\bar{X}^{\delta}_t)-f(s,\bar{X}^{\delta}_s)\right|^2\Big]\\ &\leq [f]^2_{\mathrm{Lip}}\mathbb{E}\Big[\sup_{s,t\in[0,T],\,0\leq t-s\leq\Delta t}\left|\bar{X}^{\delta}_t-\bar{X}^{\delta}_s\right|^2\Big]\\ &\leq C_{b,\sigma,T}[f]^2_{\mathrm{Lip}}\Delta t\end{aligned}$$

where the real constant $C_{b,\sigma,T}$ only depends on $[b]_{\text{Lip}}$, $[\sigma]_{\text{Lip}}$ and T. Equivalently, this reads

$$\left\|\bar{V}^{\delta}_{t_{k+1}} - \bar{V}^{\Delta,\delta}_{t_{k+1}}\right\|_2 \leq C_{b,\sigma,T}[f]_{\text{Lip}}\sqrt{\Delta t}.$$

On the other hand, by Proposition 5(a) (see Equation (3.51)), we have that

$$\left\|\bar{V}^{\Delta,\delta}_{k+1} - \bar{V}^{n}_{t_{k+1}}\right\|_2 \leq [f]_{\text{Lip}}\Big(1+\frac{\Delta t}{T}\Big)^{\frac{1}{2}} \frac{(C'\Delta t)^{\frac{n}{2}}}{\sqrt{(n+1)!}} e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}}.$$

Collecting all these results and plugging them in (3.67) yields

$$\mathbb{E}\left[\max_{j=0:J}|\bar{V}^{\delta}_{t_{k,j}} - \widetilde{V}^{\delta,n}_{t_{k,j}}|^2\right] \leq 4[f]^2_{\text{Lip}}\left[\binom{k}{n}(C\Delta t)^{2n+1} + 2\,\frac{(C'\Delta t)^n}{(n+1)!} + 2\,C_{b,\sigma,T}\Delta t\right].$$

Consequently for Δt small enough (such that $\max(C,C')\Delta t < 1$), one has (independently of n)

$$\left\|\max_{j=0:J}|\bar{V}^{\delta}_{t_{k,j}} - \widetilde{V}^{\delta,n}_{t_{k,j}}|\right\|_2 \leq C''[f]_{\text{Lip}}\Delta t. \qquad \square$$

The proof, rather technical, is deferred to an Appendix at the end of the paper. Collecting the results established in Proposition 5(b) and 7, we derive the following theorem which shows the the convergence of the parareal procedure though we are not able to highlight the parareal speeding up of the procedure.

Theorem 10 (TLPRAOA)**.**

$$\max_{k=0:K}\left\|\hat{V}^{n+1}_{t_k} - \bar{V}^{\delta}_{t_k}\right\|_2 \leq [f]_{\text{Lip}}C\sqrt{\Delta t}$$

Proof. Starting from the recursion $\hat{V}^{n+1}_k = \bar{V}^{n+1}_{t_k} + \widetilde{V}^{\delta,n}_{t_{k,0}} - \bar{V}^{n}_{t_k}$ in the algorithm TLPRAOA, we note by the triangle inequality that

$$\begin{aligned}
\left\|\hat{V}^{n+1}_{t_k} - \bar{V}^{\delta}_{t_k}\right\|_2 &\leq \left\|\bar{V}^{n+1}_{t_k} - \widetilde{V}^{\delta,\Delta}_{t_k}\right\|_2 + \left\|\bar{V}^{\delta,n}_{t_{k,0}} - \bar{V}^{\delta}_{t_k}\right\|_2 + \left\|\bar{V}^{n}_{t_k} - \widetilde{V}^{\delta,\Delta}_{t_k}\right\|_2 \\
&\leq \frac{(C''\Delta t)^{\frac{n}{2}}}{\sqrt{(n+2)!}}\Big(1+\frac{\Delta t}{T}\Big)e^{-\frac{n(n+1)}{4}\frac{\Delta t}{T}} + [f]_{\text{Lip}}C\sqrt{\Delta t} \\
&\quad + \frac{(C''\Delta t)^{\frac{n-1}{2}}}{\sqrt{(n+1)!}}\Big(1+\frac{\Delta t}{T}\Big)e^{-\frac{n(n-1)}{4}\frac{\Delta t}{T}} \\
&\leq C_{b,\sigma,T}[f]_{\text{Lip}}\Delta t
\end{aligned}$$

if Δt is small enough. $\square$

3.6 Numerical Tests

3.6.1 Convergence of the Parareal Algorithm for a SDE

We consider the Black Scholes SDE with $\sigma(x,t) = \sigma_0 x$, $b(x,t) = rx$ with $r = 5\%$, $\sigma_0 = 20\%$, $t \in [0,T]$, $T = 3$ and $X_0 = 1$. We choose a fine grid with time step $\delta = T/32$ but the time step of the coarse grid is varying between $\Delta t \in [0.187, 1.5]$. We displayed the results of the parareal algorithm with respect to the number of iterations in table 3.1. On the figure 3.2, we plotted the simulation paths at different iterations and the fine solution. The solution at the fourth iteration is overlapping the fine solution.

Time-step			Absolute error			
K	J	Δt	$n=1$	$n=2$	$n=3$	$n=4$
2	16	1.500	4.458e−01	3.211e−03	1.599e−05	5.183e−08
4	8	0.750	5.201e−01	4.403e−03	1.691e−05	4.428e−08
8	4	0.375	4.780e−01	3.741e−03	2.022e−05	6.217e−08
16	2	0.187	4.442e−01	3.201e−03	1.643e−04	5.261e−08

Table 3.1: Absolute error from the approximation of the solution Black Scholes SDE in $T = 3$ computed on the fine grid by a sequential algorithm and the same computed using the parareal iterative algorithms 3.5 and 3.5 bis. The coarse grid has K intervals; the coarse time step is $\Delta t/K$; the fine grid has a fixed number of points hence each interval $(t_k, t_{t_{k+1}})$ it has J sub-intervals.

3.6.2 Convergence of the Parareal Algorithm For an American Contract

The payoff is $f(t,x) = e^{r(T-t)}(\kappa - x)^+$ with $X_0 = 36,\ r = 6\%,\ \kappa = 40,\ T = 2$. We have chosen $M = 100\ 000$ as in Longstaff and Schwartz [124]. The interpolation used in the LSMC is on the basis $\{1,\ x,\ x^2\}$, i.e. $P = 2$. The American payoff is then 4.838 at an early exercise $\tau = 0.634$.

3.6.2.1 The Black Scholes Case

Here the underlying asset is given by the Black Scholes SDE, $\sigma(x,t) = \sigma_0 x$, $b(x,t) = rx$. In the test $\sigma_0 = 20\%$. We have chosen a fine grid with $\delta t = T/32$. The free parameters are Δt which governs the number of points on the coarse grid and n the number of parareal algorithm. The error between the American payoff computed on the fine grid by LSMC and the same computed by the parareal algorithm is displayed on Table 3.2 for both algorithms 3.5 and 3.5bis.

Time-step			Absolute error			
K	J	Δt	$n=1$	$n=2$	$n=3$	$n=4$
2	16	0.667	6.033e−01	1.523e−01	1.711e−02	8.336e−04
4	8	0.400	2.374e−01	4.377e−02	2.178e−03	7.251e−04
8	4	0.222	8.548e−02	1.562e−02	7.350e−04	5.153e−04
16	2	0.118	2.574e−02	1.205e−03	4.395e−04	2.622e−04
2	16	0.667	5.912e−01	1.434e−01	4.183e−02	4.147e−02
4	8	0.400	2.245e−01	7.437e−02	2.250e−02	2.243e−02
8	4	0.222	7.409e−02	2.054e−02	7.217e−03	7.206e−03
16	2	0.118	1.947e−02	2.175e−03	2.159e−03	2.150e−03

Table 3.2: Absolute error from the American payoff computed on the fine grid by a sequential LSMC standard algorithm and the same computed using the parareal iterative algorithms 3.5 and 3.5 bis. The coarse grid has K intervals; the coarse time step is $\Delta t/K$; the fine grid has a fixed number of points hence each interval $(t_k, t_{t_{k+1}})$ it has J sub-intervals. The top 4 lines of numbers corresponds to Algorithm 3.5 while the last 4 lines correspond to Algorithm 3.5 bis for which a partial convergence estimate can be obtained.

The same information about convergence is now displayed in the two graphs on figure 3.3 for the errors versus Δt and the errors versus n. We couldn't decrease Δt to smaller values because the computing time becomes too large.

3.6.2.2 The Constant Elasticity of Variance Case

The diffusion coefficient is now a function of price as in Cox and Ross [46]: $\sigma(x,t) = \sigma_0 x^{0.7}$ (i.e. the volatility itself is given by $\sigma_0 x^{-0.3}$. All parameters have the same values as above. The results are shown on figure 3.4 and 3.3.

Time-step			Absolute error			
K	J	Δt	$n=1$	$n=2$	$n=3$	$n=4$
2	16	0.667	8.707e−01	1.379e−01	1.144e−02	1.123e−03
4	8	0.400	4.189e−01	6.509e−02	1.985e−03	7.568e−04
8	4	0.222	1.774e−01	2.561e−02	7.946e−04	4.836e−04
16	2	0.118	5.740e−02	5.083e−03	5.925e−04	3.041e−04

Table 3.3: Absolute error from an American payoff (CEV model) computed on the fine grid by a sequential LSMC standard algorithm and the same computed using the parareal iterative procedure 3.7.1. The coarse grid has K intervals; the coarse time step is $\Delta t/K$; the fine grid has a fixed number of points hence each interval $(t_k, t_{t_{k+1}})$ it has J sub-intervals.

3.6.3 Multilevel Parareal Algorithm

Consider the two-level case with $J = K$. As the fine grid has $O(K^2)$ intervals, a precision of $O(K)$ is expected. Estimate (3.58) reveals that the error between the direct solution on the fine grid and the parareal solution is $O(K)$ when $n > 1$. Hence, the order of magnitude of the computational cost of both methods is the same.

With a multilevel parareal method, the situation is the same as finite number of parareal iterations at each level is compatible with the error estimates, thence the computing cost is equivalent to a direct solution of the algorithm on the fine grid.

Therefore, the only criterion for the number of levels is the architecture of the computer. On a shared memory machine there is no reason to chose more than 2 levels. On GPU cards, the choice is led by the architecture of the memory banks so as their hierarchical heterogeneous structure. This of course makes sense only for large problems like an American contract on a complex basket like the CAC40.

3.6.3.1 Procedure

In this section, we describe a procedure to evaluate an American option with the multilevel parareal algorithm. We denote by V_k a realization of $V(t_k, X_{t_k})$, $k = 0, \ldots, K = \frac{T}{\Delta t}$; consider N refinements of each interval $(t_k, t_{t_{k+1}})_n$ by a uniform sub-partition of time step $\delta t_n = \frac{\Delta t_{n-1}}{J_n}$, for $J_n > 1$, $n \in 1, \ldots, N$. Then

$$[t_k, t_{k+1}]_n = \cup_{j=0}^{J_n - 1} [t_{k,j}, t_{k,j+1}]$$

with $t_{k,j+1} = t_{k,j} + \delta t_n$, $j = 0, \ldots, J_n - 1$, so that $t_k = t_{k,0} = t_{k-1,J_n}$. Denote by $\mathfrak{P} f$ the L^2-projection of f on the monomials $1, x, \ldots, x^P$.

Let $n = 0, \ldots, N-1$ be the level index of the parareal algorithm.

1. **(Initialization)** Generate N time discretization grid with $\Delta t = \delta t_0 > \delta t_1 > \ldots > \delta t_n$ so $\{Z_{k,j}^{m,n}\}_{k=1,\ldots,K, j=1,\ldots,J_n}^{m=1,\ldots,M, n=1,\ldots,N}$ for the M paths of the Monte Carlo method with the coarse and the others finer grid.

2. Compute recursively forward all Monte Carlo paths $\{\hat{X}^0_{t_k}(\omega^m)\}_{m=1}^M$ from $\hat{X}^0_0 = X_0$ by using (1.4) with $h_0 = \Delta t$ and then recursively backward

$$\hat{V}^0_k = \max\{f(t_k, \hat{X}^0_{t_k}), e^{-r\Delta t}\mathfrak{P}\,\mathbb{E}[\hat{V}^0_{k+1}|\hat{X}^0_k]\},\ k = K-1,\dots,0$$

from

$$\hat{V}^0_{K_0}(\omega^m) = e^{-rT} f(T, \hat{X}^0_T(\omega^m)), m = 1\dots, M.$$

3. Compute for $n = 1,\dots,N$ the following steps:

   (a) Compute for M simulation paths

      i. **(Forward loop)** for every $k = 0,\dots,K-1$:

         A. Compute the fine solution of the grid of the level n $\{\tilde{X}^{\delta,n}_{k,j}\}_{j=0}^{J_n}$ of (1.4) with refined step $h_n = \delta t_n = \frac{\delta t_{n-1}}{J_n}$, started at $t_{k,0} = t_k$ from $\hat{X}^n_{t_k}$.

         B. Compute the coarse solution of the grid at t_{k+1}: $\bar{X}^{\Delta}_{t_{k+1}} = \hat{X}^{n+1}_{t_k} + b(t_k, \hat{X}^{n+1}_{t_k})\Delta t + \sigma(t_k, \hat{X}^{n+1}_{t_k})\Delta W_k$.

         C. Set $\hat{X}^{n+1}_{t_{k+1}} = \bar{X}^{\Delta}_{t_{k+1}} + \tilde{X}^{\delta,n}_{t_{k,J_n}} - \hat{X}^n_{t_{k+1}}$.

   (b) **(Initialization)**: Compute $\bar{V}^{n+1}_K = \hat{V}^{n+1}_K = f(T, \hat{X}^{n+1}_T)$

   (c) **(Backward loop)** for $k = K-1,\dots,0$:

      i. On each (t_k, t_{k+1}), from $\tilde{V}^{\delta,n}_{k,J_n} = \mathfrak{P}\,\mathbb{E}(\hat{V}^n_{k+1} \mid \tilde{X}^{\delta,n}_{k,J_n})$, compute by a backward loop in j

$$\tilde{V}^{\delta,n}_{k,j} = \max\left\{f(t_{k,j}, \tilde{X}^{\delta,n}_{t_{k,j}}), e^{-r\delta t}\mathfrak{P}\,\mathbb{E}[\tilde{V}^{\delta,n}_{k,j+1}|\tilde{X}^{\delta,n}_{t_{k,j}}]\right\},\ j = J_n - 1,\dots,0.$$

      ii. Compute $\bar{V}^{n+1}_k = \max\left\{f(t_k, \hat{X}^{n+1}_{t_k}), e^{-r\Delta t}\mathfrak{P}\,\mathbb{E}[\bar{V}_{k+1}|\hat{X}^{n+1}_{t_k}]\right\}$.

      iii. Set $\hat{V}^{n+1}_k = \bar{V}^{n+1}_k + \tilde{V}^{\delta,n}_k - \hat{V}^n_k$.

3.6.3.2 Results

The previous construction being recursive one can again apply the two-levels parareal algorithm to LSMC on each interval $[t_k, t_{t_{k+1}}]$. The problem of finding the optimal strategy for parallelism and computing time is complex, because of there are so many parameters; in what follows the number of levels is $L = 4$; furthermore, when an interval with $J+1$ points is divided into sub-intervals each is endowed with a partition using $J+1$ points as well. So if the coarse grid has K intervals, the 4^{th} grid has K^4 intervals. The results are compared with the reference value of Longstaff Schwartz, 4.838, and shown on table 3.4 and figure 3.5.

The number of parareal iterations is 4 and the error is displayed at each n. We have also carried out some tests with sub-partitions using $J \neq K$. Thus each level has its own number of points per intervals, J_l. These errors are also shown on Figure 3.5 for $n = 2$. It seems to be $O(K^4)$ for K small and $O(K^2)$ for K bigger. The method was implemented in parallel; each interval is allocated to a processor, at each level in a round-robin order. Almost perfect parallelism is obtained in our tests on a machine with 32 processors, as shown on Figure 3.6. In appendix C, a details about the implementation.

3.7 The Iterative Parareal Method and Sensitivities

In this section we will show that it is also possible to compute firstly the sensitivities of an American option with an automatic differentiation program with a small effort. We present a short procedure to compute the Delta of an American option.

Time-step		Absolute error			
K	K^4	$n=1$	$n=2$	$n=3$	$n=4$
2	16	3.533e−01	1.181e−01	4.776e−02	2.761e−02
3	81	2.089e−01	3.906e−02	2.252e−02	1.866e−02
4	256	1.416e−01	2.597e−02	1.925e−02	1.364e−02
5	625	1.051e−01	2.202e−02	1.797e−02	1.297e−02

Table 3.4: Absolute error between the computed payoff with the multilevels parareal method and the reference value of Longstaff Schwartz. The number of levels is L=4, each level is subdivided into K intervals; K^4 is the number of intervals at the deepest level.

Time-step					Absolute-error			
J_1	J_2	J_3	J_4	Total	$n=1$	$n=2$	$n=3$	$n=4$
6	5	4	3	360	1.085e−01	3.056e−02	2.020e−02	1.420e−02
3	4	5	6	360	3.536e−01	3.167e−02	1;674e−02	1.351e−02
20	2	2	2	160	2.312e−02	1.637e−02	1.556e−02	1.331e−02
2	20	2	2	160	3.544e−01	8.350e−02	2.312e−02	1.217e−02
2	2	20	2	160	3.518e−01	1.152e−01	1.582e−02	1.373e−02
2	2	2	20	160	3.551e−01	1.195e−01	4.447e−02	1.102e−02

Table 3.5: Absolute error between the computed payoff with the multilevel parareal method and the reference value of Longstaff Schwartz. There are $L=4$ levels; at level $l-1$ to obtain level l each interval is divided into J_l intervals. The errors are given versus the number of parareal iterations $n=1,\ 2,\ 3,\ 4$. Note that all subdivisions give more or less the same precision; computationally and for parallelism the last one is the best.

3.7.1 Procedure

We denote by V_k a realization of $V(t_k, X_{t_k}),\ k=0,\dots,K=\frac{T}{\Delta t}$; consider a refinement of each interval $(t_k, t_{t_{k+1}})$ by a uniform sub-partition of time step $\delta t=\frac{\Delta t}{J}$, for some integer $J>1$.Then

$[t_k, t_{k+1}]=\cup_{j=0}^{J-1}[t_{k,j}, t_{k,j+1}]$ with $t_{k,j+1}=t_{k,j}+\delta t,\ j=0,\dots,J-1,$ so that $t_k=t_{k,0}=t_{k-1,J}$.

Denote by $\mathfrak{P}f$ the L^2-projection of f on the monomials $1, x, \dots, x^P$.

Let $n=0,\dots,N-1$ be the iteration index of the parareal algorithm.

1. **(Initialization)** Generate $\{Z_{k,j}^m\}_{k=1,\dots,K,j=1,\dots,J}^{m=1,\dots,M}$ for the M paths of the Monte Carlo method with the coarse and fine mesh.
2. Compute recursively forward all Monte Carlo paths $\{\hat{X}_{t_k}^0(\omega^m)\}_{m=1}^M$ from $\hat{X}_0^0=X_0$ by using (1.4) with $h=\Delta t$ and then recursively backward

$$\hat{V}_K^0(\omega^m)=e^{-rT}f(T,\hat{X}_T^0(\omega^m)),\quad m=1\dots,M.$$

from

$$\hat{V}_k^0=\max\{f(t_k,\hat{X}_{t_k}^0), e^{-r\Delta t}\mathfrak{P}\ \ \mathbb{E}[\hat{V}_{k+1}^0|\hat{X}_{t_k}^0]\},\quad k=K-1,\dots,0$$

   (a) Compute for $n=0,\dots,N-1$ the following steps

      i. Compute for all M paths

A. **(Forward loop)** for every k=0,...,K-1

B. Compute the fine grid solution $\{\tilde{X}^{\delta,n}_{t_{k,j}}\}_{j=0}^{J}$ of (1.4) with refined step $h = \delta t = \frac{\Delta t}{J}$, started at $t_{k,0} = t_k$ from $\hat{X}^n_{t_k}$.

C. Compute the coarse grid solution at t_{k+1}:

$$\bar{X}^{\Delta}_{t_{k+1}} = \hat{X}^{n+1}_{t_k} + b(t_k, \hat{X}^{n+1}_{t_k})\Delta t + \sigma(t_k, \hat{X}^{n+1}_{t_k})\Delta W_k$$

.

D. Set $\hat{X}^{n+1}_{t_{k+1}} = \bar{X}^{\Delta}_{t_{k+1}} + \tilde{X}^{\delta,n}_{t_{k,J}} - \hat{X}^n_{t_{k+1}}$.

(b) **(Initialization)** Compute $\bar{V}^{n+1}_K = \hat{V}^{n+1}_K = f(T, \hat{X}^{n+1}_T)$

i. **(Backward loop)** for every $k = K-1, \ldots, 0$:

ii. n each (t_k, t_{k+1}), from $\tilde{V}^{\delta,n}_{k,J} = \mathfrak{P}\ \ \mathbb{E}(\bar{V}^n_{k+1} \ \mid \ \tilde{X}^{\delta,n}_{k,J})$, compute by a backward loop in j

$$\tilde{V}^{\delta,n}_{k,j} = \max\left\{f(t_{k,j}, \tilde{X}^{\delta,n}_{t_{k,j}}), e^{-r\delta t}\mathfrak{P}\ \ \mathbb{E}[\tilde{V}^{\delta,n}_{k,j+1}|\tilde{X}^{\delta,n}_{t_{k,j}}]\right\}, \quad j = J-1, \ldots, 0.$$

iii. Compute $\bar{V}^{n+1}_k = \max\left\{f(t_k, \hat{X}^{n+1}_{t_k}), e^{-r\Delta t}\mathfrak{P}\ \ \mathbb{E}[\hat{V}^{n+1}_{k+1}|\hat{X}^{n+1}_{t_k}]\right\}$.

iv. Set $\hat{V}^{n+1}_k = \bar{V}^{n+1}_k + \tilde{V}^{\delta,n}_{k,0} - \bar{V}^n_k$.

v. Compute $\left(\dfrac{\partial \hat{V}^{n+1}_k}{\partial X_0}\right)$ by AD on the program that implements

Remark 18. *On can use either forward or reverse automatic differentiation technique. We choose to use a forward method for our numerical experiment.*

3.7.2 The Delta in the Black Scholes Model

As done in section 3.6.2.1, the underlying asset is given by the Black Scholes SDE, $\sigma(x,t) = \sigma_0 x$, $b(x,t) = rx$. In the test the volatility is $\sigma_0 = 20\%$. We have chosen a fine grid with $\delta t = T/32$. The free parameters are Δt which governs the number of points on the coarse grid and n the number of parareal algorithm. The error between the Delta of an American payoff computed on the fine grid by LSMC and the same computed by the parareal algorithm is displayed on Table 3.6 the procedure given in the section 3.7.1.

Time-step			Absolute error			
K	J	Δt	$n=1$	$n=2$	$n=3$	$n=4$
2	16	0.667	1.143e−01	3.527e−02	8.066e−03	4.063e−03
4	8	0.400	5.790e−02	2.792e−02	1.266e−03	6.290e−04
8	4	0.222	2.473e−02	1.198e−02	5.753e−04	2.185e−04
16	2	0.118	8.058e−03	1.718e−03	2.999e−04	1.462e−04

Table 3.6: Absolute error from the Delta of an American payoff (BS model) computed on the fine grid by a sequential LSMC standard algorithm and the same computed using the parareal iterative procedure 3.7.1. The coarse grid has K intervals; the coarse time step is $\Delta t/K$; the fine grid has a fixed number of points hence each interval $(t_k, t_{t_{k+1}})$ it has J sub-intervals.

Time-step			Absolute error			
K	J	Δt	$n=1$	$n=2$	$n=3$	$n=4$
2	16	0.667	2.500e−01	1.051e−02	5.390e−03	6.347e−04
4	8	0.400	7.132e−02	3.242e−02	2.965e−03	3.124e−04
8	4	0.222	1.548e−02	7.083e−03	1.233e−03	2.001e−04
16	2	0.118	3.331e−03	1.237e−03	2.278e−04	8.109e−05

Table 3.7: Absolute error from the Delta of an American payoff (CEV model) computed on the fine grid by a sequential LSMC standard algorithm and the same computed using the parareal iterative procedure 3.7.1. The coarse grid has K intervals; the coarse time step is $\Delta t/K$; the fine grid has a fixed number of points hence each interval $(t_k, t_{t_{k+1}})$ it has J sub-intervals.

3.7.3 The Delta in the Constant Elasticity of Variance Model

The volatility is now a function of price as in Cox and Ross [46]: $\sigma(x,t) = \sigma_0 x^{0.7}$. All parameters have the same values as above. The results are shown on figure 3.8 and table 3.7.

3.8 Conclusion

In this work, we established a new numerical scheme for the approximation of the American option and for its sensitivity. We proved theoretically and numerically its effectiveness for different type of model. This scheme permits an efficient parallelization in time of a backward dynamic programming principle with having the same precision than a sequential scheme with a reduced time of execution from start to finish, but increased cost in terms of number of floating point operations.

Current developments are focused on the convergence of the multilevel parareal algorithm. We, also, have some ideas about future work; application of the parareal method to the dual algorithm for the approximation of the lower/upper bound of an American option and studying the multidimensional case. One can imagine a *multilevel* algorithm in the sense of the *multilevel* Monte Carlo of Giles [60] i.e. having a different number of simulations per iteration or an adaptive parareal scheme with different subdivisions in each intervals.

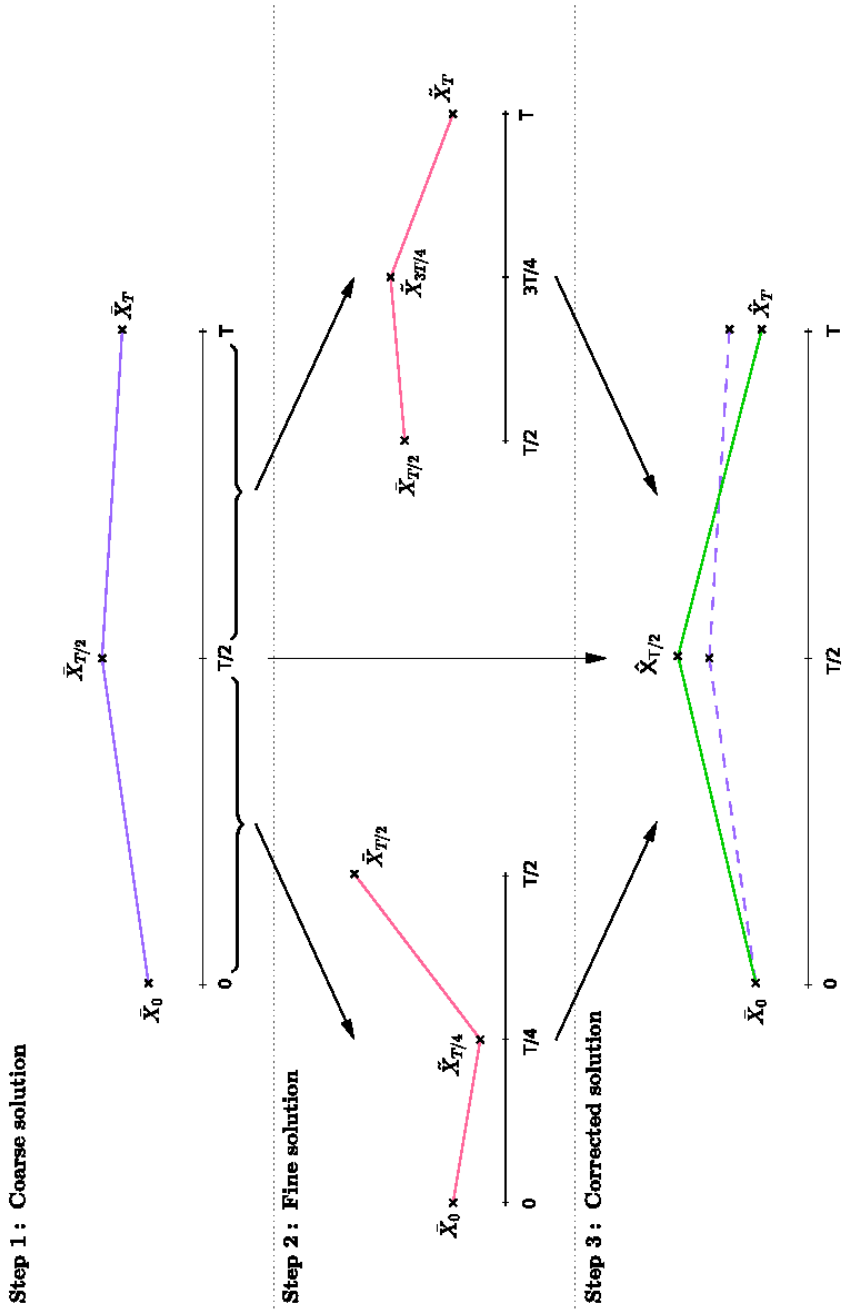

Figure 3.1: Schematic representation of the two level parareal algorithm.

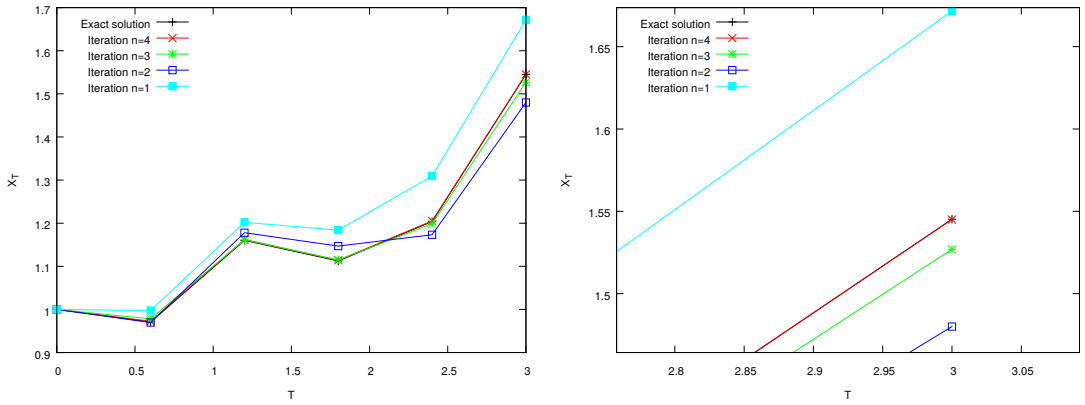

Figure 3.2: Simulation paths of an SDE at different parareal iterations (Black Scholes model). On the right, the simulation paths between $t = 0$ and $t = T = 3$. On the left, the same simulation paths but zoomed in $t = T$.

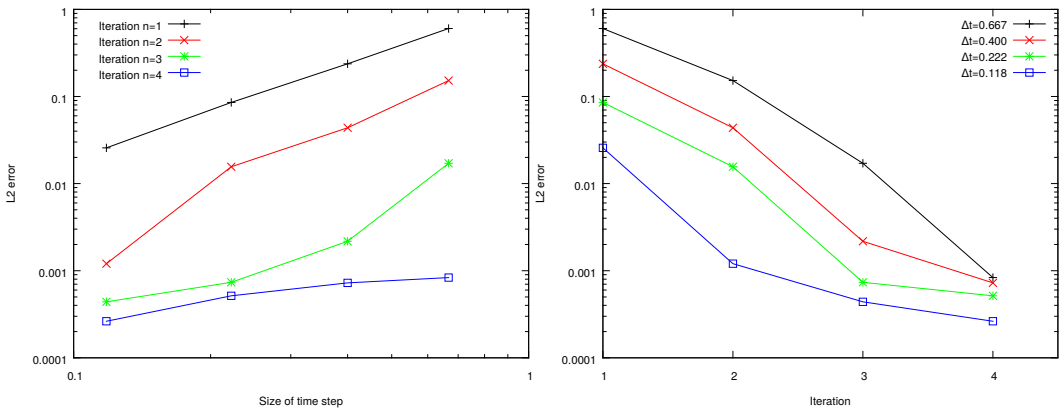

Figure 3.3: Black Scholes case: Errors on the payoff versus Δt on the left for several values of n and versus n on the right for several values of Δt. Both graphs are for Algorithm 3.5 in log-log scales and indicate a general behavior of the error ε not incompatible with (9).

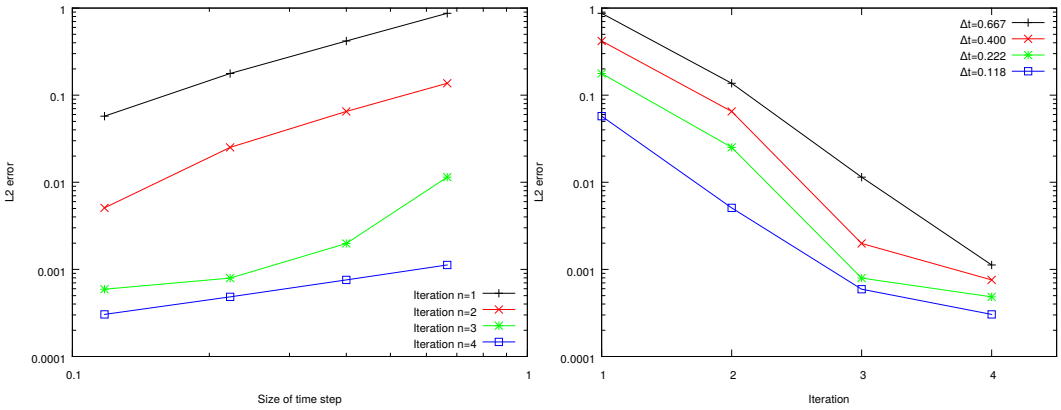

Figure 3.4: Constant Elasticity case: Errors on the payoff versus Δt on the left for several values of n and versus n on the right for several values of Δt. Both graphs are for Algorithm 3.5 in log-log scales and indicate a general behavior of the error ε not incompatible with (9).

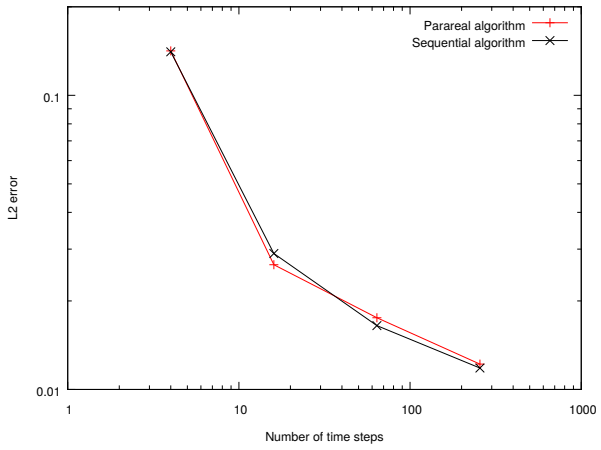

Figure 3.5: Comparison between a standard LSMC solution and the parareal solution for the same number of time intervals at the finest level. The 4 points have respectively 1, 2, 3, 4 levels; the first data point has one level and 4 intervals, the second has 2 levels and 16 intervals, the third 3 levels and 64 intervals, the fourth 4 levels and 256 intervals . The total number of time steps is on the horizontal axis, in log scale and the error at $n = 2$ is on the vertical axis in log scale as well.

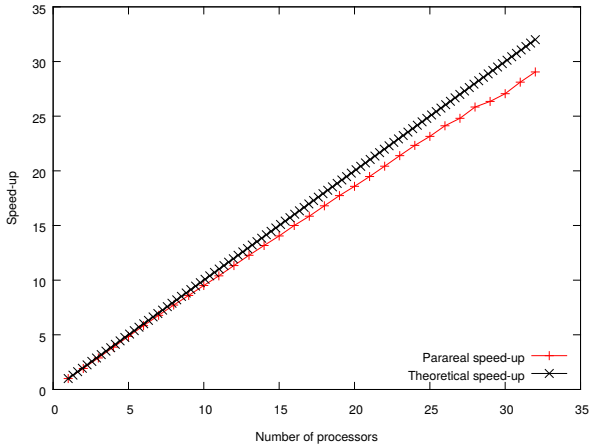

Figure 3.6: Speed-up versus the number of processors, i.e. the parareal CPU time on a parallel machine divided by the parareal CPU time on the same machine but running on one processor. There are two levels only; the parameters are $K = 1, 2.., 32$, $n = 2$ and $J = 100$ so as to keep each processor fully busy.

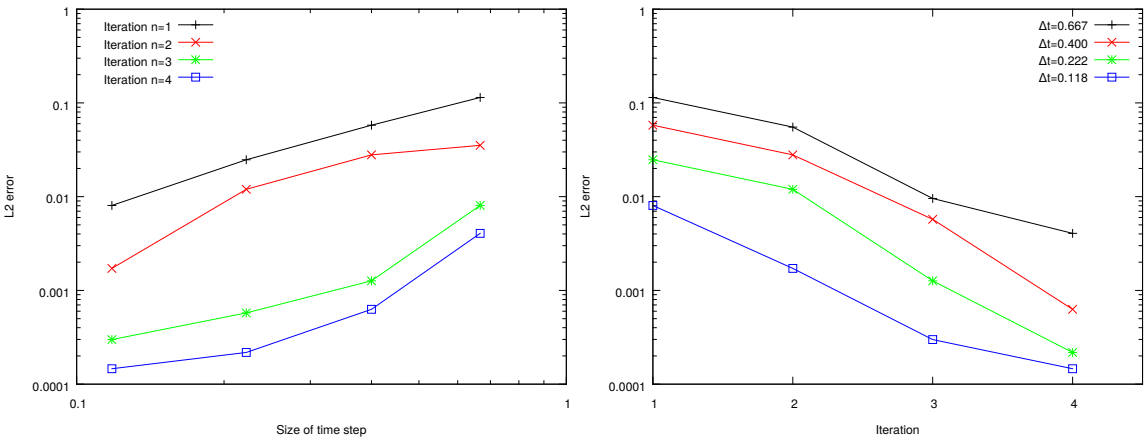

Figure 3.7: Black Scholes case: Errors on the Delta versus Δt on the left for several values of n and versus n on the right for several values of Δt. Both graphs are for procedure 3.7.1 in log-log scales and indicate a general behavior of the error ε not incompatible with (9).

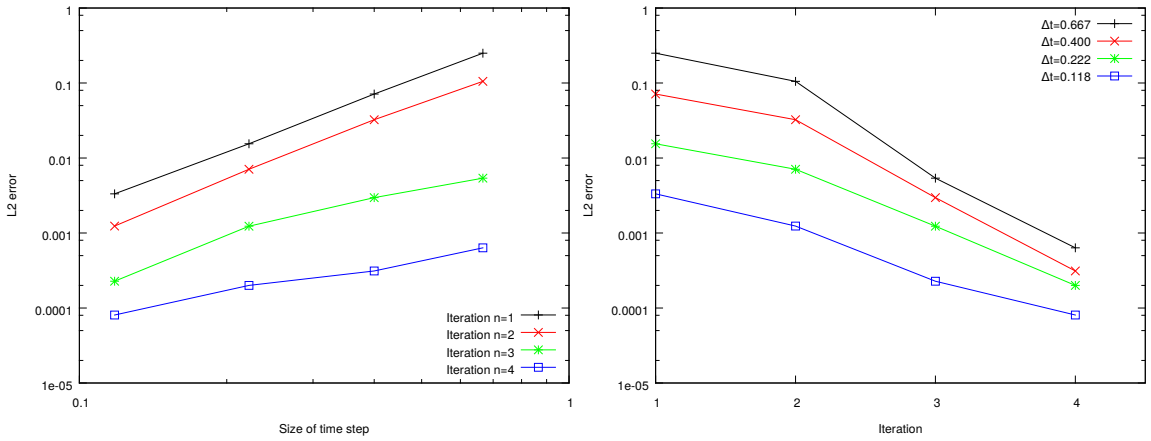

Figure 3.8: Constant Elasticity of Variance case: Errors on the Delta versus Δt on the left for several values of n and versus n on the right for several values of Δt. Both graphs are for procedure 3.7.1 in log-log scales and indicate a general behavior of the error ε not incompatible with (9).

Chapitre 4

Approximation of a (F)BSDE with a Parareal Method

"The beauty in mathematical structures, however, cannot be appreciated without understanding of a group of numerical formulas that express laws of logic"
extract from K. Itô's speech marking his Kyoto prize in 1998.

– K. Itô, Mathematician, (1915, 2008).

This chapter is an extension of the previous chapter. We make an application of the parareal method for the approximation of a backward stochastic differential equation with an application to the pricing of a Bid-Ask Spread option for interest rates. The Least squares Monte Carlo method is a popular choice to approximate the solution for these equations. The discretization scheme is quite complex because there is at least two conditional expectations to evaluate. The use of the parareal algorithm seems straightforward for solving backward stochastic differential equations.

This chapter is a part of an article in preparation.

4.1 Introduction

In this chapter we are interested in nonlinear option pricing and backward stochastic differential equation (BSDE), more precisely to the evaluation of a Call Spread option via the approximation of a decoupled forward backward stochastic differential equation (FBSDE). Backward stochastic differential equations were first introduced by Bismut in [21] where he treated the linear case, the nonlinear case has been introduced by Bismut in [22] and later the general case was studied by Pardoux and Peng in [148]. Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a given probability space endowed with a one dimensional standard Brownian motion W with its natural augmented filtration $(\mathscr{F}_t)_{0\leq t\leq T}$. Following the notations of these authors, the solution of backward stochastic differential equation is a couple of variables $(Y_t, Z_t)_{t\in[0,T]}$, which is a par of adapted process satisfying:

$$\begin{cases} -dY_t = f(t, Y_t, Z_t)dt - Z_t dW_t, \\ \quad Y_T = \xi, \end{cases} \tag{4.1}$$

where $f : [0,T] \times \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ is the generation function of the BSDE, ξ is the squares-integrable terminal condition and a $\mathscr{F}_T$-measurable random variable. $(Y_t)_{t\in[0,T]}$ is the dynamic of the value of the replicating portfolio and $(Z_t)_{t\in[0,T]}$ corresponds to the value of the hedging portfolio and is a $\mathscr{F}_T$-predictable process.

In [156], Quenez shown that these types of equations appear often in finance and there is a wide literature devoted to BSDEs El Karoui et al [107] and [108], Pardoux and Răşcanu [147], Ma and Yong [128], Zhang [183] and Peng [149], [146] and [150]. For their applications to finance, we refer the interested lecturer to El Karoui et al. [110] and [109], Pham [153].

Indeed, the price of a contingent claims can be expressed in terms of backward stochastic differential equations in a complete and incomplete market. But one can not always construct a replicating portfolio in an incomplete market environment at maturity T, thus the price can not be determined by no-arbitrage arguments. However, it is possible to determine the upper price of the contingent claim considering superstrategies (dual characterization) which are solutions of nonlinear BSDEs.

In a complete market, it is always possible to construct a replicating portfolio which have the same value than the contingent claim at maturity T but there is an infinite number of portfolios so the price is not well defined. One great aspect with the usage of BSDEs and its theory is that the problem is well defined and so there exist a unique price and hedging portfolio.

This chapter is organized as follows; in section 4.2 we begin by a short description of a decoupled forward backward differential equation and give the numerical scheme established by Zhang [184], Bouchard and Touzi [26]. Also, we give the algorithm for the two level parareal method applied to this equation.

The section 4.3 is dedicated to numerical results of the approximation of the decoupled forward backward differential equations for the evaluation of a Bid-Ask Spread option for interest rates. We recall the payoff of the financial underlying, analyze the convergence of the results and time of computation.

We conclude by summarizing the work accomplished and give some perspective for future work.

4.2 Nonlinear Option Pricing

4.2.1 Decoupled Forward Backward Stochastic Differential Equation

We are concerned with (one dimensional) decoupled forward backward stochastic differential equation (FBSDE) and with their applications to finance. Forward backward stochastic differential equations and their approximations have been widely studied by Ma and Yong [128], Milstein and Tretyakov [132], Ma et al. [127] and [126], Douglas et al. [51], Bouchard and Elie [25], Chevance [42] and Villalobos [180]. The pricing of Call Spread option could be boiled down to solving a forward backward SDE (i.e. next section 4.3.1). A solution for the decoupled forward backward SDE associated with (f, φ, b, σ) is a tripled of adapted stochastic processes $(X_t, Y_t, Z_t)_{t\in[0,T]}$ such that:

$$\begin{cases} -dY_t = f(t, X_t, Y_t, Z_t)dt - Z_t dW_t, \\ Y_T = \varphi(X_T), \\ dX_t = b(t, X_t) + \sigma(t, X_t)dW_t, \\ X_0 = x, \end{cases} \tag{4.2}$$

where $f : [0,T] \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$, $b : [0,T] \times \mathbb{R} \mapsto \mathbb{R}$, $\sigma : [0,T] \times \mathbb{R}$, $\varphi : \mathbb{R} \mapsto \mathbb{R}$ is the payoff function of a contingent claim and W is a standard Brownian motion. The tripled of adapted stochastic processes (X, Y, Z) are real valued in $\mathbb{R}^3$. Equivalently,

$$\begin{cases} Y_t = \varphi(X_T) + \displaystyle\int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s, \\ X_t = x + \displaystyle\int_0^t b(s, X_s)ds + \int_0^t \sigma(s, X_s)dW_s. \end{cases} \tag{4.3}$$

In order to solve the FBSDE 4.2, we adopt the numerical scheme proposed by Zhang [184] (see Bouchard and Touzi [26] for a detailed analysis). We consider a partition $\pi = \{t_0, \ldots, t_N\}$ for the time discretization of the interval $[0,T]$, i.e. $0 = t_0 < t_1 < \ldots < t_n = T$. For the approximation of the process X, we just apply a natural Euler scheme and for the processes Y and Z we have the following backward scheme:

$$\begin{cases} \bar{Y}_{t_n} = \varphi(\bar{X}_{t_n}), \qquad \bar{Z}_{t_n} = 0 \\ \bar{Z}_{t_i} = \dfrac{1}{\Delta t_i}\mathbb{E}\left[\Delta W_{t_i} \bar{Y}_{t_{i+1}} \mid X_{t_i}\right], \quad i = n-1, \ldots, 0 \\ \bar{Y}_{t_i} = \mathbb{E}\left[\bar{Y}_{t_{i+1}} - f(t_i, \bar{X}_{t_i}, \bar{Y}_{t_{i+1}}, \bar{Z}_{t_i})\Delta t_i \mid X_{t_i}\right], \quad i = n-1, \ldots, 0. \end{cases} \tag{4.4}$$

with $\Delta_{t_i} = t_{i+1} - t_i$ and the Brownian increment $\Delta W_{t_i} = W_{t_{i+1}} - W_{t_i}$. We have to turn this time discretization scheme into a viable numerical scheme. Most of the time, conditional expectation can not be computed explicitly and have to be replaced by an approximation technique. A special attention will be paid to the complexity of the chosen approximation in order to avoid time consuming algorithm and explosive costs. There exist several technique in the literature as approximation by trees see Briand et al. [28], cubature methods discussed by Crisan and Manolarakis [48], Quantization methods see Bally and Pagès [11], Non parametric kernel estimators studied by Bouchard and Touzi [26], finally regression based method proposed by Gobet et al. [76].

In [76], Gobet et al. suggested an approach based on the Least squares Monte Carlo to approximate the numerical scheme proposed by Zhang [184] and studied in Bender and Steiner [16]. As the two conditional expectations in 4.4 can not be evaluated explicitly, $\bar{Y}$ and $\bar{Z}$ are respectively approximated by linear combinations of finite sets of M and L real basis functions ψ and ζ:

$$\begin{aligned} \bar{Z}_{t_k} &= \mathbb{E}\left[\Delta W_{t_k} \bar{Y}_{t_{k+1}} \mid X_{t_k}\right] \\ &\simeq \sum_{i=1}^{M} \alpha_{k,i} \psi_{k,i}(\bar{X}_{t_k}), \end{aligned} \tag{4.5}$$

and

$$\begin{aligned} \bar{Y}_{t_k} &= \mathbb{E}\left[\bar{Y}_{t_{k+1}} - f(t_k, \bar{X}_{t_k}, \bar{Y}_{t_{k+1}}, \bar{Z}_{t_k})\Delta t_k \mid X_{t_k}\right] \\ &\simeq \sum_{i=1}^{L} \beta_{k,i} \zeta_{k,i}(\bar{X}_{t_k}). \end{aligned} \tag{4.6}$$

Where the $(\alpha_{k,i})_{i=1,\ldots,M}$ are computed by least squares,

$$\min_{\alpha} \left\{ \mathbb{E}\left[\left(\mathbb{E}\left[\Delta W_{t_k} \bar{Y}_{t_{k+1}} \mid X_{t_k}\right] - \sum_{i=1}^{M} \alpha_{k,i} \psi_{k,i}(\bar{X}_{t_k}) \right)^2 \right] \right\}, \tag{4.7}$$

and the $(\beta_{k,i})_{i=1,\dots,L}$,

$$\min_{\beta}\left\{\mathbb{E}\left[\left(\mathbb{E}\left[\bar{Y}_{t_{k+1}}-f(t_k,\bar{X}_{t_k},\bar{Y}_{t_{k+1}},\bar{Z}_{t_k})\Delta t_k \mid X_{t_k}\right]-\sum_{i=1}^{L}\alpha_{k,i}\beta_{k,i}(\bar{X}_{t_k})\right)^2\right]\right\}. \tag{4.8}$$

Which leads to a double Gram linear system, for $i=1,\dots,M$,

$$\sum_{j=1}^{M}\alpha_{k,i}\mathbf{Gram}\left\{\psi_{k,i}(\bar{X}_{t_k}),\psi_{k,j}(\bar{X}_{t_k})\right\}=\mathbb{E}\left[\mathbb{E}\left[\Delta W_{t_k}\bar{Y}_{t_{k+1}}\mid X_{t_k}\right]\psi_{k,i}(\bar{X}_{t_k})\right], \tag{4.9}$$

and for $i=1,\dots,L$,

$$\begin{aligned}\sum_{j=1}^{L}\beta_{k,i}\mathbf{Gram}&\left\{\zeta_{k,i}(\bar{X}_{t_k}),\zeta_{k,j}(\bar{X}_{t_k})\right\}\\&=\mathbb{E}\left[\mathbb{E}\left[\bar{Y}_{t_{k+1}}-f(t_k,\bar{X}_{t_k},\bar{Y}_{t_{k+1}},\bar{Z}_{t_k})\Delta t_k\mid X_{t_k}\right]\zeta_{k,i}(\bar{X}_{t_k})\right].\end{aligned} \tag{4.10}$$

Finally, then the discrete form of the decoupled FBSDE 4.2 could be derived by:

$$\begin{cases}\bar{X}_{t_0}=X_0,\\ \bar{X}_{t_{i+1}}=\bar{X}_{t_i}+b(t_i,X_{t_i})\Delta t_i+\sigma(t_i,X_{t_{i+1}})\Delta W_{t_i},\quad i=0,\dots,n-1,\\ \bar{Z}_{t_n}=0,\\ \bar{Z}_{t_i}=\dfrac{1}{\Delta t_i}\mathbb{E}\left[\Delta W_{t_i}\bar{Y}_{t_{i+1}}\mid X_{t_i}\right],\quad i=n-1,\dots,0\\ \bar{Y}_{t_n}=\varphi(\bar{X}_{t_n}),\\ \bar{Y}_{t_i}=\mathbb{E}\left[\bar{Y}_{t_{i+1}}-f(t_i,\bar{X}_{t_i},\bar{Y}_{t_{i+1}},\bar{Z}_{t_i})\Delta t_i\mid X_{t_i}\right],\quad i=n-1,\dots,0.\end{cases} \tag{4.11}$$

The conditional expectation $\mathbb{E}\left[\Delta W_{t_k}\bar{Y}_{t_{k+1}}\mid X_{t_k}\right]$ for the estimation of the term $\bar{Z}_{t_k}$ can be a critical aspect of this numerical scheme. Obviously, the variance will blow up as the time partition becomes finer due to the factor $\frac{\Delta W_{t_i}}{\Delta t_i}$. Furthermore, the approximation errors due to the estimation of two conditional expectations in 4.11 and over all the time steps may sum up which could result in a poor quality approximation. Taking a small Δt leads to poor convergence and to a large standard error in the estimates. However Alanko and Avellaneda [4] proposed a new numerical scheme to tackle this issue and in order to reduce variance in the numerical solution of BSDEs via a control variate technique.

Remark 19. *The motivation of applying the parareal method to the approximation of a backward stochastic differential equation comes from the numerical scheme in 4.11, the two conditional expectations to estimate is very time consuming and the idea to make them in parallel appears a good solution to tackle the problem. There is already several attempt to reduce the time of computation for the approximation of BSDEs see Labart and Lelong [116], Peng et al. [152] and [151] but it doesn't concern the critical aspect of the backward scheme which is the evaluation in several dates at the same time.*

4.2.2 The Two Level Parareal Procedure for a Decoupled FBSDE

In this section, we describe a procedure to approximate a decoupled forward backward stochastic differential equation given in 4.2 with the parareal method. We denote by (Y_k,Z_k) a realization

of $(Y(t_k, X_{t_k}), Z(t_k, X_{t_k}))$ $k = 0, \ldots, K = \frac{T}{\Delta t}$; consider a refinement of each interval $(t_k, t_{t_{k+1}})$ by a uniform sub-partition of time step $\delta t = \frac{\Delta t}{J}$, for $J > 1$. Then

$$[t_k, t_{k+1}]_n = \cup_{j=0}^{J-1} [t_{k,j}, t_{k,j+1}]$$

with $t_{k,j+1} = t_{k,j} + \delta t$, $j = 0, \ldots, J-1$, so that $t_k = t_{k,0} = t_{k-1,J}$. Denote by $\mathfrak{P} f$ the L^2-projection of f on the monomials $1, x, \ldots, x^P$.

Let $n = 0, \ldots, N-1$ be the iteration index of the parareal algorithm.

1. **(Initialization)** Generate o $\{U_{k,j}^m\}_{k=1,\ldots,K, j=1,\ldots,J}^{m=1,\ldots,M}$, $U \sim \mathcal{N}(0,1)$ for the M paths of the Monte Carlo method with the coarse and fine grid.
2. Compute recursively forward all Monte Carlo paths $\{\hat{X}_{t_k}^0(\omega^m)\}_{m=1}^M$ from $\hat{X}_0^0 = X_0$ by using (1.4) with $h_0 = \Delta t$ and then recursively backward

$$\hat{V}_k^0 = \max\{f(t_k, \hat{X}_{t_k}^0), e^{-r\Delta t} \mathfrak{P}\, \mathbb{E}[\hat{V}_{k+1}^0 | \hat{X}_k^0]\},\ k = K-1, \ldots, 0$$

from

$$\hat{V}_K^0(\omega^m) = e^{-rT} f(T, \hat{X}_T^0(\omega^m)), \quad m = 1 \ldots, M.$$

3. Compute for $n = 1, \ldots, N$ the following steps:
   (a) Compute for M simulation paths
      i. **(Forward loop)** for every $k = 0, \ldots, K-1$:
         A. Compute the fine solution of the grid of the level n $\{\tilde{X}_{k,j}^\delta\}_{j=0}^J$ of (1.4) with refined step $h = \delta t = \frac{\delta t}{J}$, started at $t_{k,0} = t_k$ from $\hat{X}_{t_k}^n$.
         B. Compute the coarse solution of the grid at t_{k+1}: $\bar{X}_{t_{k+1}}^\Delta = \hat{X}_{t_k}^{n+1} + b(t_k, \hat{X}_{t_k}^{n+1})\Delta t + \sigma(t_k, \hat{X}_{t_k}^{n+1})\Delta W_k$.
         C. Set $\hat{X}_{t_{k+1}}^{n+1} = \bar{X}_{t_{k+1}}^\Delta + \tilde{X}_{t_{k,J}}^{\delta,n} - \hat{X}_{t_{k+1}}^n$.
   (b) **(Initialization)**: Compute $\bar{Y}_K^{n+1} = \hat{Y}_K^{n+1} = \varphi(T, \hat{X}_T^{n+1})$ and set $\bar{Z}_K^{n+1} = 0$
   (c) **(Backward loop)** for $k = K-1, \ldots, 0$:
      i. On each (t_k, t_{k+1}), from

$$\tilde{Y}_{k,J}^\delta = \mathfrak{P}\, \mathbb{E}(\hat{Y}_{k+1}^n - f(t_k, \hat{X}_{k+1}^n, \hat{Y}_{k+1}^n, \hat{Z}_{k+1}^n)\Delta t \,|\, \tilde{X}_{k,J}^\delta)$$

      and

$$\tilde{Z}_{k,J}^\delta = \mathfrak{P}\, \mathbb{E}(\Delta W_k \hat{Y}_{k+1}^n \,|\, \tilde{X}_{k,J}^\delta)$$

      compute by a backward loop in j

$$\tilde{Z}_{k,j}^\delta = \frac{1}{\delta t} \left\{ \mathfrak{P}\, \mathbb{E}[\delta W_{k,j} \tilde{Y}_{k,j+1}^\delta | \tilde{X}_{t_{k,j}}^\delta] \right\},$$

      and

$$\tilde{Y}_{k,j}^\delta = \mathfrak{P}\, \mathbb{E}[\tilde{Y}_{k,j+1}^\delta - f(t_{k,j}, \tilde{X}_{k,j}^\delta, \tilde{Y}_{k,j+1}^\delta, \tilde{Z}_{k,j}^\delta)\delta t | \tilde{X}_{t_{k,j}}^\delta],$$

      ii. Compute

$$\bar{Z}_k^{n+1} = \frac{1}{\Delta t} \left\{ \mathfrak{P}\, \mathbb{E}[\bar{Y}_{k+1}^{n+1} | \hat{X}_{t_k}^{n+1}] \right\}$$

      and

$$\bar{Y}_k^{n+1} = \mathfrak{P}\, \mathbb{E}[\bar{Y}_{k+1}^{n+1} - f(t_k, \bar{X}_k^{n+1}, \bar{Y}_{k+1}^{n+1}, \bar{Z}_k^{n+1} | \hat{X}_{t_k}^{n+1}]$$

iii. Set

$$\hat{Z}_k^{n+1} = \bar{Z}_k^{n+1} + \tilde{Z}_k^{\delta} - \hat{Z}_k^n$$

and

$$\hat{Y}_k^{n+1} = \bar{Y}_k^{n+1} + \tilde{Y}_k^{\delta} - \hat{Y}_k^n$$

Remark 20. *All computations in the subintervals can be made in parallel.*

4.3 Numerical Experiments

4.3.1 Bid-Ask Spread Option for Interest Rate

In our experiment, we choose the pricing problem of a Call Spread option under different interest rates which has been studied by Bergman [20]. This derivative product allows the trader to invest into a riskless bond with an investing rate $r \geq 0$ and to borrow from the bond at rate $R \geq r$. The generation function $f(\cdot)$ of the backward stochastic differential equation in 4.2 has the following form:

$$f(t,x,y,z) = -ry - \frac{(\mu - r)}{\sigma} z - (R - r)\left(y - \frac{z}{\sigma}\right)^{-}, \tag{4.12}$$

where R and r represents respectively the lending rate and the deposit rate ($r \leq R$), μ is a constant drift of the dynamic of the underlying process x and σ its constant volatility. Following the results of Bergman, the evaluation of the Call Spread option ppr the dynamics of the replicating portfolio can be formulated in terms of a backward stochastic differential equation by

$$Y_t = Y_T - \int_t^T \left(rY_s + \frac{(\mu - r)}{\sigma} Z_s - (R - s)\left(Y_s - \frac{Z_s}{\sigma}\right)^{-} \right) - \int_t^T Z_s dWs, \tag{4.13}$$

with

$$Y_T = (X_T - K_1)^{+} - 2\,(X_T - K_2)^{+}, \tag{4.14}$$

where K_1 and K_2 are two strikes. In that case, the backward stochastic differential equation in 4.2 has no analytical solution which requires an approximation algorithm.

4.3.2 Numerical Results

For our test case, we consider the same following set of parameters of Gobet et al. [120], Lemor et al. [75], Bender and Steiner [16], Ruijter and Oosterlee [168] and take their estimations for the Call Spread option as the reference price i.e. $(Y_0, Z_0) = (2.96, 0.55)$. For the borrowing rate, we set $R = 6\%$, for the lending rate we choose $r = 1\%$, the maturity of the contingent claim $T = 0.25$. The strikes are set to $K_1 = 95$ and K_2, the initial price of the underlying is $X_0 = 100$, the volatility of the underlying price is $\sigma = 20\%$ and the drift is set to $\mu = 0.05$ (we choose a Black Scholes model for the dynamic of the process X).

Following Lemor et al. [75], we take the payoff of the Call Spread option as the basis function and the degree of the monomial are up to 7 for both approximations of Y and Z processes. We have chosen a fine grid with $\delta t = T/32$. The free parameters are Δt which governs the number of points on the coarse grid and n the number of parareal algorithm. The error between the price of a Bid-Ask Spread Call option computed on the fine grid by LSMC and the same computed by the parareal algorithm is displayed on Table 4.1 and on figure 4.1 The Monte Carlo parameters are 100 000 simulation paths. On the figure 4.2, we also compare the L_2-error between the reference solution and the ones obtained with a sequential algorithm of the

LSMC (discretization corresponds to the fine grid of the parareal method) and with the parareal procedure (for n=2). The curves are almost overlapping and we observe the same ratio than for the American case in the previous chapter.

4.4 Conclusion

In this chapter, we established a new numerical scheme for the approximation of a decoupled forward backward stochastic differential equation and shown numerically its performance. This scheme allows an efficient parallelization of the approximation so gaining time of computation without losing precision. One question remain on the choice of the subdivision in the coarse intervals as the variance of term $\frac{\Delta W_t}{\delta t}$ explode when δt is too small.

Future work will be concentrated on the demonstration of the convergence of this numerical scheme for a decoupled forward backward stochastic differential equation. Studies are in progress around the computation of the sensitivities of a Bid-Ask Spread Call option for interest rates, on the feasibility of the approximation of a coupled system and on reflected BSDE.

Time-step			Absolute error			
K	J	Δt	$n=1$	$n=2$	$n=3$	$n=4$
2	16	0.125	1.323e−01	3.271e−02	2.057e−03	3.703e−06
4	8	0.062	5.761e−02	1.630e−02	1.096e−03	2.424e−06
8	4	0.031	2.879e−02	8.610e−03	7.457e−04	1.835e−06
16	2	0.015	1.235e−02	5.043e−03	9.995e−05	8.492e−07

Table 4.1: Absolute error from the price of a Bid-Ask Spread Call option computed on the fine grid by a sequential LSMC standard algorithm and the same computed using the parareal iterative procedure 4.2.2. The coarse grid has K intervals; the coarse time step is $\Delta t/K$; the fine grid has a fixed number of points hence each interval $(t_k, t_{t_{k+1}})$ it has J sub-intervals.

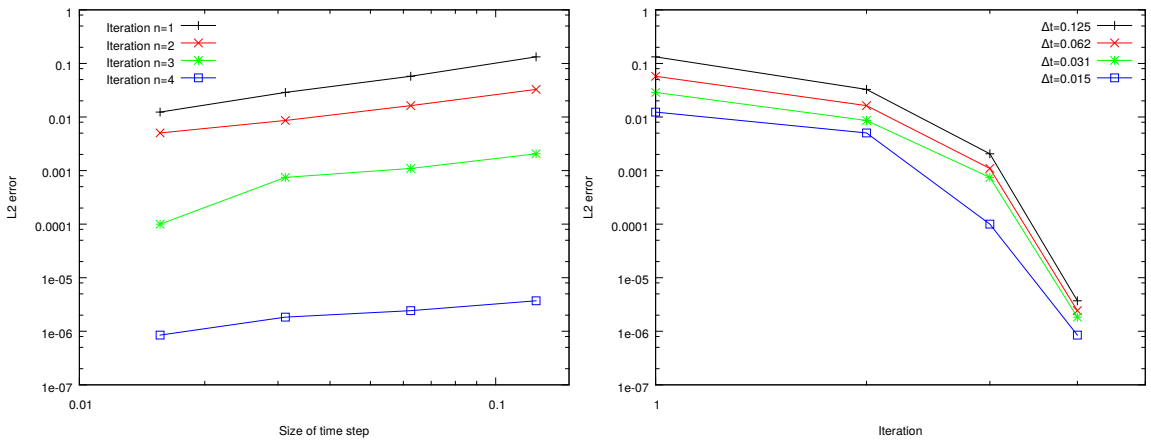

Figure 4.1: FBSDE: Errors on the payoff versus Δt on the left for several values of n and versus n on the right for several values of Δt. Both graphs are for Algorithm 3.5 in log-log scales and indicate a general behavior of the error ε.

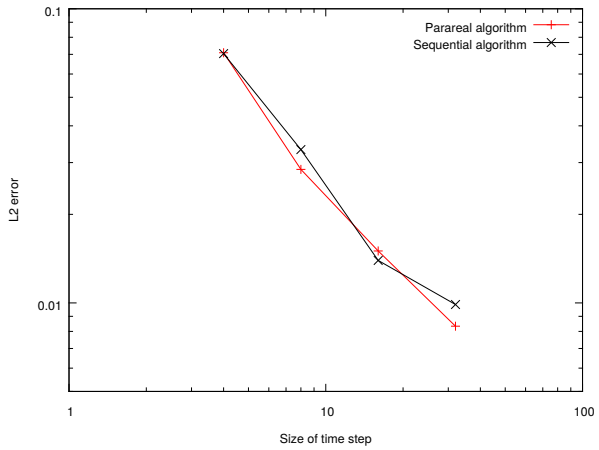

Figure 4.2: Comparison between the reference solution and a standard LSMC solution vs the parareal solution for the same number of time intervals at the finest level. The total number of time steps is on the horizontal axis, in log scale and the error at $n = 2$ is on the vertical axis in log scale as well.

Travaux Effectués en Entreprise

Chapitre 5

Moteur de Calcul Optimisé pour la Valorisation d'un Portefeuille Massif d'Actifs Financiers à Architecture Hybride et Diverse

---

"It is quite likely that future computing machines will permit simultaneous, independent arithmetic and logical operations"
extrait de The use of Parallelism in Numerical Calculations, [45].

– J. Cocke et D. S. Slotnick, Mathématiciens, 1958.

Ce chapitre est dédié au travail réalisé au sein de Global Market Solutions pour la conception d'un moteur de calcul efficient et rapide pour le calcul massif de produits dérivés de taux. Ce travail s'inscrit en réponse aux demandes croissantes d'indicateurs de risques, la valorisation rapide des portefeuilles de produits financiers est devenu un enjeu majeur pour les banques. Le moteur de calcul repose sur une architecture parallèle à mémoire partagée et distribuée. Nous verrons comment profiter de tous les types de granularité de parallélisation en vue de bénéficier de toutes les capacités des différentes architectures. Nous procédons à plusieurs tests numériques pour prouver l'efficacité de l'implémentation.

5.1 Introduction

Ici, nous nous focalisons sur le point critique de l'évaluation des mesures xVAS (voir Brigo et al. [34], Sidenius [173] et en appendice A), c'est-à-dire le calcul des prix des produits dérivés. Ces mesures nécessitent aussi de connaître la valeur du portefeuille à tous les pas de temps jusqu'à la maturité et non pas seulement à la vue d'aujourd'hui. Généralement, un portefeuille est composé de 80% de produits vanilles, qui peuvent être évaluer par des méthodes de Monte Carlo ou par des formules fermées. Dans notre cas test, nous considérons un portefeuille pouvant comprendre jusqu'à 100 000 SWAPS et 100 000 CAPs/FLOORs. La valeur du portefeuille vue en un pas de temps est la valeur de tous ses produits dérivés sous-jacents encore "en vie".

Les SWAPs sont évalués par méthode de Monte Carlo dite “brute force”. On calcule la valeur du contrat en chaque nœud qui est définit par une simulation et un pas de temps puis on moyenne la valeur sur toutes les simulations à chaque pas de temps. On stocke tous les résultats. C'est essentiellement la même idée pour l'évaluation des CAPs/FLOORs sauf qu'ils sont calculés à partir des formules fermées en chaque nœud. La distribution au cours du temps des contrats et leurs spécificités i.e. maturité, dates de paiements et nombre de dates de paiements est basée sur un portefeuille réel d'une importante banque française. Les paramètres pour la simulation Monte Carlo est de 1000 trajectoires pour chaque risque facteurs. Nous proposons une solution efficace pour tenter de régler ce problème. La grille de temps est définie par 100 pas de temps qui correspondent à la répartition suivante: 30 jours, 20 mois et 50 ans. Ici, nous nous concentrons sur le temps de calcul et nous analysons le temps de transfert des données à la carte lorsqu'on interagit avec des quantités massives de données.

Ce chapitre est organisé de la façon suivante; dans la section 5.2 nous commençons par rappeler le descriptif des produits dérivés d'intérêt de taux et leur valorisation.

La section 5.3 est dédiée à la description technique de l'architecture du Xeon Phi Coprocessor sur lequel nous avons procédé à la conception du moteur de calcul. Puis nous passerons à la retranscription du processus d'optimisation du code en section 5.4.

Dans la section 5.5, nous procédons à la réalisation de tests et de mesures sur les différentes architectures, c'est à dire un serveur seul, une carte graphie seule, un serveur avec une carte accéleratrice et un serveur avec six cartes accélératrices. Des résultats de temps sont affichés pour plusieurs types de portefeuilles comportant différents produits dérivés avec différentes tailles.

En section 5.6, nous nous consacrons à la description des codes utilisés et de leur spécificités pour obtenir des performances en terme de temps de calcul.

La section 5.7 est réservée aux expérimentations et résultats numériques, nous comparons les différents code de calcul mise au point. Enfin, nous terminons et concluons avec un récapitulatif du travail réalisé au sein de Global Market Solutions.

5.2 Produits Dérivés de Taux d'Intérêt

Il existe une littérature très large autour des produits d'intérêt de taux et sur la façon de l'évaluer. Ici, nous faisons une rapide descriptions des produits de taux qui constituent les portefeuilles mais le lecteur curieux peut se réferer à la littérature suivante Rebonato [157] and [158], Black and Cox [24], Miltersen et al. [133], Hunt and Kennedy [98] and El Karoui [106]. En appendice D, une description du modèle de simulation utilisé i.e. le Libor Market Model (LMM).

5.2.1 Échange de Taux d'Intérêt (SWAP, IRD)

Pour rappel, un SWAP est un contrat d'échange de flux financier standard de taux d'intérêt entre deux parties. Ces échanges de paiements sont réalisés à un ensemble de dates futures pré-spécifiées $T_1, \ldots, T_\beta$, $\beta \in \mathbb{N}^*$ a priori de la signature du contrat entre deux jambes indexées différemment. Généralement, l'échange se fait entre un taux variable et un taux fixe. Le client pouvant prendre deux positions différentes sur le produit soit payeur soit receveur de la jambe flottante et donc respectivement receveur ou payeur de la jambe fixe. Un SWAP de taux d'intérêt standard correspond à la somme des différentes jambes flottantes et fixes à l'ensemble des dates de paiements spécifiées qui restent encore à honorer. Ainsi, la fonction d'évaluation actualisée

au temps $t < T_\alpha$ avec $\alpha \geq 1$ d'un SWAP de taux d'intérêt standard peut être exprimée comme:

$$\text{SWAP}_t = \omega \sum_{i=\alpha}^{\beta} \text{DF}(t, T_i) N \tau_{[T_{i-1}, T_i]} \left(L(T_{i-1}, T_i) - K \right), \quad \beta \in \mathbb{N}^*, \quad t > 0, \tag{5.1}$$

où ω prend la valeur -1 si le contrat est dans la position payeur (le client paye la jambe flottante) et 1 si le contrat est dans la position receveur. $\text{DF}(t, T)$ est un taux stochastique d'actualisation entre les instants t et T, ce qui correspond au montant en t qui est équivalent à une unité dans la monnaie d'échange à l'instant T. $L(t, T)$ est le taux d'intérêt sous-jacent (LIBOR[1], EURIBOR, EONIA[2] etc...) réajusté à l'instant précédent t pour la maturité donnée de l'instant actuel T. Le terme K désigne le taux d'intérêt fixe qui a été décidé lors de la signature du contrat, N est le notionnel du contrat et $\tau_{[t,T]}$ est la fraction d'année entre deux instants de paiements t et T.

Dans le cas de notre cas test, l'évaluation d'un SWAP standard de taux d'intérêt impliquent plusieurs données de marché i.e. la diffusion et la simulation des facteurs de risques sous-jacents (le taux variable de la jambe flottante et le taux d'actualisation) dans le future. Cela nécessite la manipulation d'un paquet de données d'une certaine taille qui peut s'avérer être un point critique en terme de performance de calcul.

5.2.1.1 Option Plafonnée sur Taux d'Intérêt (CAP/FLOOR, IRD)

Un contrat CAP/FLOOR peut être considéré comme une option sur un SWAP standard de taux d'intérêt, en effet les échanges de paiements sont réalisés si et seulement si ils ont une valeur positive[3]. L'acheteur d'un CAP reçoit un montant financier à chaque date de paiement si le sous-jacent de taux d'intérêt excède le taux fixe du contrat sur lequel les deux parties se sont accordées. Ce type de produit peut être utilisé pour la couverture de positions contre les fluctuations des taux d'intérêts. Un CAP/FLOOR noté C/F_t est exprimé comme la somme d'une série d'options Caplet/Floorlet qui sont similaires à une option européenne. Ces dernières peuvent être évaluées à l'aide du modèle de Black, ainsi on aura:

$$\text{C/F}_{t=0} = N \sum_{i=\alpha}^{\beta} P(0, T_i) \tau_{[T_{i-1}, T_i]} \left(\omega F(0, T_{i-1}, T_i) \varphi \left(\omega d_+ \left(F, K, \upsilon \right) \right) - \omega K \varphi \left(\omega d_- \left(F, K, \upsilon \right) \right) \right), \tag{5.2}$$

avec

$$d_\pm(F, K, \upsilon) = \frac{\log\left(\frac{F}{K}\right) \pm \frac{\sigma^2}{2}}{\upsilon}, \quad \upsilon = \sigma_{\alpha,\beta} \sqrt{T_{i-1}}, \tag{5.3}$$

où ω prend la valeur 1 si le contrat est un CAP et -1 si le contrat est un FLOOR, le terme φ désigne la fonction de répartition de la normale centrée réduite, $\sigma_{\alpha,\beta}$ est le paramètre de la volatilité extraite des données de marché pour la période de temps entre α et β. $F(0, t, T)$ est le prix future d'un sous-jacent de taux d'intérêt vu d'aujourd'hui ayant la pair de maturité et d'expiration (t, T). $P(0, T)$ est le prix du zéro coupon vu d'aujourd'hui avec une maturité T.

Clairement, le calcul de ces formules sur l'ensemble de plusieurs milliers de simulations s'avère être extrêmement coûteux en temps. Effectivement, l'approximation de la fonction de répartition de la loi normale centrée réduite est gourmande en opération. Celle-ci peut demander plusieurs

[1]Le LIBOR et l'EURIBOR sont des taux de références indexés dans des monnaies différentes, respectivement dollar et euro. Communément appelés taux interbancaires.

[2]L'EONIA est un taux de référence journalier des dépôts interbancaires effectués dans la zone euro.

[3]Le client ne fera choix d'exercer son droit sur un versement que si l'échange de flux financier est en sa faveur

centaines de cycles d'horloge pour son calcul. Malheureusement, il n'existe pas d'astuce mathématique pour contourner cette fonction, en revanche son optimisation peut venir d'une connaissance approfondie de la programmation i.e. comme l'utilisation de la vectorisation pour toutes les opérations arithmétiques. Une seconde approche pour son optimisation a été de considérer une tabulation des valeurs de cette fonction en un ensemble de points calculés off-line. Le problème de cette solution est qu'elle nécessite de gérer encore plus de données nous n'avons donc poursuivi que la première option.

5.3 Intel Xeon-Phi Coprocessor

En septembre 2011, le centre de calcul haute performance du Texas (The Texas Advanced Computing Center i.e. TACC) a annoncé qu'il fonctionnerait désormais à partir de la nouvelle carte de calcul d'Intel - Intel Xeon Phi Coprocessor -, illustrée sur la figure 5.1. Un supercalculateur visant les 10 petaFlops[4]. puissance de calcul faisant l'un des plus puissants calculateur au monde. Le calculateur fournissait auparavant 2 petaFlops de puissance de calcul à partir d'une grille de CPUs, l'ajout des cartes accélératrices d'Intel permettant 8 petaFlops supplémentaires. En juin 2013, le supercalculateur Tianhe-2 du centre nationale de calcul de haute performance à Guangzhou (NSCC-GZ) est devenu le calculateur le plus puissant au monde, utilisant la combinaison des cartes Intel Ivy-BridgeEP Xeon et Xeon Phi Coprocessors pour culminer à 34 petaFlops. En juin 2016, le Sunway TaihuLight franchit la barre symboliques des 100 petaFlops ce qui le place à la tête des calculateurs les plus puissants. Pour revenir dans la course, les américains ainsi que les européens ont lancé deux projets visant à atteindre l'exaFlops de puissance de calcul à partir d'architecture hybride.

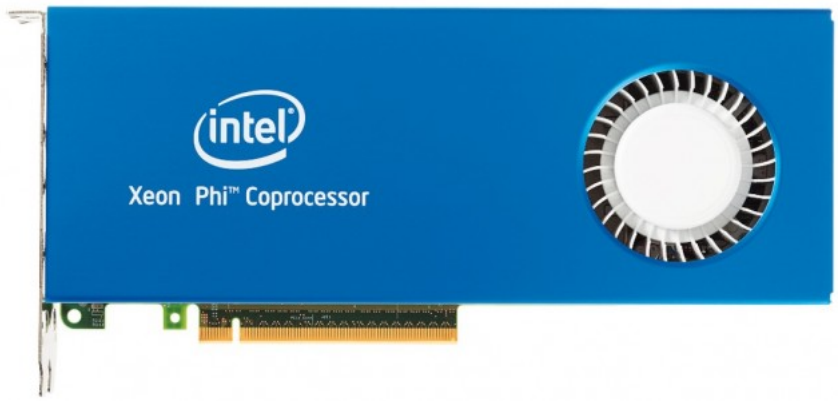

Figure 5.1: La carte Intel Xeon Phi Coprocessor.

L'architecture d'Intel MIC[5] combine ensemble plusieurs processeurs de CPUs d'Intel sur une même puce, c'est une architecture dédiée exclusivement au calcul haute performance et plus précisément au calcul scientifique. Cette architecture permet de calquer facilement des algorithmes dits massivement parallèles sur la grille de calcul. Dans notre cas, l'évaluation d'un portefeuille de produits dérivés par des méthodes de Monte Carlo. Dans le format de double de précision flottante, le performance maximale observée est de 1.2 teraFlops (DP) pour un seul

[4]Nombre d'opérations en virgule flottante par seconde, ce qui est équivalent à 10^{16} opérations par seconde.
[5]Many Integrated Core, défini dans Jeffers [102].

Xeon Phi Coprocessor. La carte n'est pas directement intégrée sur la carte mère, elle est jusqu'à présent seulement disponible à travers un port PCI[6]mais il offre une bande passante très large de 353 Gb/s consommant 300 watt pour 32 Go de capacité de mémoire type GDDR5[7] avec un pic de vitesse àe 5.5 GT/s pour les dernières configurations. L'Intel Xeon Phi Coprocessor peut embarquer son propre système d'exploitation, supporte les protocoles de parallélisation MPI[8] et openMP[9], accepte l'adressage IP.

L'intel Xeon Phi Coprocessor peut proposer des configurations jusqu'à 61 cœurs (unité de calcul vectoriel) basés sur l'architecture de la famille des processeurs Intel Pentium, description dans le manuel d'Intel[10]. Les processeurs sont cadencés à 1.238 GHz, ce qui n'est pas excessifs mais la performance est dans le nombre de processeurs engravés sur une même puce. Intel utilise une gravure à 14 nm, ce qui permet de produire des puces comportant plus de 50 cœurs. Chaque cœur physique permet de récupérer et de décoder 4 fils d'instructions simultanément ce qui, pour la configuration la plus lourde, gère jusqu'à 244 fils d'instructions. L'intel Xeon Phi Coprocessor tourne sur un environnement 64 bit, fournit la compatibilité pour l'architecture x86 et supporte les set instructions IEEE 754[11] pour l'arithmétique en virgule flottante. La carte est munit d'un environnement très flexible pour le développement d'applications. Chaque cœur est connecté à un anneau d'interconnexion à travers le protocole Core Ring Interface[12] (cf. schéma de l'architecture sur la figure 5.2), qui comprend le cache L2 de 512 Kb. Cette interface permet d'exécuter 2 instructions par cycle, sur le U-pipe (détails dans le manuel d'Intel cité précédemment) et le V-pipe mais certaines ne peuvent pas être exécutées sur cette dernière (comme les instructions vectorielles). Chaque cœur a un support de calcul de haute précision et optimisé pour les fonctions mathématiques i.e. racine carrée, exponentielle, logarithme, puissance, réciproque etc... Des fonctions spéciales liées à la gestion de mémoire (scatter, gather et streaming store) sont très efficaces. Les cœurs comprennent les instructions du cache L1 et du data cache (32 Kb pour chacun des caches). Chaque cœur est muni de registres vectoriels de 32 R12 bit.

Malheureusement, la carte ne supporte pas les instructions Intel SIMD[13] comme MMX[14], SSE[15], SSE2, SSE3, SSSE3 SSE4.1 et SSE4.2, ni les instructions AVX[16] donc la portabilité du code n'est pas direct mais Intel fournit un nouveau set d'instructions vectorielles. Ce set permet d'exécuter des instructions de calcul arithmétique par vecteur de 512 bit, et peut ainsi traiter simultanément 16 flottants en précision simple (SP) ou 8 en double précisions (DP). Chaque opération peut être une combinaison de multiplication et d'addition permettant de traiter 32 SP ou 16 DP par cycle. La plupart des instructions vectorielles a une latence de 4 cycle d'horloge et une cadence d'un cycle.

L'architecture Intel Xeon Phi Coprocessor implémente "seulement" les codes d' applications en C, C++ et Fortran. Un des meilleurs avantages est qu'elle permet la compilation et l'exécution

[6]La carte est prise en charge comme un périphérique externe.

[7]Graphics Double Data Rate, mémoire pour carte graphique optimisée.

[8]Message passing interface, parallélisation à architecture comportant une mémoire distribuée voir Delage Santacreu [171].

[9]Langage de programmation permettant la parallélisation sur une architecture à mémoire partagée, protocole décrit dans le manuel d'OpenMP (OpenMP Application Programming Interface, 2015, disponible à l'adresse suivante: `http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf`)

[10]Pentium II Processor Developer's Manual, 1997, disponible à l'adresse suivante: `https://ftp.utcluj.ro/pub/docs/publicatii/intel/pentiumII.pdf`

[11]Norme de représentation pour les nombres à virgule flottante en binaire.

[12]Voir les documentations détaillées dans Chrysos [43], Jeffers [102] et Reed [161].

[13]Single instruction multiple data. Voir `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`.

[14]Multiple Math eXtension, vecteur de 64 bit.

[15]Streaming SIMD eXtension, vecteur de 128 bit.

[16]Advanced Vector eXtension, vecteur de 256 bit.

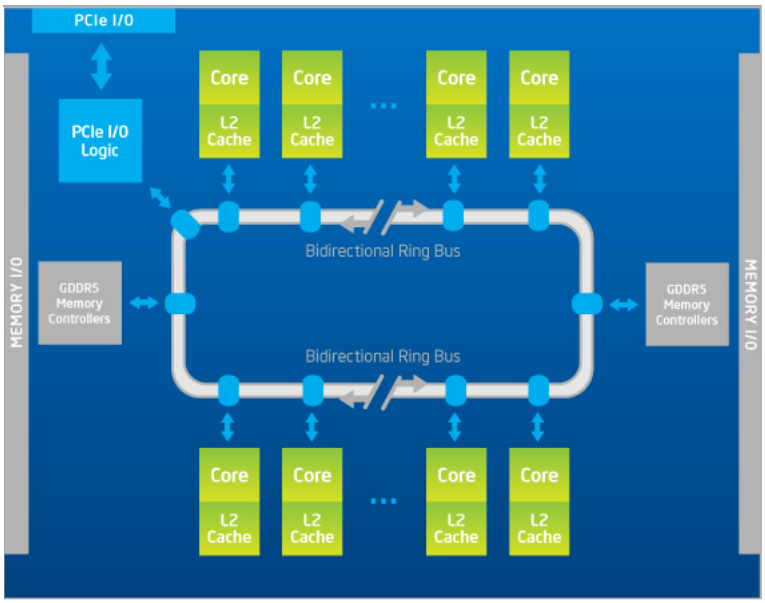

Figure 5.2: L'architecture de l'Intel Xeon Phi Coprocessor.

d'un même code source qui s'exécute sur des CPUs standards. Il y a 3 (principalement 2) mode d'exécutions décrits dans Atanassov et al. [8], James [100] et Jeffers et Reinders [103]:

- mode d'exécution "natif" permet à l'utilisation de compiler le code source et d'exécuter l'application générée directement sur le carte accélératrice, ce qui permet d'éviter les transferts de données entre l'hôte et la carte. Le but principal étant que ce mode de fonctionnement n'implique aucun changement au niveau du code mais nécessite l'ajout de quelques options de compilation (`-mmic`). Il est possible d'utiliser l'intel Xeon Phi Coprocessor comme un Intel Xeon mais la performance observée sera nettement plus faible. Effectivement, ce dernier est muni d'une architecture qui se concentre sur l'utilisation et l'optimisation de performance d'un seul cœur alors que la carte accélératrice se concentre sur la performance d'un nœud de cœurs. Les cœurs seuls sont beaucoup moins performants que ceux se trouvant sur l'Intel Xeon. Par exemple, MacCalpin dans McCalpin [131] stipule qu'il est préférable d'utiliser les CPUs pour faire tous les processus de lecture et d'écriture et de transférer les données sur la carte. Enfin pour utiliser ce mode, il faut que l'application soit en adéquation avec l'exécution native.

- mode d'exécution "offload", le second mode principale d'exécution et qui est aussi considéré comme un mode de programmation hétérogène. Une partie ou tous les processus de calculs sont déchargés sur la carte accélératrice mais l'application est exécutée sur le hôte. Sur l'exemple 5.1, la partie du code qui doit être chargée sur la carte se fait à l'aide d'une directive, il faudra tout de même envoyer les données nécessaires au calcul. Les directives "in" et "out" permettent le transfert de données[17].

```
float call_price = 0.f;
```

[17]Il est seulement nécessaire de spécifier le transfert pour les tableaux et structures de données, les variables scalaires sont automatiquement transférées.

```
#pragma offload target( mic : 0 ) in( S : length( N ) )
for( unsigned int i = 0; i < N; ++i )
        call_price += maxf( S[i] - K, 0.f );

call_price /= ( float ) N;
```

Code 5.1: Exemple d'une section offload programmée en C++. Calcul du prix d'un call à partir des simulations d'un sous-jacent noté S.

L'utilisateur peut décider d'envoyer des données sur la carte et lla laisser travailler sur les données pendant que l'hôte exécute d'autres tâches. Ils peuvent ou ne pas travailler en parallèle. Le mode "offload" est un protocole simple qui requiert uniquement l'ajout de quelques directives (compatibles avec les directives d'OpenMP) au niveau du code pour signaler au compilateur que la section doit être exécutée sur la carte. Les données doivent être explicitement transférées pour chaque section déchargée à la carte ou l'utilisateur peut utiliser la mémoire virtuelle partagée pour adresser l'hôte et la carte. Les sections sous directives "offload" initialisent les calculs sur la carte et peuvent être exécutées de façon synchone ou asynchrone. Les options dans les directives procurent un contrôle sur la structure des données ainsi que la gestion de l'allocation dynamique de la mémoire sur la carte. Comme détaillé dans Fourestey [54], il suffit de rajouter l'option de compilation `-offload` à la compilation pour prendre en compte les directives offload. Pour plus d'optimisation au niveau de l'architecture, il sera utile d'examiner l'affinité des processeurs. L'affinité peut avoir un impact très significatif sur les performances de calcul d'un algorithme. Ce mode nécessite le plus d'effort sur la programmation mais le gain potentiel en performance est probablement le plus grand. Effectivement, un code peut exploiter simultanément la carte accélératrice et l'hôte pour travailler ensemble mais cela nécessite une approche non triviale de l'algorithme. Enfin, le mode "offload" évite toutes les complexités de la communication MPI.

- mode d'exécution "symétrique" permet d'exécuter simultanément les processus d'application sur la carte et le hôte. Ils communiquent par un protocole spécifique de messagerie (MPI). Les Intel Xeon Phi Coprocessors sont interprétés comme des nœuds MPI dans un environnement hétérogène et chaque carte accélératrice peut contenir un grille de nœuds MPI (sous-ensemble du mode symétrique). Cela peut créer quelques problèmes d'équilibrage des tâches à cause de mutliple cadencement de fréquence des cœurs mais ces différences doivent être prisent en compte lors de l'implémentation de l'architecture MPI. Les codes MPI de l'hôte et de la carte doivent être compilés séparément. Comme pour le mode "natif", ce mode est moins intrusif pour les codes existants mais nécessite tout de même quelques modifications (comme l'utilisation des fonctions intrinsèques spécifiques). Il faut noter que la compléxité de l'équilibrage de tâche sur une grille à environnement hétérogène peut s'avérer être, en réalité, un challenge extrêmement difficile.

5.4 Optimisation

Les optimisations pour un gain de performance peuvent être faîtes à différents niveaux de l'architecture d'une application, cela dit il est préférable de commencer par des optimisations les moins intrusives et de prendre de plus en plus d'abstraction (top-down approach). La première

approche qu'on peut avoir est le choix adéquate du système d'exploitation (OS). Par exemple windows n'est pas conseillé pour l'utilisation de l'Intel Xeon Phi Coprocessor et l'interfaçage des OS doit être prise en compte. En effet, la carte tourne sur son propre système d'exploitation i.e. linux donc il est fortement conseillé d'avoir un système homogène. La vitesse de transfert peut diminuer dans le cas d'un système hétérogène.

La seconde étape se concentre sur l'optimisation de l'algorithme, comme l'utilisation de fonctions spécifiques mathématiques combinées à des optimisations de programmation. Et enfin la troisième étape de cette démarche de gain en performance peut être considéré comme du micro-tuning architecturale. Cela nécessite une connaissance approfondie de l'architecture de la machine pour tenter de bénéficier au maximum de toutes ses spécificités. Par exemple, contrôler la gestion de la mémoire, la structuration des données, l'utilisation optimale des registres et des mémoires caches pour éviter de perdre du temps à lire et écrire en mémoire globale. Un des aspects les plus critiques de la gestion des données est de passer de tableaux de structures à des structures de tableaux (AoS to SoA). En effet, le compilateur n'opère pas toutes les optimisations possibles sur des tableaux de structures, en général il n'y aura pas de vectorisation. L'alignement des données en mémoire ou aussi appelé contiguïté des données en mémoire (Larsen et al. [117]), les potentiels problèmes de dépendances entre les pointeurs (pointer aliasing), le préchargement des données sont des aspects importants qui nécessitent une connaissance profonde du langage de programmation dans lequel l'application est codée.

En ce qui concerne le parallélisme, gérer et définir le comportement des processus logiques sur les cœurs physiques ou encore éviter qu'un ou plusieurs cœurs ne soient inactifs lors de l'exécution de l'application. La gestion des processus logiques à l'aide des directives d'OpenMP peut avoir un impact plus que significatif sur les performances d'un code de calcul mais il faudra faire attention à la structuration des données (éviter les nœuds d'écritures etc...). On peut également faire du parallélisme intra-cœur en faisant du calcul vectoriel et non pas scalaire. Toutes ces optimisations concernent les points critiques d'une application et requiert une analyse approfondie de son architecture. Une méthode simple pour savoir si le parallélisme a bien été implémenté est de vérifier si le pourcentage d'occupation des cœurs à l'aide d'une application Open Source.

Ici, nous procédons de la façon précédemment énoncée c'est-à-dire de la moins intrusive, qui nécessite aucun ou peu de changements sur le code, à plus complexe qui pourrait demander la réécriture complète du code. Le niveau d'intrusion correspond "proportionnellement" au gain en performance obtenu, moins il y a d'intrusion moins important sera la gain en performance. Le processus d'optimisation concerne dans un premier temps l'Intel Xeon puis l'Intel Xeon Phi Coprocessor. Sur le schéma 5.3, un récapitulatif des étapes suivies.

5.4.1 Système d'Exploitation

Jusqu'à présent et comme suggéré dans le guide pratique de l'Intel Xeon Phi Coprocessor Atanassov et al. [8], le meilleur système d'exploitation pour l'utilisation de l'Intel Xeon Phi Corprocessor est le système Redhat Entreprise qui supporte des distributions Linux (la plupart sont des versions Redhat vX.X). Effectivement, nous avons rencontré quelques problèmes d'interfaçage ou des problèmes lors de transfert de données sur la carte avec d'autres distributions de type Linux. Plus récemment Windows a été ajouté à la liste des systèmes d'exploitations pouvant interagir avec la carte accélératrice d'Intel mais sachant que celle-ci opère une distribution de Linux en son sein, il est conseillé de choisir un système homogène pour ne pas perturber l'interfaçage entre les deux entités.

1. Choix du système d'exploitation
2. Gestion l'affinité des threads à l'aide de variable d'environnement
3. Ajout d'option de compilation pour optimiser le code généré
4. Parallélisation des boucles
5. Utilisation des fonctions mathématiques spécifiques et optimisées
6. Déstructuration des objets pour ne manipuler que des vecteurs de données
7. Vectorisation du code (fonctions intrinsèques, alignement et gestion de la mémoire)
8. Développement d'un code hybride pour cibler plusieurs architectures différentes

Schéma 5.3: Processus d'optimisation (approche Top-Down). On part de l'approche la moins intrusive à la plus intrusive dans le code.

5.4.2 Parallélisme et AoS to SoA

Un des points les plus importants de l'optimisation de l'application est la parallélisation du processus d'évaluation du portefeuille de produits dérivés. Il est important de noter qu'il existe plusieurs type de parallélisation avec des niveaux d'efficacités différents. Ce que l'on pourrait appeler macro, méso et micro parallélisation avec une finesse de granularité différente comme représentée dans le tableau 5.1.

Parallélisation	Protocole	Type	Taille des Données
Macro	Posix Threads, MPI	Applications	Plusieurs Go
Méso	OpenMP, Cuda	Fonctions, boucle	Quelques Mo
Micro	SSE, AVX	Calcul arithmétique	Plusieurs bit

Table 5.1: Niveau et correspondance des différents protocols de parallélisation.

Il est nécessaire de bien connaître l'architecture de la machine pour profiter au plus de ses spécificités. L'algorithme que nous avons fini par construire utilise ces trois niveaux de parallélisation. Nous utilisons de la macro pour la répartition entre les deux machines, puis la méso pour le parallélisme entre tous les cœurs d'une même machine et enfin la vectorisation pour paralléliser le calcul arithmétique.

Dans un premier temps, notre approche a été de paralléliser au niveau du contrat, chaque cœur physique reçoit un paquet de produits dérivés à évaluer. Malheureusement, cette approche ne sait pas couronner par un succès probant sur l'Intel Xeon Phi Coprocessor alors que l'approche avait prouvé son efficacité sur l'Intel Xeon. L'explication est que la gestion de grands tableaux de structures de données n'est pas la même sur les deux entités, les premiers résultats de performance ont montré que l'hôte surperformait la carte accélératrice. La conclusion a été que l'Intel Xeon Phi Coprocessor nécessitait une approche différente. La première amélioration a été de passer de tableaux de structures à des structures de tableaux, suggérée dans Jeffers [104] et Sabahl [169], pour faire en sorte que le compilateur puisse généré plus facilement du code optimisé. Ainsi, chaque paramètre d'un contrat est stocké dans un tableau comme schématisé sur la figure 5.4.

L'évaluation d'un SWAP, qui est la somme de paiements, nécessitent un taux d'actua- lisation,

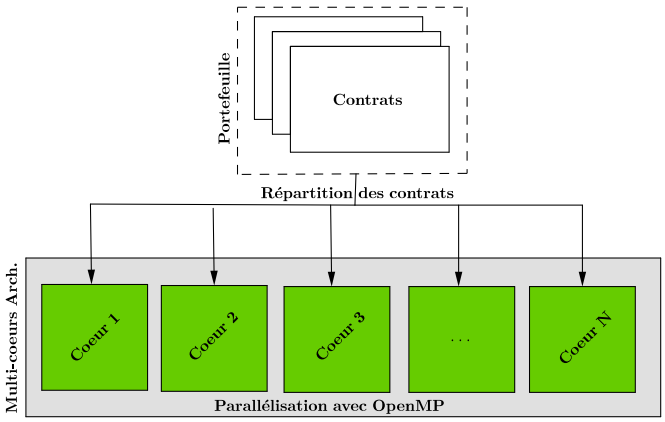

Figure 5.4: Représentation de la répartition des contrats selon la première approche et gestion du parallélisme sur une architecture à mémoire partagée.

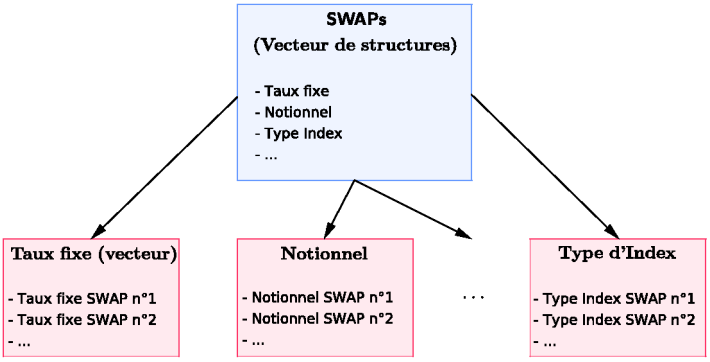

Figure 5.5: Déstructuration des données et représentation des structures en tableaux pour permettre la vectorisation et simplifier la gestion de mémoire.

un taux forward, un notionnel, un taux fixe, les dates de paiements et la difficulté est que chaque SWAP n'a pas les mêmes ni le même nombre de dates de paiements. Ce qui induit un certain niveau de complexité dans l'algorithme. Dans notre première approche, nous avions fait le choix de stocker toutes les dates de paiements de tous les contrats dans un même grand tableau mais le problème est de retrouver alors les dates correspondants à un contrat en particulier. Pour régler le problème, nous avons stocké l'indexation locale du point de départ des données dans un autre tableau. Ce qui permet de retrouver l'indexation globale d'un contrat à travers ce grand tableau de dates. Malheureusement cette solution ne permet pas un accès direct aux données, ce qui crée des problèmes de latence dûs aux accès en mémoire indirect. Cet effet est désastreux sur l'Intel Xeon Phi Coprocessor, les performances de calcul exhibées sont en-dessous de l'Intel Xeon alors que ce dernier est théoriquement moins puissant. En ce qui concerne la granularité, nous avions fait le choix de changer la stratégie de parallélisation et de se baser sur une réparition faite à

partir des paiements et non plus des contrats eux-mêmes. Cette répartition procure un meilleur équilibrage des processus logiques sur les cœurs et de la charge de travail à effectuer par cœur physique. Cette granularité est beaucoup plus fine que celle précédemment énoncée.

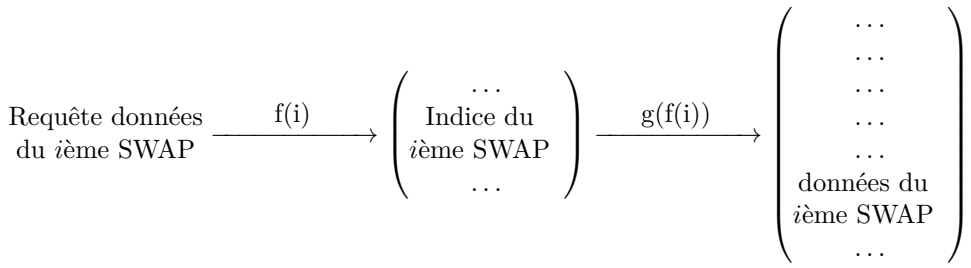

Figure 5.6: Accès indirect en mémoire des données relatives à un SWAP.

Dans notre seconde approche, nous manipulons uniquement le grand tableau de données de dates sur la carte accélératrice. Ce qui permet d'accéder directement aux données, la carte accélératrice se charge dans un premier temps d'évaluer tous les paiements en chaque dates avec une répartition de travail optimale entre chaque cœur physique. Dans un deuxième temps, nous procéderons à un travail d'agrégation pour retrouver la valeur de chaque contrat. Cette nouvelle approche permet d'obtenir les résultats de performances escomptées, ces nouveaux résultats montrent la supériorité en terme de performance de calcul de la carte accélératrice sur l'hôte. En effet, celle-ci se montre deux fois plus puissante.

Schématiquement, au lieu de répartir le portefeuille sous forme de paquet de contrat homogène entre les cœurs, nous répartissons la charge de travail au niveau des paiements. Chaque cœur reçoit un paquet de dates de paiements à évaluer. La configuration optimale pour la parallélisation de la gestion des processus logiques et des charges de travail est faite automatiquement et dynamiquement à travers des directives OpenMP. Naturellement, le nombre de processus logiques est fixé au maximum par rapport au nombre de cœurs physiques. Pour l'Intel Xeon et ses cœurs "hyperthreadés"[18] le maximum de processus logique est deux fois le nombre de cœur physique alors que sur l'Intel Xeon Phi Coprocessor, on peut placer jusqu'à 4 fois le nombre de cœur. Pour l'Intel Xeon, on placera 32 processus logiques et pour la carte accélératrice, 244 processus logiques en mode natif et 240 en mode "offload" car il est nécessaire de garder un cœur qui gère l'interfaçage avec l'hôte.

5.4.3 Fonctions Mathématiques Spécifiques

En prenant des fonctions mathématiques spécifiques à la précision voulue, on peut obtenir un gain en performance non négligeable (voir le guide de programmation d'IBM [19] et Jeffers et Reinders [104]) mais il faudra avoir une attention particulière à la précision des résultats obtenus... Dans notre cas test d'évaluation d'un portefeuille d'actifs financiers, nous utiliserons les fonctions sémantiques suivantes: `expf()`, `logf()` et `erff()` qui correspondent respectivement, en simple précision, aux fonctions exponentielle, logarithmique et fonction erreur qui permet de calculer

[18] L'hyperthreading est une technologie qui permet à un cœur physique de gérer simultanément deux processus logiques. Un cœur virtuel permet de gérer un deuxième processus logique mais la puissance observée n'égale pas celle dégagée par deux cœurs physiques. Sur les dernières générations de processeurs, cette technologie permet un gain de 30% de performance.

[19] Disponible à l'adresse suivante: `http://www-01.ibm.com/support/docview.wss?uid=swg27024743&aid=1`

la fonction de répartition de la loi normale par un changement de variable. L'utilisation de ces trois fonctions font diminuer drastiquement le temps de calcul dans le processus d'évaluation des produits dérivés.

5.4.4 Vectorisation

La vectorisation ou l'auto-vectorisation, comme indiquée dans Sabahl [169], consiste à dérouler une boucle d'instruction de calcul arithmétique sur un tableau de données et combiner ces instructions scalaires en instructions vectorielles. En effet, depuis maintenant une vingtaine d'année les processeurs d'Intel sont capables de procéder à une même instruction de calcul unitaire $+$, $-$, $\times$ et $/$ sur plusieurs données simultanément.

Les instructions vectorielles sont dotées de mémoire allant de 64 à 256 bit pour les processeurs standard d'Intel, seules les cartes d'Intel Xeon Phi Coprocessor ont des vecteurs de données de 512 bit. Sur la figure 5.7, la représentation du nombre à virgule flottante selon la précision voulue i.e. simple ou double dans les différents tailles de mémoire vectorielle.

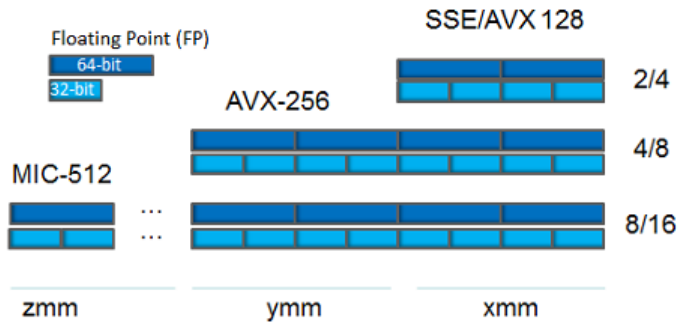

Figure 5.7: Tailles des vecteurs et des registres 128, 256 et 512 bit selon la précision voulue.

Dans le cas de l'auto-vectorisation, il suffit simplement d'ajouter au moins l'option d'optimisation `-O2` pour que le compilateur tente de vectoriser les boucles de calculs[20]. Le problème avec l'auto-vectorisation c'est que toutes les boucles contenant des processus de branchement ne seront pas automatiquement vectorisées alors que par certaines astuces mathématiques et informatiques, il sera possible de le faire. Pour forcer cette vectorisation, on sera obligé de passer à la réécriture du code et d'utiliser les fonctions intrinsèques vectorisées. Par exemple, pour le calcul d'un call en simple précision avec des vecteurs de 128 bit, sur les codes 5.2 et 5.3 on passera de:

```
float call_price = 0.f;

for( unsigned int i = 0; i < N; ++i )
        call_price += maxf( S[i] - K, 0.f );

call_price /= ( float ) N;
```

[20]Le compilateur ne peut vectoriser que la boucle la plus profonde et s'il n'y a pas d'appel à des fonctions autres que mathématiques et arithmétiques. Il ne faut pas de dépendance d'une itération à une autre i.e. $a[i+1] = \alpha * a[i] + \beta$. Il faut que les données soient alignées et contigües en mémoire.

Code 5.2: Exemple de boucle de code à vectoriser en C++.

à

```
__m128 *Ssse = reinterpret_cast<__m128*>(&S);

__m128 Osse = _mm_setzero_ps( );
__m128 Ksse = _mm_set_ps( K );
__m128 Nsse = _mm_set_ps( N );
__m128 call_price = Osse, tmp;

for( unsigned int i = 0; i < N; i+=4 ){
        tmp = _mm_sub_ps( Ssse[i], Ksse );
        tmp = _mm_max_ps( tmp, Osse );
        call_price = _mm_add_ps( call_price, tmp );}

call_price = _mm_div_ps( call_price, Nsse );
```

Code 5.3: Code 5.2 vectorisé à l'aide de fonctions intrinsèques.

Pour faire l'analogie avec notre cas test, il sera nécessaire d'avoir un certain ordre dans les données de simulations et de cette structuration, une autre difficulté en ressort. On ne diffuse pas un sous-jacent mais un vecteur de sous-jacent de taux d'intérêts et la structure de l'algorithme nécessite à ce que les données soient ordonnées la façon [21] suivante:

$$\left(L_1^{1,1},\ldots,L_k^{1,1},\ldots,L_1^{i,1},\ldots,L_k^{i,1},\ldots,L_1^{1,j},\ldots,L_k^{1,j},\ldots\right),\quad i,j,k\in\mathbb{N}^{*,3} \tag{5.4}$$

où $L_k^{i,j}$ désigne le prix future du $i$ème ténor au $j$ème pas de temps et de la $k$ème simulation. Cette structuration permet d'avoir une contig'uité d'accès en mémoire et réduit le temps d'accès à la mémoire globale car une partie des tableaux peut être contenu directement dans les registres. L'alignement spatiale des données est faite à l'aide de la fonction d'allocation `_mm_malloc()` et sa désallocation par `_mm_free()`, la taille de l'alignement doit être spécifiée. Le guide complet des fonctions intrinsèques d'Intel est disponible à l'adresse suivante: `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`. Il est possible de transmettre la taille de l'alignement à la carte d'Intel Xeon Phi Coprocessor par une option dans les directives pragma. Enfin, toutes ces optimisations permettent au compilateur d'opérer une vectorisation sur le code, il en ressort un gain en performance incomparable avec les optimisations précédemment énoncées.

5.4.5 Affinité des Processus Logiques

La gestion de l'affinité des processus logiques permet également de les "fixer" sur un cœur physique et par conséquent créer une certaine inertie des données, en général ce comportement augmente la vitesse de calcul en évitant au cœur de faire des instructions inutiles. Heuristiquement, nous avons montré que le mode "compact"[22](une cartographie spécifique des processus logiques) donnait les meilleurs résultats en terme de performance de calcul pour l'Intel Xeon.

[21] C'est une structure en 3 dimensions qui a été linéarisée pour qu'on puisse opérer la vectorisation sur la boucle la plus profonde

[22] Le mode "compact" assigne le processus logique $n+1$ à la localité physique (processeur) la plus proche et libre de celle du processus logique n, ce comportement est l'inverse de "scatter".

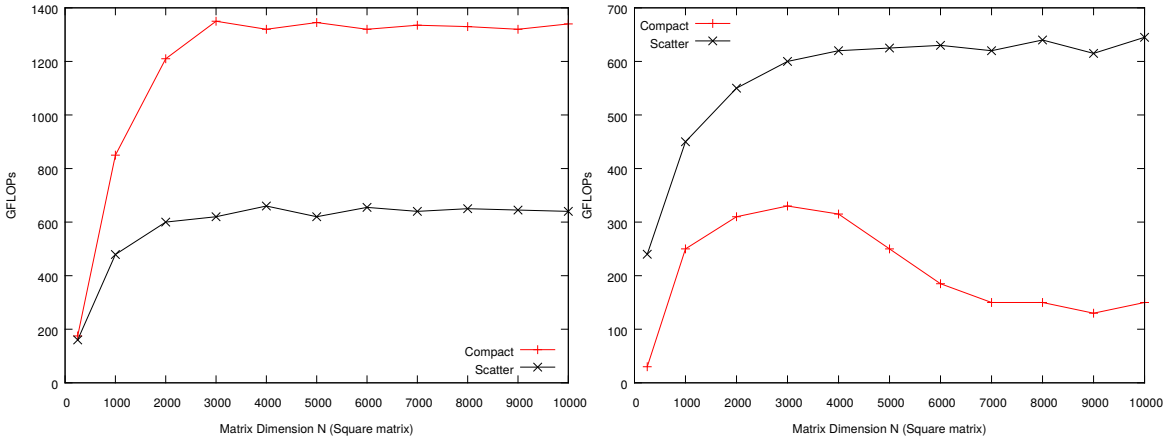

Figure 5.8: Performance observée en GFLOPs en fonction du paramètre N sur la multiplication de matrices carrées denses de dimension $N \times N$. A droite, on peut observer les performances réalisées sur l'Intel Xeon Phi Coprocessor. A gauche, sur l'Intel Xeon. Les architectures utilisées sont spécifiées dans la section 5.5

Et pour l'Intel Xeon Phi Coprocessor, la meilleure configuration s'est avérée être "balanced"[23]. Pour notre cas test, ces changements n'ont pas un grand impact sur les performances de calcul mais la gestion de l'affinité des processus logiques peut être un aspect critique. En effet, on peut observer la différence de performance sur la figure 5.8. On a testé la puissance des deux machines sur la multiplication de matrices carrées à l'aide de la bibliothèque optimisée de calcul algébrique BLAS[24] level 3e t tester deux modes de comportement différents i.e. "compact" et "scatter". Les résultats confirment ce qu'il a été enoncé précédemment.

5.4.6 Architecture Hybride

Une des dernières optimisations possible concernant les spécificités de la machine et de son environnement est l'utilisation simultanée de l'Intel Xeon et de l'Intel Xeon Phi Coprocessor. Cela a été possible en suivant le protocole Pthreads (Posix threads programming, programmation détaillée par Buthenhof dans Butenhof [38]). Sur une architecture de multiprocesseurs physiques à mémoire partagée, il est possible d'implémenter du parallélisme à l'aide de ce protocole. Pthreads est un langage d'interfaçage de programmation de processus logiques en C. Une implémentation efficiente permet de bénéficier au maximum de l'architecture hybride. La répartition des tâches et la masse de travail envoyées à l'hôte et à la carte accélératrice est réalisée expérimentalement. Une répartition optimale est nécessaire pour éviter que l'une des machines ne terminent sa tâche de travail bien avant l'autre donc de profiter au maximum de l'architecture sur laquelle nous travaillons. Nous avons trouvé que la répartition optimale pour notre configuration est un tiers de la masse de travail est donné à l'hôte et le reste est donné à la carte. Cette répartition sera probablement différente pour une autre architecture.

Sur le code 5.4, la création des processus logiques pour l'Intel Xeon et l'Intel Xeon Phi

[23] Le mode "balanced" place les processus logiques sur chaque cœur physique jusqu'à que tous les cœurs disposent d'au moins une tâche à accomplir. Ce mode de fonctionnement n'est disponible que sur l'Intel Xeon Phi Coprocessor.

[24] Available at `http://www.netlib.org/blas/`.

Coprocessor se fait simplement à l'aide des fonctions:

```
pthread_create( &threadCpu, NULL, OnCPU, (void*)cpu);
pthread_create( &threadMic, NULL, OnMIC, (void*)mic);
```

Code 5.4: Création des processus logiques avec Posix Threads.

où `threadCpu` et `threadMic` sont deux variables de type `pthread_t` qui est un Posix thread. Et les entités `OnCPU` et `OnMIC` sont les fonctions/applications qui doivent être exécutées respectivement sur l'Intel Xeon et sur l'Intel Xeon Phi Coprocessor. Sur le code 5.5, le lancement des processus logiques sur chaque architecture se fait avec

```
pthread_join( threadCpu, NULL);
pthread_join( threadMic, NULL);
```

Code 5.5: Exécution des processus logiques avec Posix Threads.

On a, bien entendu, copier toutes les données nécessaires à l'évaluation des contrats sur la carte avant de lancer le Posix thread dessus.

Dans un deuxième temps et avec l'aide de 2CRSI, nous avons pu tester une architecture différente c'est-à-dire un serveur avec six cartes accélératrices comme représentée sur la figure 5.9.

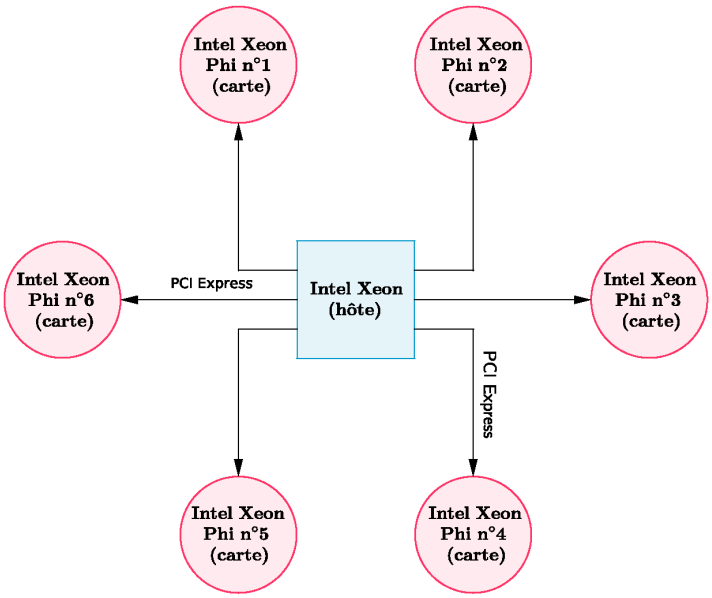

Figure 5.9: Architecture hybride à six cartes accélératrices reliées par des bus PCI express au server Intel Xeon.

Cette architecture procure la possibilité de disposer de près de 13 000 GFLOPs de puissance de calcul en simple précision et de 6 500 GFLOPs en double précision i.e. tableau 5.2 et mais

il faut prendre en compte que les six cartes sont branchées en port PCI express au serveur. Il n'y a qu'un hôte et on verra dans les tests que le temps de chargement de données devient le point critique de l'algorithme. Effectivement, le transfert de données se fait séquentiellement sur les cartes accélératrices. Il n'y pas de parallélisation dans l'envoie de donnée ou il faudrait imaginer une structure d'algorithme différente. Comme faire du parallélisme asynchrone ou faire du streaming de donnée. Malheureusement, nous n'avons pas eu le temps de nous étendre sur ces deux options.

5.4.7 Option de Compilation

Voici les options de compilation utilisées pour augmenter les performances de calcul du code généré, celles-ci sont décrites exhaustivement dans le guide du compilateur d'Intel C++ [25] . Pour les tests, nous avons utilisé le compilateur d'Intel C++ v15.0.

- `-Ofast` active toutes les options d'optimisation de code et défère la précision de la division en changeant la multiplication de A/B par A * (1/B) c'est-à-dire en calculant directement l'inverse de B (équivalent à l'option de compilation `-no-prec-div`). Ce qui permet de calculer plus rapidement toutes les divisions.

- `-qopenmp` permet à l'utilisateur de générer du code parallélisé basé sur les directives OpenMP

- `-fno-alias` fait la supposition qu'il n'y a pas d'inter-dépendance entre les pointeurs, Le compilateur ne fera pas les vérifications usuelles et donc réduit le temps de calcul.

- `-align` analyse et réarrange la structure des données des variables et des tableaux pour faire de la vectorisation et diminuer les accès en mémoire. Cela force la contig´uité en mémoire.

- `-mavx` génére les instructions d'Intel de vectorisation AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2 et SSE.

- `-funroll-loops` déroule les boucles pour diminuer le nombre d'instructions à exécuter, la contrepartie est que le code généré est plus long.

5.4.8 OpenMP

L'interface d'application de programmation (API) OpenMP permet de faire de la parallélisation au niveau d'une même machine, c'est-à-dire à l'exécution simultanée de plusieurs tâches sur des cœurs distincts.. Il faut que les cœurs physiques reposent sur une architecture à mémoire partagée. Ce protocole brille par sa simplicité d'utilisation, sa flexibilité ainsi que sa portabilité. Il permet soit de faire tourner des petites applications en même temps ou à paralléliser des boucles de calculs. Cette API vise les plateformes de langage de programmation C, C++ et Fortran. Il est possible de le combiner à du langage MPI[26] sur une grille de serveurs pour profiter au plus de toutes les spécificités d'un supercalculateur. Le niveau de parallélisation n'est pas le même que pour MPI ou Posix threads. La gestion des processus logiques est transparente et accessible

[25] Intel C++ Compiler User and Reference Guides, 2008, disponible à l'adresse suivante: `http://www.physics.udel.edu/~bnikolic/QTTG/shared/docs/intel_c_user_and_reference_guide.pdf`

[26] Le protocole MPI servira à faire de la parallélisation inter-serveur alors que le protocole d'OpenMP servira à faire de la parallélisation intra-serveur.

via quelques options dans les directives dans le code et/ou par variables d'environnement dans un terminal de contrôle.

Il existe quelques contraintes à ce langage de programmation comme la non hiérar- chisation de la mémoire, les problèmes de localité des données, une efficacité non garantie (dûe à la structure de la machine) ou encore une scalabilité limitée. Il n'est pas possible de faire de la parallélisation "automatique" en ajoutant une ou plusieurs options de compilation à l'encontre de la vectorisation. On sera obligé de modifier le code mais les changements ne sont pas intrusifs et ne demande pas un refactoring complet du code. Par exemple, pour paralléliser une boucle de calcul, il suffira de passer d'ajouter une directive en amont de la boucle `for` comme indiqué sur l'exemple de code 5.6 ci-après,

```
float call_price = 0.f;

#pragma omp parallel for reduction( + : call_price )
for( unsigned int i = 0; i < N; ++i )
        call_price += maxf( S[i] - K, 0.f );

call_price /= ( float ) N;
```

Code 5.6: Exemple d'une section OpenMP programmée en C++.

Il est possible d'accéder à la gestion des tâches à exécuter et à la répartition des données aux processus logiques ainsi qu'au nombre de processus logiques créés et distribués. Le mot clé `schedule(option)` permet cette gestion, les options peuvent être:

- `static`, les itérations sont divisées en taille de données qui sont réparties selon une équidistribution sur les processus logiques. Si par malheur une itération est plus longue qu'une autre, on pourrait se retrouver avec un ou plusieurs processeurs inactifs, ce qui n'est pas optimal si le temps de calcul d'une itération à une autre n'est pas identique.
- `dynamic`, chaque processus exécute un nombre d'itérations puis redemande un nouveau nombre d'itération à procéder. Cette option permet d'avoir une meilleure répartition des tâches et permet d'avoir une occupation optimale des processeurs.
- `guided`, le nombre d'itérations est réparti par tour sur tous les processus logiques et est de plus en plus petit.
- `auto`, laisse le contrôle au compilateur de décider quelle est la meilleure répartition.

Nous utilisons le mode `dynamic` pour la gestion et le nombre de processus logiques qui peuvent être gérés sans diminution de performance. Sur des boucles de calcul simple mais suffisamment longue, le facteur d'accélération est pratiquement égale au nombre de cœurs physiques utilisés.

5.4.9 Algorithme du Moteur de Calcul à Architecture Hybride

La première étape du cas test, consiste à charger les données sur le serveur. La lecture n'étant pas optimisée sur l'Intel Xeon Phi Coprocessor, nous laissons la tâche de l'organisation, de l'allocation du portefeuille d'actifs financiers et de la structuration des données au serveur. Les données sont chargées à partir de fichier CSV, on trouve en entrée les taux de change anticipés à la vue d'aujourd'hui et les données des contrats des SWAPs et des CAPs/FLOORs. Les données sont structurées en tableaux alignés par taille de 256 bit pour le serveur et de 512 bit pour les

cartes accélératrices. La simulation et la diffusion des risques facteurs sont stockées de façon à obtenir de la vectorisation par vecteur de 256 ou 512 bit dans la boucle de calcul la plus profonde. On a rajouté une calibration de la répartition optimale de la charge de travail associée à chaque machine, on exécute un calcul pour tester le rapport de puissance entre les machines. Par la suite, c'est ce ratio que l'on utilisera pour la division des tâches de travail. Lorsque cette calibration est terminée, on passe à l'initialisation et à l'exécution des processus logiques pour chaque machine à l'aide du protocole Posix threads.

La deuxième étape consiste à transférer les données sur la ou les cartes accélératrices en fonction du ratio précédemment déterminé. Une fois le transfert terminé, on débloque une variable de verrou et on passe au processus d'évaluation de façon synchronisée entre toutes les machines. On répartit de façon dynamique et optimale la charge de travail affectée à une même machine sur tous les cœurs de celle-ci à l'aide du protocole de parallélisation OpenMP. La phase d'évaluation en toutes les dates de paiements est donc lancée.

La troisième étape consiste à rapatrier les données se trouvant sur chaque carte accélératrice vers le serveur et à reconstituer la grille d'évaluation du portefeuille sur chaque pas de temps pour tous les contrats. Enfin, on désalloue toutes les données sur les cartes et sur le serveur. Sur l'algorithme 8, un récapitulatif des étapes à procéder du moteur de calcul à architecture hybride.

Algorithm 8 Moteur de calcul pour machine hybride

Requis: Données de marché, courbe de taux, contrats
Sortie: Grille d'évaluation du portefeuille, temps de calcul
1: Lecture des données de marché à partir des fichiers CSV
2: Allocation des données sur le serveur
3: Génération des contrats selon la distribution réelle d'un portefeuille
4: Simulation et diffusion des facteurs de risques
5: Structuration des données en tableaux et alignement en mémoire
6: *(Optionnel): Calibration de la répartition optimale par rapport à l'architecture*
7: Répartition optimale des données à évaluer entre les machines
8: Allocation et copie des données sur le ou les Intel Xeon Phi Coprocessor
9: Initialisation des processus logiques (Posix threads)
10: Lancement des applications d'évaluation sur les machines
11: Répartition équilibrée des tâches sur les cœurs d'une même machine (OpenMP)
12: Évaluation à toutes les dates de paiements (par calcul vectoriel)
13: Copie des données de la ou les cartes accélératrices vers l'hôte
14: Reconstitution de la grille d'évaluation du portefeuille
15: Désallocation des données sur les machines

En appendice E, un exemple de la sortie de l'implémentation de l'algorithme 8 sur une architecture hybride comportant un serveur et six cartes accélératrices.

5.5 Tests et Mesures

5.5.1 Un Serveur avec Une Carte Accélératrice

Tous les tests pour l'architecture hybride avec un hôte et une carte accélératrice ont été réalisés sur les deux modèles de machines. L'hôte est un Intel Xeon version E5-2680 (2 sockets), 32 Go de RAM les cœurs sont hyperthreadés donc il dispose de 16 processeurs physiques et de 16

cœurs virtuels. Il est possible de faire tourner 32 processus logiques simultanément. Le nom de la famille de la micro-architecture du Xeon est Sandy Bridge, les processeurs sont cadencés à 2.70 GHz. La carte Intel Xeon Phi Coprocessor est la version x100, 16Go de RAM, à 61 cœurs physiques, hyperthreadés à 244 processus logiques en mode natif et 240 en mode offload. Les processeurs sont de la famille Pentium 1, cadencés à 1.24 GHz. Les performances théoriques en simple précision et double précison sont inscrites dans le tableau 5.2. On peut noter que les performances observées sur la figure 5.8 correspondent à peu près aux performances théoriques annoncées en simple précision. Les résultats en double précisions correspondent également aux performances attendues.

Précision	CPU	MIC	Hybride (1 x MIC)	Hybide (6 x MIC)
Simple	660	2047	2707	12942
Double	330	1023	1353	6468

Table 5.2: Puissance théorique en GFLOPs pour les architectures utilisées en simple précision et double précision.

La première version du moteur de calcul d'évaluation de portefeuille massif d'actifs financiers exhibait des temps de calcul d'un peu plus d'une heure sur l'Intel Xeon. Sur la carte accélératrice, en mode natif nous obtenions un temps de calcul autour de la demi-heure; ce qui correspond aux performances attendues, c'est-à-dire 2 fois plus puissante que l'hôte. Tous les résultats succédants un gain en performance après chaque optimisation n'a malheureusement pas été gardés, nous exhibons seulement les résultats finaux.

Cette version du problème test est effectuée séparément sur des portefeuilles allant jusqu'à 100 000 SWAPs et 100 000 CAPs/FLOORs, on ne considère qu'un seul facteur de risque. La distribution et les spécificités des portefeuilles ont été établies sur des portefeuilles d'une grande banque française et tente de calquer au plus la réalité. La génération des contrats a tout de même été simplifiée pour diminuer le nombre d'interpolation à faire sur les dates de paiements. Sur les tableaux 5.3 et 5.4, on peut observer les performances en terme de temps de calcul (exprimé en seconde) sur un portefeuille contenant uniquement des SWAPs, les calculs sont faits en précision simple. On a testé trois types d'implémentation différentes[27]:

- Un code visant uniquement l'Intel Xeon.
- Un code visant l'architecture hybride en mode offload (faisant travailler simultanément les deux machines).
- Un code visant uniquement l'Intel Xeon Phi Coprocessor en mode natif

Une fois encore, les résultats viennent confirmer nos attentes et les temps de calculs sont infimes en comparaison de la première version implémentée. On aura les mêmes conclusions pour les résultats obtenus en double précision. Ces résultats sont exhibés dans les tableaux 5.5 et 5.6.

5.5.2 Un Serveur avec Six Cartes Accélératrices

Avec l'aide de 2CRSI, nous avons pu tester une architecture comportant un serveur et six cartes accélératrices. La configuration est pour l'hôte, on dispose d'un Intel Xeon E5-2630 v2 (2 sockets), 12 cœurs physiques et 12 cœurs virtuels. Il est possible de faire tourner 24 processus logiques et on

[27] le code ne change pratiquement pas, mise à part quelques options de compilation et d'options dans certaines directives

Contrats	CPU	MIC (natif)	Hybride (offload)
5000	4.75	2.32	1.60 (-0.58)
10000	9.21	4.01	2.50 (-0.73)
25000	25.12	11.79	7.76 (-1.15)
50000	48.93	22.36	14.25 (-1.38)
75000	73.24	32.58	21.08 (-1.56)
100000	86.04	40.03	27.09 (-1.89)

Table 5.3: Temps de calculs en seconde pour plusieurs tailles de portefeuilles contenant uniquement des SWAPs. Le calcul (double précision) est testé sur trois architectures. On exhibe les temps de chargement des données pour l'architecture hybride.

Contrats	CPU	MIC (natif)	Hybride (offload)
5000	5.11	2.89	1.98 (-0.54)
10000	10.39	4.56	3.15 (-0.61)
25000	24.74	10.95	7.82 (-0.89)
50000	49.32	23.84	15.26 (-1.01)
75000	74.99	35.74	22.44 (-1.14)
100000	89.81	41.35	28.02 (-1.27)

Table 5.4: Temps de calculs en seconde pour plusieurs tailles de portefeuilles contenant uniquement des CAPs/FLOORs. Le calcul (simple précision) est testé sur trois architectures. On exhibe les temps de chargement des données pour l'architecture hybride.

Contrats	CPU	MIC (natif)	Hybride (offload)
5000	19.02	9.16	5.92 (-0.58)
10000	37.34	18.30	11.60 (-1.07)
25000	102.43	49.86	31.05 (-1.99)
50000	190.59	94.02	57.48 (-2.52)
75000	286.49	139.74	86.18 (-3.13)
100000	363.99	177.85	108.13 (-3.48)

Table 5.5: Temps de calculs en seconde pour plusieurs tailles de portefeuilles contenant uniquement des SWAPs. Le calcul (double précision) est testé sur trois architectures. On exhibe les temps de chargement des données pour l'architecture hybride.

Contrats	CPU	MIC (natif)	Hybride (offload)
5000	18.78	9.75	6.02 (-0.27)
10000	35.83	19.62	12.21 (-0.29)
25000	114.62	48.63	32.37 (-0.39)
50000	183.02	97.05	62.08 (-0.56)
75000	273.49	135.96	84.97 (-0.82)
100000	376.34	183.12	112.33 (-1.11)

Table 5.6: Temps de calculs en seconde pour plusieurs tailles de portefeuilles contenant uniquement des CAPs/FLOORs. Le calcul (double précision) est testé sur trois architectures. On exhibe les temps de chargement des données pour l'architecture hybride.

dispose de 64 Go de RAM. Les cartes accélératrices sont du même modèle que sur l'architecture testée précédemment. Le problème test est similaire à celui de la section 5.5.1 à la différence que l'on génère un portefeuille massif d'actifs financiers sur plusieurs milliers de risques facteurs (5 000 sous-jacents). Par conséquent, la masse de données à transférer aux cartes est dramatiquement élevée. En effet, il faudra charger plus d'une dizaine de Go de données par carte accélératrice.

Cette spécificité nous rapproche encore plus de la réalité d'un vrai portefeuille d'une grande banque. Dans les résultats, nous n'exhibons pas les temps de simulations et de diffusion des risques facteurs car ceux-ci sont normalement gérés par d'autres systèmes d'information au sein d'une banque. Le travail ici s'effectue seulement sur l'évaluation des actifs financiers.

Sur le tableau 5.7, on peut observer les performances en seconde pour du calcul en simple précision[28] sur l'architecture à six cartes accélératrices. On évalue jusqu'à 100 000 SWAPs et 100 000 CAPs/FLOORs indépendamment. La dernière colonne exhibe des temps de calculs pour des portefeuilles comportant jusqu'à 200 000 produits financiers. on peut voir que pour les SWAPs et les CAPs/FLOORs, les temps de calculs correspondent à peu près à ce qu'on attendait en terme de performance. Si on compare ceux obtenus avec l'architecture précédente, on obtient un ratio de 4 sachant que la configuration du serveur est plus puissante que celle-ci.

Contrats	SWAPs	CAPs/FLOORs	SWAPs + CAPs/FLOORs
5000	0.64	1.01	1.72
10000	1.18	1.82	3.03
25000	2.43	4.01	6.50
50000	4.53	6.38	10.92
75000	5.93	7.88	13.87
100000	7.63	8.68	16.42

Table 5.7: Temps de calculs en seconde pour plusieurs tailles de portefeuilles contenant uniquement des SWAPs, des CAPs/FLOORs et les deux produits. Le calcul est en simple précision.

Malheureusement, il faut rajouter les temps de transfert à ces temps de calculs. Pour 1 000 simulations par risques facteurs, il faut en tout 40 secondes de temps de transfert sur les cartes. Si on compare ce temps avec les temps de calculs, on voit que le temps de chargement des données devient le nœud critique de l'algorithme[29]. Ce temps est six fois plus important que pour l'évaluation de 100 000 SWAPs. Pour 5 000 simulations par risques facteurs, on obtient 205 secondes de temps de transfert et pour 10 000 simulations, 404 secondes. Cette problématique implique d'avoir une autre approche sur la configuration de l'architecture ou une implémentation différente du transfert de données sur les cartes.

Enfin sur la figure 5.10, on peut observer les ratios de facteurs d'accélération avec un seul serveur[30]. Les résultats correspondent pratiquement aux performances théoriques attendues, c'est-à-dire qu'une carte accélératrice est 2 fois plus puissante que l'Intel Xeon, l'architecture hybride montre 3 fois celle du serveur. Et l'architecture à six cartes accélératrices développe 13 fois plus puissant que l'hôte.

Comme ses travaux s'inscrivaient dans une démarche de démonstration de performance aux industries, il est important d'examiner quelques caractéristiques aux architectures proposées. Pour une entreprise, il est nécessaire de connaître, pour minimiser les dépenses, l'énergie consommée et son rapport qualité prix. Sur le tableau 5.8, on peut observer les prix de chacune

[28] nous n'avions pas assez de RAM pour faire du calcul en double précision

[29] Il faut savoir qu'une fois les données des risques facteurs simulées, on les transfert pas plusieurs fois à chaque demande d'évaluation du portefeuille. Les données sont statiques.

[30] La comparaison se fait avec la deuxième configuration

des architectures qui ont été testées. La consommation en électricité d'un serveur Intel Xeon est 3 fois plus élevée qu'une carte accélératrice Intel Xeon Phi Coprocessor et son prix est 2 fois plus grand. Donc une architecture à plusieurs cartes accélératrices permet d'obtenir le plus de puissance de calcul au prix le moins élevé. Et à l'heure où l'on parle sans cesse de réchauffement climatique, il est important d'étudier l'empreinte carbone de chaque configuration.

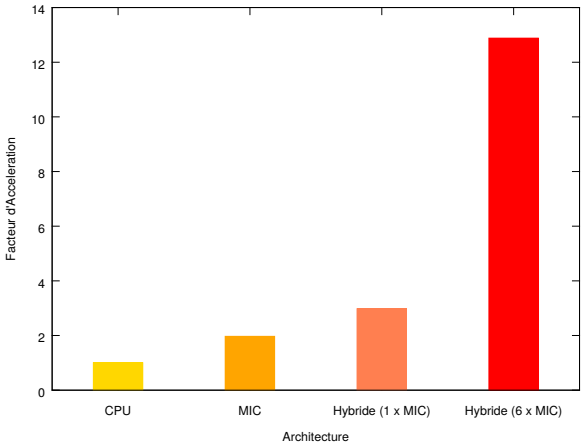

Figure 5.10: Facteur d'accélération développé en comparaison d'une architecture composée uniquement d'un serveur avec comme configuration Intel Xeon E5-2630 v2.

–	CPU	MIC	Hybride (1 x MIC)	Hybide (6 x MIC)
Energie (Watt/h)	423	188	611	1551
Consommation (euro)	296	132	428	1087
Carbone ($kgCO_2$/an)	1853	824	2676	6793
Machine (euro)	1800	800	2600	6600

Table 5.8: Énergie consommée en Watt par heure, prix de chaque architecture, prix de la consommation en électricité à l'année et émission de carbone par an en kilogramme des différentes configuration testées.

5.6 Un Code, Plusieurs Architectures

5.6.1 Carte Graphique

Au cours des années 2000, l'augmentation des fréquence de cadençage des nouveaux processeurs a connu une dure réalité et s'est exposée à un plafond de verre.[31] Ce problème a imposé de réfléchir autrement à la conception de machine plus puissante et a introduit un nouveau paradigme de

[31] La fréquence de cadençage des processeurs est dûe au nombre de transistors engravés sur une puce électronique. Il y a deux facteurs limitants: la taille des gravure (de plus en plus petite) a atteint une limite physique que nous sommes plus capable de dépasser aujourd'hui et la chaleur dégagée est de plus en plus grande. Cette chaleur peut faire fondre la puce, il faut donc des systèmes de refroidissement toujours plus gros et plus coûteux. Ce qui ne s'accorde pas avec la demande du marché, c'est-à-dire la démocratisation de l'accès à cette technologie et la concurrence des prix toujours plus bas.

programmation i.e. le parallélisme. Au lieu d'augmenter la fréquence de cadençage des processeurs, la solution choisie a été de produire des puces multiprocesseurs et de faire les faire fonctionner simultanément. La demande en ressource de calcul de plus en plus puissante augmente drastiquement avec l'essor d'internet et le début du data mining (prologue du big data). L'inconvénient est que ces nouvelles machines sont extrêmement coûteuses, par exemple IBM BlueGene qui développe 2 teraFLOPs de puissance de calcul vaut aux alentours de 200 000 euro alors qu'une carte graphique Nvidia Tesla S870 n'en vaut que 1 500 euro pour la même puissance. C'est dans ce contexte particulier que la carte graphique a pris son essor et avec l'avènement du parallélisme. En effet, une carte graphique est composée de plusieurs dizaines, centaines voire milliers de petits processeurs et d'unité de calcul scalaire comme représentée sur la figure 5.11, la différence architecturale entre les CPUs et les GPUs.

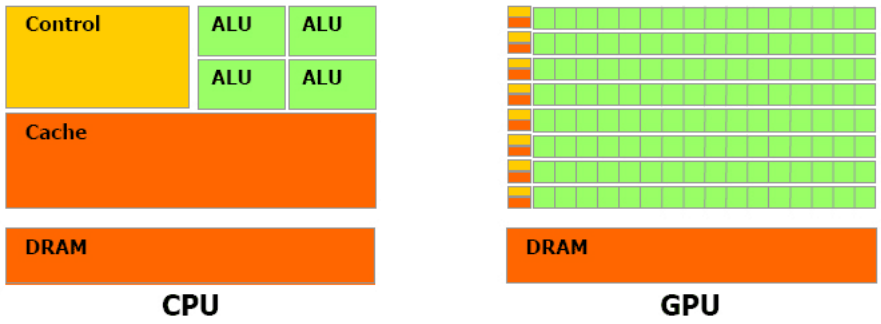

Figure 5.11: Schéma d'architecture d'un CPU et d'un GPU. Ce dernier comporte beaucoup plus d'unités de calcul qu'un CPU.

5.6.2 Architecture Nvidia et CUDA

Selon le guide programmation de Nvidia[32], CUDA[33] est une extension du langage de programmation C permettant aux programmeurs de définir des fonctions en C et d'appeler des noyaux d'applications aussi appelés kernel qui sont exécutés N fois en parallèle à travers N processus logiques CUDA sur une carte graphique. Un noyau d'application est défini par le mot clé `__global__`, ce mot doit être placé en amont de la déclaration d'une fonction. Les fonctions qui sont uniquement destinées à fonctionner sur la carte graphique ont le mot clé `__device__`. La configuration du parallélisme et la structuration de l'algorithme sur la grille de calcul sont données en entrée lors de l'appel de cette fonction par la syntaxe `<<<...>>>`. Chaque processus logique qui exécute le noyau d'application est attribué par une unique identité d'indexation accessible par la variable syntaxique `threadIdx`.

5.6.2.1 Processus Logique

Par convention, `threadIdx` est un vecteur en 3 dimensions donc chaque processus logique peut être identifié selon une indexation en une, deux ou trois dimensions formant des blocs de processus

[32]GPU Programming Guide, 2006, disponible à l'adresse suivante: `https://developer.download.nvidia.com/GPU_Programming_Guide/GPU_Programming_Guide.pdf`

[33]`http://docs.nvidia.com/cuda/`, on pourra retrouver des exemples de programmation dans Sanders et Kandrot [170].

en une, deux ou trois dimensions. Les blocs regroupent un paquet de processus logiques, sur les premières générations de GPU un bloc ne pouvait contenir que 512 processus alors que sur les dernières générations, un bloc contient jusqu'à 1024 processus. Il est possible de générer une grille de 65 536 blocs en une, deux ou trois dimensions, voir figure 5.12 de représentation d'une grille . L'identité d'un bloc est accessible via le mot clé `blockId`. L'indexation globale d'un processus logique est retrouvé à partir d'un peu d'arithmétique. Par exemple, sur un bloc en trois dimensions de paramètres (n_x, n_y, n_z) sur une grille à une dimension (n_u), l'indexation est définie par la relation suivante:

$$un_x n_y n_z + x + yn_x + zn_x n_y. \tag{5.5}$$

Pour une grille en 3 dimensions de paramètres (n_u, n_v, n_w) avec des blocs en 3 dimensions:

$$n_x n_y n_z (u + vn_u + wn_u n_v) + x + yn_x + zn_x n_y. \tag{5.6}$$

Ce choix est laissé à l'utilisateur en fonction de son confort et de la structure de l'algorithme mais cela procure une façon naturelle de calquer les processus logiques aux éléments de calcul vectoriel, matriciel ou tensoriel. Le choix de la représentation de la grille ne joue pas en rôle en terme de performance. Les processus logiques au sein d'un même bloc sont exécutés simultanément

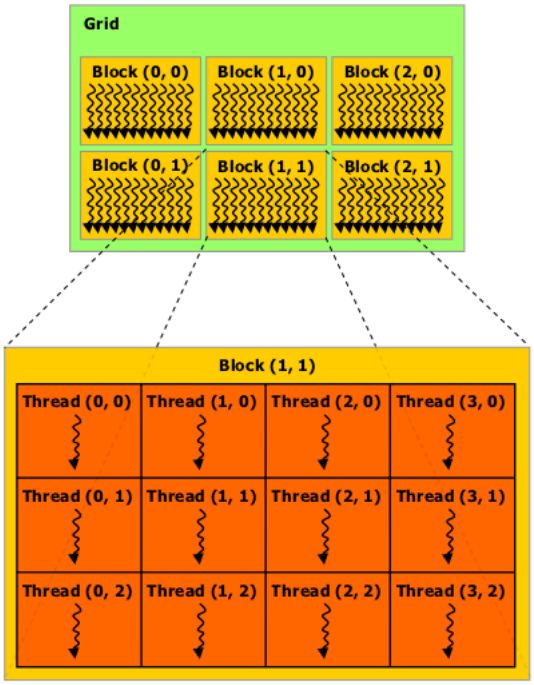

Figure 5.12: Réprésentation de la grille découpée en blocs sous-découpés en processus logiques.

(conceptuellement et dans la limite physique du nombre d'unité de calcul disponible) mais les processus de différents blocs sont complètement indépendants et reposent sur des mémoires indépendantes. Une programmation optimisée demandera de connaître le nombre de processeurs de la machine pour profiter au plus de ses spécificités.

5.6.2.2 Structure Mémorielle

Les processus logiques d'une carte graphique peut avoir accès à une mémoire multi-niveaux pendant leurs exécutions. Chaque processus a sa propre région privée de mémoire accessible uniquement par lui-même. Chaque bloc dispose d'une mémoire partagée auquel tous les processus de ce bloc ont accès mais pas ceux d'un autre bloc. Tous les processus ont accès à la mémoire globale. Il y a en addition trois zones de mémoire spécifiques; la mémoire constante, locale et de texture. Ces trois mémoires sont des espaces optimisés pour des usages différents. Par exemple, la mémoire de texture a un adressage spécial pour des formats de données spécifiques. Les mémoires globales, constante et de texture sont persistantes pendant l'exécution du noyau d'application alors que les mémoires partagées, locales et registres sont éphèmères. On peut retrouver sur la figure 5.13 un schéma représentatif de l'architecture mémorielle d'une carte graphique.

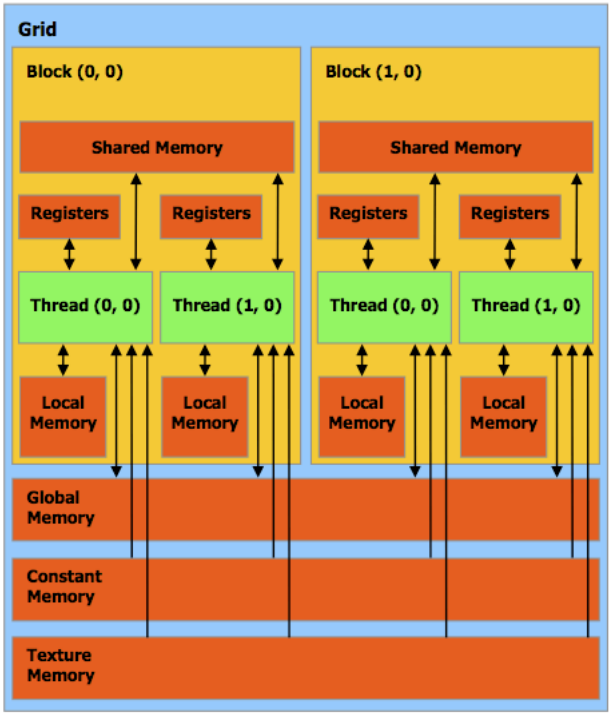

Figure 5.13: Réprésentation hiérarchique des différentes mémoires sur une carte graphique.

Plus précisément, ces mémoires ont d'autres caractéristiques: la mémoire globale et locale sont sur la carte graphique et accessibles par paquet de 32, 64 ou 128 bit (paramètre à prendre en compte pour optimiser les accès mémoire). Ces accès mémoires sont alignés et très lents, voir Barlas [12] et Kirk et Hwu [113]. La mémoire partagée et les registres sont placés directement sur la puce donc ils disposent d'accès très rapide. Les accès à la mémoire partagée sont très rapide s'il n'y a pas de conflits d'indexation entre les banques de mémoires et les processus logiques. La mémoire constante et de texture résident sur la carte graphique et sont accessibles par l'hôte ainsi que la mémoire globale. Les accès sur les deux premières sont plus rapides que ceux à la mémoire globale.

5.6.2.3 Caractéristiques

Une spécificité à prendre en compte des cartes graphiques est qu'elles sont hautement optimisées dans la gestion et le calcul des nombres à virgule flottante en simple précision. Le calcul double précision est encore à la traîne, par exemple la Nvidia Tesla K10 développe 4.58 teraFLOPs en simple précision mais seulement 0.19 teraFlops en double précision. Sur les architectures plus récentes, le retard commence à être comblé mais n'exhibe toujours pas un rapport de 1 à 2 avec le calcul simple précision. Un des gros inconvénients du calcul sur carte graphique est le temps de copie des données entre l'hôte et celle-ci. Ce point important doit être pris en compte lors de présentation de performance entre deux dispositifs différents.

Les spécifications des langages de programmation CUDA et OpenCL n'en disent pas beaucoup sur l'architecture sur lequel se repose ces interfaces. Or il est primordial pour le développeur de connaître les caractéristiques de la machine à un très bas niveau pour pouvoir produire un code très efficient. Les spécificités des cartes Nvidia dépendent énormément de l'attribut "capability". Par exemple, les cartes graphiques qui ont une capability en dessous de 3.0 ne peuvent pas faire de l'I/O[34]. Les cartes graphiques qui ont une capability 3.5 sont capables de faire du parallélisme dynamique, ce qui permet à un kernel ou une application exécutée sur la carte de lancer un autre kernel avec une configuration de grille indépendante de la première.

Nvidia se réfère a ces unités de calcul en tant que "multithreaded streaming processeurs", c'est-à-dire à des multiprocesseurs capable de gérer plusieurs processus logiques en même temps. Ces unités de calculs comportent une architecture dite SIMT[35] où une instruction pour plusieurs processus simultanément. Chaque multiprocesseurs est capable d'exécuter des fils d'instructions en même temps pour plusieurs processus logiques. Les processus d'un bloc s'exécute simultanément sur le même multiprocesseur et plusieurs blocs peuvent être gérés simultanément par un multiprocesseur. Chaque bloc est partitionné en warp ou grappe. Sur les machines contemporaines, chaque grappe est composée de 32 processus logiques indexés de façon consécutives et par ordre croissant au sein d'un même bloc. C'est une composante importante à connaître lors de l'implémentation d'une solution, on prendra par exemple un multiple de 32 comme taille de dimension des processus logiques par bloc. On peut omettre la synchronisation des processus si on est sûr qu'il exécute tous la même instruction en même temps.

Chaque processus logique d'une grappe a son propre fil d'instruction et son registre de telle sorte qu'il peut faire des opérations indépendantes des autres processus appartenant à la même grappe. L'état de chaque grappe est stocké directement sur la puce pendant le temps de vie de celui-ci. Une grappe exécute les instructions d'un code une à la fois et durant chaque cycle, un gestionnaire de grappe en sélectionne une qui est prête à exécuter la prochaine instruction. Les processus évoluent de façon "symétrique" , c'est-à-dire qu'ils sont tous à peu près au même point chronologique de l'application à exécuter.

En cas de divergence de fil d'instruction au sein d'une grappe entre un ou plusieurs processus logiques, les différents fils sont exécutés séquentiellement. Les processus logi- ques qui n'exécutent pas l'instruction sont désactivés au moment du passage sur le multiprocesseur. Par exemple, dans le cas d'un processus de branchement par une condition `if`, en supposant que les 16 premiers processus effectuent l'instruction si elle est remplie et les autres le cas contraire. La chronologie sera condition remplie puis condition non remplie ce qui prendra plus de cycle que s'ils avaient eu tous la même instruction à exécuter. Pour une implémentation performante, il faudra éviter au plus les processus de branchement. Les ressources d'une unité de calcul sont partitionnées entre les différents blocs qui sont exécutés simultanément, ce qui limite donc le nombre de bloc

[34]Lecture et écriture en sortie de terminal
[35]Single instruction, multiple threads.

sur le multiprocesseur. En général, ce nombre se limite à 8 blocs.

Une des plus gros contraintes de la programmation sur carte graphique est que le processus de débogage, qui est pratiquement inexistant et il y a beaucoup moins de vérifications sur le calcul que sur un CPU. La latence qui est le nombre de cycle d'horloge que peut prendre une grappe avant d'être prête à exécuter la prochaine instruction a un gros impact en terme de performance. Une performance maximum est atteinte seulement si le gestionnaire de grappe émet une instruction pour une grappe par cycle. Les accès en mémoire provoquent de la latence, en particulier si la mémoire globale est constamment sollicitée. Ce qui peut être équivalent à 400-600 cycles d'horloge alors que l'accès à la mémoire partagée n'en prend que 40.

Grâce à une analyse poussée du code généré à l'aide de certaines options de compilation, en examinant le fichier `.ptx` et les différentes tailles de mémoire utilisées, on peut déduire une configuration optimale de la grille pour maximiser le pourcentage d'occupation des multiprocesseurs.

5.6.2.4 Exemple de Code

```
__global__ void vectorMulScalar( float *A, float c )
{
        int idx = blockIdx.x * blockDim.x + threadIdx.x;
        A[idx] = A[idx] * c;
}

int main( void )
{
        int nT = 4, nB  = 2,  n = nT * nB;
        int t = n * sizeof( float );

        float * hA = ( float* )malloc( t ), * dA, c = 0.5;
        int t = n * size(float);

        for( int i=0; i<n; ++i ) hA[i] = (float) i;

        cudaMalloc( (void**) &dA, t );
        cudaMemcpy( dA, hA, t, cudaMemcpyHostToDevice );
        vectorMulScalar<<< nB, nT >>>( dA, c );
        cudaMemcpy( hA, dA, t, cudaMemcpyDeviceToHost );

        cudaFree( dA );
        free( hA );

        return 0;
}
```

Code 5.7: Code CUDA permettant la multiplication d'un vecteur par un scalaire.

5.6.3 OpenCL

Le langage OpenCL[36] est une interface de programmation en open source qui a pour objectif l'implémentation de solution parallèle dans un environnement hétérogène architecturale. Le

[36] https://www.khronos.org/opencl/

but étant de fournir aux ingénieurs une structure de code portable et efficiente qui est capable d'exploiter la puissance développée par les CPUs multicœurs, les GPUs et autres cartes accélératrices (FPGA[37]. MIC, ASICs[38] etc...). Un des atouts extrêmement important de l'OpenCL est qu'il cible toutes les différentes architectures de carte graphique i.e. AMD, Intel etc... à l'encontre de CUDA.

5.6.3.1 Composition du Langage

La plateforme OpenCL fournit un modèle qui sépare conceptuellement l'ordinateur (ou le serveur) qui est considéré comme l'hôte et les différentes cartes de calculs. En général, les nœuds de calculs sont confiés aux cartes puisqu'elle dispose d'une puissance de calcul nettement supérieure au serveur. L'hôte garde le contrôle sur l'exécution d'une application plus générale et distribue les tâches plus petites aux cartes de calculs. Ces cartes sont composées d'une ou plusieurs unités de calcul qui sont eux-mêmes divisées en plusieurs plus petits composants de calcul. Par exemple, les cartes graphiques de Nvidia sont composés de plusieurs multiprocesseurs qui sont composés de plusieurs centaines d'unité de calcul scalaire.

Ces unités sont soient scalaire (instruction SISD) soient vectoriel (instruction SIMD), c'est-à-dire respectivement capable de procéder à une instruction d'une donnée et d'une instruction pour plusieurs données à la fois.

5.6.3.2 Application, Groupe de Travail et Entité de Travail

Un programme en OpenCL (guide de programmation dans Munshi et al. [135]) est exécuté sur l'hôte de la même façon qu'un autre programme, à la différence est que cette première peut envoyer des ordres aux cartes accélé- ratrices en utilisant l'interface de programmation des différentes machines. Ces ordres sont appelés kernel ou application qui sont soient une ou un ensemble de fonctions d'instructions à exécuter. Ces petites applications sont écrites en C ou C++ et avec quelques extensions dûes à la librairie d'OpenCL. Il est possible de créer un fil d'exécution des applications sur les cartes de calculs.

Il y a deux niveaux de granularités des applications exécutées sur les cartes, les work-item ou les entités de travail qui définissent un point en espace. Ce point en espace peut être représenté en une, deux ou trois dimensions en fonction de la structure de l'algorithme et du confort voulu par le développeur sur la maintenabilité du code produit. Ces entités de travail sont regroupés en work-group ou groupe de travail. Les entités de travail au sein d'un même groupe ont une indexation locale. L'indexation globale du groupe de travail permet de retrouver l'indexation globale d'une entité de travail dans un groupe.

5.6.3.3 Hiérarchie des Mémoires

OpenCL dispose d'un modèle à plusieurs niveaux pour la mémoire. La hiérarchie des mémoires est plus détaillée dans Tay [177]. Chaque entité de travail a accès à 4 types de mémoires qui sont les suivantes:

- Mémoire globale, une région de mémoire où tous les entités et les groupes de travail peuvent aller écrire et lire des informations. Généralement, c'est elle qui comporte la plus

[37]Un dispositif FPGA est un circuit logique programmable, son circuit intégré logique peut être reconfiguré après sa fabrication (modification des connexions, du comportement du composant en connectant ou non les portes logiques entre elles)

[38]Une carte ASIC est un circuit propre intégré à une application ou appelé circuit intégré spécialisé qui peut regrouper un certain nombre de fonctionnalités uniques ou spécifiques.

grande taille de mémoire. Mais, le temps d'accès est très lent en comparaison des 3 autres mémoires.

- Mémoire locale, cette région de mémoire n'est visible que par les entités de travail appartenant au même groupe. Ils peuvent tous écrire et lire dans cette partie. Elle est beaucoup plus petite que la mémoire globale mais le temps d'accès est plus rapide.
- Mémoire constante, une région qui est inchangée pendant toute l'exécution de l'application. Seulement l'hôte peut modifier cette zone de mémoire.
- Mémoire privée, une très petite zone de mémoire qui n'est accessible (lecture et écriture) seulement par une entité de travail. Cette mémoire dispose d'un accès très rapide.

L'hôte alloue, gère et peut lire ou écrire en mémoire globale ainsi que dans la mémoire constante tout au long de l'exécution d'un programme OpenCL. La mémoire locale et privée sont seulement accessibles par le dispositif de calcul pendant l'exécution d'un noyau d'application. Les mémoires des deux machines sont séparées physiquement ou reliées virtuellement mais seul l'hôte peut communiquer. Il est important de noter que cette mémoire hiérarchique n'est pas physique mais virtuelle, ce qui permet au langage OpenCL d'adresser n'importe quelle architecture.

5.6.3.4 Exemple de Code

```
int main() {
        const size_t N = 1 << 20;

        std::vector<cl::Platform> platform;
        cl::Platform::get(&platform);

        cl::Context t;
        std::vector<cl::Device> device;

        device = getDevices(CL_DEVICE_TYPE_GPU);
        t = cl::Context(device);

        cl::CommandQueue queue(t, device[0]);

        cl::Program program(t, cl::Program::Sources(
                    1, std::make_pair(source, strlen(source))));

        program.build(device);

        cl::Kernel add(program, "add");

        std::vector<double> a(N, 1), b(N, 2), c(N);
        int s = a.size() * sizeof(double);

        cl::Buffer A(t, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                s, a.data());
        cl::Buffer B(t, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                s, b.data());
        cl::Buffer C(t, CL_MEM_READ_WRITE, s);
```

```
        add.setArg(0, static_cast<cl_ulong>(N));
        add.setArg(1, A);
        add.setArg(2, B);
        add.setArg(3, C);

        queue.enqueueNDRangeKernel(add, cl::NullRange, N,
                cl::NullRange);
        queue.enqueueReadBuffer(C, CL_TRUE, 0, s, c.data());

        return 0;
}
```

Code 5.8: Code OpenCL permettant l'addition de deux vecteurs.

On peut voir la nette différence de complexité entre les programmations du code CUDA 5.7 et du code OpenCL 5.8. Ce dernier demande un effort plus conséquent.

5.6.4 Pro/Con

La difficulté d'avoir un code qui cible plusieurs architectures est qu'on ne peut généraliser les optimisations faîtes d'une machine à une autre. Par exemple, l'algorithme de réduction de Harris [86] pour la somme des éléments d'un vecteur n'a pas la même efficacité sur un CPU et sur une GPU. L'implémentation d'une solution générique demande de prendre plus de recul dans l'abstraction, ce qui induit de ne pas pouvoir profiter à 100% des caractéristiques de chaque machine. Un autre exemple qui montre la complexité de cette tâche est que le CPU ne dispose pas d'une structure hiérarchique de mémoire i.e. l'utilisation de la mémoire partagée. Un CPU ne dispose que de 3 niveaux de cache, de registre et d'une mémoire globale, cette disposition change la façon d'imaginer un algorithme et les optimisations faîtes à partir de la représentation des données en mémoire.

L'alignement des accès en mémoire par rapport à l'indexation des processus logiques est une spécificité qu'on ne trouve uniquement sur une carte graphique à l'instar de la notion de grappe d'entité de travail et de la divergence des processus logiques dans l'exécution de leur fil d'instruction sur une unité de calcul. La vectorisation n'est pas possible sur une carte graphique, les unités de calcul sont seulement scalaires. La finesse de granularité sur le CPU est beaucoup plus grossière que sur une carte graphique et il n'y a pas de contrôle sur la gestion de l'affinité des processus logiques.

Quoiqu'il en soit, il existe certains aspects chez les différentes architectures qui sont similaires et les optimisations de code auront le même effet. Le passage de tableaux de structures à des structures de tableaux est une optimisation de code qui fonctionne sur les trois machines. L'alignement des données en mémoire, le déroulement des boucles, l'utilisation limite des registres pour éviter la lecture et l'écriture en mémoire globale pour diminuer la latence d'accès sont toutes des optimisations qui prouvent leurs efficacités. Tous ces aspects d'optimisations sont retrouvables dans livre de Scarpino [172].

Toutes ces points font que l'écriture du code pour viser les trois architectures que nous avons voulu a rendu la tâche très ardue. Le choix de l'élaboration d'une telle solution se fait dans un contexte industriel visant à proposer une libraire de calcul maintenable, à moindre coût et pouvant fonctionner sur plusieurs architectures en même temps ou séparément. Le marché des cartes de calculs n'étant pas stable, il est dangereux pour une industrie de prendre le pari

de se lancer dans telle ou telle technologie si celle-ci n'est pas pérenne [39]. L'idée principale de cette implémentation est d'avant tout de permettre aux industriels de se couvrir face aux fluctuations du marché technologique et de ne pas être dépendant d'un support physique de calcul en particulier.

5.7 Expérimentation et Résultat Numérique

5.7.1 Architecture Homogène

Le cas test est similaire à celui réalisé dans la section 5.5 sauf que la génération des contrats financiers a été nettement simplifiée, par exemple les SWAPs ont beaucoup moins de dates de paiements à évaluer. Ce portefeuille reflète moins la réalité que ceux précédemment testés. Nous avons voulu nous focalisé sur la faisabilité de l'élaboration d'une telle solution. Les codes ne ciblent qu'une architecture à la fois. On teste l'efficacité d'un code CUDA avec une implémentation optimisée pour une carte graphique spécifique. C'est une carte graphique Nvidia Quadro 1000M avec 2 multiprocesseurs cadencés à 1,40 GHz, chaque multiprocesseur est équipé de 48 cœurs. La carte est dotée d'une architecture de la famille Fermi et a une capability 2.0. le serveur et la carte accélératrice d'Intel ont la même configuration donnée dans la section 5.5, on a également testé une version sur un Intel i7 à 4 cœurs physiques (et 4 cœurs virtuels) cadencés à 2,40 GHz pouvant gérer 8 processus logiques.

Langage	Intel Core i7	Intel Xeon	Intel Xeon Phi Co.	Nvidia Quadro 1000M
C++	9.45	1.29	1.27	1.42
OpenCL	15.58	3.22	3.20	7.60

Table 5.9: Comparaison des performances réalisées (temps de calcul exprimé en seconde) des différentes implémentations en OpenCL et C++ sur 4 architectures.

Sur le tableau 5.9, les performances d'une implémentation en C++/CUDA pour la carte graphique et une implémentation OpenCL visant n'importe quelle architecture. Le cas test est seulement testé avec 5 000 contrats, le portefeuille est composé uniquement de CAPs/FLOORs. On peut voir que l'implémentation OpenCL ne produit pas les meilleurs temps de calcul, ce qui est cohérent avec la complexité de produire un code à architecture hétérogène. On ne peut pas bénéficier de tous les avantages de la machine sous-jacente. Sur la figure 5.14, on peut observer les performances obtenues pour différentes tailles de portefeuille (allant jusqu'à 100 000 contrats), le temps de copie de l'hôte au dispositif de calcul n'est pas pris en compte. Les portefeuilles sont composés à 50% de SWAPs et 50% de CAPs/FLOORs.

Par la suite, nous avons fait un effort d'implémentation sur le code OpenCL mais seulement pour viser un serveur ou une carte accélératrice d'Intel. L'architecture d'une carte graphique étant beaucoup trop différente pour l'élaboration d'une solution hétéro- gène satisfaisante. Cette fois-ci, nous avons gardé les temps obtenus au fur et à mesure des optimisations. Pour ce cas test, on prendra 10 000 simulations pour les risques facteurs. Dans le tableau 5.10, on peut noter les performances en fonction des optimisations faites sur le code.

Les optimisations sont comme suit:

[39] Comme dit précédemment, CUDA n'implémente une solution ciblant uniquement les cartes graphiques produites par Nvidia. Le compilateur n'est pas un projet Open Source donc est complètement dépendant de la stratégie de Nvidia et de sa santé économique.

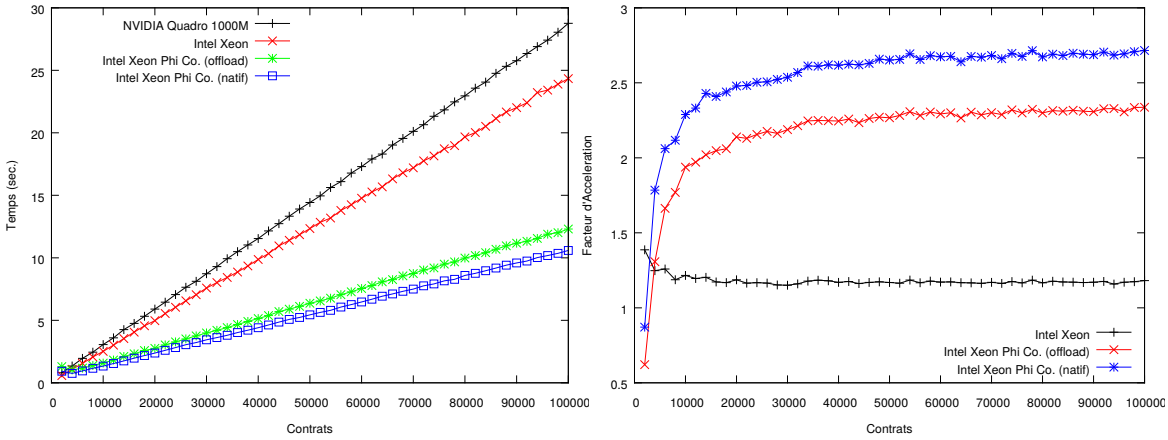

Figure 5.14: A gauche, le temps de calcul en seconde par rapport aux nombres de contrats traités pour chaque architecture avec une implémentation OpenCL. A droite, le facteur d'accélération par rapport aux nombres de contrats valorisés. La comparaison se fait en comparaison des performances obtenues sur la carte graphique de Nvidia.

–	Intel Xeon	Intel Xeon Phi Co.
Optimisation 1	7.78	21.62
Optimisation 2	4.09	3.53
Optimisation 3	3.84	3.30
Optimisation 4	2.91	1.57

Table 5.10: Comparaison des performances réalisées (temps de calcul exprimé en seconde) des différentes optimisations réalisées sur l'implémentation OpenCL.

1. Mise en place de directive pour tenter l'auto vectorisation du code par le compilateur OpenCL.

2. Ajout de fonction intrinsèques pour une vectorisation effective.

3. Utilisation de Vtune[40] pour trouver les nœuds critiques de l'application et ajout du parallélisme dynamique des processus logiques.

4. Réarrangement des données afin de linéariser l'accès en mémoire et quelques optimisations de fonctions mathématiques.

On peut voir que les optimisations sur le code n'ont pas le même impact en fonction de l'architecture. Les optimisations ne sont pas indépendantes des autres, l'optimisation 3 a été faite sur un code qui possède déjà l'optimisation 1 et 2. Il faut noter que la nature des compilateurs jouent un rôle important dans les performances de calcul des codes générés. Entre un compilateur Open Source GNU et celui d'Intel, nous avons observé des ratios de performance allant jusqu'à 8 en faveur de ce dernier.

[40] Application permettant de profiler le code et de faire une analyse de performance sur des machines 32 ou 64 bit x86. C'est un logiciel fournit par Intel.

5.8 Conclusion

Les travaux réalisés au sein de Global Market Solutions ont montré l'efficacité potentielle du Xeon Phi Coprocessor pour le calcul massif de produit vanilles en situation proche d'un cas "réel". En combinant des optimisations à des niveaux d'abstraction différents (informatique et mathématique), il nous a été possible de concevoir un moteur de calcul performant et qui bénéficie de toutes les spécificités de son architecture.

Une des principales caractéristiques du Xeon Phi Coprocessor qui pourrait favoriser son choix par rapport à une carte graphique est qu'il ne demande pas une refactorisation complète du code. Son utilisation est simple et l'implémentation d'un code assez performant ne requiert pas une grande connaissance de son architecture ainsi sa maintenance ne sera pas très dépendante de son développeur.

Ces travaux sont une sorte de "proof' of concept" pour l'élaboration d'un super calculateur de petit taille à moindre coût. La comparaison de la puissance de calcul, son prix et la consommation en électricité en fait un excellent choix pour la conception d'un calculateur puissant.

L'implémentation d'une solution visant tout type d'architecture a montré ses limites mais l'entité désireuse de ne pas être dépendante de l'architecture sous-jacente pourrait préferer cette option. La généricité impose des barrières qui ne peuvent pas êtres levées car un certain degré d'abstraction n'est pas atteignable dans le processus d'optimisation. En effet, les différences des spécificités de chacune des architectures n'est pas toujours réconciliable.

Des travaux futures pourront porter sur l'intégration de cette architecture dans le progiciel financier IRIS et généraliser l'approche pour le calcul des sensibilités, pour les autres indicateurs de risques ainsi que pour des produits exotiques. Une étude pourra également être poursuivie sur les nouvelles architectures et sur les nouveaux protocoles de programmation i.e. OpenACC qui permettent de viser plusieurs architectures à l'aide de simple directive pragma en C++.

Remerciements

C'est un projet collaboratif qui a fait intervenir plusieurs entreprises. Je tiens à remercier les partenaires de chez Intel, Raphaêl Monten qui nous a très gentiment appuyé tout au long de ce projet et nous a mis à disposition le matériel adéquate et les ressources humaines nécessaires. Mais aussi Laurent Duhem, Ayal Zaks, Dahnken Christopher, Hans Pabst et Wolfgang Rosenberg pour leurs aides dans l'implémentation des différentes solutions. La réalisation d'une solution avec 6 cartes Intel Xeon Phi Coprocesseurs a été permise grâce à la collaboration de 2CRSI et de son équipe de développement: Karim Gasmi, Frédéric Mossman, Adrien Badina ainsi que Claire Chupin. Je remercie Massimiliano Fattica de chez Nvidia, pour son aide dans l'implémentation de la solution en CUDA. Je remercie également Arnaud Renard pour nous avoir laisser travailler sur le super calculateur de la région Champagne-Ardenne situé à l'université de Reims. Je souhaite particulièrement remercier Jacques Portès pour toutes les discussions autour de l'architecture des cartes accélératrices et graphiques, ainsi qu'à Alain Dominguez pour ces passionnantes sessions d'implémentation et d'échanges.

Chapitre 6

Régularisation de Fonction Discontinue en Finance et Différentiation Automatique

---

"Les machines ne fonctionnent pas pour permettre aux hommes de vivre, mais on se résigne à nourrir les hommes afin qu'ils servent les machines"
extrait de Oppression et Liberté, 1934.

– S. A. Weil, Philosophe, (1909, 1943).

Dans ce chapitre, nous présentons les librairies de différentiation automatique qui ont été implémentées dans le cadre de la collaboration avec Global Market Solutions et qui ont servies aux expérimentations faîtes dans les chapitres 1, 3 et 2. Ces librairies ont également été utilisées dans le cadre de test numérique interne et de la faisabilité de l'intégration de telles libraires dans le progiciel IRIS au sein du département de recherche & développement de l'entreprise partenaire. Nous présentons succinctement les deux librairies implémentées i.e. une version C++ pour le calcul du premier et du second ordre de dérivation ainsi qu'une version CUDA destinée aux cartes graphiques pour le calcul du premier de dérivation. Nous faisons une rapide description de la régularisation utilisée. Quelques applications en finance sont présentées à titre illustratif sur des produits vanilles et exotiques dans plusieurs modèles stochastiques.

6.1 Introduction

L'utilisation du calcul des sensibilités par différentiation automatique est de plus en plus répandue dans la finance. Les récentes régulations i.e. FRTB, SIMM, SA-CCR ont imposé vision différente de l'activité de risque. L'approche est basée sur les sensibilités et on demande des infrastructures performantes en terme de précision et de temps de calcul.

Il y a une communauté de plus en plus grande autour de la différentiation automatique, nous suggérons le lecteur intéressé de s'intéresser au site internet de cette communauté et disponible à l'adresse suivante: `http://www.autodiff.org/`. La communauté ne s'intéresse pas uniquement

à des applications en finance. Cette technique est très présente depuis plusieurs années dans les autres industries et poursuit une forte évolution.

Quelques contributeurs importants de travaux sur la différentiation automatique avec des applications en finance Capriotti [39] et [40], Reghai [162], Giles et Glasserman [58], Homescu [96], Pironneau [154]...

Ce chapitre est dédié aux travaux réalisé au sein de Global Market Solutions pour l'étude de la faisabilité de l'intégration d'une librairie de différentiation automatique dans un progiciel de calcul de risque (IRIS) par rapport à différentes architectures.

Ce dernier petite chapitre est organisé de la façon suivante: la section 6.2, nous commençons par introduire très succinctement la librairie de différentiation automatique (1er ordre de dérivation) implémentée en CUDA pour viser une carte graphique. La section 6.3 est dédiée à des applications pour le calcul des sensibilités d'un Call Européen.

La section 6.4 est réservée à la description de la différentiation automatique (2ième ordre de dérivation) en C++, nous faisons également la description de la régularisation à l'aide de la fonction de Gauss et comment implémenté les payoffs financiers avec cette bibliothèque. En section 6.5, nous procédons à plusieurs applications numériques sur des produits exotiques i.e. options barrières, digitale et Américaine.

Nous concluons par un résumé rapide des travaux réalisés ainsi que par quelques perspectives.

6.2 Différentiation Automatique du Premier Ordre sur GPU

6.2.1 Structure CUDA Différentiation par Induction

Dans cette section, nous présentons la librairie implémentée en CUDA pour le calcul du premier ordre de dérivation. Cette librairie est inspirée de l'implémentation de Grundmann and Masmodi [84], c'est une implémentation de différentiation automatique par induction ou mode direct effectuée à l'aide de surcharge d'opérateur. Pour une explication détaillée du fonctionnement de la différentiation automatique par surcharge d'opérateur, nous renvoyons le lecteur au chapitre 1, section 1.3.1 ainsi qu'aux référence suivantes Greiwank [81], Griewank and Walther [82], Naumann [137] and Hascœt [87]. La structure du code CUDA repose sur la syntaxe du C++ puisque l'implémentation est faîte à partir d'une `class`. Sur le code 6.1, une partie de l'implémentation de la `class CudaD`.

```
class CudaD{
public:
        float val[2];

        __device__ CudaD ()
                { val[0] = 0; val[1] = 0; }

        __device__ CudaD ( const CudaD & a )
                 { val[0] = a.val[0]; val[1] = a.val[1]; }
                ...
};

__device__ friend CudaD operator*
        ( const CudaD&, const CudaD& );
```

```
__device__ friend CudaD operator/
        ( const CudaD&, const CudaD& );

__device__ inline CudaD log (const CudaD & );
```

Code 6.1: Partie de l'implémentation de la `class CudaD` en Cuda.

On présente également quelques implémentation de surcharge d'opérateur sur le code 6.2.

```
__device__ CudaD operator*(const CudaD &x, const CudaD &y)
{
        CudaD r;
        r[0] = x[0] * y[0];
        r[1] = x[0] * y[1] + x[1] * y[0];
        return r;
}

__device__ CudaD operator/ (const CudaD &x, const CudaD &y)
{
        CudaD r;
        r[0] = x[0] / y[0];
        r[1] = ( x[1] - x[0] * y[1] / y[0] ) / y[0];
        return r;
}

__device__ inline CudaD log (const CudaD & x)
{
        CudaD r;
        r[0] = log(x[0]);
        r[1] = x[1]/x[0];
        return r;
}
```

Code 6.2: Exemple de surcharge d'opérateur pour la `class CudaD`.

L'ensemble de cette librairie est en accès libre à l'adresse suivante `http://www.lpma-paris.fr/pageperso/sall/Documents/1st_derivative.tar.gz`.

6.3 Applications à un Produit Dérivé Européen

6.3.1 Call Européen

Dans ce cas test, nous procédons à l'évaluation du Delta et du Vega d'un Call Européen de payoff:

$$\varphi(X_T) = (X_T - K)^+, \tag{6.1}$$

où X_T représente le prix du sous-jacent à la maturité T et K le prix d'exercice du contrat. Le prix du Call est donné par:

$$\text{Call}_{t=0}^{\text{euro}} = e^{-rT}\mathbb{E}\left[(X_T - K)^+\right]. \tag{6.2}$$

Et r est le taux d'actualisation sans risque. On choisit le modèle Black Scholes pour la dynamique du sous-jacent X. On peut en déduire une solution fermée pour le prix du Call Européen et qui est donnée comme:

$$\text{Call}_{t=0}^{\text{BS}} = X_0\Phi(d_+) - Ke^{-rT}\Phi(d_-), \tag{6.3}$$

avec

$$d_+ = \frac{\ln(X_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}, \quad d_- = d_+ - \sigma\sqrt{T}, \tag{6.4}$$

avec σ la volatilité et Φ la fonction de répartition de la loi normale. On dérivera l'équation 6.3 pour obtenir la formule fermée du Gamma.

Les paramètres du Call Européen sont $K = 95$ avec une maturité de $T = 1$. Le prix initial du sous-jacent à la date $t = 0$ est la volatilité est donnée à $\sigma = 20\%$ et le taux sans risque est $r = 5\%$. Dans le tableau 6.1, on calcule l'erreur L_2 entre la vraie solution donnée du Delta par la formule fermée et l'estimation obtenue par m{ethode de Monte Carlo. L'erreur L_2 est calculée à partir de 256 réplications de Monte Carlo. Les paramètres de Monte Carlo sont 1 pas de temps et 100 000 trajectoires. On fait varier le prix initial du sous-jacent sur une plage de valeur $X_0 \in [80, 100]$ pour tester les différents régimes i.e dans la monnaie, à la monnaie et en dehors de la monnaie.

X_0	80	90	100	110	120
Erreur L_2	1.235e−03	1.316e−03	1.239e−03	9.999e−04	8.023e−04
Ref. Sol.	3.052e−01	5.317e−01	7.279e−01	8.606e−01	9.355e−01

Table 6.1: Erreur L_2 pour l'approximation du Delta d'un Call Européen entre la solution fermée et la solution approchée, Le Delta est calculé à partir de la librairie AD en CUDA.

Dans le tableau 6.2, on calcule l'erreur L_2 entre la solution fermée du Vega et la solution approchée par méthode de Monte Carlo. On fait varier la volatilité sur une plage de valeur $\sigma \in [0.05, 0.65]$ pour tester les différents régimes de marché et tester la robustesse de la méthode. La valeur de X_0 est fixée à 100.

σ	0.05	0.20	0.35	0.50	0.65
Erreur L_2	1.694e−03	1.619e−03	1.920e−03	2.329e−03	2.842e−03
Ref. Sol.	4.871e−02	3.319e−01	3.581e−01	3.601e−01	3.553e−01

Table 6.2: Erreur L_2 pour l'approximation du Vega d'un Call Européen entre la solution fermée et la solution approchée, Le Vega est calculée à partir de la librairie AD en CUDA.

6.4 Différentiation Automatique du Second Ordre sur CPU

6.4.1 Structure C++ de Différentiation par Induction

Ici, nous présentons la librairie implémentée en C++ pour le calcul du second ordre de dérivation d'une fonction. Cette librairie reprend la logique de l'implémentation de Grundmann and Masmodi [84] et est une extension de celle-ci pour le calcul du second ordre. La surcharge d'opérateur produit trois résultats celui de la fonction, de sa dérivée ainsi que sa dérivée seconde. Sur le code 6.3, une partie de la `class ddouble` pour le calcul de la dérivée seconde.

```
class ddouble
{
 public:  double val[3];

        ddouble()
                { val[0] = 0; val[1] = 0; val[2] = 0;}

        ddouble(const ddouble& a)
                {
                        val[0] = a.val[0];
                        val[1] = a.val[1];
                        val[2] = a.val[2];
                }

                ...
};

friend ddouble operator *
        (const ddouble&, const ddouble&);

friend ddouble operator /
        (const ddouble&, const ddouble& );

ddouble H (ddouble & );
```

Code 6.3: Partie de l'implémentation de la `class ddouble` en C++ pour la dérivée seconde.

La surcharge d'opérateur est présentée sur le code 6.4, on définit également l'implémentation de la fonction Heaviside pour le calcul de la dérivée seconde.

```
ddouble operator * (const ddouble& x, const ddouble& y)
{
        ddouble r;

        r[0]= x[0]*y[0];
        r[1]= x[0]*y[1]+x[1]*y[0];
        r[2]= x[2]*y[0]+2*x[1]*y[1]+x[0]*y[2];

        return r;
}

ddouble operator / (const ddouble& x, const ddouble& y)
{
        ddouble r;

        r[0] = x[0]/y[0];
        r[1] = (x[1]-x[0]*y[1]/y[0])/y[0];
        r[2] = (( x[2]*y[0]-x[0]*y[2] )*y[0]
                 -2.*(x[1]*y[0]-x[0]*y[1])*y[1])
```

```
                /(y[0]*y[0]*y[0]);

        return r;
}

ddouble H (ddouble & x)
{
        double a=1.;

        double r = x[0] > 0. ? 1. : 0.;
        double dr = 1./(a*sqrt(M_PI))*exp(-x[0]*x[0]/a/a);
        double ddr = dr*x[2] - 2.0*x[1]*x[1]*x[0]/a/a*dr;

        return ddouble( r, x[1]*dr, ddr );
}
```

Code 6.4: Surcharge d'opérateur pour la `class ddouble` en C++ pour la dérivée seconde et implémentation de la fonction Heaviside.

L'ensemble de cette librairie est en accès libre à l'adresse suivante `http://www.lpma-paris.fr/pageperso/sall/Documents/2nd_derivative.tar.gz`.

6.5 Applications aux Produits Exotiques

Dans cette partie, le paramètre de régularisation utilisée dans la section 1.3.4 est toujours fixé à 1. Nous prendrons d'autres valeurs pour l'option Américaine, elles sont énoncées dans la section 6.5.5. Le paramètre de la régularisation est toujours fixé à 1 sauf dans le cas Américain.

6.5.1 Power Call (modèle Black Scholes)

Ici, nous procédons à l'évaluation du Gamma d'une option Power Call de payoff:

$$\varphi(X_T) = \left((X_T - K)^+\right)^\alpha, \tag{6.5}$$

où α est une constante positive. Dans le modèle Black Scholes, il existe une formule fermée (voir Reed [159] et [160]) pour le prix d'un tel produit et est donnée par:

$$\mathrm{P}^{\mathrm{BS}}_{t=0} = \sum_{i=0}^{\alpha} (-1)^i \binom{\alpha}{i} X_0^{\alpha-i} K^i \exp\left\{\left[(\alpha - i - 1) + \frac{\sigma^2(\alpha-i)(\alpha-i-1)}{2}\right] T\right\} \Phi(\xi_i) \tag{6.6}$$

avec

$$\xi_i = \frac{\ln(X_0/K) + (r - \sigma^2/2)}{\sigma\sqrt{T}} + (\alpha - i)\sigma\sqrt{T}, \quad i = 0, \ldots, \alpha \tag{6.7}$$

L'équation 6.6 est obtenue sachant que pour un stock lognormal, on a:

$$\mathbb{E}\left[X_T^{\alpha-i} \mathbb{1}_{X_T > K}\right] = X_0^{\alpha-i} \exp\left\{\left[(\alpha - i - 1) + \frac{\sigma^2(\alpha-i)(\alpha-i-1)}{2}\right] T\right\} \Phi(\xi_i), \tag{6.8}$$

et que $(x-y)^\alpha = \sum_{i=0}^{\alpha} \binom{\alpha}{i} x^{\alpha-i} y^i$. On obtiendra une formule fermée pour le Gamma en dérivant deux fois l'équation 6.6. Le prix de l'option Power Call est donné par

$$\mathrm{P}_{t=0}^{\mathrm{C}} = e^{-rT}\mathbb{E}\left[\left((X_T - K)^+\right)^\alpha\right]. \tag{6.9}$$

Pour la différentiation au second ordre de la fonction 6.5, on utilise la même régularisation que dans le section 1.3.4. On réécrit la fonction comme:

$$\varphi(X_T) = \left(H(X_T - K)(X_T - K)\right)^\alpha, \tag{6.10}$$

Avec H la fonction Heaviside et surchargée dans l'implémentation pour la dérivation au second ordre.

Les paramètres du Power Call sont la puissance du Call est $\alpha = 2$, le prix d'exercice est fixé à $K = 10$ et la maturité du contrat est de $T = 1$. On évalue le produit dans le modèle Black Scholes et ses paramètres sont la volatilité du sous-jacent est de $\sigma = 25\%$, le taux sans risque est $r = 5\%$. Comme réalisé précédemment, on fait varier le prix du sous-jacent sur une place de valeur $X_0 \in [8, 12]$ pour tester les différents régimes de marché. Sur le tableau 6.3, on calcule l'erreur L_2 entre la solution obtenue par approximation par méthode de Monte Carlo et la solution fermée dérivée de l'équation 6.6. L'erreur L_2 est calculée à partir de 256 réplications de Monte Carlo. Les paramètres de Monte Carlo sont 100 000 trajectoires de simulation et 25 pas de temps.

X_0	8	9	10	11	12
Erreur L_2	3.268e−04	3.536e−04	3.388e−04	3.040e−04	2.633e−04
Ref. Sol.	8.402e−01	1.255e+00	1.605e+00	1.858e+00	2.023e+00

Table 6.3: Erreur L_2 pour l'approximation du Gamma d'un Power Call entre la solution fermée et la solution approchée, Le Gamma est calculé à partir de la librairie AD en C++.

6.5.2 Option Chooser (modèle Black Scholes)

Dans cet exemple, nous avons choisi de valoriser une option Chooser (voir Rubinstein [166]), ce produit dérivé donne la possibilité au détenteur de l'option d'exercer à la maturité T_1 son droit de choisir entre un Call Européen et un Put Européen de maturité T_2. Les deux produits dérivés sous-jacent ont la même date d'expiration T_2 mais n'ont pas forcément le même prix d'exercice K_1 et K_2. À la date d'expiration T_1, le payoff d'une option Chooser est donné par:

$$\varphi(X_{T_1}) = \max\left(\mathbb{E}\left[(X_{T_2} - K_1)^+ \mid X_{T_1}\right], \mathbb{E}\left[(K_2 - X_{T_2})^+ \mid X_{T_1}\right]\right). \tag{6.11}$$

Ainsi le prix d'une option Chooser est donné par la formule suivante:

$$\mathrm{Chooser}_{t=0} = e^{-rT_2}\mathbb{E}\left\{\max\left(\mathbb{E}\left[(X_{T_2} - K_1)^+ \mid X_{T_1}\right], \mathbb{E}\left[(K_2 - X_{T_2})^+ \mid X_{T_1}\right]\right)\right\}. \tag{6.12}$$

La démonstration de la formule fermée est détaillée dans Clewlow et al. [44] pour le modèle Black Scholes et est donnée comme suit:

$$\begin{aligned}\mathrm{Chooser}_{t=0}^{\mathrm{BS}} \quad = \quad & X_0\Phi_2(d_+, D_{i,+} : \rho) - K_1 e^{-rT_2}\Phi_2(d_-, D_{i,-}; \rho) \\ & + K_2^{-rT_2}\Phi_2(-d_-, -D_{i,-}; \rho) - X_0\Phi_2(-d_+, D_{i,+}; \rho),\end{aligned} \tag{6.13}$$

avec

$$d_+ = d_- + \sigma\sqrt{T_1}, \quad D_{i,+} = D_{i,-} + \sigma\sqrt{T_2}, \quad i = 1, 2, \tag{6.14}$$

et

$$d_- = \frac{\ln\left(X_0/X_{T_1}\right) + (r - \sigma^2/2)T_1}{\sigma\sqrt{T_1}}, \quad D_{i,-} = \frac{\ln\left(X_0/K_i\right) + (r - \sigma^2/2)T_2}{\sigma\sqrt{T_2}} \tag{6.15}$$

où $\rho = \sqrt{T_1/T_2}$ et Φ_2 désigne la fonction de répartition de la loi normale bivariée. Pour obtenir la formule du Gamma, il suffira de dériver deux fois l'équation 6.13. En réutilisant la régularisation présentée dans le chapitre 1.3.4 pour le calcul de la dérivée seconde, on réécrira les fonctions des payoffs des Put et Call Européen de l'équation 6.11 de l'option chooser comme:

$$\mathrm{Call}_{X_{T_2}|X_{T_1}} = \mathbb{E}\left[H\left(X_{T_2} - K_1\right)\left(X_{T_2} - K_1\right) \mid X_{T_1}\right] \tag{6.16}$$

et

$$\mathrm{Put}_{X_{T_2}|X_{T_1}} = \mathbb{E}\left[H\left(K_2 - X_{T_2}\right)\left(K_2 - X_{T_2}\right) \mid X_{T_1}\right] \tag{6.17}$$

Donc la fonction de payoff de l'option Chooser de l'équation 6.11 devient:

$$\begin{aligned} \varphi(X_{T_1}) \quad = \quad & H\left(\mathrm{Call}_{X_{T_2}|X_{T_1}} - \mathrm{Put}_{X_{T_2}|X_{T_1}}\right)\mathrm{Call}_{X_{T_2}|X_{T_1}} \\ & + H\left(\mathrm{Put}_{X_{T_2}|X_{T_1}} - \mathrm{Call}_{X_{T_2}|X_{T_1}}\right)\mathrm{Put}_{X_{T_2}|X_{T_1}}. \end{aligned} \tag{6.18}$$

Les paramètres de l'option Chooser sont les prix d'exercice fixés à $K_1 = 60$ et $K_2 = 45$ avec les maturités suivantes $T_1 = 0.5$ et $T_2 = 1$. La dynamique du sous-jacent X est modélisée à l'aide du modèle Black Scholes avec une volatilité donnée à $\sigma = 25\%$ et un taux sans risque $r = 2\%$. Comme testé dans les sections précédentes, on calcule l'erreur L_2 entre la solution fermée du Gamma d'une option Chooser et la solution approchée par méthode de Monte Carlo. L'erreur L_2 est calculée à partir de 256 réplications de Monte Carlo. Les résultats sont affichés dans le tableau 6.4. Les modèlees de Monte Carlo sont 1000 trajectoires de simulation pour le calcul des espérances conditionnelles des deux produits en T_1 par trajectoire de simulation (1000) en $t = 0$. On a en tout 1 000 000 de trajectoires simulées et 2 pas de temps. Le prix du sous-jacent varie entre $X_0 \in [48, 56]$.

X_0	48	50	52	54	56
Erreur L_2	6.254e−04	5.862e−04	5.342e−04	6.489e−04	7.012e−04
Ref. Sol.	5.958e−02	5.772e−02	5.477e−02	5.080e−02	4.647e−02

Table 6.4: Erreur L_2 pour l'approximation du Gamma d'une option Chooser entre la solution fermée et la solution approchée, Le Gamma est calculé à partir de la librairie AD en C++.

6.5.3 Digital Call Asiatique (modèle Constant Elasticity of Variance)

Ici, nous avons choisi de tester notre bibliothèque de différentiation automatique sur une option Digital sur un Call Asiatique (voir Kemna et Vorst [111] et Levy [121]). Ce produit prend la moyenne de la performance réalisée par le sous-jacent entre la date du démarrage du contrat $t = 0$ et sa maturité T. Le contrat paye 1 si la performance moyennée est au-dessus du prix d'exercice fixé à l'avance. Le payoff d'un tel produit est donné par la formule suivante:

$$\varphi\left(\int_0^T X_s ds\right) = \mathbb{1}_{\frac{1}{T}\int_0^T X_s ds > K}, \tag{6.19}$$

Le prix est donc donnée par:

$$\text{Digital}_{t=0}^{\text{Asian}} = e^{-rT}\mathbb{E}\left[\mathbb{1}_{\frac{1}{T}\int_0^T X_s ds > K}\right]. \tag{6.20}$$

Suivant la même logique dans la section 6.5.1 pour le calcul de la dérivée d'ordre deux, la fonction de payoff 6.19 est réécrite et implémentée comme :

$$\varphi\left(\int_0^T X_s ds\right) = H\left(\frac{1}{T}\int_0^T X_s ds - K\right). \tag{6.21}$$

La modélisation du prix du sous-jacent X est faîte par un modèle CEV comme défini dans le chapitre 3.6.2.2. Comme ce produit n'admet pas de solution fermée dans ce modèle, nous calculons l'erreur L_2 entre la solution obtenue par méthode Monte Carlo plus différentiation automatique et la solution obtenue par méthodes des différences finies. Les paramètres du modèle CEV sont $\gamma = 0.75$, la volatilité du sous-jacent est donnée à $\sigma = 15\%$ et le taux sans risque à $r = 3\%$. Les paramètres du Digital sur Call Asiatique sont fixés à $K = 100$ pour le prix d'exercice et $T = 1$ pour la maturité. L'erreur L_2 pour l'approximation du Gamma est estimée à partir de 256 réplications de Monte Carlo et est calculée entre la solution obtenue par méthode Monte Carlo plus AD et celle obtenue par méthode des différences finies (1% shift). Les paramètres de Monte Carlo sont 50 pas de temps et 100 000 trajectoires de simulations. Les résultats sont affichés dans le tableau 6.5. On fait varier le prix du sous-jacent sur une plage de valeur de $X_0 \in [80, 120]$ pour tester différents régimes de marché.

X_0	90	95	100	105	110
Erreur L_2	4.736e−05	1.451e−04	2.155e−04	5.400e−05	1.026e−06
Ref. Sol.	1.701e−03	2.923e−02	−2.430e−02	−6.921e−03	−5.543e−05

Table 6.5: Erreur L_2 pour l'approximation du Gamma d'une option Digital sur Call Asiatique entre les solutions approchées par méthode de Monte Carlo et différences finies, Le Gamma est calculé à partir de la librairie AD en C++.

6.5.4 Up & Out Call (modèle Heston)

Dans ce cas test, nous procédons à l'évaluation du Gamma d'une option à barrière Américaine Up & Out Call. Pour rappel, dans Reiner et Rubinstein [164], le produit est défini comme: si le sous-jacent dépasse la barrière B entre $t = 0$ et la maturité du produit T, le Call Européen n'est pas exerçable à maturité autrement dit on sort du produit. La fonction de payoff d'une option Up & Out Call est donnée comme:

$$\varphi(X_T) = \mathbb{1}_{\max\limits_{0\leq t\leq T} X_t < B}(X_T - K)^+, \tag{6.22}$$

où B est la barrière du produit à ne pas dépasser. Donc le prix d'une option Up & Out Call est exprimê selon la formule suivante:

$$\text{U/O}_{t=0}^{\text{C}} = e^{-rT}\mathbb{E}\left[\mathbb{1}_{\max\limits_{0\leq t\leq T} X_t < B}(X_T - K)^+\right], \tag{6.23}$$

En utilisant le même raisonnement que dans la section 6.5.1 pour le calcul de la dérivée d'ordre deux, la fonction de payoff 6.22 est réécrite et implémentée comme :

$$\varphi(X_T) = H(B - \max_{0\leq t\leq T} X_t)H(X_T - K)(X_T - K),. \tag{6.24}$$

Dans le modèle d'Heston, il n'existe pas de solution fermée pour un tel produit donc l'erreur L_2 du Gamma sera calculée entre l'approximation obtenue par méthode de Monte Carlo et la méthode des différences finies. Les paramètres du modèle Heston défini dans le chapitre 1.7 sont $\kappa = 2$ pour la vitesse de retour à la moyenne, $\eta = 0.1$ pour la moyenne à long terme de la volatilité et $\xi = 20\%$ pour la volatilité de la volatilité avec une volatilité initiale à $\nu = 10\%$. La corrélation entre les deux mouvement Browniens est fixé $\rho = -50\%$. Les paramètres de la dynamique du prix du sous-jacent est $r = 1.3\%$ pour le taux sans-risque. Les données de l'option Up & Out Call sont $B = 120$ pour le niveau de la barrière, $K = 100$ pour le prix d'exercice avec une maturité à $T = 1$. L'erreur L_2 pour l'approximation du Gamma est estimée à partir de 256 réplications de Monte Carlo et la méthode des différences finies utilise un shift de 1% de la valeur du prix sous-jacent initial. Les paramètres de Monte Carlo sont 100 pas de temps et 100 000 trajectoires de simulations. Les résultats sont inscrits dans le tableau 6.6. On fait varier le prix du sous-jacent sur une plage de valeur de $X_0 \in [95, 115]$.

X_0	95	100	105	110	115
Erreur L_2	2.560e−05	1.733e−05	5.372e−06	4.413e−06	1.899e−06
Ref. Sol.	−1.500e−03	−1.039e−03	−7.192e−04	−2.185e−04	−1.731e−04

Table 6.6: Erreur L_2 pour l'approximation du Gamma d'une option Up & Out Call entre les solution approchées par méthode de Monte Carlo et différences finies, Le Gamma est calculé à partir de la librairie AD en C++.

6.5.5 American Put (modèle Black Scholes)

Pour le dernier cas test, nous avons choisi de tester notre librairie de différentiation automatique sur un problème plus complexe i.e. l'option Américaine et sa valorisation faîte par l'algorithme de Longstaff Schwartz [124]. Ce produit procure le droit à son possesseur d'exercer un Put à n'importe quel moment entre la date de signature du contrat et son échéance $T \in \mathbb{R}^+$. La stratégie naturelle sera donc d'essayer de maximiser le gain de ce produit, son prix est donné par

$$\text{Put}_t^{\text{amer}} = \operatorname*{ess\,sup}_{\tau\in\mathcal{T}_{t,T}} \mathbb{E}[e^{-r(\tau-t)}(K - X_\tau)^+ \mid X_t]. \tag{6.25}$$

Où $\tau \in \mathcal{T}_{t,T} = \{\text{temps d'arrêt à valeurs dans } [t, T]\}$, r est taux d'actualisation sans risque (taux d'intérêt). X_τ désigne le prix du sous-jacents à l'instant τ (temps optimal d'exercice) et K le strike du contrat. Comme vu dans le chapitre 1.6, soit la partition de temps $0 = t_0 < \ldots < t_n = T$ avec $n \geq 1$ le problème d'arrêt optimal 6.25 est équivalent au problème de programmation dynamique suivant:

$$\begin{cases} V_T = (K - X_T)^+ \\ V_{t_k} = \max\left((K - X_{t_k})^+, e^{-r(t_{k+1}-t_k)}\mathbb{E}[V_{t_{k+1}} \mid X_{t_k}]\right), \quad k = n-1, \ldots, 0. \end{cases} \tag{6.26}$$

On approchera l'espérance conditionnelle comme expliqué dans la section 1.6, ainsi on pourra réecrire la fonction de prix 6.26 pour quelle puisse être différenciée deux fois comme

$$\begin{cases} V_T = H(K - X_T)(K - X_T) \\ V_{t_k} = H(P_{t_k} - Q_{t_k})P_{t_k} + H(Q_{t_k} - P_{t_k})Q_{t_k}, \quad k = n-1, \dots, 0. \end{cases} \tag{6.27}$$

avec

$$\begin{aligned} P_{t_k} &= H(K - X_{t_k})(K - X_{t_k}), \\ Q_{t_k} &= e^{-r(t_{k+1} - t_k)} \sum_{i=1}^{R} \alpha_{k,i} \varphi_{k,i}(X_{t_k}). \end{aligned} \tag{6.28}$$

Il n'existe pas de solution fermée pour une option Américaine mais seulement des méthodes d'approximation. Nous calculerons donc l'erreur L_2 du Gamma entre la méthode des différences finies (shift de 1% du prix initial du sous-jacent) sur l'algorithme de Longstaff Schwartz et la solution obtenue par différentiation automatique. Les paramètres du modèle Black Scholes sont le taux d'intérêt est fixé à $r = 6\%$, $\sigma = 20\%$ pour la volatilité. Les paramètres du Put Américain sont la maturité $T = 1$, le strike du contrat est $K = 40$, on prendra 50 dates d'exercice (nombre de pas de temps pour les paramètres de Monte Carlo). On calcule l'erreur L_2 à partir de 256 réplications de Monte Carlo avec 100 000 trajectoires de simulations. On teste différents régimes de marché avec des prix de sous-jacent $X_0 \in [36, 44]$. Les résultats sont donnés dans le tableau 6.7. Le paramètre de régularisation varie entre 0.025 et 0.05.

X_0	36	38	40	42	44
Erreur L_2	3.019e−04	2.373e−04	5.914e−04	1.014e−04	3.784e−04
Ref. Sol.	8.745e−02	7.362e−02	6.027e−02	4.776e−02	3.663e−02

Table 6.7: Erreur L_2 pour l'approximation du Gamma d'un Put Américain entre les solution approchées par méthode de Monte Carlo et différences finies, Le Gamma est calculé à partir de la librairie AD en C++.

6.6 Conclusion

Ces travaux proposent une solution simple et efficace pour le calcul des sensibilités des produits vanilles et exotiques sur différentes architectures. Les résultats numériques ont prouvés l'efficacité de cette solution. Son implémentation est facile puisqu'elle ne nécessite qu'une simple reformulation du payoff et le changement de mot clé `double` en `ddouble` pour un CPU ou `CudaD` pour l'implémentation en GPU.

Ces travaux ne constituent en soit qu'une "proof of concept", il sera nécessaire de faire un effort de réflexion pour le cas en plusieurs dimensions et de changer la librairie. Celles proposées ne sont valables que pour la différentiation pour une variable, pour obtenir une autre dérivée il faudra exécuter une nouvelle fois l'application.

Il est important de noter qu'il a été nécessaire d'ajuster le paramètre dans la régularisation pour obtenir un Gamma correct dans le cas Américain. Nous en avons déduit qu'il dépendait probablement des paramètres et du type de l'option. Étant donné que la pente du Delta est fonction des paramètres du Call, il est clair que l'ajustement de la régularisation ne peut pas être fixe. Malheureusement, nous n'avons pas eu le temps d'explorer cette corrélation.

Un travail future pourra porter sur cette étude, ce qui rendrait cette méthode extrêmement efficace et complètement automatique ainsi que sur le cas en plusieurs dimensions et pour des produits plus complexes.

Appendix A

Some Definitions

CVA and Risk Measures

In the risk management world, there was a unique single measure that has proven its usefulness which is the value at risk or also know as VAR. Nevertheless, the identification and a proper evaluation of the future exposure for the counterparty credit risk will necessitate many other risk measures and metrics. Therefore, the complexity of those metrics has dramatically increased because credit exposure requires to be determined over a set of time horizons to have a deep understanding of the consequence of the time and customized of counterparty securities (unlike the value at risk which is traditionally a single horizon risk metric). Counterparty credit risk is observed at from different perspective of view (risk management and pricing) which necessitate different risk measures. And because it is a key concern to comprehend the effective exposure with respect to each different counterparty. There is a wide literature covering the counterparty credit risk and the different risk measures, the lecturer is referred to the following papers Brigo and Masetti [30], Basurto and Singh [13], Zhu and Pykhin [185], Jarrow and Yu [101] and Gregory [79].

Counterparty Credit Risk

The Credit Value Adjustment

Here, we want to formulate the risky value and find an expression $\tilde{V}(t,T)$ of an aggregated set of financial derivatives with the longest maturity date T. We define the time of default of the counterparty as τ and the riskless value of the previous portfolio as $V(t,T)$. Therefore, we can consider two different cases:

- Before the longest maturity date T, the counterparty dœs default then the payoff is made up of two terms
  - the value of the position before the time of default of the counterparty (before default, all payments will still be fulfilled)
  - the payoff at the time of default

The payoff paid up to the time of default are expressed as:

$$\mathbb{1}_{\tau \leq T} V(t, \tau), \tag{A.1}$$

where $\mathbb{1}_{\tau \leq T}$ denotes the indicator function indicating if whether or not the default has occurred prior to the longest maturity inside the portfolio of financial derivatives (it takes the value 1 in case of default).

In the case of a non negative value of the positions at default time, the institution will get a recovery (only a part or a fraction δ) of the financial securities positions. Whereas, in the case of a negative value the counterparties have to pay off this amount. Therefore, the payoff at the default time is given by

$$\mathbb{1}_{\tau \leq T} \left(\delta V(\tau, T)^+ + V(\tau, T)^- \right). \tag{A.2}$$

By adding up the previous payoffs jointly and under the risk neutral probability measure, we finally obtain the following expression

$$\tilde{V}(t, T) = \mathbb{E}\left[\mathbb{1}_{\tau > T} V(t, T) + \mathbb{1}_{\tau \leq T} V(t, \tau) + \mathbb{1}_{\tau \leq T} \left(\delta V(\tau, T)^+ + V(\tau, T)^- \right) \right], \tag{A.3}$$

using the decomposition of $x = x^+ - x^-$ and by re-arranging terms, we have:

$$\tilde{V}(t, T) = \mathbb{E}\left[\mathbb{1}_{\tau > T} V(t, T) + \mathbb{1}_{\tau \leq T} V(t, \tau) + \mathbb{1}_{\tau \leq T} \left((1 - \delta) V(\tau, T)^+ + V(\tau, T) \right) \right], \tag{A.4}$$

and since $V(t, \tau) + V(\tau, T) = V(t, T)$ and $\mathbb{1}_{\tau > T} V(t, T) + \mathbb{1}_{T \leq \tau} V(t, T) = V(t, T)$ we, therefore, have

$$\tilde{V}(t, T) = V(t, T) + \mathbb{E}\left[(\bar{\delta} - 1) \mathbb{1}_{\tau \leq T} V(\tau, T)^+ \right]. \tag{A.5}$$

This important term is also known as the so-called credit value adjustment or as the famous acronym (CVA). It is a correction to the riskless assessment of the positions inside the portfolios of derivative securities to take the counterparty credit risk into account within the netting set,

$$\mathrm{CVA}(t, T) = \mathbb{E}\left[(1 - \bar{\delta}) \mathbb{1}_{\tau \leq T} V(\tau, T)^+ \right]. \tag{A.6}$$

Considering there is no links between the exposure, the time of default and the recovery assessment. Then, we write the following equation:

$$\mathrm{CVA}(t, T) = -(1 - \delta) \mathbb{E}\left[\mathbb{1}_{\tau \leq T} V(\tau, T)^+ \right], \tag{A.7}$$

where $\bar{\delta}$ is the expected recovery assessment and since the expectation from the previous equation considers all times possibility before the longest maturity, we have the possibility to integrate over all default instants so we obtain

$$\mathrm{CVA}(t, T) = -(1 - \delta) \mathbb{E}\left[\int_t^T B(t, u) V(u, T)^+_{|\tau = u} dS(t, u) \right], \tag{A.8}$$

where $S(t, u)$ is the survival probability or the probability to not do default before the last maturity and $B(t, u)$ is the riskless discount factor. The probability to not do default is given by

$$S(t, u) = \mathbb{E}[\mathbb{1}_{\tau > t}], \tag{A.9}$$

considering that the survival and the discount terms are are deterministic, we have:

$$\mathrm{CVA}(t, T) = -(1 - \delta) \int_t^T B(t, u) dS(t, u) \mathbb{E}\left[V(u, T)^+_{|\tau = u} \right]. \tag{A.10}$$

Computing the previous equation with specific integration scheme using n periods defined by $(t_0 = t), t_1, ..., (t_m = T)$, we finally obtain the following equation for the credit value adjustment

$$\mathrm{CVA}(t,T) \approx -(1-\delta)\sum_{i=1}^{m} \mathbb{E}\left[V(t_i,T)^+\right] B(t,t_i)\left(S(t,t_{i-1}) - S(t,t_i)\right). \tag{A.11}$$

- Before the longest maturity date T, the counterparty dœs not default then the risk free position corresponds to the risky one. Therefore, the corresponding payoff can be written as:

$$\mathbb{1}_{\tau<T} V(t,T). \tag{A.12}$$

Metrics and Risk Measures

Expected Exposure (EE)

The formula of the credit value adjustment given in A.11 exposes a new whole term $\mathbb{E}^{\mathbb{Q}}[V(u,T)^+]$ also called the exposure of the derivative positions (the value of the position that could be lost in case of the default of the counterparty) in the markets. Commonly, the exposures initiated by derivatives from OTC market represents only a little fraction of the notional engages in the securities on the financial markets.

A fundamental element of the counterparty credit risk comes from the fact that the potential loses are not the same for both of the parties, they are asymmetric with respect to the value $V(u,T)$ at the current time t. When a counterparty is defaulting, an institution or a financial corporation has to round off its positions in the market and is not forced to fulfill its obligations to make future payments settled in the contract (there is only a little chance that the payments will be received). Nevertheless, at the default time the underlying securities must be settled conditionally to the assessment of $V(u,T)$.

Potential Future Exposure (PFE)

The Potential future exposure or known as the acronym (PFE) can be defined as the top value of the expected exposure over a determined instant of time evaluated at a specific level of confidence. The notion of potential future exposure comes from the necessity to describe the assessment of the portfolio of contingent claims could be at specific instant in the future.

The value of the derivatives positions are known at the current time t and any time the past and is one of the characterization of the potential future exposure. Nevertheless, there is some randomness over the future exposure because the position could take any possible assessment (in some range). At a specific time in the future, we could try to describe the potential future exposure with a particular probability distribution.

Controlling the cost price of the risk of a counterparty over time is one of most basic use of exposure. It can be computed by according a credit limit to each different counterparty. The notion is to associate the potential future exposure to a counterparty for any given time and to guarantee that the counterparty dœs not outreached a credit limit. The credit limitation will be given arbitrarily in accordance to the risk picture of a specific institution in question. It has to be time varying because the exposures can be considered differently at different instant of time in the future.

The Mark to Market (MtM)

The mark to market or also known as the MtM can be described as the current potential lost with respect to a specific counterparty. Consequently, one can see the mark to market as the sum of every contracts with the same counterparty. Nevertheless, it is depending on the fact that the institution or financial corporation is able to aggregate the contracts at the instant the counterparty was to default. Moreover, there is other features that will lower the exposure at the time of default as the collateralization of the securities or the hedging of the positions etc...

The present mark to market dœs not form an instantaneous debt from a counterparty to the other party but is instead the current value of all cash flows an entity is expecting to get less these it is forced to fulfill. These cash flows could be scheduled to happen several years in the future and could potentially take values that are highly depending on the market parameters. Consequently, the mark to market could be either negative or positive according to whether a contract is in entity's favor or not.

The mark to market constitutes the substitution cost, which determines the starting point into a similar transaction with a different counterparty but only following the assumption that there is no transaction fees.

Hence, all these definitions enable us to reformulate the expression of the credit value adjustment as a function of the mark to market and is given by

$$\mathrm{CVA}(t,T) = -(1-\delta)\sum_{i=1}^{m}\mathbb{E}[\mathrm{MtM}_i(T)^+]B(t,t_i)\left(S(t,t_{i-1}) - S(t,t_i)\right). \tag{A.13}$$

Other Metrics:

- Expected negative exposure (ENE): the discounted cash flows and losses amount forecasted to be paid to the counterparty.
- Expected positive exposure (EPE): is the top level of the expected exposure that is realized at a specific date or in the past.
- Effective expected positive exposure (EEPE): is the weighted expectation of the effective positive exposure over time, the weights is depending on the proportion that each expected exposure constitutes over time.
- Wrong way risk: is the risk that comes from the probability of default of a counterparty being highly and positively correlated with other market risk factors.
- Right way risk: the risk that comes from a position's collateral or an exposure that is correlated (positively) with the credit performance of that counterparty.

xVAs

Following the 2008 crisis, the need of a proper assessment of the counterparty credit risk highlighted the relevance of the credit value adjustment. The regulators focused their attention on that measure and the risk management activity has been completely reinvented. Any kind of risk involved in a financial transaction proved their crucial importance and have to be monitored with utmost caution. Hence, the counterparty credit risk has to take several different adjustments into account as the debit value adjustment, the capital value adjustment, the funding value adjustment, the collateral value adjustment, the margin value adjustment, the hedging

value adjustment and the margin value adjustment (for a complete analysis of these value adjustments see Crépey [47], Green [78], Gregory [80] and Lu [125]. These adjustments are somewhat intertwined (even strong dependency among them) and necessitate to be approximated jointly. Their impacts outreach the risk management activity and spread out over other domains e.g. the front office and back office activities. The regulation forces to negotiate financial transactions via clearing houses or to provide stout guarantees via the collateralization of the transaction, causing liquidity problem and systemic risk. This section is dedicated to the definition of each value adjustment previously uttered.

Debit Value Adjustment (DVA)

The debt value adjustment can be seen as the opposite of the credit value adjustment. It represents the credit risk that a party can encounter when entering in a financial transaction against the other party. It describes the gap between the assessment of a financial security within the assumption that the financial entity whose emitting the security is default riskless and the default risk of the financial entity. If their credit risk has a modification, the debit value adjustment can also be modified. The debit value adjustment is highly correlated to the credit spread and the probability of default of the financial entity and to any changes of the value of the expected exposure. The credit value adjustment and the debit value adjustment has the opposite signs but the first one lowers the assessment of the financial instrument whereas the debit value adjustment increases it. One can formulate the debit value adjustment as:

$$\mathrm{DVA}(t,T) = -(1-\delta)\mathbb{E}\left[\int_t^T \mathbb{1}_{\tau<u} B(t,u) V(u,T)^+ du\right]. \tag{A.14}$$

Funding Value Adjustment (FVA)

The point of the incorporation of the funding value adjustment into bank risk systems is to take the funding cost associated to the generation of a derivative security into account i.e. the funding cost corresponds to the hedging until the expiry of the contract. Assume a financial institution (let's say a bank) and a non financial entity - a company - enter into a uncollateralized contract (as a swap of interest rate). The financial institution hedges the uncollateralized interest rate swap perfectly by stepping into a secured contract with a credit support annex as the underlying. As the company often demands the financial institution an uncollateralized contract but not otherwise. Indeed, the engagements of a credit support annex are frequently uncontrollable for companies. Consequently, we can not assume the bilateral funding value adjustment. To simplify, we assume that the credit support annex states that cash (in dollar) is the unique collateral accepted and we suppose that the collateral rate is the LIBOR rate overnight. Furthermore, we assume that the trading desk of the financial institution has to borrow money from its treasury desk to trade the collateral. Hence, the lending rate inside the bank is the cost of funding for the trading desk and incorporates inside costs as the cost of the system, labor and administration. Finally, we suppose that the treasury desk realizing bonds in the market to fund itself. Therefore, there is two different cases:

- positive exposure generates funding costs because the financial institution dœs not get collateral from the company and be forced to collateralized the margin account assigned with the hedged interest rate swap.
- negative exposure generates funding benefits because the financial institution dœs get the collateral.

Hence, for the evaluation of the funding value adjustment, we have to take into account the positive and negative exposure. Nevertheless, when the financial institution or the company makes default prior to the expiration of the contract, the funding costs and benefits will disappear. Therefore, the funding value adjustment can be defined as:

$$\mathrm{FVA}(t,T) = -(1-\delta)\mathbb{E}\left[\int_t^T \mathbb{1}_{\tau>u} B(t,u)\gamma(u)\left(V(u,T)^+_{|\tau=u} + V(u,T)^-_{|\tau=u}\right)du\right], \tag{A.15}$$

where the funding cost at time t is denoted by $\gamma(t)$.

Hedging Value Adjustment (HVA)

In practice and in the real world, a perfect hedge dœsn't exist. Therefore, a supplementary funding adjustment will be added which corresponds to the gap in cash between the position required to be taken to hedge the associated market risk and the transaction with the counterparty. An important point, this is independent of any collateral disposal. Consider the following example to enlight this correction, a bank to hedge a 5Y (year) interest rate swap with yearly coupon sold to a corporation with a 5Y interest swap with biannually coupons. Consider the following expression: $(x_t + h_t)dt$ which represents the amount of payment in the current hedging transaction and x_t denotes the "perfect" hedging transaction. Therefore, the bank have to fund the extra term h_t if the bank have to borrow it (in case of surplus, the bank can lend it). These two corrections or adjustments are called: hedging cost adjustment (HCA) and hedging benefit adjustment (HBA). Finally, the hedging value adjustment is just the sum of these two terms.

$$\mathrm{HVA}(t,T) = -(1-\delta)\mathbb{E}\left[\int_t^T \mathbb{1}_{\tau>u} B(t,u)\left(V(u,T)^+_{|\tau=u}\eta(u) + V(u,T)^-_{|\tau=u}\kappa(u)\right)du\right], \tag{A.16}$$

where $\kappa(u)$ is the borrowing benefit at time u and $\eta(u)$ the funding cost to hedge at time u.

Collateral value adjustment (CollVA)

A crucial credit risk mitigation technique is the collateralization of financial transactions, collateral can be constantly exchanged by the financial institutions. Posting net collateral everyday on a transaction requires to be funded (e.g. lent the net collateral received). Therefore, this is including the hedging transactions, the over the counter derivatives even transactions in clearing houses etc... Hence, we can define the collateral value adjustment as the sum of the collateral cost adjustment (collCA) and the collateral benefit adjustment (collBA). The collateral agreements are often divided into an initial margin and a variation margin. The following expression defines the collateral value adjustment:

$$\mathrm{CollVA}(t,T) = -(1-\delta)\mathbb{E}\left[\int_t^T \mathbb{1}_{\tau<u} B(u,t)\left(V^c(u,T)^+_{\tau=u}\theta(u) + V^c(u,T)^-_{\tau=u}\zeta(u)\right)du\right], \tag{A.17}$$

where $\zeta(u)$ defines the lending cost for receiving collateral at time u, $\theta(u)$ the borrowing cost for posting collateral at time u and V^c denotes the value collateralized portfolio of financial transactions.

Liquidity Value Adjustment (LVA)

The liquidity value adjustment can be viewed as a funding risk or as a cost associated to the liquidity premium. This amount expresses the fact that the credit default swap market has a different liquidity environment than the bond market. Fundamentally, two entities or corporations can enter in a credit insurance transaction by signing the credit default swap paper, but if either a party yearns for selling or buying an actual bond, it has to come from somewhere else and unfortunately there is only a limited quantity of them. As a matter of fact, the Black Scholes model assumption is not sustainable here. Therefore, the liquidity value adjustment is a required correction to the assessment of a portfolio of derivative securities under the risk neutral measure to take the genuine liquidity constraints into account, that can be encountered on the credit and funding market. It represents the gap between the cost of both parties to hedge the same risk.

The formal definition of the liquidity value adjustment is the discounted value of the difference between the risk free rate and the collateral paid or received on the collateral, and it is the gain or loss produced by the liquidation of the net present value of the derivative contract due to the collateralization agreement. The liquidity value adjustment can be computed with the other adjustments, hence we have the following relationship:

$$\mathrm{LVA}(t,T) = \mathrm{FVA}(t,T) - \mathrm{CollVA}(t,T) - \mathrm{HVA}(t,T) \tag{A.18}$$

Other Adjustments:

- Capital value adjustment (KVA): the tail risk capital constitutes a cost for an entity and it represents the cost that the entity has to deal with the tail risk. The tail risk capital cost is concrete for financial institutions (at least the regulated ones) because they have to allocate a capital as stated by the regulators in their balance sheet. Nevertheless, the non regulated institutions could take this risk into account to prevent unexpected losses.
- Margin value adjustment (MVA): the margin value adjustment come from initial margin is needed on a financial transaction. Since Q3 of 2016, initial margin for bilateral OTC derivatives are needed for new transactions between financial entities thus, they have to calculate the margin value adjustment.

Appendix B

Semi-Smooth Newton Method for American Option

With an semi-implicit FD Euler time scheme the American option problem becomes

$$\begin{array}{l} \dfrac{u^m - u^{m-1}}{\Delta t} - \dfrac{\partial}{\partial S}(\dfrac{X^2\sigma^2}{2}\dfrac{\partial u^m}{\partial S}) + ru^m \geq S(r - \dfrac{\sigma^2}{2})\dfrac{\partial u^{m-1}}{\partial S}, \\ u^m(S) \geq \phi(S), \quad S \in \mathbb{R}^+, \ t \in (0,T), \end{array} \tag{B.1}$$

with equality at each S on one of the two inequations, and initialized by $u^0(S) = \phi(S), \ S \in \mathbb{R}^+$.

Thus, at each time step, one must solve a problem of the type

$$Au \geq V, \quad u \geq \phi \text{ in } \mathbb{R}^+, \tag{B.2}$$

where A is the strongly elliptic symmetric operator

$$u \to \left(r + \frac{1}{\Delta t}\right) u - \frac{\partial}{\partial S}\left(\frac{X^2\sigma^2}{2}\frac{\partial u}{\partial S}\right) \quad \text{and } V = \frac{u^{m-1}}{\Delta t} + S\left(r - \frac{\sigma^2}{2}\right)\frac{\partial u^{m-1}}{\partial S}$$

The problem is also

$$\min_{u \in H^1(\mathbb{R}^+), u \geq \phi} \left\{ \frac{1}{2}a(u,u) - (V,u) \right\}$$

with

$$a(u,v) = \int_0^\infty \left(\alpha uv + \mu \frac{\partial u}{\partial S}\frac{\partial v}{\partial S}\right), \quad \alpha = r + \frac{1}{\Delta t} \quad \text{and } \mu = \frac{X^2\sigma^2}{2}.$$

Recall that an equation like $F(x) = 0$ (with $F : \mathbb{R}^n \to \mathbb{R}$) can be solved by Newton's method:

$$x_{k+1} = x_k - G(x_k)^{-1}F(x_k)$$

with $G = F'$ the Jacobian of F. Hintermuller et al. [93] observed that Newton's algorithm converges even if F is not differentiable provided that there exists G such that

$$\text{for all } x \quad \lim_{\|h\| \to 0} \|F(x+h) - F(x) - G(x+h)h\| = 0.$$

Such a property which is satisfied by $F(x) = \max\{0, x\}$ for instance with $G(x) = \max\{0, x\}/x$.

Itô and Kunisch [99] suggested to apply the idea to (B.2) reformulated as

$$\begin{aligned} &a(u, v) - (\lambda, v) = (f, v) \ \ \forall v \in H^1(\mathbb{R}^+), \ \ \textit{i.e. } Au - \lambda = f \\ &\lambda - \min\{0, \lambda + c(u - \phi)\} = 0, \end{aligned} \tag{B.3}$$

The last equality is equivalent to $\lambda \leq 0, \ \lambda \leq \lambda + c(u - \phi)$ i.e. $u \geq \phi, \ \lambda \leq 0$, with equality on one of them for each S. This problem is equivalent to (B.2) for a real constant $c > 0$ because λ is the Lagrange multiplier of the constraint.

Algorithm

Newton's algorithm 9 applied to (B.3) gives

Algorithm 9 Semi-Smooth Newton algorithm

1: Choose $c > 0, \ , u_0, \ \lambda_0$, set $k = 0$.
2: Determine $A_k := \{S \ : \ \lambda_k(S) + c(u_k(S) - \phi(S)) < 0\}$
3: Set $u_{k+1} = \arg\min_{u \in H^1(\mathbb{R}^+)} \{\frac{1}{2} a(u, u) - (f, u) \ : \ u = \phi \text{ on } A_k\}$
4: Set $\lambda_{k+1} = f - Au_{k+1}$

Appendix C

Asynchronous Parallelism

It is possible to achieve a very efficient optimization but it is not a mathematical one. Let's say we have n time steps on the coarse grid and N time steps on the fine grid. As soon as the first step is computed in the parareal algorithm, we have n independent problems. Actually, it is not required to wait until the end of this first step to start the computation inside each interval. One can use asynchronous parallelism to perform an efficient implementation of the parareal algorithm as displayed on figure C.1.

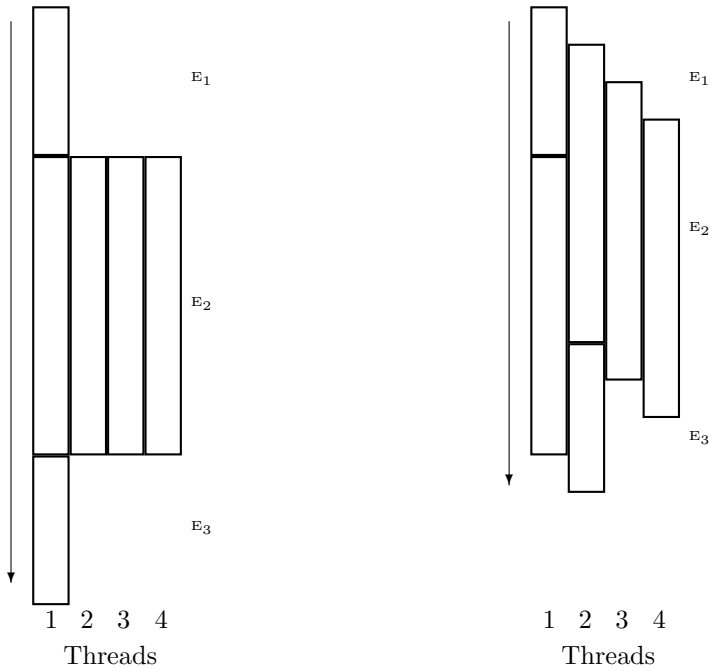

Figure C.1: Asynchronous parallelism vs Synchronous Parallelism. E_1 is the first step, E_2 is the second step i.e. computation inside each time step and E_3 is the correcting step. We take the example with 4 threads and the size of the rectangles denotes the size of the task to do.

Appendix D

Libor Market Model (LMM)

Dans cet appendice, nous introduisons le Libor Market Model pour la dynamique des taux de change anticipé. Cette description ne couvre pas tous les aspects de ce modèle qui est très répandu dans la pratique, nous suggérons au lecteur la bibliographie suivante Brace et al. [27], Brigo et al. [32], Musiela and M. Ruykowski [136] and Davis [50].

Préliminaire

Soit $t = 0$ la date d'aujourd'hui et un ensemble de dates dans le future notées $T_0, \ldots, T_N$, $N \in \mathbb{N}^*$ avec un couple de date de maturité $(T_{i-1}, T_i)_{i \in \mathbb{N}^*}$ pour un ensemble de taux de change anticipés. Soit $\tau_1, \ldots, \tau_N$ l'ensemble des fractions d'année correspondant à $\tau_i = \dfrac{T_i - T_{i-1}}{\text{un an}}$, $i \in \mathbb{N}^*$ et les taux $L_t^k = L(t, T_{k-1}, T_k)$, $k \in \mathbb{N}^*$ qui n'expire qu'à la maturité T_{k-1}. On note $P_t^k = P(t, T_k)$[1]. On sait que:

$$L(t, S, T) = \frac{1}{(T-S)} \left(\frac{P(t,T)}{P(t,S)} - 1 \right), \quad t < S < T. \tag{D.1}$$

Sachant que la probabilité de mesure $\mathbb{Q}^k$[2] associée au numéraire P^k, on peut écrire d'après l'équation D.1 et selon Brigo et Mercurio [31] que

$$L_k^t = \frac{\left[\frac{P_t^{k-1} - P_t^k}{\tau_k} \right]}{P_t^k}, \quad t > 0. \tag{D.2}$$

Par conséquence, le prix de la différence entre deux coupons actualisés avec comme notionnel $\dfrac{1}{\tau_k}$ par rapport à la mesure P_t^k doit être une martingale par rapport à la mesure $\mathbb{Q}^k$. Donc si L^k est approché par un processus de diffusion, il est nécessaire que celui-ci soit sans drift sous la mesure $\mathbb{Q}^k$. Nous obtenons la dynamique suivante:

$$dL_t^k = L_t^k \sigma_t^k dW_t^k, \quad t \leq T_{k-1}, \tag{D.3}$$

[1] La maturité du taux de change anticipé correspond à la maturité du coupon

[2] $\mathbb{Q}^k$ est aussi appelée mesure du taux de change anticipé

où W_t^k est un mouvement Brownien standard de dimension M avec une matrice de covariance associée ρ telle que $dW_t^k dW_t^l = (\rho_t^{k,l})_{1\leq k,l\leq M}$ et σ_t^k est un vecteur de volatilité de dimension M.

Changement de Numéraire

Dans cette section, on note $\mathbb{E}$ l'espérance sous la probabilité risque neutre et par $\tilde{\mathbb{E}}$, l'espérance sous la probabilité de la mesure du taux de change anticipé.

Un numéraire est un processus strictement positif $\mathscr{F}_t$-adapté et $t0$, le processus du numéraire noté N_t peut être vu comme une unité de base dans la monnaie de référence lors de l'évaluation d'un l'actif financier sous-jacent. La plupart du temps, le prix de l'actif S_t dans le numéraire N_t est noté X_t et est défini par

$$X_t = \frac{S_t}{N_t} \tag{D.4}$$

Quelques exemples de numéraires

- Le taux des marchés monétaires est donné par:

$$N_t = \exp\left(\int_0^t r_s ds\right), \tag{D.5}$$

  où r_t est le taux d'intérêt sans risque qui peut être potentiellement stochastique et dépendant du temps. Dans ce cas, on aura

$$X_t = e^{-\int_0^t r_s ds} S_t, \tag{D.6}$$

  ce qui est équivalent au prix actualisé de l'actif sous-jacent à la vu d'aujourd'hui soit en $t = 0$.

- Un taux de change par rapport à une monnaie étrangère $N_t = R_t$. Alors, dans ce cas on a

$$X_t = \frac{S_t}{R_t}, \tag{D.7}$$

  ce qui est égale au prix de l'actif sous-jacent dans l'unité de base de la monnaie étrangère.

- Le prix $P(t,T) = \mathbb{E}\left[e^{-\int_t^T r_s ds} \mid \mathscr{F}_t\right]$ d'un coupon valant 1 à maturité T (i.e. $P(T,T) = 1$). Si tel est le cas, nous pouvons vérifier que $e^{-\int_0^t r_s ds} P(t,T) = \mathbb{E}\left[e^{-\int_0^T r_s ds} \mid \mathscr{F}_t\right]$ est une martingale.

Retournons à l'évaluation d'une option par un changement de mesure d'un numéraire N_t, en supposant que

- Sous la mesure de probabilité risque neutre, le numéraire actualisé $\left(e^{-\int_0^t r_s ds} N_t\right)_t$ est une $\mathscr{F}_t$-martingale.

Sachant que le processus N_t est une probabilité de mesure, nous pouvons établir que la mesure de probabilité du taux de change anticipé $\mathbb{Q}$ est défini par $\frac{d\mathbb{Q}}{d\mathbb{P}} = e^{-\int_0^T r_s ds} \frac{N_T}{N_0}$. Dans ce cas, on peut réécrire

$$\tilde{\mathbb{E}}[\xi] = \mathbb{E}\left[e^{-\int_0^T r_s ds} \frac{N_t}{N_0} \xi\right], \tag{D.8}$$

pour toute variable ξ intégrable et $\mathscr{F}_t$ mesurable. Alors, on peut prouver la proposition suivante:

Modèle de Simulation (Libor Market Model)

Proposition 8. *Soit X_t un processus continu $\mathscr{F}_t$-adapté définissant le prix d'un actif financier sous-jacent tel que $e^{-\int_0^t r_s ds} X_t$ est une martingale sous la mesure de probabilité risque neutre $\mathbb{P}$, alors par changement de numéraire, le prix du processus X_t, exprimé en unité de base du numéraire N_t, le processus $\left(\frac{X_t}{N_t}\right)$ est aussi une martingale sous la mesure de probabilité risque $\mathbb{P}$, c'est-à-dire*

$$\tilde{\mathbb{E}}\left[\frac{X_t}{N_t} \mid \mathscr{F}_s\right] = \frac{X_s}{N_s} = \mathbb{E}\left[e^{-\int_0^t r_s ds} X_t \mid \mathscr{F}_s\right], \quad 0 \leq s \leq t. \tag{D.9}$$

Preuve. Pour toutes variables aléatoires $\mathscr{F}_s$-mesurable Y bornées , on peut voir que

$$\begin{aligned}
\tilde{\mathbb{E}}\left[Y\frac{X_t}{N_t}\right] &= \mathbb{E}\left[Y\frac{X_t}{N_t}e^{-\int_0^t r_u du}\frac{N_t}{N_0}\right] \\
&= \mathbb{E}\left[Y\frac{N_t}{N_0}e^{-\int_0^s r_u du}\mathbb{E}\left[\frac{X_t}{N_t}e^{-\int_s^t r_u du}\frac{N_t}{N_s} \mid \mathscr{F}_s\right]\right] \\
&= \mathbb{E}\left[Y\mathbb{E}\left[\frac{d\mathbb{Q}}{d\mathbb{P}} \mid \mathscr{F}_s\right]\mathbb{E}\left[\frac{X_t}{N_t}e^{-\int_s^t r_u du}\frac{N_t}{N_s} \mid \mathscr{F}_s\right]\right] \\
&= \mathbb{E}\left[Y\frac{d\mathbb{Q}}{d\mathbb{P}}\mathbb{E}\left[\frac{X_t}{N_t}e^{-\int_s^t r_u du}\frac{N_t}{N_s} \mid \mathscr{F}_s\right]\right] \\
&= \tilde{\mathbb{E}}\left[Y\mathbb{E}\left[\frac{X_t}{N_t}e^{-\int_s^t r_u du}\frac{N_t}{N_s} \mid \mathscr{F}_s\right]\right] \\
&= \tilde{\mathbb{E}}\left[Y\frac{X_s}{N_s}\right],
\end{aligned}$$

parce que $\left(e^{-\int_0^t r_u du} X_t\right)_t$ est une martingale sous la mesure de probabilité risque neutre $\mathbb{P}$. Finalement, on obtient

$$\tilde{\mathbb{E}}\left[Y\frac{X_t}{N_t}\right] = \tilde{\mathbb{E}}\left[Y\frac{X_s}{N_s}\right], \tag{D.10}$$

pour toutes $\mathscr{F}_s$-mesurable Y bornées impliquent D.9. □

Dynamique du Taux de Change Anticipé

Soit le numéraire de l'actif sous-jacent $B_t = \dfrac{P_t^{\mu_t}}{\prod_{j=0}^{\mu_t-1} P_{T_{k-1}}^j}$ et en appliquant la technique du changement de numéraire, on obtient la dynamique suivante pour le taux de change anticipé:

$$\frac{dL_t^k}{L_t^k} = \sigma_t^k \cdot \left(\sum_{j=\mu_t}^{k} \frac{\tau_j \sigma_t^j L_t^j}{1+\tau_j L_t^j} + dW_t^k\right), \quad t > 0, \tag{D.11}$$

où $\sigma_t^k = \|\sigma_t^k\|$ est la volatilité instantanée du taux de change anticipé. La corrélation est les taux de change anticipé est définie comme:

$$\rho_t^{k,l} = \frac{\sigma_t^k \sigma_t}{\|\sigma_t\|\|\sigma_t\|}, \quad t > 0, \tag{D.12}$$

ce qui nous permet de réécrire l'équation précédente D.11 de l'équation de diffusion des taux de change anticipé donnés par:

$$L_{t+\delta t}^{k} = L_t^k \exp\left[\sigma_t^k \left(\sum_{j=\mu_t}^{k} \frac{\tau_j \rho_{j,k} \sigma_t^k L_t^j}{1 + t_j L_t^j} \delta t - \frac{1}{2}\sigma_t^k \delta t + \sqrt{\delta t} Z^k\right)\right], \quad Z_k \sim \mathcal{N}(0,1), \quad t > 0, \quad \text{(D.13)}$$

avec $\delta t > 0$.

Appendix E

Résultats de l'Implémentation Visant une Architecture Hybride

---

```
The pricer has already been configured with this hardware env.
Optimal split MIC/CPU for the pricing of SWAP: 0.178716
Optimal split MIC/CPU for the pricing of CAP: 0.135991
Allocation and data creation + loading...
End of Allocations
pthread_create() for thread 0 (for cpu) returns: 0
pthread_create() for thread 1 (for mic 0) returns: 0
pthread_create() for thread 2 (for mic 1) returns: 0
pthread_create() for thread 3 (for mic 2) returns: 0
pthread_create() for thread 4 (for mic 3) returns: 0
pthread_create() for thread 5 (for mic 4) returns: 0
pthread_create() for thread 6 (for mic 5) returns: 0
 (preparing the MIC 4)5.639844,
 (preparing the MIC 0)5.656252,
 (preparing the MIC 3)5.656505,
 (preparing the MIC 1)5.663153,
 (preparing the MIC 5)5.668649,
 (preparing the MIC 2)5.667520,
 (MIC 2)0.044108s, 0th
 (MIC 3)0.045394s, 0th
 (CPU)0.045561s, 0th
 (MIC 0)0.071489s, 0th
 (MIC 4)0.082102s, 0th
 (MIC 5)0.082161s, 0th
 (MIC 1)0.082706s, 0th
 (CPU)0.038778s, 1th
 (MIC 2)0.077374s, 1th
 (MIC 5)0.042199s, 1th
 (CPU)0.042466s, 2th
 (MIC 0)0.070718s, 1th
 (MIC 3)0.105413s, 1th
```

```
 (MIC 1)0.070287s, 1th
...


 (MIC 1)0.003216s, 99th
 (MIC 5)0.010040s, 99th
 (MIC 3)0.012885s, 99th
 (MIC 0)0.002786s, 99th
 (MIC 2)0.011828s, 99th
 (MIC 4)0.002446s, 99th

Pricing time: 3.68959s (substracting 5.67992s)
```

Bibliographie

[1] L. Abbas-Turki and B. Lapeyre. American option pricing on multi-core graphic cards. *Business Intelligence and Financial Engineering*, 1:307–311, 2009. International Conference, BIFE 09.

[2] Y. Achdou and O. Pironneau. *Computation methods for option pricing.* Frontiers in Applied Mathematics. SIAM, Philadelphia, United States of America, 2005. xviii+297 pp., ISBN 0-89871-573-3.

[3] F. Aitsahlia and P. Carr. American options: A comparison of numerical methods. *Numerical Methods in Finance*, pages 67–87, 1997.

[4] S. Alanko and M. Avellaneda. Reducing variance in the numerical solution of BDSEs. *Comptes Rendus Mathématiques de l'Académie des Sciences*, 351(3–4):135–138, 2012.

[5] M. B. Alaya and A. Kebaier. Multilevel Monte Carlo for Asian options and limit theorems. *Monte Carlo Methods and Applications*, 20(3):181–193, 2014.

[6] T. Alm, B. Harrach, T. Harrach, and M. Keller. A Monte Carlo algorithm for autocallables that allows for stable differentiation. *Journal of Computational Finance*, 17(1):43–70, 2013.

[7] M. Altmayer and A. Neuenkirch. Multilevel Monte Carlo quadrature of discontinuous payoffs in the generalized Heston model using Malliavin integration by parts. *SIAM, Journal of Financial Mathematics*, 6(1):22–52, 2015.

[8] E. Atanassov, M. Barth, M. Byckling, V. Codreanu, N. Ilieva, T. Karasek, J. Rodriguez, S. Saarinen, O. W. Saastad, M. Schliephake, M. Stachon, J. Strassburg, and V. Weinberg. *Best Practive Guide Intel Xeon Phi v2.0.* Prace, United States of America, 2017. iv+118 pp., Available at `http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Intel-Xeon-Phi-1.pdf`.

[9] M. L. Bachelier. *Théorie de la Spéculation.* PhD thesis, Université de Paris, Paris, France, 1900.

[10] G. Bal and Y. Maday. A "parareal" time discretization for non-linear PDE's with application to the pricing of an American put. *Recent Developments in Domain Decomposition Methods*, 23:189–202, 2002.

[11] V. Bally and G. Pagès. A quantization algorithm for solving discrete time multidimensional optimal stopping problems. *Bernoulli*, 6:1003–1049, 2003.

[12] G. Barlas. *Multicore and GPU Programming: An Integrated Approach.* Design & Architecture. Morgan Kaufmann, Waltham, Massachusets, United States of America, 1 edition, 2014. xix+673 pp., ISBN 97860-124-17137-4.

[13] M. S. Basurto and M. Singh. Counterparty risk in the Over-The-Counter derivatives market. FMI, Available at `https://www.imf.org/en/Publications/WP/Issues/2016/12/31/Counterparty-Risk-in-the-Over-The-Counter-Derivatives-Market-22447`, 2008.

[14] D. Belomestny, F. Dickmann, and T. Nagapetyan. Multilevel dual approach for pricing American style derivatives. *Finance and Stochastics*, 17(4):717–742, 2013.

[15] D. Belomestny, F. Dickmann, and T. Nagapetyan. Pricing Bermudan option via multilevel approximation methods. *SIAM, Journal of Financial Mathematics*, 6(1):448–466, 2015.

[16] C. Bender and J. Steiner. Least-squares Monte Carlo for backward SDEs. *Numerical Methods in Finance*, 12:257–289, 2010.

[17] M. Benguigui. *Valorisation d'options Américaines et Value-At-Risk de portefeuille sur cluster de GPUs\CPUs*. PhD thesis, Université Nice Sophia Antipolis, France, 2015.

[18] M. Benguigui and F. Baude. Towards parallel and distributed computing on GPU for American basket option pricing. International Workshop on GPU Computing in Cloud in conjunction with 4th IEEE international conference on Cloud Computing Technology and Science, Taipei, 2012.

[19] E. Benhamou. Optimal Malliavin weighting function for the computation of the greeks. *Mathematical Finance*, 13:37–53, 2003.

[20] Y. Z. Bergman. Option pricing with differential interest rates. *Review of Financial Studies*, 8(2):475–500, 1995.

[21] J. M. Bismut. Conjugate convex functions in optimal stochastic control. *Journal of Mathematical Analysis and Applications*, 44:383–404, 1973.

[22] J. M. Bismut. *Contrôle des Systèmes Linéaires Quadratiques: Applications de l'intégrale Stochastique*, volume 649 of *Lecture Notes in Mathematics*. Springer-Verlag, New York, United States of America, 1978.

[23] J. M. Bismut, K. D. Elworthy, and X. M. Li. Bismut type formulae for differential forms. *Probability Theory*, 327:87–92, 1998.

[24] F. Black and J. C. Cox. Valuing corporate securities: some effects of bond indenture provisions. *Journal of Finance*, 31:351–367, 1976.

[25] B. Bouchard and R. Elie. Discrete-time approximation of decoupled forward-backward SDE with jumps. *Stochastic Processes and applications*, 118:53–75, 2008.

[26] B. Bouchard and N. Touzi. Discrete-time approximation and Monte Carlo simulation of backward stochastic differential equations. *Stochastic Processes and their Applications*, 111:175–206, 2004.

[27] A. Brace, D. Gatarek, and M. Musiela. The market model of interest rate dynamics. *Mathematical finance*, 7:127–155, 1997.

[28] P. Briand, B. Delyon, and J. Mèmin. Donsker-type theorem for BSDEs. *The Electronic Communications in Probability*, 1:1–14, 2001.

[29] M. Briane and G. Pagès. *Théorie de l'intégration, convolution et transformée de Fourier, Cours et Exercices*. Vuibert, 7 edition, 2017. 400 pp.

[30] D. Brigo and M. Masetti. *Risk neutral pricing of counterparty risk*, 2005. Risk book.

[31] D. Brigo and F. Mercurio. *Interest Rate Models and Practice*. Finance. Springer-Verlag, Berlin, Germany, 2 edition, 2006. xxii+852 pp. ISBN 978-3-540-34604-3.

[32] D. Brigo, F. Mercurio, and M. Morini. The Libor model dynamics: approximations, calibrations and diagnostics. *European Journal of Operation Research*, 163:30–41, 2005.

[33] D. Brigo, F. Mercurio, F. Rapisarda, and R. Scotti. Approximated moment-matching dynamics for Basket options simulation. *Quantitative Finance*, 1:1–16, 2003.

[34] D. Brigo, M. Morini, and A. Pallavicini. *Counterparty Credit Risk, Collateral and Funding: With Pricing Cases For Asset Classes*. Finance. Wiley, Washington, United States of America, 1 edition, 2013. xv+449 pp., ISBN 978-0-470-74846-6.

[35] M. Broadie and P. Glasserman. Estimating security price derivatives using simulation. *Management Science*, 42(2):269–285, 1996.

[36] S. Burgos. *The computation of greeks with multilevel Monte Carlo*. PhD thesis, University of Oxford, Oxford, United Kingdom, 2014.

[37] S. Burgos and M. B. Giles. Computing greeks using jmultilevel path simulation. In L. Plaskota and H. Woźniakowski, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2010*, pages 281–296. Springer-Verlag, Berlin, Germany, 2010.

[38] D. R. Butenhof. *Programming with POSIX Threads*. Professional Computing. Addison-Wesley, Massachusets, United States of America, 1 edition, 1997. xvi+381 pp., ISBN 0-201-63392-2.

[39] L. Capriotti. Fast greeks by algorithmic differentiation. *Journal of Computational Finance*, 14(3):3–35, 2011.

[40] L. Capriotti. Likelihood ratio method and algorithmic differentiation: fast second order greeks. *Algorithmic Finance*, 4(1–2):81–87, 2015.

[41] A. P. Caverhill and N. Webber. American options: theory and numerical analysis. In S. Hodges, editor, *Options: recent advances in theory and practise*, pages 80–94. Manchester University Press, Manchester, England, 1990.

[42] D. Chevance. Numerical methods for backward stochastic differential equations. *Numerical methods in finance*, pages 232–244, 1997.

[43] G. Chrysos. *Intel Xeon Phi Coprocessor Architecture Overview*, 2012. Available at `http://www.training.prace-ri.eu/uploads/tx_pracetmo/MIC_Intro_Architecture.pdf`.

[44] L. Clewlow, J. Llanos, and C. Strickland. Pricing exotic options in a Black Scholes world. Available at `http://www2.warwick.ac.uk/fac/soc/wbs/subjects/finance/research/wpaperseries/1994/94-54.pdf`, 1994.

[45] J. Cocke and D. S. Slotnick. The use of parallelism in numerical calculations. RC-55, IBM Research Center, Yorktown Heights, New York, United States of America, July 1958.

[46] J. Cox and S. Ross. The evaluation of options for alternative stochastic processes. *Journal of Finance Economics*, 3(1):145–166, 1976.

[47] S. Crépey. XVA: about CVA, DVA, FVA and other market adjustments. Preprint of Opinion and Debates, 2014.

[48] D. Crisan and K. Manolarakis. Solving backward stochastic differential equations using cubature method. *SIAM, Journal of Financial Mathematics*, 3(1):534–571, 2012.

[49] D. M. Dang, C. C. Christara, and K. R. Jackson. An efficient graphics processing unit-based parallel algorithm for pricing multi-asset American options. *Concurrency and Computation: Practice and Experience*, 24(18):849–866, 2012.

[50] M. Davis. A note on the Forward measure. *Finance and Stochastics*, 2:19–28, 1998.

[51] J. Douglas, J. Ma, and P. Protter. Numerical methods for forward-backward stochastic differential equations. *The Annals of Applied Probability*, 6:940–968, 1996.

[52] P. S. Dywer and M. S. Macphail. Symbolic matrix derivatives. *The Annals of Mathematical Statistics*, 19(4):517–534, 1948.

[53] M. Fatica and E. Phillips. Pricing American options with least squares Monte carlo on GPUs. ACM, 6th Workshop on High Performance Computational Finance, 2013.

[54] G. Fourestey. *Intel Xeon Phi Programming Models*, 2013. Available at `http://www.lnm.mw.tum.de/fileadmin/w00bis/www/pictures/conferences/EuroTUG2013/fourestey_eurotug.pdf`.

[55] E. Fournié, J. M. Lasry, J. Lebuchoux, and P. L. Lions. Application of Malliavin calculus to Monte Carlo methods in finance. *Finance and Stochastics*, 2(5):201–236, 2001.

[56] M. J. Gander and S. Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM, Journal of Scientific Computing*, 29(2):556–578, 2007.

[57] A. Al Gerbi, B. Jourdain, and E. Clément. Ninomiya-Victoir scheme: strong convergence, antithetic version and application to multilevel estimators. *Monte Carlo Methods and Applications*, 22(3):197–228, 2016.

[58] M. Giles and P. Glasserman. Smoking adjoints: fast evaluation of greeks in Monte Carlo calculations. NA-05/15, Numerical Analysis Group, Oxford University, Oxford, England, July 2005.

[59] M. B. Giles. Improved multilevel Monte Carlo convergence using the Milstein scheme. *Monte Carlo and Quasi Monte Carlo Methods 2006*, pages 343–358, 2008.

[60] M. B. Giles. Vibrato Monte Carlo sensitivities. In P. L'Ecuyer and A. Owen, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2008*, pages 369–382. Springer-Verlag, New York, United States of America, 2008.

[61] M. B. Giles. Multilevel Monte Carlo methods. *Acta Numerica*, 24:259–328, 2015.

[62] M. B. Giles. Monte Carlo evalutation of sensitivities in computational finance. Available at `eprints.maths.ox.ac.uk/1090/1/NA-07-12.pdf`, September 20–22, 2007.

[63] M. B. Giles, D. J. Higham, and X. Mao. Analysing multilevel Monte Carlo for options with non-globally Lipstchitz payoff. *Finance Stochastics*, 13(3):403–413, 2009.

[64] M. B. Giles and L. Szpruch. Multilevel Monte Carlo methods for applications in finance. In T. Gerstner and P. Kloeden, editors, *Recent Developements in Computational Finance*, volume 14, pages 3–47. World Scientific, 2013.

[65] M. B. Giles and B. J. Waterhouse. Multilevel quasi Monte Carlo path simulation. *Radon Series on Computational and Applied Mathematics*, pages 165–181, 2009.

[66] M. G. Giles and L. Szpruch. Antithetic multilevel Monte Carlo estimation for multidimensional SDEs without lévy area simulation. *The Annals of Applied Probability*, 24(4):1585–1620, 2014.

[67] D. Giorgi. *Théorèmes limites pour estimateurs Multilevel avec et sans poids. Comparaisons et Applications*. PhD thesis, Université Pierre et Marie curie, Paris, France, 2017.

[68] D. Giorgi, V. Lemaire, and G. Pagès. Limit theorems for weighted and regular multilevel estimators. *Monte Carlo Methods and Applications*, 3(1):43–70, 2017.

[69] J.-Y. Girard and I. M. Toke. Monte Carlo valuation of multidimensional American option through grid computing. *Lecture Notes in Computer Science*, 3743:462–469, 2006.

[70] P. Glasserman. *Gradient estimation via pertubation analysis*. Mathematics. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991. xiv+229 pp., ISBN 0-7923-9095-4.

[71] P. Glasserman. *Monte Carlo methods in financial engineering*, volume 53 of *Application of Mathematics*. Springer-Verlag, New York, United States of America, 2003. xiii+598 pp., ISBN 0-387-00451-3.

[72] P. Glasserman and J. Staum. Conditioning on one-step survival for Barrier option simulations. *Operations Research*, 49(6):923–937, 2001.

[73] P. Glasserman and X. Zhao. Fast greeks by simulation in forward LIBOR models. *Journal of Computational Finance*, 3(1):5–39, 1999.

[74] P. W. Glynn. Likelihood ratio gradient estimation: an overview. In *Proceedings of the Winter Simulation Conference*, pages 366–374, New York, United States of America, 1987. IEEE Press.

[75] E. Gobet and J. P. Lemor. Numerical solution of BSDEs using empirical regression methods: theory and practice. *Proceedings of the Fifth Colloquium on BSDEs*, 2006.

[76] E. Gobet, J. P. Lemor, and X. Warin. A regression-based Monte Carlo method to solve backward stochastic differential equations. *The Annals of Applied Probability*, 15:2172–2202, 2005.

[77] E. Gobet and R. Munos. Sensitivity analysing using Itô-Malliavin calculus and martingales: applications to stochastic optimal control. *SIAM Journal on Control and Optimization*, 43(5):1676–1713, 2005.

[78] A. Green. KVA: Capital valuation adjustment. Available at: `https://arxiv.org/abs/1405.0515`, 2014.

[79] J. Gregory. *Counterparty credit risk, the new challenge for global financial markets*. Applications of Mathematics. John Wiley & Sons, Chippenham, Great Britain, wiley finance series edition, 2010. xxiv+424 pp., ISBN 978-0-470-68576-1.

[80] J. Gregory. *The xVA Challenge: counterparty credit risk, funding, collateral and capital*. Wiley series in financial engineering. John Wiley& Sons Ltd, Cornwall, United Kingdom, 2015. xxvi+455 pp., ISBN 978-1-119-10942-6.

[81] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 83–108. Kluwer Akedemic Publishern Dordrecht, 1989.

[82] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Frontiers in Applied Mathematics. SIAM, Philadelphia, United States of AmericaF, 2008. xxi+426 pp., ISBN 978-0-89871-659-7.

[83] A. Griewank and A. Walther. *ADOL-C: A Package for the Automatic differentiation of algorithm written in C/C++*. University of Paderborn, Germany, 2010.

[84] M. Grundmann and M. Masmodi. MIPAD: an AD package. AD'2000, Nice, France, 2000.

[85] A. L. Haji-Ali, F. Nobile, and R. Tempon. Multi-index Monte Carlo: when sparsity meets sampling. *Numerische Mathematik*, 132(4):767–806, 2016.

[86] M. Harris. *Optimizing Parallel Reduction in CUDA*, 2012. Available at `http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf`.

[87] L. Hascoët and V. Pascual. The Tapenade automatic differentiation tool: principles, model, and specification. *ACM Transactions On Mathematical Software*, 39(3), 2013.

[88] S. Heinrich. Monte Carlo complexity of global solution of integral equations. *Journal on Complexity*, 14:151–175, 1998.

[89] S. Heinrich. The multilevel method of dependent tests. In N. Balakrishnan, V. B. Melas, and S. M. Ermakov, editors, *Advances in Stochastic Simulation Methods*, Statistics for Industry and Technology, pages 47–62. Birkäuser, Boston, United States of America, 2000.

[90] S. Heinrich. Multilevel Monte Carlo methods. *Lecture Notes in Computer Science*, 2179:58–67, 2001.

[91] C. Herrera and L. Paulot. Parallel American Monte Carlo. Sophis Quantitative Research, Paris, Working paper, 2014.

[92] S. L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, 6:327–343, 1993.

[93] M. Hintermüller, K. Ito, and K. Kunish. The Primal-Dual active set strategy as a Semi-smooth Newton method. *SIAM Journal on Control and Optimization*, 3(13):865–888, 2003.

[94] Y. C. Ho and X. R. Cao. Optimization and perturbation analysis of queuing networks. *Journal of Optimization Theory and Applications*, 40:559–582, 1983.

[95] R. J. Hogan. Fast reverse-mode automatic differentiation using expression templates in C++. *Transactions on Mathematical Software*, 40(26):1–26, 2014.

[96] C. Homescu. Adjoints and automatic (algorihmtic) differentiation in computational finance. arXiv:1107.1831, 2011.

[97] Y. Hu, Q. Lu, Z. Cao, and J. Wang. Parallel simuation of high-dimensional American option pricing based on CPU versus MIC. *Concurrency and Computation: Practice and Experence*, 27(15):1110–1121, 2015.

[98] P. J. Hunt and J. E. Kennedy. *Financial Derivatives in theory and practice*. Wiley series in probability and statistics. John Wiley& Sons Ltd, Chichester, England, 2004. xxi+449 pp., ISBN 0-470-86359-5.

[99] K. Ito and K. Kunisch. Semi-smooth Newton method for variational inequalities of the first kind. *ESAIM: Mathematical Modelling and Numerical Analysis*, 37(1):41–62, 2010.

[100] D. James. *Intel Xeon Phi MIC Offload Programming Models*, 2013. Available at `https://portal.tacc.utexas.edu/documents/13601/901837/offload_slides_DJ2013-3.pdf`.

[101] R. A. Jarrow and F. Yu. Counterparty risk and the pricing of defautable securities. *Journal of Finance*, 56:1765–1799, 2001.

[102] J. Jeffers. *Intel Many Integrated Core Architecture: An Overview and Programming Models*, 2012. Available at `https://www.olcf.ornl.gov/wp-content/training/electronic-structure-2012/ORNL_Elec_Struct_WS_02062012.pdf`.

[103] J. Jeffers and J. Reinders. *Intel Xeon Phi Coprocessor High Performance Programming*. Hardware and Architecture. Morgan Kaufmann, Waltham, Massachusets, United States of America, 1 edition, 2013. xx+409 pp., ISBN 978-0-124-10414-3.

[104] J. Jeffers and J. Reinders. *High Performance Parallelism Pears Volume Two*. Hardware and Architecture. Morgan Kaufmann, Waltham, Massachusets, United States of America, 1 edition, 2015. xliv+502 pp., ISBN 978-0-128-03819-2.

[105] C. Joy, P. P. Boyle, and K. S. Tan. Quasi Monte Carlo methods in numerical finance. *Journal of Economics and Management Science*, 42(6):926–938, 1996.

[106] N. El Karoui. *Couverture des risques dans les marchés financiers*, 2004. Available at: `www.cmap.polytechnique.fr/~elkaroui/masterfin034.pdf`.

[107] N. El Karoui, C. Kapoudjian, and E. Pardoux. Reflected solutions of backward SDEs and related obstacle problem for PDE's. *The Annals of Probability*, 25(2):702–737, 1997.

[108] N. El Karoui and L. Mazliak. *Backward Stochastic Differential Equations*, volume 364 of *Mathematics Series*. Addison Wesley Longman Limited, Harlow, England, 1997. v+223 pp., ISBN 0-582-30733-3.

[109] N. El Karoui, E. Pardoux, and M. C. Quenez. Reflected backward SDEs and American options. *Numerical Methods in Finance*, pages 215–231, 1997.

[110] N. El Karoui, S. Peng, and M. C. Quenez. Backward stochastic differential equation in finance. *Mathematical Finance*, 7(1):1–71, 1997.

[111] A. G. Z. Kemna and A. C. F. Vorst. A pricing method for options based on average asset values. *Journal of Banking and Finance*, 14:113–129, 1990.

[112] L. Khodja, J.-Y. Girard, R. Couturier, and P. Spitéri. Parallel solution of American option derivatives on GPU clusters. *Computers and Mathematics with Applications*, 65(111):1830–1848, 2013.

[113] D. B. Kirk and W. W. Hwu. *Programming Massively Parallel Processors, A Hands-on Approach*. Parallel Programming. Morgan Kaufmann, Waltham, Massachusets, United States of America, 2 edition, 2016. xxiii+553 pp., ISBN 978-0-128-11986-0.

[114] P. E. Kloeden and E. Platen. *Numerical solution of stochastic differential equations*. Stochastic Modelling and Applied Probability. Springer-Verlag, Berlin, Germany, 1992. xxxvi+630 pp., ISBN 978-3-642-08107-1.

[115] H. Kunita. *Stochastic Flows and Stochastic Differential Equations*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, England, 1990. xiv+361 pp., ISBN 0-521-35050-6.

[116] C. Labart and J. Lelong. A parallel algorithm for solving BSDEs. *Monte Carlo Methods and Applications*, 19(1):11–39, 2013.

[117] S. Larsen, E. Witchel, and S. P. Amarasingha. Increasing and detecting memory address congruence. *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, pages 18–29, 2002.

[118] P. L'Ecuyer. A unified view of the IPA, SF and LR gradient estimation techniques. *Management Science*, 36(11):1364–1383, 1990.

[119] V. Lemaire and G. Pagès. Multistep Richardson-Romberg extrapolation. *Bernoulli*, 23(4A):2643–2692, 2014.

[120] J. P. Lemor, E. Gobet, and X. Warin. Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations. *Bernoulli*, 12:889–916, 2006.

[121] E. Levy. Pricing European average rate and currency options. *Journal of International Money and Finance*, 11:474–491, 1992.

[122] J.-L. Lions, Y. Maday, and G. Turinici. Résolution d'EDP par un schéma en temps "pararéel". *Comptes Rendus Matématiques de l'Académie des Sciences*, 332:661–668, 2000.

[123] J. London. *Modeling derivatives in C++*. Wiley finance series. John Wiley & Sons, Hoboken, New Jersey, United States of America, 2005. xix+820 pp., ISBN 0-471-65464-7.

[124] F. A. Longstaff and E. S. Schwartz. Valuing American options by simulation: a simple least squares approach. *Review of Financial Studies*, 14:113–148, 2001.

[125] D. Lu. *The XVA of Financial Derivatives: CVA, DVA and FVA Explained*. Palgrave Macmillan, London, England, 2015. xix+217 pp., ISBN 978-1-137-43584-2.

[126] J. Ma, P. Protter, J. San Martin, and S. Torres. Numerical method for forward stochastic differential equations. *The Annals of Applied Probability*, 12:302–316, 2002.

[127] J. Ma, J. Shen, and Y. Zhao. On numerical approximation of forward-backward stochastic differential equations. *SIAM, Journal Numerical Analysis*, 46:2636–2661, 2009.

[128] J. Ma and J. Yong. *Forward-Backward Stochastic Differential Equations and Their Applications*, volume 1702 of *Lecture Notes in Mathematics*. Springer-Verlag, Germany, Berlin, 2 edition, 1999. xiii+273 pp., ISBN 978-3-540-65960-0.

[129] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM, Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation*, 8(1):3–30, 1998.

[130] C. Mbaye, G. Pagès, and F. Vrins. An antithetic approach of multilevel Richardson-Romberg extrapolation estimafor for multidimensional SDEs. In I. Dimov, I. Faragó, and L. Vulkov, editors, *NAA 2016, Lecture Notes in Computer Science*, volume 10187, pages 482–491. Springer, Cham, numerical analysis and its applications edition, 2017.

[131] J. D. McCalpin. *Native Computing and Optimization on the Intel Xeon Phi Coprocessor*, 2013. Availabe at `https://www.ixpug.org/docs/IXPUG_2014_MIC_Native-new.pdf`.

[132] G. N. Milstein and M. V. Tretyakov. Numerical algorithm for forward-backward stochastic differential equations. *SIAM, Journal on Scientific Computing*, 28:561–582, 2006.

[133] K. R. Miltersen, K. Sandmann, and D. Sondermann. Closed form solutions for term structure derivatives and log-normal interest rates. *The Journal of Finance*, 52:409–430, 1997.

[134] R. L. Mishkov. Generalization of the formula of fa'a di Bruno for a composite function with a vector argument. *Internation Journal of Mathematics and Mathematical Sciences*, 24(7):481–491, 2000.

[135] A. Munshi, B. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg. *OpenCL Programming Guide*. Parallel Programming. Addison-Wesley Professional, Boston, United States of America, 1 edition, 2011. xxviii+620 pp., ISBN 978-0-321-74964-2.

[136] M. Musiela and M. Ruykowski. Continuous-time term structure models: Forward measure approach. *Finance and Stochastics*, 4:261–929, 1997.

[137] U. Naumann. *The art of differentiating computer programs: an introduction to algorithmic differentiation.* Software, Enivronments and Tools. SIAM, RWTH Aachen University, Aachen, Germany, 2012. xviii+333 pp., ISBN 978-1-61197-206-1.

[138] D. Nualart. *The Malliavin calculus and related topics.* Probability and its Applications. Springer-Verlag, Berlin, Germany, 2006. x+390 pp., ISBN 978-3-540-28328-7.

[139] G. Pagès. Multistep Richardson-Romberg extrapolation: remarks on variance and complexity. *Monte Carlo Applications*, 13(1):37–70, 2007.

[140] G. Pagès. *Introduction to numerical probability for finance*, 2016. Available at `http://www.proba.jussieu.fr/dw/lib/exe/fetch.php?media=users:pages:probnum_gilp_pf16.pdf`, 354p.

[141] G. Pagès, O. Pironneau, and G. Sall. The parareal algorithm for American options. *Comptes Rendus Mathématiques de l'Académie des Sciences*, 354(11):1132–1138, 2016.

[142] G. Pagès, O. Pironneau, and G. Sall. Second senstivities in quantitative finance. *ESAIM, Proceedings and Surveys*, 1:1–10, 2017.

[143] G. Pagès, O. Pironneau, and G. Sall. Vibrato and automatic differentiation for high order derivatives and sensitivities of financial options. *Risk Journal, The Journal of Computational Finance*, 2017.

[144] G. Pagès and J. Printems. Functional quantization based for numerics with an application to option pricing. *Monte Carlo Methods and Applications*, 11(11):407–446, 2005.

[145] G. Pagès and B. Wilbertz. GPGPUs in computational finance: massive parallel computing for American style options. *Concurrency and Computation: Practice and Experience*, 24(18):837–848, 2012.

[146] E. Pardoux and S. Peng. Backward stochastic differential equations and quasilinear parabolic partial differential equations. *Lecture Notes in Computation and Information Sciences*, 76:200–217, 1992.

[147] E. Pardoux and A. Răşcanu. *Stochastic Differential Equations, Backward SDEs, Partial Differential Equations*, volume 69 of *Stochastic Modelling and Applied Probability.* Springer-Verlag, New York, United States of America, 2014. xvii+663 pp. ISBN 978-3-319-05713-2.

[148] P. Pardoux and S. Peng. Adapted solution of a backward stochastic differential equation. *Systems and Control Letters*, 14:55–61, 1990.

[149] S. Peng. A generalized dynamic programming principle and Hamilton-Jacobi-bellman equation. *Stochastics*, 38:119–134, 1992.

[150] S. Peng. Backward stochastic differential equation and its application in optimal control. *Applied Mathematics and Optimization*, 27:125–144, 1993.

[151] Y. Peng, B. Gong, H. Liu, and B. Dai. Option pircing on the GPU with backward stochastic differential equation. *Algorithms and Programming*, pages 19–23, 2011.

[152] Y. Peng, H. Liu, S. Yang, and B. Gong. Parallel algorithm for BSDEs based high dimensional american option pricing on the GPU. *Journal of Computational Information Systems*, 10(2):763–771, 2014.

[153] H. Pham. *Continuous-time Stochastic Control and Optimization with Financial Applications*, volume 61 of *Stochastid Modelling and Applied Probability.* Springer-Verlag, Berlin, Germany, 2009. xvii+233 pp., ISBN 978-3-540-89500-8.

[154] O. Pironneau. *Automatic differentiation for financial engineering.* Université Pierre et Marie Curie, Paris VI, 2008.

[155] J. Pitman and M. Yor. A decomposition of Bessel bridges. *Z. Wahrsch. Verw. Gebiete*, 59(4):425–452, 1982.

[156] M. C. Quenez. *Méthodes de contrôle stochastique en Finance.* PhD thesis, Université Pierre et Marie curie, Paris, France, 1993.

[157] R. Rebonato. *Interest rate options models: understanding, analysing and using models for exotic interest rate options*. Wiley series in financial engineering. John Wiley& Sons Ltd, Chichester, England, 1998. xix+427 pp., ISBN 978-0-471-97958-6.

[158] R. Rebonato. *Modern pricing of interest rate derivatives: the LIBOR market model and beyond*. Princeton & Oxford. Princeton University Press, New Jersey, United States, 2012. xvii+467 pp., ISBN 0-691-08973-6.

[159] N. Reed. Square deal. *Risk*, 7(6), 1994.

[160] N. Reed. The power and the glory. *Risk*, 8(8), 1995.

[161] R. Reed. *An Introduction to Intel Xeon Phi Coprocessor*, 2012. Available at `https://software.intel.com/sites/default/files/`.

[162] A. Reghai, A. Langnau, and A. Ben Haj Yedder. Decoding the american vanilla prices. Available at: `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2347910`, 2013.

[163] M. Reiman and A. Weiss. Sensitivity analysis for simulations via likelihood ratios. *Operations Research*, 37:830–844, 1989.

[164] E. Reiner and M. Rubinstein. Breaking down the barriers. *Risk*, 4:28–35, 1991.

[165] C. H. Rhee and P. W. Glynn. Unbiased estimatin with square root convergence for SDE models. *Operations Research*, 63(5):1026–1043, 2015.

[166] M. Rubinstein. Options for the undecided. *Risk Magazine, Black Scholes to Black Holes*, pages 187–189, 1992.

[167] R. Rubinstein. Sensitivity analysis and performance extrapolation for computer simulation models. *Operations Research*, 37:72–81, 1989.

[168] M. J. Ruijter and C. W. Oosterlee. A fourier cosine method for an efficient computation of solution of BSDEs. *SIAM, Journal of Scientific computation*, 37(2):859–889, 2015.

[169] M. Sabahl. *A Guide to (Auto)-Vectorization with Intel C++ Compilers*, 2013. iii+38 pp., Available at `https://software.intel.com/sites/default/files/8c/a9/CompilerAutovectorizationGuide.pdf`.

[170] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Parallel Programming. Addison-Wesley Professional, Boston, United States of America, 1 edition, 2010. xx+290 pp., ISBN 978-0-131-38768-3.

[171] S. Delage Santacreu. *Introduction au Calcul parallèle avec la bibliothèque MPI (Message Passing Interface)*, 2008. Available at `https://www.ljll.math.upmc.fr/SGI/pdf/pau-mpi.pdf`.

[172] M. Scarpino. *OpenCL in Action: How to Accelerate Graphics and Computations*. Parallel Programming. Manning Publications, Shelter Island, New York, United States of America, 1 edition, 2011. xxii+426 pp., ISBN 978-1-617-29017-6.

[173] J. Sidenius. Credit valuation adjustment in derivatives trading. Available at `http://www.cbs.dk/files/cbs.dk/fric_cva_js.pdf`, 2013.

[174] A. L. Speight. A multilevel approach to control variates. *Journal of Computational Finance*, 12(4):3–27, 2009.

[175] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Review*, 40(1):110–112, 1998.

[176] R. Suri and M. Zazanis. Pertubation analysis gives strongly consistent sensitivity estimates for the M/G/1 queue. *Management Science*, 34:39–64, 1988.

[177] R. Tay. *OpenCL Parallel Programming Development Cookbook*. Graphics & Design. Packt Publishing, Birmingham, United Kingdom, 1 edition, 2011. xiii+303 pp., ISBN 978-1-849-69452-0.

[178] J. N. Tsitsiklis and B. Van Roy. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks*, 12(4):694–703, 2001.

[179] J. A. Varela, C. Brugger, S. Tang, N. Wehn, and R. Korn. Pricing high-dimensional American options on hybrid CPU/FPGA systems. *FPGA Based Accelerators for Financial Applications*, pages 143–166, 2017.

[180] J. M. Villalobos. *Monte Carlo methods for forward backward stochastic differential equations in High dimensions*. PhD thesis, University of Souther California, California, United States of America, 2007.

[181] J. Wan, K. Lai, A. Kolkiewicz, and K. Tan. A parallel quasi-Monte Carlo approach to pricing American options on multiple assets. *International Journal of High Performance Computing and Networking*, 4:321–330, 2006.

[182] N. Wiener. *The Human Use of Human Beings: Cybernetics and Society*. Free Association Books, London, England, 1954. xxiv+199 pp., ISBN 1-85343-075-7.

[183] J. Zhang. *Some fine properties of backward stochastic differential equations*. PhD thesis, Perdue University, Indiana, United States of America, 2001.

[184] J. Zhang. A numerical scheme for BSDEs. *The Annals of Applied Probability*, 14:459–488, 2004.

[185] S. H. Zhu and M. Pykhin. A guide to modeling counterparty credit risk. *GARP Risk Review*, 2007.

Index