Analysis and numerical methods for stochastic volatility models in valuation of financial derivatives

Autor: José Germán López Salas

Tese de doutoramento UDC / 2016

Director: Carlos Vázquez Cendón

Programa de Doctorado Métodos Matemáticos y Simulación Numérica en Ingeniería y Ciencias Aplicadas

UNIVERSIDADE DA CORUÑA

UNIVERSIDADE DA CORUÑA

PhD. Thesis

Analysis and numerical methods for stochastic volatility models in valuation of financial derivatives

AUTOR:

José Germán López Salas

DIRECTOR:

Carlos Vázquez Cendón

TESE PRESENTADA PARA A OBTENCIÓN DO TÍTULO DE DOUTOR NA UNIVERSIDADE DA CORUÑA

DEPARTAMENTO DE MATEMÁTICAS

FACULTADE DE INFORMÁTICA, A CORUÑA (SPAIN)

SETEMBRO, 2016

UNIVERSIDADE DA CORUÑA

El abajo firmante hace constar que es director de la Tesis Doctoral titulada "**Analysis and numerical methods for stochastic volatility models in valuation of financial derivatives**" desarrollada por José Germán López Salas, cuya firma también se incluye, dentro del programa de doctorado "**Métodos Matemáticos y Simulación Numérica en Ingeniería y Ciencias Aplicadas**" en el Departamento de Matemáticas (Universidade da Coruña), dando su consentimiento para su presentación y posterior defensa.

The undersigned hereby certifies that he is supervisor of the Thesis entitled "**Analysis and numerical methods for stochastic volatility models in valuation of financial derivatives**" developed by José Germán López Salas, whose signature is also included, inside the Ph.D Program "**Mathematical Modelling and Numerical Methods in Applied Sciences and Engineering**" at the Department of Mathematics (University of A Coruña), consenting to its presentation and posterior defense.

September, 2016

Director:

Carlos Vázquez Cendón

Doctorando:

José Germán López Salas

Funding

This research has been partially funded by the following projects:

- FPU grant whose call was published by resolution of April 25, 2012 from Ministerio de Educación, Cultura y Deporte. Grant holder reference AP2012-4975.
- Galician grant for Ph.D. students from Xunta de Galicia (05.12.2012 - 21.03.2013).
- I-Math Consolider Project, Reference COMP-C6-0393, with Fundación Cesga.
- Project MTM2010-21135-C02-01 from Ministerio de Ciencia e Innovación.
- Grant CN2011/004 Grupos de Referencia Competitiva by Xunta de Galicia.
- Grant GRC2014/044 Grupos de Referencia Competitiva by Xunta de Galicia.
- Project HIPeCA-High performance calibration and computation in finance, Reference 57049700 2014 DAAD, from German Federal Ministry of Education and Research.
- Project MTM2013-47800-C2-1-P from Ministerio de Economía y Competitividad.
- FEDER funds.

A mi familia.

Agradecimientos

A la par del esfuerzo personal para la realización de esta tesis doctoral, quiero reconocer la ayuda de cuantos de me aportasteis vuestro apoyo y supisteis compartir vuestros conocimientos.

Comienzo teniendo presentes la ilusión y confianza que mis padres pusieron en ampliar mi formación, junto a los ejemplos de perseverancia y constancia de mi hermana *Fani* y de *Carmen González*.

Amplia gratitud merece *Carlos Vázquez* como profesor, transmisor de conocimientos e impulsor de nuevas iniciativas. Me siento afortunado de haber trabajado con él, su apoyo ha sido fundamental para lograr culminar este proyecto.

La cordial y grata colaboración de *José Antonio García* y *Ana M. Ferreiro* supuso una enriquecedora contribución al desarrollo de este trabajo.

Le agradezco al laboratorio CMAP de la École Polytechnique su acogida. Quiero destacar la ayuda ofrecida durante mi estancia por parte del profesor *Emmanuel Gobet* y de *Plamen Turkedjiev*.

También quiero dar las gracias a *José Luis Fernández* y a *María R. Nogueiras* por su asesoramiento, aportaciones y ánimos a lo largo de este período.

Durante los dos últimos años he tenido la oportunidad de colaborar en la docencia de Cálculo en la Facultad de Informática. Quiero agradecer a los profesores de la asignatura toda su ayuda y muy especialmente a *Íñigo Arregui* y a *Teresa Iglesias*.

Quiero reconocer la disponibilidad de *José Antonio García* y *Álvaro Leitao* a tenderme la mano en todo lo posible, atesoro gratos recuerdos de nuestras innumerables discusiones sobre GPUs y HPC en general.

Álvaro, *Dani*, *Paula*, *Marta*, *Alejandro*, *Ana*, *María*, *Aldana* y *Carmen*, mis compañeros de laboratorio, merecen un agradecimiento cercano por toda la comprensión y apoyo que hemos compartido.

A Coruña, 2016.

Table of Contents

Abstract **xv**

Resumen **xvii**

Resumo **xix**

Introduction **1**

I Stochastic volatility models **9**

1 Simulated Annealing **17**
1.1 Introduction 17
1.2 Simulated annealing 20
1.2.1 Sequential Simulated Annealing 20
1.2.2 Parallel Simulated Annealing 24
1.3 Implementation on GPUs 27
1.3.1 General-Purpose Computing on Graphics Processing Units (GPGPU) 27
1.3.2 Nvidia GPUs, many core computing 29
1.3.3 Notes on the CUDA implementation 33
1.4 Numerical experiments: academic tests 36
1.4.1 Analysis of a sample test problem: Normalized Schwefel function 36
1.4.2 The set of performed tests 44
1.5 Conclusions 45

2 SABR models for equity **49**
2.1 Introduction 49
2.2 The SABR model 52
2.2.1 Static SABR model 53

2.2.2 Dynamic SABR model and the choice of the functional parameters 54
2.3 Calibration of the SABR model . . . 56
2.4 Pricing with Monte Carlo using GPUs . . . 59
2.5 Calibration of the parameters using GPUs . . . 62
2.5.1 Calibration with Technique I . . . 62
2.5.2 Calibration with Technique II . . . 63
2.6 Numerical results . . . 64
2.6.1 Pricing European options . . . 65
2.6.2 Calibration . . . 66
2.6.3 Pricing a cliquet option . . . 76
2.7 Conclusions . . . 78

3 SABR/LIBOR market models: Monte Carlo approach 79
3.1 Introduction . . . 79
3.2 SABR/LIBOR market models . . . 84
3.2.1 Hagan model . . . 84
3.2.2 Mercurio & Morini model . . . 86
3.2.3 Rebonato model . . . 87
3.3 Model calibration . . . 88
3.4 Numerical results . . . 90
3.4.1 Hagan model . . . 94
3.4.2 Mercurio & Morini model . . . 95
3.4.3 Rebonato model . . . 96
3.5 Conclusions . . . 115

4 SABR/LIBOR market models: PDE approach 117
4.1 Introduction . . . 117
4.2 Derivation of the PDE from the stochastic processes . . . 118
4.3 Finite Difference Method . . . 123
4.3.1 Boundary conditions . . . 127
4.3.2 Numerical results . . . 128
4.4 Sparse grids and the combination technique . . . 133
4.4.1 Sparse grids . . . 134
4.4.2 Combination technique . . . 141
4.4.3 Numerical results . . . 143

II BSDEs 149

5 Backward Stochastic Differential Equations 155
5.1 Introduction . . . 155

5.2 Mathematical framework and basic properties 159
5.3 Stratified algorithm and convergence results 164
5.3.1 Algorithm 164
5.3.2 Error analysis 167
5.3.3 Proof of Theorem 5.3.5 169
5.4 GPU implementation 174
5.4.1 Explicit solutions to **OLS** in Algorithm 4 175
5.4.2 Pseudo-algorithms for GPU 176
5.4.3 Theoretical complexity analysis 178
5.5 Numerical experiments 181
5.5.1 Model, stratification, and performance benchmark 181
5.5.2 CPU and GPU performance 182

A Test functions for the Simulated Annealing **191**

B SABR equity **201**
B.1 Expression of implied volatility in the general case 201
B.2 Market data 203

C BSDEs **207**
C.1 Proof of Proposition 5.2.1 207
C.2 Stability results for discrete BSDE 209

Conclusions **211**

Resumen extenso **215**

Resumo extenso **229**

Bibliography **242**

List of Tables

1.1 Error of the solution obtained by the algorithm, both in the value of the function at the minimum (columns $|f_a - f_r|$, where f_a is the objective function value found by the algorithm and f_r is the exact function value in the real minimum) and in the minimum (columns Relative error, measured in $\|\cdot\|_2$). 37

1.2 Performance of CUDA version vs. sequential version with one CPU core for different number of parameters. 38

1.3 Behavior of the errors when increasing the number of launched threads. Tests were performed with $n = 16$, $T_0 = 5$, $T_{min} = 0.5$, $\varphi = 0.7$, $L = 5$. 42

1.4 Behavior of the speedup when increasing the number of launched threads. Tests were performed with $T_0 = 1000$, $T_{min} = 0.01$, $\varphi = 0.99$, $L = 100$. 42

1.5 Behavior of the speedup when increasing L. These tests were performed with the following configuration of Simulated Annealing, $T_0 = 1000$, $T_{min} = 0.01$, $\varphi = 0.99$, $b = 256$, $g = 64$. 43

1.6 Behavior of the speedup when increasing the number of function evaluations. . . 43

1.7 Computational times in seconds and relative quadratic errors with single and double-precision for the next Simulated Annealing configuration: $n = 16$, $T_0 = 1000$, $T_{min} = 0.01$, $\varphi = 0.99$, $b = 256$, $g = 64$. 44

1.8 Set of test problems, where first column indicates the assigned reference to display results. 46

1.9 Results for the test problem suite. In the column Error we indicate the relative error in $||\cdot||_2$ when the location of the minimum is non zero, otherwise the absolute error is presented. Cells with '-' mark correspond to cases in which the exact minima are unknown. 47

1.10 Results of the hybrid algorithm. The first part shows the results of the annealing algorithm. The second one shows the results of the Nelder-Mead algorithm starting at the point at which the annealing algorithm was stopped prematurely. 47

2.1 Pricing results for European options. `RE` denotes the relative error with respect to the reference value 225.887329. 65

2.2 Pricing European options. Execution times for CPU and GPU calibration (in seconds), considering single and double precision with $\Delta t = 1/250$. 66

2.3 Pricing with `DSabr_II` model in GPU. Influence of the number of strikes in computational times (time in seconds), $\Delta t = 1/250$. We consider 1, 5 and 41 strikes, with values K (in % of S_0) of $\{100\}$, $\{96, 98, 100, 102, 104\}$ and $\{80, 81 \ldots, 119, 120\}$, respectively. 66

2.4 EURO STOXX 50. `SSabr` model: Calibrated parameters for each maturity. 67

2.5 EURO STOXX 50. `SSabr` model: Performance of OpenMP vs. GPU versions, in single precision for $T = 24$ months. 68

2.6 EURO STOXX 50. `DSabr_I` model: Calibrated parameters. 68

2.7 EURO STOXX 50. `DSabr_I` model: σ_{market} vs. σ_{model}. 69

2.8 EURO STOXX 50. `DSabr_I` model: Performance of OpenMP vs. GPU versions, in single precision. 70

2.9 EURO STOXX 50. `DSabr_II` model: Calibrated parameters. 70

2.10 EURO STOXX 50. `DSabr_II` model: V_{market} vs. V_{model}. 70

2.11 EURUSD. `SSabr` model: Calibrated parameters for each maturity. . . 71

2.12 EURUSD. `SSabr` model: Performance of OpenMP vs. GPU versions, in single precision for $T = 24$ months. 72

2.13 EURUSD. DSabr_I model: Calibrated parameters. 73
2.14 EURUSD. DSabr_I model: σ_{market} vs. σ_{model}. 73
2.15 EURUSD. DSabr_I model: Performance of OpenMP vs. GPU versions, in single precision. 74
2.16 EURUSD. DSabr_II model: Calibrated parameters. 74
2.17 EURUSD. DSabr_II model: V_{market} vs. V_{model}. 75

3.1 Execution times (in seconds) and speedups in the pricing of caplets with Monte Carlo and using single precision (Hagan model). 90
3.2 Discount factor curve. 91
3.3 Smiles of forward rates. Fixing dates (first column) and moneyness (first row). 92
3.4 Smiles of swap rates. Length of the underlying swaps (first column), swaptions maturities (second column) and moneyness (first row). . . . 93
3.5 Hagan model, calibration to caplets with SABR formula (3.4): calibrated parameters. 94
3.6 Hagan model, calibration to caplets, σ_{market} vs. σ_{model}. 98
3.7 Hagan model, calibration to swaptions, S_{Black} vs. S_{MC}, prices in %. . 100
3.8 Mercurio & Morini model, calibration to caplets with SABR formula (3.4): calibrated parameters. 103
3.9 Mercurio & Morini model, calibration to caplets, σ_{market} vs. σ_{model}. . 104
3.10 Mercurio & Morini model, calibration to swaptions, S_{Black} vs. S_{MC}, prices in %. 106
3.11 Rebonato model, calibration to caplets with SABR formula (3.4): calibrated parameters. 109
3.12 Rebonato model, calibration to caplets, σ_{market} vs. σ_{model}. 110
3.13 Rebonato model, calibration to swaptions, S_{Black} vs. S_{MC}, prices in %. 112
3.14 Mean relative errors of the three models. 115

4.1 Specification of the interest rate model. 129

4.2 Market data used in pricing. Data taken from 27th July 2004. 130

4.3 Convergence of the PDE solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0$, $V(0) = 1$, $\beta = 1$, 128 time steps. Exact solution, 0.659096 basis points. 131

4.4 Convergence of Monte Carlo solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0$, $V(0) = 1$, $\beta = 1$, 128 time steps. Exact solution, 0.659096 basis points. 131

4.5 Convergence of the PDE solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 1.657662 basis points. . . . 132

4.6 Convergence of the PDE solution in basis points for 2 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 4.652644 basis points. . . . 132

4.7 Convergence of the PDE solution in basis points for 3 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 8.177764 basis points. . . . 133

4.8 Convergence of the PDE solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0$, $V(0) = 1$, $\beta = 1$, 128 time steps. Exact solution, 0.659096 basis points. 145

4.9 Convergence of the PDE solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 1.657662 basis points. . . . 145

4.10 Convergence of the PDE solution in basis points for 2 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 4.652644 basis points. . . . 145

4.11 Convergence of the PDE solution in basis points for 3 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 8.177764 basis points. . . . 146

4.12 Convergence of the PDE solution in basis points for 4 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 12.288113 basis points. . . 146
4.13 Convergence of the PDE solution in basis points for 5 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 16.903377 basis points. . . 146
4.14 Convergence of the PDE solution in basis points for 6 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 2 time steps. Monte Carlo value using 10^7 paths, 21.979879 basis points. 147
4.15 Convergence of the PDE solution in basis points for 8 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 2 time steps. Monte Carlo value using 10^7 paths, 27.222777 basis points. 147
4.16 Convergence of the PDE solution in basis points for 8 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 2 time steps. Monte Carlo value using 10^7 paths, 32.553432 basis points. 147

5.1 Comparison of numerical parameters with or without stratified sampling, as a function of N. 180
5.2 Comparison of memory requirement as a function of N. 181
5.3 **LP0** local polynomials, $d = 4$, #C=$\left\lfloor 4\sqrt{N} \right\rfloor$, $M = N^2$. 184
5.4 **LP0** local polynomials, $d = 6$, #C=$\left\lfloor \sqrt{N} \right\rfloor$, $M = N^2$. 185
5.5 **LP0** local polynomials, $d = 6$, #C=$\left\lfloor 2\sqrt{N} \right\rfloor$, $M = N^2$. 185
5.6 **LP0** local polynomials, $d = 11$, #C=$\left\lfloor \sqrt{N} \right\rfloor$, $M = N^2$. 186
5.7 **LP1** local polynomials, $d = 4$, #C=$\left\lfloor 3\sqrt{d\sqrt{N}} - 5 \right\rfloor$, $M = (d+1)N^2$. . 187
5.8 **LP1** local polynomials, $d = 6$, #C=$\left\lfloor 1.5\sqrt{d\sqrt{N}} - 3 \right\rfloor$, $M = (d+1)N^2$. 188
5.9 **LP1** local polynomials, $d = 11$. 188
5.10 **LP1** local polynomials, $d = 12$. 189
5.11 **LP1** local polynomials, $d = 13$. 189
5.12 **LP1** local polynomials, $d = 14$. 189

5.13 **LP1** local polynomials, $d = 15, \ldots, 19$, $\Delta_t = 0.2$, $\texttt{\#C} = 2$. 189

B.1 EURO STOXX 50 (Dec. 2011). Spot value $S_0 = 2311.1$ €. Interest rates and dividend yields. 203

B.2 EURO STOXX 50 (Dec. 2011). Implied volatilities for each maturity with different strikes K (% of the spot S_0). 204

B.3 EUR/USD (Dec. 2011). Spot value $S_0 = 1.2939$ US dollars. Interest rates and dividend yields. 204

B.4 EUR/USD (Dec. 2011). Implied volatilities for each maturity with different strikes K. 205

List of Figures

1 Shape of a typical volatility smile. 13

1.1 Sketch of the asynchronous parallel algorithm. 26
1.2 Sketch of the synchronous parallel algorithm. 27
1.3 Nvidia GPU hardware structure. 29
1.4 For the three versions V0, V1 and V2, convergence rate for runs with $n = 8$ and 16. 39
1.5 For the three versions V0, V1 and V2, convergence rate for runs with $n = 32$ and 64. 40
1.6 For the three versions V0, V1 and V2, convergence rate for runs with $n = 128$ and 256. 41

2.1 Sketch of the parallel SA algorithm using two GPUs and OpenMP. . 61
2.2 EURO STOXX 50. `SSabr` model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right). 67
2.3 EURO STOXX 50. `DSabr_I` model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right). 69
2.4 EURO STOXX 50. `DSabr_II` model: V_{model} vs. V_{market} for the whole prices surface. Maturities: 3 and 12 months (left), 6 and 24 months (right). 71

2.5 EURUSD. `SSabr` model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right). 72
2.6 EURUSD. `DSabr_I` model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right). 73
2.7 EURUSD. `DSabr_II` model: V_{model} vs. V_{market} for the whole prices surface. Maturities: 3 and 12 months (left), 6 and 24 months (right). 74
2.8 EURO STOXX 50. `DSabr_I` model for pricing European options. Prices (left) and relative errors (right). 76
2.9 EURUSD. `DSabr_I` model for pricing European options. Prices (left) and relative errors (right). 77

3.1 Sketch of the parallel SA using OpenMP and considering a Monte Carlo method in the cost function. 91
3.2 Hagan model, σ_{market} vs. σ_{model}, smiles of $F_1, \ldots, F_{13}$. 99
3.3 Hagan model, calibration to swaptions, S_{Black} vs. S_{MC}, part I. 101
3.4 Hagan model, calibration to swaptions, S_{Black} vs. S_{MC}, part II. . . . 102
3.5 Mercurio & Morini model, σ_{market} vs. σ_{model}, smiles of $F_1, \ldots, F_{13}$. . . 105
3.6 Mercurio & Morini model, calibration to swaptions, S_{Black} vs. S_{MC}, part I. 107
3.7 Mercurio & Morini model, calibration to swaptions, S_{Black} vs. S_{MC}, part II. 108
3.8 Rebonato model, σ_{market} vs. σ_{model}, smiles of $F_1, \ldots, F_{13}$. 111
3.9 Rebonato model, calibration to swaptions, S_{Black} vs. S_{MC}, part I. . . 113
3.10 Rebonato model, calibration to swaptions, S_{Black} vs. S_{MC}, part II. . . 114

4.1 Two-dimensional full grid hierarchy up to level $n = 4$. 135
4.2 Two-dimensional sparse grid hierarchy up to level $n = 4$. 136
4.3 Three dimensional sparse grids for levels $n = 5, 6, 7$ and 8. 138
4.4 Two dimensional sparse grids for levels $n = 5, \ldots, 10$. 139
4.5 Combination technique with level $n = 4$ in two dimensions. 140

1 Forma común de la sonrisa de volatilidad. 218
1 Forma común do sorriso de volatilidade. 232

Abstract

The main objective of this thesis concerns to the study of the SABR stochastic volatility model for the underlyings (equity or interest rates) in order to price several market derivatives. When dealing with interest rate derivatives the SABR model is joined with the *LIBOR market model* (LMM) which is the most popular interest rate model in our days. In order to price derivatives we take advantage not only of Monte Carlo algorithms but also of the numerical resolution of the partial differential equations (PDEs) associated with these models. The PDEs related to SABR/LIBOR market models are high dimensional in space. In order to cope with the curse of dimensionality we will take advantage of *sparse grids*. Furthermore, a detailed discussion about the calibration of the parameters of these models to market prices is included. To this end the *Simulated Annealing* global stochastic minimization algorithm is proposed.

The above mentioned algorithms involve a high computational cost. In order to price derivatives and calibrate the models as soon as possible we will make use of high performance computing (HPC) techniques (multicomputers, multiprocessors and GPUs).

Finally, we design a novel algorithm based on *Least-Squares Monte Carlo* (LSMC) in order to approximate the solution of *Backward Stochastic Differential Equations* (BSDEs).

Resumen

El objetivo principal de la tesis se centra en el estudio del modelo de volatilidad estocástica SABR para los subyacentes (activos o tipos de interés) con vista a la valoración de diferentes productos derivados. En el caso de los derivados de tipos de interés, el modelo SABR se combina con el modelo de mercado de tipos de interés más popular en estos momentos, el *LIBOR market model* (LMM). Los métodos numéricos de valoración son fundamentalmente de tipo Monte Carlo y la resolución numérica de los modelos de ecuaciones en derivadas parciales (EDPs) correspondientes. Las EDPs asociadas a modelos SABR/LIBOR tienen alta dimensión en espacio, por lo que se estudian técnicas de *sparse grid* para vencer la maldición de la dimensión. Además, se discute detalladamente cómo calibrar los parámetros de los modelos a las cotizaciones de mercado, para lo cual se propone el uso del algoritmo de optimización global estocástica *Simulated Annealing*.

Los algoritmos citados tienen un alto coste computacional. Con el objetivo de que tanto las valoraciones como las calibraciones se hagan en el menor tiempo posible se emplean diferentes técnicas de computación de altas prestaciones (multicomputadores, multiprocesadores y GPUs.)

Finalmente se diseña un nuevo algoritmo basado en *Least-Squares Monte Carlo* (LSMC) para aproximar la solución de *Backward Stochastic Differential Equations* (BSDEs).

Resumo

O obxectivo principal da tese céntrase no estudo do modelo de volatilidade estocástica SABR para os subxacentes (activos ou tipos de xuro) con vista á valoración de diferentes produtos derivados. No caso dos derivados de tipos de xuro, o modelo SABR combínase co modelo de mercado de tipos de xuro máis popular nestos momentos, o *LIBOR market model* (LMM). Os métodos numéricos de valoración son fundamentalmente de tipo Monte Carlo e a resolución numérica dos modelos de ecuacións en derivadas parciais (EDPs) correspondentes. As EDPs asociadas aos modelos SABR/LIBOR teñen alta dimensión en espazo, polo que se estudan técnicas de *sparse grid* para vencer a maldición da dimensión. Ademais, discútese detalladamente como calibrar os parámetros dos modelos ás cotizacións de mercado, para o cal se propón o emprego do algoritmo de optimización global estocástica *Simulated Annealing*.

Os algoritmos citados teñen un alto custo computacional. Co obxectivo de que tanto as valoracións como as calibracións se fagan no menor tempo posible empréganse diferentes técnicas de computación de altas prestacións (multicomputadores, multiprocesadores e GPUs.)

Finalmente deséñase un novo algoritmo baseado en *Least-Squares Monte Carlo* (LSMC) para aproximar a solución de *Backward Stochastic Differential Equations* (BSDEs).

Introduction

In this thesis we analyze the valuation of financial derivatives using some mathematical models. Our goal is to illustrate the use of these models with an emphasis on the implementation and calibration.

A financial derivative is a contract whose value depends on one or more assets, called underlying assets. Typically the underlying asset is a stock (or *equity*), a currency exchange rate, the market price of commodities (such us oil or wheat) or a bond (*interest rate*). Among the large variety of financial derivatives being traded nowadays, an option is the simplest example. An option is a contract that gives the right (but not the obligation) to its holder to buy or sell some amount of the underlying asset at a future date, for an agreed price. A *call* option gives the right to buy, whilst a *put* option gives the right to sell. An option is called *European* if the right to buy or sell can be exercised only at maturity, and it is called *American* if it can be exercised at any time before maturity. Call and put options are the basic derivative instruments and for this reason they are often called *plain vanilla* options. However, there exists a great amount of derivatives, usually known as *exotic*, having very complicated structures. Pricing these financial derivatives is non-trivial because the future evolutions of the prices of the underlying assets are not known at present. The price of the derivative is the *premium* that the buyer of the derivative has to pay at the initial time to get the right guaranteed by the contract. The main two reasons for using financial derivatives are hedging the risk and speculation purposes.

The starting point of trading financial derivatives in organized markets was on 26th April 1973 in The Chicago Board Options Exchange (CBOE). Initially there were just

calls on 16 stocks. Puts weren't even introduced until 1977. Also in 1973, Merton [104] and Black and Scholes [13] published the basic building blocks of derivatives theory, delta hedging and no arbitrage theory. Using these strategies, the authors obtained the celebrated Black-Scholes partial differential equation (PDE) and the Black-Scholes formula for European plain vanilla options. Despite the huge popularity of Black-Scholes formula, after the stock market crash of October 1987, it was clear that the fact of assuming a constant volatility for the underlying asset leads to a significant mispricing of options. It is well known that the market prices of European options on the same underlying asset have different Black-Scholes implied volatilities that vary with strike and maturity, this is known as *volatility smile*. Generally, we can say that market prices tend to give more value (greater implied volatility) to the extreme cases in or out of the money. This reflects that some situations in the market are perceived as more risky, in particular the case of extreme falls or rises of the quotations of the underlying asset. Modelling the volatility as a random variable itself is a natural way to overcome the problems of assuming constant volatility. These models are called stochastic volatility models and are useful because they are able to fit market volatility smiles. Moreover, unlike alternative models that can recover the smile (such as local volatility models, for example), stochastic volatility models assume realistic dynamics for the underlying.

Among the different stochastic volatility models proposed in the literature, the SABR model proposed by Hagan, Kumar, Lesniewski and Woodward [67] in the year 2002 stands out for becoming the market standard to reproduce the price of European options. Although local volatility models can fit by construction the volatility smile of the market even better than the SABR model, these models predict unrealistic evolutions for the underlying. In fact, SABR model tells us about changes in option prices versus *strike*, as opposed to local volatility models which state changes in option prices as the underlying moves.

Among the large variety of financial derivatives being traded nowadays, when the underlying is a particular interest rate or a set of them, the class of interest

rate derivatives arises. In this thesis we will mainly consider bonds, caplets, caps, swaps and swaptions. A bond is a contract that periodically pays coupons depending on certain floating rates. A caplet is a call option that pays the positive difference between a floating rate and a fixed one (strike). A cap contract is a set of caplets associated with several maturity dates. A swap is a contract that exchanges two different interest rates. A swaption is an option giving the right to enter in a swap at a given future time. For a detailed description about interest rate derivatives we refer the reader to the book of Brigo and Mercurio [19]. Unlike in the case of equity markets, in interest rate markets the long term of contracts and the behaviour of the involved interest rates motivated the consideration of stochastic interest rate models. These models can be mainly classified into two categories, short rate models and market models.

Short rate models specify one-factor dynamics for the evolution of just one short rate, which determines the future evolution of the entire yield curve. The popular models of Vasicek (1977) [136] and Cox, Ingersoll and Ross (1985) [29] lie in this category. The main drawback of short rate models is the impossibility of calibrate their parameters to the initial curve of discount factors, specially for those models that are not analytically tractable.

In 1986, Ho and Lee [75] proposed the first alternative to short rate models, which was the initial work on market models. They modelled the evolution of the entire yield curve in a binomial tree framework. Later, in 1992, Heath, Jarrow and Morton [69] translate in continuous time the basic assumption of the Ho and Lee model. Their HJM model became the standard framework for interest rates in the early 1990s. However, the main problem of HJM model was its incompatibility with the market's use of the Black caplet and Black swaption formula, i.e. HJM model broke out when the instantaneous forward rates were modelled as lognormal.

In order to overcome the main obstacle of HJM model, in 1999, Miltersen, Sandmann and Sondermann [108] published a PDE method to derive the Black caplet formula within the arbitrage free HJM framework. Their main contribution was

modelling the forward rates as lognormal, but under the forward measure at the end of their interval. Taking into account this realization, Brace, Gatarek and Musiela [16] derived the so-called BGM model, also known as the LIBOR market model (LMM), because it models forward LIBOR rates through a lognormal distribution under the relevant measures. Jamshidian (1997) [79] also contributed significantly to its development. The most important benchmark interest rates are the London Interbank Offered Rates or LIBORs, which are calculated daily through an average of rates offered by banks in London. The LMM has become the most popular interest rate model. The main reason is the agreement between this model and Black's formulas. In fact, the LIBOR market model prices caps with Black's formula, which is the standard formula employed in the cap market. Besides, the Swap market model (SMM) prices swaptions with Black's swaption formula, which again is the standard formula employed in the swaption market. Taking into account that caps and swaptions are the most traded interest rate derivatives, it is very important for a market model to be compatible with such market formulas. In addition, the parameters of these models can be easily calibrated to market prices using liquid products.

The standard LIBOR market model considers constant volatilities for the forward rates. However, this is a very limited hypothesis since it is impossible to reproduce market volatility smiles. The SABR model can not be used to price derivatives whose payoff depends on several forward rates. In fact, SABR model works in the terminal measure, under which both the forward rate and its volatility are martingales. This can always be done if we work with one forward rate in isolation at a time. Under this same measure, however, the process for another forward rate and for its volatility would not be driftless. In order to allow LMM to fit market volatility smiles, different extensions of the LMM that incorporate the volatility smile by means of the SABR model were proposed. These models are known as SABR/LIBOR market models. In this work we will study the models proposed by Hagan [68], Mercurio and Morini [103] and Rebonato [122]. These models involve a high number of parameters. Calibrating these parameters to market data becomes a relevant target in practice. In this

thesis we address this calibration taking advantage of High Performance Computing techniques in order to optimize execution times. Undoubtedly, this issue is crucial in the *real time* financial world.

From the numerical point of view, in the LIBOR market model setting the pricing of interest rate derivatives in mainly carried out using Monte Carlo simulation [49]. Nevertheless, taking into account that Monte Carlo simulation could involve excessively long computational times, in this work we also address for first time in the literature the alternative pricing approach offered by PDEs. Thus, we pose the original PDE formulations associated to the three SABR/LIBOR models proposed by Hagan, Mercurio & Morini and Rebonato. Nevertheless, the PDEs associated to SABR/LIBOR market models are high dimensional in space. Therefore, traditional full grid methods, like standard finite difference or finite elements, will not be able to price derivatives over more than three or four underlying interest rates, due to the so-called curse of dimensionality [7]. In order to overcome the curse of dimensionality, the sparse grid combination technique first proposed by Zenger, Griebel and Schneider [63] will be analyzed.

In the second part of the thesis we design a novel algorithm based on Least-Squares Monte Carlo (LSMC) in order to approximate the (Y, Z) components of the solution to the decoupled forward-backward stochastic differential equation (BSDE)

$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)\mathrm{d}s - \int_t^T Z_s \mathrm{d}W_s,$$
$$X_t = x + \int_0^t b(s, X_s)\mathrm{d}s + \int_0^t \sigma(s, X_s)\mathrm{d}W_s,$$

where W is a $q \geq 1$ dimensional Brownian motion. The algorithm will also approximate the solution to the related semilinear, parabolic PDE.

In recent times, there has been an increasing interest to have algorithms which work efficiently when the dimension d of the space occupied by the process X is large. This interest has been principally driven by the mathematical finance community, where nonlinear valuation rules are becoming increasingly important. Currently available algorithms [18, 54, 59, 60] rarely handle the case of dimension greater than

8. The main constraint is not only due to the computational time, but mainly due to memory consumption requirements by the algorithms.

The purpose of this second part of the thesis is to drastically rework the algorithm in [60] to first minimize the exposure to the memory due to the storage of simulations. This will allow computation in larger dimension d. Secondly, in this way the algorithm can be implemented in parallel on graphic processor units (GPUs) which enables us to obtain substantial speedups compared to CPU implementations. For instance, we can solve problems in dimension $d = 11$ within eight seconds using 2000 simulations per hypercube. We present several numerical examples in order to illustrate the performance of the scheme, being able to solve problems up to dimension $d = 19$. Moreover, we provide an error analysis of the proposed algorithm.

The outline of this thesis is as follows.

Part I, which consists of four chapters, deals with SABR-like stochastic volatility models both in equity/foreign exchange and interest rate markets, with the focus in the pricing of several market derivatives and in the calibration to real market prices.

Chapter 1 is devoted to the presentation of the Simulated Annealing global optimization algorithm which will be used in the forthcoming calibration of the models studied in Chapters 2 and 3. As the calibration in the financial world should be carried out almost in real-time we will implement the algorithms taking advantage of High Performance Computing techniques.

In Chapter 2 we study the SABR stochastic volatility model in equity and foreign exchange markets. Not only the classical SABR model called static SABR but also another extension known as dynamic SABR will be analyzed. For the dynamic SABR model we will propose an original and more general expression for the functional parameters. Then we will calibrate the models for EURO STOXX 50 index and EUR/USD exchange rate. Finally, a cliquet option on EUR/USD will be priced.

In Chapter 3 we present the SABR/LIBOR market models proposed by Hagan, Mercurio & Morini and Rebonato. The main objective of this chapter is to efficiently calibrate these models to real market prices of caplets and swaptions. We exhibit a set

of algorithms implemented using several GPUs which allow to calibrate the models using Monte Carlo simulation. This approach is particularly useful when dealing with products and models such that approximation formulas are not available or are not accurate enough.

In Chapter 4 we also operate with the cited SABR/LIBOR market models as in the previous chapter. Nevertheless, taking into account the drawbacks of Monte Carlo simulation when pricing market derivatives, i.e. the slow convergence, the valuation of options with early exercise and the computation of the "Greeks", see [139], we take advantage of the alternative PDE approach. The formulation of the corresponding PDE models for the discussed SABR/LIBOR market models is posed. These PDEs are high dimensional in space. In order to overcome the curse of dimensionality we propose the use of the sparse grid combination technique.

Part II deals with Backward Stochastic Differential Equations and contains one chapter, Chapter 5..

In Chapter 5 we design a novel algorithm based on Least-Squares Monte Carlo in order to approximate the solution of discrete time Backward Stochastic Differential Equations. Our algorithm allows massive parallelization of the computations on many core processors, such as GPUs. Our approach consists of a novel method of stratification which appears to be crucial for large scale parallelization. In this way, we minimize the exposure to the memory requirements due to the storage of simulations. Indeed, we note the lower memory overhead of the method compared with previous works.

Appendix A contains a brief description of the different optimization problems than have been tested using the proposed Simulated Annealing implementation. In Appendix B the expression of the implied volatility in the general dynamic SABR model is obtained. Besides, the employed market data are shown. Appendix C contains two mathematical results concerning the proposed algorithm for solving backward stochastic differential equations.

We finish with a short section outlining the main conclusions of this thesis.

Part I

Stochastic volatility models

Introduction to stochastic volatility models

One of the well-known limitations of the classical Black-Scholes model [13]

$$dS(t) = rS(t)dt + \sigma S(t)dW(t), \tag{1}$$

is the assumption that the volatility σ of the underlying asset S is constant. In (1) we consider the risk-neutral probability measure, where r represents the risk-free interest rate and W is a Brownian motion. To fix ideas, let us consider the time t price of a T-maturity European call option with strike K. Such a contract pays out the amount

$$\max(S(T) - K, 0) = (S(T) - K)^+,$$

at time T. Its value at time $t < T$ is given by Black's formula

$$V^{\text{Black}}(S, t, \sigma, r, K, T) = S\Phi(d_1) - K\mathrm{e}^{-r(T-t)}\Phi(d_2),$$

where Φ is the cumulative distribution function of the standard normal distribution and

$$d_1 = \frac{\log(S/K) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},$$

$$d_2 = \frac{\log(S/K) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}.$$

Black's formula is the standard in the European call options market. If we know σ and the rest of the parameters we can compute the option price. Conversely, if we

know the option price $V^{\text{Black}}(S, t, \sigma, r, K, T)$ we can calculate σ. This is the so-called implied volatility.

Next, let us consider two different strikes K_1 and K_2. Suppose the market provides us with the prices of the two related call options $V^{\text{Black}}(S, t, \sigma, r, K_1, T)$ and $V^{\text{Black}}(S, t, \sigma, r, K_2, T)$. Note that both call options have the same underlying S and the same maturity T. The point is that there is no a *single* volatility parameter σ such that

$$V^{\text{Market}}(S, t, K_1, T) = V^{\text{Black}}(S, t, \sigma, r, K_1, T),$$

and

$$V^{\text{Market}}(S, t, K_2, T) = V^{\text{Black}}(S, t, \sigma, r, K_2, T),$$

hold, i.e, two different volatilities $\sigma(T, K_1)$ and $\sigma(T, K_2)$ are required to match the observed market prices, that is:

$$V^{\text{Market}}(S, t, K_1, T) = V^{\text{Black}}(S, t, \sigma(T, K_1), r, K_1, T),$$

$$V^{\text{Market}}(S, t, K_2, T) = V^{\text{Black}}(S, t, \sigma(T, K_2), r, K_2, T).$$

An analogous argument could be followed with a fixed strike K and two different maturities T_1 and T_2. Therefore, each call option market price requires its own Black volatility $\sigma(T, K)$ depending on the strike K and the maturity T of the call option.

The shape of the implied volatility versus strike is commonly seen to exhibit "smiley" or "skewed" shapes (see Figure 1), so that it is known as the volatility smile or skew. In some markets it shows a considerable asymmetry, a skew. If we plot implied volatilities against strikes and maturities non-flat structures are normally observed not only in the equity or foreign-exchange markets but also in the interest rate markets. Ignoring volatility smiles can lead to a significant mispricing of options.

Taking into account that the dynamics described by (1) can not properly fit market implied volatilities, researchers try to find alternative models that are suitable for this purpose. We now briefly review the major approaches proposed in the literature.

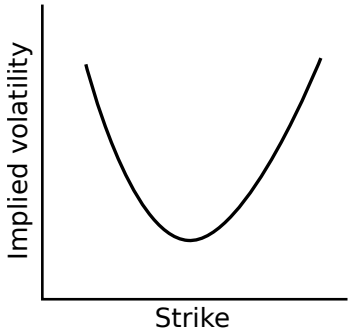


Figure 1: Shape of a typical volatility smile.

Local volatility models These models are straightforward analytical extensions of a geometric Brownian motion that allow for smiles in the implied volatility. The main examples are the following. In 1976 Cox and Ross [30] proposed the constant-elasticity-of-variance (CEV) processes. They considered stochastic differential equations of the form

$$dS(t) = rS(t)dt + \sigma S(t)^{\beta} dW(t), \quad \beta \in (0,1),$$

where the β coefficient adds skew to the model. In 2000 Andersen and Andreasen [4] developed the related application to the LIBOR market model. In 1994 and 1997 Dupire [37, 38] suggested the model

$$dS(t) = rS(t)dt + \sigma(S,t)S(t)dW(t),$$

where the instantaneous volatility σ is a deterministic function of both the asset price S and time t.

Jump-diffusion models These models have been introduced to model discontinuities in the underlying stochastic process. In the equity market these models were imported in 1976 by Merton [105] and are usually employed with the aim of calibrating volatility smiles. In the interest rate market, jump diffusion LIBOR models have been developed by Glasserman and Merener (2001, [51]) and Glasserman and Kou (2003, [50]).

Stochastic volatility models These models have been designed to fit market volatility smiles and to capture the stochastic nature of the volatility. The main examples are those of Hull and White (1987), Heston (1993) and Hagan, Kumar, Lesniewski and Woodward (2002). These models are more realistic at the expense of a greater complexity in terms of pricing options.

In the Hull and White model [77] the stock and its volatility are modelled as

$$\begin{aligned} dS(t) &= rS(t)dt + \sqrt{\sigma(t)}S(t)dW(t), \quad S(0) = S_0, \\ d\sigma(t) &= \kappa\sigma(t)dt + \zeta\sigma(t)dZ(t), \quad \sigma(0) = \sigma_0, \end{aligned}$$

where dW and dZ have correlation coefficient ρ. The other parameters of the model are the volatility return κ, the volatility of the volatility ζ and the initial value of the volatility σ_0.

The Heston model [73] is given by

$$\begin{aligned} dS(t) &= rS(t)dt + \sqrt{\sigma(t)}S(t)dW(t), \quad S(0) = S_0, \\ d\sigma(t) &= \kappa(\theta - \sigma(t))dt + \zeta\sqrt{\sigma(t)}dZ(t), \quad \sigma(0) = \sigma_0, \end{aligned}$$

where $dW(t)dZ(t) = \rho dt$. The other parameters of the model are the speed of mean reversion κ, the long-term volatility θ, the volatility of the volatility ζ and the initial value of the volatility σ_0. This model is popular among practitioners because there are closed-form solutions for European options, which are particularly useful in the calibration process. These analytical formulas are derived using the characteristic function computed by solving the associated Heston PDE and by inversion of a Fourier Transform. The application of the Heston model to the LIBOR market model was developed by Wu and Zhang [140].

In [67] Hagan, Kumar, Lesniewski and Woodward proposed the so-called SABR model, which is the natural extension of the classical CEV model to stochastic volatility. SABR is the acronym for Stochastic, Alpha, Beta and Rho, three of the four model

parameters. The dynamics of the forward price $F(t) = \mathrm{e}^{(r-d)(T-t)}S(t)$ is given by

$$dF(t) = \alpha(t)F(t)^{\beta}dW(t), \quad F(0) = F_0,$$
$$d\alpha(t) = \nu\alpha(t)dZ(t), \quad \alpha(0) = \alpha_0,$$

where (W, Z) is a bidimensional Brownian motion with constant correlation ρ. The other parameters of the model are the variance elasticity $\beta \in [0, 1]$, the volatility of the volatility ν and the volatility's reference level α_0. The main applications of the SABR model to the LIBOR market model were developed by Hagan [68], Mercurio and Morini [103] and Rebonato [122]. Hagan model arises as the natural coupling between SABR and LMM models. In the Mercurio & Morini model, the existence of a lognormal common volatility process to all forward rates is assumed, while each forward rate satisfies a particular stochastic differential equation. Rebonato model is analogous to Hagan one, except for the dynamics of the volatilities. In this thesis we will focus on SABR-like models because they are widely used in practice for several reasons. Firstly, using singular perturbation techniques it is possible to derive an approximation formula of the implied volatility in the SABR model. Secondly, the model is simple and tractable. Thirdly, its parameters, which play specific roles in the generation of smiles, have an intuitive meaning. Finally, it has become the market standard for interpolating and extrapolating prices of plain vanilla caplets and swaptions.

In the first part of this thesis we analyze the valuation of financial options using SABR-like models both in equity/foreign-exchange and interest rate markets. Our aim is to illustrate the use of these models with an emphasis on the implementation and calibration.

The calibration is the procedure to fit model parameters by matching quoted option prices in the market. The standard calibration approach minimizes the distance between model prices, V^{model}, and market prices, V^{market}. A common error measure is the squared error

$$SE = \sum_{k=1}^{N} \left(V_k^{\mathrm{market}} - V_k^{\mathrm{model}}(\boldsymbol{x})\right)^2,$$

where N is the number of quoted option prices and $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ is the vector of model parameters. The error measure is a function $f : \mathbb{R}^n \to \mathbb{R}$ of $\boldsymbol{x}$. Since we are looking for a parameter vector $\boldsymbol{x}^\star$ that best fits the model to available market prices, the calibration procedure can be interpreted as an optimization problem of the form

$$\min_{\boldsymbol{x} \in I} f(\boldsymbol{x}),$$

where $I \subseteq \mathbb{R}^n$ is the admissible set of the model parameters $\boldsymbol{x}$, $I = I_1 \times \ldots \times I_n$, where $I_k = [l_k, u_k]$, with l_k, $u_k \in \mathbb{R}$ for $k = 1, ..., n$. Due to the kind of cost functions we will be dealing with, it is preferable to use derivative-free minimization algorithms because analytical formulas will not be available and numerical differentiation is computationally expensive. In this thesis we will focus in stochastic optimization, particularly in the well-known Simulated Annealing algorithm [88].

Chapter 1

Simulated Annealing

1.1 Introduction

In this chapter we consider the box-constrained global minimization problem:

$$\min_{\boldsymbol{x} \in I} f(\boldsymbol{x}), \tag{1.1}$$

where f is the cost function, $\boldsymbol{x} = (x_1, \ldots, x_n) \in I \subset \mathbb{R}^n$, the search space being $I = I_1 \times \ldots \times I_n$, where $I_k = [l_k, u_k]$, with l_k, $u_k \in \mathbb{R}$ for $k = 1, ..., n$. This kind of problems arises in many fields of application, such as physics, finance, industry, biology, etc. Usually the dimension of the optimization problem is very large, and the evaluation of the function involves a high computational cost.

There exists a large variety of global optimization methods. They can be classified into deterministic and stochastic ones. Among the first ones are gradient-based methods, than can be applied when the cost function has adequate analytical properties. However, if the cost function is not smooth enough, it results difficult or impossible to apply these algorithms, and stochastic methods (such as Monte Carlo based ones) are more convenient. Moreover, a heuristic can be incorporated to the optimization algorithm to decide the next candidate to be tested or the way to compute the new candidate. Metaheuristic global optimization algorithms are those ones proposed to solve a general class of problems by using a combination of the cost function values

and certain abstract reasoning rules, without paying attention to the specific nature of the problem. Sometimes, this combination is carried out in a stochastic way, either by considering samples in the search space or by using somehow randomness to obtain the optimal solution. A clear example of a metaheuristic stochastic global optimization algorithm is the standard Simulated Annealing (SA) method, in which the decision of the next candidate to be considered depends on the Boltzman probability distribution, as it will be described later in this chapter. Other important examples of stochastic metaheuristic methods are genetic, swarm intelligence, parallel tempering and grenade explosion algorithms. Recently, metaheuristic algorithms have gained increasing scientific attention.

In this chapter, we focus on SA algorithm and its efficient parallelization on GPUs, which will lead us to use optimization algorithms that can also be understood as a kind of hybrid ones, combining SA and genetic algorithms (GA) (see [129]). They mainly consist of SA/GA with simple deterministic crossover operations (see [28, 27, 81]).

SA is a metaheuristic stochastic optimization method that formulates the problem of finding the optimum of a cost function as the equilibrium configuration for a statistical thermodynamical system (see [20, 33, 135]). For a fixed temperature level, it has been first introduced by Metropolis et al. in [106]. Next, SA has been extended to the case of several temperatures, emulating the annealing process of steel forming, by Kirkpatrick et al. in [88].

Due to the great computational cost of SA, its parallelization has been analyzed by several authors and using different hardware architectures along time. In [96] Lee et al. studied different parallelization techniques based on the multiple Markov chains framework. Also several authors have analyzed different approaches in a SIMD (Single Instruction, Multiple Data) machine [28], depending on the number of communications performed between the independent Markov chains, and ranging from asynchronous to synchronous schemes with different periodicity in the communications. Special attention has been addressed in reducing the number of communications between processing threads, due the high latency of the communication network. In

[32] a hybrid OpenMP/MPI implementation has been developed.

The parallelization of hybrid SA/GA algorithms has been analyzed by Chen et al. in [28]. Moreover, in [27] a parallel hybrid SA/GA in MIMD PC clusters has been implemented, analyzing different crossover operations for generating the species.

Nowadays GPUs have become a cheap alternative to parallelize algorithms. The main objective of the present chapter is to develop a generic and highly optimized version of a SA algorithm for Nvidia GPUs in CUDA [109]. For this purpose, first the more efficient versions of SA presented in [20, 135] have been analyzed, tested and adapted to the GPU technology.

In the first Section 1.2.1 we present an introduction to the sequential Simulated Annealing algorithm. Next in Section 1.2.2 we introduce the alternatives for the parallelization of the algorithm following the Multiple Markov chain approach. First a naive asynchronous implementation and then a synchronous implementation following [96] with communication between Markov chains at each temperature level is detailed. Then in Section 1.3 the algorithm is parallelized using GPUs. In GPUs decreasing the periodicity of the communications does not give a relevant difference in performance, because of the very low latency communication network between computing cores.

In the following Section 1.4 the precision of the algorithm is studied. Several classical optimization tests have been analyzed. A numerical convergence analysis is performed by comparing the sequential and parallel algorithms. Next the speedup of the parallel algorithm is studied attending to the different parameters of the optimization function and SA configurations.

Usually, the SA algorithm is used to obtain a starting point for a local optimization algorithm. In this chapter we also present some examples of the precision and computational time, using our CUDA SA implementation and a Nelder-Mead algorithm.

Most of the results in this chapter are included in the reference [43].

1.2 Simulated annealing

1.2.1 Sequential Simulated Annealing

As indicated in the introduction, SA is a stochastic optimization method which is mainly based on some statistical mechanics concepts. Thus, it formulates the problem of finding the optimum of a cost function in terms of obtaining the equilibrium configuration for a statistical thermodynamical system. Statistical mechanics is based on the description of a physical system by means of a set representing all possible system configurations and the probabilities of achieving each configuration. Thus, each set is associated with a partition function.

We say that a system is in equilibrium if the transition probability from state S_i to state S_j, $P(S_i \to S_j)$, is the same as the probability of going from state S_j to state S_i, $P(S_j \to S_i)$. A sufficient condition for equilibrium is the so-called *detailed balance* or *local balance* condition, that can be written using the Bayesian properties:

$$\pi_i P(S_i \to S_j) = \pi_j P(S_j \to S_i), \tag{1.2}$$

where π_i and π_j are the probabilities of being in the states i and j, respectively. These conditions can also be formulated in terms of Markov chains. A Markov process is said to be reversible (or time reversible), if it has a detailed balance where $P(S_i \to S_j)$ denotes the Markov transition probability between the states i and j. That is, the forward and backwards Markov chains have the same transition probabilities.

Metropolis et al. proposed in [106] an algorithm for the simulation of atoms in equilibrium at a given fixed temperature. It was based on the notion of detailed balance that describes equilibrium for thermodynamical systems, whose configurations have probability proportional to the Boltzmann factor. The algorithm finds the transition probabilities for a Markov chain that yields the desired steady state distribution. They introduced a random walk (Markov chain of configurations) through the configuration space, using a fictitious kinetics. In this Markov chain approach, the time refers to the number of iterations of the procedure. Moreover, we assume

that our statistical system is considered to be in equilibrium so that the time results to be irrelevant. Starting from a set of transition probabilities, a new set of transition probabilities satisfying the detailed balance condition can be found. This can be done by only accepting some of the transitions (see [14]). By appropriately using this procedure, the Markov chain converges to the steady state equilibrium distribution.

We aim to sample the space of possible configurations using a statistical thermodynamical system, that is in a thermal way. So, we force this system to satisfy the equation (1.2). For the distribution function we chose the Boltzmann one, with degeneracy factor 1, i.e. without repeated arrangements; which indicates the way the particles are distributed among the energy levels in a system in thermal equilibrium. More precisely, in order to define the probability of being at state S_i at temperature T we choose

$$\pi(S_i, T) = \frac{1}{Z(T)} \exp\Big(-\frac{E_i}{k_b T}\Big), \tag{1.3}$$

where k_b is the Boltzmann constant, E_i is the energy level at state S_i and Z is a normalization function, also referred as the partition function, which depends on the temperature T in the form

$$Z(T) = \sum_{j=1}^{L} \exp\left(-\frac{E_j}{k_b T}\right),$$

where L is the length of the Markov chain. Moreover, if the probability is given in terms of the Boltzmann distribution (1.3) then we have

$$\frac{P(S_i \to S_j)}{P(S_j \to S_i)} = \frac{\pi(S_i, T)}{\pi(S_j, T)} = \frac{\frac{1}{Z(T)} \exp\Big(-\frac{E_i}{k_b T}\Big)}{\frac{1}{Z(T)} \exp\Big(-\frac{E_j}{k_b T}\Big)} = \exp\Big(-\frac{\Delta E_{ij}}{k_b T}\Big), \tag{1.4}$$

with $\Delta E_{ij} = E_i - E_j$. Thus, the ratio in (1.4) does not depend on Z.

In [106] Metropolis et al. introduced a sufficient condition for the system to satisfy the detailed balance property. More precisely, the authors noticed that the relative

probability of equation (1.4) could be obtained at simulation level by choosing

$$\frac{P(S_i \to S_j)}{P(S_j \to S_i)} = \begin{cases} \exp\left(\dfrac{-\Delta E_{ij}}{k_b T}\right) & \text{if} \quad \Delta E_{ij} \geq 0, \\ \\ 1 & \text{if} \quad \Delta E_{ij} < 0. \end{cases} \tag{1.5}$$

By using the above election, the Markov chain satisfies the detailed balance condition. Therefore, if the trial satisfies condition (1.5) for the Boltzmann probability then the new configuration is accepted. Otherwise, it is rejected and the system remains unchanged. By using the appropriate physical units for energy and temperature we can take $k_b = 1$, so that this strategy can be summarized as follows:

1. Starting from a configuration S_i, with known energy E_i, make a change in the configuration to obtain a new (neighbor) configuration S_j.

2. Compute E_j (usually, it will be close to E_i, at least near the limit).

3. If $E_j < E_i$ then assume the new configuration, since it has lower energy (a desirable property, according to the Boltzmann factor).

4. If $E_j \geq E_i$ then accept with probability $\exp(-\Delta E_{ij}/T)$ the new configuration (with higher energy) . This strategy implies that even when the temperature is higher in the new configuration, we don't mind following steps in the *perhaps wrong* direction. Nevertheless, at lower temperatures we are more forced to accept the lowest configuration we can find in our neighborhood and a jump to another region is more unlikely to happen.

Note that the original Metropolis algorithm is designed to find the optimum configuration of the system at a fixed temperature. Later on, the Metropolis algorithm has been generalized by Kirkpatrick et al. in [88], where an annealing schedule is introduced by defining how the temperature can be reduced. The algorithm starts with a high enough initial temperature, T_0, and the temperature is slowly decreased by following a geometric progression, that is $T_n = \varphi T_{n-1}$ with $\varphi < 1$ (usually $0.9 \leq \varphi < 1$

to obtain a slow freezing procedure). Thus, the SA algorithm consists of a temperature loop [20], where the equilibrium state at each temperature is computed using the Metropolis algorithm. Therefore, the SA algorithm can be decomposed in the following steps (for example, see [20] for details):

- **Step 1**: Start with the given temperature, T_0, and the initial point, $\boldsymbol{x}_0$, with energy of configuration $E_0 = f(\boldsymbol{x}_0)$, where f denotes the cost function of the problem (1.1).

- **Step 2**: Select a random coordinate of $\boldsymbol{x}_0$ and a random number to modify the selected coordinate to obtain another point $\boldsymbol{x}_1 \in V$ in the neighborhood of $\boldsymbol{x}_0$.

- **Step 3**: Compare the function value at the two previous points, by using the Metropolis criterion as follows: let $E_1 = f(\boldsymbol{x}_1)$ and select a sample, u_1, of a uniform random variable $\mathcal{U}(0,1)$. Then, move the system to the new point if and only if $u_1 < \exp(-(E_1 - E_0)/T_0)$, where T_0 is the current temperature. In this way, $E_1 - E_0$ has been compared with an exponential random variable with mean T_0. Note that we always move to the new point if $E_1 < E_0$, and that at any temperature there is a chance for the system to move "upwards". Note that we need three uniform random numbers: one to choose the coordinate, one to change the selected coordinate and the last one for the acceptance criterion.

- **Step 4**: Either the system has moved or not, repeat steps $2-3$. At each stage we compare the function at the new point with the function at the previous point until the sequence of accepted points fulfills some test of achieving an equilibrium state.

- **Step 5**: Once the loop of the previous step has finished and an equilibrium state has been achieved for a given temperature, T_0, the temperature is decreased according to the annealing schedule, $T_1 = \varphi T_0$ (with a decreasing factor φ, $0 < \varphi < 1$, usually φ close to one). Next, step 2 starts again with temperature T_1 from the point obtained in the last iteration of the algorithm as initial state.

The iteration procedure continues until a stopping criterion indicating that the system is enough frozen, in our case until achieving a target temperature T_{min}.

Notice that since we continue steps $2-3$ until an equilibrium state, the starting values in step 1 have no effect on the solution. The algorithm can be implemented in numerous ways.

1.2.2 Parallel Simulated Annealing

The pseudocode of the algorithm described in the previous Section 1.2.1 can be sketched as follows:

```
x = x_0; T = T_0;
do
    for j = 1 to L do
        x' = ComputeNeighbour(x);
        ΔE = f(x') − f(x); // Energy increment
        if (ΔE < 0 or AcceptWithProbability exp(−ΔE/T))
            x = x'; // The trial is accepted
    end for
    T = φT; // with 0 < φ < 1
while (T > T_min);
```

The SA algorithm is intrinsically sequential and thus it results difficult to parallelize it without changing its recursive nature (see [28]).

Several strategies can be followed in order to parallelize SA (see [96], for example):

- *Application dependent parallelization.* The operations of the cost function are decomposed among processors.
- *Domain decomposition.* The search space is sliced in several subdomains, each processor searches the minimum at each subdomain and then shares its results with the rest of processors.
- *Multiple Markov chains approach.* The most natural way to parallelize SA is to follow a multiple Markov chain strategy, where multiple Markov chains are

executed asynchronously and they communicate their final states every certain periods or at the end of the process. This enables independent movements on each worker (SA chain) during the intervals between consecutive communications. Attending to the number of communications performed we can classify parallel implementation in different categories.

The most straightforward approach is the case where the Markov chains only communicate their states at the end of process. This is called asynchronous approach (see [89, 96, 114]).

On the other hand, in the synchronous approach the Markov chains communicate their states at intermediate temperature levels. Only function values are exchanged among workers. The communication can be performed at each temperature level (intensive exchange of solutions) or at a fixed number of temperature levels (periodic exchange of solutions).

Also we can classify the synchronous schemes attending to the type of the performed exchange operation. This exchanged operation can be understood as a crossover genetic algorithm operation where each Markov chain corresponds to an individual of a genetic algorithm. The most simple crossover operation is taking the minimum among all the values returned by the Markov chains at the current temperature level (see [27, 96, 114]).

Asynchronous

In order to parallelize SA, the most straightforward approach to take advantage of the number of processors consists of simultaneously launching a great number of SA processes. Thus, each processor performs a SA process asynchronously. At the final stage a reduce operation to obtain the best optimum among all of the computed ones is performed. In this procedure, either the initial configuration can be the same for all SA chains or a different starting configuration for each processor can be randomly chosen (see Figure 1.1).

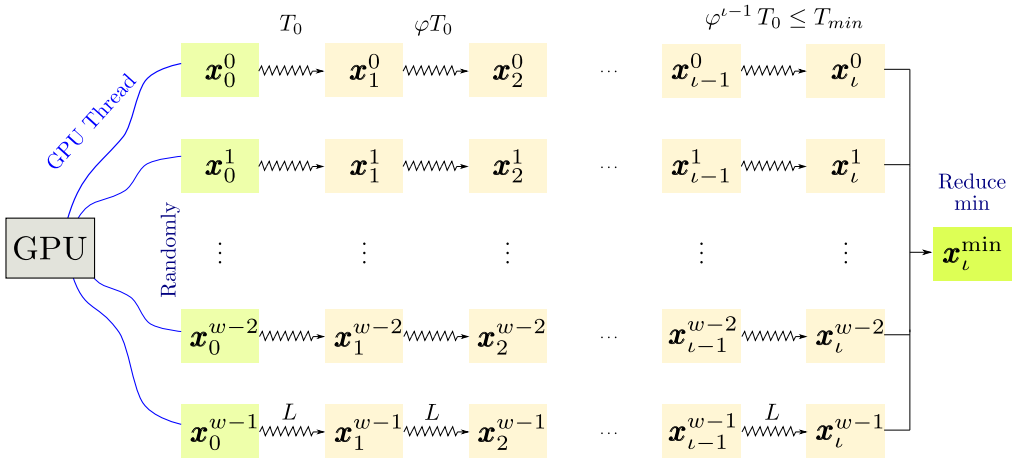


Figure 1.1: Sketch of the asynchronous parallel algorithm.

Synchronous approach with solution exchange at each temperature level

In the so-called synchronous implementation, threads start from a random initial solution $\boldsymbol{x}_0$, so that each thread runs independently a Markov chain of constant length L until reaching the next level of temperature. As the temperature is fixed, each thread actually performs a Metropolis process. Once all threads have finished, they report their corresponding final states $\boldsymbol{x}^p$ and the value $f(\boldsymbol{x}^p)$, $p = 0, \ldots, w-1$ (where w denotes the number of threads). Next, a reduce operation to obtain the minimum of the cost function is performed. So, if the minimum is obtained at a particular thread $p^\star$ then $\boldsymbol{x}^{p^\star}$ is used as starting point for all threads at the following temperature level (see Figure 1.2). In the case of two or more points with the same objective function value, the algorithm selects one of them and this choice does not affect the final result.

This algorithm can be interpreted as a mixed technique of a genetic algorithm and a SA one in which each independent Markov chain (SA process) corresponds to a different individual in a genetic algorithm. Moreover, the reduce operator can be understood as a crossover operation of a genetic algorithm to select the evolution of these species. In [96] and [114] it is noted that for this algorithm the independence of the Markov chains is lost: actually they depend on each other due to the use of a

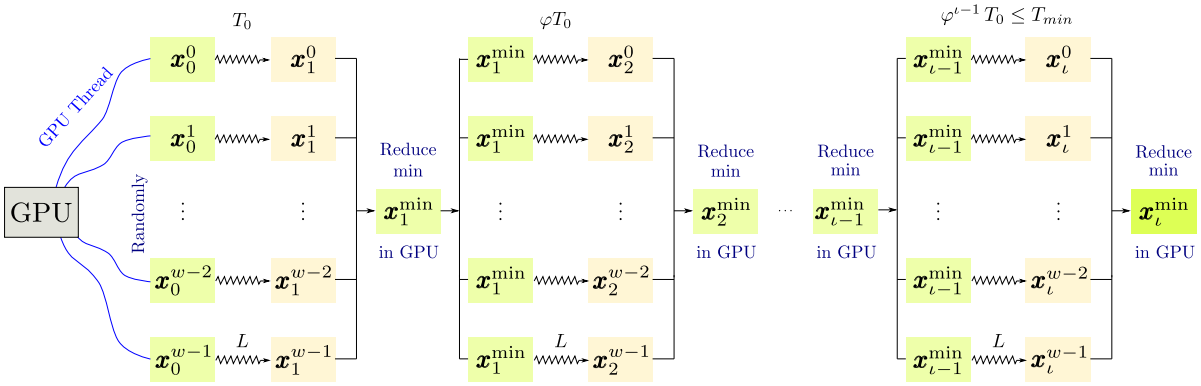


Figure 1.2: Sketch of the synchronous parallel algorithm.

deterministic crossover operation (the minimum). This fact is overcome in [114] by introducing the so-called *Synchronous approach with occasional solution exchanges* (SOS) algorithm, where the authors propose a stochastic crossover operator.

1.3 Implementation on GPUs

1.3.1 General-Purpose Computing on Graphics Processing Units (GPGPU)

From the mid nineties of 20th century, 3D capable Graphics Processing Units (GPUs), specialized graphics chips (coprocessors) independent from the CPU, started to be commonly used and integrated in computers. Pushed by the spectacular growth of graphics and videogames industries, always demanding more and more computing power, GPUs have spectacularly evolved during the last 10 years, becoming powerful and complex pieces of supercomputing hardware, with a massive parallel architecture.

Nowadays, a modern GPU consists of a many core processor, that can pack several hundreds (or even thousands) of computing cores/processors that work simultaneously and allows to execute many computing threads in parallel. Furthermore, all these cores can access to a common off-chip RAM memory by using a hardware

topology that allows these threads to retrieve simultaneously several data from this memory, by performing memory access operations also in parallel (under certain constraints). With all these processors working together, the GPU can execute many jobs in parallel. In Nvidia notation, we could call this architecture SIMT (Single Instruction, Multiple Threads), where a common program/piece-of-code (or computing kernel) is simultaneously executed by several threads over different data. This reminds the philosophy relying on the SIMD architecture.

As modern GPUs become more and more powerful in the last years, they increasingly attract the scientific community attention, which realized their potential to accelerate general-purpose scientific and engineering computing codes. This trend is called General-Purpose Computing on Graphics Processing Units (GPGPU), and consists of taking advantage of modern GPUs to perform general scientific computations.

Besides their intensive computational power, nowadays GPUs have become very popular in the supercomputing world, mainly because of the following advantages: they allow to save energy (as they are cheap and efficient in terms of Gflop per Watt), they are cheap (in terms of money per Gflop), and they also allow to save space (as many cores are packed into a small area).

As shown in the Top500 list (in June 2012), which lists the 500 more powerful supercomputers in the world (see [151]), three of the top ten supercomputers are heterogeneous systems, that use Nvidia GPUs for offloading calculus.

However, GPUs are very specialized and cannot live on their own. The GPU is a coprocessor that is used to accelerate applications running on the CPU, by offloading the most compute-intensive and time consuming portions of the code, although the rest of the application still runs on the CPU. So, they depend on a CPU to control their execution.

Taking into account the large number of Markov chains that can be simultaneously computed to solve the minimization problem, the here treated algorithms are particularly well suited to be implemented in GPU technology.

Currently, there are two main GPU manufacturers, Nvidia and AMD (formerly ATI graphics). These two architectures are conceptually similar, although each one presents its own hardware peculiarities. In this thesis we have chosen Nvidia GPUs, whose architecture is detailed in the next section.

1.3.2 Nvidia GPUs, many core computing

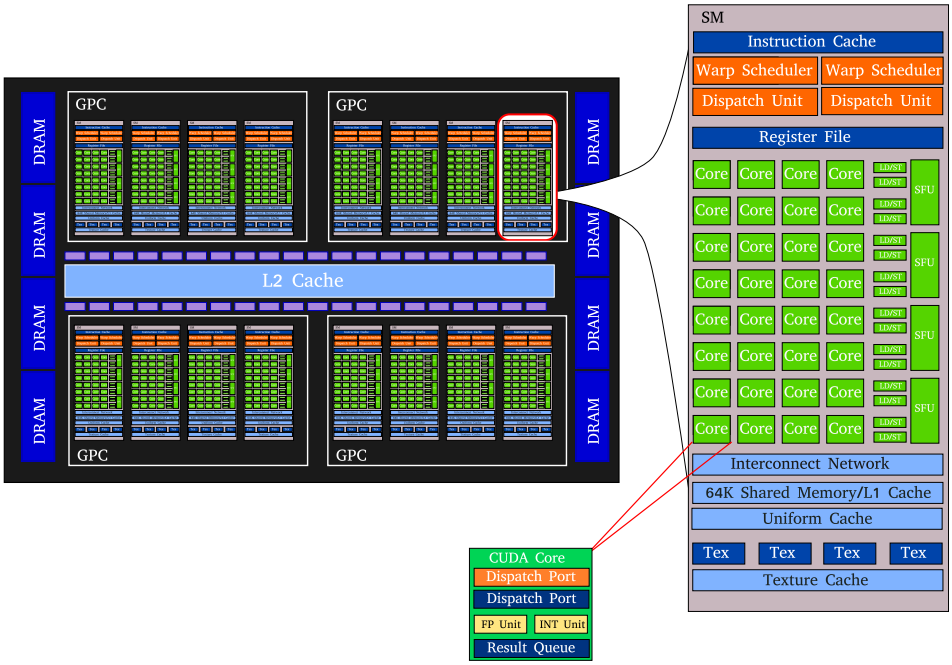


Figure 1.3: Nvidia GPU hardware structure.

A GPU (from now on, "*the device*") can be seen as a powerful SIMD coprocessor, endowed with a huge floating point processing power. Such coprocessor must be managed from a common CPU (from now on, "*the host*"). In this chapter we have used Nvidia GPUs, more precisely, the so-called Fermi architecture (or GF100) introduced in early 2010. For a detailed explanation about this architecture, see [112].

As a physical layout (see Figure 1.3), Nvidia Fermi GPUs chipsets are organized as a set of a variable number of Streaming Multiprocessors (SM) (from one to a

maximum of sixteen, in the top Fermi models) grouped into Graphics Processing Clusters (GPC). Each SM contains a variable number of cores (or processors), 32 in the case of the reference model of the GF100 chipsets. Each core contains a floating point unit and an integer unit. Each floating point unit can perform IEEE 754-2008 compliant double-precision floating point operations, in two clock cycles (half the performance of single-precision math). The SM can process several execution threads at a time: they are planned and launched by a thread scheduler. The main differences between GPU and CPU threads are: firstly, GPU threads are extremely lightweight, i.e. very little creation overhead and instant switching; secondly, GPUs use thousands of threads to achieve efficiency (instead, multi-core CPUs can only use a few).

Similarly to the CPU, Fermi GPUs have its own memory hierarchy:

- *Device global memory.* The GPU has its own high latency Random Access Memory (RAM) space (called device global memory), that is completely separate from the host memory. All transfers between these two memories have to be explicitly instructed by the programmer and these transfers have to be carefully designed because of the connection bandwidth (PCI Express 2.0) and the memories latencies. Global memory space can be accessed at any time by all cores. The bandwidth from global memory to the SMs is much bigger than the one of the CPU to its memory, with a peak of ≈ 200 GB/s in top models. That is because under certain assumptions the global memory access by 32 threads is coalesced into a single memory transaction, as soon as the data lie in the same segment of memory of size 128 Bytes. On Fermi coalesced size is a full warp (32 threads). Also if data are not aligned we can achieve a high bandwidth using textures, specifically designed to exploit data spatial locality.

  Previously to performing any calculus, the data to be processed must be pulled from the CPU to the GPU device memory and, once the calculations have finished, the computed results must be retrieved from the GPU.

  Even with the device memory bandwidth being really high, it is not enough to feed all the processors, i.e. to keep them fully occupied (note that all the

processors have a theoretical peak performance of 520 double-precision Gflops), so that a cache hierarchy is necessary.

- A "huge" 768 KB *L2 cache*. It is shared by all SMs and it manages the read-/write and texture requests.

- *Shared memory/L1 cache*. For low-latency access to shared data by cooperating threads in the same SM (implemented on chip).

  Moreover, to benefit from frequently accessed data, each SM contains a low latency cache SRAM, referred as shared memory, of 64 KB that is shared by all the cores of the SM, as a shared resource. In Fermi, this memory can be partitioned into a self-managed cache and a programmer-managed shared memory (in blocks of 48 KB and 16 KB).

- *Texture cache*. With 12 KB per SM, designed for small texture filtering operations, with spatial locality.

- *Registers*. In addition to all these memories each SM contains a certain number of registers to store instruction operands (more precisely, in our case 32 K registers of 32-bits per multiprocessor).

From the programming point of view, similarly to the SIMD (Single Instruction, Multiple Data) execution model used for general data-parallel programming, the Nvidia model is SIMT (Single Instruction, Multiple Threads): the code execution unit is called a kernel and is executed simultaneously on all SMs by independent blocks of threads; each thread is assigned to a single processor and executes within its own execution environment (instruction address and register state), but they all run the same instruction at a time, although over different data. In order to efficiently execute hundreds of threads in parallel, the SM hardware is multithreaded. The SM also implements the synchronization barrier with a single instruction. Once a block has its threads assigned to a SM, it is further divided by the SIMT multithreaded instruction unit into 32-threads units called warps.

Each SM manages a pool of up to 48 warps (giving a total of 1536 threads), although only one of these warps will be actually executed by the hardware at any time instant. Threads are assigned to Streaming Multiprocessors in block granularity (up to 8 blocks to each SM, for example 6 blocks of 256 threads). The size and number of blocks are defined by the programmer. Threads run concurrently and the SM maintains thread/block id's and manages/schedules the threads execution.

There is an API for programming Nvidia GPUs called CUDA (Compute Unified Device Architecture) [109], which consists of: some drivers for the graphics card, a compiler and a language that is basically a set of extensions for the C/C++ language, that allows to control the GPU (the memory transfer operations, the work assignment to the processors and the processors/threads synchronization).

CUDA provides all the means of a parallel programming model with the particularity of the previously cited types of memory. There are CUDA instructions to manage all of these memories and to declare variables that are stored in any of those memory spaces. Inside the device, threads are able to access data from multiple memory spaces. Each thread block contains a shared memory which is visible to all threads of the block and with the same lifetime as the block. All threads from any grid have access to the same global memory which is persistent across kernel launches by the same application. Each transfer between these memory spaces must be also explicitly managed. CUDA also allows to work with texture memory to exploit data locality and with constant memory, used to store small structures that are reused by all threads and that is also persistent across kernel launches. Transferring data between different types of memory inside the device results also important because of the different access patterns and the specific size and latency of the memory.

Thus due to the execution model and memory hierarchy, GPUs support two levels of parallelism:

- An outer fully-parallel loop level that is supported at the grid level with no synchronization. Thread blocks in a grid are required to execute independently. It must be possible to execute them in any order, in parallel or in series. This

independence requirement allows thread blocks to be scheduled in any order across any number of cores, thus enabling programmers to write scalable code.

- An inner synchronous loop level at the level of thread blocks where all threads within a block can cooperate and synchronize.

1.3.3 Notes on the CUDA implementation

In this section we detail the pseudocodes for the proposed asynchronous and synchronous versions of the parallel code.

The CUDA pseudocode of the asynchronous version is shown below:

```
Initialize T = T_0
Initialize L, varphi, T_min
Initialize n_blocks, n_threads_per_block
Initialize d_points = startPoint
Initialize bestPoint = 0

cusimann_kernel<<<n_blocks,n_threads_per_block>>>(T, L, varphi, d_points, bestPoint)
bestPoint = reduceMin(d_points)
```

Listing 1.1: Asynchronous Simulated Annealing.

A kernel, `cusimann_kernel`, that executes a sequential Simulated Annealing in each thread, is launched from the host (see the Listing 1.1). The CUDA kernel is simultaneously executed in parallel by a large number of threads, thus allowing to compute a large number of Markov chains (in the here used GPU, GeForce GTX 470, the number of available CUDA cores is 448). More precisely, this kernel launches a grid of `n_blocks` thread blocks and each thread block groups `n_threads_per_block` threads.

```
__global__ void cusimann_kernel(T, L, varphi, d_points, bestPoint) {

  Initialize global_tid
  Initialize x0 = d_points[global_tid] = bestPoint
  Initialize f_x0 = f(x0)
  do {
    for(i=0;i<L;i++){
      // Generate another point x1 in the neighborhood of x0
      d = Select randomly a coordinate of x0
```

```
      u = Select a random number to modify the selected d coordinate
      x1 = ComputeNeighbour(x0,d,u)
      f_x1 = f(x1)

      // If x1 satisfies the Metropolis criterion, move the system to x1
      if ( GenerateUniform(0,1) <= exp( -(f_x1-f_x0)/T ) )
        x0 = x1
        f_x0 = f_x1;
      end

    }
    T = T*varphi
  } while (T>T_min)
}
```

Listing 1.2: Asynchronous Simulated Annealing kernel.

Moreover, we take advantage of the constant memory to store the constant parameters, like n, L and the box limits (l_k and u_k), so that these data can be broadcasted to all threads. Furthermore, constant memory is cached, so that several consecutive accesses to the same memory position do not increase memory traffic. This is important because these consecutive accesses are repeatedly required by the SA algorithm.

As indicated in the Step 3 of the algorithm described in Section 1.2.1, at each step of the Markov chain three uniform random numbers are required. At this point we take advantage of the Nvidia CURAND library [110], that allows parallel generation of random numbers to use them immediately by the kernels, without the extra time cost of writing and reading them from global memory.

As indicated in Section 1.2.2, once all threads have finished to compute the Markov chains, the minimum of the function is obtained by a reduction operation (see Listing 1.1). More precisely, for this purpose we need to find the index associated to the minimum of the vector storing the cost function values at the points returned by the threads. This operation is carried out in parallel inside the GPU, by means of the specific optimized Nvidia Thrust library, [147], that takes advantage of the coalesced memory access and the involved partial reductions are performed in shared memory.

Unlike the asynchronous version, in the synchronous one the temperature loop

is carried out at CPU level, as detailed in the pseudocode in Listing 1.3. Thus, at each temperature step the execution of the kernel detailed in Listing 1.4, as well as the reduction operation are required. As illustrated in the forthcoming Table 1.2, the repeated use of the optimized reduction operation does not cause a significant performance overhead.

We also notice that in all implementations slow data transfers between CPU and global GPU memory do not appear.

```
Initialize T = T_0
Initialize L, varphi, T_min
Initialize n_blocks, n_threads_per_block
Initialize d_points = startPoint
Initialize bestPoint = 0
do {
  cusimann_kernel<<<n_blocks, n_threads_per_block>>>(T,L,varphi,d_points,bestPoint)
  bestPoint = reduceMin(d_points)
  T = T*varphi
} while (T>T_min)
```

Listing 1.3: Synchronous Simulated Annealing.

```
__global__ void cusimann_kernel(T, L, varphi, d_points, bestPoint) {

  Initialize global_tid
  Initialize x0 = d_points[global_tid] = bestPoint
  Initialize f_x0 = f(x0)

  for(i=0;i<L;i++){
    // Generate another point x1 in the neighborhood of x0
    d = Select randomly a coordinate of x0
    u = Select a random number to modify the selected d coordinate
    x1 = ComputeNeighbour(x0,d,u)
    f_x1 = f(x1)

    // If x1 satisfies the Metropolis criterion, move the system to x1
    if ( GenerateUniform(0,1) <= exp( -(f_x1-f_x0)/T ) )
      x0 = x1
      f_x0 = f_x1;
    end

  }

}
```

Listing 1.4: Synchronous Simulated Annealing kernel.

1.4 Numerical experiments: academic tests

In this section several experiments are presented to check the correctness and performance of the here proposed CUDA implementation of SA. This CUDA implementation has been developed from an optimized C code, following the ideas of Section 1.2.2, so that both codes perform exactly the same operations and their performance can thus be compared. The numerical experiments have been performed with the following hardware and software configurations: a GPU Nvidia Geforce GTX470, a recent CPU Xeon E5620 clocked at 2.4 Ghz with 16 GB of RAM, CentOS Linux, Nvidia CUDA SDK 3.2 and GNU C++ compiler 4.1.2.

In what follows, we denote by V0 the sequential implementation, by V1 the parallel asynchronous version and by V2 the parallel synchronous one.

1.4.1 Analysis of a sample test problem: Normalized Schwefel function

A typical benchmark for testing optimization techniques is the normalized Schwefel function (see [150], for example):

$$f(\boldsymbol{x}) = -\frac{1}{n}\sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right), \quad -512 \leq x_i \leq 512, \quad \boldsymbol{x} = (x_1, \ldots, x_n). \tag{1.6}$$

For any dimension n, the global minimum is achieved at the point $\boldsymbol{x}^\star$, the coordinates of which are $x_i^\star = 420.968746$, $i = 1, \ldots, n$, and $f(\boldsymbol{x}^\star) = -418.982887$.

Table 1.1 illustrates the accuracy for the three versions of the SA algorithm: sequential (V0), asynchronous (V1) and synchronous (V2). For these three versions we use the following configuration: $T_0 = 1000$, $T_{min} = 0.01$, $L = 100$ and $\varphi = 0.99$. For the parallel versions we use the choice $b = 256$ and $g = 64$, for the number of threads per block (block size) and the number of blocks per grid (grid size), respectively, so that the number of Markov chains is 16384. With this configuration, the algorithm performs 1.8776×10^9 function evaluations in all cases.

n	V0		V1		V2	
	$\lvert f_a - f_r \rvert$	Relative error	$\lvert f_a - f_r \rvert$	Relative error	$\lvert f_a - f_r \rvert$	Relative error
8	1.3190×10^{-1}	2.4283×10^{-3}	1.2891×10^{-2}	7.4675×10^{-4}	1.7000×10^{-5}	4.1656×10^{-5}
16	2.3712×10^{-1}	3.2557×10^{-3}	7.4586×10^{-2}	1.8240×10^{-3}	1.9000×10^{-6}	5.0686×10^{-5}
32	3.3774×10^{-1}	3.8852×10^{-3}	2.8171×10^{-1}	3.5468×10^{-3}	1.5730×10^{-4}	6.0577×10^{-5}
64	7.9651×10^{-1}	5.9664×10^{-3}	9.7831×10^{-1}	6.6126×10^{-3}	3.1880×10^{-4}	1.2132×10^{-4}
128	1.9198	9.2648×10^{-3}	3.0461	1.1674×10^{-2}	1.2225×10^{-4}	1.5304×10^{-4}
256	3.6230	1.2733×10^{-2}	9.5765	8.0283×10^{-2}	1.4953×10^{-2}	8.2214×10^{-4}
512	7.3054	1.8097×10^{-2}	26.2282	4.0424×10^{-1}	4.6350×10^{-1}	4.5503×10^{-3}

Table 1.1: Error of the solution obtained by the algorithm, both in the value of the function at the minimum (columns $|f_a - f_r|$, where f_a is the objective function value found by the algorithm and f_r is the exact function value in the real minimum) and in the minimum (columns Relative error, measured in $\| \cdot \|_2$).

In order to take into account the impact of the random number seeds, we execute each algorithm 30 times in all performed minimization examples. The synchronous version provides much better convergence results than the other two ones. Note that we have chosen the same SA configuration for all executions with different values of n ($n = 8, 16, 32, 64, 128, 256, 512$), so that the error obviously increases with the value of n. When the size of the problem increases, we should have selected a more restrictive SA setting because the minimization problem becomes more complex. Nevertheless, in order to compare in Table 1.2 the speedups of the parallel versions for different values of n it results more convenient to consider the same SA setting for all cases.

Table 1.2 shows the performance of the GPU implementation with respect to a one core CPU. When n increases the algorithm needs larger memory transfers, so that the speedup decreases. Notice that even for moderate values of n the execution of the SA algorithm becomes memory-bounded, which means that its performance is limited by the memory bandwidth and not by the floating point performance.

In short, Tables 1.1 and 1.2 show that the asynchronous version results to be a bit faster than the synchronous one, this is mainly because it does not perform reduction operations. Nevertheless, the errors are much larger in the asynchronous version. Notice that in all presented tables the computational time is expressed in seconds.

n	V0	V1		V2	
	Time	Time	Speedup	Time	Speedup
8	1493.7686	5.5436	269.4595	5.6859	262.7121
16	2529.3072	15.3942	164.3027	15.5889	162.2502
32	4618.5820	56.9808	81.0550	60.1882	76.7356
64	8773.0560	106.6075	82.2930	110.2702	79.5596
128	17169.0000	210.9499	81.3890	215.5416	79.6552
256	34251.9240	455.4910	75.1978	462.8035	74.0096
512	68134.5760	871.7434	78.1589	893.7668	76.2330

Table 1.2: Performance of CUDA version vs. sequential version with one CPU core for different number of parameters.

Numerical convergence analysis

In this case we compare the convergence of the two parallel algorithms with the sequential version. For this purpose the same number of explored points in the function domain are considered. In Figure 1.4 three graphics of the relative error vs. the number of explored points for $n = 8$ and 16 are presented. In Figure 1.5 the same graphics for $n = 32$ and 64 are shown, and in Figure 1.6 the plots are presented for $n = 128$ and 256. From them, it is clear that the synchronous version converges more quickly. In order to compare the asynchronous version with the other two ones at a given temperature step of SA, we must choose a point that summarizes the state where the different threads are. For this purpose we have chosen the best point of all threads, so that we are very optimistic in representing the convergence of the asynchronous version.

As expected, all the results presented so far show that the synchronous version results better to approximate the solution, specially for higher dimensional problems. Therefore, in the forthcoming subsections we only analyze the behavior of this synchronous version.

Increasing the number of launched threads

At this point we analyze the algorithm behavior when increasing the number of launched threads. Table 1.3 illustrates how the error of the obtained solution is

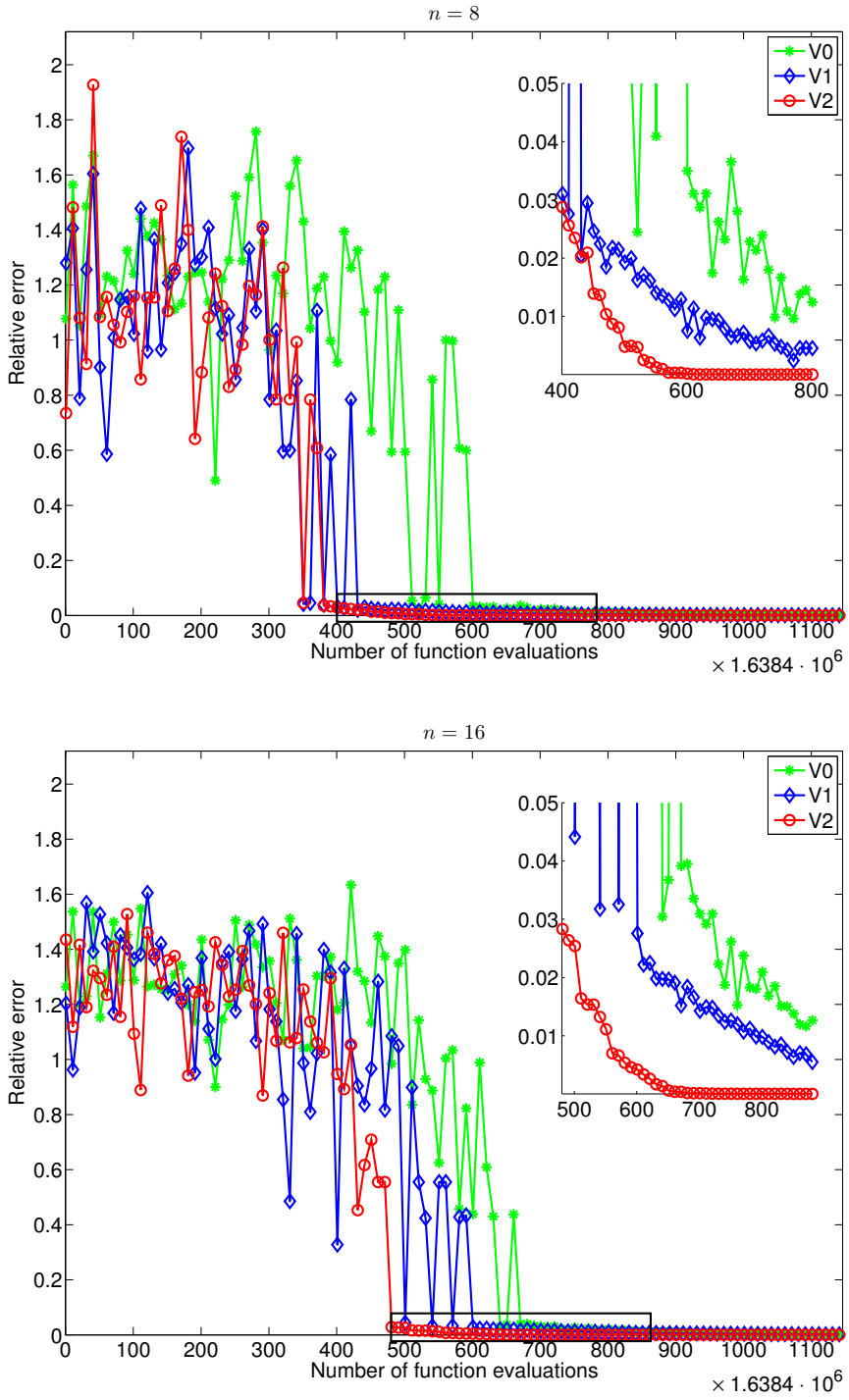


Figure 1.4: For the three versions V0, V1 and V2, convergence rate for runs with $n = 8$ and 16.

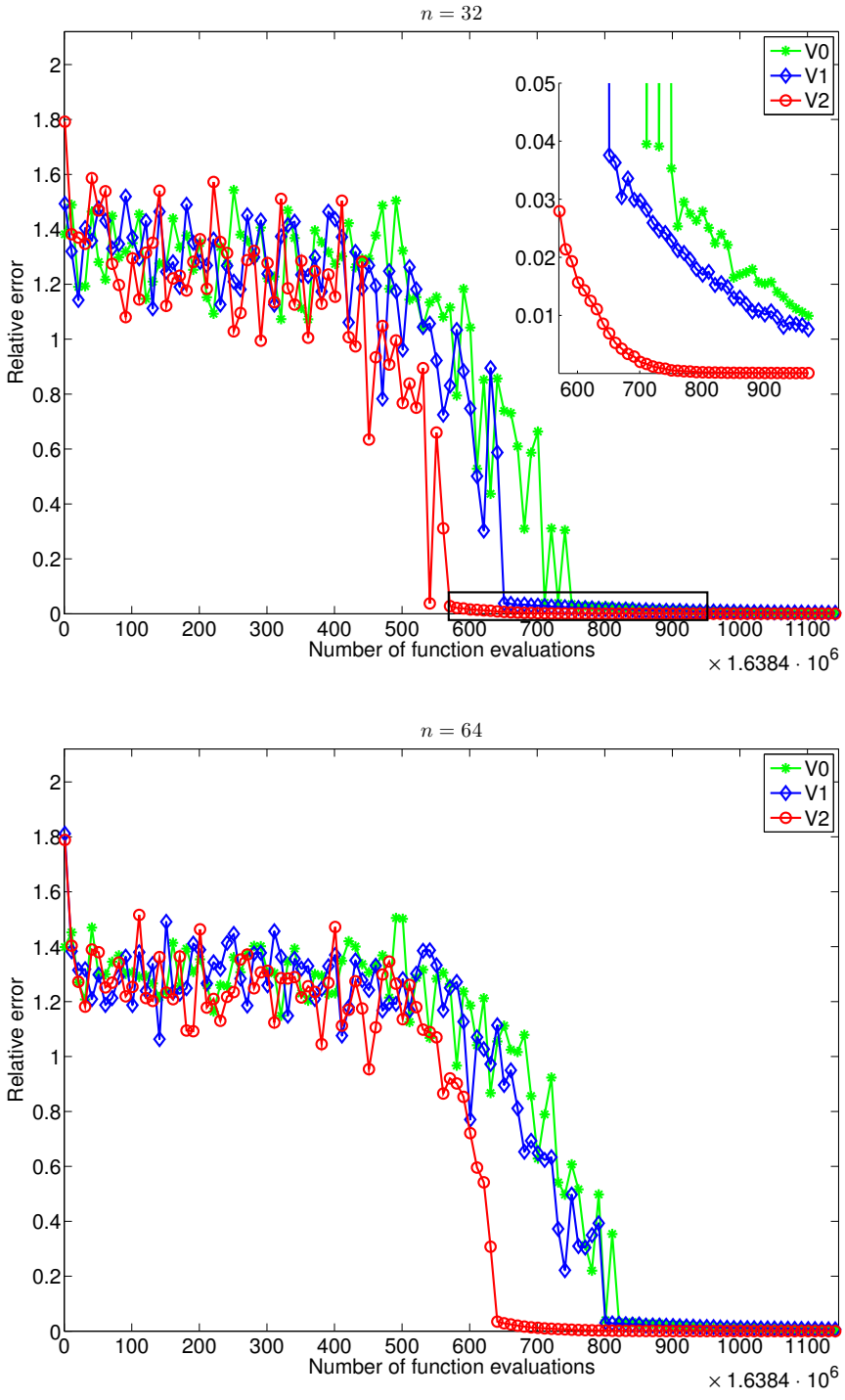


Figure 1.5: For the three versions V0, V1 and V2, convergence rate for runs with $n = 32$ and 64.

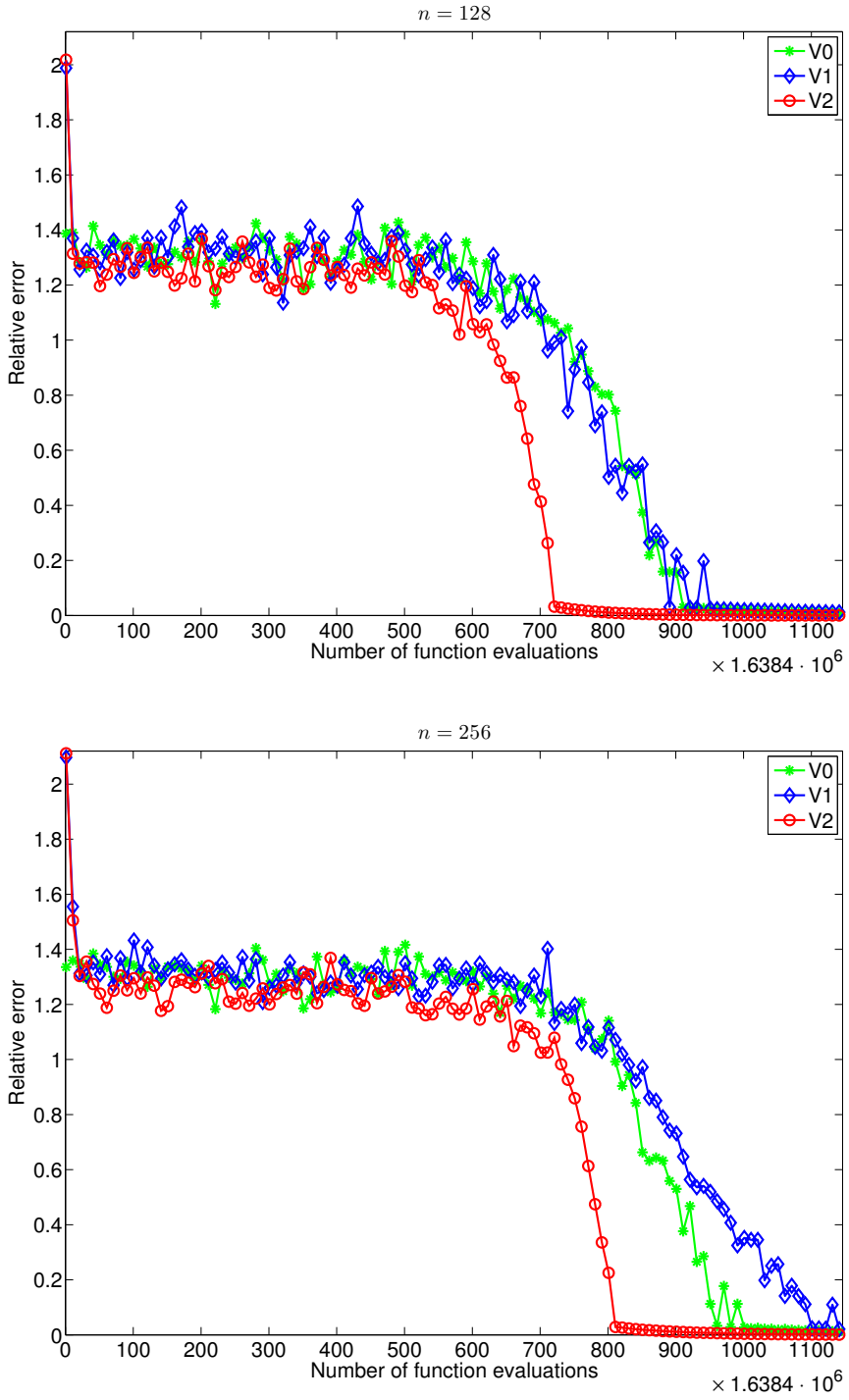


Figure 1.6: For the three versions V0, V1 and V2, convergence rate for runs with $n = 128$ and 256.

Threads	Function evaluations	$\lvert f_a - f_r \rvert$	Relative error
768	2.7648×10^4	47.7821	1.1085
76800	2.7648×10^6	8.0830	1.9117×10^{-2}
7680000	2.7648×10^8	1.4345	8.0156×10^{-3}

Table 1.3: Behavior of the errors when increasing the number of launched threads. Tests were performed with $n = 16$, $T_0 = 5$, $T_{min} = 0.5$, $\varphi = 0.7$, $L = 5$.

Threads	Function evaluations	$n = 16$		$n = 32$	
		Time	Speedup	Time	Speedup
128×64	9.3881×10^8	10.5519	122.1563	31.8715	73.4358
256×64	1.8776×10^9	15.5889	162.2502	57.0946	81.9460
256×128	3.7552×10^9	25.4284	203.3326	109.0251	85.9172
256×256	7.5105×10^9	46.4328	222.1904	211.5324	88.5479
256×512	1.5021×10^{10}	87.7999	235.5945	414.3638	90.5019

Table 1.4: Behavior of the speedup when increasing the number of launched threads. Tests were performed with $T_0 = 1000$, $T_{min} = 0.01$, $\varphi = 0.99$, $L = 100$.

reduced when we successively multiply by 100 the initial number of launched threads. Table 1.4 shows how the speedup increases when we multiply by 2 the initial number of launched threads, not only in the cases where the execution is not memory-bounded ($n = 16$), but also in the cases where it is memory-bounded ($n = 32$). Note that even in the memory-bounded case the obtained speedup is around 90.

Increasing the length of Markov chains

Table 1.5 shows the behavior of the speedup when successively doubling L, which denotes the length of the Markov chains, also for both cases $n = 16$ (not memory-bounded) and $n = 32$ (memory-bounded). Notice that the speedups are maintained even for large lengths of Markov chains.

Increasing the number of function evaluations

Table 1.6 shows how the speedup evolves when we increase the number of function evaluations by approximately successively doubling the initial value, also in both cases $n = 16$ (not memory-bounded) and $n = 32$ (memory-bounded). In practice,

L	Function evaluations	$n = 16$		$n = 32$	
		Time	Speedup	Time	Speedup
50	9.3881×10^8	9.3158	138.4039	30.1144	77.9015
100	1.8776×10^9	15.5889	162.2502	57.0946	81.9460
200	3.7552×10^9	28.4907	181.4357	111.2561	84.2041
400	7.5104×10^9	54.1433	191.1686	219.3096	85.6900
800	1.5021×10^{10}	105.4553	196.1659	435.2572	86.1849
1600	3.0042×10^{10}	208.1954	198.1213	869.2079	86.2000
3200	6.0083×10^{10}	413.3363	199.5752	1732.5052	86.5688

Table 1.5: Behavior of the speedup when increasing L. These tests were performed with the following configuration of Simulated Annealing, $T_0 = 1000$, $T_{min} = 0.01$, $\varphi = 0.99$, $b = 256$, $g = 64$.

Function evaluations	$n = 16$		$n = 32$	
	Time	Speedup	Time	Speedup
1.8776×10^9	15.6681	162.2502	60.1882	76.7356
3.7552×10^9	25.4315	203.0942	109.1611	85.6823
7.5105×10^9	47.8059	215.8074	215.0048	87.0605
1.5021×10^{10}	92.7941	222.3886	426.6933	87.6947
3.0042×10^{10}	182.6766	225.6187	850.9985	88.0599

Table 1.6: Behavior of the speedup when increasing the number of function evaluations.

the doubling of the number of function evaluations is achieved by different procedures: doubling the length of the Markov chain, doubling the number of launched threads, increasing the gap between the initial and the target minimum temperature or increasing the value of φ.

Double vs. Float

Table 1.7 shows that executions in double-precision are twice slower than in single one. Notice that in the best scenarios for the HPC versions of the Fermi architecture the double precision speed results to be limited to one half of the single-precision one. Obviously, the obtained error with double-precision is lower, but single-precision accuracy is enough because the purpose of the SA algorithm is to find an approximate minimum (see [88]). This is the reason why the results presented in all tables have been obtained with single-precision.

	Time	Relative error
Single-precision	15.5889	5.0686×10^{-5}
Double-precision	32.3916	2.1166×10^{-7}

Table 1.7: Computational times in seconds and relative quadratic errors with single and double-precision for the next Simulated Annealing configuration: $n = 16$, $T_0 = 1000$, $T_{min} = 0.01$, $\varphi = 0.99$, $b = 256$, $g = 64$.

1.4.2 The set of performed tests

In the previous section a particular minimization problem has been deeply analyzed. Furthermore, the proposed CUDA implementation for SA algorithm in GPUs has been tested against a large enough number of appropriate examples. A brief description of the different optimization problems that have been considered in the benchmark is listed in the Appendix A. The number and the kind of problems included in this appendix are chosen so that they are enough to obtain some conclusions from them and the test suite should not be overwhelming so that this study is unmanageable. Finally, the suite contains 41 examples. Table 1.8 lists these problems with the corresponding number of variables of each one. Moreover, the comparative analysis of results mainly focuses on the objective function values and in the locations of the solutions in the domain space obtained by the SA algorithm. Table 1.9 shows the obtained results, both for the asynchronous and synchronous versions. In Table 1.9 SA configurations that achieve small errors in the synchronous version are considered. Therefore, execution times become high for functions with a large number of parameters or with a large number of local minima. Since typically in many real applications the hybrid approaches (in which SA provides a starting point for a local minimization algorithm) are widely used [20], we present in Table 1.10 the obtained results using a hybrid strategy with Nelder-Mead as local minimizer. Both execution times and errors are much smaller when appropriately combining the SA and the local minimization algorithm.

1.5 Conclusions

The extremely long execution times associated to SA algorithm in its sequential version results to be its main drawback when applied to realistic optimization problems that involve high dimensional spaces or function evaluations with high computational cost. This is the reason why many authors in the literature have designed different alternatives to parallelize sequential SA by using different high-performance computing techniques. In the present chapter we have developed an efficient implementation of a SA algorithm by taking advantage of the power of GPUs. After analyzing a sequential SA version, a straightforward asynchronous and a synchronous implementations have been developed, the last one including an appropriate communication among Markov chains at each temperature level. The parallelization of the SA algorithm in GPUs has been discussed and the convergence of the different parallel techniques has been analyzed. Moreover, the parallel SA algorithm implementations have been checked by using classical experiments. A deeper analysis of results for a model example problem is detailed and the list of test examples defining the benchmark is included in the Appendix A. In summary, the results illustrate a better performance of the synchronous version in terms of convergence, accuracy and computational cost. Moreover, some results illustrate the behavior of the SA algorithm when combined with the Nedler-Mead local minimization method as an example of hybrid strategy that adequately balances accuracy and computational cost in real applications. The resulting code, called CUSIMANN, was leveraged in open source (see [152]).

Function f reference	Name of the problem	Dimension n
F0_a	Schwefel problem	8
F0_b	Schwefel problem	16
F0_c	Schwefel problem	32
F0_d	Schwefel problem	64
F0_e	Schwefel problem	128
F0_f	Schwefel problem	256
F0_g	Schwefel problem	512
F1_a	Ackley problem	30
F1_b	Ackley problem	100
F1_c	Ackley problem	200
F1_d	Ackley problem	400
F2	Branin problem	2
F3_a	Cosine problem	2
F3_b	Cosine problem	4
F4	Dekkers and Aarts problem	2
F5	Easom problem	2
F6	Exponential problem	4
F7	Goldstein and Price problem	2
F8_a	Griewank problem	100
F8_b	Griewank problem	200
F8_c	Griewank problem	400
F9	Himmelblau problem	2
F10_a	Levy and Montalvo problem	2
F10_b	Levy and Montalvo problem	5
F10_c	Levy and Montalvo problem	10
F11_a	Modified Langerman problem	2
F11_b	Modified Langerman problem	5
F12_a	Michalewicz problem	2
F12_b	Michalewicz problem	5
F12_c	Michalewicz problem	10
F13_a	Rastrigin problem	100
F13_b	Rastrigin problem	400
F14	Generalized Rosenbrock problem	4
F15	Salomon problem	10
F16	Six-Hump Camel Back problem	2
F17	Shubert problem	2
F18_a	Shekel 5 problem, $m = 5$	4
F18_b	Shekel 7 problem, $m = 7$	4
F18_c	Shekel 10 problem, $m = 10$	4
F19_a	Modified Shekel Foxholes problem	2
F19_b	Modified Shekel Foxholes problem	5

Table 1.8: Set of test problems, where first column indicates the assigned reference to display results.

f	Funct. eval.	V1			V2		
		Time	$\lvert f_a - f_r \rvert$	Error	Time	$\lvert f_a - f_r \rvert$	Error
F1_a	2.25×10^{9}	56.03	8.84×10^{-2}	9.79×10^{-2}	56.42	3.20×10^{-5}	4.56×10^{-5}
F1_b	2.25×10^{9}	221.36	9.45×10^{-1}	1.11	212.11	1.69×10^{-4}	4.26×10^{-4}
F1_c	2.63×10^{11}	52132.20	1.52×10^{-1}	3.95×10^{-1}	52916.78	1.93×10^{-4}	6.89×10^{-4}
F1_d	2.63×10^{11}	100481.20	3.34×10^{-1}	1.01	101754.88	3.90×10^{-4}	1.95×10^{-3}
F2	1.50×10^{9}	4.14	1.00×10^{-7}	9.90×10^{-4}	4.22	1.00×10^{-7}	1.09×10^{-4}
F3_a	1.87×10^{9}	4.18	1.00×10^{-6}	2.21×10^{-4}	4.32	1.00×10^{-7}	2.08×10^{-5}
F3_b	1.87×10^{9}	4.93	1.02×10^{-4}	3.01×10^{-3}	5.05	1.00×10^{-7}	3.63×10^{-5}
F4	1.87×10^{9}	4.38	3.20×10^{-3}	2.11×10^{-5}	4.53	4.20×10^{-4}	1.92×10^{-5}
F5	1.87×10^{9}	4.19	3.00×10^{-6}	2.54×10^{-4}	4.32	1.00×10^{-7}	4.15×10^{-5}
F6	2.25×10^{9}	4.39	9.00×10^{-6}	4.20×10^{-3}	4.63	1.00×10^{-7}	3.58×10^{-4}
F7	1.87×10^{9}	4.13	3.00×10^{-5}	1.16×10^{-4}	4.15	2.70×10^{-5}	3.46×10^{-5}
F8_a	3.37×10^{9}	674.93	1.11	2.16×10^{1}	666.02	1.00×10^{-7}	2.80×10^{-3}
F8_b	5.25×10^{9}	2102.46	1.59	4.87×10^{1}	2090.60	3.00×10^{-6}	2.69×10^{-2}
F8_c	3.37×10^{9}	2586.84	4.27	1.14×10^{2}	2536.51	5.43×10^{-3}	1.40
F9	1.87×10^{9}	3.90	2.00×10^{-6}	5.06×10^{-5}	4.08	1.00×10^{-7}	1.00×10^{-7}
F10_a	2.25×10^{9}	4.77	1.00×10^{-7}	4.45×10^{-4}	4.93	1.00×10^{-7}	3.28×10^{-7}
F10_b	2.25×10^{9}	6.83	3.20×10^{-5}	1.14×10^{-2}	6.97	1.00×10^{-7}	9.71×10^{-7}
F10_c	2.25×10^{9}	12.01	3.42×10^{-4}	3.78×10^{-2}	12.20	1.00×10^{-7}	6.60×10^{-6}
F11_a	3.75×10^{9}	10.37	1.04×10^{-4}	1.78×10^{-3}	10.43	1.00×10^{-6}	1.04×10^{-5}
F11_b	3.75×10^{9}	15.90	1.87×10^{-3}	5.26×10^{-3}	15.96	1.00×10^{-7}	1.80×10^{-5}
F12_a	1.87×10^{9}	6.18	1.00×10^{-7}	–	6.34	1.00×10^{-7}	–
F12_b	1.87×10^{9}	10.04	3.00×10^{-4}	–	10.21	1.00×10^{-7}	–
F12_c	1.87×10^{9}	16.96	5.30×10^{-3}	–	17.15	4.00×10^{-6}	–
F13_a	2.62×10^{10}	2517.40	6.36×10^{1}	5.61	2515.49	5.49×10^{-4}	2.56×10^{-3}
F13_b	2.63×10^{11}	108239.15	1.46×10^{2}	8.50	108113.13	9.52×10^{-3}	1.00×10^{-2}
F14	1.25×10^{9}	53.53	5.37	4.63	53.06	1.00×10^{-6}	1.11×10^{-3}
F15	3.39×10^{13}	52220.44	4.96×10^{-3}	1.35×10^{-2}	52008.19	1.00×10^{-7}	1.45×10^{-6}
F16	1.87×10^{9}	4.03	4.53×10^{-7}	4.35×10^{-4}	4.18	4.53×10^{-7}	6.53×10^{-5}
F17	1.87×10^{9}	5.87	1.00×10^{-7}	1.20×10^{-5}	5.92	1.00×10^{-7}	2.20×10^{-6}
F18_a	1.87×10^{9}	7.36	1.49×10^{-4}	1.36×10^{-4}	7.46	1.00×10^{-7}	2.00×10^{-5}
F18_b	1.87×10^{9}	8.89	1.25×10^{-4}	1.84×10^{-4}	9.00	1.00×10^{-7}	1.39×10^{-4}
F18_c	1.87×10^{9}	11.19	3.11×10^{-4}	3.30×10^{-4}	11.31	1.00×10^{-7}	1.47×10^{-4}
F19_a	1.87×10^{9}	17.09	1.00×10^{-7}	1.07×10^{-5}	17.58	4.00×10^{-6}	4.92×10^{-6}
F19_b	1.87×10^{9}	33.52	2.10×10^{-3}	2.89×10^{-4}	33.96	4.00×10^{-6}	4.61×10^{-6}

Table 1.9: Results for the test problem suite. In the column Error we indicate the relative error in $||\cdot||_2$ when the location of the minimum is non zero, otherwise the absolute error is presented. Cells with '-' mark correspond to cases in which the exact minima are unknown.

f	**V2**				**Hybrid**		
	Function eval.	Time	$\lvert f_a - f_r \rvert$	Error	Time	$\lvert f_a - f_r \rvert$	Error
F0_g	5.40×10^{7}	31.13	5.10	1.51×10^{-2}	2.24	2.10×10^{-12}	1.01×10^{-8}
F1_d	8.33×10^{7}	36.96	1.53	3.67	0.79	2.17×10^{-8}	1.50×10^{-12}
F8_c	9.01×10^{7}	81.26	1.38×10^{-1}	6.91	1.44	3.33×10^{-16}	1.08×10^{-6}
F13_b	3.47×10^{8}	165.57	2.36×10^{1}	3.45×10^{-1}	1.40	3.63×10^{-12}	2.44×10^{-7}

Table 1.10: Results of the hybrid algorithm. The first part shows the results of the annealing algorithm. The second one shows the results of the Nelder-Mead algorithm starting at the point at which the annealing algorithm was stopped prematurely.

Chapter 2

SABR models for equity

2.1 Introduction

Mathematical models have become of great importance in order to price financial derivatives on different underlying assets. However, in most cases there is no explicit solution to the governing equations, so that accurate robust fast numerical methods are required. Furthermore, financial models usually depend on many parameters that need to be calibrated to market data (market data assimilation). As in practice the model results are required almost at real time, the speed of numerical computations becomes critical and this calibration process must be performed as fast as possible.

In the classical Black-Scholes model [13], the underlying asset follows a lognormal process with constant volatility. However, in real markets the volatilities are not constant and they can vary for each maturity and strike (volatility surface). In order to overcome this problem, different local and stochastic volatility models have been introduced (see [37, 38, 67, 73, 78], for example). Here we consider the SABR model first proposed in [67], where a first order approximation formula for the implied volatility of European plain-vanilla options with short maturities is obtained. In [113] this formula is improved. In [116] a general method to compute a Taylor expansion of the implied volatility is described. In particular, a second order asymptotic SABR volatility formula is also computed. Next, in [130] a fifth order asymptotic expansion

is proposed for $\lambda-$SABR model; thus providing an extension with a mean-reversion term in [72].

The existence of a closed-form formula simplifies the calibration of the parameters to fit market data. However, when considering constant parameters (static SABR model), the volatility surface of a set of market data for several maturities cannot be suitably fitted. In [67, 137], the calibration of the static SABR model to fit a single volatility smile is analyzed. Among the different techniques to deal with a set of different maturities (see [47, 76], for example), in [67] a SABR model with time dependent parameters (dynamic SABR) is introduced and in [115] an asymptotic expression for the implied volatility is obtained. Also by means of piecewise constant parameters, in [52] the static SABR model is extended. In [94] a second order approximation to call options prices and implied volatilities is proposed and a closed form approximation of the option price extending dynamically the original SABR model is gained. In [85] the SABR model with a time-dependent volatility function and a mean reverting volatility process is obtained. In [25] a hybrid SABR$-$Hull-White model for long-maturity equity derivatives is considered.

However, time dependent parameters highly increase the computational cost and it is not always possible to compute an analytical approximation for the implied volatility or the prices (or the expression results to be very complex). In this case, we can use numerical methods (for example, Monte Carlo) in the calibration process. In order to calibrate a model, an efficient, robust and fast optimization algorithm has to be chosen, either a local optimization algorithm (such as Nelder-Mead or Levenberg-Marquardt ones) or a global optimization one (such as Simulated Annealing, genetic or Differential Evolution based ones). Although local optimization algorithms are efficient, if the calibration function presents several local minima they can stuck in any of these ones (note that the volatility surface can be uneven for some markets). On the other hand, global optimization algorithms are more robust, although they involve greater computational cost and are much slower. As financial instruments analysis should be carried out almost in real-time, we use the efficient implementation of the

Simulated Annealing (SA) algorithm in Graphics Processing Units (GPUs) proposed in the previous Chapter 1 and published in the article [43]. Particularly, we consider Nvidia Fermi GPUs and the API for its programming, named CUDA (Compute Unified Device Architecture), see [126, 109]. CUDA consists of some drivers for the graphic card, a compiler and a language that is basically a set of extensions for the C/C++ language. This framework allows to control the GPU (memory transfer operations, work assignment to the processors and threads synchronization).

Once the parameters have been calibrated, the model can be used to price exotic options. There are different techniques for pricing, such as Monte Carlo simulation, finite difference methods, binomial trees or integration-based methods (Fourier transform, for example). Among them, Monte Carlo simulation is a flexible and powerful tool that allows to price complex options (see [49, 84]). From the computational viewpoint, it results to be very expensive mainly due to its slow convergence. Once again this is a handicap, particularly in pricing and risk analysis in the financial sector. However, as illustrated in the present chapter, Monte Carlo simulation is suitable for pricing options on GPUs [95].

In this chapter, we have parallelized the Monte Carlo method on GPUs for the static and dynamic SABR models. In the literature the implementation of efficient Monte Carlo methods for pricing options has been analyzed. In [83] Asian options pricing with Black-Scholes model by a quasi-Monte Carlo method is considered, thus getting a speedup factor up to 150. In [10] a Monte Carlo GPU implementation for the multidimensional Heston and hybrid Heston−Hull-White models (using a hybrid Taus−Mersenne-Twister random number generator) is presented, achieving speedup factors around 50 for Heston model, and from 4 to 25 for the Heston−Hull-White model. In [131], European and American option pricing methods on GPUs under the static SABR model are presented. For the European case a speedup of around 100 is obtained, while for American ones it is around 10. In [132] the pricing of barrier and American options using a parallel version of least squares Monte Carlo algorithm is carried out, following the techniques presented in [131]. They obtain speedups up to

134 in the case of barrier options and around 22 in American ones. In our present chapter, the random number generation is performed "on the fly" by using the Nvidia CURAND library (see [110], for details). This is an important difference with previous works, where random numbers are previously generated and then transferred to the GPU global RAM memory.

The outline of this chapter is as follows. In Section 2.2 the static and dynamic SABR models are described, with a new proposal for functional parameters in the dynamic case. In Section 2.3 the calibration to market data is detailed. In Section 2.4, the Monte Carlo method and its parallel implementation in CUDA is described. In this chapter we use the SA method to calibrate the models. Depending on whether the cost function uses a direct expression or a Monte Carlo method, in Section 2.5 we discuss different techniques to parallelize the SA algorithm. In Section 2.6 we illustrate the performance of the implemented pricing technique for European call options. Next, we present results about calibration to real market data. Finally, the pricing of a cliquet option with the calibrated parameters is detailed.

Most of the contents presented in this chapter are included in reference [42].

2.2 The SABR model

The SABR (*Stochastic* α, β, ρ) model was introduced in [67], arguing that local volatility models were not able to reproduce market volatility smiles and that their predicted volatility dynamics contradicted market smiles and skews. The main advantage of the SABR model comes from its great simplicity compared to alternative stochastic volatility models [67]. The dynamics of the forward price and its volatility satisfy the following system of stochastic differential equations

$$dF(t) = \alpha(t)F(t)^{\beta}dW(t), \qquad F(0) = \hat{f}, \tag{2.1}$$

$$d\alpha(t) = \nu\alpha(t)dZ(t), \qquad \alpha(0) = \alpha, \tag{2.2}$$

where $F(t) = S(t)e^{(r-y)(T-t)}$ denotes the *forward* price of the underlying asset $S(t)$, r being the constant interest rate and y being the constant dividend yield. Moreover,

$\alpha(t)$ denotes the asset volatility process, dW and dZ are two correlated Brownian motions with constant correlation coefficient ρ (i.e. $dWdZ = \rho dt$) and $S(0)$ is the spot price of the asset. The parameters of the model are: $\alpha > 0$ (the volatility's reference level), $0 \leq \beta \leq 1$ (the variance elasticity), $\nu > 0$ (the volatility of the volatility) and ρ (the correlation coefficient). Note the two special cases: $\beta = 1$ (lognormal model) and $\beta = 0$ (normal model).

2.2.1 Static SABR model

The static SABR model corresponds to a constant parameters assumption. When working with options with the same maturity, the static SABR model provides good results [67]. The great advantage is that the following asymptotically approximated explicit formula for the implied Black-Scholes volatility can be obtained:

$$\sigma_{model}(K,\hat{f},T) = \frac{\alpha}{(K\hat{f})^{(1-\beta)/2}\left[1+\frac{(1-\beta)^2}{24}\ln^2\left(\frac{\hat{f}}{K}\right)+\frac{(1-\beta)^4}{1920}\ln^4\left(\frac{\hat{f}}{K}\right)+\cdots\right]}\cdot\left(\frac{z}{x(z)}\right)\cdot$$
$$\left\{1+\left[\frac{(1-\beta)^2}{24}\frac{\alpha^2}{(K\hat{f})^{1-\beta}}+\frac{1}{4}\frac{\rho\beta\nu\alpha}{(K\hat{f})^{(1-\beta)/2}}+\frac{2-3\rho^2}{24}\nu^2\right]\cdot T+\cdots\right\}, \tag{2.3}$$

where z is a function of K, $\hat{f}$ and T given by

$$z = \frac{\nu}{\alpha}(K\hat{f})^{(1-\beta)/2}\ln\left(\frac{\hat{f}}{K}\right),$$

and

$$x(z) = \ln\left(\frac{\sqrt{1-2\rho z+z^2}+z-\rho}{1-\rho}\right). \tag{2.4}$$

In this chapter, we consider the following correction to (2.3) proposed by Oblój in [113],

$$\sigma_{model}(K,\hat{f},T) = \frac{1}{\left[1+\frac{(1-\beta)^2}{24}\ln^2\left(\frac{\hat{f}}{K}\right)+\frac{(1-\beta)^4}{1920}\ln^4\left(\frac{\hat{f}}{K}\right)+\cdots\right]}\cdot\left(\frac{\nu\ln\left(\frac{\hat{f}}{K}\right)}{x(z)}\right)\cdot$$

$$\left\{1+\left[\frac{(1-\beta)^2}{24}\frac{\alpha^2}{(K\hat{f})^{1-\beta}}+\frac{1}{4}\frac{\rho\beta\nu\alpha}{(K\hat{f})^{(1-\beta)/2}}+\frac{2-3\rho^2}{24}\nu^2\right]\cdot T+\cdots\right\}, \tag{2.5}$$

where the following new expression for z is considered:

$$z=\frac{\nu\left(\hat{f}^{1-\beta}-K^{1-\beta}\right)}{\alpha(1-\beta)},$$

and $x(z)$ is given by (2.4). The omitted terms after $+\cdots$ can be neglected, so that (2.5) turns to

$$\sigma_{model}(K,\hat{f},T)=\frac{1}{\omega}\left(1+A_1\ln\left(\frac{K}{\hat{f}}\right)+A_2\ln^2\left(\frac{K}{\hat{f}}\right)+BT\right), \tag{2.6}$$

where the coefficients A_1, A_2 and B are given by

$$\begin{aligned}
A_1 &= -\frac{1}{2}(1-\beta-\rho\nu\omega),\\
A_2 &= \frac{1}{12}\Big((1-\beta)^2+3\big((1-\beta)-\rho\nu\omega\big)+\big(2-3\rho^2\big)\nu^2\omega^2\Big),\\
B &= \frac{(1-\beta)^2}{24}\frac{1}{\omega^2}+\frac{\beta\rho\nu}{4}\frac{1}{\omega}+\frac{2-3\rho^2}{24}\nu^2,
\end{aligned}$$

and the value of ω is given by $\omega=\alpha^{-1}\hat{f}^{1-\beta}$.

2.2.2 Dynamic SABR model and the choice of the functional parameters

The main drawback of the static SABR model arises when market data for options with several maturities are considered. In this case, too large errors could appear.

In order to overcome this problem, the following dynamic SABR model allows time dependency in some parameters [67]:

$$dF(t) = \alpha(t)F(t)^{\beta}dW(t), \qquad F(0) = \hat{f}, \tag{2.7}$$

$$d\alpha(t) = \nu(t)\alpha(t)dZ(t), \qquad \alpha(0) = \alpha, \tag{2.8}$$

where the correlation coefficient ρ is also time dependent. As in the static SABR model, the dynamic one also provides the following expression to approximate the implied volatility [115]:

$$\sigma_{model}(K, \hat{f}, T) = \frac{1}{\omega}\left(1 + A_1(T)\ln\left(\frac{K}{\hat{f}}\right) + A_2(T)\ln^2\left(\frac{K}{\hat{f}}\right) + B(T)T\right), \tag{2.9}$$

where

$$\begin{aligned} A_1(T) &= \frac{\beta - 1}{2} + \frac{\eta_1(T)\omega}{2}, \\ A_2(T) &= \frac{(1-\beta)^2}{12} + \frac{1 - \beta - \eta_1(T)\omega}{4} + \frac{4\nu_1^2(T) + 3\left(\eta_2^2(T) - 3\eta_1^2(T)\right)}{24}\omega^2, \\ B(T) &= \frac{1}{\omega^2}\left(\frac{(1-\beta)^2}{24} + \frac{\omega\beta\eta_1(T)}{4} + \frac{2\nu_2^2(T) - 3\eta_2^2(T)}{24}\omega^2\right), \end{aligned}$$

with

$$\begin{aligned} &\nu_1^2(T) = \frac{3}{T^3}\int_0^T (T-t)^2\nu^2(t)dt, \quad \nu_2^2(T) = \frac{6}{T^3}\int_0^T (T-t)t\nu^2(t)dt, \\ &\eta_1(T) = \frac{2}{T^2}\int_0^T (T-t)\nu(t)\rho(t)dt, \quad \eta_2^2(T) = \frac{12}{T^4}\int_0^T\int_0^t \left(\int_0^s \nu(u)\rho(u)du\right)^2 dsdt. \end{aligned} \tag{2.10}$$

Note that if ν and ρ are taken as constants, i.e. $\nu = \nu_0$ and $\rho = \rho_0$, then it follows that $\nu_1(T) = \nu_2(T) = \nu_0$, $\eta_1(T) = \eta_2(T) = \nu_0\rho_0$ and the dynamic SABR model reduces to the static one.

The choice of the functions ρ and ν in (2.10) constitutes a very important decision. The values of $\rho(t)$ and $\nu(t)$ have to be smaller for long terms (t large) rather than for short terms (t small). Thus, in this chapter we consider two possibilities with exponential decay:

- **Case I**: It is more classical and corresponds to the choice

$$\rho(t) = \rho_0 e^{-at}, \quad \nu(t) = \nu_0 e^{-bt}, \tag{2.11}$$

with $\rho_0 \in [-1,1]$, $\nu_0 > 0$, $a \geq 0$ and $b \geq 0$. In this case, the expressions of the functions ν_1^2, ν_2^2, η_1 and η_2^2, defined by (2.10), can be exactly calculated and are given by:

$$\nu_1^2(T) = \frac{6\nu_0^2}{(2bT)^3}\left[\big((2bT)^2/2 - 2bT + 1\big) - e^{-2bT}\right],$$

$$\nu_2^2(T) = \frac{6\nu_0^2}{(2bT)^3}\left[2(e^{-2bT} - 1) + 2bT(e^{-2bT} + 1)\right],$$

$$\eta_1(T) = \frac{2\nu_0\rho_0}{T^2(a+b)^2}\left[e^{-(a+b)T} - \big(1 - (a+b)T\big)\right],$$

$$\eta_2^2(T) = \frac{3\nu_0^2\rho_0^2}{T^4(a+b)^4}\left[e^{-2(a+b)T} - 8e^{-(a+b)T} + \Big(7 + 2(a+b)T\big(-3 + (a+b)T\big)\Big)\right]. \tag{2.12}$$

- **Case II**: In the present chapter we propose the original and more general choice

$$\rho(t) = (\rho_0 + q_\rho t)e^{-at} + d_\rho, \quad \nu(t) = (\nu_0 + q_\nu t)e^{-bt} + d_\nu. \tag{2.13}$$

  In this case, the symbolic software package Mathematica allows to calculate exactly the functions ν_1^2, ν_2^2 and η_1 (see Appendix B.1). However, an explicit expression for η_2^2 cannot be obtained, and therefore we use an appropriate quadrature formula for its approximation.

  In order to guarantee that the correlation $\rho(t) \in [-1,1]$ and the volatility $\nu(t) > 0$ for the involved parameters, an adequate optimization algorithm has to be used during calibration.

2.3 Calibration of the SABR model

The goal of calibration is to fit the model parameters to reproduce the market prices or volatilities as close as possible. In order to calibrate the model, we choose a set of vanilla options on the same underlying asset and the market prices are collected at

the same moment. We can either consider only one maturity or several maturities. In the second case, a dynamic SABR model should be applied. Next, the model is used to price other options (such as exotic options). The computational cost increases with the increasing complexity of the pricing models.

Hereafter we denote by $Data_{market}(K_j, \hat{f}, T_i)$ the observed market data, for the maturity T_i $(i = 1, \ldots, n)$ and the strike K_j $(j = 1, \ldots, m_i)$, where n denotes the number of maturities and m_i the number of strikes for the maturity T_i. The market implied volatility is denoted by $\sigma_{market}(K_j, \hat{f}, T_i)$ and the corresponding market option price by $V_{market}(K_j, \hat{f}, T_i)$. Moreover, we denote by $Data_{model}(K_j, \hat{f}, T_i)$ the model value ($V_{model}(K_j, \hat{f}, T_i)$ or $\sigma_{model}(K_j, \hat{f}, T_i)$) for the same option.

The calibration process tries to obtain a set of model parameters that minimizes the error between market and model values for a given error measure. In order to achieve this target we must follow several steps:

- Decide how to perform the calibration process, in prices or in volatilities.
- Choose market data that should be highly representative of the market situation.
- Decide which error measure will be used to compare model and market values:
  - If the calibration is made for one maturity T_i, we use the cost function

$$f_{i,E}(\boldsymbol{x}) = \sum_{j=1}^{m_i} \left(\frac{Data_{market}(K_j, \hat{f}, T_i) - Data_{model}(K_j, \hat{f}, T_i)}{Data_{market}(K_j, \hat{f}, T_i)} \right)^2 (\boldsymbol{x}), \quad (2.14)$$

    where $\boldsymbol{x}$ denotes the parameters to calibrate.
  - If the calibration is made for a set of maturity dates the cost function we use is

$$f_E(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \left(\frac{Data_{market}(K_j, \hat{f}, T_i) - Data_{model}(K_j, \hat{f}, T_i)}{Data_{market}(K_j, \hat{f}, T_i)} \right)^2 (\boldsymbol{x}). \quad (2.15)$$

- Choose the (local or global) optimization algorithm to minimize the error.

- Fix (if it is convenient) some of the parameters on beforehand, by taking into account the previous experience or the existing information.

- Calibrate and compare the obtained results. If they are satisfactory, the parameters are accepted and used for pricing more complex financial instruments.

An advantage of the SABR model is the existence of an asymptotic approximation formula for the implied volatility that can be used in the calibration. It is important to take into account the meaning of the different model parameters [67, 137]. The value of β is related to the type of the underlying stochastic process of the model and it is usually fixed on beforehand. For example, $\beta = 1$ (lognormal model) is mostly used in equity and currency markets, like foreign exchange markets (for example, EUR/USD). The choice $\beta = 0.5$ corresponds to a CIR model and is used in US interest rate desks. The value $\beta = 0$ (normal model) is commonly used to manage risks, as for example in Yen interests rate markets. Alternatively, β can be computed from historical data of the at-the-money volatilities, $\sigma_{ATM} = \sigma_{model}(\hat{f}, \hat{f}, T)$, that can be obtained by taking $K = \hat{f}$ in (2.5). Next, after some computations the identity

$$\ln \sigma_{ATM} = \ln \alpha - (1 - \beta) \ln \hat{f},$$

is obtained and β can be computed from a log-log regression of $\hat{f}$ and σ_{ATM} (see [137], for details). Once β has been fixed and the term BT in (2.5) has been neglected, the value of α can be computed by using the value of σ at-the-money ($\sigma_{ATM} = \sigma_{model}(\hat{f}, \hat{f}, T)$) with the following approximation:

$$\alpha \approx \hat{f}^{(1-\beta)} \sigma_{model}(\hat{f}, \hat{f}, T).$$

In the case $\beta = 1$, α is equal to the at-the-money volatility [67]. Thus, by fixing $\beta = 1$ and α, the (possibly large) calibration cost is reduced: only ρ and ν are calibrated.

In this chapter the whole set of SABR parameters is calibrated, thus leading to a more time consuming calibration. Nevertheless, the GPUs technology highly speeds up this procedure.

2.4 Pricing with Monte Carlo using GPUs

Monte Carlo technique for the SABR model involves the simulation of a huge number of forward and volatility paths. Let $S_0 = S(0)$ and $F_0 = F(0)$ be the asset spot value and the initial value of the forward, respectively. For a given option data and a given initial value of the forward $F_0 = \hat{f} = S_0 e^{(r-y)(T-t)}$, the European option price at the time $t = 0$ is equal to $V(S_0, K) = e^{-rT}\mathbb{E}\big(V(S_T, K)\big)$, where $V(S_T, K)$ is the option payoff (for example, $V(S_T, K) = \max\{S_T - K, 0\}$ for a European call option).

In order to discretize the stochastic differential equations of the SABR model, we take a constant time step Δt, such that $M = \frac{T}{\Delta t}$ denotes the number of subintervals in $[0, T]$. In order to preserve some properties of the continuous model (such as positivity), we use the following log-Euler discretization:

$$\begin{aligned} \alpha_{i+1} &= \alpha_i e^{\nu(t_i) Z_i^1 \sqrt{\Delta t} - \nu^2(t_i)\Delta t/2}, \\ \hat{\nu} &= \alpha_i F_i^{\beta-1}, \\ F_{i+1} &= F_i e^{\hat{\nu}\left(\rho(t_i) Z_i^1 + Z_i^2 \sqrt{1-\rho^2(t_i)}\right)\sqrt{\Delta t} - \hat{\nu}^2 \Delta t/2}, \end{aligned} \tag{2.16}$$

where Z_i^1 and Z_i^2 denote two independent standard normal distribution samples. In [26] it is pointed out that the log-Euler scheme may become unstable for specific time-steps. So, the simulation of the conditional distribution of the SABR model over a discrete time step (which is proved to be a transformed squared Bessel process) and the use of an approximation formula for the integrated variance are proposed. However, we did not find this unstable behavior in our experiments. In case of unstability, this alternative low-bias simulation scheme in [26] could be adapted.
Note that $S_T = F_T$. If N denotes the number of simulated paths, then the option price is given by

$$\widehat{V}(S_0, K) = e^{-rT} \frac{1}{N} \sum_{j=1}^{N} V(S_T^j, K). \tag{2.17}$$

Option pricing with the Monte Carlo method is an accurate and robust technique. However, as Monte Carlo method exhibits an order of convergence equal to $1/\sqrt{N}$, a large number of paths is required to obtain precise results, therefore leading to

not affordable computational costs in real practice. For this reason, option pricing with Monte Carlo is usually implemented in high performance systems. Here, we propose a parallel version of Monte Carlo efficiently implemented in CUDA by using the XORWOW pseudo-random number generator (xorshift RNG) included in the Nvidia CURAND library [101].

The process can be summarized as follows:

1. Firstly we read the input data and send it to constant memory: Monte Carlo parameters, SABR parameters and market data. Next, in order to store the computed final price of each simulation, we allocate a global memory vector of size N.

2. We compute a Monte Carlo kernel for the generation of the different paths, where uniform random numbers are generated "on the fly" inside this kernel, calling the `curand_uniform()` CURAND function. This allows random numbers to be generated and used by the Monte Carlo kernel without requiring them to be written to and then read from global memory. Then Box-Muller method is used to transform the random uniform distributed numbers in normally distributed ones.

3. The option price is computed with the `thrust::transform_reduce` method of the Thrust Library [147] (included in the Nvidia toolkit since its version 4.0). With this function we apply kernel fusion reduction kernels. Then, the sum in (2.17) is computed by a standard `plus` reduction also available in the Thrust Library. By fusing the payoff operation with the reduction kernel we have a highly optimized implementation which offers the same performance as hand-written kernels. All these operations are performed inside the GPU, so that transfers to the CPU memory are avoided.

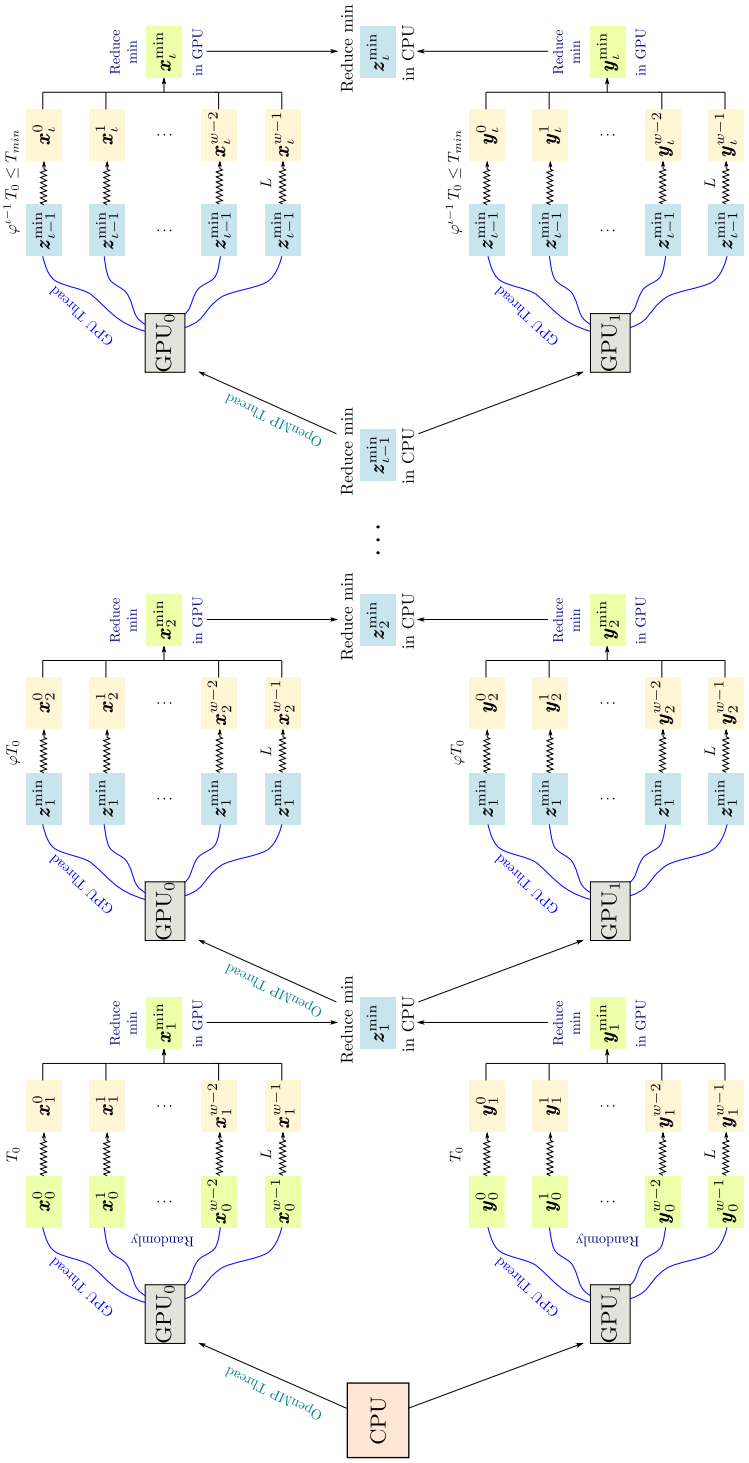


Figure 2.1: Sketch of the parallel SA algorithm using two GPUs and OpenMP.

2.5 Calibration of the parameters using GPUs

The calibration of the SABR model parameters can be done using the implied volatility formula or the Monte Carlo simulation method. Usually, in trading environments the second one is not used, mainly due to its high execution times. However, if we have a parallel and efficient implementation of the Monte Carlo method, we can consider its usage in the calibration procedure.

In this chapter, the calibration of the parameters has been done with the Simulated Annealing (SA) stochastic global optimization method presented in the Chapter 1.

In this chapter, we propose the two calibration techniques described in next paragraphs.

2.5.1 Calibration with Technique I

In this calibration technique we use the implied volatility formula and an efficient implementation in GPU of the Simulated Annealing algorithm. Thus, we consider $Data_{market} = \sigma_{market}$ and $Data_{model} = \sigma_{model}$ in the cost functions (2.14) and (2.15), respectively, both for individual and joint calibrations. Note that σ_{market} denotes the market quoted implied volatilities and σ_{model} the model ones, (2.6) and (2.9) for the static and dynamic models, respectively.

The calibration of the parameters has been done using the CUSIMANN library [152] which implements the synchronous version detailed in the Section 1.2.2 of the Chapter 1 (see Figure 1.2).

A host can have several discrete graphic cards attached. When dealing with computationally expensive processes, as in this case, it is interesting to use the computing power of these extra GPUs to further reduce the execution times. One possible approach is to use OpenMP [149] to control the GPUs inside one node. The strategy is to launch as many CPU threads as GPUs available on a node. Thus, each CPU thread handles a GPU.

The idea of this new synchronous parallel version of SA considering OpenMP and

several GPUs is the following; by simplicity we will consider two GPUs (see Figure 2.1): in each GPU the algorithm starts from a fixed initial point, and each thread runs independently a Markov chain of constant length L until reaching the next level of temperature. As the temperature is fixed, each thread actually performs a Metropolis process. Once all the threads have finished, in each GPU a reduction operation is made. Then, each GPU reports the corresponding final state $\boldsymbol{x}_*^{\min}$ and $\boldsymbol{y}_*^{\min}$, and in the CPU another reduction operation is done to get the minimum point $\boldsymbol{z}_*^{\min}$. This point is used as starting point for all GPUs threads at the following temperature level.

Note that if we had a cluster of GPUs, we could exploit an additional level of parallelism. The final configuration which merges all the ideas is to use MPI framework [148] for intercommunication between the nodes of a GPU cluster and to take advantage of OpenMP to control the several graphic cards located inside this node. When we double the number of GPUs used, the benefit of this approach is that the execution times are reduced by about a half, or, from another point of view, we could double the number of function evaluations "without" increasing the runtime. Indeed, the time reduction would be somewhat less than a half because this extra parallelization involves a small additional cost.

2.5.2 Calibration with Technique II

In this calibration technique the cost function is computed in GPU by a Monte Carlo method. It turns out to be necessary when an expression for the implied volatility is not available, as in the dynamic SABR model when considering the **Case II** (see Section 2.2.2). For the joint calibration, in the cost function (2.15) we consider $Data_{market} = V_{market}$ and $Data_{model} = V_{model}$; V_{market} being the option market price and V_{model} being the computed price with the Monte Carlo parallel implementation (Section 2.4).

As the Monte Carlo method is carried out inside the GPU and Fermi GPUs do not allow nesting kernels, the SA minimization algorithm has to be run on CPU. In order to use all available GPUs in the system, we propose a CPU SA parallelization

using OpenMP. So, each OpenMP SA thread uses a GPU to assess on the Monte Carlo objective function. This approach can be easily extrapolated to a cluster of GPUs using, for example, MPI. Particularly, in this case the SA procedure rejects all points violating the constraints over $\rho(t)$ and $\nu(t)$, i.e. $\rho(t) \in [-1, 1]$ and $\nu(t) > 0$, $\forall t$ $0 < t \leq T$.

2.6 Numerical results

From now on we denote by `SSabr` and `DSabr` the static and dynamic SABR models, respectively. As detailed in Section 2.2.2 concerning the dynamic SABR model, depending on the choice of the time dependent functions ρ and ν, we denote by:

- `DSabr_I`: dynamic SABR model in the **Case I**, where ρ and ν are given by (2.11).
- `DSabr_II`: dynamic SABR model in the **Case II** (general case), where ρ and ν are given by (2.13).

Moreover, depending on the technique used to calibrate the model (see Section 2.5), we use the notation:

- `T_I`: **Technique I**, where the model parameters are calibrated with the implied volatility formula and using the efficient implementation of Simulated Annealing in GPU.
- `T_II`: **Technique II**, where the cost function is evaluated by the efficient implementation in GPUs of the Monte Carlo pricing method, as detailed in Section 2.4.

Numerical experiments have been performed with the following hardware and software configurations: GPUs Nvidia Geforce GTX470, a CPU Xeon E5620 clocked at 2.4 Ghz with 16 GB of RAM, CentOS Linux, Nvidia CUDA SDK 4.0 and GNU C/C++ compilers 4.1.2.

2.6.1 Pricing European options

In this section we focus on pricing European options with both SABR models, when using the GPU Monte Carlo method. The fixed data are $S_0 = 2257.37$, $K = 2257.37$, $r = 0.018196$, $y = 0.034516$, $T = 0.495890$, while the SABR parameters are:

- `SSabr`: $\alpha = 0.375162$, $\beta = 0.999999$, $\nu = 0.331441$ and $\rho = -0.999999$.
- `DSabr_I`: $\alpha = 0.393329$, $\beta = 1.0$, $\nu_0 = 0.941565$, $\rho_0 = -1.0$, $a = 0.001$ and $b = 1.246906$.
- `DSabr_II`: $\alpha = 0.398436$, $\beta = 0.999579$, $\nu_0 = 1.285129$, $\rho_0 = -0.964678$, $a = 0.0$, $b = 2.059560$, $d_\rho = 0.101632$, $d_\nu = -0.086294$, $q_\rho = 0.0$ and $q_\nu = 1.302296$.

In Table 2.1 the results for the three presented models are shown. They correspond to 2^{20} simulations, $\Delta t = 1/250$ (123 time steps) and both single and double precision computations. In Table 2.2 the execution times for CPU an GPU programs are compared. In single precision, the maximum speedup varies, approximately, from 357 to 567 times depending on the model. In double precision, the maximum speedup is around of 74 times in all models. Table 2.3 shows the variation of execution times when increasing the number of strikes, considering the `DSabr_II` model with $\Delta t = 1/250$. Note that execution times hardly vary: pricing 41 options by generating 2^{20} paths takes around 0.28 and 0.86 seconds in single and double precision, respectively, while pricing one option takes 0.25 and 0.84 seconds. Therefore, Monte Carlo pricing method results to be affordable for calibration.

Precision	`SSabr`		`DSabr_I`		`DSabr_II`	
	Result	`RE`	Result	`RE`	Result	`RE`
Single	224.544601	0.005944	222.432648	0.015294	224.652390	0.005467
Double	224.545954	0.005938	222.434009	0.015288	224.653642	0.005461

Table 2.1: Pricing results for European options. `RE` denotes the relative error with respect to the reference value 225.887329.

SSabr						
Paths	Single			Double		
	CPU	GPU	Speedup	CPU	GPU	Speedup
2^{16}	2.890	0.142	×20.313	2.762	0.175	×15.783
2^{20}	46.287	0.206	×223.726	44.142	0.723	×61.054
2^{24}	721.592	1.271	×567.373	704.617	9.474	×74.373

DSabr_I						
Paths	Single			Double		
	CPU	GPU	Speedup	CPU	GPU	Speedup
2^{16}	3.638	0.144	×25.112	3.358	0.179	×18.759
2^{20}	58.228	0.255	×227.674	53.663	0.840	×63.884
2^{24}	929.254	2.047	×453.918	860.556	11.515	×74.733

DSabr_II						
Paths	Single			Double		
	CPU	GPU	Speedup	CPU	GPU	Speedup
2^{16}	3.694	0.145	×25.388	3.392	0.182	×18.637
2^{20}	59.790	0.287	×207.762	54.144	0.853	×64.474
2^{24}	949.459	2.658	×357.095	868.156	11.590	×74.905

Table 2.2: Pricing European options. Execution times for CPU and GPU calibration (in seconds), considering single and double precision with $\Delta t = 1/250$.

2.6.2 Calibration

In order to check the accuracy and performance of the GPU calibration code, the calibration of the SABR model to the volatility surfaces generated by the EURO STOXX 50 index and EUR/USD foreign exchange rate has been tested. For both data, we detail the calibrated parameters for `SSabr`, `DSabr_I` and `DSabr_II` models, the computational times and the comparisons between the market volatilities or prices and those ones with the model after parameter calibration.

Num. Strikes	2^{16}		2^{20}		2^{24}	
	Single	Double	Single	Double	Single	Double
1	0.134374	0.183608	0.259048	0.841376	2.234662	11.595257
5	0.135871	0.184051	0.262661	0.850931	2.242463	11.605406
41	0.157753	0.190635	0.280509	0.862674	2.265047	11.645788

Table 2.3: Pricing with `DSabr_II` model in GPU. Influence of the number of strikes in computational times (time in seconds), $\Delta t = 1/250$. We consider 1, 5 and 41 strikes, with values K (in % of S_0) of $\{100\}$, $\{96, 98, 100, 102, 104\}$ and $\{80, 81 \ldots, 119, 120\}$, respectively.

EURO STOXX 50 index

The data correspond to EURO STOXX 50 quotes of December of 2011. The asset spot value is 2311.1 €. In Tables B.1 and B.2 of the Appendix B.2, the interest rates, dividend yields and implied volatilities for 3, 6, 12 and 24 months maturities are shown. Next, we present the calibrated parameters for these data.

1. **Calibration of the `SSabr` model using the technique `T_I`**

   By using the technique `T_I` and the asymptotic expression (2.6) in the cost function, the individual calibration (all strikes and one maturity) of the `SSabr` model has been carried out. In Table 2.4 the calibrated parameters are detailed. For each maturity (3, 6, 12 and 24 months), Figure 2.2 shows market and model volatilities with the parameters of Table 2.4.

	3 months	6 months	12 months	24 months
α	0.298999	0.302060	0.289271	0.277844
β	1.0	1.0	1.0	1.0
ν	0.382558	0.381724	0.308560	0.264178
ρ	-1.0	-1.0	-0.999729	-1.0

Table 2.4: EURO STOXX 50. `SSabr` model: Calibrated parameters for each maturity.

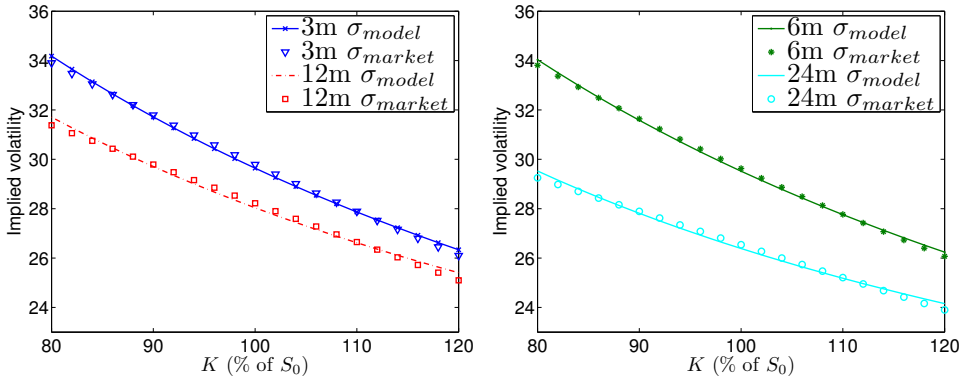


Figure 2.2: EURO STOXX 50. `SSabr` model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

Table 2.5 shows the computational times and the speedups of the calibration

process for $T = 24$ months. As expected, the speedup is near 7.7 when using OpenMP with 8 threads, while the speedup respect to the CPU code is around 190 when we use one GPU. Furthermore, if we use two GPUs then the computational time is additionally almost divided by two.

	CPU (1 thread)	2 threads	4 threads	8 threads	GPU	2 GPUs
Time (s)	4172.746	2096.857	1056.391	545.456	21.955	12.609
Speedup	-	1.99	3.95	7.65	190.06	330.93

Table 2.5: EURO STOXX 50. `SSabr` model: Performance of OpenMP vs. GPU versions, in single precision for $T = 24$ months.

2. **Calibration of the `DSabr_I` model using the technique `T_I`**

   The joint calibration for all strikes and maturities, using the `DSabr_I` model is made with the GPU version of SA and using the asymptotic formula (2.9), when ρ and ν are given by (2.11). In Table 2.6 the calibrated parameters are detailed. In Figure 2.3, the whole volatility surface for all maturities is shown. By using the parameters in Table 2.6, for several strikes, Table 2.7 shows the market volatilities (σ_{market}) vs. the model ones (σ_{model}, computed with formulas (2.9), (2.11) and (2.12)). For single precision, the maximum relative error is 7.608205×10^{-2} and the mean relative error is 2.073025×10^{-2}.

$\alpha = 0.294722$	$\beta = 1.0$	$\rho_0 = -1.0$	$\nu_0 = 0.388539$	$a = 0.001000$	$b = 0.131466$

Table 2.6: EURO STOXX 50. `DSabr_I` model: Calibrated parameters.

   In Table 2.8, the performance of the calibration procedure is illustrated. The speedup of the mono-GPU version is around 225, while the 2 GPUs version achieves a speedup up to 421.

3. **Calibration of the `DSabr_II` model using the technique `T_II`**

   An asymptotic expression of implied volatility for the `DSabr_II` model is not available. So, for its calibration we use the technique `T_II` (see Section 2.5). The calibration process is performed in prices. For the Monte Carlo pricing method

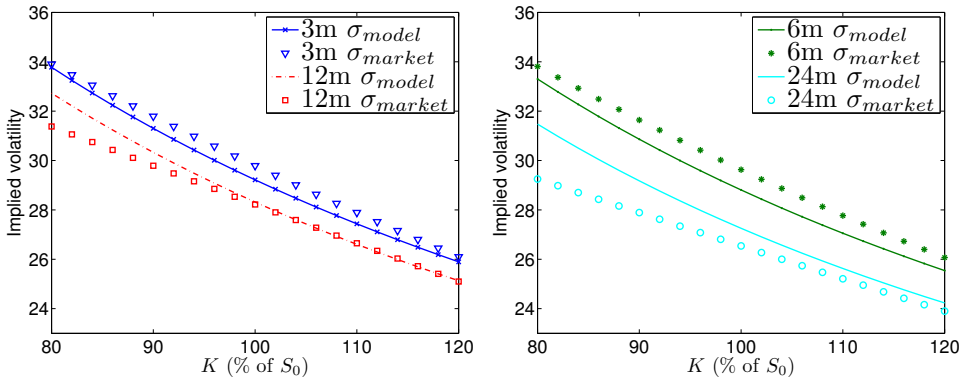


Figure 2.3: EURO STOXX 50. `DSabr_I` model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

K (% of S_0)	3 months			6 months		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
88%	32.21	31.7628	1.388389×10^{-02}	32.07	31.3150	2.354225×10^{-02}
100%	29.79	29.2166	1.924807×10^{-02}	29.63	28.8068	2.778265×10^{-02}
112%	27.52	27.1094	1.492006×10^{-02}	27.42	26.7345	2.500000×10^{-02}
K (% of S_0)	12 months			24 months		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
88%	30.11	30.7756	2.210561×10^{-02}	28.16	29.6026	5.122869×10^{-02}
100%	28.22	28.3187	3.497519×10^{-03}	26.54	27.2549	2.693670×10^{-02}
112%	26.34	26.2941	1.742597×10^{-03}	24.95	25.3308	1.526253×10^{-02}

Table 2.7: EURO STOXX 50. `DSabr_I` model: σ_{market} vs. σ_{model}.

we have considered 2^{20} paths with $\Delta t = 1/250$. In Table 2.9 the calibrated parameters are detailed. In this case, double precision computations have been used to ensure the convergence to the real minimum.

In Figure 2.4, the whole prices surface at maturities 3, 6, 12 and 24 months is shown. In Table 2.10 the market and model prices are compared. The mean relative error is 1.741038×10^{-2} and the maximum relative error is 5.344231×10^{-2}.

In double precision, the computational time with one GPU is 37709.57 seconds and with 2 GPUs is 19520.73 seconds, getting a speedup around 1.93. In this

	CPU (1 thread)	2 threads	4 threads	8 threads	GPU	2 GPUs
Time (s)	20162.40	10089.16	5116.84	2643.87	89.95	47.81
Speedup	-	1.99	3.94	7.63	224.15	421.72

Table 2.8: EURO STOXX 50. `DSabr_I` model: Performance of OpenMP vs. GPU versions, in single precision.

$\alpha = 0.296790$	$\beta = 1.000000$	$\rho_0 = -0.360610$	$\nu_0 = 0.000100$	$a = 15.0000000$
$b = 15.000000$	$d_\rho = -0.715716$	$d_\nu = 0.847244$	$q_\rho = 15.000000$	$q_\nu = -8.969205$

Table 2.9: EURO STOXX 50. `DSabr_II` model: Calibrated parameters.

case the CPU computation cost results prohibitive. As expected, the computational time of the calibration process with the Monte Carlo pricing method is much higher than with the previous calibration procedures.

`DSabr_I` vs. `DSabr_II`

In order to compare the accuracy of `DSabr_II` and `DSabr_I` models, we compute the mean relative error of the `DSabr_I` model in prices, with Black-Scholes formula applied to σ_{model}. This error is 2.154846×10^{-2}, so that `DSabr_II` model captures better the market dynamics.

EUR/USD exchange rate

In this section we consider the EUR/USD exchange rate, that expresses the amount of American Dollars equivalent to one Euro. In Tables B.3 and B.4 of the Appendix

K (% of S_0)	3 months			6 months		
	V_{market}	V_{model}	$\frac{\lvert V_{market} - V_{model} \rvert}{V_{market}}$	V_{market}	V_{model}	$\frac{\lvert V_{market} - V_{model} \rvert}{V_{market}}$
88%	316.679	316.119	1.769641×10^{-03}	347.371	346.830	1.556418×10^{-03}
100%	134.605	132.952	1.227935×10^{-02}	180.353	177.429	1.621589×10^{-02}
112%	37.252	36.895	9.572343×10^{-03}	74.680	72.682	2.676232×10^{-02}
K (% of S_0)	12 months			24 months		
	V_{market}	V_{model}	$\frac{\lvert V_{market} - V_{model} \rvert}{V_{market}}$	V_{market}	V_{model}	$\frac{\lvert V_{market} - V_{model} \rvert}{V_{market}}$
88%	403.205	416.516	3.301443×10^{-02}	463.037	482.939	4.298152×10^{-02}
100%	245.905	251.015	2.077787×10^{-02}	316.081	322.821	2.132105×10^{-02}
112%	132.454	133.687	9.304021×10^{-03}	201.189	200.374	4.048420×10^{-03}

Table 2.10: EURO STOXX 50. `DSabr_II` model: V_{market} vs. V_{model}.

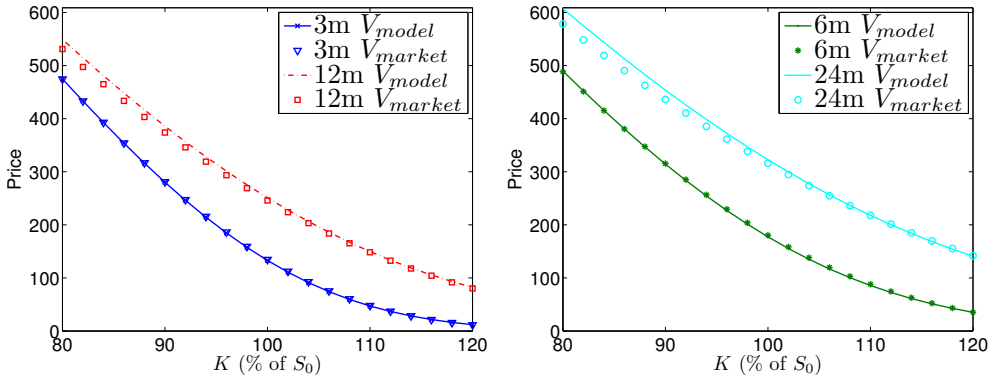


Figure 2.4: EURO STOXX 50. `DSabr_II` model: V_{model} vs. V_{market} for the whole prices surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

B.2 the interests rates, dividend yields and volatility smiles for 3, 6, 12 and 24 months maturities are shown. The EUR/USD *spot* rate is $S_0 = 1.2939$ US dollars quoted in December of 2011. From now on we denote the EUR/USD foreign rate as EURUSD. Next, we present the results of calibrating the introduced models to these data.

1. **Calibration of the `SSabr` model using the technique `T_I`**

   By using the technique `T_I` and the asymptotic expression (2.6), the individual calibration of the `SSabr` model has been carried out. The parameters in Table 2.11 have been obtained. For them, in Figure 2.5 the market and the model volatilities are shown.

	3 months	6 months	12 months	24 months
α	0.146859	0.152825	0.158210	0.154572
β	1.0	0.990518	0.945088	0.999993
ν	0.911966	0.675457	0.491647	0.328907
ρ	−0.447718	−0.490521	−0.511180	−0.560022

Table 2.11: EURUSD. `SSabr` model: Calibrated parameters for each maturity.

   In Table 2.12, for $T = 24$ months, the GPU performance is compared to one CPU. Time is measured in seconds and computation has been carried out in single precision. The speedup with 8 OpenMP threads is near 8, specifically

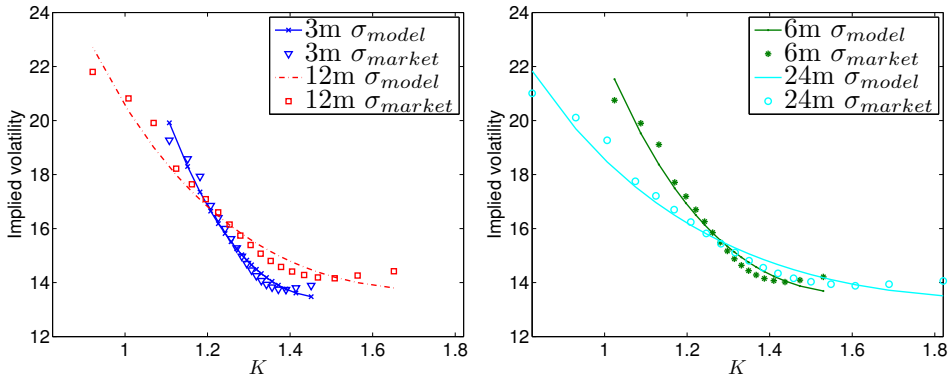


Figure 2.5: EURUSD. `SSabr` model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

7.66. As in EURO STOXX 50 calibration, 1 GPU speedup is around 190 and 2 GPUs around 333.

	CPU (1 thread)	2 threads	4 threads	8 threads	GPU	2 GPUs
Time (s)	4198.03	2109.56	1062.79	548.04	21.84	12.58
Speedup	-	1.99	3.95	7.66	192.19	333.52

Table 2.12: EURUSD. `SSabr` model: Performance of OpenMP vs. GPU versions, in single precision for $T = 24$ months.

2. **Calibration of the `DSabr_I` model using the technique `T_I`**

`DSabr_I` model calibration to all strikes and maturities has been performed with the SA GPU version and formula (2.9), with ρ and ν given by (2.11). In Table 2.13 the calibrated parameters are detailed. In Figure 2.6, the whole volatility surface at maturities 3, 6, 12 and 24 months is shown. Note that the dynamic SABR model captures correctly the volatility skew. In Table 2.14, the market volatilities vs. the model ones (2.9) are shown. The mean relative error is 2.441714×10^{-2} and the maximum relative error is 6.954307×10^{-2}. In Table 2.15, the computational times and the speedups in single precision are shown. Note that for 1 GPU the speedup is around 240, while for 2 GPUs is nearly 451.

$\alpha = 0.155464$	$\beta = 0.971908$	$\rho_0 = -0.642617$	$\nu_0 = 0.800275$	$a = 0.001$	$b = 2.6093$

Table 2.13: EURUSD. `DSabr_I` model: Calibrated parameters.

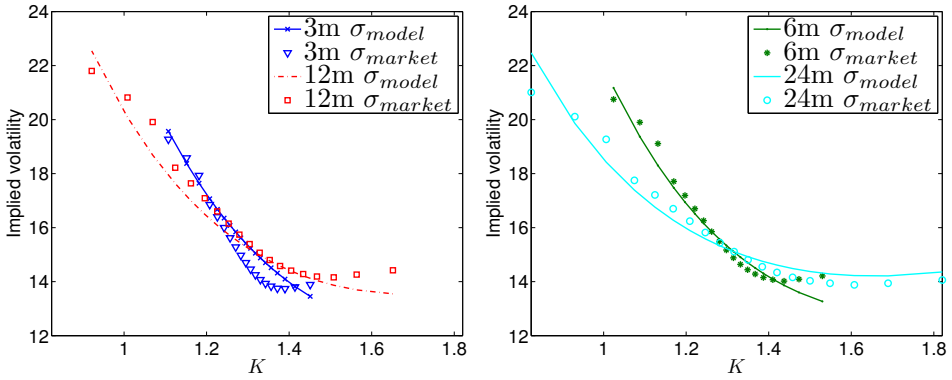


Figure 2.6: EURUSD. `DSabr_I` model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

3. Calibration of the `DSabr_II` model using technique `T_II`

Analogously to the previous Section 2.6.2, an asymptotic expression for implied volatility in the `DSabr_II` model is not available. The calibration has been carried with Technique II (see Section 2.5). In this case, calibration is performed in prices and using double precision. The set of calibrated parameters is detailed in Table 2.16.

3 months				6 months			
K	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	K	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
1.2075	16.85	17.0683	1.295549×10^{-02}	1.1700	17.71	17.4751	1.326369×10^{-02}
1.2950	14.70	15.4197	4.895918×10^{-02}	1.2975	15.17	15.3398	1.119314×10^{-02}
1.3715	13.75	14.3171	4.124364×10^{-02}	1.4099	14.07	14.0914	1.520967×10^{-03}
12 months				24 months			
K	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	K	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
1.1240	18.22	17.6324	3.225027×10^{-02}	1.0746	17.75	17.3887	2.035493×10^{-02}
1.3043	15.39	15.2020	1.221572×10^{-02}	1.3161	15.11	15.1075	1.654533×10^{-04}
1.4673	14.19	14.0396	1.059901×10^{-02}	1.5485	13.94	14.2853	2.477044×10^{-02}

Table 2.14: EURUSD. `DSabr_I` model: σ_{market} vs. σ_{model}.

	CPU (1 thread)	2 threads	4 threads	8 threads	GPU	2 GPUs
Time (s)	16793.41	8389.14	4240.11	2204.20	69.73	37.16
Speedup	-	2.00	3.96	7.62	240.83	451.92

Table 2.15: EURUSD. `DSabr_I` model: Performance of OpenMP vs. GPU versions, in single precision.

$\alpha = 0.154037$	$\beta = 1.000000$	$\rho_0 = -0.693682$	$\nu_0 = 7.541424$	$a = 0.000000$
$b = 150.000000$	$d_\rho = -0.200342$	$d_\nu = 0.339807$	$q_\rho = 0.345973$	$q_\nu = -0.992551$

Table 2.16: EURUSD. `DSabr_II` model: Calibrated parameters.

Figure 2.7 shows the comparison between market and model prices. The maximum relative error is 1.418863×10^{-1} and the mean relative error is 2.192849×10^{-2}. In Table 2.17 the market and model prices for some strikes are shown.

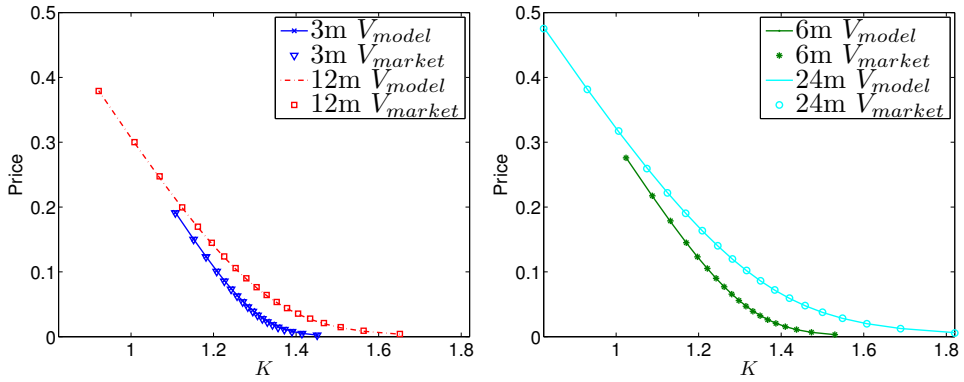


Figure 2.7: EURUSD. `DSabr_II` model: V_{model} vs. V_{market} for the whole prices surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

The performance of calibration using 1 GPU is 35033.72 seconds and with 2 GPUs is equal to 19143.35 seconds; getting a speedup around 1.83 times. We do not make a comparison between CPU and GPU because the computation time in CPU results to be prohibitive.

`DSabr_I` vs. `DSabr_II`

The accuracy of `DSabr_II` model with respect to `DSabr_I` model is compared using the mean relative error in prices. The `DSabr_I` model error is 3.647441×10^{-2}. Therefore,

3 months				6 months			
K	V_{market}	V_{model}	$\frac{\lvert V_{market}-V_{model}\rvert}{V_{market}}$	K	V_{market}	V_{model}	$\frac{\lvert V_{market}-V_{model}\rvert}{V_{market}}$
1.2075	0.100489	0.101712	1.217204×10^{-02}	1.1700	0.144905	0.144950	3.072116×10^{-04}
1.2950	0.038794	0.040476	4.335060×10^{-02}	1.2975	0.055770	0.056139	6.611759×10^{-03}
1.3715	0.010770	0.011697	8.603287×10^{-02}	1.4099	0.015472	0.015539	4.271299×10^{-03}
12 months				24 months			
K	V_{market}	V_{model}	$\frac{\lvert V_{market}-V_{model}\rvert}{V_{market}}$	K	V_{market}	V_{model}	$\frac{\lvert V_{market}-V_{model}\rvert}{V_{market}}$
1.1240	0.199490	0.198897	2.971219×10^{-03}	1.0746	0.259398	0.260539	4.399383×10^{-03}
1.3043	0.076379	0.075409	1.269954×10^{-02}	1.3161	0.102106	0.101766	3.330301×10^{-03}
1.4673	0.020951	0.020010	4.495205×10^{-02}	1.5485	0.028409	0.028189	7.756103×10^{-03}

Table 2.17: EURUSD. `DSabr_II` model: V_{market} vs. V_{model}.

again the `DSabr_II` model captures better the market dynamics.

Calibration test: pricing European options

In order to validate the correct calibration of model parameters, we price European options with the `DSabr_I` model and Monte Carlo pricing method. The price is denoted with $V_{\sigma_{model}}$. Note that for the `DSabr_II` model this task is redundant, since the calibration was also carried out with the same Monte Carlo method.

EURO STOXX 50: In Figure 2.8, the comparison between market prices (calculated with Black-Scholes formula and market volatilities) and model prices (calculated with the expression (2.9) and parameters of Table 2.6 in the Black-Scholes formula) are plotted. The mean relative error is 2.840228×10^{-2} and the maximum relative error is 1.008734×10^{-1}. Pricing all options with GPU takes 0.257248 seconds.

EURUSD: Analogously to the EURO STOXX 50 case, in Figure 2.9, the market prices vs. the model ones are shown, using the parameters of Table 2.13. Furthermore, the relative errors are shown. The mean relative error is 3.577857×10^{-02} and the maximum relative error is 2.582023×10^{-01}. Pricing all options in GPU takes 0.248013 seconds.

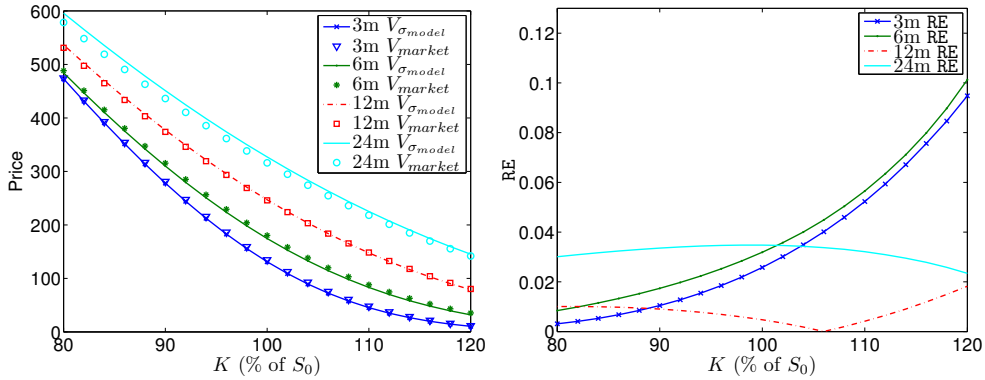


Figure 2.8: EURO STOXX 50. `DSabr_I` model for pricing European options. Prices (left) and relative errors (right).

2.6.3 Pricing a cliquet option

In this section the pricing of a cliquet option on the EUR/USD exchange rate is described. For this purpose, we use the GPU Monte Carlo method and consider `DSabr_I` and `DSabr_II` models. For the first one, we choose the parameters of Table 2.13 and for the second one those in Table 2.16.

Cliquet options are options where the strike price is periodically reset several times before final expiration date [138]. At the resetting date, the option will expire worthless if the current security price is below the strike price, and the strike price will be reset to this lower price. If the security price at resetting date is higher than the strike, the investor will earn the difference and the strike price will be reset to this higher price. Thus, a cliquet option is equivalent to a serie of forward-starting at-the-money options with local limits, the so-called cap and floor limits. The option payoff function is:

$$Cliquet\big(F_l, C_l, \{d_i\}_{i=1\ldots D}, \{S_{d_i}\}_{i=1\ldots D}\big) = \sum_{i=2}^{D} \max\left(\min\left(\frac{S_{d_i} - S_{d_{i-1}}}{S_{d_{i-1}}}, C_l\right), F_l\right), \tag{2.18}$$

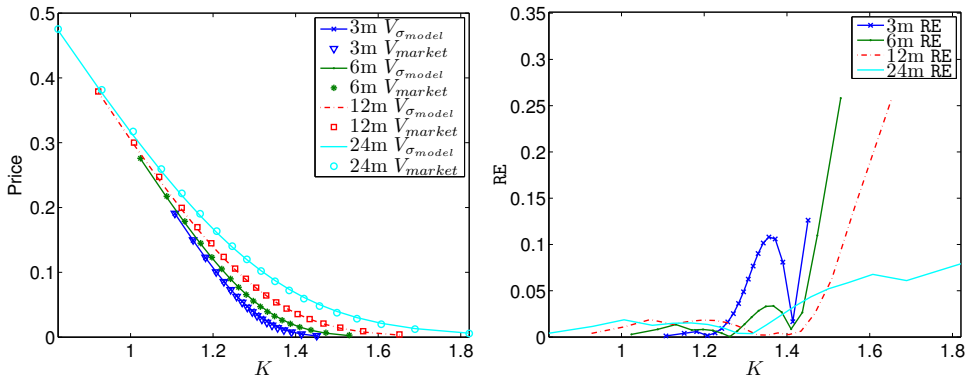


Figure 2.9: EURUSD. `DSabr_I` model for pricing European options. Prices (left) and relative errors (right).

where C_l and F_l are the local cap and floor limits, respectively, d_i denotes a resetting date and S_{d_i} is the underlying price at time d_i. Thus, $R_i = \frac{S_{d_i} - S_{d_{i-1}}}{S_{d_{i-1}}}$ is known as the *return* between dates d_{i-1} and d_i.

Usually, global limits can also be added, so that the payoff function is:

$$Cliquet_{global}(C_g, F_g, Cliquet) = \max\big(C_g, \min(F_g, Cliquet)\big), \tag{2.19}$$

where C_g is the global cap limit, F_g is the global floor limit and $Cliquet$ is given by (2.18).

We consider the following data: $T = 12$ months, $D = 4$, $d_i = \frac{i \times T}{D}$, $F_l = -0.02$, $C_l = 0.02$, $F_g = 0$, $C_g = 0.2$ and the payoff (2.19). We price the cliquet option with `DSabr_I` and `DSabr_II` models using single precision. By using the `DSabr_I` model and the calibrated parameters in Table 2.13, the simulated cliquet option price is $V_{\sigma_{model}}$= 0.098289, while when using `DSabr_II` model and the parameters in Table 2.16 the simulated price is V_{MC}= 0.120846. Note that $|V_{\sigma_{model}} - V_{MC}| = 0.022557$ and the execution time is 0.495199 seconds in both cases.

2.7 Conclusions

Static or dynamic SABR models should be chosen depending on the particular financial derivative and the available market data. In both cases, the calibration of parameters can be carried out either by using an asymptotic implied volatility formula or a Monte Carlo simulation method. When using standard hardware tools, due to the high computational time with Monte Carlo strategy, the formula is mainly used. Nevertheless, the recently increasing use of GPU technology for scientific computing can also be extended to the algorithms involved in the calibration procedure. In the present chapter we propose the application of this technology to the SA global optimization algorithm and to the Monte Carlo simulation for the calibration in static and dynamic SABR models. In this GPU setting, the calibration by Monte Carlo is affordable in terms of computational time and the cost of calibration with the asymptotic formula can be highly reduced. In order to illustrate the performance of our GPU implementations, the calibration of static and dynamic SABR models for EURO STOXX 50 index and EUR/USD exchange rate have been carried out with asymptotic formula and Monte Carlo method. Once the parameters have been calibrated, a cliquet option on EUR/USD has been priced. For the dynamic SABR model we propose an original more general expression for the functional parameters that reveals specially well suited for a EUR/USD exchange rate market data set.

Numerical results illustrate the expected behavior of both SABR models and the accuracy of the calibration. In terms of computational time, if we use the formula then the achieved speedup with respect to CPU computation is around 200 with one GPU. Also, it is illustrated that GPU technology allows the use of Monte Carlo simulation for calibration purposes, the corresponding computational time with CPU being unaffordable.

Chapter 3

SABR/LIBOR market models: Monte Carlo approach

3.1 Introduction

Since the seminal papers by Brace, Gatarek and Musiela [16], Jamshidian [79] and Miltersen, Sandmann and Sondermann [108] to introduce the *Libor Market Model* (LMM), several authors extended it to reproduce volatility smiles appearing in real markets.

The basic LMM has some desirable features: it is flexible, supports multiple factors and rich volatility structures and it justifies the use of Black's formula for caplet prices, which is the standard formula employed in the cap market (see [19]). The last one constitutes its major advantage since it allows an implicit calibration of at-the-money caps volatilities. In addition, it is possible to calibrate at-the-money swaption volatilities via closed formula approximations with high accuracy (e.g. the Rebonato swaption approximation, see [119]). These reasons explain the success of the model and why it has been widely accepted by the financial industry.

Nevertheless, the standard LMM presents the same drawbacks as the classical Black-Scholes theory. The major disadvantage comes from the assumption of deterministic volatility coefficients that prevents matching cap and swaption volatility

smiles and skews observed in the markets. This implies that after calibrating the model to at-the-money options, the model underprices the off-the-money options.

In order to overcome this drawback, there has been great research in extending the standard LMM to correctly capture market volatility smiles and skews. Different extended LMMs were suggested and can mainly fall into three categories: local volatility models, stochastic volatility models and jumps-diffusion models.

In the local volatility models the volatility is a function of the underlying asset price (forward rate) and the time. These models were introduced by Dupire [37] and Derman and Kani [34], who proposed this extension for equity and foreign-exchange options. Andersen and Andreasen [4] introduced a special case of local volatility models, the Constant Elasticity of Variance (CEV), to develop an extension of the LMM for capturing the skew. They showed how to obtain swaption smile asymptotically. Their method is still based in the Rebonato "freezing" argument, see [119], which is not completely mathematically justified. CEV model can generate a monotone skew of implied volatilities but fails to reproduce a smile, which is often the case in reality.

Jump-diffusion models for assets were introduced by Merton [105] and Eberlein [39]. Jamshidian [80], Glasserman and Kou [50], Glasserman and Merener [51] and Belomestny and Schoenmakers [8] proposed alternative extensions of the LMM by adding jumps in the forward rate dynamics. Lévy LIBOR models have been studied by Eberlein and Özkan [40]. With these models one can manipulate the slope and the curvature of a skewed smile by changing jump intensity and jump sizes. While this jump approach can generate stationary nonmonotonic volatility smiles, it involves several technical difficulties to develop numerical schemes for the resulting model. Moreover, these models result unsuitable to generate asymmetric smiles and skews, since the jump component of the forward rate dynamics typically needs to be of substantial magnitude. While such dynamics are probably reasonable for equity prices (see [3]) they might be less natural for the term structure of interest rate forwards.

In order to correctly capture the stochastic behaviour of the volatility and to

reproduce market smiles, different stochastic volatility models have been proposed. The main examples are Hull and White [77] and Heston [73] models. In the Hull and White model, lognormal variance process is modelled. When the correlation between spot and variance is zero, by using a mixing approach, the authors obtained asymptotic expansions for options prices. However, the main drawback of this model comes from its inability to capture nonsymmetric smiles. Heston proposed a model where the volatility is a mean-reverting square-root process. By using Fourier transforms he derived a closed form formula for option prices. The main advantages of this model are its nice empirical properties and its analytical tractability. An application of the Heston model to the LMM appears in Wu and Zhang [140]. They adopt a multiplicative stochastic factor to the volatility functions of all relevant forward rates. The stochastic factor follows a square-root diffusion process, and it can be correlated to the forward rates. They also develop a closed-form formula for swaptions in terms of Fourier transforms. Other extensions of the LIBOR market model allowing stochastic volatility are those we mention hereafter. Andersen and Brotherton-Ratcliffe [5] proposed a general framework for extending the LIBOR market model. Their model allows for nonparametric volatility structures and includes a multiplicative perturbation of the forward volatility surface by a one-dimensional mean reverting volatility process. This volatility process is driven by a Brownian motion independent of the Brownian motions driving the forward rates, so that, under different numeraires, the dynamics of the volatilities remains the same. Using asymptotic expansion techniques, they provided closed-form pricing formulas for caplets and swaptions prices. In [118], Piterbarg has extended this approach with a model where forward rates follow shifted-lognormal diffusion processes with stochastic volatility. The volatility is a mean reverting square-root process uncorrelated with the Brownian motions governing the dynamics of the forward rates. Using Markovian projection and parameter averaging, Piterbarg derives fast and accurate European option pricing techniques under general time-dependent parameters. In [82], Joshi and Rebonato proposed a

shifted-lognormal LIBOR model with a volatility parameterization based on a functional form with stochastic coefficients. This model has very similar properties to the Andersen and Andreasen [4] one, among them its major problem being the monotonicity of implied volatility curves. All the stochastic volatility models presented so far have one single volatility factor.

In [67], Hagan, Kumar, Lesniewski and Woodward proposed a stochastic volatility model known as the SABR model (acronym for stochastic, alpha, beta and rho, three of the four model parameters), arguing that local volatility models could not reproduce market volatility smiles and that their predicted volatility dynamics contradicts market smiles and skews. The forward price of an asset follows, under the assets canonical measure, a CEV type process with stochastic volatility driven by a driftless process. The Brownian motion driving the volatility can be correlated with the one associated to the forward price. The main advantages of the model are the following. Firstly, it is able to correctly capture market volatility smiles. Secondly, its parameters, which play specific roles in the generation of smiles and skews, have an intuitive meaning. Thirdly, the authors obtained an analytical approximation for the implied volatility (known as Hagan formula) through singular perturbation techniques, thus allowing an easy calibration of the model. Finally, it has become the market standard for interpolating and extrapolating prices of plain vanilla caplets and swaptions (see [121]). In [113] Oblój improved Hagan formula.

Several authors have recently tried to unify SABR and LIBOR market models. In the more standard LIBOR market model [16], the dynamics of each LIBOR forward rate under the corresponding terminal measure are assumed to be martingales with constant volatility. When adding the SABR model, the forward rates and volatility processes satisfy the following coupled dynamics

$$\begin{aligned} dF_i(t) &= V_i(t)F_i(t)^{\beta_i}dW_i(t), \\ dV_i(t) &= \sigma_i V_i(t)dZ_i(t). \end{aligned}$$

We note that if the interest rate derivative only depends on one particular forward rate, then it is convenient to use the corresponding terminal measure. However,

when derivatives depend on several forward rates, a common measure needs to be used. Thus, in the case of pricing complex derivatives a change of measure produces the appearance of drift terms in forward rates and volatilities dynamics.

In [70, 71], Labordère presents a unification of LIBOR and SABR models using hyperbolic geometry and heat kernel expansion to fit Taylor expansions for swaption implied volatilities. In [68], Hagan et al. studied the natural extension of both the LMM and the SABR model. They used the technique of low noise expansions in order to produce accurate and workable approximations to swaption volatilities. Mercurio and Morini, arguing that a number of volatility factors lower than the number of state variables is often chosen, proposed in [103] a SABR/LIBOR market model with one single volatility factor. They designed a LIBOR market model starting from the reference SABR dynamics, with the purpose of preserving the SABR closed formula. In [121, 120, 122], Rebonato et al. designed a time-homogeneous SABR-consistent extension of the LMM. More precisely, they specified financially motivated dynamics for the LMM forward rates and volatilities that match the SABR prices very close. They also suggested a simple financially justifiable and computationally affordable way to calibrate the model. In this chapter we focus in these last three different SABR/LIBOR market models.

By using heuristic, empirical or very qualitative arguments, in all the here presented extensions of the LMM, the authors obtain accurate analytical approximations for caps/swaptions to calibrate the model. In general, swaptions cannot be priced in closed form in the LMM and the main challenge of these works comes from the analytical approximations to price these swaptions.

All the previous papers argue that the "brute-force" approach, which consists in calibrating the models using Monte Carlo simulation to price swaptions, is not a practical choice, because each Monte Carlo evaluation results computationally very expensive. However, in this chapter we propose the use of relatively old Simulated Annealing type algorithms [88], which reveal as highly efficient when implemented using High Performance Computing techniques. This combination makes possible

the calibration in a reasonable computational time. Such algorithms have already been successfully applied in other related contexts, see [43, 42] for more details.

In this chapter we propose an efficient calibration strategy to some market prices for the parameters appearing in the three selected SABR/LIBOR market models. More precisely, we consider the market prices of caplets and swaptions and we pose the corresponding global optimization problems to calibrate the model parameters. Moreover, we use a Simulated Annealing algorithm to solve the problem. In order to speed up the algorithm we propose a parallel implementation in GPUs.

The chapter is organized as follows. In Section 3.2, the SABR/LIBOR market models proposed by Hagan, Mercurio & Morini and Rebonato are introduced. In Section 3.3, the calibration procedures are explained. In Section 3.4, the obtained numerical results are shown. Finally, in Section 3.5 some conclusions are discussed.

Most of the results in this chapter are included in the reference [44].

3.2 SABR/LIBOR market models

3.2.1 Hagan model

This model arises as the natural coupling between SABR and LMM models [68]. Thus, for each $i = 1, \ldots, M$ let F_i and V_i be the i-th forward rate that matures at time T_i and its corresponding stochastic volatility, respectively. Then, under a common measure their dynamics are given by

$$dF_i(t) = \mu^{F_i}(t)F_i(t)^{\beta_i}dt + V_i(t)F_i(t)^{\beta_i}dW_i(t), \tag{3.1}$$

$$dV_i(t) = \mu^{V_i}(t)V_i(t)dt + \sigma_i V_i(t)dZ_i(t), \tag{3.2}$$

with the associated correlations denoted by

$$\mathbb{E}[dW_i(t) \cdot dW_j(t)] = \rho_{ij}dt, \quad \mathbb{E}[dW_i(t) \cdot dZ_j(t)] = \phi_{ij}dt, \quad \mathbb{E}[dZ_i(t) \cdot dZ_j(t)] = \theta_{ij}dt,$$

and the initial given values $\alpha_i = V_i(0)$ and $F_i(0)$. Thus, the correlation structure is given by the block-matrix

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{\rho} & \boldsymbol{\phi} \\ \boldsymbol{\phi}^\top & \boldsymbol{\theta} \end{bmatrix},$$

where the submatrix $\boldsymbol{\rho} = (\rho_{ij})$ contains all the correlations between the forward rates F_i and F_j, the submatrix $\boldsymbol{\phi} = (\phi_{ij})$ includes the correlations between the forward rates F_i and the instantaneous volatilities V_j, and the submatrix $\boldsymbol{\theta} = (\theta_{ij})$ contains the correlations between the instantaneous volatilities V_i and V_j.

More precisely, if we introduce the bank-account numeraire $\beta(t)$, defined by

$$\beta(t) = \prod_{j=0}^{i-1} \big(1 + \tau_j F_j(T_j)\big) \quad \text{if } t \in [T_i, T_{i+1}],$$

then, under the associated spot probability measure, the drift terms of the processes defined in (3.1) and (3.2) are

$$\mu^{F_i}(t) = V_i(t) \sum_{j=h(t)}^{i} \frac{\tau_j \rho_{ij} V_j(t) F_j(t)^{\beta_j}}{1 + \tau_j F_j(t)}, \quad \mu^{V_i}(t) = \sigma_i \sum_{j=h(t)}^{i} \frac{\tau_j \phi_{ij} V_j(t) F_j(t)^{\beta_j}}{1 + \tau_j F_j(t)},$$

where $h(t)$ denotes the index of the first unfixed F_i, i.e.,

$$h(t) = j, \text{ if } t \in [T_{j-1}, T_j). \tag{3.3}$$

In terms of the moneyness[1], defined as $\ln\Big(\dfrac{K}{F_i(0)}\Big)$, the implied volatility[2] for this model is given by the Hagan second order approximation formula (also including the

[1]Moneyness measures the ratio between the strike price, K, and the current value of the underlying, $F_i(0)$. Thus, if $K = F_i(0)$ then the call or put options are said to be *at the money* (moneyness is equal zero). If $K < F_i(0)$ then a call option is said to be *in the money* (moneyness is negative) and if $K > F_i(0)$ then the call option is said to be *out of the money* (moneyness is positive). For put options, *out of the money* and *in the money* correspond to negative and positive moneyness, respectively.

[2]The implied volatility is the one that reproduces the market price when inserted in Black-Scholes formula.

correction of Oblój in [113]):

$$\sigma\big(K,F_i(0)\big) \approx \frac{\alpha_i}{F_i(0)^{(1-\beta_i)}} \times \Bigg\{1 - \frac{1}{2}(1-\beta_i-\phi_{ii}\sigma_i\omega_i)\cdot\ln\Big(\frac{K}{F_i(0)}\Big) + \frac{1}{12}\Big((1-\beta_i)^2+(2-3\phi_{ii}^2)\sigma_i^2\omega_i^2+3\big((1-\beta_i)-\phi_{ii}\sigma_i\omega_i\big)\Big)\cdot\Big[\ln\Big(\frac{K}{F_i(0)}\Big)\Big]^2\Bigg\}, \tag{3.4}$$

where $\omega_i = \alpha_i^{-1}F_i(0)^{(1-\beta_i)}$.

For the correlations, we consider the following functional parameterizations:

$$\rho_{ij} = \eta_1 + (1-\eta_1)\exp[-\lambda_1|T_i - T_j|], \tag{3.5}$$

$$\theta_{ij} = \eta_2 + (1-\eta_2)\exp[-\lambda_2|T_i - T_j|], \tag{3.6}$$

$$\phi_{ij} = \mathrm{sign}(\phi_{ii})\sqrt{|\phi_{ii}\phi_{jj}|}\exp\left[-\lambda_3(T_i-T_j)^+ - \lambda_3(T_j-T_i)^+\right], \tag{3.7}$$

where the terms ϕ_{ii} have been previously calibrated using (3.4) for the whole volatilities surfaces. Moreover, parameters η_i, λ_i and ϕ_{ij} are calibrated to fit the smiles of swap rates.

3.2.2 Mercurio & Morini model

For this model [103], the existence of a lognormal common volatility process to all forward rates is assumed, while each F_i satisfies a particular SDE. More precisely, we have

$$dF_i(t) = \mu^{F_i}(t)F_i(t)^\beta dt + \alpha_i V(t)F_i(t)^\beta dW_i(t), \tag{3.8}$$

$$dV(t) = \sigma V(t)dZ(t), \tag{3.9}$$

with

$$\mathbb{E}[dW_i(t)\cdot dW_j(t)] = \rho_{ij}dt, \quad \mathbb{E}[dW_i(t)\cdot dZ(t)] = \phi_i dt,$$

and the initial given values $V(0)=1$ and $F_i(0)$. In this case, the correlation block-matrix is

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{\rho} & \boldsymbol{\phi} \\ \boldsymbol{\phi}^\top & 1 \end{bmatrix},$$

where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_M)^\top$. Under the spot probability measure, the drift terms in equation (3.8) are

$$\mu^{F_i}(t) = \alpha_i V(t) \sum_{j=h(t)}^{i} \frac{\tau_j \rho_{ij} \alpha_j V(t) F_j(t)^\beta}{1 + \tau_j F_j(t)},$$

where $h(t)$ is given by the expression (3.3).

The calibration is similar to the previous case. By using *SABR* superindexes, the parameters of the Hagan implied volatility formula (3.4) are

$$\begin{aligned} &\beta_i^{SABR} = \beta, \quad \phi_{ii}^{SABR} = \phi_i, \quad \sigma_i^{SABR} = \sigma, \\ &\alpha_i^{SABR} = \alpha_i \left[\mathrm{e}^{\int_0^{T_i} M_i(s) ds} \right], \text{ where } M_i(t) = -\sigma \sum_{j=h(t)}^{i} \frac{\tau_j \phi_j \alpha_j F_j(0)^\beta}{1 + \tau_j F_j(0)}. \end{aligned} \tag{3.10}$$

Note that in this case we only need to consider (3.5) for the forward rates correlations.

3.2.3 Rebonato model

This model is analogous to Hagan one, except for the dynamics of the volatilities. More precisely, this model assumes the following dynamics [120]:

$$dF_i(t) = \mu^{F_i}(t) F_i(t)^{\beta_i} dt + V_i(t) F_i(t)^{\beta_i} dW_i(t), \tag{3.11}$$

$$V_i(t) = \kappa_i(t) g_i(t), \tag{3.12}$$

$$d\kappa_i(t) = \mu^{\kappa_i}(t) \kappa_i(t) dt + \kappa_i(t) h_i(t) dZ_i(t), \tag{3.13}$$

where

$$g_i(t) = \big(a + b(T_i - t)\big) \exp\big(-c(T_i - t)\big) + d, \quad h_i(t) = \big(\alpha + \beta(T_i - t)\big) \exp\big(-\gamma(T_i - t)\big) + \delta,$$

and the correlation structure is given by the parameterizations (3.5)-(3.7).

Again, using the spot probability measure, the drift terms of the previous processes are

$$\mu^{F_i}(t) = V_i(t) \sum_{j=h(t)}^{i} \frac{\tau_j \rho_{ij} V_j(t) F_j(t)^{\beta_j}}{1 + \tau_j F_j(t)}, \quad \mu^{\kappa_i}(t) = h_i(t) \sum_{j=h(t)}^{i} \frac{\tau_j \phi_{ij} V_j(t) F_j(t)^{\beta_j}}{1 + \tau_j F_j(t)}.$$

Furthermore, in this model the parameters of the Hagan implied volatility formula (3.4) are

$$\beta_i^{SABR} = \beta_i, \quad \phi_{ii}^{SABR} = \phi_{ii}, \quad \alpha_i^{SABR} = \kappa_i(0)\left(\frac{1}{T_i}\int_0^{T_i} g_i(t)^2 dt\right)^{\frac{1}{2}},$$
$$\sigma_i^{SABR} = \frac{\kappa_i(0)}{\alpha_i^{SABR} T_i}\left(2\int_0^{T_i} g_i(t)^2 \hat{h}_i(t)^2 t dt\right)^{\frac{1}{2}}, \text{ where } \hat{h}_i(t) = \sqrt{\frac{1}{t}\int_0^t (h_i(s))^2 ds}. \tag{3.14}$$

3.3 Model calibration

Model parameters are calibrated in two stages, firstly to caplets[3] and secondly to swaptions[4]. We note that model parameters can be classified into two categories (volatility and correlation parameters):

- The volatility parameters for each model are given by:
  - Hagan: $\boldsymbol{x} = (\phi_{ii}, \sigma_i, \alpha_i)$.
  - Mercurio & Morini: $\boldsymbol{x} = (\phi_i, \sigma, \alpha_i)$.
  - Rebonato: $\boldsymbol{x} = (\phi_{ii}, \kappa_i, \text{parameters of the volatility functions } g \text{ and } h)$.
- The correlation parameters for each model are given by:
  - Hagan: $\boldsymbol{y} = (\eta_1, \lambda_1, \eta_2, \lambda_2, \lambda_3)$.
  - Mercurio & Morini: $\boldsymbol{y} = (\eta_1, \lambda_1)$.
  - Rebonato: $\boldsymbol{y} = (\eta_1, \lambda_1, \eta_2, \lambda_2, \lambda_3)$.

According to the previous classification, the cost functions to be minimized in the calibration process are the following:

[3] A caplet is a basic interest rate derivative which mainly consists in a call option that pays the positive difference between a floating rate and a fixed one (strike). A cap contract is a set of caplets associated with related maturity dates (tenor structure). See [19], for example.

[4] A swap contract is an interest rate derivative that exchanges two different interest rates. A swaption is an option giving the right to enter in a swap contract at a given future time. See [19], for example.

- Function to calibrate the market prices of caplets:

$$f_c(\boldsymbol{x}) = \sum_{i=1}^{M} \sum_{j=1}^{numK} \Big(\sigma\big(K_j, F_i(0)\big) - \sigma_{market}\big(K_j, F_i(0)\big)\Big)^2(\boldsymbol{x}),$$

  where σ is given by Hagan formula ((3.4), (3.10) or (3.14), depending on the model), σ_{market} are the market volatilities and $\boldsymbol{x}$ is the vector containing the volatility parameters of the model. Moreover, M and $numK$ denote the number of maturities and strikes of the caplets, respectively.

- Function to calibrate the market prices of swaptions:

$$f_s(\boldsymbol{y}) = \sum_{i=1}^{numSws} \left(S_{\text{Black}}(swaption_i) - S_{MC}(swaption_i)\right)^2(\boldsymbol{y}),$$

  where $swaption_i$ denotes the i-th swaption, S_{Black} represents the Black formula for swaptions and $S_{MC}(swaption_i)$ denotes the value of the *i-th* swaption computed with Monte Carlo method. Moreover, the vector $\boldsymbol{y}$ contains the correlation parameters and $numSws$ is the number of swaptions.

In this chapter, the calibration of the parameters has been performed with a Simulated Annealing (SA) global optimization algorithm introduced in the Chapter 1. In real applications the hybrid approaches (in which SA provides a starting point for a local minimization algorithm) are widely used, as we have explained in Chapter 1. In this chapter we have considered the Nelder-Mead algorithm as the local minimizer. In order to calibrate the models with fewer parameters (Hagan and Mercurio & Morini), the mono-GPU version introduced in the Chapter 1 (see Figure 1.2) results to be enough. However, in order to calibrate models with more parameters (Rebonato), the multi-GPU version explained in the Chapter 2 (see Figure 2.1) becomes more suitable, since the minimization process is much more costly. Section 3.4 contains the achieved speedups when implied volatility formulas are available.

In the SABR/LIBOR market models, for the general calibration to swaption market prices an explicit formula to price swaptions is not available. Therefore, we use Monte Carlo simulation technique to price swaptions, thus leading to two nested

Monte Carlo loops: one for the SA and the other one for the swaption pricer. So, as the Monte Carlo swaption pricer is carried out inside the GPU, the SA minimization algorithm is run on CPU. At this point we illustrate in Table 3.1 the obtained speedups in the SABR/LIBOR pricing with Monte Carlo simulation for different number of paths and values of Δt. Notice that speedups around 200 are obtained for 10^6 paths. In order to use all available GPUs in the system, we propose a CPU SA parallelization using OpenMP [149]. So, each OpenMP SA thread uses a GPU to evaluate the Monte Carlo objective function (see Figure 3.1). This approach could be easily extrapolated to a cluster of GPUs using MPI [148]. Notice that in this case the sequential Monte Carlo pricing with CPU leads to prohibited times for the whole calibration procedure.

Number of paths	Δt	CPU (s)	GPU (s)	*Speedup*
10^3	10^{-1}	0.558	0.094	×5.936
	10^{-2}	5.580	0.222	×25.135
	10^{-3}	55.956	1.406	×39.798
10^4	10^{-1}	5.572	0.119	×46.823
	10^{-2}	55.740	0.390	×142.923
	10^{-3}	557.698	3.081	×181.012
10^5	10^{-1}	55.692	0.323	×172.421
	10^{-2}	558.331	2.886	×193.462
	10^{-3}	5601.292	28.550	×196.192
10^6	10^{-1}	557.696	2.375	×234.819
	10^{-2}	5588.070	27.950	×199.931
	10^{-3}	55904.184	283.970	×196.866

Table 3.1: Execution times (in seconds) and speedups in the pricing of caplets with Monte Carlo and using single precision (Hagan model).

3.4 Numerical results

In this section we present a test where we calibrate Hagan, Mercurio & Morini and Rebonato models to real market data. Market data correspond to the 6 months EURIBOR rate. We show in Table 3.2 the discount factor curve, in Table 3.3 the

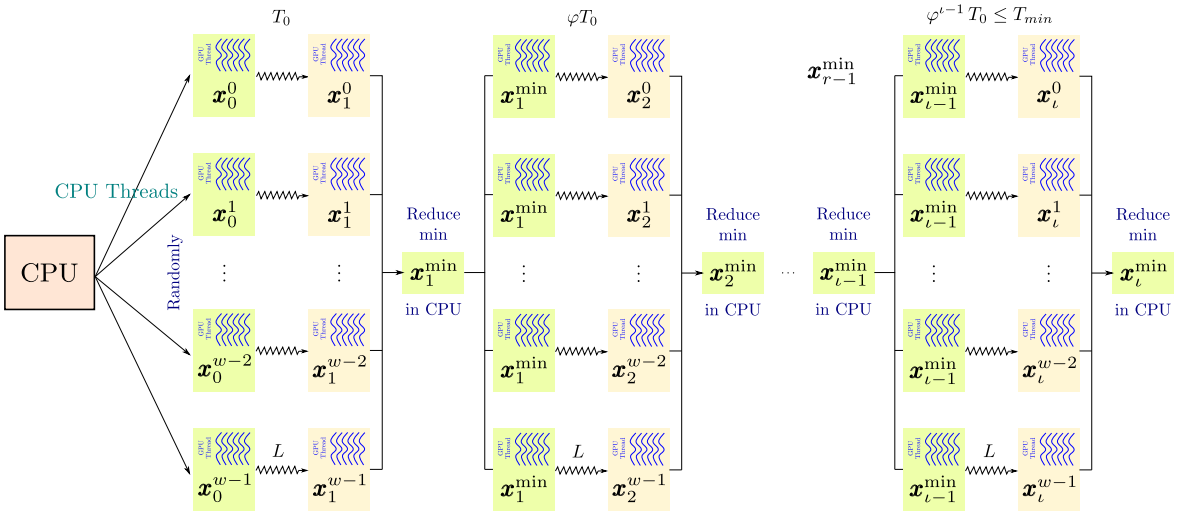


Figure 3.1: Sketch of the parallel SA using OpenMP and considering a Monte Carlo method in the cost function.

smiles of the forward rates and in Table 3.4 the smiles of the swap rates.

Numerical experiments have been performed with the following hardware and software configurations: two GPUs Nvidia Geforce GTX470, two quad-core CPUs Xeon E5620 clocked at 2.4 Ghz with 16 GB of RAM, CentOS Linux, Nvidia CUDA SDK 4.0 and GNU C/C++compilers 4.1.2.

Date	$P(0,t)$	Date	$P(0,t)$	Date	$P(0,t)$
21/11/2011	1	19/09/2013	0.97713559399	23/11/2021	0.77845715189
22/11/2011	0.99998041705	25/11/2013	0.97412238564	23/11/2023	0.72565274014
23/02/2012	0.99622554093	24/11/2014	0.95730277130	23/11/2026	0.65317498182
21/03/2012	0.99575871128	23/11/2015	0.93611709432	24/11/2031	0.56564376817
21/06/2012	0.99263851754	23/11/2016	0.91144251116	24/11/2036	0.50321672724
20/09/2012	0.98966227733	23/11/2017	0.88505982818	25/11/2041	0.45392855927
19/12/2012	0.98673874563	23/11/2018	0.85798260233	23/11/2051	0.34982415774
19/03/2013	0.98372608449	25/11/2019	0.83116001862	23/11/2061	0.26125094146
20/06/2013	0.98048414547	23/11/2020	0.80486541573	23/11/2071	0.19657659346

Table 3.2: Discount factor curve.

	-80%	-60%	-40%	-20%	0%	20%	40%	60%	80%
21-05-12	142.61%	117.05%	97.26%	82.58%	72.29%	70.89%	69.49%	68.08%	66.67%
21-11-12	112.74%	99.23%	88.27%	79.62%	73.03%	71.95%	70.87%	69.77%	68.69%
21-05-13	91.55%	83.75%	77.09%	71.50%	67.93%	67.10%	66.41%	65.88%	65.49%
21-11-13	64.82%	60.95%	57.08%	53.21%	52.49%	51.34%	50.61%	50.30%	50.46%
21-05-14	66.96%	61.84%	56.69%	52.43%	50.32%	48.72%	47.70%	47.14%	46.97%
21-11-14	69.20%	62.75%	56.30%	51.65%	48.19%	46.19%	44.91%	44.12%	43.66%
21-05-15	71.49%	63.67%	55.92%	50.89%	46.19%	43.83%	42.32%	41.35%	40.64%
21-11-15	73.89%	64.61%	55.54%	50.13%	44.25%	41.56%	39.84%	38.71%	37.78%
21-05-16	76.34%	65.56%	55.16%	49.39%	42.40%	39.43%	37.54%	36.26%	35.15%
21-11-16	78.90%	66.53%	54.78%	48.65%	40.61%	37.38%	35.34%	33.94%	32.68%
21-05-17	81.50%	67.50%	54.41%	47.94%	38.93%	35.47%	33.30%	31.81%	30.42%
21-11-17	84.24%	68.50%	54.03%	47.22%	37.29%	33.63%	31.36%	29.78%	28.28%
21-05-18	87.02%	69.50%	53.67%	46.53%	35.74%	31.92%	29.55%	27.90%	26.32%

Table 3.3: Smiles of forward rates. Fixing dates (first column) and moneyness (first row).

		-80%	-60%	-40%	-20%	0%	20%	40%	60%	80%
1 year	21/05/2012	122.30%	102.40%	87.12%	76.45%	70.40%	66.47%	64.20%	63.03%	62.56%
	21/11/2012	102.86%	89.97%	79.85%	72.49%	67.90%	64.58%	62.16%	60.39%	59.19%
	21/05/2013	95.64%	83.17%	73.42%	66.40%	62.10%	59.03%	56.84%	55.26%	54.18%
	21/11/2013	88.11%	76.06%	66.69%	60.00%	56.00%	53.18%	51.22%	49.84%	48.87%
2 years	21/05/2012	111.50%	91.60%	76.32%	65.65%	59.60%	55.67%	53.40%	52.23%	51.76%
	21/11/2012	89.66%	76.77%	66.65%	59.29%	54.70%	51.38%	48.96%	47.19%	45.99%
	21/05/2013	82.94%	70.47%	60.72%	53.70%	49.40%	46.33%	44.14%	42.56%	41.48%
	21/11/2013	77.81%	65.76%	56.39%	49.70%	45.70%	42.88%	40.92%	39.54%	38.57%
3 years	21/05/2012	106.40%	86.50%	71.22%	60.55%	54.50%	50.57%	48.30%	47.13%	46.66%
	21/11/2012	83.66%	70.77%	60.65%	53.29%	48.70%	45.38%	42.96%	41.19%	39.99%
	21/05/2013	78.34%	65.87%	56.12%	49.10%	44.80%	41.73%	39.54%	37.96%	36.88%
	21/11/2013	73.61%	61.56%	52.19%	45.50%	41.50%	38.68%	36.72%	35.34%	34.37%
4 years	21/05/2012	101.90%	82.00%	66.72%	56.05%	50.00%	46.07%	43.80%	42.63%	42.16%
	21/11/2012	80.26%	67.37%	57.25%	49.89%	45.30%	41.98%	39.56%	37.79%	36.59%
	21/05/2013	75.24%	62.77%	53.02%	46.00%	41.70%	38.63%	36.44%	34.86%	33.78%
	21/11/2013	70.91%	58.86%	49.49%	42.80%	38.80%	35.98%	34.02%	32.64%	31.67%
5 years	21/05/2012	96.15%	74.25%	58.83%	49.88%	47.40%	45.74%	44.61%	43.76%	43.05%
	21/11/2012	89.58%	68.82%	54.14%	45.54%	43.00%	39.36%	37.33%	36.15%	35.37%
	21/05/2013	83.91%	64.51%	50.71%	42.51%	39.90%	36.48%	34.59%	33.50%	32.76%
	21/11/2013	79.13%	61.09%	48.17%	40.37%	37.70%	34.50%	32.74%	31.75%	31.05%

Table 3.4: Smiles of swap rates. Length of the underlying swaps (first column), swaptions maturities (second column) and moneyness (first row).

3.4.1 Hagan model

Calibration to caplets

In Table 3.5 the calibrated parameters with SABR formula (3.4) are shown. The execution time was 8.739 seconds, 8.565 seconds employed by the mono-GPU SA (launched with a relaxed configuration, specifically, $T_0 = 10$, $T_{min} = 0.01$, $\varphi = 0.99$, $L = 10$, $w = 256 \times 64$, the cost function was evaluated 112738304 times) and 0.174 seconds to the Nelder-Mead algorithm. The sequential time of the minimization with SA is 971.960 seconds. Thus, the speedup of the proposed SA parallelization is 113.480 times.

	ϕ_{ii}	σ_i	α_i		ϕ_{ii}	σ_i	α_i
F_1	−0.4712	1.0000	0.0847	F_8	−0.4552	0.4658	0.0723
F_2	−0.1879	0.7354	0.0830	F_9	−0.5215	0.5369	0.0703
F_3	0.0719	0.5260	0.0822	F_{10}	−0.5663	0.6116	0.0706
F_4	0.2636	0.3329	0.0686	F_{11}	−0.5973	0.6858	0.0684
F_5	0.0273	0.3242	0.0662	F_{12}	−0.6204	0.7609	0.0674
F_6	−0.1942	0.3505	0.0714	F_{13}	−0.6378	0.8337	0.0652
F_7	−0.3514	0.4008	0.0696				

Table 3.5: Hagan model, calibration to caplets with SABR formula (3.4): calibrated parameters.

In Table 3.6 market vs. model volatilities (both in %) for the first twelve smiles and the moneyness varying from −40% to 40% are shown. In addition, the mean relative error (MRE) considering the whole set of smiles is presented.

In order to validate the algorithm we also performed the equivalent calibration with Monte Carlo simulation thus obtaining the same parameters as in Table 3.5, except $\phi_{11} = 0.0287$. Moreover the computational time is approximately 2 hours. We note that with formula (3.4) the mean absolute error (MAE, in %) in prices is 4.14×10^{-2}, while when using Monte Carlo simulation the obtained MAE is 4.08×10^{-2}.

In Figure 3.2 the model fitting for the smiles of all forward rates is shown. Market volatilities are plotted with triangles, while model volatilities are shown in continuous

line.

Calibration to swaptions

The calibrated parameters are $\eta_1 = 0.814904$, $\lambda_1 = 3.378797$, $\eta_2 = 0.975928$, $\lambda_2 = 3.777324$ and $\lambda_3 = 0.013940$. In Table 3.7 market vs. model swaptions prices (in %) for the first fourteen swaptions and the moneyness varying from -40% to 40% are shown, each pair with its corresponding absolute error. In addition, for the whole set of swaptions the mean absolute error (MAE) is presented.

In Figures 3.3 and 3.4 the model fitting when considering the whole swaption matrix is shown. Market prices are shown using triangles and the model ones using stars.

In the forthcoming sections 3.4.2 and 3.4.3, the analogous analysis for the other two models using the same scheme for figures and tables is presented.

3.4.2 Mercurio & Morini model

Calibration to caplets

In Table 3.8 the calibrated parameters are shown. The execution time was 9.165 seconds, 9.124 seconds employed by the mono-GPU SA (launched with a relaxed configuration, specifically, $T_0 = 10$, $T_{min} = 0.01$, $\varphi = 0.99$, $L = 10$, $w = 256 \times 64$, the cost function was evaluated 112738304 times) and 0.041 seconds to the Nelder-Mead algorithm. The speedup is very similar to the previous Hagan case.

In Table 3.9 market vs. model volatilities (both in %) for the first twelve smiles and the moneyness varying from -40% to 40% are shown. In addition, the mean relative error (MRE) considering the whole set of smiles is presented.

We also performed the equivalent calibration with Monte Carlo simulation thus obtaining the same parameters as in Table 3.8, except for $\phi_1 = -0.5714$. We note that the MAE in prices is 3.83×10^{-2} with formula (3.4), while MAE is 3.84×10^{-2} using Monte Carlo.

In Figure 3.5 the model fitting for the smiles of all forward rates is shown. Market volatilities are plotted with triangles, while model volatilities are shown in continuous line.

Calibration to swaptions

The calibrated parameters are $\eta_1 = 0.779175$ and $\lambda_1 = 2.722489$. In Table 3.10 market vs. model swaptions prices (in %) for the first fourteen swaptions and the moneyness varying from -40% to 40% are shown, each pair with its corresponding absolute error. In addition, for the whole set of swaptions the mean absolute error (MAE) is presented.

In Figures 3.6 and 3.7 the model fitting when considering the whole swaption matrix is shown. Market prices are shown using triangles and the model ones using stars.

3.4.3 Rebonato model

Calibration to caplets

The calibrated parameters are shown in Table 3.11. The execution time was 146.729 seconds, 119.913 seconds employed by the multi-GPU SA (launched with a more demanding configuration, specifically, $T_0 = 10$, $T_{min} = 0.01$, $\varphi = 0.99$, $L = 100$, $w = 256 \times 64$, $\#GPUs = 2$, the cost function was evaluated roughly two billion times) and the Nelder-Mead local optimization algorithm consumed the remaining time. When using the multi-GPU approach (see Figure 2.1) we obtain a speedup of 1.88 with respect to mono-GPU version, and of 207.796 with respect to sequential computations.

In Table 3.12, market vs. model volatilities for the smiles of F_1 to F_{12} and the moneyness -40% to 40% are shown. The mean relative error considering all smiles is presented.

We also performed the equivalent calibration with Monte Carlo simulation thus

obtaining the same parameters as in Table 3.11, except for $\phi_{11} = 0.0940$. We note that the MAE in prices is 3.43×10^{-2} with formula (3.4), while MAE is 3.39×10^{-2} using Monte Carlo.

In Figure 3.8 the model fitting for the smiles of all forward rates is shown. Market volatilities are plotted with triangles, while model volatilities are shown in continuous line.

Calibration to swaptions

The calibrated parameters are $\eta_1 = 0.650997$, $\lambda_1 = 3.617546$, $\eta_2 = 0.999000$, $\lambda_2 = 0.380984$ and $\lambda_3 = 0.001000$. Using two GPUs the execution time was approximately 2 hours, as in the previous models (by using a cluster of GPUs time could be substantially reduced). In Table 3.13 market vs. model swaptions prices (in %) for the first fourteen swaptions and the moneyness varying from -40% to 40% are shown, each pair with its corresponding absolute error. In addition, for the whole set of swaptions the mean absolute error is presented.

In Figures 3.9 and 3.10 the model fitting when considering the whole swaption matrix is shown. Market prices are shown using triangles, and the model ones using stars.

In the recent paper [122] an approximation formula for swaptions is proposed, so that we used it to check the obtained results with our Monte Carlo simulation. Thus, the calibration with the approximation formula provides the parameters $\eta_1 = 0.619778$, $\lambda_1 = 3.617546$, $\eta_2 = 0.858516$, $\lambda_2 = 0.380984$ and $\lambda_3 = 0.001000$. Moreover the obtained MAE with the approximation formula for swaptions is 1.05×10^{-1}, a bit larger than the one obtained with Monte Carlo ($MAE = 6.30 \times 10^{-2}$ as shown in Table 3.13).

Moneyness	Smile of F_1			Smile of F_2		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	97.26	100.61	3.44×10^{-2}	88.27	89.06	8.97×10^{-3}
-20%	82.58	87.53	6.00×10^{-2}	79.62	80.85	1.55×10^{-2}
0%	72.29	77.45	7.13×10^{-2}	73.03	74.70	2.28×10^{-2}
20%	70.89	70.36	7.48×10^{-3}	71.95	70.61	1.85×10^{-2}
40%	69.49	66.26	4.64×10^{-2}	70.87	68.59	3.21×10^{-2}
Moneyness	Smile of F_3			Smile of F_4		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	77.09	77.41	4.08×10^{-3}	57.08	57.14	1.05×10^{-3}
-20%	71.50	72.46	1.34×10^{-2}	53.21	54.37	2.18×10^{-2}
0%	67.93	68.77	1.24×10^{-2}	52.49	52.29	3.82×10^{-3}
20%	67.10	66.34	1.13×10^{-2}	51.34	50.90	8.56×10^{-3}
40%	66.41	65.17	1.88×10^{-2}	50.61	50.19	8.23×10^{-3}
Moneyness	Smile of F_5			Smile of F_6		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	56.69	56.92	4.05×10^{-3}	56.30	56.74	7.69×10^{-3}
-20%	52.43	53.22	1.51×10^{-2}	51.65	52.10	8.76×10^{-3}
0%	50.31	50.37	1.07×10^{-3}	48.19	48.48	6.03×10^{-3}
20%	48.72	48.36	7.29×10^{-3}	46.19	45.89	6.52×10^{-3}
40%	47.70	47.21	1.03×10^{-2}	44.91	44.32	1.31×10^{-2}
Moneyness	Smile of F_7			Smile of F_8		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	55.92	56.59	1.19×10^{-2}	55.54	56.47	1.68×10^{-2}
-20%	50.89	51.04	3.00×10^{-3}	50.13	50.01	2.35×10^{-3}
0%	46.19	46.70	1.09×10^{-2}	44.25	44.95	1.59×10^{-2}
20%	43.83	43.56	6.33×10^{-3}	41.56	41.28	6.80×10^{-3}
40%	42.32	41.61	1.67×10^{-2}	39.84	39.00	2.12×10^{-2}
Moneyness	Smile of F_9			Smile of F_{10}		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	55.16	56.39	2.22×10^{-2}	54.78	56.34	2.85×10^{-2}
-20%	49.39	49.04	7.16×10^{-3}	48.65	48.09	1.15×10^{-2}
0%	42.40	43.28	2.07×10^{-2}	40.61	41.65	2.54×10^{-2}
20%	39.43	39.11	8.06×10^{-3}	37.38	37.00	1.02×10^{-2}
40%	37.54	36.53	2.68×10^{-2}	35.34	34.15	3.36×10^{-2}
Moneyness	Smile of F_{11}			Smile of F_{12}		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	54.41	56.33	3.52×10^{-2}	54.03	56.35	4.28×10^{-2}
-20%	47.94	47.20	1.54×10^{-2}	47.22	46.34	1.87×10^{-2}
0%	38.93	40.09	2.99×10^{-2}	37.29	38.57	3.44×10^{-2}
20%	35.47	35.00	1.33×10^{-2}	33.63	33.04	1.76×10^{-2}
40%	33.30	31.92	4.16×10^{-2}	31.36	29.75	5.12×10^{-2}
$MRE = 1.80 \times 10^{-2}$						

Table 3.6: Hagan model, calibration to caplets, σ_{market} vs. σ_{model}.

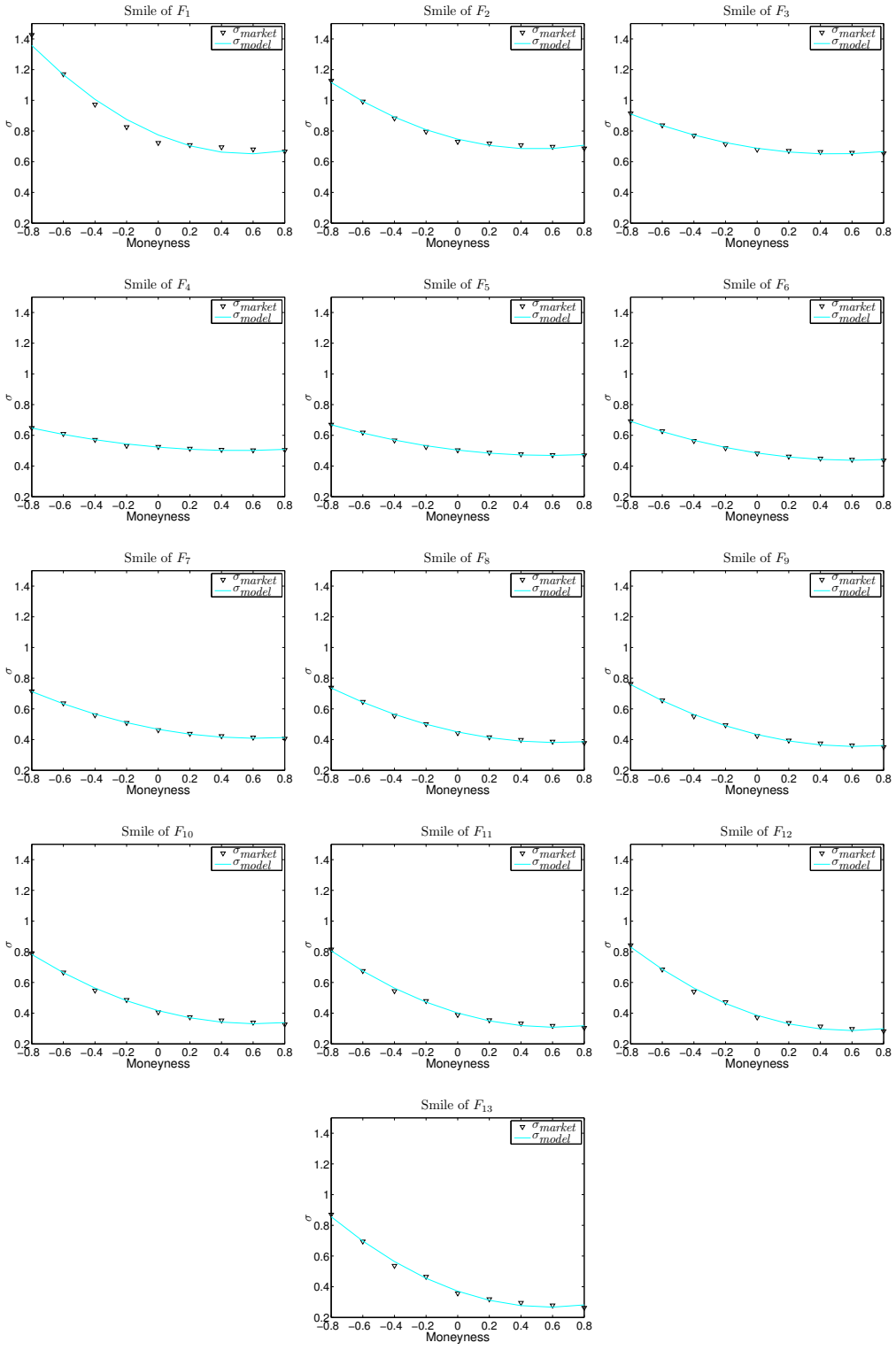


Figure 3.2: Hagan model, σ_{market} vs. σ_{model}, smiles of $F_1, \ldots, F_{13}$.

Moneyness	0.5×1 swaptions			1×1 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	0.4866	0.4842	2.40×10^{-3}	0.5917	0.5758	1.59×10^{-2}
-20%	0.3562	0.3628	6.60×10^{-3}	0.4661	0.4602	5.90×10^{-3}
0%	0.2356	0.2427	7.10×10^{-3}	0.3467	0.3450	1.70×10^{-3}
20%	0.1363	0.1390	2.70×10^{-3}	0.2394	0.2399	5.00×10^{-4}
40%	0.0680	0.0659	2.10×10^{-3}	0.1517	0.1539	2.20×10^{-3}
Moneyness	1.5×1 swaptions			2×1 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	0.7357	0.6840	5.17×10^{-2}	0.8184	0.7490	6.94×10^{-2}
-20%	0.5908	0.5548	3.60×10^{-2}	0.6603	0.6068	5.35×10^{-2}
0%	0.4536	0.4270	2.66×10^{-2}	0.5118	0.4651	4.67×10^{-2}
20%	0.3277	0.3095	1.82×10^{-2}	0.3754	0.3340	4.14×10^{-2}
40%	0.2213	0.2101	1.12×10^{-2}	0.2587	0.2229	3.58×10^{-2}
Moneyness	0.5×2 swaptions			1×2 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.0570	1.0144	4.26×10^{-2}	1.2427	1.1963	4.64×10^{-2}
-20%	0.7440	0.7275	1.65×10^{-2}	0.9322	0.9163	1.59×10^{-2}
0%	0.4555	0.4573	1.80×10^{-3}	0.6394	0.6460	6.60×10^{-3}
20%	0.2299	0.2418	1.19×10^{-2}	0.3886	0.4116	2.30×10^{-2}
40%	0.0925	0.1046	1.21×10^{-2}	0.2037	0.2343	3.06×10^{-2}
Moneyness	1.5×2 swaptions			2×2 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.4884	1.4260	6.24×10^{-2}	1.6938	1.6160	7.78×10^{-2}
-20%	1.1367	1.1168	1.99×10^{-2}	1.3077	1.2732	3.45×10^{-2}
0%	0.8059	0.8141	8.20×10^{-3}	0.9466	0.9320	1.46×10^{-2}
20%	0.5154	0.5446	2.92×10^{-2}	0.6269	0.6229	4.00×10^{-3}
40%	0.2919	0.3304	3.85×10^{-2}	0.3736	0.3748	1.20×10^{-3}
Moneyness	0.5×3 swaptions			1×3 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.7380	1.6538	8.42×10^{-2}	2.0341	1.9648	6.93×10^{-2}
-20%	1.1980	1.1506	4.74×10^{-2}	1.4851	1.4628	2.23×10^{-2}
0%	0.7011	0.6838	1.73×10^{-2}	0.9696	0.9812	1.16×10^{-2}
20%	0.3242	0.3277	3.50×10^{-3}	0.5413	0.5748	3.35×10^{-2}
40%	0.1128	0.1214	8.60×10^{-3}	0.2479	0.2882	4.03×10^{-2}
Moneyness	1.5×3 swaptions			2×3 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	2.3898	2.3012	8.86×10^{-2}	2.6885	2.6037	8.48×10^{-2}
-20%	1.7850	1.7586	2.64×10^{-2}	2.0311	2.0155	1.56×10^{-2}
0%	1.2175	1.2288	1.13×10^{-2}	1.4178	1.4296	1.18×10^{-2}
20%	0.7304	0.7676	3.72×10^{-2}	0.8856	0.9064	2.08×10^{-2}
40%	0.3749	0.4203	4.54×10^{-2}	0.4832	0.5044	2.12×10^{-2}
Moneyness	0.5×4 swaptions			1×4 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	2.5381	2.4226	1.15×10^{-1}	2.9426	2.8472	9.54×10^{-2}
-20%	1.7151	1.6480	6.71×10^{-2}	2.1123	2.0763	3.60×10^{-2}
0%	0.9584	0.9314	2.70×10^{-2}	1.3344	1.3375	3.10×10^{-3}
20%	0.4031	0.4035	4.00×10^{-4}	0.7016	0.7295	2.79×10^{-2}
40%	0.1188	0.1250	6.20×10^{-3}	0.2907	0.3268	3.61×10^{-2}
$MAE = 6.19 \times 10^{-2}$						

Table 3.7: Hagan model, calibration to swaptions, S_{Black} vs. S_{MC}, prices in %.

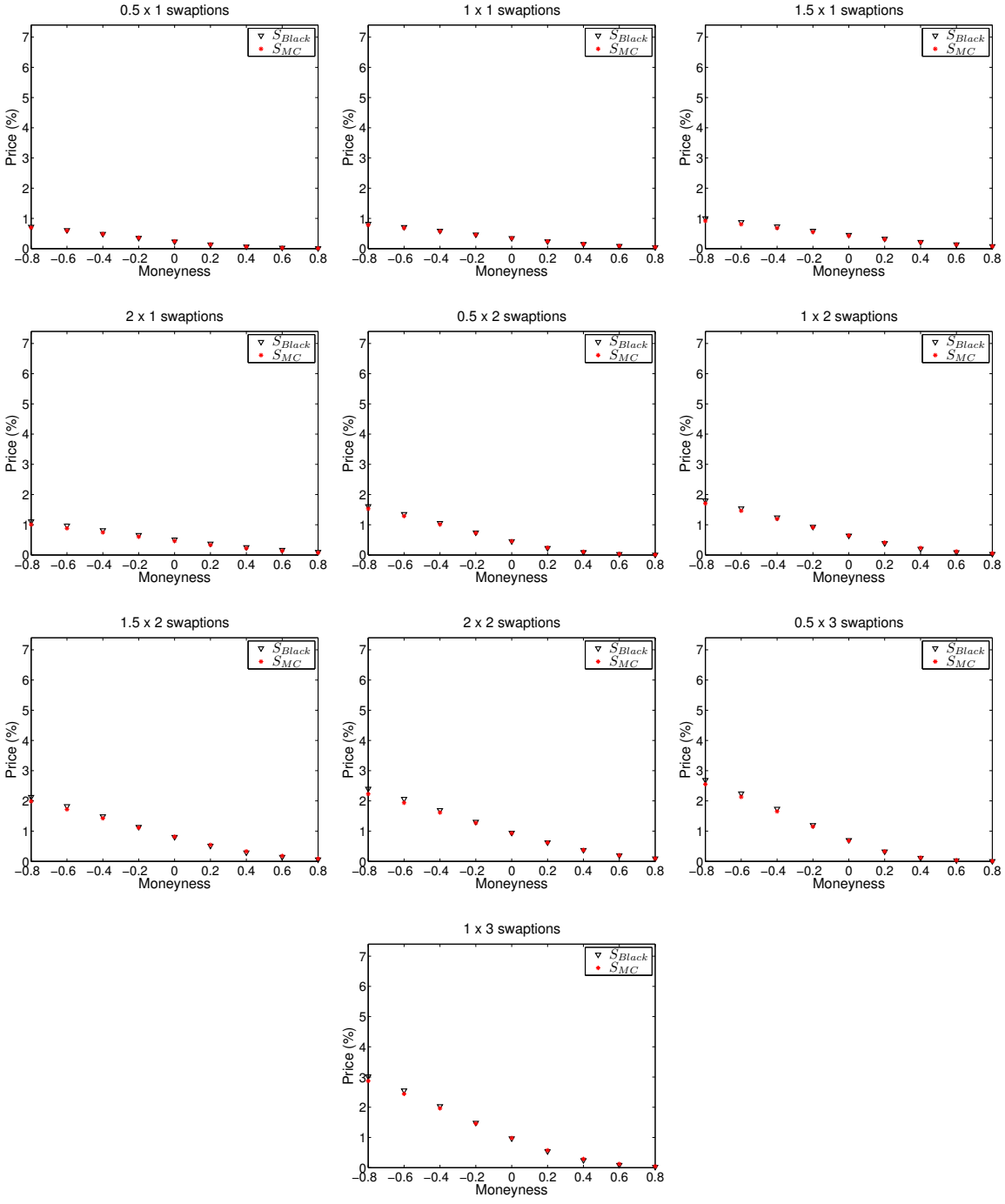


Figure 3.3: Hagan model, calibration to swaptions, S_{Black} vs. S_{MC}, part I.

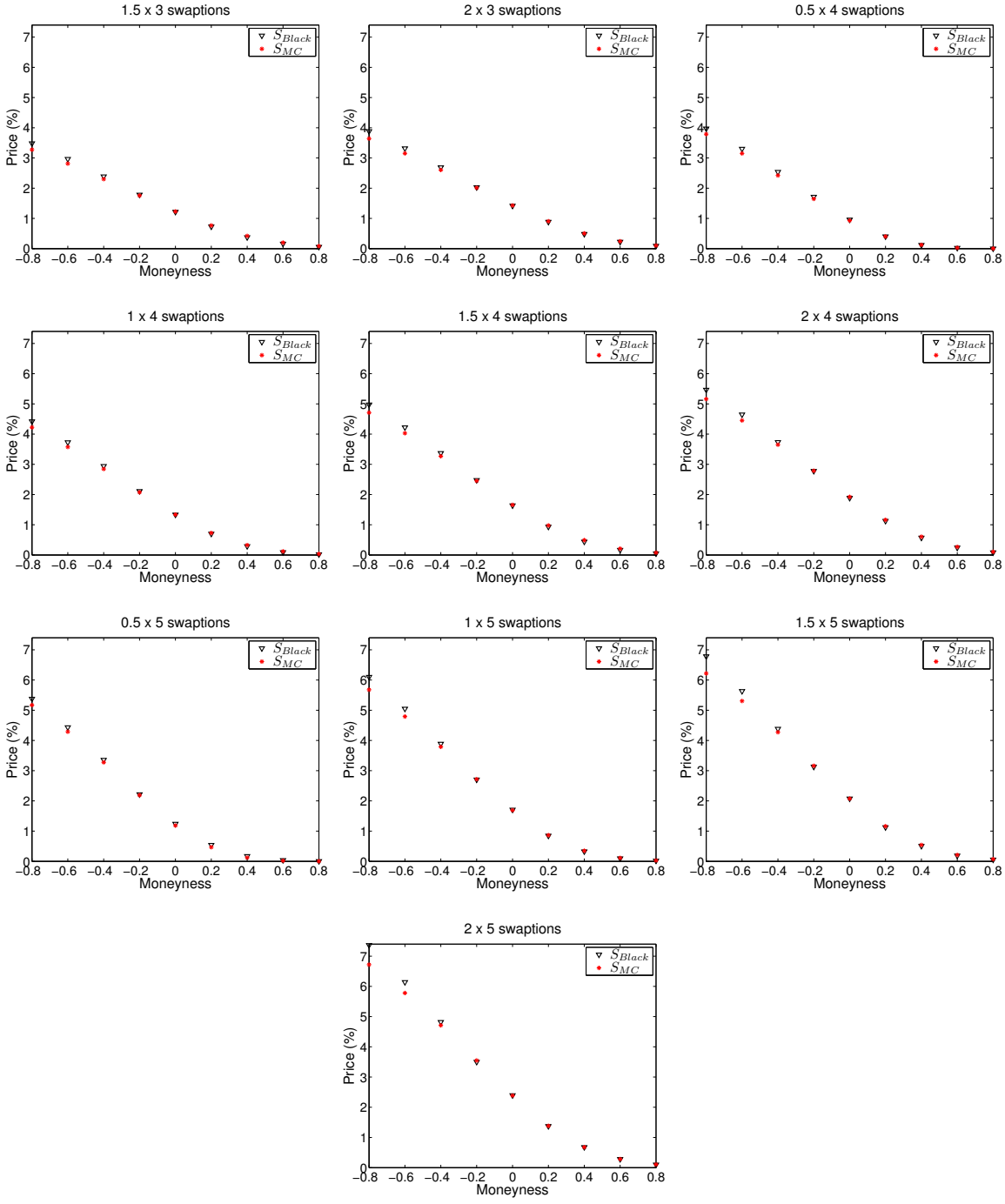


Figure 3.4: Hagan model, calibration to swaptions, S_{Black} vs. S_{MC}, part II.

	ϕ_i	α_i		ϕ_i	α_i
F_1	-0.7549	0.0888	F_8	-0.3661	0.0696
F_2	-0.2309	0.0842	F_9	-0.4770	0.0683
F_3	0.0666	0.0817	F_{10}	-0.5760	0.0693
F_4	0.1698	0.0662	F_{11}	-0.6615	0.0682
F_5	0.0302	0.0635	F_{12}	-0.7380	0.0682
F_6	-0.1098	0.0684	F_{13}	-0.8044	0.0669
F_7	-0.2417	0.0667			
$\sigma = 0.5986$					

Table 3.8: Mercurio & Morini model, calibration to caplets with SABR formula (3.4): calibrated parameters.

Moneyness	Smile of F_1			Smile of F_2		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	97.26	102.19	5.07×10^{-2}	88.27	89.59	1.50×10^{-2}
-20%	82.58	90.71	9.85×10^{-2}	79.62	81.81	2.75×10^{-2}
0%	72.29	81.16	1.23×10^{-1}	73.03	75.77	3.74×10^{-2}
20%	70.89	73.55	3.76×10^{-2}	71.95	71.47	6.69×10^{-3}
40%	69.49	67.88	2.31×10^{-2}	70.87	68.91	2.77×10^{-2}

Moneyness	Smile of F_3			Smile of F_4		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	77.09	77.13	4.45×10^{-4}	57.08	55.98	1.92×10^{-2}
-20%	71.50	71.99	6.92×10^{-3}	53.21	52.54	1.26×10^{-2}
0%	67.93	68.27	5.11×10^{-3}	52.49	50.39	4.00×10^{-2}
20%	67.10	65.96	1.69×10^{-2}	51.34	49.53	3.51×10^{-2}
40%	66.41	65.07	2.03×10^{-2}	50.61	49.97	1.27×10^{-2}

Moneyness	Smile of F_5			Smile of F_6		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	56.69	55.76	1.65×10^{-2}	56.30	55.70	1.08×10^{-2}
-20%	52.43	51.25	2.25×10^{-2}	51.65	50.20	2.81×10^{-2}
0%	50.31	48.26	4.08×10^{-2}	48.19	46.38	3.77×10^{-2}
20%	48.72	46.79	3.96×10^{-2}	46.19	44.25	4.21×10^{-2}
40%	47.70	46.83	1.82×10^{-2}	44.91	43.79	2.47×10^{-2}

Moneyness	Smile of F_7			Smile of F_8		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	55.92	55.77	2.78×10^{-3}	55.54	55.93	7.03×10^{-3}
-20%	50.89	49.40	2.92×10^{-2}	50.13	48.83	2.60×10^{-2}
0%	46.19	44.82	2.97×10^{-2}	44.25	43.55	1.58×10^{-2}
20%	43.83	42.03	4.12×10^{-2}	41.56	40.09	3.54×10^{-2}
40%	42.32	41.02	3.08×10^{-2}	39.84	38.45	3.49×10^{-2}

Moneyness	Smile of F_9			Smile of F_{10}		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	55.16	56.14	1.78×10^{-2}	54.78	56.39	2.94×10^{-2}
-20%	49.39	48.45	1.91×10^{-2}	48.65	48.23	8.78×10^{-3}
0%	42.40	42.56	3.80×10^{-3}	40.61	41.81	2.95×10^{-2}
20%	39.43	38.48	2.39×10^{-2}	37.38	37.15	6.21×10^{-3}
40%	37.54	36.21	3.53×10^{-2}	35.34	34.24	3.12×10^{-2}

Moneyness	Smile of F_{11}			Smile of F_{12}		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	54.41	56.66	4.14×10^{-2}	54.03	56.96	5.41×10^{-2}
-20%	47.94	48.13	3.97×10^{-3}	47.22	48.13	1.93×10^{-2}
0%	38.93	41.27	6.02×10^{-2}	37.29	40.89	9.65×10^{-2}
20%	35.47	36.08	1.72×10^{-2}	33.63	35.21	4.70×10^{-2}
40%	33.30	32.57	2.22×10^{-2}	31.36	31.11	7.79×10^{-3}

$MRE = 3.11 \times 10^{-2}$

Table 3.9: Mercurio & Morini model, calibration to caplets, σ_{market} vs. σ_{model}.

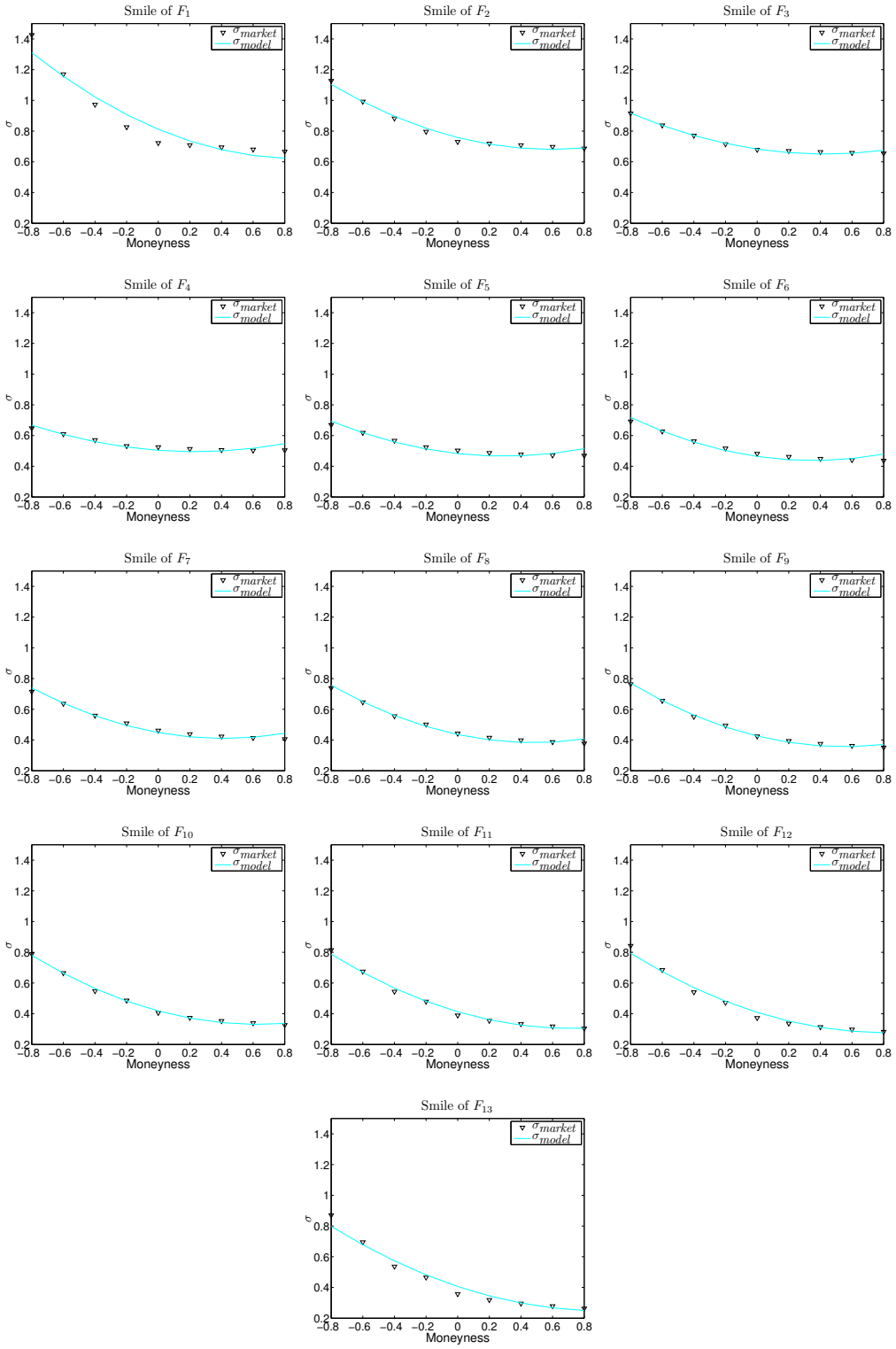


Figure 3.5: Mercurio & Morini model, σ_{market} vs. σ_{model}, smiles of $F_1, \ldots, F_{13}$.

Moneyness	0.5×1 swaptions			1×1 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	0.4866	0.4870	4.00×10^{-4}	0.5917	0.5870	4.70×10^{-3}
-20%	0.3562	0.3670	1.08×10^{-2}	0.4661	0.4699	3.80×10^{-3}
0%	0.2356	0.2478	1.22×10^{-2}	0.3467	0.3517	5.00×10^{-3}
20%	0.1363	0.1427	6.40×10^{-3}	0.2394	0.2422	2.80×10^{-3}
40%	0.0680	0.0657	2.30×10^{-3}	0.1517	0.1514	3.00×10^{-4}

Moneyness	1.5×1 swaptions			2×1 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	0.7357	0.6872	4.85×10^{-2}	0.8184	0.7465	7.19×10^{-2}
-20%	0.5908	0.5516	3.92×10^{-2}	0.6603	0.5959	6.44×10^{-2}
0%	0.4536	0.4170	3.66×10^{-2}	0.5118	0.4469	6.49×10^{-2}
20%	0.3277	0.2951	3.26×10^{-2}	0.3754	0.3137	6.17×10^{-2}
40%	0.2213	0.1957	2.56×10^{-2}	0.2587	0.2078	5.09×10^{-2}

Moneyness	0.5×2 swaptions			1×2 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.0570	1.0338	2.32×10^{-2}	1.2427	1.2143	2.84×10^{-2}
-20%	0.7440	0.7452	1.20×10^{-3}	0.9322	0.9266	5.60×10^{-3}
0%	0.4555	0.4679	1.24×10^{-2}	0.6394	0.6460	6.60×10^{-3}
20%	0.2299	0.2428	1.29×10^{-2}	0.3886	0.4038	1.52×10^{-2}
40%	0.0925	0.0984	5.90×10^{-3}	0.2037	0.2242	2.05×10^{-2}

Moneyness	1.5×2 swaptions			2×2 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.4884	1.4382	5.02×10^{-2}	1.6938	1.6298	6.40×10^{-2}
-20%	1.1367	1.1173	1.94×10^{-2}	1.3077	1.2746	3.31×10^{-2}
0%	0.8059	0.8024	3.50×10^{-3}	0.9466	0.9220	2.46×10^{-2}
20%	0.5154	0.5266	1.12×10^{-2}	0.6269	0.6116	1.53×10^{-2}
40%	0.2919	0.3182	2.63×10^{-2}	0.3736	0.3751	1.50×10^{-3}

Moneyness	0.5×3 swaptions			1×3 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.7380	1.6737	6.43×10^{-2}	2.0341	1.9880	4.61×10^{-2}
-20%	1.1980	1.1659	3.21×10^{-2}	1.4851	1.4761	9.00×10^{-3}
0%	0.7011	0.6868	1.43×10^{-2}	0.9696	0.9803	1.07×10^{-2}
20%	0.3242	0.3198	4.40×10^{-3}	0.5413	0.5666	2.53×10^{-2}
40%	0.1128	0.1112	1.60×10^{-3}	0.2479	0.2825	3.46×10^{-2}

Moneyness	1.5×3 swaptions			2×3 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	2.3898	2.3268	6.30×10^{-2}	2.6885	2.6302	5.83×10^{-2}
-20%	1.7850	1.7690	1.60×10^{-2}	2.0311	2.0258	5.30×10^{-3}
0%	1.2175	1.2215	4.00×10^{-3}	1.4178	1.4218	4.00×10^{-3}
20%	0.7304	0.7526	2.22×10^{-2}	0.8856	0.8935	7.90×10^{-3}
40%	0.3749	0.4168	4.19×10^{-2}	0.4832	0.5068	2.36×10^{-2}

Moneyness	0.5×4 swaptions			1×4 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	2.5381	2.4434	9.47×10^{-2}	2.9426	2.8764	6.62×10^{-2}
-20%	1.7151	1.6621	5.30×10^{-2}	2.1123	2.0935	1.88×10^{-2}
0%	0.9584	0.9298	2.86×10^{-2}	1.3344	1.3357	1.30×10^{-3}
20%	0.4031	0.3918	1.13×10^{-2}	0.7016	0.7174	1.58×10^{-2}
40%	0.1188	0.1160	2.80×10^{-3}	0.2907	0.3205	2.98×10^{-2}

$MAE = 5.50 \times 10^{-2}$

Table 3.10: Mercurio & Morini model, calibration to swaptions, S_{Black} vs. S_{MC}, prices in %.

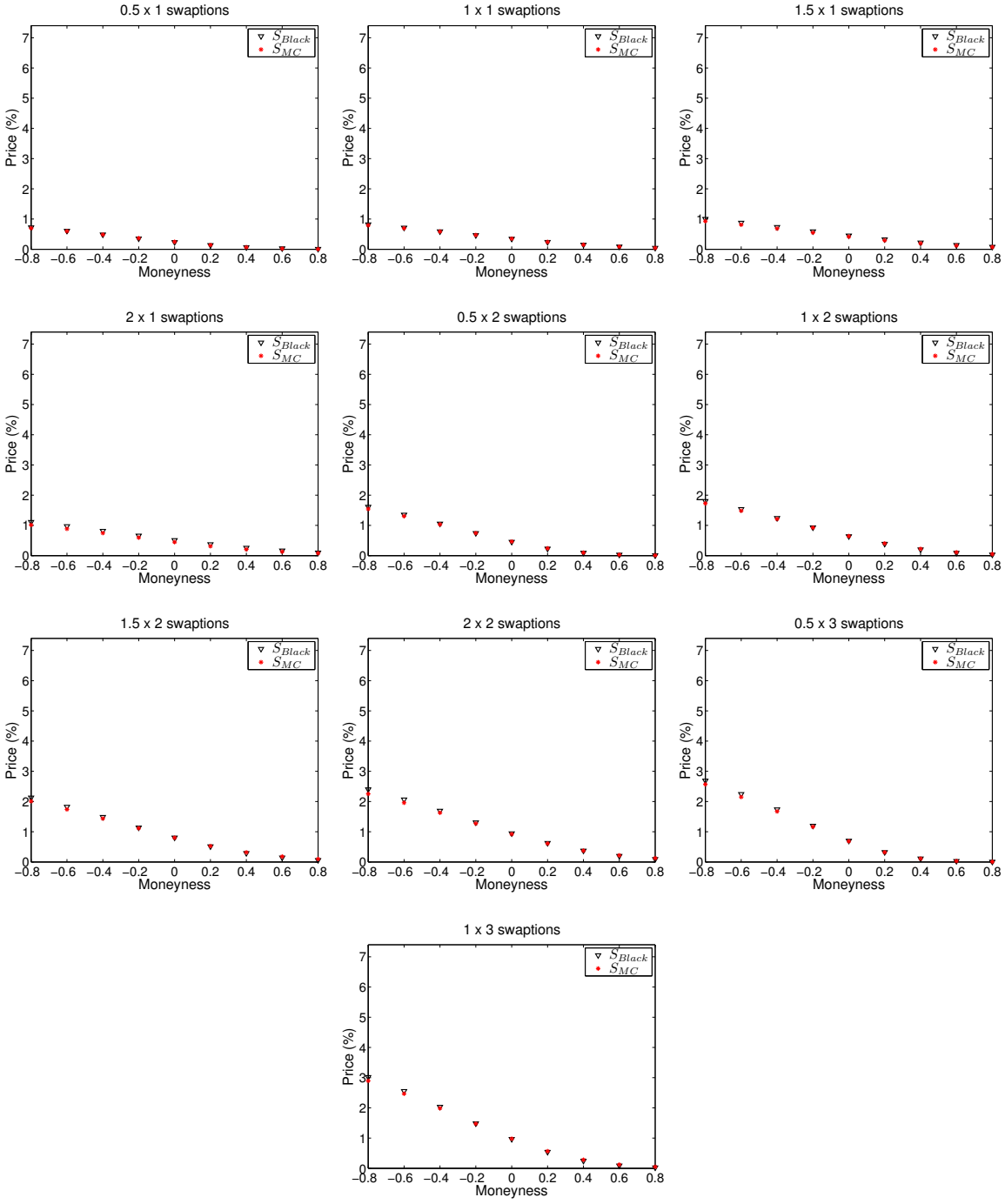


Figure 3.6: Mercurio & Morini model, calibration to swaptions, S_{Black} vs. S_{MC}, part I.

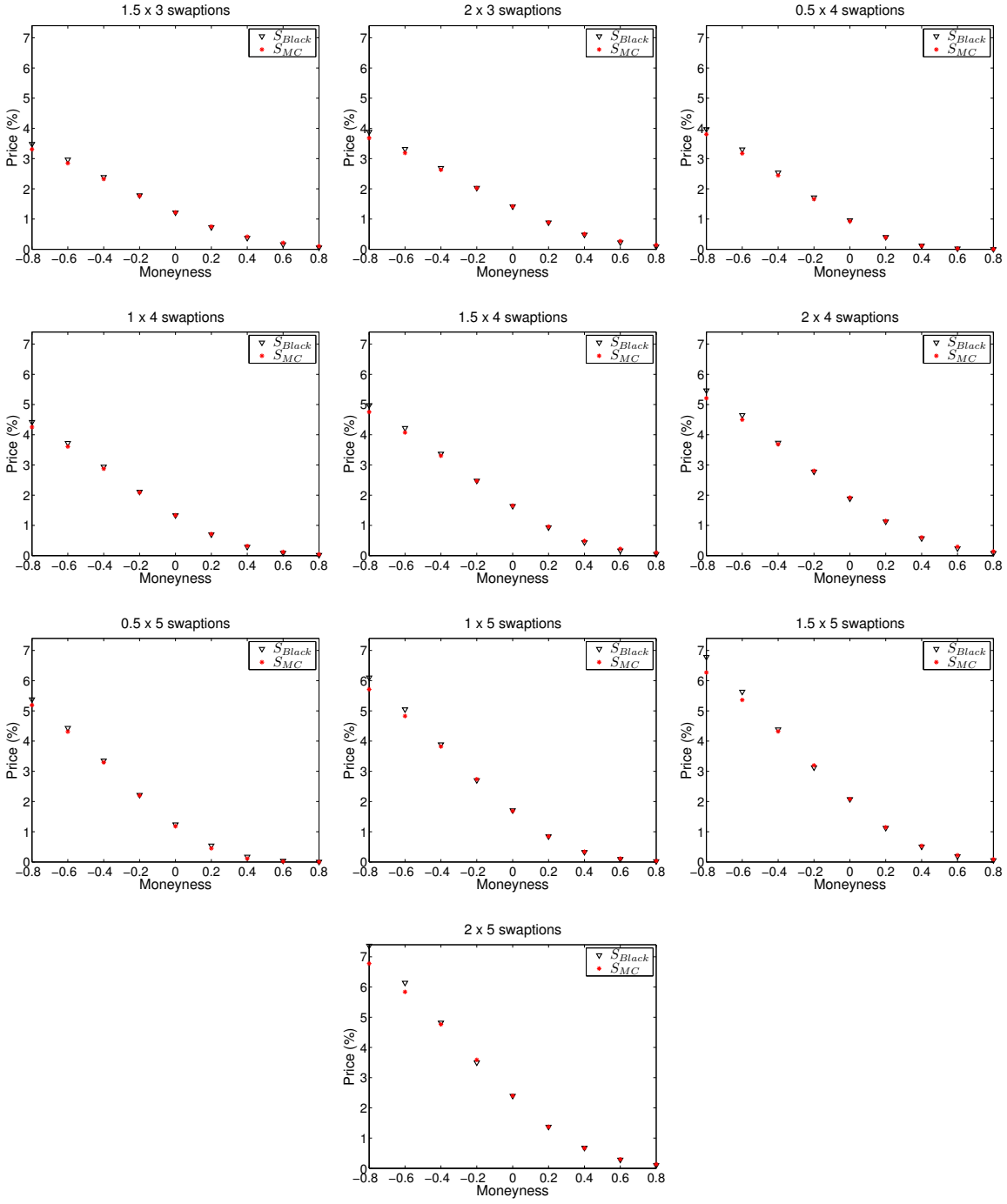


Figure 3.7: Mercurio & Morini model, calibration to swaptions, S_{Black} vs. S_{MC}, part II.

	ϕ_{ii}	κ_i		ϕ_{ii}	κ_i
F_1	-0.4060	0.0021	F_8	-0.4218	0.0010
F_2	-0.1935	0.0017	F_9	-0.5355	0.0010
F_3	0.0684	0.0015	F_{10}	-0.6466	0.0010
F_4	0.1825	0.0011	F_{11}	-0.7413	0.0010
F_5	0.0158	0.0010	F_{12}	-0.8175	0.0010
F_6	-0.1306	0.0010	F_{13}	-1.0000	0.0010
F_7	-0.2665	0.0010			
$a = 3.7789, \quad b = 44.7668,$ $c = 0.3076, \quad d = 25.3412.$			$\alpha = 0.0010, \quad \beta = 19.5812,$ $\gamma = 6.2339, \quad \delta = 0.5533.$		

Table 3.11: Rebonato model, calibration to caplets with SABR formula (3.4): calibrated parameters.

Moneyness	Smile of F_1			Smile of F_2		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	97.26	99.64	2.45×10^{-2}	88.27	89.14	9.83×10^{-3}
-20%	82.58	85.31	3.31×10^{-2}	79.62	81.00	1.73×10^{-2}
0%	72.29	74.78	3.45×10^{-2}	73.03	74.86	2.51×10^{-2}
20%	70.89	68.05	4.00×10^{-2}	71.95	70.74	1.68×10^{-2}
40%	69.49	65.12	6.28×10^{-2}	70.87	68.63	3.16×10^{-2}

Moneyness	Smile of F_3			Smile of F_4		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	77.09	77.26	2.13×10^{-3}	57.08	56.36	1.27×10^{-2}
-20%	71.50	72.20	9.78×10^{-3}	53.21	53.13	1.56×10^{-3}
0%	67.93	68.49	8.29×10^{-3}	52.49	51.00	2.85×10^{-2}
20%	67.10	66.13	1.45×10^{-2}	51.34	49.96	2.69×10^{-2}
40%	66.41	65.12	1.95×10^{-2}	50.61	50.02	1.17×10^{-2}

Moneyness	Smile of F_5			Smile of F_6		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	56.69	55.85	1.48×10^{-2}	56.30	56.13	3.11×10^{-3}
-20%	52.43	51.58	1.61×10^{-2}	51.65	50.99	1.28×10^{-2}
0%	50.31	48.60	3.42×10^{-2}	48.19	47.25	1.96×10^{-2}
20%	48.72	46.89	3.76×10^{-2}	46.19	44.92	2.75×10^{-2}
40%	47.70	46.46	2.59×10^{-2}	44.91	44.00	2.01×10^{-2}

Moneyness	Smile of F_7			Smile of F_8		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	55.92	56.74	1.46×10^{-2}	55.54	55.75	3.90×10^{-3}
-20%	50.89	50.82	1.45×10^{-3}	50.13	49.08	2.09×10^{-2}
0%	46.19	46.39	4.14×10^{-3}	44.25	43.95	6.82×10^{-3}
20%	43.83	43.44	8.87×10^{-3}	41.56	40.35	2.92×10^{-2}
40%	42.32	41.99	7.69×10^{-3}	39.84	38.28	3.92×10^{-2}

Moneyness	Smile of F_9			Smile of F_{10}		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	55.16	57.11	3.54×10^{-2}	54.78	56.57	3.26×10^{-2}
-20%	49.39	49.84	9.09×10^{-3}	48.65	48.84	3.91×10^{-3}
0%	42.40	44.10	4.00×10^{-2}	40.61	42.61	4.90×10^{-2}
20%	39.43	39.89	1.17×10^{-2}	37.38	37.85	1.26×10^{-2}
40%	37.54	37.21	8.79×10^{-3}	35.34	34.59	2.13×10^{-2}

Moneyness	Smile of F_{11}			Smile of F_{12}		
	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{\lvert\sigma_{market}-\sigma_{model}\rvert}{\sigma_{market}}$
-40%	54.41	57.20	5.13×10^{-2}	54.03	56.71	4.95×10^{-2}
-20%	47.94	49.07	2.36×10^{-2}	47.22	48.33	2.34×10^{-2}
0%	38.93	42.36	8.81×10^{-2}	37.29	41.29	1.07×10^{-1}
20%	35.47	37.07	4.51×10^{-2}	33.63	35.59	5.82×10^{-2}
40%	33.30	33.21	2.95×10^{-3}	31.36	31.23	3.95×10^{-3}

$MRE = 2.93 \times 10^{-2}$

Table 3.12: Rebonato model, calibration to caplets, σ_{market} vs. σ_{model}.

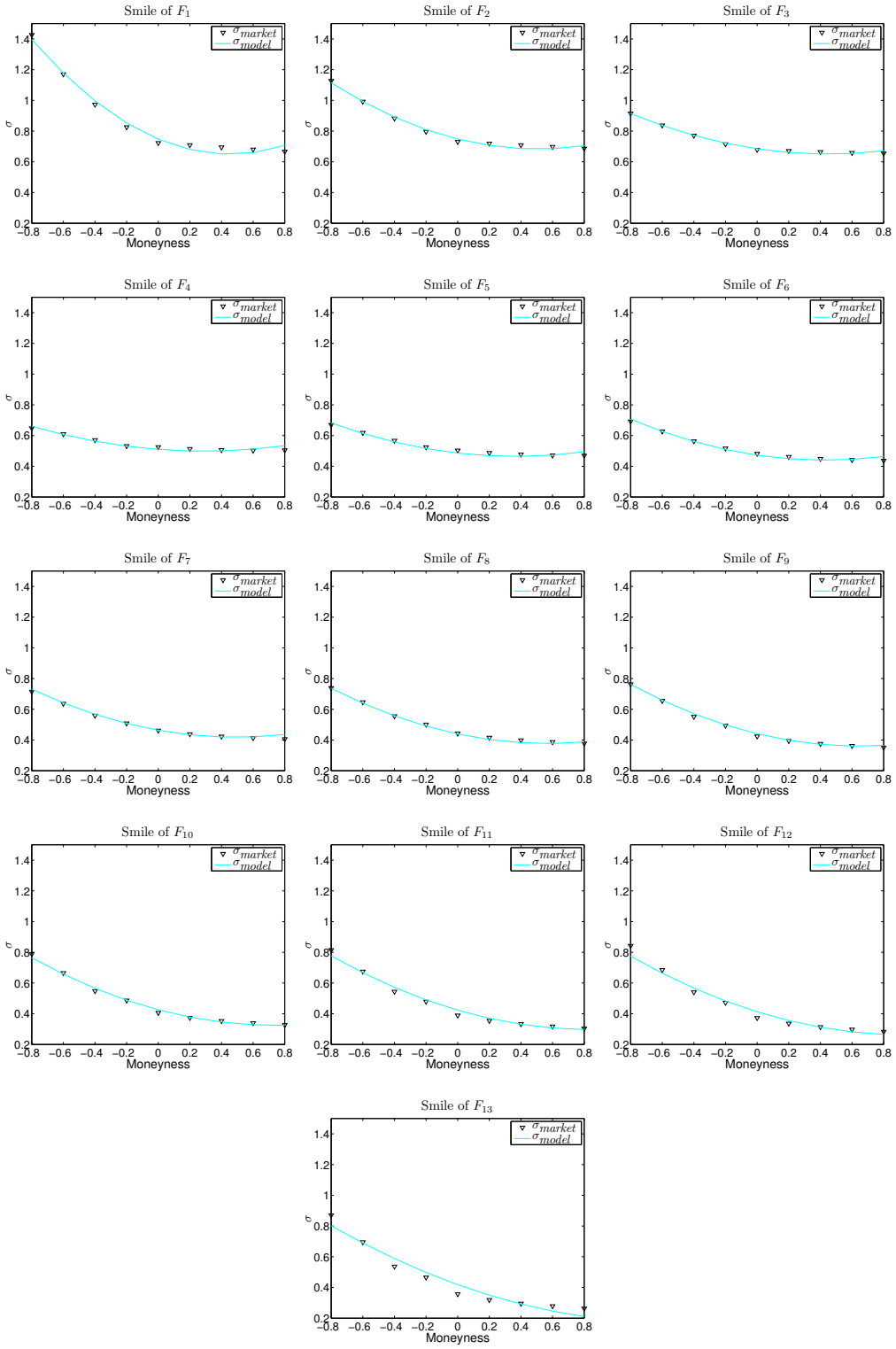


Figure 3.8: Rebonato model, σ_{market} vs. σ_{model}, smiles of $F_1, \ldots, F_{13}$.

Moneyness	0.5×1 swaptions			1×1 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	0.4866	0.4870	4.00×10^{-4}	0.5917	0.5839	7.80×10^{-3}
-20%	0.3562	0.3669	1.07×10^{-2}	0.4661	0.4693	3.20×10^{-3}
0%	0.2356	0.2477	1.21×10^{-2}	0.3467	0.3546	7.90×10^{-3}
20%	0.1363	0.1441	7.80×10^{-3}	0.2394	0.2488	9.40×10^{-3}
40%	0.0680	0.0699	1.90×10^{-3}	0.1517	0.1606	8.90×10^{-3}

Moneyness	1.5×1 swaptions			2×1 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	0.7357	0.6902	4.55×10^{-2}	0.8184	0.7465	7.19×10^{-2}
-20%	0.5908	0.5612	2.96×10^{-2}	0.6603	0.6028	5.75×10^{-2}
0%	0.4536	0.4339	1.97×10^{-2}	0.5118	0.4620	4.98×10^{-2}
20%	0.3277	0.3171	1.06×10^{-2}	0.3754	0.3354	4.00×10^{-2}
40%	0.2213	0.2188	2.50×10^{-3}	0.2587	0.2308	2.79×10^{-2}

Moneyness	0.5×2 swaptions			1×2 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.0570	1.0333	2.37×10^{-2}	1.2427	1.2175	2.52×10^{-2}
-20%	0.7440	0.7514	7.40×10^{-3}	0.9322	0.9400	7.80×10^{-3}
0%	0.4555	0.4841	2.86×10^{-2}	0.6394	0.6713	3.19×10^{-2}
20%	0.2299	0.2674	3.75×10^{-2}	0.3886	0.4369	4.83×10^{-2}
40%	0.0925	0.1237	3.12×10^{-2}	0.2037	0.2578	5.41×10^{-2}

Moneyness	1.5×2 swaptions			2×2 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.4884	1.4357	5.27×10^{-2}	1.6938	1.6184	7.54×10^{-2}
-20%	1.1367	1.1256	1.11×10^{-2}	1.3077	1.2721	3.56×10^{-2}
0%	0.8059	0.8250	1.91×10^{-2}	0.9466	0.9309	1.57×10^{-2}
20%	0.5154	0.5599	4.45×10^{-2}	0.6269	0.6292	2.30×10^{-3}
40%	0.2919	0.3520	6.01×10^{-2}	0.3736	0.3931	1.95×10^{-2}

Moneyness	0.5×3 swaptions			1×3 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	1.7380	1.6792	5.88×10^{-2}	2.0341	1.9953	3.88×10^{-2}
-20%	1.1980	1.1850	1.30×10^{-2}	1.4851	1.4966	1.15×10^{-2}
0%	0.7011	0.7235	2.24×10^{-2}	0.9696	1.0176	4.80×10^{-2}
20%	0.3242	0.3653	4.11×10^{-2}	0.5413	0.6130	7.17×10^{-2}
40%	0.1128	0.1484	3.56×10^{-2}	0.2479	0.3241	7.62×10^{-2}

Moneyness	1.5×3 swaptions			2×3 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	2.3898	2.3112	7.86×10^{-2}	2.6885	2.6048	8.37×10^{-2}
-20%	1.7850	1.7661	1.89×10^{-2}	2.0311	2.0098	2.13×10^{-2}
0%	1.2175	1.2360	1.85×10^{-2}	1.4178	1.4192	1.40×10^{-3}
20%	0.7304	0.7797	4.93×10^{-2}	0.8856	0.9005	1.49×10^{-2}
40%	0.3749	0.4417	6.68×10^{-2}	0.4832	0.5124	2.92×10^{-2}

Moneyness	0.5×4 swaptions			1×4 swaptions		
	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$	S_{Black}	S_{MC}	$\lvert S_{Black} - S_{MC} \rvert$
-40%	2.5381	2.4493	8.88×10^{-2}	2.9426	2.8751	6.75×10^{-2}
-20%	1.7151	1.6834	3.17×10^{-2}	2.1123	2.1053	7.00×10^{-3}
0%	0.9584	0.9709	1.25×10^{-2}	1.3344	1.3649	3.05×10^{-2}
20%	0.4031	0.4401	3.70×10^{-2}	0.7016	0.7572	5.56×10^{-2}
40%	0.1188	0.1506	3.18×10^{-2}	0.2907	0.3538	6.31×10^{-2}
$MAE = 6.30 \times 10^{-2}$						

Table 3.13: Rebonato model, calibration to swaptions, S_{Black} vs. S_{MC}, prices in %.

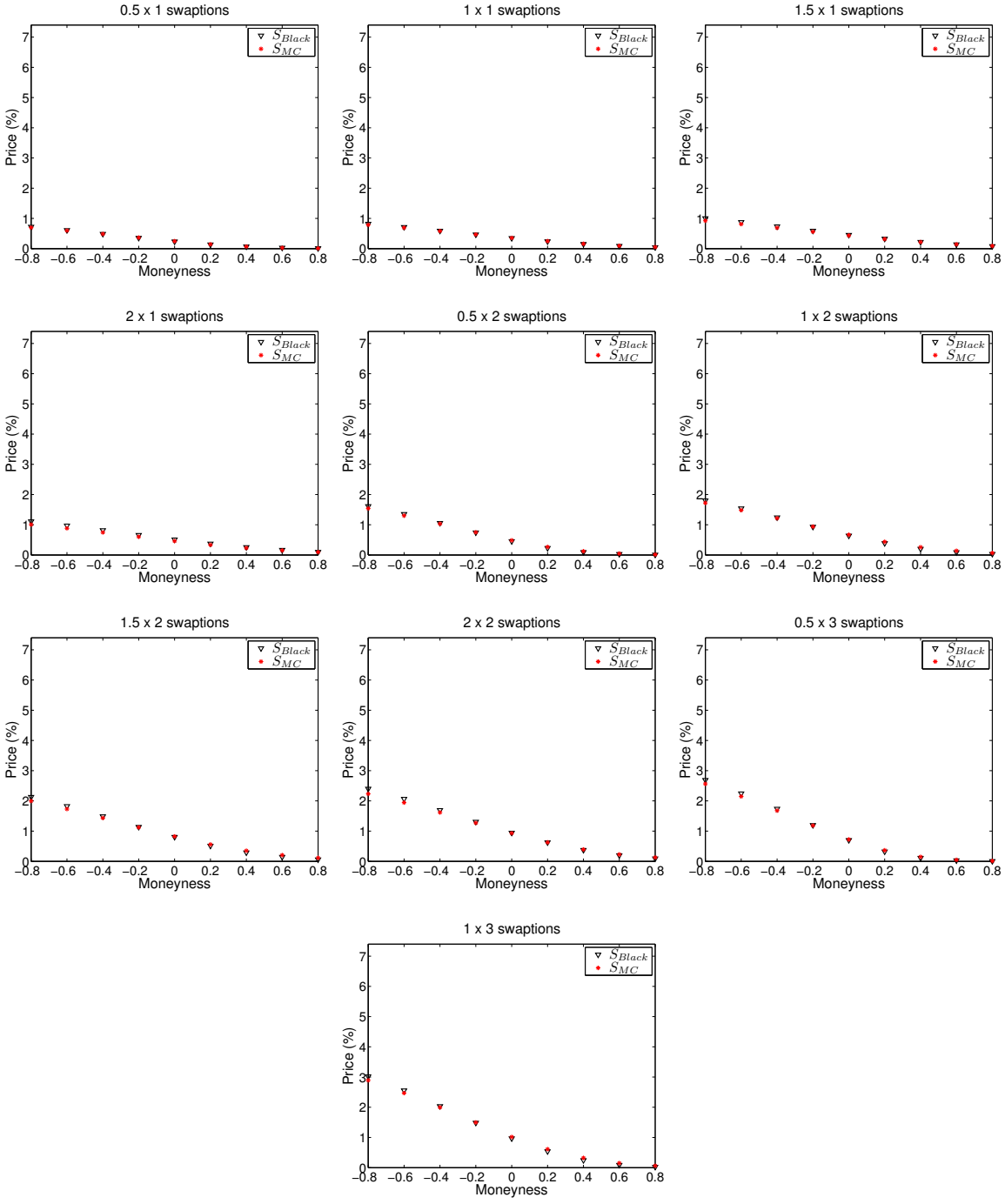


Figure 3.9: Rebonato model, calibration to swaptions, S_{Black} vs. S_{MC}, part I.

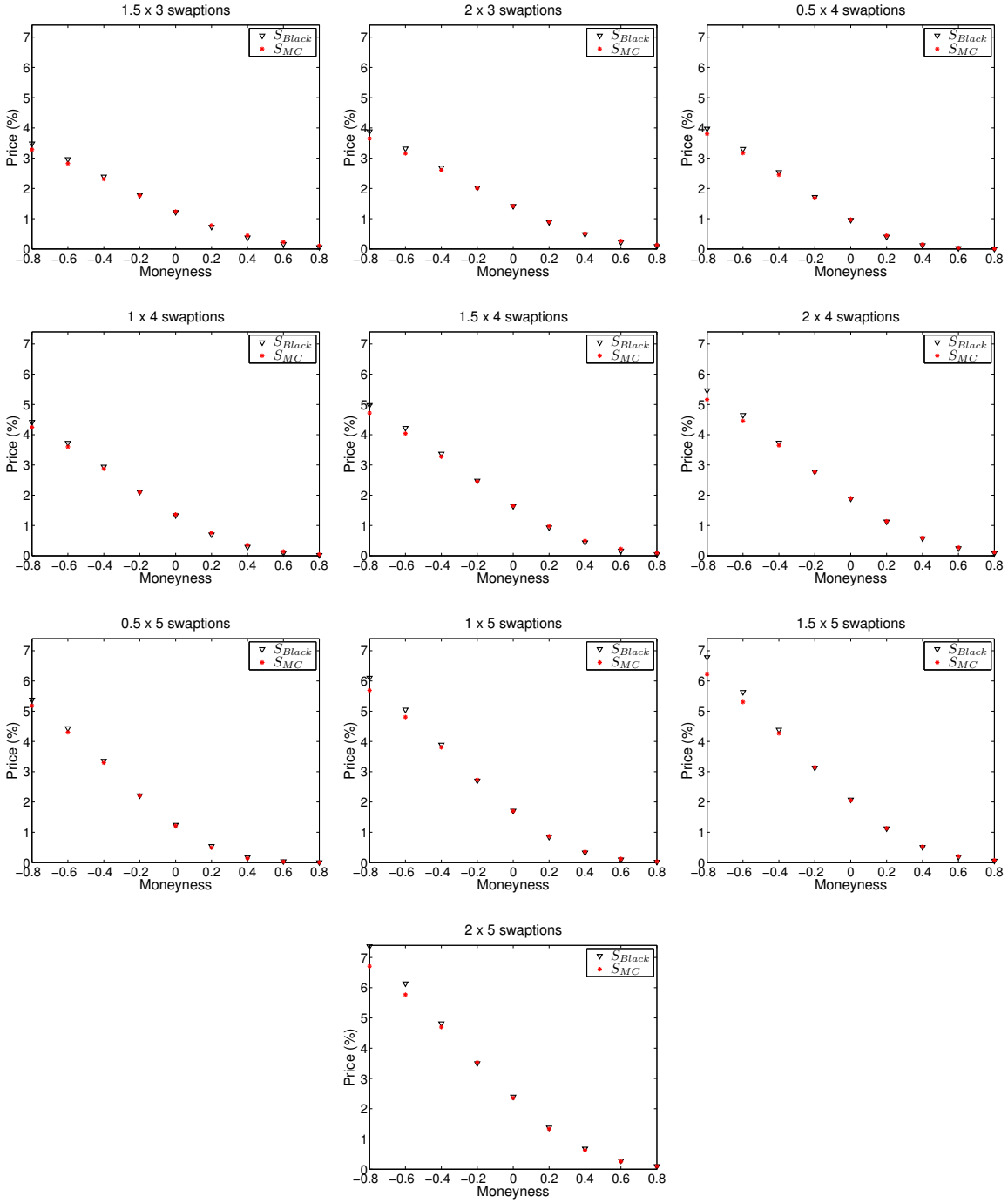


Figure 3.10: Rebonato model, calibration to swaptions, S_{Black} vs. S_{MC}, part II.

3.5 Conclusions

As a summary, in Table 3.14 the mean errors obtained in the calibration to caplets and swaptions of the three previous models are shown. The same measures are used, MRE for the volatilities and $MAE(\%)$ for the swaptions. In both cases, the model which achieves the best fit is highlighted.

	Hagan	Mercurio & Morini	Rebonato
Caplets (MRE)	$\mathbf{1.80 \times 10^{-2}}$	3.11×10^{-2}	2.93×10^{-2}
Swaptions (MAE)	6.19×10^{-2}	$\mathbf{5.50 \times 10^{-2}}$	6.30×10^{-2}

Table 3.14: Mean relative errors of the three models.

The three presented models are able to correctly capture market data. An indicator of the quality of the fit is the one used by Piterbarg in [117]: a mean absolute error considered acceptable in the calibration to swaptions is 0.1%. The three models have mean absolute errors less than this value.

In the case of the calibration to the smiles of the forward rates, Hagan model achieves the best fit, followed by Rebonato and Mercurio & Morini models. In the case of the calibration to the smiles of the swap rates, Mercurio & Morini model is the best one, followed by Hagan and Rebonato models.

Therefore, a model with one single volatility factor is able to obtain a satisfactory fit to the swaption market. Mercurio & Morini argue that models with only one stochastic volatility disturbance can capture better market regularities on the movements of the term structure, while when each rate is calibrated independently of the others the important common factors driven the market could be missed.

In Hagan model, and mainly in Rebonato model, a set of parameters must be specified for each forward rate. This may lead to overparameterization, with risk of instability, considering also the presence of many cross-correlations between stochastic volatilities not easy to determine based on market quotes. This issue is reflected in our calibrations: in the case of the calibration to caplets, Hagan and Mercurio &

Morini models are easier to calibrate than the Rebonato one. When dealing with the calibration to swaptions, Mercurio & Morini model is also simpler to calibrate than the other two models.

Once the models have been calibrated, when pricing products using Monte Carlo simulation, the most relevant factor in execution times is the number of processes to be simulated. Obviously, regarding this issue, Mercurio & Morini model is the fastest. The pricing of caplets with the Mercurio & Morini model is approximately 1.40 times faster than the pricing using the other two models. Although the difference is not huge, the fact that a model is a little faster or slower, could have a big impact in the execution time of a calibration process using Monte Carlo simulation.

Moreover, in order to validate the proposed Monte Carlo calibration approach we have successfully compared its results with the ones obtained by using the classical SABR formula for caplets and the more recent approximated formula for swaptions.

Note that the speedup with GPUs of the Monte Carlo calibration techniques can be applied to more complex products, for example CMS options or CMS spread options which contain more information on the smile structure and the correlation of LIBOR rates. In these and other complex products it is not clear that alternative approximation formulas are easily available and accurate enough [87].

As a brief final conclusion, for the set of used market data, the model with the best performance is the Mercurio & Morini one, since it is the easiest to calibrate, it achieves the best fit to the swaption market prices and it results the fastest one in the pricing with Monte Carlo simulation. The main drawback of the Rebonato model comes from its complexity in the calibration procedure. The performance of Hagan model falls in between the other two models: market data are reasonable well fitted and the model results not overly difficult to calibrate.

Chapter 4

SABR/LIBOR market models: PDE approach

4.1 Introduction

While Monte Carlo [49] simulation remains the industry's tool of choice for pricing interest rate derivatives within *LIBOR market model* (LMM) frameworks and related, several difficulties motivate researchers appeal to alternative *partial differential equation* (PDE) approaches. The first issue is that the convergence of Monte Carlo methods, although it depends only very weakly on the dimension of the problem, is very slow. Indeed, if the standard deviation of the result using a single simulation is ϵ then the standard deviation of the error after N simulations is $\epsilon/\sqrt{N}$. Therefore, to improve the accuracy of the solution by a factor of 10, 100 times as many simulations must be performed. The second drawback of Monte Carlo methods is the valuation of options with early-exercise, like in the case of the American options, due to the so-called "Monte Carlo on Monte Carlo" effect. Available Monte Carlo methods for American options are also quite costly, see [100] for example. In contrast, the modification of the PDE to a linear complementarity problem is usually straightforward. Finally, the weakest point of Monte Carlo methods appears to be the computation of the sensitivities of the solution with respect to the underlyings,

the so-called "Greeks", which are very used by traders, and are directly given by the partial derivatives of the PDE solution. Besides, path-dependent options, like barrier options, can be easily priced in the PDE context where only the boundary conditions need to be changed, in contrast to Monte Carlo methods, where Brownian bridge techniques [53] must be applied.

Many problems in finance lead to high dimensional PDEs which need to be solved efficiently. In order to cope with the so-called curse of dimensionality several methods are available in the literature, see [48, 12] for example, which can be put into three categories. The first group uses the Karhunen-Loeve transformation to reduce the stochastic differential equation to a lower dimensional equation, therefore this results in a lower dimensional PDE associated to the previously reduced SDE. The second category gathers those methods which try to reduce the dimension of the PDE itself, like for example dimension-wise decomposition algorithms. Finally, the third category groups the methods which reduce the complexity of the problem in the discretization layer, like for example the method of sparse grids, which we use in the present chapter.

4.2 Derivation of the PDE from the stochastic processes

In the previous chapter we have analyzed the three SABR/LIBOR market models proposed by Hagan, Mercurio & Morini and Rebonato using Monte Carlo simulation in order to price several interest rate derivatives. We have concluded that the Mercurio & Morini model is the one with the best performance: it is the easiest to calibrate, it achieves the best fit to swaption market prices and it results the fastest one in the pricing with Monte Carlo simulation. Taking into account these reasons, we mainly choose this model in the present chapter to develop the numerical solution of each PDE formulation. Nevertheless, at the end of this section the PDEs for the models of Hagan and Rebonato will be also posed.

Next, consider a set of $N-1$ LIBOR forward rates F_i, $1 \leq i \leq N-1$, $\mathbf{F} =$

$(F_1, \ldots, F_{N-1})$ on the tenor structure $[T_0, T_1, \ldots, T_{N-1}, T_N]$, $\tau_i = T_{i+1} - T_i$. The Mercurio & Morini model is defined by the following system of stochastic differential equations [103]:

$$\begin{aligned} dF_i(t) &= \mu_i(t) F_i(t)^\beta dt + \alpha_i V(t) F_i(t)^\beta dW_i^{\mathcal{Q}}(t), \quad F_i(0) \text{ given}, \\ dV(t) &= \sigma V(t) dZ^{\mathcal{Q}}(t), \quad V(0) = \alpha, \end{aligned} \tag{4.1}$$

on a probability space $\{\Omega, \mathcal{F}, \mathcal{P}\}$ with filtration $\{\mathcal{F}_t\}$, $t \in [T_0, T_N]$. Here μ_i is the drift of the i-th forward rate, $\beta \in [0, 1]$ is the local volatility coefficient, α_i is a deterministic (constant) instantaneous volatility coefficient, $W_i^{\mathcal{Q}}$ are standard Brownian motions under the risk neutral measure $\mathcal{Q}$, ρ is the correlation matrix between the forward rates, i.e.

$$< dW_i^{\mathcal{Q}}(t), dW_j^{\mathcal{Q}}(t) >= \rho_{ij} dt, \quad \forall i, j \in \{1, \ldots, N-1\},$$

V is the stochastic volatility of the forward rates, $dZ^{\mathcal{Q}}$ is a standard Brownian motion correlated with the Brownian motions of the forward rates and ϕ is the correlation vector between the forward rates and the stochastic volatility, i.e.

$$< dW_i^{\mathcal{Q}}(t), dZ^{\mathcal{Q}}(t) >= \phi_i dt, \quad \forall i \in \{1, \ldots, N-1\}.$$

Due to the fact that the volatility process is lognormal, one can set the initial value of the volatility equal to one, i.e. $\alpha = 1$ with no loss of generality, since any different initial value can be embedded in the model by adjusting the deterministic coefficients α_i. This is the choice we adopt in the following.

The drifts of the forward rates are determined by the chosen numeraire. Under the terminal probability measure $\mathcal{Q}^{T_N}$ associated with choosing the bond $P(t, T_N)$ as numeraire, the drifts of the forwards rates are given by

$$\mu_i(t) = \begin{cases} -\alpha_i V(t)^2 \displaystyle\sum_{j=i+1}^{N-1} \frac{\tau_j F_j(t)^\beta}{1 + \tau_j F_j(t)} \rho_{ij} \alpha_j & \text{if} \quad j < N-1, \\ 0 & \text{if} \quad j = N-1. \end{cases}$$

Under the spot probability measure associated to the bank-account numeraire $\beta(t)$,

$$\beta(t) = \prod_{j=0}^{i-1} \big(1 + \tau_j F_j(T_j)\big) \text{ if } t \in [T_i, T_{i+1}],$$

the drift terms of the forward rates are given by

$$\mu_i(t) = \alpha_i V(t)^2 \sum_{j=h(t)}^{i} \frac{\tau_j F_j(t)^\beta}{1 + \tau_j F_j(t)} \rho_{ij} \alpha_j,$$

where $h(t)$ denotes the index of the first unfixed F_i, i.e.

$$h(t) = j, \text{ if } t \in [T_{j-1}, T_j).$$

Our model for the correlation structure is taken from Rebonato [119], who suggests the time independent function

$$\rho_{ij} = \mathrm{e}^{-\lambda |T_i - T_j|}. \tag{4.2}$$

This function reflects the fact that the correlation increases as the time between the forward rates expiry decreases, so that two consecutive forward rates influence each other more than a forward rate in many years time.

A European option is characterized by its payoff function G, which determines the amount $G(T, \mathbf{F}(T), V(T))$ its holder receives at time $t = T$. The arbitrage-free value of the option relative to a numeraire $\mathcal{N}$ is then given by

$$u(t, \mathbf{F}(t), V(t)) = \mathbb{E}^{\mathcal{Q}} \left(\frac{G(T, \mathbf{F}(T), V(T))}{\mathcal{N}(T)} \,\middle|\, \mathcal{F}_t \right). \tag{4.3}$$

Closed-form solutions based on (4.3) are rarely available due to the multi-asset feature of most LIBOR derivatives. In the next paragraphs we sketch the derivation of the PDE formulation associated to the Mercurio & Morini model.

By using Itô's formula, see [127] for example, the stochastic differential equation

for u is given by

$$\begin{aligned} du(t,\mathbf{F}(t),V(t)) \;&= \frac{\partial u}{\partial t}dt + \sum_{i=1}^{N-1}\frac{\partial u}{\partial F_i(t)}dF_i(t) + \frac{\partial u}{\partial V(t)}dV(t) + \\ &\frac{1}{2}\sum_{i,j=1}^{N-1}\frac{\partial^2 u}{\partial F_i(t)\partial F_j(t)}dF_i(t)dF_j(t) + \frac{1}{2}\frac{\partial^2 u}{\partial V(t)^2}(dV(t))^2 + \\ &\sum_{i=1}^{N-1}\frac{\partial^2 u}{\partial F_i(t)\partial V(t)}dF_i(t)dV(t), \end{aligned} \tag{4.4}$$

with box algebra:

	dt	$dW_i^{\mathcal{Q}}$	$dW_j^{\mathcal{Q}}$	$dZ^{\mathcal{Q}}$
dt	0	0	0	0
$dW_i^{\mathcal{Q}}$	0	dt	$\rho_{ij}dt$	$\phi_i dt$
$dW_j^{\mathcal{Q}}$	0	$\rho_{ij}dt$	dt	$\phi_j dt$
$dZ^{\mathcal{Q}}$	0	$\phi_i dt$	$\phi_j dt$	dt

.

The interpretation of the box algebra [102] is the following. In an expansion to terms of order dt, as $dt \to 0$ higher order terms such as $(dt)^j$ are all negligible for $j > 0$. For example, $(dt)^2$ is of order 0 as $dt \to 0$, which is denoted as $(dt)(dt) \sim 0$. Similarly, cross terms such as $(dt)(dW_i^{\mathcal{Q}})$ are negligible because the increment $dW_i^{\mathcal{Q}}$ is normally distributed with mean 0 and standard deviation $(dt)^{1/2}$ and so $(dt)(dW_i^{\mathcal{Q}})$ has standard deviation $(dt)^{3/2}$ which tends to 0 as $dt \to 0$.

Substituting equations (4.1) in (4.4) and using the box algebra, one can obtain

$$\begin{aligned} du(t,\mathbf{F}(t),V(t)) = \Bigg(&\frac{\partial u}{\partial t} + \sum_{i=1}^{N-1}\mu_i(t)F_i(t)^\beta\frac{\partial u}{\partial F_i(t)} + \\ &\frac{1}{2}\sum_{i,j=1}^{N-1}\alpha_i\alpha_j V(t)^2 F_i(t)^\beta F_j(t)^\beta\rho_{ij}\frac{\partial^2 u}{\partial F_i(t)\partial F_j(t)} + \\ &\frac{1}{2}\sigma^2 V(t)^2\frac{\partial^2 u}{\partial V(t)^2} + \sum_{i=1}^{N-1}\sigma V(t)^2\alpha_i F_i(t)^\beta\phi_i\frac{\partial^2 u}{\partial F_i(t)\partial V(t)}\Bigg)dt + \\ &\sum_{i=1}^{N-1}\alpha_i V(t)F_i(t)^\beta\frac{\partial u}{\partial F_i(t)}dW_i^{\mathcal{Q}} + \sigma V(t)\frac{\partial u}{\partial V(t)}dZ^{\mathcal{Q}}. \end{aligned} \tag{4.5}$$

In order to comply with the no-arbitrage conditions and (4.3), the process $du(t, \mathbf{F}, V)$ has to be martingale under the measure $\mathcal{Q}$. Thus, to satisfy this requirement, the drift term dt in (4.5) must be equal to zero. The same result could be directly obtained by applying Feynman-Kac theorem, see [127, 19]. The final parabolic PDE takes the following form:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sigma^2 V^2 \frac{\partial^2 u}{\partial V^2} + \frac{1}{2} V^2 \sum_{i,j=1}^{N-1} \rho_{ij}\alpha_i\alpha_j F_i^\beta F_j^\beta \frac{\partial^2 u}{\partial F_i \partial F_j} + \sigma V^2 \sum_{i=1}^{N-1} \phi_i \alpha_i F_i^\beta \frac{\partial^2 u}{\partial F_i \partial V} + \sum_{i=1}^{N-1} \mu_i(t) F_i^\beta \frac{\partial u}{\partial F_i} = 0, \tag{4.6}$$

with the terminal condition given by the derivative payoff,

$$u(T, \mathbf{F}(T), V(T)) = g(T, \mathbf{F}(T), V(T)),$$

on $[0, T] \times \mathbb{R}^{N-1} \times \mathbb{R}$. For simplicity of notation, we have used the relative payoff $g(\cdot) = \dfrac{G(\cdot)}{\mathcal{N}(T)}$.

Analytic solutions for (4.6) can be only found for suitable simple specifications of the functionals forms of the PDE and for straightforward boundary conditions (e.g. simple caplets without stochastic volatility, i.e. $\sigma = 0$, see Section 4.3.2).

Finally, we are going to present the PDE for Hagan model, which is defined by the following system of stochastic differential equations [68]:

$$\begin{aligned} dF_i(t) &= \mu^{F_i}(t) F_i(t)^{\beta_i} dt + V_i(t) F_i(t)^{\beta_i} dW_i^{\mathcal{Q}}(t), \quad F_i(0) \text{ given}, \\ dV_i(t) &= \mu^{V_i}(t) V_i(t) dt + \sigma_i V_i(t) dZ_i^{\mathcal{Q}}(t), \quad V_i(0) = \alpha_i, \end{aligned} \tag{4.7}$$

with the associated correlations denoted by

$$\begin{aligned} < dW_i^{\mathcal{Q}}(t), dW_j^{\mathcal{Q}}(t) > &= \rho_{ij} dt, \\ < dW_i^{\mathcal{Q}}(t), dZ_j^{\mathcal{Q}}(t) > &= \phi_{ij} dt, \\ < dZ_i^{\mathcal{Q}}(t), dZ_j^{\mathcal{Q}}(t) > &= \theta_{ij} dt. \end{aligned}$$

The PDE for this model is obtained in the same way as previously with the Mercurio & Morini model, thus obtaining:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sum_{i,j=1}^{N-1} \theta_{ij}\sigma_i V_i \sigma_j V_j \frac{\partial^2 u}{\partial V_i \partial V_j} + \frac{1}{2}\sum_{i,j=1}^{N-1} \rho_{ij} V_i F_i^{\beta_i} V_j F_j^{\beta_j} \frac{\partial^2 u}{\partial F_i \partial F_j} + \frac{1}{2}\sum_{i,j=1}^{N-1} \phi_{ij} V_i F_i^{\beta_i} \sigma_j V_j \frac{\partial^2 u}{\partial F_i \partial V_j} + \sum_{i=1}^{N-1} \mu^{F_i}(t) F_i^{\beta_i} \frac{\partial u}{\partial F_i} + \sum_{i=1}^{N-1} \mu^{V_i}(t) V_i \frac{\partial u}{\partial V_i} = 0. \quad (4.8)$$

Rebonato model is analogous to Hagan one, therefore its PDE will be also quite similar to (4.8).

4.3 Finite Difference Method

Hereafter, as we have motivated in the previous section, we are going to just focus on the PDE (4.6) of the Mercurio & Morini model. This backward parabolic PDE must be supplemented with a terminal condition, which describes the value of the variable u at the final time T. Moreover, appropriate boundary conditions are required, which prescribe how the function u, or its derivatives, behave at the boundaries of the necessarily bounded computational domain.

We are going to define a $(N+1)$-dimensional mesh with the time sampled from today (time 0) to the final expiry of the option (time T) at $M+1$ points uniformly spaced by the time step $\Delta t = \frac{T}{M}$.

The variables representing the forward rates $\mathbf{F} = (F_1, \ldots, F_{N-1})$ and their stochastic volatility V, often referred as the "space variables" will be sampled at $M_i + 1$ ($i = 1, \ldots, N-1$) and $S+1$ points spaced by $h_i = \frac{F_i^{max} - F_i^{min}}{M_i}$ and $h_v = \frac{V^{max} - V^{min}}{S}$, respectively.

Notice that while the choice of the range of the time variable is totally unambiguous, $[0, T]$, an a priori choice must be made about which values of the space variables are too high or too low to be of interest, so far we will denote them by $[F_i^{min}, F_i^{max}]$ and $[V^{min}, V^{max}]$. Selecting boundary values such that the option of interest is too deeply in or out-of-the money is a common and reasonable choice.

For a given mesh, each point is uniquely determined by the time level m ($m = 0, \ldots, M$), the index vector of the $N-1$ forward rates $\mathbf{f} = (f_1, \ldots, f_i, \ldots, f_{N-1})$ ($f_i = 0, \ldots, M_i$) and the stochastic volatility level v ($v = 0, \ldots, S$). We seek approximations of the solution at these mesh points, which will be denoted by

$$U^m_{\mathbf{f},v} \approx u(m\Delta t, (f_i h_i)_{1\le i\le N-1}, v h_v).$$

It is natural for this PDE to be solved backwards in time. We approximate the time derivative by the time-forward approximation

$$\left.\frac{\partial u}{\partial t}\right|_{t=m\Delta t, \mathbf{F}=(f_i h_i)_{1\le i\le N-1}, V=vh_v} = \left.\frac{\partial u}{\partial t}\right|_{m,\mathbf{f},v} \approx \frac{U^{m+1}_{\mathbf{f},v} - U^m_{\mathbf{f},v}}{\Delta t}.$$

For the space derivatives we have chosen second-order approximations. We will write $\mathbf{f}_{i\pm1}$ to mean the forward rates index vector $(f_1, \ldots, f_i \pm 1, \ldots, f_{N-1})$ which corresponds to the forward rates point $(f_1 h_1, \ldots, (f_i \pm 1)h_i, \ldots, f_{N-1}h_{N-1})$.
The first derivatives are approximated by central differences:

$$\left.\frac{\partial u}{\partial F_i}\right|_{m,\mathbf{f},v} \approx \frac{U^m_{\mathbf{f}_{i+1},v} - U^m_{\mathbf{f}_{i-1},v}}{2h_i}.$$

The second derivatives are approximated by:

- $\left.\dfrac{\partial^2 u}{\partial F_i^2}\right|_{m,\mathbf{f},v} \approx \dfrac{U^m_{\mathbf{f}_{i+1},v} - 2U^m_{\mathbf{f}_i,v} + U^m_{\mathbf{f}_{i-1},v}}{h_i^2},$
- $\left.\dfrac{\partial^2 u}{\partial V^2}\right|_{m,\mathbf{f},v} \approx \dfrac{U^m_{\mathbf{f},v+1} - 2U^m_{\mathbf{f},v} + U^m_{\mathbf{f},v-1}}{h_v^2}.$

The cross derivatives terms are approximated by:

- For $i \neq j$, $\left.\dfrac{\partial^2 u}{\partial F_i \partial F_j}\right|_{m,\mathbf{f},v} \approx \dfrac{U^m_{\mathbf{f}_{i+1,j+1},v} + U^m_{\mathbf{f}_{i-1,j-1},v} - U^m_{\mathbf{f}_{i+1,j-1},v} - U^m_{\mathbf{f}_{i-1,j+1},v}}{4h_i h_j},$
- $\left.\dfrac{\partial^2 u}{\partial F_i \partial V}\right|_{m,\mathbf{f},v} \approx \dfrac{U^m_{\mathbf{f}_{i+1},v+1} + U^m_{\mathbf{f}_{i-1},v-1} - U^m_{\mathbf{f}_{i+1},v-1} - U^m_{\mathbf{f}_{i-1},v+1}}{4h_i h_v}.$

The finite difference solution under the so-called θ-scheme is:

$$\frac{U^{m+1}_{\mathbf{f},v} - U^{m}_{\mathbf{f},v}}{\Delta t} + \theta W^{m}_{\mathbf{f},v} + (1-\theta) W^{m+1}_{\mathbf{f},v} = 0,$$

where $\theta \in [0,1]$ and $W^{m}_{\mathbf{f},v}$ is the discretization given by

$$\begin{aligned}
W^{m}_{\mathbf{f},v} =& \frac{1}{2}\sigma^2 V^2 \frac{U^{m}_{\mathbf{f},v+1} - 2U^{m}_{\mathbf{f},v} + U^{m}_{\mathbf{f},v-1}}{h_v^2} + \\
& \frac{1}{2} V^2 \sum_{\substack{i,j=1 \\ i \neq j}}^{N-1} \rho_{ij}\alpha_i\alpha_j F_i^{\beta} F_j^{\beta} \frac{U^{m}_{\mathbf{f}_{i+1,j+1},v} + U^{m}_{\mathbf{f}_{i-1,j-1},v} - U^{m}_{\mathbf{f}_{i+1,j-1},v} - U^{m}_{\mathbf{f}_{i-1,j+1},v}}{4h_i h_j} + \\
& \frac{1}{2} V^2 \sum_{i=1}^{N-1} \alpha_i^2 F_i^{2\beta} \frac{U^{m}_{\mathbf{f}_{i+1},v} - 2U^{m}_{\mathbf{f}_{i},v} + U^{m}_{\mathbf{f}_{i-1},v}}{h_i^2} + \\
& \sigma V^2 \sum_{i=1}^{N-1} \phi_i \alpha_i F_i^{\beta} \frac{U^{m}_{\mathbf{f}_{i+1},v+1} + U^{m}_{\mathbf{f}_{i-1},v-1} - U^{m}_{\mathbf{f}_{i+1},v-1} - U^{m}_{\mathbf{f}_{i-1},v+1}}{4h_i h_v} + \\
& \sum_{i=1}^{N-1} \mu_i(m\Delta t) F_i^{\beta} \frac{U^{m}_{\mathbf{f}_{i+1},v} - U^{m}_{\mathbf{f}_{i-1},v}}{2h_i},
\end{aligned} \tag{4.9}$$

and with terminal condition $U^{M}_{\mathbf{f},v} = g(T, \mathbf{F}(T), V(T))$.

Three different θ values represent three canonical discretization schemes, $\theta = 0$ is the explicit scheme, $\theta = 1$ the fully implicit scheme and $\theta = 0.5$ the Crank-Nicolson scheme. The fully implicit discretization is the best method with respect to stability, whereas the Crank-Nicolson timestepping provides the best convergence rate. Although the explicit method is the simplest to implement, it has the disadvantage of not being unconditionally stable.

We shall first discriminate explicit and implicit parts as follows:

$$\frac{U^{m}_{\mathbf{f},v}}{\Delta t} - \theta W^{m}_{\mathbf{f},v} = \frac{U^{m+1}_{\mathbf{f},v}}{\Delta t} + (1-\theta) W^{m+1}_{\mathbf{f},v}. \tag{4.10}$$

As a result of such discretization we arrive to the linear system of equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is the band matrix of known coefficients, $\mathbf{x}$ is the vector of the unknown solutions $U^{m}_{\mathbf{f},v}$ and $\mathbf{b}$ is the vector of known values corresponding to the right-hand side of (4.10).

Equation (4.10) can be rewritten as:

$$
\begin{aligned}
&d\theta U^m_{\mathbf{f},v-1} + d\theta U^m_{\mathbf{f},v+1} + \sum_{i=1}^{N-1}(b_i - r_i)\theta U^m_{\mathbf{f}_{i-1},v} + \sum_{i=1}^{N-1}(b_i + r_i)\theta U^m_{\mathbf{f}_{i+1},v} + \\
&\sum_{i=1}^{N-1}\left(a_i\theta U^m_{\mathbf{f}_{i-1},v-1} + a_i\theta U^m_{\mathbf{f}_{i+1},v+1} - a_i\theta U^m_{\mathbf{f}_{i-1},v+1} - a_i\theta U^m_{\mathbf{f}_{i+1},v-1}\right) + \\
&\sum_{ij\in C}\left(\psi_{ij}\theta U^m_{\mathbf{f}_{i-1,j-1},v} + \psi_{ij}\theta U^m_{\mathbf{f}_{i+1,j+1},v} - \psi_{ij}\theta U^m_{\mathbf{f}_{i-1,j+1},v} - \psi_{ij}\theta U^m_{\mathbf{f}_{i+1,j-1},v}\right) + \\
&\left(-1 - 2d\theta - 2\theta\sum_{i=1}^{N-1} b_i\right) U^m_{\mathbf{f},v} = \\
&- d\hat{\theta} U^{m+1}_{\mathbf{f},v-1} - d\hat{\theta} U^{m+1}_{\mathbf{f},v+1} - \sum_{i=1}^{N-1}(b_i - r_i)\hat{\theta} U^{m+1}_{\mathbf{f}_{i-1},v} - \sum_{i=1}^{N-1}(b_i + r_i)\hat{\theta} U^{m+1}_{\mathbf{f}_{i+1},v} \\
&- \sum_{i=1}^{N-1}\left(a_i\hat{\theta} U^m_{\mathbf{f}_{i-1},v-1} + a_i\hat{\theta} U^{m+1}_{\mathbf{f}_{i+1},v+1} - a_i\hat{\theta} U^{m+1}_{\mathbf{f}_{i-1},v+1} - a_i\hat{\theta} U^{m+1}_{\mathbf{f}_{i+1},v-1}\right) \\
&- \sum_{ij\in C}\left(\psi_{ij}\hat{\theta} U^{m+1}_{\mathbf{f}_{i-1,j-1},v} + \psi_{ij}\hat{\theta} U^{m+1}_{\mathbf{f}_{i+1,j+1},v} - \psi_{ij}\hat{\theta} U^{m+1}_{\mathbf{f}_{i-1,j+1},v} - \psi_{ij}\hat{\theta} U^{m+1}_{\mathbf{f}_{i+1,j-1},v}\right) + \\
&\left(-1 + 2d\hat{\theta} + 2\hat{\theta}\sum_{i=1}^{N-1} b_i\right) U^{m+1}_{\mathbf{f},v}, \qquad (4.11)
\end{aligned}
$$

where $\hat{\theta} = (1-\theta)$, C is the set containing the combinations of numbers $1, 2, \ldots, N-1$ taken two at a time without repetition (the number of elements in C is $\binom{N-1}{2} = 2^{-1}(N-1)(N-2)$) and the known coefficients d, b_i, r_i, a_i and ψ_{ij} are defined as

$$
\begin{aligned}
d &= \frac{\Delta t \sigma^2 V^2}{2h_v^2}, \quad b_i = \frac{\Delta t V^2 \alpha_i^2 F_i^{2\beta}}{2h_i^2}, \\
r_i &= \frac{\Delta t \mu_i(t) F_i^{\beta}}{2h_i}, \quad a_i = \frac{\Delta t \sigma V^2 \phi_i \alpha_i F_i^{\beta}}{4h_i h_v}, \\
\psi_{ij} &= \frac{\Delta t V^2 \rho_{ij} \alpha_i \alpha_j F_i^{\beta} F_j^{\beta}}{4h_i h_j},
\end{aligned}
$$

where we have denoted $\mathbf{F} = (F_i = f_i h_i)_{1\le i\le N-1}$ and $V = v h_v$.

4.3.1 Boundary conditions

In order to specify boundary conditions, a combination of mathematical, financial and heuristic reasoning allows us to find consistent and acceptable ones. There are several possibilities, see [36] for example.

In principle forward rates and their stochastic volatility are non negative and hence take values in the range zero to infinity. We first truncate the unbounded interval to a bounded one and then we must specify conditions at the new boundary. Thus we will consider the truncated domain $[F_i^{min}, F_i^{max}] \times [V^{min}, V^{max}]$, with $F_i^{min} = 0$ and $V^{min} = 0$.

For the forward rates we have considered Dirichlet boundary conditions. Particularly, the terminal condition will hold on the forward rates boundaries, i.e.

$$U^m_{\{\mathbf{f}|\exists f_i=0\},v} = U^M_{\mathbf{f},v}, \quad \forall m = 0, \ldots, M-1,$$

$$U^m_{\{\mathbf{f}|\exists f_i=M_i\},v} = U^M_{\mathbf{f},v}, \quad \forall m = 0, \ldots, M-1.$$

For the stochastic volatility we have considered the following boundary conditions:

$$\frac{\partial u}{\partial t} + \sum_{i=1}^{N-1} \mu_i(t) F_i^\beta \frac{\partial u}{\partial F_i} = 0, \quad V = 0, \tag{4.12}$$

$$\frac{\partial u}{\partial V} = 0, \quad V = V_{max}. \tag{4.13}$$

When $V = 0$ we require that the PDE itself must be satisfied on this boundary, this is known as a smoothing condition. When V approaches to infinity, the price of the derivative becomes independent of V. This is reflected by using Neumann conditions instead of the Dirichlet ones used for the forward rates boundaries.

At the boundary $V = 0$, after discretizing the boundary condition (4.12) we obtain (note that the coefficients d, b_i, a_i and ψ_{ij} of equation (4.11) are zero):

$$-\sum_{i=1}^{N-1} r_i \theta U^m_{\mathbf{f}_{i-1},0} + \sum_{i=1}^{N-1} r_i \theta U^m_{\mathbf{f}_{i+1},0} - U^m_{\mathbf{f},0} = \sum_{i=1}^{N-1} r_i \hat{\theta} U^{m+1}_{\mathbf{f}_{i-1},0} - \sum_{i=1}^{N-1} r_i \hat{\theta} U^{m+1}_{\mathbf{f}_{i+1},0} + U^{m+1}_{\mathbf{f},0}.$$

For the boundary $V = V_{max}$ in order to maintain the second order accuracy in the discretization of the first derivative the ghost point method is considered. The

ghost grid points $U_{\mathbf{f},S+1}$ are added. Then, the finite difference scheme of equation (4.11) can also be applied at the points $U_{\mathbf{f},S}$. However, we now have more unknowns than equations. The additional equations come from the central finite difference discretization of the Neumann boundary condition (4.13):

$$\frac{U_{\mathbf{f},S+1} - U_{\mathbf{f},S-1}}{2h_v} = 0,$$

which yields $U_{\mathbf{f},S+1} = U_{\mathbf{f},S-1}$. Inserting this into the finite difference equation at $V = V_{max}$ we achieve

$$\begin{aligned}
&\hat{d}\theta U^m_{\mathbf{f},S-1} + \sum_{i=1}^{N-1}(b_i - r_i)\theta U^m_{\mathbf{f}_{i-1},S} + \sum_{i=1}^{N-1}(b_i + r_i)\theta U^m_{\mathbf{f}_{i+1},S} + \\
&\sum_{ij\in C}\left(\psi_{ij}\theta U^m_{\mathbf{f}_{i-1,j-1},S} + \psi_{ij}\theta U^m_{\mathbf{f}_{i+1,j+1},S} - \psi_{ij}\theta U^m_{\mathbf{f}_{i-1,j+1},S} - \psi_{ij}\theta U^m_{\mathbf{f}_{i+1,j-1},S}\right) + \\
&\left(-1 - \hat{d}\theta - 2\theta\sum_{i=1}^{N-1} b_i\right) U^m_{\mathbf{f},S} = \\
&- \hat{d}\hat{\theta} U^{m+1}_{\mathbf{f},S-1} - \sum_{i=1}^{N-1}(b_i - r_i)\hat{\theta} U^{m+1}_{\mathbf{f}_{i-1},S} - \sum_{i=1}^{N-1}(b_i + r_i)\hat{\theta} U^{m+1}_{\mathbf{f}_{i+1},S} + \\
&- \sum_{ij\in C}\left(\psi_{ij}\hat{\theta} U^{m+1}_{\mathbf{f}_{i-1,j-1},S} + \psi_{ij}\hat{\theta} U^{m+1}_{\mathbf{f}_{i+1,j+1},S} - \psi_{ij}\hat{\theta} U^{m+1}_{\mathbf{f}_{i-1,j+1},S} - \psi_{ij}\hat{\theta} U^{m+1}_{\mathbf{f}_{i+1,j-1},S}\right) + \\
&\left(-1 + \hat{d}\hat{\theta} + 2\hat{\theta}\sum_{i=1}^{N-1} b_i\right) U^{m+1}_{\mathbf{f},S},
\end{aligned}$$

where $\hat{d} = 2d = \dfrac{\Delta t \sigma^2 V_{max}^2}{h_v^2}$.

4.3.2 Numerical results

It is not clear where to place F_i^{max} and V^{max}. On one hand, it is advantageous to place them far away of the initial forward rates. This reduces the error of the artificial boundary conditions. On the other hand a large computational domain requires a large discretization width. This increases the error of the approximation of the derivatives. In our experiments we will consider $F_i^{max} = 0.1$ and $V^{max} = 2.0$.

We are going to value $T_\alpha \times (T_\beta - T_\alpha)$ European swaptions, meaning that the swaption has maturity at time T_α and the length of the underlying swap is $(T_\beta - T_\alpha)$ (also known as the tenor of the swaption).

The pricing model is described in Table 4.1 and the employed market data are given in Table 4.2. We will consider $\lambda = 0.1$ in the model for the correlation structure (4.2). Besides, the Crank-Nicolson scheme will be used in (4.10).

For solving the system (4.11) the Gauss-Seidel iterative solver has been employed using a tolerance of 10^{-6}.

The numerical experiments have been performed with the following hardware and software configurations: two recent multicore Intel Xeon CPUs E5-2620 v2 clocked at 2.10 GHz (6 cores per socket) with 62 GBytes of RAM, CentOS Linux, GNU C++ compiler 4.8.2.

Currency	EUR
Index	EURIBOR
Day Count	e30/360
Strike	5.5%

Table 4.1: Specification of the interest rate model.

First of all, the results from pricing a 1×1 European swaption are discussed. The value ϑ of this swaption is the same as the price of the corresponding caplet, and so depends only on F_1. Hence, in one dimension a closed form expression for the price of a European swaption can be found by using Black's formula:

$$\vartheta = P(T_0, T_2)\tau_1 \mathrm{Bl}(K, F(T_1, T_2; T_0), \nu_1),$$

where

$$\mathrm{Bl}(K, F, \nu) = F\Phi\big(d_1(K, F, \nu)\big) - K\Phi\big(d_2(K, F, \nu)\big),$$
$$d_1(K, F, \nu) = \frac{\ln(F/K) + \nu^2/2}{\nu},$$
$$d_2(K, F, \nu) = \frac{\ln(F/K) - \nu^2/2}{\nu},$$

	Start date	End date	LIBOR Rate (%)	Volatility (%)
T_0	29-07-04	29-07-05	2.423306	0
T_1	29-07-05	29-07-06	3.281384	24.73
T_2	29-07-06	29-07-07	3.931690	22.45
T_3	29-07-07	29-07-08	4.364818	19.36
T_4	29-07-08	29-07-09	4.680236	17.43
T_5	29-07-09	29-07-10	4.933085	16.15
T_6	29-07-10	29-07-11	5.135066	15.02
T_7	29-07-11	29-07-12	5.273314	14.24
T_8	29-07-12	29-07-13	5.376115	13.42

Table 4.2: Market data used in pricing. Data taken from 27th July 2004.

$$\nu_i = \sigma_{Black}\sqrt{T_i},$$

where $P(T_0, T_2)$ is the price at time T_0 of a bond with maturity T_2 and σ_{Black} is the constant volatility of the forward rate. This value is equal to 0.659096 basis points (one basis point is one hundredth of one percent, $\frac{1\%}{100} = \frac{1}{10000}$). As Black-Scholes formula for caplets considers constant volatility σ_{Black}, in this first test the volatility of the volatility parameter of Mercurio & Morini model is considered equal to zero, i.e., $\sigma = 0$, therefore a standard LIBOR market model is used. The solution was found on several levels and Table 4.3 shows the convergence of the model. In all tables of this chapter, *Level* refers to the refinement level n, i.e., the mesh size is $h_i = 2^{-n} \cdot c_i$ in each coordinate direction, where c_i denotes the computational domain length in direction i, which is F_i^{max} in the case of the forward rates and V^{max} in the case of the stochastic volatility. Besides, the solution and the error with respect to the exact solution are also shown in basis points. Additionally, the execution time is measured in seconds and the column labeled as *Grid points* shows the number of grid points employed in the full grid used by the finite difference method without taking into account the time coordinate.

When the volatility of the volatility σ of the model is non zero or when the length of the underlying swap of the swaption being considered is greater than one, no closed

form solutions are available. However, an estimate can be obtained from Monte Carlo simulations. On Table 4.4 Monte Carlo values for the 1×1 European swaption with $\sigma = 0$ are shown for several numbers of paths ($\#Paths$). More details about Monte Carlo simulation of SABR/LIBOR market models can be found in Chapter 3.

Level	Solution	Error	Time	Grid points
3	2.078086	1.418989	0.0024	81
4	1.108211	0.449114	0.0094	289
5	0.779033	0.119936	0.07	1089
6	0.672004	0.012907	0.53	4225
7	0.665176	0.006079	6.34	16641
8	0.661164	0.002067	84.12	66049
9	0.659380	0.000283	1122.86	263169
10	0.659032	0.000064	14288.34	1050625

Table 4.3: Convergence of the PDE solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0$, $V(0) = 1$, $\beta = 1$, 128 time steps. Exact solution, 0.659096 basis points.

#Paths	Solution
10^5	0.616799
10^7	0.658598
10^9	0.659506

Table 4.4: Convergence of Monte Carlo solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0$, $V(0) = 1$, $\beta = 1$, 128 time steps. Exact solution, 0.659096 basis points.

In Table 4.5 the pricing of the 1×1 European swaption with $\sigma = 0.3$ for different resolution levels n are shown.

In Table 4.6 the results for the 1×2 swaption are given. Note that with this numerical method it was not feasible to price the swaption past refinement level $n = 7$ due to the huge number of required grid points.

In Table 4.7 the results for the 1×3 swaption are given. Full grid pricing is only possible on low grid levels. It is not achievable to obtain a solution for a level greater

Level	Solution	Time	Grid points
3	6.254822	0.0039	81
4	2.501988	0.0122	289
5	1.991646	0.07	1089
6	1.597470	0.62	4225
7	1.526047	7.48	16641
8	1.519841	98.45	66049
9	1.519742	1291.76	263169
10	1.519732	16238.98	1050625

Table 4.5: Convergence of the PDE solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 1.657662 basis points.

Level	Solution	Time	Grid points
3	7.036058	0.05	729
4	6.644769	0.54	4913
5	5.500283	8.59	35937
6	4.943895	182.56	274625
7	4.909506	4689.62	2146689

Table 4.6: Convergence of the PDE solution in basis points for 2 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 4.652644 basis points.

than 6 in reasonable computational time due to the high number of grid points.

Theoretically, it is possible to solve the discrete system (4.11) for a general number of dimensions. However, in computational science, a major problem occurs when the number of dimensions increases. A natural way to diminish the discretization error is to decrease the mesh width in each coordinate direction. However, then the number of grid points in the resulting full grid grows exponentially with the dimension, i.e. the size of the discrete solution increases drastically. This is called the *curse of dimensionality* [7]. Therefore, this procedure of improving the accuracy by decreasing the mesh width is mainly bounded by two factors, the storage and the computational complexity. Due to these limitations, using a full grid discretization method which

Level	Solution	Time	Grid points
3	6.070182	0.78	6561
4	6.149721	22.03	83521
5	6.513167	752.60	1185921
6	6.721081	34081.58	17850625

Table 4.7: Convergence of the PDE solution in basis points for 3 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 8.177764 basis points.

achieves sufficiently accurate approximations is only possible for problems with up to three or four dimensions, even on the most powerful machines presently available [21]. Two approaches to try to overcome the curse of dimensionality are increasing the order of accuracy of the applied numerical approximation scheme or reducing the dimension of the problem by choosing suitable coordinates. Both approaches are not always possible for every option pricing problem. In this chapter we will take advantage of the sparse grid combination technique first introduced by Zenger and co-workers [63] in order to try to overcome the curse of dimensionality. The combination technique replicates the structure of a so-called sparse grid by linearly combining solutions on coarser grids of the same dimensionality. This technique will reduce the computational effort and the storage space involved with the mentioned traditional finite difference discretization methods. The number of sub-problems to solve will increase, while the computational time per problem decreases drastically. This method can be implemented in parallel as each sub-grid is independent of the others. In the next section we present this technique.

4.4 Sparse grids and the combination technique

The sparse grid method was originally developed by Smolyak [128], who used it for numerical integration. It is based on a hierarchical basis [141, 142], a representation of a discrete function space which is equivalent to the conventional nodal basis, and

a sparse tensor product construction. Zenger [144] and Bungartz and Griebel [21] extended this idea and applied sparse grids to solve partial differential equations with finite elements, finite volumes and finite differences methods. Besides working directly in the hierarchical basis the sparse grid can also be computed using the combination technique [63] by linearly combining solutions on traditional Cartesian grids with different mesh widths. This is the approach we have followed in this chapter. In the next two subsections we give a brief introduction to sparse grids and the combination technique. For a detailed discussion we refer to [21].

4.4.1 Sparse grids

First of all we introduce some notation and definitions. Let $\mathbf{l} = (l_1, l_2, \ldots, l_d) \in \mathbb{N}_0^d$ denote a d-dimensional multi-index. Let $|\mathbf{l}|_1$ and $|\mathbf{l}|_\infty$ denote the discrete L_1-norm and $L_\infty-$norm of the multi-index $\mathbf{l}$, respectively, that are defined as

$$|\mathbf{l}|_1 = \sum_{k=1}^{d} l_k \quad \text{and} \quad |\mathbf{l}|_\infty = \max_{1 \le k \le d} l_k.$$

We define the anisotropic grid $\Omega_{\mathbf{l}}$ with mesh size $\mathbf{h} = (h_1, h_2, \ldots, h_d) = (2^{-l_1}c_1, 2^{-l_2}c_2, \ldots, 2^{-l_d}c_d)$ with multi-index $\mathbf{l}$ and grid length $\mathbf{c} = (c_1, c_2, \ldots, c_d)$.

Then, the full grid at refinement level $n \in \mathbb{N}$ and mesh size $h_i = 2^{-n} \cdot c_i$ for all i can be defined via the sequence of subgrids

$$\Omega^n = \Omega_{(n,\ldots,n)} = \bigcup_{|\mathbf{l}|_\infty \le n} \Omega_{\mathbf{l}}.$$

Figure 4.1 visualizes two dimensional full grids for levels $n = 0, \ldots, 4$. The number of grid points in each coordinate direction of the full grid is $2^n + 1$ and therefore the number of grid nodes in the full grid increases with $O(2^{n \cdot d})$, i.e. grows exponentially with the dimensionality d of the problem.

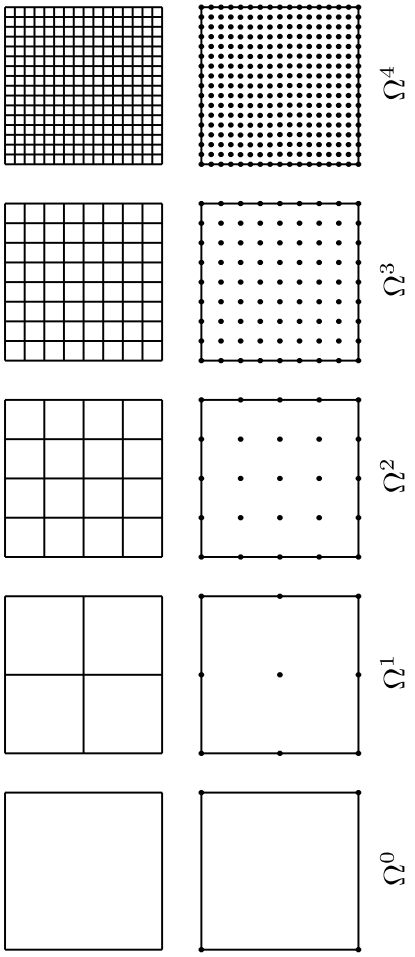


Figure 4.1: Two-dimensional full grid hierarchy up to level $n = 4$.

The sparse grid Ω_s^n at refinement level n consists of all anisotropic Cartesian grids $\Omega_{\mathbf{l}}$, where the total sum of all refinement factors l_k in each coordinate direction equals the resolution n. Then, the sparse grid Ω_s^n is given by

$$\Omega_s^n = \bigcup_{|\mathbf{l}|_1 \leq n} \Omega_{\mathbf{l}} = \bigcup_{|\mathbf{l}|_1 = n} \Omega_{\mathbf{l}}.$$

Figure 4.2 shows the two-dimensional grid hierarchy for levels $n = 0, \ldots, 4$.

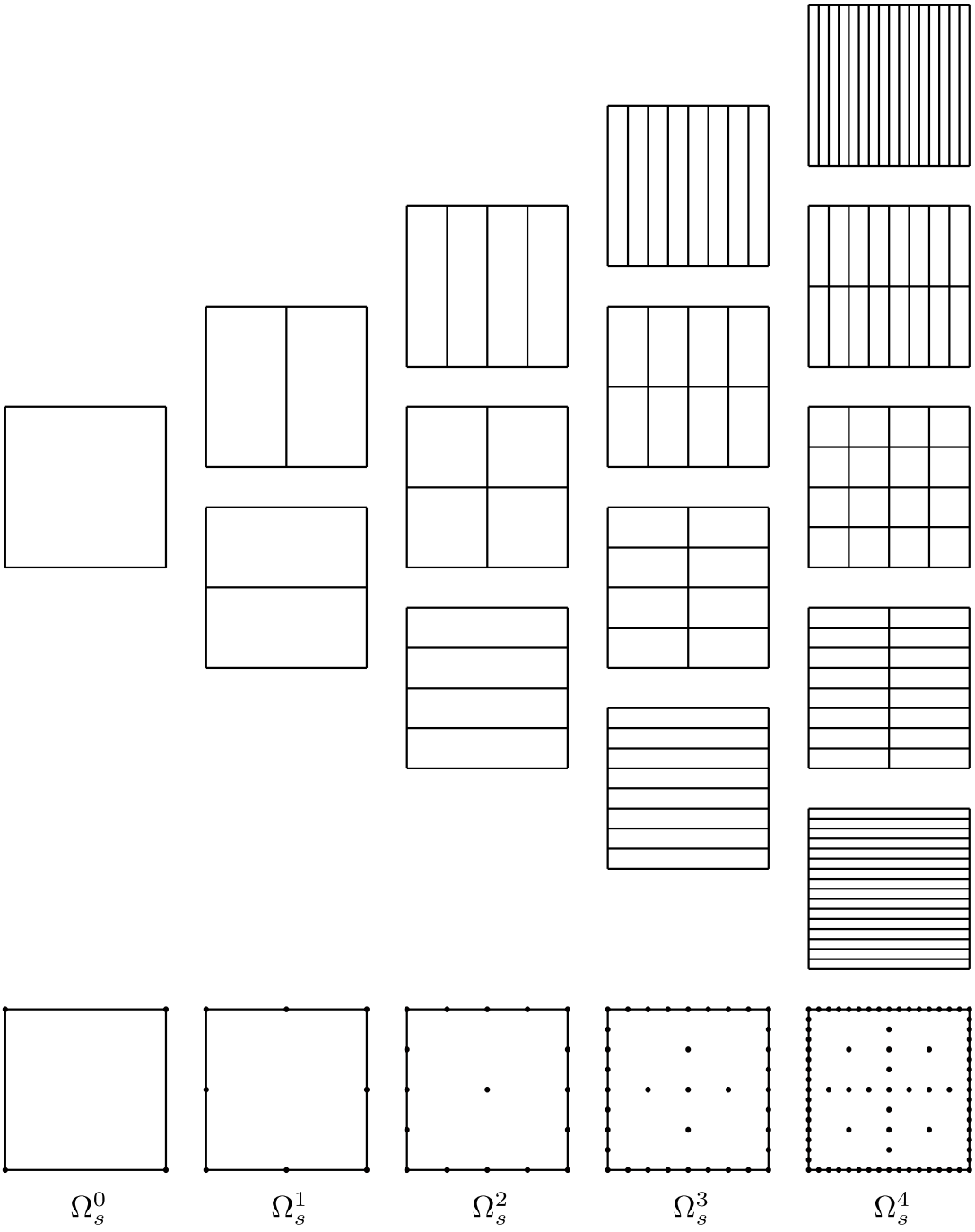


Figure 4.2: Two-dimensional sparse grid hierarchy up to level $n = 4$.

The total number of nodes in the grid $\Omega_{\mathbf{l}}$ is $\prod_{k=1}^{d}(2^{l_k}+1) = O(2^{|\mathbf{l}|_1}) = O(2^n)$. In addition, there exist exactly $\binom{n+d-1}{d-1}$ grids $\Omega_{\mathbf{l}}$ with $|\mathbf{l}|_1 = n$,

$$\begin{aligned}\binom{n+d-1}{d-1} &= \frac{(n+d-1)!}{(d-1)!n!} = \frac{(n+d-1)\cdot\ldots\cdot(n+1)n!}{(d-1)!n!} \\ &= \frac{n+(d-1)}{d-1}\cdot\frac{n+(d-2)}{d-2}\cdot\ldots\cdot\frac{n+(d-(d-1))}{d-(d-1)} \\ &= \left(1+\frac{n}{d-1}\right)\cdot\left(1+\frac{n}{d-2}\right)\cdot\ldots\cdot\left(1+\frac{n}{2}\right)\cdot\left(1+\frac{n}{1}\right) \\ &\leq (1+n)^{d-1} = O(n^{d-1}).\end{aligned}$$

Thus, the total number of grid points of the sparse grid Ω_s^n grows according to

$$\binom{n+d-1}{d-1}\cdot\prod_{k=1}^{d}(2^{l_k}+1) = O(n^{d-1})O(2^n) = O(n^{d-1}2^n), \tag{4.14}$$

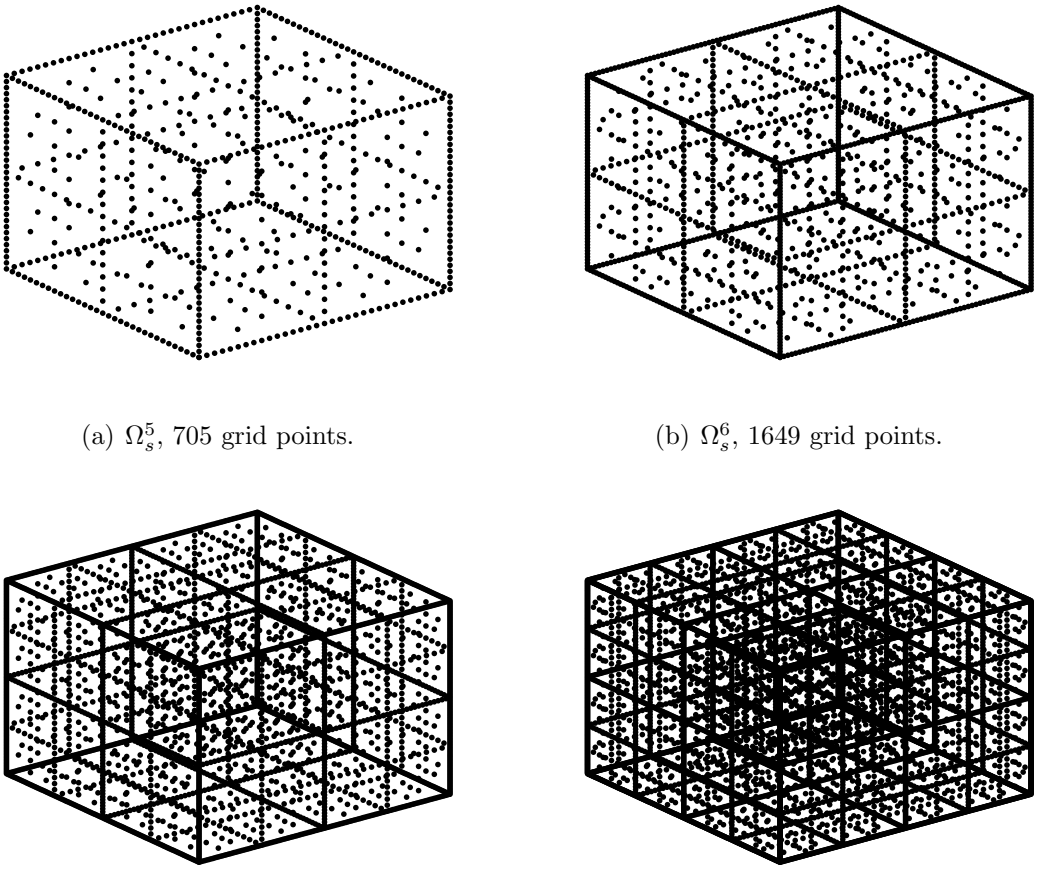
which is far less the size of the corresponding full grid with $O(2^{nd})$ grid points. Let $h_n = 2^{-n}$, therefore the sparse grid employs $O(h_n^{-1}\cdot\log_2(h_n^{-1})^{d-1})$ grid points compared to $O(h_n^{-d})$ nodes in the full grid.

Bungartz and Griebel [21] show that the accuracy of the sparse grid using $O(h_n^{-1}\cdot\log_2(h_n^{-1})^{d-1})$ nodes is of order $O(h_n^2\log_2(h_n^{-1})^{d-1}))$ in the case of finite elements discretization and under certain smoothness conditions. Thus, the accuracy of the sparse grid is only slightly deteriorated from the accuracy $O(h_n^2)$ of conventional full grid methods which need $O(h_n^{-d})$ grid points. Therefore, to achieve a similar approximation quality sparse grids need much less points than regular full grids.

However, the structure of a sparse grid is more complicated than of a full grid. Common partial differential equation solvers usually only manage full grid solutions. Existing sparse grid methods working directly in the hierarchical basis involve a challenging implementation [1, 143]. This handicap can be circumvented with the help of the sparse grid combination technique which not only exploits the economical structure of the sparse grids but also allows for the use of traditional full grid PDE solvers.

Finally, three and two dimensional sparse grids for several resolution levels n are

shown in Figures 4.3 and 4.4, respectively. Additionally, the growth of the grid points when increasing n can be observed.



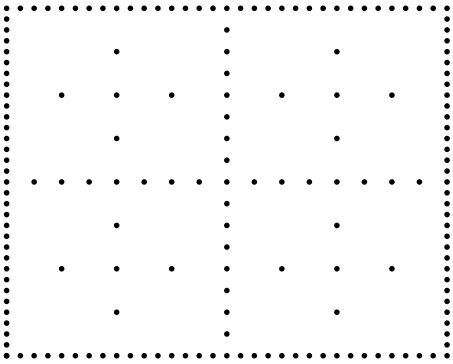
(a) Ω_s^5, 705 grid points.

(b) Ω_s^6, 1649 grid points.

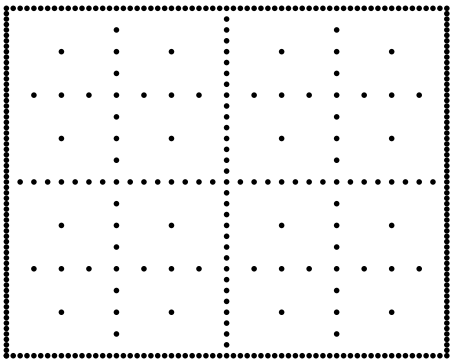
(c) Ω_s^7, 3809 grid points.

(d) Ω_s^8, 8705 grid points.

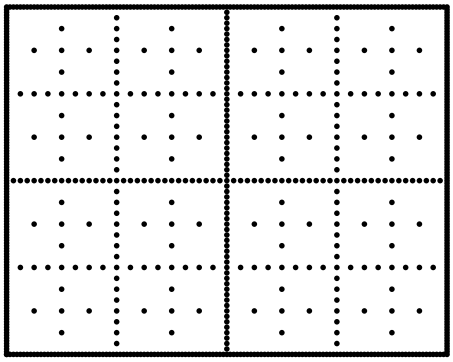
Figure 4.3: Three dimensional sparse grids for levels $n = 5, 6, 7$ and 8.



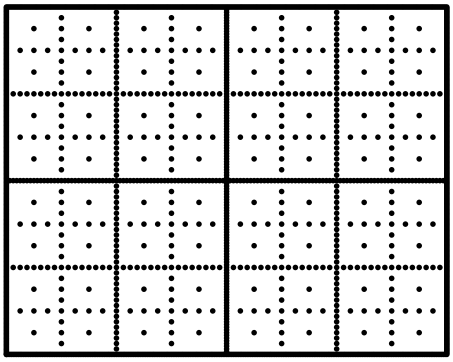
(a) Ω_s^5, 177 grid points.



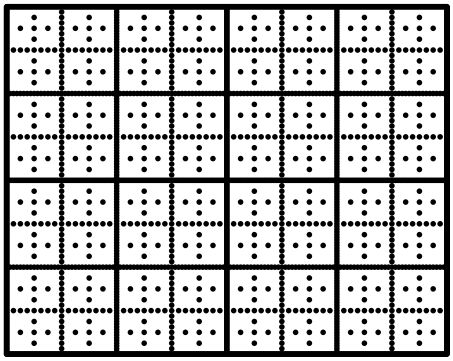
(b) Ω_s^6, 385 grid points.



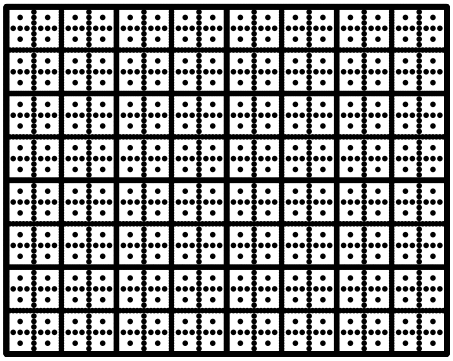
(c) Ω_s^7, 833 grid points.



(d) Ω_s^8, 1793 grid points.



(e) Ω_s^9, 3841 grid points.



(f) Ω_s^{10}, 8193 grid points.

Figure 4.4: Two dimensional sparse grids for levels $n = 5, \ldots, 10$.

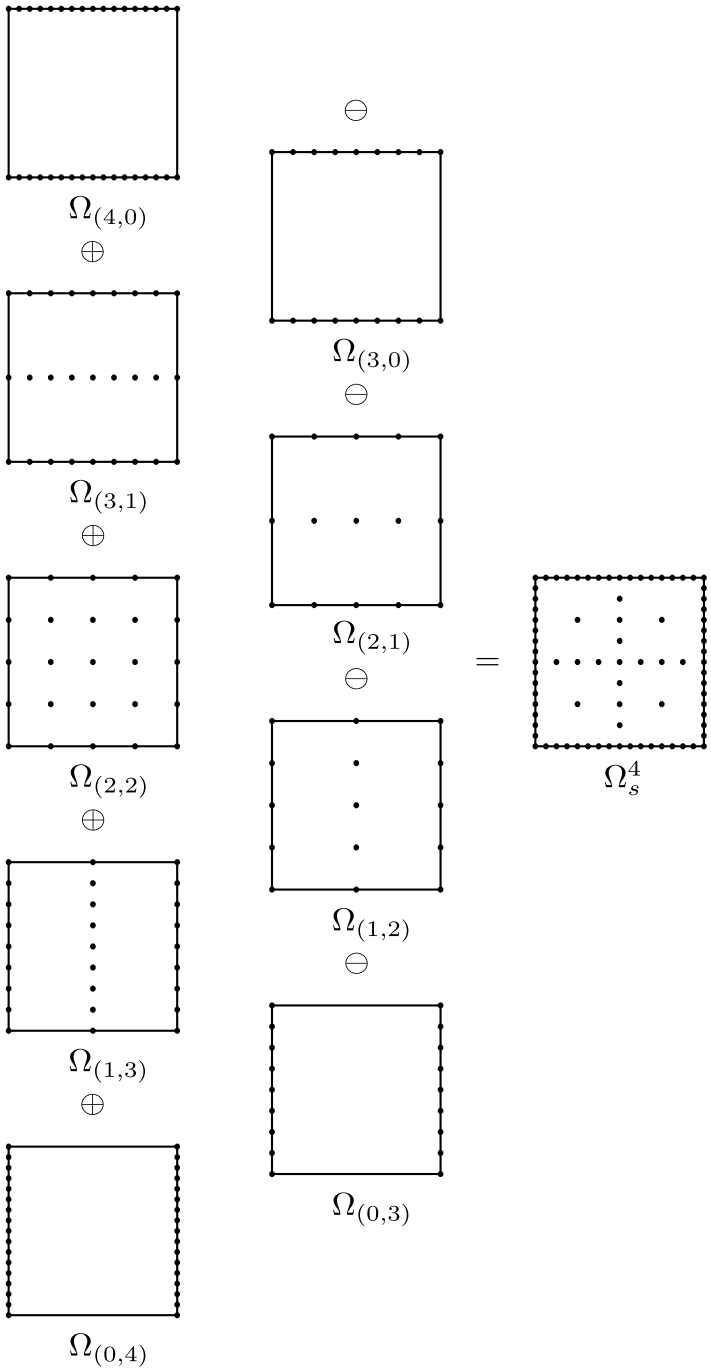


Figure 4.5: Combination technique with level $n = 4$ in two dimensions.

4.4.2 Combination technique

Similar to the Richarson extrapolation, the so-called combination technique linearly combines the numerical solution on the sequence of anisotropic grids $\Omega_{\mathbf{l}}$ where

$$|\mathbf{l}|_1 = n - q, \quad q = 0, \ldots, d-1.$$

The combination technique reads

$$U_s^n = \sum_{q=0}^{d-1} (-1)^q \cdot \binom{d-1}{q} \cdot \sum_{|\mathbf{l}|_1 = n-q} U_{\mathbf{l}}, \quad l_k \geq 0, \quad \forall k = 1, \ldots, d, \tag{4.15}$$

where $U_{\mathbf{l}}$ denotes the numerical solution on the grid $\Omega_{\mathbf{l}}$ and U_s^n the combined solution on the sparse grid Ω_s^n.

The grids employed by the combination technique of level $n = 4$ in two dimensions are shown in Figure 4.5.

The idea of this technique is that the leading order errors from the dicretization on each grid cancel each other out in the combination solution.

The number of grid points involved in the approximation of U_s^n grows according to $O(n^{d-1} \cdot 2^n)$. In fact, from the formula (4.14) we have to solve $\binom{n+d-1}{d-1}$ problems with $O(2^n)$ unknowns, $\binom{n+d-2}{d-1}$ problems with $O(2^{n-1})$ unknowns, ... and $\binom{n}{d-1}$ problems with $O(2^{n-(d-1)})$ unknowns. This results in a total number of $O(n^{d-1} \cdot 2^n)$ grid points which is much less than the $O(2^{n \cdot d})$ grid nodes used by traditional full grid methods. Thus, the efficient use of sparse grids greatly reduces the computing time and the storage requirements which allows for the treatment of problems with ten variables and even more [21].

We have seen that the combination technique linearly combines the numerical solution on several traditional full grids. The solution can be calculated on each of these grids by using any existing PDE numerical method like finite difference, finite volume or finite elements. In addition, since all these sub-problems are independent the combination technique can be parallelized [62].

The combination technique approach presumes the existence of a so-called error splitting. It requires for an associated numerical approximation method on the full

grid $\Omega_{\mathbf{l}}$ an error splitting of the form

$$u(\mathbf{x}) - U_{\mathbf{l}}(\mathbf{x}) = \sum_{k=1}^{d} \sum_{\substack{\{j_1,\ldots,j_k\} \\ \subseteq\{1,\ldots,d\}}} C_{j_1,\ldots,j_k}(\mathbf{x}, h_{j_1}, \ldots, h_{j_k}) \cdot h_{j_1}^p \cdot \ldots \cdot h_{j_k}^p, \tag{4.16}$$

at each grid point $\mathbf{x} \in \Omega_{\mathbf{l}}$. Here u denotes the exact solution of the partial differential equation under consideration, $U_{\mathbf{l}}$ the numerical solution on the grid $\Omega_{\mathbf{l}}$, $p > 0$ is the order of accuracy of the numerical approximation method with respect to each coordinate direction and the coefficient functions $C_{j_1,\ldots,j_k}$ of $\mathbf{x}$ and the mesh sizes h_{j_k}, $k = 1, \ldots, d$ are required to be bounded by a positive constant K such that

$$|C_{j_1,\ldots,j_k}(\mathbf{x}, h_{j_1}, \ldots, h_{j_k})| \leq K, \quad \forall k, 1 \leq k \leq d, \quad \forall \{j_1, \ldots, j_m\} \subseteq \{1, \ldots, d\}.$$

In [64] Griebel and Thurner showed that if the solution of the PDE is sufficiently smooth, the pointwise accuracy of the sparse grid combination technique is $O(n^{d-1} \cdot 2^{-n \cdot p}) = O([\log_2 h_n^{-1}]^{d-1} h_n^p)$, which is only slightly worse than $O(2^{-n \cdot p}) = O(h_n^p)$ obtained by the full grid solution.

The solution at points which does not belong to the sparse grid can be computed through interpolation. The applied interpolation method should provide at least the same order of accuracy of the numerical discretization scheme used to solve the PDE. Otherwise, the accuracy of the numerical scheme will be deteriorated.

Up to now we have presupossed the existence of an error splitting of type (4.16). However, such an error splitting has to be shown for every model problem and depends on the specific discretization used. In fact, proving the existence of this error splitting is usually very complex. Bungartz et al. [22, 23] showed the existence of such an error splitting for the finite difference discretization of the 2-d Laplace equation with sufficiently smooth boundary conditions with the help of Fourier series. Arciniega and Allen [6] proved the existence of this error splitting for the fully implicit as well as the Crank-Nicolson discretization scheme of the European call option. More recently, Reisinger [123] showed that such a splitting also holds for a wider class of linear PDEs, for example convection-diffusion equations. The author gives general conditions which need to be fulfilled to ensure the existence of the desired splitting

structure: sufficiently smooth initial data and compatible boundary data, a consistent numerical scheme which provides a truncation error of the desired splitting structure and stability of the discretization scheme. As a summary, we can say that the deduction of the error splitting formula is very complex and was until now only performed for some reference problems. However, we will see in the following Section 4.4.3 that the numerical results for the sparse grid combination technique seem promising, even for more complex payoff functions.

4.4.3 Numerical results

Taking advantage of the discussed sparse grid combination technique, in this section we are pricing the same interest rate derivatives that have been valued in the former Section 4.3.2 where traditional full grid finite difference methods were considered. In addition to those products, we are going to price interest rate derivatives with up to eight underlying LIBORs and their stochastic volatility, showing that the sparse grid combination technique is able to cope with the curse of dimensionality up to a certain extent. As in the previous Section 4.3.2, we will use Crank-Nicolson scheme, we will consider the Gauss-Seidel iterative solver and the same boundary conditions as in Section 4.3.1. In the present case, we are interested in the evaluation of the solution at a single point which corresponds with the value of the forward rates at time zero (see Table 4.2) and $V(0) = 1$, the numerical solution on each grid handled by the combination technique is interpolated at this point using multilinear interpolation and then added up with the appropriate weights.

The sparse grid combination technique has been implemented to run on multicore CPUs. The program was optimized and parallelized using OpenMP [149]. CPU times, measured in seconds, correspond to executions using 24 threads, so as to take advantage of Intel Hyperthreading. The speedups of the parallel version with respect to the pure sequential code are around 16. To the best of our knowledge, GPUs are not well-suited to parallelize the combination technique, due to the fact that the different grids employed by the combination technique involve memory accesses

patterns totally different, therefore, it is not possible to access the device memory in a coalesced way [109], thus GPU global memory can not serve threads in parallel. In this scenario, the GPU code will be ill performing. In the work [46] the authors take advantage of GPUs to parallelize the solver of each full grid considered by the combination technique. However, they do not parallelize the combination technique itself.

In Table 4.8 a 1×1 European swaption is priced. The exact price of this derivative is 0.659096 basis points, as discussed in Section 4.3.2. These results are to be compared with those of Table 4.3, where it can be seen how the computational times and the grid points employed by the sparse grid combination technique have been substantially reduced.

Next, in Table 4.9 a 1×1 European swaption is priced considering stochastic volatility. These results are to be compared with those of Table 4.5.

In the following Tables 4.10 and 4.11, the pricing of 1×2 and 1×3 European swaptions taking into account stochastic volatility is shown, as in Tables 4.6 and 4.7, respectively. For the higher resolution levels, the full grid method became very slow, while the sparse grid combination technique results much faster. Note that the combination technique is able to price successfully the 1×3 European swaption, this was not attainable in Table 4.6.

Finally, in Tables from 4.12 to 4.16, 1×4, ..., 1×8 European swaptions are priced considering stochastic volatility. The pricing of these interest rate derivatives was not viable with the full grid approach of Section 4.3. In order to be able to price derivatives with more than 9 underlyings, the combination technique method should be parallelized to run on a cluster of processors. In the Chapter 13 of the book [48] Philipp Schröder et al. discuss the parallelization of the combination technique using MPI (*Message Passing Interface*) API. In [93] the authors parallelize the sparse grid combination technique taking advantage of a MapReduce framework, algorithms that are inherently fault tolerant.

Level	Solution	Error	Time	Grid points
3	6.715346	6.056250	0.04	37
4	2.182057	1.522961	0.05	81
5	1.097761	0.438665	0.05	177
6	0.782767	0.123671	0.05	385
7	0.663808	0.004712	0.06	833
8	0.657536	0.001560	0.11	1793
9	0.658183	0.000913	0.46	3841
10	0.659363	0.000267	2.32	8193

Table 4.8: Convergence of the PDE solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0$, $V(0) = 1$, $\beta = 1$, 128 time steps. Exact solution, 0.659096 basis points.

Level	Solution	Time
3	6.818116	0.05
4	2.694770	0.05
5	1.919198	0.05
6	1.596501	0.08
7	1.499332	0.12
8	1.505709	0.14
9	1.515855	0.64
10	1.521027	2.83

Table 4.9: Convergence of the PDE solution in basis points for 1 LIBOR and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 1.657662 basis points.

Level	Solution	Time
5	7.169601	0.07
6	5.167631	0.09
7	4.924462	0.13
8	4.543098	0.25
9	4.336457	0.62
10	4.346780	2.37

Table 4.10: Convergence of the PDE solution in basis points for 2 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 4.652644 basis points.

Level	Solution	Time
7	1.407581	0.19
8	10.824603	0.41
9	5.879247	1.15
10	7.390891	3.90
11	9.395706	13.66
12	9.079540	68.85
13	7.998961	402.88
14	7.934538	2780.26

Table 4.11: Convergence of the PDE solution in basis points for 3 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 8.177764 basis points.

Level	Solution	Time
9	6.781234	4.30
10	7.759745	11.16
11	14.756789	33.88
12	16.827907	106.35
13	8.431096	408.81
14	11.928714	1946.60

Table 4.12: Convergence of the PDE solution in basis points for 4 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 12.288113 basis points.

Level	Solution	Time
11	21.324254	158.15
12	9.715573	433.61
13	18.790682	1227.26
14	18.342280	3786.28

Table 4.13: Convergence of the PDE solution in basis points for 5 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 128 time steps. Monte Carlo value using 10^7 paths, 16.903377 basis points.

Level	Solution	Time
13	17.325454	143.20
14	10.310617	506.85
15	22.378643	2611.43

Table 4.14: Convergence of the PDE solution in basis points for 6 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 2 time steps. Monte Carlo value using 10^7 paths, 21.979879 basis points.

Level	Solution	Time
15	17.931891	5939.81
16	13.890495	19719.87
17	27.590595	87733.59

Table 4.15: Convergence of the PDE solution in basis points for 8 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 2 time steps. Monte Carlo value using 10^7 paths, 27.222777 basis points.

Level	Solution	Time
17	50.609493	228319.90
18	27.228950	774389.53

Table 4.16: Convergence of the PDE solution in basis points for 8 LIBORs and stochastic volatility, $\sigma = 0.3$, $\phi_i = 0.4$, $V(0) = 1$, $\beta = 1$, 2 time steps. Monte Carlo value using 10^7 paths, 32.553432 basis points.

Part II

BSDEs

Introduction to BSDEs

Backward Stochastic Differential Equations (BSDEs) form an interesting recent concept in financial mathematics. Their range of applicability has increased, for example by nonlinear pricing formulas derived from Credit Valuation Adjustment (CVA) frameworks [31], dynamic measures of risk, portfolio optimization with trading constraints, stochastic optimal control (Hamilton-Jacobi-Bellman equation) or optimal execution of American options. BSDEs are directly connected to semilinear partial differential equations as the solution to these PDEs can be found by solving the corresponding decoupled forward-backward stochastic differential equation (FBSDE) problem. Recently, several advanced probabilistic numerical methods have been developed for FBSDEs, like advanced Monte Carlo methods [55], integration methods [146] and also Fourier methods [124]. In this second part of the thesis, our goal is to design highly efficient Monte Carlo methods.

Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{0\leq t\leq T}, \mathbb{P})$ be a filtered probability space supporting a q-dimensional Brownian motion $(W_t)_{t\geq 0}$, with $(\mathcal{F}_t)_{0\leq t\leq T}$ the natural filtration of the Brownian motion, and T a fixed time horizon. A general *backward stochastic differential equation* is given by

$$\begin{aligned} -dY_t &= f(t, Y_t, Z_t)dt - Z_t dW_t, \\ Y_T &= \xi, \end{aligned}$$

where the function f is the so-called driver or generator of the process and the terminal condition ξ is a $\mathcal{F}_T$-measurable random variable, f and ξ are called *standard parameters* for the BSDE. The solution of this BSDE is a pair of adapted processes

(Y, Z) satisfying $\int_0^T |Z_t|^2 dt < \infty$,

$$Y_t = \xi + \int_t^T f(s, Y_s, Z_s) ds - \int_t^T Z_s dW_s, \quad 0 \le t \le T.$$

A result from [86] is that, given a pair of standard parameters (f, ξ) there exists a unique solution $(Y, Z) \in \mathbb{H}_T^2(\mathbb{R}) \times \mathbb{H}_T^2(\mathbb{R})$ to this BSDE.

Now we are in position of introducing the so-called decoupled forward-backward stochastic differential equations (FBSDEs). Assume $0 \le t \le T$, $f(t, w, x, y) = f(t, X_t, Y_t, Z_t)$ and $\xi = g(X_T)$ where X is given by the forward SDE (FSDE),

$$dX_t = b(t, X_t)dt + \sigma(t, X_t)dW_t, \quad X_0 = x_0,$$

and Y by the backward SDE (BSDE),

$$dY_t = -f(t, X_t, Y_t, Z_t)dt + Z_t dW_t, \quad Y_T = g(X_T),$$

whose terminal condition is determined by the terminal value of FSDE. This FBSDE can also be presented in its integral form

$$\begin{aligned} X_t &= x_0 + \int_0^t b(s, X_s) ds + \int_0^t \sigma(s, X_s) dW_s, \\ Y_t &= g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s) ds - \int_t^T Z_s dW_s. \end{aligned}$$

Let u be the solution to the related semilinear parabolic PDE

$$\begin{aligned} &\partial_t u(t, x) + \sum_i b_i(t, x) \partial_{x_i} u(t, x) \\ &\qquad + \frac{1}{2} \sum_{i,j} [\sigma\sigma^*]_{i,j}(t, x) \partial^2_{x_i, x_j} u(t, x) + f\big(t, x, u(t, x), \nabla u(t, x)\sigma(t, x)\big) = 0, \\ &u(T, x) = g(x). \end{aligned}$$

By Itô's formula one knows that $Y_t = u(t, X_t)$ and $Z_t = \nabla u(t, X_t)\sigma(t, X_t)$ solves the BSDE. Therefore, solving the semilinear PDE and the corresponding decoupled FBSDE result in the same solution. The PDE can be solved by applying numerical

discretization techniques, while for the FBSDE probabilistic numerical methods are available, for example Monte Carlo methods. In fact, by using Feynman-Kac theorem one achieves $Y_t = u(t,x) = \mathbb{E}\left(g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)ds \mid X_t = x\right)$, which is the starting point for the probabilistic solution of BSDE or related semilinear PDEs.

Chapter 5

Backward Stochastic Differential Equations

5.1 Introduction

The problem The aim of the algorithm in this chapter is to approximate the (Y, Z) components of the solution to the decoupled forward-backward stochastic differential equation (BSDE)

$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)\mathrm{d}s - \int_t^T Z_s \mathrm{d}W_s, \tag{5.1}$$

$$X_t = x + \int_0^t b(s, X_s)\mathrm{d}s + \int_0^t \sigma(s, X_s)\mathrm{d}W_s, \tag{5.2}$$

where W is a $q \geq 1$ dimensional Brownian motion. The algorithm will also approximate the solution u to the related semilinear, parabolic partial differential equation (PDE) of the form

$$\partial_t u(t,x) + \mathcal{A}u(t,x) + f(t,x,u(t,x),\nabla_x u\sigma(t,x)) = 0 \text{ for } t < T \text{ and } u(T,.) = g(.), \tag{5.3}$$

where $\mathcal{A}$ is the infinitesimal generator of X, through the Feynman-Kac relation $(Y_t, Z_t) = (u(t, X_t), (\nabla_x u\sigma)(t, X_t))$. In recent times, there has been an increasing

interest to have algorithms which work efficiently when the dimension d of the space occupied by the process X is large. This interest has been principally driven by the mathematical finance community, where nonlinear valuation rules are becoming increasingly important.

In general, currently available algorithms [54, 9, 15, 92, 18, 59, 60, 58] rarely handle the case of dimension greater than 8. The main constraint is not only due to the computational time, but mainly due to memory consumption requirements by the algorithms. For example, the recent work [60] uses a Regression Monte Carlo approach (a.k.a. Least Squares MC), in which the solutions $(u, \nabla_x u\sigma)$ of the semi-linear PDE are approximated on a $\mathcal{K}$-dimensional basis of functions at each point of a time grid of cardinality N. Popular choices of basis functions are global [92] or local polynomials [60]. In both cases, the approximation error behaves in general like $\mathcal{K}^{-\alpha'/d}$ where α' measures the smoothness of the function of interest and d is the dimension (curse of dimensionality): see [45, Theorem 6.2.6] for global polynomials, see [66, Section 11.2] for local polynomials. Later, we use local approximations in order to allow parallel computing. We restrict to affine polynomials for implementation in GPU. The coefficients of the basis functions are computed at every time point t_i with the aid of $\mathcal{M}$ simulations of a discrete time Markov chain (which approximates X) in the interval $[t_i, T]$. The main memory constraints of this scheme are (a) to store the $\mathcal{K} \times N$ coefficients of the basis functions, and (b) to store the $\mathcal{M} \times N$ simulations used to compute the coefficients. To illustrate the problem of high dimension, in order to ensure the convergence the dimension of the basis is typically $\mathcal{K} = const \times N^{\alpha d}$, for some $\alpha > 0$ (which decreases with the regularity of the solution), so $\mathcal{K}$ increases geometrically with d. Moreover, the error analysis of these algorithms demonstrates that the local statistical error is proportional to $N\mathcal{K}/\mathcal{M}$, so that one must choose $\mathcal{M} = const \times \mathcal{K}N^2$ to ensure a convergence $O(N^{-1})$ of the scheme. This implies that the simulations pose by far the most significant constraint on the memory.

Objectives The purpose of this chapter is to drastically rework the algorithm of [60] to first minimize the exposure to the memory due to the storage of simulations. This

will allow computation in larger dimension d. Secondly, in this way the algorithm can be implemented in parallel on GPU processors to optimize the computational time.

New Regression Monte Carlo paradigm We develop a novel algorithm called the Stratified Regression MDP (SRMDP) algorithm; the name is aimed to distinguish from the related LSMDP algorithm [60]. The key technique is to use *stratified simulation* of the paths of X. In order to estimate the solution at t_i, we first define a set of hypercubes $(\mathcal{H}_k \subset \mathbb{R}^d : 1 \leq k \leq K)$. Then, for each hypercube $\mathcal{H}_k$, we simulate M paths of the process X in the interval $[t_i, T]$ starting from i.i.d. random variables valued in $\mathcal{H}_k$; these random variables are distributed according to the conditional logistic distribution, see $(\mathbf{A}_\nu)$ later. By using only the paths starting in $\mathcal{H}_k$, we approximate the solution to the BSDE restricted to $X_{t_i} \in \mathcal{H}_k$ on linear functions spaces $\mathcal{L}_{Y,k}$ and $\mathcal{L}_{Z,k}$ (both of small dimension), see $(\mathbf{A}_{\text{Strat.}})$ later. [1] This allows us to minimize the amount of memory consumed by the simulations, since we only need to generate samples on one hypercube at a time. In Theorem 5.3.5, we demonstrate that the error of our scheme is proportional to $N \max(\dim(\mathcal{L}_{Y,k}), \dim(\mathcal{L}_{Z,k}))/M$ and, since $\max(\dim(\mathcal{L}_{Y,k}), \dim(\mathcal{L}_{Z,k})) = const$, we require only $M = const \times N^2$ to ensure the convergence $O(N^{-1})$. Therefore, the memory consumption of the algorithm will be dominated by the storage of the coefficients, which equals $const \times N^{\alpha d}$ (the theoretical minimum). Moreover, the computations are performed in parallel across the hypercubes, which allows for massive parallelization. The speedup compared to sequential programming increases as the dimension d increases, because of the geometric growth of the number of hypercubes with respect to d. In the subsequent tests (Section 5.5), for instance we can solve problems in dimension $d = 11$ within eight seconds using 2000 simulations per hypercube.

[1] To distinguish from previous algorithms, we use two notations for the number of simulations in this section: $\mathcal{M}$ and M. $\mathcal{M}$ stands for the overall number of simulations for computing the full approximation in the unstratified algorithms, while M stands for the number of simulations used to evaluate the approximation locally in each stratum (our stratified regression algorithm). Later we will mainly use M.

This regression Monte Carlo approach is very different from the algorithm proposed in [60]. Although local approximations were already proposed in that work, the paths of the process X were simulated from a fixed point at time 0 rather than directly in the hypercubes. This implies that one must store *all* the simulated paths at any given time, rather than only those for the specific hypercubes. This is because the trajectories are random, and one is not certain which paths will end up in which hypercubes a priori. Therefore, our scheme essentially removes the main constraint on the memory consumption of LSMC algorithms for BSDEs.

The choice of the logistic distribution for the stratification procedure is crucial. Firstly, it is easy to simulate from the conditional distribution. Secondly, it possesses the important USES property (see later $(\mathbf{A}_\nu)$), which enables us to recover equivalent $\mathbf{L}_2$-norms (up to constant) for the marginal of the forward process initialized with the logistic distribution (Proposition 5.2.1).

Literature review Parallelization of Monte-Carlo methods for solving non-linear probabilistic equations has been little investigated. Due to the non-linearity, this is a challenging issue. For optimal stopping problems, we can refer to the works [90, 91, 41] with numerical results up to dimension 4. To the best of our knowledge, the only work related to BSDEs in parallel version is [92]. It is based on a Picard iteration for finding the solution, coupled with iterative control variates. The iterative solution is computed through an approximation on sparse polynomial basis. Although the authors report efficient numerical experiments up to dimension 8, this study is not supported by a theoretical error analysis. Due to the stratification, our proposed approach is quite different from [92] and additionally, we provide an error analysis (Theorem 5.3.5).

Most of the results in this chapter are included in the reference [56].

Notation

(i) $|x|$ stands for the Euclidean norm of the vector x.

(ii) $\log(x)$ stands for the natural logarithm of $x \in \mathbb{R}_+$.

(iii) For a multidimensional process $U = (U_i)_{0\leq i\leq N}$, its l-th component is denoted by $U_l = (U_{l,i})_{0\leq i\leq N}$.

(iv) For any finite $L > 0$ and $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, define the truncation function

$$\mathcal{T}_L(x) := (-L \vee x_1 \wedge L, \ldots, -L \vee x_n \wedge L). \tag{5.4}$$

(v) For a probability measure ν on a domain D, and function $h : D \to \mathbb{R}^l$ in $\mathbf{L}_2(D, \nu)$, denote the $\mathbf{L}_2$ norm of h by $|h|_\nu := \sqrt{\int_D |h|^2(x)\nu(\mathrm{d}x)}$.

(vi) For a probability measure ν, disjoint sets $\{\mathcal{H}_1, \ldots, \mathcal{H}_K\}$ in the support of ν, and finite dimensional function spaces $\mathcal{L}\{\mathcal{L}_1, \ldots, \mathcal{L}_K\}$ such that the domain of $\mathcal{L}_k$ is in the respected set $\mathcal{H}_k$

$$\nu(\dim(\mathcal{L})) = \sum_{k=1}^{K} \nu(\mathcal{H}_k)\dim(\mathcal{L}_k).$$

(vii) For function $g : \mathbb{R}_+ \to \mathbb{R}_+$, the order notation $g(x) = O(x)$ means that there exists some universal unspecified constant, $const > 0$, such that $g(x) \leq const \times x$ for all $x \in \mathbb{R}_+$.

5.2 Mathematical framework and basic properties

We work on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{0\leq t\leq T}, \mathbb{P})$ containing a q-dimensional ($q \geq 1$) Brownian motion W. The filtration $(\mathcal{F}_t)_{0\leq t\leq T}$ satisfies the usual hypotheses. The existence of a unique strong solution X to the forward equation (5.2) follows from usual Lipschitz conditions on b and σ, see $(\mathbf{A_X})$. The BSDE (5.1) is approximated using a multistep-forward dynamical programming equation (MDP) studied in [59]. Let $\pi := \{t_i := i\Delta_t : 0 \leq i \leq N\}$ be the uniform time-grid with time step $\Delta_t = T/N$.

The solution $(Y_i, Z_i)_{0\leq i\leq N-1}$ of the MDP can be written in the form:

$$\left.\begin{array}{rcl} Y_i & = & \mathbb{E}_i\left(g(X_N) + \sum_{j=i}^{N-1} f_j(X_j, Y_{j+1}, Z_j)\Delta_t\right), \\ \Delta_t Z_i & = & \mathbb{E}_i\left((g(X_N) + \sum_{j=i+1}^{N-1} f_j(X_j, Y_{j+1}, Z_j)\Delta_t)\Delta W_i\right) \end{array}\right\} \quad \text{for } i \in \{0, \ldots, N-1\}, \tag{5.5}$$

where $(X_j)_{i\leq j\leq N}$ is a Markov chain approximating the forward component (5.2) (typically the Euler scheme, see Algorithm 2 below), $\Delta W_i := W_{t_{i+1}} - W_{t_i}$ is the $(i+1)$-th Brownian motion increment, and $\mathbb{E}_i(\cdot) := \mathbb{E}(\cdot \mid \mathcal{F}_{t_i})$ is the conditional expectation. Our working assumptions on the functions g and f are as follows:

($\mathbf{A_g}$) g is a bounded measurable function from $\mathbb{R}^d$ to $\mathbb{R}$, the upper bound of which is denoted by C_g.

($\mathbf{A_f}$) for every $i < N$, $f_i(x, y, z)$ is a measurable function $\mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^q$ to $\mathbb{R}$, and there exist two finite constants L_f and C_f such that, for every $i < N$,

$$\begin{aligned} |f_i(x, y, z) - f_i(x, y', z')| &\leq L_f(|y - y'| + |z - z'|), \\ &\qquad \forall (x, y, y', z, z') \in \mathbb{R}^d \times (\mathbb{R})^2 \times (\mathbb{R}^q)^2, \\ |f_i(x, 0, 0)| &\leq C_f, \quad \forall x \in \mathbb{R}^d. \end{aligned}$$

The definition of the Markov chain $(X_j)_j$ is made under the following assumptions.

($\mathbf{A_X}$) The coefficients functions b and σ satisfy

(i) $b : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d \otimes \mathbb{R}^q$ are bounded measurable, uniformly Lipschitz in the space dimensions;

(ii) there exists $\zeta \geq 1$ such that, for all $\xi \in \mathbb{R}^d$, the following inequalities hold: $\zeta^{-1}|\xi|^2 \leq \xi^\top \sigma(t, x)\sigma(t, x)^\top \xi \leq \zeta|\xi|^2$.

Let X_i be a random variable with some distribution η (more details on this to follow). Then X_j for $j > i$ is generated according to one of the two algorithms below:

Algorithm 1 (SDE dynamics)**.** $X_{j+1} = \bar{X}_{t_{j+1}} = X_j + \int_{t_j}^{t_{j+1}} b(s, \bar{X}_s)\mathrm{d}s + \int_{t_j}^{t_{j+1}} \sigma(s, \bar{X}_s)\mathrm{d}W_s$;

Algorithm 2 (Euler dynamics)**.** $X_{j+1} = X_j + b(t_i, X_i)\Delta_t + \sigma(t_i, X_i)\Delta W_i$.

The above ellipticity condition (ii) will be used in the proof of Proposition 5.2.1.

As in the continuous time framework (5.1), the solution of the MDP (5.5) admits a Markov representation: under $(\mathbf{A_g})$, $(\mathbf{A_f})$ and $(\mathbf{A_X})$ (and using for X either the SDE itself or its Euler scheme), for every i, there exist measurable deterministic functions $y_i : \mathbb{R}^d \to \mathbb{R}$ and $z_i : \mathbb{R}^d \to \mathbb{R}^q$, such that $Y_i = y_i(X_i)$ and $Z_i = z_i(X_i)$, almost surely. In fact, the value functions $y_i(\cdot)$ and $z_i(\cdot)$ are independent of how we initialize the forward component.[2]

For the subsequent stratification algorithm, X_i will be sampled randomly (and independently of the Brownian motion W) according to different squared-integrable distributions η. When $X_i \sim \eta$, we will write $(X_j^{(i,\eta)})_{i \le j \le N}$ the Markov chain given in $(\mathbf{A_X})$, using either the SDE dynamics (better when possible) or the Euler one. One can recover the value functions from the conditional expectations: almost surely,

$$y_i(X_i^{(i,\eta)}) = \mathbb{E}\left(g(X_N^{(i,\eta)}) + \sum_{j=i}^{N-1} f_j(X_j^{(i,\eta)}, y_{j+1}(X_{j+1}^{(i,\eta)}), z_j(X_j^{(i,\eta)}))\Delta_t \mid X_i^{(i,\eta)} \right), \tag{5.6}$$

$$\Delta_t z_i(X_i^{(i,\eta)}) = \mathbb{E}\left((g(X_N^{(i,\eta)}) + \sum_{j=i+1}^{N-1} f_j(X_j^{(i,\eta)}, y_{j+1}(X_{j+1}^{(i,\eta)}), z_j(X_j^{(i,\eta)}))\Delta_t)\Delta W_i \mid X_i^{(i,\eta)} \right);$$

the proof of this is the same as [60, Lemma 4.1].

Approximating the solution to (5.5) is actually achieved by approximating the functions $y_i(\cdot)$ and $z_i(\cdot)$. In this way, we are directly approximating the solution to the semilinear PDE (5.3). Our approach consists in approximating the restrictions of the functions y_i and z_i to subsets of a cubic partition of $\mathbb{R}^d$ using finite dimensional

[2]Actually under our assumptions, the measurability of y_i and z_i can be easily established by induction on i. More precisely, we can write y_i and z_i as a $(N-i)$-fold integrals in space, using the C^2 transition density of X given in Algorithms 1 or 2. From this, we observe that z_i is a C^2 function of x_i; regarding y_i all the contributions in the sum for $j > i$ are also smooth, and only the $j = i$ term may be non-smooth (because of $x_i \mapsto f_i(x_i, .)$ is only assumed measurable). From this, we easily see that the initialization x_i of X at time i can be made arbitrary, provided that this is independent of W.

linear function spaces. The basic assumptions for this local approximation approach are given below.

($\mathbf{A}_{\text{Strat.}}$) There are $K \in \mathbb{N}^*$ disjoint hypercubes $(\mathcal{H}_k : 1 \leq k \leq K)$, that is

$$\mathcal{H}_k \cap \mathcal{H}_l = \emptyset, \qquad \bigcup_{k=1}^{K} \mathcal{H}_k = \mathbb{R}^d \quad \text{and} \quad \mathcal{H}_k = \prod_{l=1}^{d} [x_{k,l}^-, x_{k,l}^+)$$

for some $-\infty \leq x_{k,l}^- < x_{k,l}^+ \leq +\infty$. Additionally, there are linear function spaces $\mathcal{L}_{Y,k}$ and $\mathcal{L}_{Z,k}$, valued in $\mathbb{R}$ and $\mathbb{R}^q$ respectively, which are subspaces of $\mathbf{L}_2(\mathcal{H}_k, \nu_k)$ w.r.t. a probability measure ν_k on $\mathcal{H}_k$ defined in ($\mathbf{A}_\nu$) below.

Common examples of hypercubes are:

(i) Hypercubes of equal size: $x_{k,l}^+ - x_{k,l}^- = const > 0$ for all k and l, except for exterior strata that must be infinite.

(ii) Hypercubes of equal probability: $\nu(\mathcal{H}_k) = 1/K$ for some probability ν to be defined later in ($\mathbf{A}_\nu$).

Common examples of local approximations spaces $\mathcal{L}_{Y,k}$ and $\mathcal{L}_{Z,k}$ are:

(i) Piece-wise constant approximation (**LP0**): $\mathcal{L}_{Y,k} := \text{span}\{\mathbf{1}_{\mathcal{H}_k}\}$, and $\mathcal{L}_{Z,k} := (\mathcal{L}_{Y,k})^q$; $\dim(\mathcal{L}_Y) = 1$ and $\dim(\mathcal{L}_{Z,k}) = q$.

(ii) Affine approximations (**LP1**): $\mathcal{L}_{Y,k} := \text{span}\{\mathbf{1}_{\mathcal{H}_k}, x_1\mathbf{1}_{\mathcal{H}_k}, \ldots, x_d\mathbf{1}_{\mathcal{H}_k}\}$, and $\mathcal{L}_{Z,k} := (\mathcal{L}_{Y,k})^q$; $\dim(\mathcal{L}_Y) = d+1$ and $\dim(\mathcal{L}_{Z,k}) = q(d+1)$.

The key idea in this chapter is to select a distribution ν, the restriction of which to the hypercubes $\mathcal{H}_k$, ν_k, can be explicitly computed. Then, we can easily simulate i.i.d. copies of $X_i^{(i,\nu_k)}$ directly in $\mathcal{H}_k$ and use the resulting paths of the Markov chain to estimate $y_k(\cdot)|_{\mathcal{H}_k}$. This sampling method is traditionally known as *stratification*, and for this reason we will call the hypercubes in ($\mathbf{A}_{\text{Strat.}}$) the *strata*. For the stratification, the components $X_i^{(i,\nu_k)}$ are sampled as i.i.d. conditional logistic random variables, which is precisely stated in the following assumption.

($\mathbf{A}_\nu$) Let $\mu > 0$. The distribution of $X_i^{(i,\nu_k)}$ is given by $\mathbb{P} \circ (X_i^{(i,\nu_k)})^{-1}(\mathrm{d}x) = \nu_k(\mathrm{d}x)$, where

$$\nu_k(\mathrm{d}x) = \frac{\mathbf{1}_{\mathcal{H}_k}(x)\nu(\mathrm{d}x)}{\nu(\mathcal{H}_k)},$$

and

$$\nu(\mathrm{d}x) = p^{(\mu)}_{\text{logis.}}(x)\mathrm{d}x, \quad p^{(\mu)}_{\text{logis.}}(x) := \prod_{l=1}^{d} \frac{\mu e^{-\mu x_l}}{(1+e^{-\mu x_l})^2}, \quad x = (x_1, \ldots, x_d) \in \mathbb{R}^d.$$

Remark 5.2.1. *An important relation of ν and ν_k is that one has the $\mathbf{L}_2$-norm identity $|\cdot|_\nu^2 = \sum_{k=1}^K \nu(\mathcal{H}_k)\,|\cdot|_{\nu_k}^2$.*

In order to generate the random variable $X_i^{(i,\nu_k)}$, we make use of the *inverse* conditional distribution function of ν_k and the simulation of uniform random variables, as shown in the following algorithm:

Algorithm 3. *Draw d independent random variables $(U_1, \ldots, U_d)$ which are uniformly distributed on $[0,1]$, and compute*

$$X_i^{(i,\nu_k)} := \left(F^{-1}_{\nu,[x^-_{k,1},x^+_{k,1})}(U_1), \ldots, F^{-1}_{\nu,[x^-_{k,d},x^+_{k,d})}(U_d) \right) \overset{d}{\sim} \nu_k,$$

where we use the functions $F_\nu(x) := \int_{-\infty}^x \nu(\mathrm{d}x') = 1/\left(1+\exp(-\mu x)\right)$ and

$$F^{-1}_{\nu,[x^-,x^+)}(U) = -\frac{1}{\mu}\log\left(\frac{1}{F_\nu(x^-) + U(F_\nu(x^+) - F_\nu(x^-))} - 1\right).$$

A further reason for the choice of the logistic distribution is that it induces the following stability property on the $\mathbf{L}_2$ norms of the Markov chain $(X_j^{(i,\nu)})_{i \le j \le N}$; this property will be crucial for the error analysis of the stratified regression scheme in Section 5.3.2. The proof is postponed to Appendix C.1.

Proposition 5.2.1. *Suppose that ν is the logistic distribution defined in ($\mathbf{A}_\nu$). There is a constant $c_{(\mathbf{A}_\nu)} \in [1, +\infty)$ such that, for any function $h : \mathbb{R}^d \mapsto \mathbb{R}$ or $\mathbb{R}^q$ in $\mathbf{L}_2(\nu)$, for any $0 \le i \le N$, and for any $i \le j \le N-1$, we have*

$$\frac{1}{c_{(\mathbf{A}_\nu)}}\mathbb{E}[|h(X_j^{(i,\nu)})|^2] \le |h|_\nu^2 \le c_{(\mathbf{A}_\nu)}\mathbb{E}[|h(X_j^{(i,\nu)})|^2].$$

To conclude this section, we recall standard uniform absolute bounds for the functions $y_i(\cdot)$ and $z_i(\cdot)$.

Proposition 5.2.2 (*a.s.* upper bounds, [60, Proposition 3.3])**.** *For N large enough such that $\frac{T}{N}L_f^2 \leq \frac{1}{12q}$, we have for any $x \in \mathbb{R}^d$ and any $0 \leq i \leq N-1$,*

$$|y_i(x)| \leq C_y := e^{\frac{T}{4}+6q(1\vee L_f^2)(T\vee 1)}\Big(C_g + \frac{T}{2\sqrt{q}}C_f\Big), \qquad |z_{l,i}(x)| \leq C_z := \frac{C_y}{\sqrt{\Delta_t}}. \quad (5.7)$$

5.3 Stratified algorithm and convergence results

5.3.1 Algorithm

In this section, we define the SRMDP algorithm mathematically, and then expose in Section 5.4 how to efficiently perform it using GPUs. Our algorithm involves solving a sequence of Ordinary linear Least Squares regression (**OLS**) problems. For a precise mathematical statement, we recall the seemingly abstract but very convenient definition from [60]; explicit algorithms for the computation of **OLS** solutions are exposed in Section 5.4.1.

Definition 5.3.1 (Ordinary linear least-squares regression)**.** *For $l, l' \geq 1$ and for probability spaces $(\tilde{\Omega}, \tilde{\mathcal{F}}, \tilde{\mathbb{P}})$ and $(\mathbb{R}^l, \mathcal{B}(\mathbb{R}^l), \eta)$, let S be a $\tilde{\mathcal{F}} \otimes \mathcal{B}(\mathbb{R}^l)$-measurable $\mathbb{R}^{l'}$-valued function such that $S(\omega, \cdot) \in \mathbf{L}_2(\mathcal{B}(\mathbb{R}^l), \eta)$ for $\tilde{\mathbb{P}}$-a.e. $\omega \in \tilde{\Omega}$, and $\mathcal{L}$ a linear vector subspace of $\mathbf{L}_2(\mathcal{B}(\mathbb{R}^l), \eta)$ spanned by deterministic $\mathbb{R}^{l'}$-valued functions $\{p_k(.), k \geq 1\}$. The least squares approximation of S in the space $\mathcal{L}$ with respect to η is the ($\tilde{\mathbb{P}} \times \eta$-a.e.) unique, $\tilde{\mathcal{F}} \otimes \mathcal{B}(\mathbb{R}^l)$-measurable function $S^\star$ given by*

$$S^\star(\omega, \cdot) = \arg\inf_{\phi \in \mathcal{L}} \int |\phi(x) - S(\omega, x)|^2 \eta(\mathrm{d}x).$$

We say that $S^\star$ solves $\mathbf{OLS}(S, \mathcal{L}, \eta)$.

On the other hand, suppose that $\eta_M = \frac{1}{M}\sum_{m=1}^M \delta_{\mathcal{X}^{(m)}}$ is a discrete probability measure on $(\mathbb{R}^l, \mathcal{B}(\mathbb{R}^l))$, where δ_x is the Dirac measure on x and $\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(M)} : \tilde{\Omega} \to \mathbb{R}^l$ are i.i.d. random variables. For an $\tilde{\mathcal{F}} \otimes \mathcal{B}(\mathbb{R}^l)$-measurable $\mathbb{R}^{l'}$-valued function S

such that $\left|S\big(\omega, \mathcal{X}^{(m)}(\omega)\big)\right| < \infty$ *for any* m *and* $\tilde{\mathbb{P}}$*-a.e.* $\omega \in \tilde{\Omega}$*, the least squares approximation of* S *in the space* $\mathcal{L}$ *with respect to* η_M *is the (*$\tilde{\mathbb{P}}$*-a.e.) unique,* $\tilde{\mathcal{F}} \otimes \mathcal{B}(\mathbb{R}^l)$*–measurable function* $S^\star$ *given by*

$$S^\star(\omega, \cdot) = \arg\inf_{\phi \in \mathcal{L}} \frac{1}{M} \sum_{m=1}^{M} |\phi\big(\mathcal{X}^{(m)}(\omega)\big) - S\big(\omega, \mathcal{X}^{(m)}(\omega)\big)|^2.$$

We say that $S^\star$ *solves* $\mathbf{OLS}(S, \mathcal{L}, \eta_M)$*.*

Definition 5.3.2 (Simulations and empirical measures)**.** *Recall the Markov chain* $(X_j^{(i,\nu_k)})_{i \le j \le N}$ *initialized as in* $(\mathbf{A}_\nu)$*. For any* $i \in \{0, \dots, N-1\}$ *and* $k \in \{1, \dots, K\}$*, define* $M \ge \dim(\mathcal{L}_{Y,k}) \vee \dim(\mathcal{L}_{Z,k})$ *independent copies of* $(\Delta W_i, (X_j^{i,\nu_k})_{i \le j \le N})$ *that we denote by*

$$\mathcal{C}_{i,k} := \left\{ (\Delta W_i^{(i,k,m)}, (X_j^{(i,k,m)})_{i \le j \le N}) \ : \ m = 1, \dots, M \right\}.$$

The random variables $\mathcal{C}_{i,k}$ *form a* cloud of simulations *used for the regression at time* i *and in the stratum* k*. Furthermore, we assume that the clouds of simulations* $(\mathcal{C}_{i,k} : 0 \le i \le N-1, 1 \le k \le K)$ *are independently generated. All these random variables are defined on a probability space* $(\Omega^{(M)}, \mathcal{F}^{(M)}, \mathbb{P}^{(M)})$*. Denote by* $\nu_{i,k,M}$ *the empirical probability measure of the* $\mathcal{C}_{i,k}$*-simulations, i.e.*

$$\nu_{i,k,M} = \frac{1}{M} \sum_{m=1}^{M} \delta_{(\Delta W_i^{(i,k,m)}, X_i^{(i,k,m)}, \dots, X_N^{(i,k,m)})}.$$

Denoting by $(\Omega, \mathcal{F}, \mathbb{P})$ *the probability space supporting* $(\Delta W_i, X^{i,\nu_k} : 0 \le i \le N-1, 1 \le k \le K)$*, which serves as a generic element for the clouds of simulations* $\mathcal{C}_{i,k}$*, the full probability space used to analyze our algorithm is the product space* $(\bar{\Omega}, \bar{\mathcal{F}}, \bar{\mathbb{P}}) = (\Omega, \mathcal{F}, \mathbb{P}) \otimes (\Omega^{(M)}, \mathcal{F}^{(M)}, \mathbb{P}^{(M)})$*. By a slight abuse of notation, we write* $\mathbb{P}$ *(resp.* $\mathbb{E}$*) to mean* $\bar{\mathbb{P}}$ *(resp.* $\bar{\mathbb{E}}$*) from now on.*

We now come to the definition of the stratified LSMDP algorithm, which computes random approximations $y_i^{(M)}(.)$ and $z_i^{(M)}(.)$

Algorithm 4 (SRMDP)**.** *Recall the linear spaces* $\mathcal{L}_{Y,k}$ *and* $\mathcal{L}_{Z,k}$ *from* $(\mathbf{A}_{\text{Strat.}})$*, the bounds* (5.7) *and the truncation function* $\mathcal{T}_L$ *(see* (5.4)*).*

Initialization. *Set* $y_N^{(M)}(\cdot) := g(\cdot)$.

Backward iteration for $i = N-1$ **to** $i = 0$. *For any stratum index* $k \in \{1, \dots, K\}$, *generate the empirical measure* $\nu_{i,k,M}$ *as in Definition 5.3.2, and define*

$$\begin{cases} \psi_{Z,i,k}^{(M)}(\cdot) \quad \textit{solution of} \quad \mathbf{OLS}(S_{Z,i}^{(M)}(w, \underline{\mathbf{x}}_i) \ , \ \mathcal{L}_{Z,k} \ , \ \nu_{i,k,M}) \\ \qquad \textit{for} \quad S_{Z,i}^{(M)}(w, \underline{\mathbf{x}}_i) := \dfrac{1}{\Delta_t} S_{Y,i+1}^{(M)}(\underline{\mathbf{x}}_i) \ w, \\ z_i^{(M)}(\cdot)|_{\mathcal{H}_k} := \mathcal{T}_{C_z}\big(\psi_{Z,i,k}^{(M)}(\cdot)\big) \quad (\textit{truncation}), \\ \psi_{Y,i,k}^{(M)}(\cdot) \quad \textit{solution of} \quad \mathbf{OLS}(S_{Y,i}^{(M)}(\underline{\mathbf{x}}_i) \ , \ \mathcal{L}_{Y,k} \ , \ \nu_{i,k,M}) \\ \qquad \textit{for} \quad S_{Y,i}^{(M)}(\underline{\mathbf{x}}_i) := g(x_N) + \displaystyle\sum_{j=i}^{N-1} f_j\big(x_j, y_{j+1}^{(M)}(x_{j+1}), z_j^{(M)}(x_j)\big)\Delta_t, \\ y_i^{(M)}(\cdot)|_{\mathcal{H}_k} := \mathcal{T}_{C_y}\big(\psi_{Y,i,k}^{(M)}(\cdot)\big) \quad (\textit{truncation}), \end{cases} \tag{5.8}$$

where $w \in \mathbb{R}^q$ *and* $\underline{\mathbf{x}}_i = (x_i, \dots, x_N) \in (\mathbb{R}^d)^{N-i+1}$.

An important difference between SRMDP and established Monte Carlo algorithms [55, 98, 59, 60] is that the number of simulations falling in each hypercube is no more random but fixed and equal to M. Observe first that this is likely to improve the numerical stability of the regression algorithm: there is no risk that too few simulations will land in the hypercube, leading to under-fitting. Later, in Section 5.4, we shall explain how to implement Algorithm 4 on a GPU device. The key point is that the calculations at every time point are fully independent between the different hypercubes, so that we can perform them in parallel across the hypercubes. The choice of M independent on k is made in order to maintain a computational effort equal on each of the strata. In this way, the gain in parallelization is likely to be the largest. However, the subsequent mathematical analysis can be easily adapted to make the number of simulations vary with k whenever necessary.

An easy but important consequence of Algorithm 4 and of the bounds of Proposition 5.2.2 is the following absolute bound; the proof is analogous to that of [60, Lemma 4.7].

Lemma 5.3.3. *With the above notation, we have*

$$\sup_{0\leq i\leq N}\sup_{\underline{\mathbf{x_i}}\in(\mathbb{R}^d)^{N-i+1}}|S_{Y,i}^{(M)}(\underline{\mathbf{x_i}})|\leq C_{5.3.3}:=C_g+T\left(L_fC_y\left[1+\frac{\sqrt{q}}{\sqrt{\Delta_t}}\right]+C_f\right).$$

5.3.2 Error analysis

The analysis will be performed according to several $\mathbf{L}_2$-norms, either w.r.t. the probability measure ν, or the empirical norm related to the cloud simulations. They are defined as follows:

$$\begin{aligned}
\mathcal{E}(Y,M,i)&:=\sum_{k=1}^{K}\nu(\mathcal{H}_k)\mathbb{E}\left(\left|y_i^{(M)}(\cdot)-y_i(\cdot)\right|_{i,k,M}^2\right),\\
\bar{\mathcal{E}}(Y,M,i)&:=\sum_{k=1}^{K}\nu(\mathcal{H}_k)\mathbb{E}\left(\left|y_i^{(M)}(\cdot)-y_i(\cdot)\right|_{\nu_k}^2\right)=\mathbb{E}\left(\left|y_i^{(M)}(\cdot)-y_i(\cdot)\right|_{\nu}^2\right),\\
\mathcal{E}(Z,M,i)&:=\sum_{k=1}^{K}\nu(\mathcal{H}_k)\mathbb{E}\left(\left|z_i^{(M)}(\cdot)-z_i(\cdot)\right|_{i,k,M}^2\right),\\
\bar{\mathcal{E}}(Z,M,i)&:=\sum_{k=1}^{K}\nu(\mathcal{H}_k)\mathbb{E}\left(\left|z_i^{(M)}(\cdot)-z_i(\cdot)\right|_{\nu_k}^2\right)=\mathbb{E}\left(\left|z_i^{(M)}(\cdot)-z_i(\cdot)\right|_{\nu}^2\right),
\end{aligned}$$

where

$$|h|_{i,k,M}:=\left(\int|h|^2(\omega,\underline{\mathbf{x}}_i)\nu_{i,k,M}(\mathrm{d}\omega,\mathrm{d}\underline{\mathbf{x}}_i)\right)^{1/2}.$$

In fact, the norms $\mathcal{E}(.,M,i)$ and $\bar{\mathcal{E}}(.,M,i)$ are related through model-free concentration-of-measures inequalities. This relation is summarized in the proposition below.

Proposition 5.3.4. *For each $i\in\{0,\ldots,N-1\}$, we have*

$$\begin{aligned}
\bar{\mathcal{E}}(Y,M,i)&\leq 2\mathcal{E}(Y,M,i)+\frac{2028C_y^2\log(3M)}{M}\left(\nu(\dim(\mathcal{L}_{Y,.}))+1\right),\\
\bar{\mathcal{E}}(Z,M,i)&\leq 2\mathcal{E}(Z,M,i)+\frac{2028qC_y^2\log(3M)}{\Delta_tM}\left(\nu(\dim(\mathcal{L}_{Z,.}))+1\right).
\end{aligned}$$

Proof. It is clearly sufficient to show that

$$\mathbb{E}\left(\left|y_i^{(M)}(\cdot)-y_i(\cdot)\right|_{\nu_k}^2\right) \leq 2\mathbb{E}\left(\left|y_i^{(M)}(\cdot)-y_i(\cdot)\right|_{i,k,M}^2\right) + \frac{2028C_y^2\log(3M)}{M}\left(\dim(\mathcal{L}_{Y,\cdot})+1\right),$$

$$\mathbb{E}\left(\left|z_i^{(M)}(\cdot)-z_i(\cdot)\right|_{\nu_k}^2\right) \leq 2\mathbb{E}\left(\left|z_i^{(M)}(\cdot)-z_i(\cdot)\right|_{i,k,M}^2\right) + \frac{2028qC_y^2\log(3M)}{\Delta_t M}\left(\dim(\mathcal{L}_{Z,\cdot})+1\right),$$

which follows exactly as in the proof of [60, Proposition 4.10]. □

From the previous proposition, the controls on $\bar{\mathcal{E}}(Y,M,i)$ and $\bar{\mathcal{E}}(Z,M,i)$ stem from those on $\mathcal{E}(Y,M,i)$ and $\mathcal{E}(Z,M,i)$, which are handled in Theorem 5.3.5 below. In order to study the impact of basis selection, we define the squared *quadratic approximation errors* associated to the basis in hypercube $\mathcal{H}_k$ by

$$T_{i,k}^Y := \inf_{\phi\in\mathcal{L}_{Y,k}} |\phi-y_i|_{\nu_k}^2, \quad T_{i,k}^Z := \inf_{\phi\in\mathcal{L}_{Z,k}} |\phi-z_i|_{\nu_k}^2.$$

These terms are the minimal error that can possibly be achieved by the basis $\mathcal{L}_{Y,k}$ (resp. $\mathcal{L}_{Z,k}$) in order to approximate the restriction $y_i(\cdot)|_{\mathcal{H}_k}$ (resp. $z_i(\cdot)|_{\mathcal{H}_k}$) in the $\mathbf{L}_2$ norm. Consequently, the global squared quadratic approximation error is given by

$$T_i^Y := \sum_{k=1}^K \nu(\mathcal{H}_k)T_{i,k}^Y = \inf_{\phi \text{ s.t. } \phi|_{\mathcal{H}_k}\in\mathcal{L}_{Y,k}} |\phi-y_i|_\nu^2, \tag{5.9}$$

$$T_i^Z := \sum_{k=1}^K \nu(\mathcal{H}_k)T_{i,k}^Z = \inf_{\phi \text{ s.t. } \phi|_{\mathcal{H}_k}\in\mathcal{L}_{Z,k}} |\phi-z_i|_\nu^2. \tag{5.10}$$

As we shall see in Theorem 5.3.5 below, the terms T_i^Y and T_i^Z are closely associated to the limit of the expected quadratic error of the numerical scheme in the asymptotic $M\to\infty$; for this reason, these terms are usually called *bias* terms.

Now, we are in the position to state our main result giving non-asymptotic error estimates.

Theorem 5.3.5 (Error for the Stratified LSMDP scheme)**.** *Recall the constants C_y from Proposition 5.2.2, $C_{5.3.3}$ from Lemma 5.3.3, and $c_{(\mathbf{A}_\nu)}$ from Proposition 5.2.1. For each $i \in \{0, \dots, N-1\}$, define*

$$\begin{aligned}\mathcal{E}(i) := 2\sum_{j=i}^{N-1} \Delta_t \Big(T_j^Y + 3C_{5.3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Y,.}))}{M} + 12168 L_f^2 \Delta_t \frac{(\nu(\dim(\mathcal{L}_{Z,.})) + 1) q C_y^2 \log(3M)}{M} \\ + 3T_j^Z + 6qC_{5.3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Z,.}))}{\Delta_t M} \Big) \\ + (T - t_i) \frac{1014 C_y^2 \log(3M)}{M} \left((\nu(\dim(\mathcal{L}_{Y,.})) + 1) + \frac{q}{\Delta_t} (\nu(\dim(\mathcal{L}_{Z,.})) + 1) \right).\end{aligned}$$

For Δ_t small enough such that $L_f \Delta_t \leq \sqrt{\frac{2}{15}}$ and $\Delta_t L_f^2 \leq \frac{1}{288 c_{(\mathbf{A}_\nu)}^2 C_{C.2.1}(1+T)}$, we have, for all $0 \leq i \leq N-1$,

$$\begin{aligned}\mathcal{E}(Y, M, i) \leq T_i^Y + 3C_{5.3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Y,.}))}{M} + 12168 L_f^2 \Delta_t \frac{(\nu(\dim(\mathcal{L}_{Z,.})) + 1) q C_y^2 \log(3M)}{M} \\ + (1 + 15 L_f^2 \Delta_t) C_{5.3.5} \mathcal{E}(i), \qquad (5.11)\end{aligned}$$

$$\sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Z, M, j) \leq C_{5.3.5} \mathcal{E}(i), \qquad (5.12)$$

where $C_{5.3.5} := \exp(288 c_{(\mathbf{A}_\nu)}^2 C_{C.2.1}(1+T) L_f^2 T)$.

5.3.3 Proof of Theorem 5.3.5

We start by obtaining estimates on the *local* empirical quadratic errors terms

$$\mathbb{E}\left(\left| y_i^{(M)}(\cdot) - y_i(\cdot) \right|_{i,k,M}^2 \right), \quad \mathbb{E}\left(\left| z_i^{(M)}(\cdot) - z_i(\cdot) \right|_{i,k,M}^2 \right),$$

on each of the hypercubes $\mathcal{H}_k$ $(k = 1, \dots, K)$. We first reformulate (5.6) with $\eta = \nu_k$ in terms of the Definition 5.3.1 of OLS. For each $i \in \{0, \dots, N-1\}$ and $k \in \{1, \dots, K\}$,

let $\nu_{i,k} := \mathbb{P} \circ (\Delta W_i, X_i^{i,\nu_k}, \ldots, X_N^{i,\nu_k})^{-1}$, so that we have

$$\begin{cases} y_i(\cdot)|_{\mathcal{H}_k} \text{ solution of } \mathbf{OLS}(\ S_{Y,i}(\underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_k^{(1)}\ ,\ \nu_{i,k}\ ) \\ \qquad \text{where } S_{Y,i}(\underline{\mathbf{x}}_i) := g(x_N) + \displaystyle\sum_{j=i}^{N-1} f_j\big(x_j, y_{j+1}(x_{j+1}), z_j(x_j)\big)\Delta_t, \\ z_i(\cdot)|_{\mathcal{H}_k} \text{ solution of } \mathbf{OLS}(\ S_{Z,i}(w, \underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_k^{(q)}\ ,\ \nu_{i,k}\ ) \\ \qquad \text{where } S_{Z,i}(w, \underline{\mathbf{x}}_i) := \dfrac{1}{\Delta_t} S_{Y,i+1}(\underline{\mathbf{x}}_i)\ w, \end{cases}$$

where $w \in \mathbb{R}^q$, $\underline{\mathbf{x}}_i := (x_i, \ldots, x_N) \in (\mathbb{R}^d)^{N-i+1}$ and where $\mathcal{L}_k^{(l')}$ is any dense separable subspace in the $\mathbb{R}^{l'}$-valued functions belonging to $\mathbf{L}_2(\mathcal{B}(\mathcal{H}_k), \nu_k)$. The above OLS solutions and those defined in (5.8) will be compared with other intermediate OLS solutions given by

$$\begin{cases} \psi_{Y,i,k}(\cdot) \quad \text{solution of} \quad \mathbf{OLS}(\ S_{Y,i}(\underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_{Y,k}\ ,\ \nu_{i,k,M}), \\ \psi_{Z,i,k}(\cdot) \quad \text{solution of} \quad \mathbf{OLS}(\ S_{Z,i}(w, \underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_{Z,k}\ ,\ \nu_{i,k,M}). \end{cases}$$

In order to handle the dependence on the simulation clouds, we define the following σ-algebras.

Definition 5.3.6. *Define the σ-algebras*

$$\mathcal{F}_i^{(*)} := \sigma(\mathcal{C}_{i+1,k}, \ldots, \mathcal{C}_{N-1,k} : 1 \le k \le K), \quad \mathcal{F}_{i,k}^{(M)} := \mathcal{F}_i^{(*)} \vee \sigma(X_i^{(i,k,m)} :\ 1 \le m \le M).$$

For every $i \in \{0, \ldots, N-1\}$ and $k \in \{1, \ldots, K\}$, let $\mathbb{E}_{i,k}^{(M)}(\cdot)$ (resp. $\mathbb{P}_{i,k}^{M}(\cdot)$) with respect to $\mathcal{F}_{i,k}^{(M)}$.

Defining additionally the functions

$$\begin{aligned} \xi_{Y,i}^*(x) &:= \mathbb{E}\left(S_{Y,i}^{(M)}(\underline{\mathbf{X}}_i) - S_{Y,i}(\underline{\mathbf{X}}_i) \mid X_i = x, \mathcal{F}^{(M)}\right), \\ \xi_{Z,i}^*(x) &:= \mathbb{E}\left(S_{Z,i}^{(M)}(\Delta W_i, \underline{\mathbf{X}}_i) - S_{Z,i}(\Delta W_i, \underline{\mathbf{X}}_i) \mid X_i = x, \mathcal{F}^{(M)}\right), \end{aligned}$$

now we are in the position to prove that

$$\mathbb{E}\left(\left|y_i(\cdot)-y_i^{(M)}(\cdot)\right|^2_{i,k,M}\right) \leq T^Y_{i,k}+6\mathbb{E}\left(\left|\xi^*_{Y,i}(\cdot)\right|^2_{\nu_k}\right)+3C^2_{5.3.3}\frac{\dim(\mathcal{L}_{Y,k})}{M}, \\ +15L_f^2\Delta_t^2\mathbb{E}\left(\left|z_i(\cdot)-z_i^{(M)}(\cdot)\right|^2_{i,k,M}\right) \\ +12168L_f^2\Delta_t\frac{(\dim(\mathcal{L}_{Z,k})+1)qC_y^2\log(3M)}{M}, \tag{5.13}$$

$$\mathbb{E}\left(\left|z_i(\cdot)-z_i^{(M)}(\cdot)\right|^2_{i,k,M}\right) \leq T^Z_{i,k}+2\mathbb{E}\left(\left|\xi^*_{Z,i}(\cdot)\right|^2_{\nu_k}\right)+2qC^2_{5.3.3}\frac{\dim(\mathcal{L}_{Z,k})}{\Delta_t M}. \tag{5.14}$$

In fact, the proof of (5.13)–(5.14) follows analogously the proof of [60, (4.12)–(4.13)]; in order to follow the steps of that proof, one must note that the term R_π of that paper is equal to 1 here, C_π is equal to Δ_t, and $\theta_L = 1$. Moreover, one must exchange all norms, **OLS** problems, σ-algebras, and empirical functions from the reference to the localized versions defined in the preceding paragraphs. Indeed, the proof method of [60, (4.12)–(4.13)] is model free in the sense that it does not care about the distribution of the Markov chain at time t_i.

We now aim at aggregating the previous estimates across the strata and propagating them along time. For this, let

$$\mathcal{E}_1(i) := \sum_{j=i}^{N-1}\Delta_t\Big(T^Y_j+3C^2_{5.3.3}\frac{\nu(\dim(\mathcal{L}_{Y,.}))}{M}+12168L_f^2\Delta_t\frac{(\nu(\dim(\mathcal{L}_{Z,.}))+1)qC_y^2\log(3M)}{M} \\ +3T^Z_j+6qC^2_{5.3.3}\frac{\nu(\dim(\mathcal{L}_{Z,.}))}{\Delta_t M}\Big)\Gamma_j, \tag{5.15}$$

where $\Gamma_i := (1+\gamma\Delta_t)^i$ with γ to be determined below. Next, defining

$$\gamma := 288c^2_{(\mathbf{A}_\nu)}C_{C.2.1}(1+T)L_f^2. \tag{5.16}$$

and recalling that $\Delta_t L_f^2 \leq \frac{1}{288c^2_{(\mathbf{A}_\nu)}C_{C.2.1}(1+T)}$, then γ and Δ_t satisfy

$$\max\left(\frac{1}{\gamma}\times 12c^2_{(\mathbf{A}_\nu)}C_{C.2.1}(1+T)L_f^2, \Delta_t\times 12c^2_{(\mathbf{A}_\nu)}C_{C.2.1}(1+T)L_f^2\right) \leq \frac{1}{6}\times\frac{1}{4}. \tag{5.17}$$

Additionally, $\Gamma_i \leq \exp(\gamma T) := C_{5.3.5}$ for every $0 \leq i \leq N$. Now, multiply (5.13) and (5.14) by $\nu(\mathcal{H}_k)\Delta_t\Gamma_i$ and sum them up over i and k to ascertain that

$$\begin{aligned}
&\sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Y,M,j)\Gamma_j + \sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Z,M,j)\Gamma_j \\
&\qquad \leq \sum_{j=i}^{N-1} \Delta_t \left(T_j^Y + 3C_{5.3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Y,k}))}{M} + 12168 L_f^2 \Delta_t \frac{(\nu(\dim(\mathcal{L}_{Z,k}))+1) q C_y^2 \log(3M)}{M} \right) \Gamma_j \\
&\quad + \sum_{j=i}^{N-1} \Delta_t \left\{ \left(T_j^Z + 2qC_{5.3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Z,k}))}{\Delta_t M} + 2\mathbb{E}\left(\left| \xi^*_{Z,j}(\cdot) \right|_\nu^2 \right) \right) (1 + 15L_f^2\Delta_t^2) + 6\mathbb{E}\left(\left| \xi^*_{Y,j}(\cdot) \right|_\nu^2 \right) \right\} \Gamma_j \\
&\qquad \leq \mathcal{E}_1(i) + 6 \sum_{j=i}^{N-1} \Delta_t \left(\mathbb{E}\left(\left| \xi^*_{Y,j}(\cdot) \right|_\nu^2 \right) + \mathbb{E}\left(\left| \xi^*_{Z,j}(\cdot) \right|_\nu^2 \right) \right) \Gamma_j \,, \qquad (5.18)
\end{aligned}$$

where we have used $(1 + 15L_f^2\Delta_t^2) \leq 3$ (since $L_f\Delta_t \leq \sqrt{\frac{2}{15}}$), and the term $\mathcal{E}_1$ from (5.15) above. Next, from Proposition 5.2.1, we have

$$\mathbb{E}\left(\left| \xi^*_{Y,j}(\cdot) \right|_\nu^2 \right) + \mathbb{E}\left(\left| \xi^*_{Z,j}(\cdot) \right|_\nu^2 \right) \leq c_{(\mathbf{A}_\nu)} \left(\mathbb{E}\left(|\xi^*_{Y,j}(X_j^{0,\nu})|^2 \right) + \mathbb{E}\left(|\xi^*_{Z,j}(X_j^{0,\nu})|^2 \right) \right).$$

Furthermore, note that $(\xi^*_{Y,j}(X_j^{0,\nu}), \xi^*_{Z,j}(X_j^{0,\nu}) : 0 \leq j \leq N-1)$ solves a discrete BSDE (in the sense of Appendix C.2) with terminal condition 0 and driver

$$f_{\xi^*,j}(y,z) := f_j(X_j^{0,\nu}, y_{j+1}^{(M)}(X_{j+1}^{0,\nu}), z_j^{(M)}(X_j^{0,\nu})) - f_j(X_j^{0,\nu}, y_{j+1}(X_{j+1}^{0,\nu}), z_j(X_j^{0,\nu})).$$

This allows the application of Proposition C.2.1, with the first BSDE $(\xi^*_{Y,j}(X_j^{0,\nu}), \xi^*_{Z,j}(X_j^{0,\nu}) : 0 \leq j \leq N-1)$, and the second one equal to 0: since $L_{f_2} = 0$, any choice of $\gamma > 0$ is valid and we take γ as in (5.16). We obtain

$$\begin{aligned}
&\sum_{j=i}^{N-1} \Delta_t \left(\mathbb{E}\left(\left| \xi^*_{Y,j}(\cdot) \right|_\nu^2 \right) + \mathbb{E}\left(\left| \xi^*_{Z,j}(\cdot) \right|_\nu^2 \right) \right) \Gamma_j \\
&\leq 6 c_{(\mathbf{A}_\nu)} C_{C.2.1} (1+T) \left(\frac{1}{\gamma} + \Delta_t \right) L_f^2 \sum_{j=i}^{N-1} \Delta_t \\
&\times \left[\mathbb{E}\left(|y_{j+1}^{(M)}(X_{j+1}^{0,\nu}) - y_{j+1}(X_{j+1}^{0,\nu})|^2 \right) + \mathbb{E}\left(|z_j^{(M)}(X_j^{0,\nu}) - z_j(X_j^{0,\nu})|^2 \right) \right] \Gamma_j.
\end{aligned}$$

Now, Proposition 5.2.1 yields to

$$\begin{aligned}
&\mathbb{E}\left(|y_{j+1}^{(M)}(X_{j+1}^{0,\nu}) - y_{j+1}(X_{j+1}^{0,\nu})|^2\right) + \mathbb{E}\left(|z_j^{(M)}(X_j^{0,\nu}) - z_j(X_j^{0,\nu})|^2\right) \\
&\leq c_{(\mathbf{A}_\nu)}[\bar{\mathcal{E}}(Y, M, j+1) + \bar{\mathcal{E}}(Z, M, j)] \\
&\leq 2c_{(\mathbf{A}_\nu)}[\mathcal{E}(Y, M, j+1) + \mathcal{E}(Z, M, j)] + c_{(\mathbf{A}_\nu)}\frac{2028C_y^2\log(3M)}{M}\left(\nu(\dim(\mathcal{L}_{Y,\cdot})) + 1\right) \\
&\qquad + c_{(\mathbf{A}_\nu)}\frac{2028qC_y^2\log(3M)}{\Delta_t M}\left(\nu(\dim(\mathcal{L}_{Z,\cdot})) + 1\right) ,
\end{aligned}$$

where the last inequality follows from the concentration-measure inequalities in Proposition 5.3.4. In order to summarize this, we define

$$\mathcal{E}_2(i) := \frac{1014C_y^2\log(3M)}{M}\left(\sum_{j=i}^{N-1}\Delta_t\Gamma_j\right)\left(\left(\nu(\dim(\mathcal{L}_{Y,\cdot})) + 1\right) + \frac{q}{\Delta_t}\left(\nu(\dim(\mathcal{L}_{Z,\cdot})) + 1\right)\right)$$

and make use of (5.17), and that $\Gamma_j \leq \Gamma_{j+1}$ in order to ascertain that we have

$$\begin{aligned}
&\sum_{j=i}^{N-1}\Delta_t\left(\mathbb{E}\left(\left|\xi_{Y,j}^*(\cdot)\right|_\nu^2\right) + \mathbb{E}\left(\left|\xi_{Z,j}^*(\cdot)\right|_\nu^2\right)\right)\Gamma_j \\
&\leq 12c_{(\mathbf{A}_\nu)}^2 C_{C.2.1}(1+T)\left(\frac{1}{\gamma} + \Delta_t\right)L_f^2\left[\sum_{j=i}^{N-1}\Delta_t\left(\mathcal{E}(Y, M, j) + \mathcal{E}(Z, M, j)\right)\Gamma_j + \mathcal{E}_2(i)\right] \\
&\leq \frac{1}{6}\times\frac{1}{2}\left[\sum_{j=i}^{N-1}\Delta_t\left(\mathcal{E}(Y, M, j) + \mathcal{E}(Z, M, j)\right)\Gamma_j + \mathcal{E}_2(i)\right].
\end{aligned}$$

By plugging this into (5.18) readily yields to

$$\begin{aligned}
&\sum_{j=i}^{N-1}\Delta_t\mathcal{E}(Y, M, j)\Gamma_j + \sum_{j=i}^{N-1}\Delta_t\mathcal{E}(Z, M, j)\Gamma_j \\
&\qquad\qquad \leq \mathcal{E}_1(i) + \frac{1}{2}\left[\sum_{j=i}^{N-1}\Delta_t\left(\mathcal{E}(Y, M, j) + \mathcal{E}(Z, M, j)\right)\Gamma_j + \mathcal{E}_2(i)\right]
\end{aligned}$$

and therefore

$$\sum_{j=i}^{N-1}\Delta_t\mathcal{E}(Y, M, j)\Gamma_j + \sum_{j=i}^{N-1}\Delta_t\mathcal{E}(Z, M, j)\Gamma_j \leq 2\mathcal{E}_1(i) + \mathcal{E}_2(i). \tag{5.19}$$

This completes the proof of the estimate (5.12) on z as stated in Theorem 5.3.5, using $1 \leq \Gamma_i \leq C_{5.3.5}$ and $2\mathcal{E}_1(i) + \mathcal{E}_2(i) \leq C_{5.3.5}\mathcal{E}(i)$. It remains to derive (5.11). Starting from (5.13), multiplying by $\nu(\mathcal{H}_k)$ and summing over k yields to

$$\begin{aligned}\mathcal{E}(Y,M,i) \leq & T_i^Y + 6\mathbb{E}\left(\left|\xi_{Y,i}^*(\cdot)\right|_\nu^2\right) + 3C_{5.3.3}^2\frac{\nu(\dim(\mathcal{L}_{Y,\cdot}))}{M} \\ & + 15L_f^2\Delta_t(2\mathcal{E}_1(i) + \mathcal{E}_2(i)) \\ & + 12168L_f^2\Delta_t\frac{(\nu(\dim(\mathcal{L}_{Z,\cdot})) + 1)qC_y^2\log(3M)}{M} \end{aligned} \tag{5.20}$$

where we use the inequality (5.19) to control $\Delta_t\mathcal{E}(Z,M,i)$. Using the same arguments as before, we upper bound $\mathbb{E}\left(\left|\xi_{Y,i}^*(\cdot)\right|_\nu^2\right)$ by

$$6c_{(\mathbf{A}_\nu)}^2 C_{C.2.1}\left(\frac{1}{\gamma} + \Delta_t\right) L_f^2 \sum_{j=i}^{N-1} \Delta_t \left(\bar{\mathcal{E}}(Y,M,j) + \bar{\mathcal{E}}(Z,M,j)\right)\Gamma_j.$$

By additionally bounding $\bar{\mathcal{E}}(Y,M,j)$ and $\bar{\mathcal{E}}(Z,M,j)$ using the concentration-measure inequalities of Proposition 5.3.4 and plugging this in (5.20), we finally obtain

$$\begin{aligned}\mathcal{E}(Y,M,i) \leq & T_{1,i}^Y + 3C_{5.3.3}^2\frac{\nu(\dim(\mathcal{L}_{Y,\cdot}))}{M} + 15L_f^2\Delta_t\left(2\mathcal{E}_1(i) + \mathcal{E}_2(i)\right) \\ & + 12168L_f^2\Delta_t\frac{(\nu(\dim(\mathcal{L}_{Z,\cdot})) + 1)qC_y^2\log(3M)}{M} \\ & + 72c_{(\mathbf{A}_\nu)}^2 C_{C.2.1}\left(\frac{1}{\gamma} + \Delta_t\right) L_f^2\left[\sum_{j=i}^{N-1}\Delta_t\left(\mathcal{E}(Y,M,j) + \mathcal{E}(Z,M,j)\right)\Gamma_j + \mathcal{E}_2(i)\right].\end{aligned}$$

From (5.17) and (5.19), the last term in previous inequality is bounded by

$$\left(\frac{1}{4(1+T)} + \frac{1}{4(1+T)}\right)(2\mathcal{E}_1(i) + \mathcal{E}_2(i) + \mathcal{E}_2(i)) \leq \mathcal{E}_1(i) + \mathcal{E}_2(i) \leq 2\mathcal{E}_1(i) + \mathcal{E}_2(i).$$

This completes the proof of (5.13), using again $2\mathcal{E}_1(i) + \mathcal{E}_2(i) \leq C_{5.3.5}\mathcal{E}(i)$. □

5.4 GPU implementation

In this section, we consider the computation of $y_i^{(M)}(\cdot)$ for a given stratum $\mathcal{H}_k$ and time point i. The calculation of $z_i^{(M)}(\cdot)$ is rather similar, only requiring component-wise calculations to be taken into account, so that we do not provide details. The

theoretical description of the calculation was given in Section 5.3.1. In this section, we first describe the required computations to implement the approximations with **LP0** and **LP1** local polynomials in Section 5.4.1, and then present their implementation on the GPU in Section 5.4.2. [3]

5.4.1 Explicit solutions to OLS in Algorithm 4

LP0 This piecewise solution is given by the simple formula [66, Ch. 4]

$$y_i^{(M)}(\cdot)|_{\mathcal{H}_k} = \mathcal{T}_{C_y}\left(\frac{\sum_{m=1}^{M} S_{Y,i}^{(M)}(\underline{\mathbf{X}}_i^{(i,k,m)})}{M}\right). \tag{5.21}$$

Observe that there will be a memory consumption of $O(1)$ per hypercube to store the simulations needed for the computation of $S_{Y,i}^{(M)}(\underline{\mathbf{X}}_i^{(i,k,m)})$. Once added in the sum (5.21), their allocation can be freed.

LP1 Let A be the $\mathbb{R}^M \otimes \mathbb{R}^{d+1}$ matrix, the components of which are given by $A[m,j] = 1\mathbf{1}_{\{0\}}(j) + X_{i,j}^{(i,k,m)}\mathbf{1}_{\{0\}^c}(j)$, where $X_{i,j}^{(i,k,m)}$ is the j-th component of $\underline{\mathbf{X}}_i^{(i,k,m)}$, and let S be the $\mathbb{R}^M$ vector given by $S[m] = S_{Y,i}^{(M)}(\underline{\mathbf{X}}_i^{(i,k,m)})$. In order to compute $y_i^{(M)}(\cdot)|_{\mathcal{H}_k}$, we first perform a QR-factorization $A = QR$, where Q is an $\mathbb{R}^M \otimes \mathbb{R}^M$ orthogonal matrix, and R is an $\mathbb{R}^M \otimes \mathbb{R}^{d+1}$ upper triangular matrix. The computational cost to compute this factorization is $(d+1)^2\,(M-(d+1)/3)$ flops using the Householder reflections method [61, Alg. 5.3.2]. Using the form of **LP1** and the density of ν_k, we can prove that the rank of A is $d+1$ with probability 1, i.e. R is invertible a.s. (the OLS problem is non-degenerate).

Then, we obtain the approximation $y_i^{(M)}(\cdot)|_{\mathcal{H}_k}$ by computing the coefficients $\alpha = (\alpha_0, \ldots, \alpha_d) \in \mathbb{R}^{d+1}$ using the QR factorization and backward-substitution method as

[3] Theoretically, we are not restricted from going to higher order local polynomials. We restrict to **LP1** for implementation in GPU due to memory limitations. In our forthcoming numerical examples, the GPU's global memory is limited to 6GB. Higher order polynomials would require not only more memory per hypercubes but also more memory for storing the regression coefficients, therefore we would be able to estimate over fewer hypercubes in parallel. Note that this is not an issue for parallel computing on CPUs.

follows:

$$R\alpha = Q^{\top}S, \qquad y_i^{(M)}(x(k)) = \mathcal{T}_{C_y}\left(\alpha_0 + \sum_{j=1}^{d} \alpha_j \times x_j(k)\right), \tag{5.22}$$

for any vector $(x(k) = (x_1(k), \ldots, x_d(k))$ in $\mathcal{H}_k$. By using the Householder reflection algorithm for computing the QR-factorization, there will be memory consumption of $O\left(M \times (d+1)\right)$ for the storage of the matrix A on each hypercube. This memory can be deallocated once the computation (5.22) is completed. We remark that the memory consumption is considerably lower than other alternative QR-factorization methods, as for example the Givens rotations method [61, Alg. 5.2.2], which requires a memory consumption $O(M^2)$ to store the matrix Q. This reduced memory consumption is instrumental in the GPU approach, as we explain in forthcoming Section 5.5.2.

5.4.2 Pseudo-algorithms for GPU

Algorithm 4 will be implemented on an NVIDIA GPU device. The device architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs); each multiprocessor is designed to execute hundreds of threads concurrently. To manage such a large amount of threads, it employs a unique architecture called SIMT (Single-Instruction, Multiple-Thread). The code execution unit is called a kernel and is executed simultaneously on all SMs by independent blocks of threads. Each thread is assigned to a single processor and executes within its own execution environment. Thus, all threads run the same instruction at a time, although over different data. In this section we briefly describe pseudo-codes for the Algorithm 4.

The algorithm has been programmed using the Compute Unified Device Architecture (CUDA) toolkit, specially designed for NVIDIA GPUs, see [109]. The code was built from an optimized C code. The below pseudo-algorithms reflect this programming feature. For the generation of the random numbers in parallel we took advantage of the NVIDIA CURAND library, see [110].

The time loop corresponding to the backward iteration of Algorithm 4 is shown in Listing 5.1; the kernel corresponds to the use of either the **LP0** or the **LP1** basis. In

Listing 5.2, a sketch for the **LP0** kernel is given. Notice that we are paralellizing the loop for any stratum index $k \in \{1, \ldots, K\}$ in the Algorithm 4; the terms $S_{Y,i}^{(M)}(\underline{\mathbf{x}}_i)$ and $S_{Z,i}^{(M)}(w, \underline{\mathbf{x}}_i)$ are computed in the `compute_responses_i` function, and the coefficients for $\psi_{Y,i,k}^{(M)}(\cdot)$ and $\psi_{Z,i,k}^{(M)}(\cdot)$ are computed in `compute_psi_Y` and `compute_psi_Z`, respectively, according to (5.21). Having in view an optimal performance, matrices storing the simulations, responses and regression coefficients are fully interleaved, thus allowing coalesced memory accesses, see [109]. Note that all device memory accesses are coalesced except those accesses to the regression coefficients in the resimulation stage during the computation of the responses, because one is not certain in which hypercube up each path will land. In Listing 5.3, the sketch for the **LP1** kernel is given. Additionally to the tasks of the kernel in Listing 5.2, each thread builds the matrix A and applies a QR factorization, as detailed in Section 5.4.1. Note that in addition to the matrices just explained in **LP0**, the matrix A is fully interleaved thus allowing fully coalesced accesses. The global memory is allocated at the beginning of the program and is freed at the end, thus allowing kernels to reuse already-allocated memory wherever possible. In addition to global memory, kernels are also using local memory, for example for storing the resimulated forward paths used for computing the responses. The coefficients for $\psi_{Y,i,k}^{(M)}(\cdot)$ and $\psi_{Z,i,k}^{(M)}(\cdot)$ are computed according to (5.22).

```
int i
curandState *devStates
Initialize devStates
Initialize n_blocks, n_threads_per_block
for(i=N-1; i>=0; i--)
  kernel_bsde<<<n_blocks,n_threads_per_block>>>(i, devStates, ...)
```

Listing 5.1: Backward iteration for $i = N - 1$ to $i = 0$.

```
__global__ void kernel_bsde_LP0(int i, curandState* devStates, ...) {
  const unsigned int global_tid = blockDim.x * blockIdx.x + threadIdx.x
  curandState localState = devStates[global_tid]

  unsigned long long int bin
  for(bin=global_tid; bin<K; bin+=n_blocks*n_threads_per_block) {
    simulates_x(&localState, global_tid, bin, ...)
```

```
    compute_responses_i(&localState, global_tid, i, ...)

    compute_psi_Z(global_tid, bin, i, ...)

    compute_psi_Y(global_tid, bin, i, ...)
  }
  devStates[global_tid] = localState
}
```

Listing 5.2: Kernel for the approximation with **LP0**.

```
__global__ void kernel_bsde_LP1(int i, curandState *devStates, ...) {
  const unsigned int global_tid = blockDim.x * blockIdx.x + threadIdx.x
  curandState localState = devStates[global_tid]

  unsigned long long int bin
  for(bin=global_tid;bin<K;bin+=n_blocks*n_threads_per_block) {
    simulates_x(&localState, global_tid, bin, ...)

    compute_responses_i(&localState, global_tid, i, ...)

    build_d_A(global_tid, d_A, ...)

    qr(global_tid, d_A, ...)

    compute_psi_Z(global_tid, bin, i, d_A, ...)

    compute_psi_Y(global_tid, bin, i, d_A, ...)
  }
  devStates[global_tid] = localState
}
```

Listing 5.3: Kernel for the approximation with **LP1**.

5.4.3 Theoretical complexity analysis

In this section, we assume that the functions $y_i(\cdot)$ and $z_i(\cdot)$ are smooth, namely globally Lipschitz (resp. C^1 and the first derivatives are globally Lipschitz) in the **LP0** (resp. **LP1**) case. The strata will be composed of uniform hypercubes of side length $\delta > 0$ in the domain $[-L, L]^d$, where $L = \log(N)/\mu$ and μ is the parameter

of the logistic distribution. This choice ensures $\nu\left(\mathbb{R}^d \backslash [-L, L]^d\right) \leq 2d \exp(-\mu L) = O(N^{-1})$. Our aim is to calibrate the numerical parameters (number of simulations and number of strata) so that the error given in Theorem 5.3.5 is $O(N^{-1})$, where N is the number of time-steps. This tolerance error is the one we usually obtain after time discretization with N time points [145, 57, 133]. In the following, we focus on polynomial dependency w.r.t. N, keeping only the highest degree, ignoring constants and $\log(N)$ terms.

Squared bias errors $T_{1,i}^Y$ and $T_{1,i}^Z$ in (5.9)-(5.10). First, we remark that the approximation error of the numerical scheme, namely the error due to basis selection, depends principally on the size δ of strata. In the case of **LP0**, the squared bias error is proportional to the squared hypercube diameter plus the tail contribution, i.e. $O\left(\delta^2 + \nu\left(\mathbb{R}^d \backslash [-L, L]^d\right)\right)$; to calibrate this bias to $O(N^{-1})$, we require $\delta = O(N^{-1/2})$. In contrast, the squared bias in $[-L, L]^d$ using **LP1** is proportional to the fourth power of the hypercube diameter, whence $\delta = O(N^{-1/4})$. As a result, ignoring the log terms the number of required hypercubes is

$$\textbf{LP0}: \quad K = O(N^{d/2}), \qquad\qquad \textbf{LP1}: \quad K = O(N^{d/4}),$$

in both cases.

Statistical and interdependence errors These error terms depend on the number of local polynomials, as well as on the number of simulations. Indeed, denoting $K' = \dim(\mathcal{L}_{Y \text{ or } Z,\cdot})$ the number of local polynomials and M the number of simulations in the hypercube, then both errors are dominated by $O\left(NK' \log(M)/M\right)$, where the factor N comes from the Z part of the solution (see $\mathcal{E}(i)$ in Theorem 5.3.5). For **LP0** (resp. **LP1**), $K' = 1$ or q (resp. $K' = d + 1$ or $q(d + 1)$). This implies to select

$$\textbf{LP0}: \quad M = O(N^2), \qquad\qquad \textbf{LP1}: \quad M = O(N^2),$$

again omitting the log terms.

Computational cost The computational cost (in flops) of the simulations per hypercube is equal to $O(M \times N)$, because we simulate M paths (of length N) of

the process X. The cost of the regression per hypercube is $O(M \times N)$, see Section 5.4.1, and thus equivalent to the simulation cost. Putting in the values of M from the last paragraph, the overall computational cost $\mathcal{C}_{\text{cost}}$ (summed over all hypercubes and time steps) is

$$\textbf{LP0}: \quad \mathcal{C}_{\text{cost}}^{\texttt{SEQ}} = O(N^{4+d/2}), \qquad \textbf{LP1}: \quad \mathcal{C}_{\text{cost}}^{\texttt{SEQ}} = O(N^{4+d/4}).$$

This quantity is related to the computational time for a sequential system (SEQ implementation) where there is no parallel computing. For the GPU implementation, described in Section 5.4.2, there is an additional computational time improvement since the computations on the hypercubes will be threaded across the cores of the card. Thus, the computational cost on GPU is

$$\textbf{LP0}: \quad \mathcal{C}_{\text{cost}}^{\texttt{GPU}} = O(N^{4+d/2})/\mathcal{C}_{\text{Load factor}}, \quad \textbf{LP1}: \quad \mathcal{C}_{\text{cost}}^{\texttt{GPU}} = O(N^{4+d/4})/\mathcal{C}_{\text{Load factor}}.$$

where the load factor $\mathcal{C}_{\text{Load factor}}$ is ideally the number of threads on the device.

Finally, we quantify the improvement in memory consumption offered by the SRMDP algorithm compared to the LSMDP algorithm of [60]. This is a very important improvement, because, as explained in the introduction, the memory is the key constraint in solving problems in high dimension. We only compare sequential versions of the algorithms, meaning that the computational costs will be the same. The main difference between the two schemes is then in the number of simulations that must be stored in the machine at any given time. We summarize this in Table 5.1 below.

Algorithm	Number of simulations		Computational cost	
	LP0	**LP1**	**LP0**	**LP1**
SRMDP	N^2	N^2	$N^{4+d/2}$	$N^{4+d/4}$
LSMDP	$N^{2+d/2}$	$N^{2+d/4}$	$N^{4+d/2}$	$N^{4+d/4}$

Table 5.1: Comparison of numerical parameters with or without stratified sampling, as a function of N.

In SRMDP, the memory consumption is mainly related to storing coefficients representing the solutions on hypercubes, that is $O(N \times \dim(\mathcal{L}_{Y \text{ or } Z,\cdot}) \times K)$; if one

is using the **LP1** basis, one must also take into account the memory consumption per strata $M \times (d+1) = O(N^2)$ for the QR factorization, explained in Section 5.4.1. In contrast, the memory consumption for LSMDP is mainly $O(K \times N^2)$, which represents the number of simulated paths of the Markov chains that must be stored in the machine at any given time. We summarize the memory consumption of the two algorithms in Table 5.2.

Algorithm	**LP0**	**LP1**
SRMDP	$N^{1+d/2}$	$N^{1+d/4} \vee N^2$
LSMDP	$N^{2+d/2}$	$N^{2+d/4}$

Table 5.2: Comparison of memory requirement as a function of N.

Observe that SRMDP requires N times less memory than LSMDP with the **LP0** basis. This implicitly implies a gain of 2 on the dimension d that can be handled. On the other hand, if the **LP1** basis is used, the SRMDP requires $O(N^{d/4})$ less memory for $d \leq 4$ than LSMDP, and N times less memory for $d \geq 4$. Therefore, there is an implicit gain of 4 in the dimension that can be handled by the algorithm.

5.5 Numerical experiments

5.5.1 Model, stratification, and performance benchmark

We use the Brownian motion model $X = W$ $(d = q)$. Moreover, the numerical experiments will consider the performance according to the dimension d. We introduce the function $\omega(t,x) = \exp(t + \sum_{k=1}^{q} x_k)$. We perform numerical experiments on the BSDE with data $g(x) = \omega(T,x)(1+\omega(T,x))^{-1}$ and

$$f(t,x,y,z) = \left(\sum_{k=1}^{q} z_k\right)\left(y - \frac{2+q}{2q}\right),$$

where $z = (z_1, \ldots, z_q)$. The BSDE has explicit solutions in this framework, given by

$$y_i(x) = \omega(t_i,x)(1+\omega(t_i,x))^{-1}, \qquad z_{k,i}(x) = \omega(t_i,x)(1+\omega(t_i,x))^{-2},$$

where $z_{k,i}(x)$ is the k-th component of the q-dimensional cylindrical function $z_i(x) \in \mathbb{R}^q$.

The logistic distribution for Algorithm 4 is parameterized by $\mu = 1$ and we consider $T = 1$. For the least-squares Monte Carlo, we stratify the domain $[-6.5, 6.5]^q$ with uniform hypercubes. To assess the performance of the algorithm, we compute the average mean squared error (MSE) over 10^3 independent runs of the algorithm for three error indicators:

$$MSE_{Y,\max} := \ln\left\{10^{-3}\max_{0\le i\le N-1}\sum_{m=1}^{10^3}|y_i(R_{i,m}) - y_i^{(M)}(R_{i,m})|^2\right\},$$

$$MSE_{Y,\text{av}} := \ln\left\{10^{-3}N^{-1}\sum_{m=1}^{10^3}\sum_{i=0}^{N-1}|y_i(R_{i,m}) - y_i^{(M)}(R_{i,m})|^2\right\},$$

$$MSE_{Z,\text{av}} := \ln\left\{10^{-3}N^{-1}\sum_{m=1}^{10^3}\sum_{i=0}^{N-1}|z_i(R_{i,m}) - z_i^{(M)}(R_{i,m})|^2\right\},$$

where the simulations $\{R_{i,m}; i = 0, \ldots, N-1,\ m = 1, \ldots, 10^3\}$ are independent and identically ν-distributed, and independently drawn from the simulations used for the LSMC scheme. We parameterize the hypercubes according to the instructions given in the theoretical complexity analysis, see Section 5.4.3. In particular, we consider different values of N and always set $K = O(N^{d/2})$ in **LP0** (resp. $O(N^{d/4})$ in **LP1**) and $M = O(N^2)$. Note, however, that we do not specify the value of δ, but rather the number of hypercubes per dimension $K^{1/q}$, which we denote `#C` in what follows; this being equivalent to setting δ, but is more convenient to program. As we shall illustrate, the error converges as predicted as N increases, although the exact error values will depend on the constants that we choose in the parameterization of K and M.

5.5.2 CPU and GPU performance

In this section, several experiments based on Section 5.5.1 are presented to assess the performance of CUDA implementation of Algorithm 4; the pseudo-algorithms are

given in Section 5.4.2. We shall compare its performance with a version of SRMDP implemented to run on multicore CPUs. For the design of this comparison we have followed some ideas in [97]. Moreover, in order to test the theoretical results of Section 5.4.3, we compare the performance of the two algorithms according to the choice of the basis functions, the impact of this choice on the convergence of the approximation of the BSDE, and the impact of this choice on the computational performance in terms of computational time and memory consumption.

There are two types of basis functions we investigate: **LP0** in Section 5.5.2, and **LP1** in Section 5.5.2. As explained in Section 5.4.3, the **LP0** basis is highly suited to GPU implementation because it has a very low memory requirement per thread of computation. On the other hand, it has a very high global memory requirement for storing coefficients. This represents a problem in high dimensions because one needs many coefficients to obtain a good accuracy. On the other hand, the **LP1** basis involves a higher cost per thread, although requires a far lower global memory for storing coefficients; this implies that the impact of the GPU implementation is lower in moderate dimensional problems, but that one can solve problems in higher dimension. Moreover, the full performance impact of the GPU implementation on the **LP1** basis is in high dimension, where the number of strata is very high and therefore the GPU is better saturated with computations. We illustrate numerically all of these effects in the following sections.

The numerical experiments have been performed with the following hardware and software configurations: a GPU GeForce GTX TITAN Black with 6 GBytes of global memory (see [111] for details in the architecture), two recent multicore Intel Xeon CPUs E5-2620 v2 clocked at 2.10 GHz (6 cores per socket) with 62 GBytes of RAM, CentOS Linux, NVIDIA CUDA SDK 7.5 and INTEL C compiler 15.0.6. The CPU programs were optimized and parallelized using OpenMP [149]. Since the CUDA code has been derived from an optimized C code, both codes perform the same algorithms, and their performance can be fairly compared according to computational times; the multicore CPUs time (CPU) and the GPU time (GPU) will all be measured in seconds

in the forthcoming tables. CPU times correspond to executions using 24 threads so as to take advantage of Intel Hyperthreading. The speedups of the CPU parallel version with respect to pure sequential CPU code are around 16. The results are obtained in single precision, both in CPU and GPU.

Examples with the approximation with LP0 local polynomials

All examples will be run using 64 thread blocks each with 256 threads. In Table 5.3 we show results for $d = 4$, with `#C`$=\left\lfloor 4\sqrt{N} \right\rfloor$ and $M = N^2$. Except for the cases $\Delta_t = 0.2$ and $\Delta_t = 0.1$ where there are not enough strata to fully take advantage of the GPU, the GPU implementation provides a significant reduction in the computational time: the GPU speedup reaches the value 18.90. Moreover, the speedup improves as we increase the `#C`.

Δ_t	`#C`	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	8	4096	25	−3.712973	−3.774071	−0.964842	0.23	2.00
0.1	12	20736	100	−4.066741	−4.303750	−1.607104	5.23	2.20
0.05	17	83521	400	−4.337988	−4.698645	−2.302092	171.92	12.39
0.02	28	614656	2500	−4.472564	−4.988069	−3.225411	58066.33	3070.92

Table 5.3: **LP0** local polynomials, $d = 4$, `#C`$=\left\lfloor 4\sqrt{N} \right\rfloor$, $M = N^2$.

Tables 5.4 and 5.5 show results for $d = 6$ with `#C`$=\left\lfloor \sqrt{N} \right\rfloor$ and `#C`$=\left\lfloor 2\sqrt{N} \right\rfloor$, respectively. Convergence is clearly improved by doubling `#C`. In Table 5.5 the case of $\Delta_t = 0.02$ is not shown due to insufficient GPU global memory.[4] In Table 5.4, the GPU speedup reaches 15.93, whereas in Table 5.5 it reaches 14.85. As in Table 5.3, the increase in the speedup is explained due to the increased number of hypercubes, thus demonstrating how important it is to have many hypercubes in the GPU implementation. However, the finer basis requires 2^6 times as much memory for storing coefficients.

[4]For $\Delta_t = 0.02$, the array for storing the regression coefficients will be of size $N \times K \times (D + 1)$, i.e. $50 \times 14^6 \times 7 \times 4$ bytes using single precision, which equals 9.81 GBytes, much greater than the available 6 GBytes of device memory.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	64	25	-2.392320	-2.451332	-0.431059	0.21	1.99
0.1	3	729	100	-2.440274	-2.500775	-1.096603	0.47	2.05
0.05	4	4096	400	-2.829757	-2.905192	-1.687142	17.21	3.15
0.02	7	117649	2500	-3.235130	-3.539011	-2.557686	13930.70	874.25

Table 5.4: **LP0** local polynomials, $d = 6$, #C=$\left\lfloor \sqrt{N} \right\rfloor$, $M = N^2$.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	4	4096	25	-2.707882	-2.784022	-0.477751	0.29	1.94
0.1	6	46656	100	-3.195937	-3.294488	-1.133834	13.72	2.44
0.05	8	262144	400	-3.505867	-3.664396	-1.795697	775.33	52.20

Table 5.5: **LP0** local polynomials, $d = 6$, #C=$\left\lfloor 2\sqrt{N} \right\rfloor$, $M = N^2$.

Table 5.6 shows that the algorithm can work for $d = 11$ in several seconds with a reasonable accuracy in a GPU. The corresponding speedup with respect to CPU version is around 13.35. For the execution with $\Delta_t = 0.1$ we are going to report the GFlop rate, and also the memory transfer to/from the global memory. Inside the kernel, the functions computing the regression coefficients (denoted by `compute_psi_Z` and `compute_psi_Y` in the Listing 5.2) are memory bounded, reaching 236.795 GBytes/s when reading/writing from/to global memory. The rest of the kernel is more computationally limited. In the overall kernel, the memory transfer from/to global memory is around 160 GBytes/s and the Gflop rate is around 238 GFlop/s, although around the 30% of the instructions executed by the kernel are integer instructions to assign the simulations to the strata in the resimulation stage during the computation of the responses.

Examples with the approximation with LP1 local polynomials

In this section we show the results corresponding to the approximation with the **LP1** basis. Compared to **LP0**, this basis consumes much less global memory to store

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	2048	25	-2.152253	-2.202357	0.211590	0.27	1.99
0.1	3	177147	100	-2.144843	-2.267742	-0.469759	67.96	6.29
0.05	4	4194304	400	-2.484169	-2.633602	-1.070096	28154.07	2108.64

Table 5.6: **LP0** local polynomials, $d = 11$, #C$=\left\lfloor \sqrt{N} \right\rfloor$, $M = N^2$.

coefficients, because it requires far fewer hypercubes, see Section 5.4.3. On the other hand, the approximation with **LP1** basis demands higher thread memory due to the storage of a large matrix for each hypercube, as explained in Section 5.4.3. This may have an impact on the computational time on the GPU: recalling from Section 5.4.2 the GPU handles multiple hypercubes at any given time, each one requiring the storage of a matrix A, the global memory capacity of the GPU device restricts the number of threads we can handle at any given time. This issue is much less significant with the **LP0** basis. In order to optimize the performance of the **LP1** basis, we must minimize the thread memory storage. We implement the Householder reflection method for QR-factorization, [61, Alg. 5.3.2]. For this, we must store a matrix containing $M \times (d+1) = O(N^2)$ floating point values per thread on the GPU memory. Thanks to the reduced global memory storage for coefficients, we are able to work in a rather high dimension $d = 19$.

Remark 5.5.1. *There are many methods to implement QR-factorization. However, the choice of method has a substantial impact on the performance of the GPU implementation. For example, the Givens rotation method [61, Alg. 5.2.2] requires the storage of an $M \times M$ matrix, which corresponds to $O(N^4)$ floating points. This is rather more than the required $O(N^2)$ for the Householder reflection method given in Section 5.4.1. Therefore, the Givens rotation method would be far slower when implemented on a GPU than the Householder reflection method, because it may not be possible to use an optimal thread configuration.*

Remark 5.5.2. *In the forthcoming examples, we use more simulations per stratum*

*for the **LP1** basis compared to the equivalent results for **LP0**. This is to account for the additional statistical and interdependence errors, as explained in Section 5.4.3.*

In Table 5.7, we present results for $d = 4$. These results are to be compared with Table 5.3, where in particular the $MSE_{Z,\text{av}}$ results are closer line to line. The computational time is substantially improved for the CPU and GPU calculations. Also note that, unlike for the Z component, the accuracy for the Y component is substantially better for the **LP1** basis than for the **LP0** one. The difference in the accuracy results between the Y and Z components is likely explained by the fact that the function $x \mapsto z_i(x)$ is rather flat, so it is much better approximated by **LP0** basis functions than $x \mapsto y_i(x)$. The GPU speedup reaches 8.05.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	3	81	125	−4.021483	−4.131725	−0.900286	0.11	0.23
0.1	5	625	500	−4.290881	−4.695769	−1.551480	1.26	0.79
0.05	7	2401	2000	−4.541253	−5.022405	−2.281332	43.56	7.83
0.02	10	10000	12500	−4.574551	−5.143310	−3.228237	6827.98	847.83

Table 5.7: **LP1** local polynomials, $d = 4$, #C$= \left\lfloor 3\sqrt{d\sqrt{N}} - 5 \right\rfloor$, $M = (d+1)N^2$.

Next, results for $d = 6$ are shown. Thus, we compare Table 5.8 below with Table 5.5. For a given precision on the Z component of the solution, we observe substantial improvements in the CPU codes, but no such gains on the GPU version. In contrast, the accuracy of the Y approximation is, as in the $d = 4$ case, substantially better. Moreover, whereas we were not able to do computations for $\Delta_t = 0.02$ with the **LP0** basis due to insufficient GPU memory, we are now able to make these calculations with the **LP1** basis. The GPU speedup reaches 6.13, which is lower than the **LP0** basis speedup factor, as expected.

In the high dimensional $d = 11$ setting shown in Table 5.9, we compare with Table 5.6. We observe a speedup of order 5.63 compared to the CPU implementation.

In the remainder of this section, we present results in dimension $d = 12$ to $d = 19$ (in Tables 5.10, 5.11, 5.12 and 5.13, respectively) for which the capacity of the GPU is

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	64	175	-3.504153	-3.668801	-0.461077	0.20	0.32
0.1	3	729	700	-3.804091	-3.911488	-1.133263	1.84	1.66
0.05	4	4096	2800	-4.075928	-4.231639	-1.791519	125.81	20.50
0.02	6	46656	17500	-3.809734	-4.529827	-2.689432	82529.21	15283.18

Table 5.8: **LP1** local polynomials, $d = 6$, #C$=\left\lfloor 1.5\sqrt{d\sqrt{N}} - 3 \right\rfloor$, $M = (d+1)N^2$.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	2048	2000	-3.271648	-3.368051	-1.455388	10.33	3.41
0.2	3	177147	4000	-3.269004	-3.403994	-1.975300	1635.95	290.56

Table 5.9: **LP1** local polynomials, $d = 11$.

maximally used to provide the highest possible accuracy. The GPU speedup reaches up to 5.67 compared to the CPU implementation. Note that for the example with $d = 19$ in Table 5.13 the LSMDP algorithm would require 118 GBytes of memory to store all the simulations at a given time, whereas the here proposed SRMDP algorithm can be executed with less than 6 GBytes and with much less computational time owing to it does not need to associate the simulations to hypercubes. Finally, for the example with $\Delta_t = 0.2$, #C$= 2$ and $M = 4000$ of Table 5.11 we next report the GFlop rate and the memory transfer from/to global memory. In the overall kernel, the memory transfer from/to global memory is around 132 GBytes/s and the GFlop rate is around 136 GFlop/s. In order to understand why the device memory bandwidth used by LP1 kernel is lower than the one used by LP0 kernel, observe that at any given time, for each strata we are accessing $(d + 1)$ times more elements in the LP1 framework in the re-simulation stage of the responses computation. Moreover, these accesses are potentially non-coalesced, because the forward process is randomly re-simulated and we do not know a priori in which strata is going to fall.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	4096	2000	−3.111153	−3.232051	−1.297737	22.29	4.95
0.2	3	531441	4000	−3.214096	−3.272644	−1.821935	5554.49	1196.28

Table 5.10: **LP1** local polynomials, $d = 12$.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	8192	3000	−2.995413	−3.153302	−1.460911	69.45	12.46
0.2	2	8192	4000	−3.022855	−3.158471	−1.649632	94.07	16.58

Table 5.11: **LP1** local polynomials, $d = 13$.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	16384	2000	−3.011673	−3.092870	−1.026128	102.11	19.55
0.2	2	16384	4000	−3.029663	−3.105833	−1.558935	205.82	50.62

Table 5.12: **LP1** local polynomials, $d = 14$.

d	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
15	32768	5000	−2.981181	−3.106590	−1.574532	578.88	139.60
16	65536	6000	−2.795353	−2.959375	−1.588716	1411.75	429.53
17	131072	5000	−2.772595	−2.936549	−1.371146	2580.06	793.61
18	262144	4000	−2.845755	−2.918057	−1.114600	4275.13	1589.30
19	524288	3200	−2.726427	−2.851617	−0.839849	7245.91	4370.31

Table 5.13: **LP1** local polynomials, $d = 15, \ldots, 19$, $\Delta_t = 0.2$, #C $= 2$.

Appendix A

Test functions for the Simulated Annealing

In this appendix we present the expressions of the functions in our test problem suite for the implemented Simulated Annealing algorithm (see Table 1.8).

1. **Test 1** (*Ackley problem*):

   Originally the Ackley's problem (see [2]) was defined for two dimensions, but the problem has been generalized to n dimensions [24].

   Formally, this problem can be described as finding a point $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, with $x_i \in [-30, 30]$, that minimizes the following equation:

   $$f(\boldsymbol{x}) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e.$$

   The minimum of the Ackley's function is located at the origin with $f(\mathbf{0}) = 0$. This test was performed for $n = 30$, $n = 100$, $n = 200$ and $n = 400$.

2. **Test 2** (*Branin problem*):

   The expression of the Branin function (see [35]) is,

$$f(\boldsymbol{x}) = \left(x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos(x_1) + 10,$$

with x_1, $x_2 \in [-20, 20]$. The minimum of the objective function value is equal to $f(\boldsymbol{x}^\star) = 0.397887$, and it is located at the following three points: $\boldsymbol{x}^\star = (-\pi, 12.275)$, $\boldsymbol{x}^\star = (\pi, 2.275)$ and $\boldsymbol{x}^\star = (9.425, 2.475)$.

3. **Test 3** (*Cosine mixture problem*):

   The expression of this function is (see [17]):

$$f(\boldsymbol{x}) = -0.1 \sum_{i=1}^{n} \cos(5\pi x_i) - \sum_{i=1}^{n} x_i^2,$$

   with $x_i \in [-1, 1]$, $i = 1, 2 \ldots, n$. The global minimum is located at the origin with the function values -0.2 and -0.4 for $n = 2$ and $n = 4$, respectively.

4. **Test 4** (*Dekkers and Aarts problem*):

   The Dekkers and Aarts function (see [33]) has the following expression

$$f(\boldsymbol{x}) = 10^5 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4,$$

   with x_1, $x_2 \in [-20, 20]$. This function has more than three local minima, but there are two global minima located at $\boldsymbol{x}^\star = (0, -14.945)$ and $\boldsymbol{x}^\star = (0, 14.945)$ with $f(\boldsymbol{x}^\star) = -24776.518$.

5. **Test 5** (*Easom problem*):

   The Easom function (see [107]) has the following definition

$$f(\boldsymbol{x}) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2),$$

   where the considered search space is x_1, $x_2 \in [-10, 10]$. The minimum value is located at $\boldsymbol{x}^\star = (\pi, \pi)$ with $f(\boldsymbol{x}^\star) = -1$.

6. **Test 6** (*Exponential problem*):

   The definition of the Exponential problem (see [17]) is the following

$$f(\boldsymbol{x}) = -\exp\left(-0.5\sum_{i=1}^{n} x_i^2\right),$$

   with $x_i \in [-1, 1]$, $i = 1, \ldots, n$. The optimal objective function value is $f(\boldsymbol{x}^\star) = -1$, and it is located at the origin. In our tests we consider $n = 4$.

7. **Test 7** (*Goldstein and Price problem*):

   The Goldstein and Price function (see [35]) has the following definition,

$$\begin{aligned} f(x) = \; & [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2) + 3x_2^2] \\ & \times[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)], \end{aligned}$$

   with x_1, $x_2 \in [-2, 2]$. There are four local minima and the global minimum is located at $\boldsymbol{x}^\star = (0, -1)$ with $f(\boldsymbol{x}^\star) = 3$.

8. **Test 8** (*Griewank problem*):

   The Griewank function (proposed in [65]) is defined as follows,

$$f(\boldsymbol{x}) = 1 + \sum_{i=1}^{n}\left[\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)\right],$$

   where $x_i \in [-600, 600]$, $i = 1, \ldots, n$. The global minimum is located at the origin and its function value is 0; moreover the function has also a very large number of local minima, exponentially increasing with n (in the two dimensional case there are around 500 local minima). Tests were performed for $n = 100$, $n = 200$ and $n = 400$.

9. **Test 9** (*Himmelblau problem*):

   The expression of the Himmelblau's function (see [74]) is the following

$$f(\boldsymbol{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2,$$

where $x_1,\ x_2 \in [-6, 6]$. The global minima is located at the following four points $\boldsymbol{x}^\star = (3.0, 2.0)$, $\boldsymbol{x}^\star = (-2.805118, 3.131312)$, $\boldsymbol{x}^\star = (-3.779310, -3.283186)$ and $\boldsymbol{x}^\star = (3.584428, -1.848126)$, with $f(\boldsymbol{x}^\star) = 0$.

10. **Test 10** (*Levy and Montalvo problem*):

    The expression of the Levy and Montalvo function (see [99]) is,

$$f(\boldsymbol{x}) = \frac{\pi}{n}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left(1 + 10\sin^2(\pi y_{i+1})\right) + (y_n - 1)^2\right),$$

    where $y_i = 1 + \frac{1}{4}(x_i + 1)$ for $x_i \in [-10, 10]$, $i = 1, \ldots, n$. This function has approximately 5^n local minima and the global minimum is located at the point $\boldsymbol{x}^\star = (-1, \ldots, -1)$ with $f(\boldsymbol{x}^\star) = 0$. Tests were performed for $n = 2$, $n = 5$ and $n = 10$.

11. **Test 11** (*Modified Langerman problem*):

    The expression of the Modified Langerman function (see [11]) is,

$$f(\boldsymbol{x}) = -\sum_{i=1}^{5} c_i \left[\exp\left(-\frac{1}{\pi}\sum_{j=1}^{n}(x_j - a_{ij})^2\right)\cos\left(\pi\sum_{j=1}^{n}(x_j - a_{ij})^2\right)\right],$$

    where $x_i \in [0, 10]$, $i = 1, \ldots, n$ and

$$\boldsymbol{A} = (a_{ij}) = \begin{pmatrix} 9.681 & 0.667 & 4.783 & 9.095 & 3.517 & 9.325 & 6.544 & 0.211 & 5.122 & 2.020 \\ 9.400 & 2.041 & 3.788 & 7.931 & 2.882 & 2.672 & 3.568 & 1.284 & 7.033 & 7.374 \\ 8.025 & 9.152 & 5.114 & 7.621 & 4.564 & 4.711 & 2.996 & 6.126 & 0.734 & 4.982 \\ 2.196 & 0.415 & 5.649 & 6.979 & 9.510 & 9.166 & 6.304 & 6.054 & 9.377 & 1.426 \\ 8.074 & 8.777 & 3.467 & 1.863 & 6.708 & 6.349 & 4.534 & 0.276 & 7.633 & 1.567 \end{pmatrix},$$

$$\boldsymbol{c} = (c_i) = \begin{pmatrix} 0.806 & 0.517 & 0.100 & 0.908 & 0.965 \end{pmatrix}.$$

In this case it is unknown the number of local minima. The global optimum when $n = 2$ is searched at $\boldsymbol{x}^\star = (9.6810707, 0.6666515)$ with $f(\boldsymbol{x}^\star) = -1.080938$, and for $n = 5$ the global minimun is located at $\boldsymbol{x}^\star = (8.074000, 8.777001, 3.467004, 1.863013, 6.707995)$ with $f(\boldsymbol{x}^\star) = -0.964999$.

12. **Test 12** (*Michalewicz problem*):

    The definition of the Michalewicz function (see [107]) is the following,

$$f(\boldsymbol{x}) = -\sum_{i=1}^{n} \sin(x_i) \left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2m},$$

    where $x_i \in [0, \pi]$, $i = 1, \ldots, n$. It is usual to set $m = 10$. The objective function value at the global minimum is $f(\boldsymbol{x}^\star) = -1.8013$ for $n = 2$, $f(\boldsymbol{x}^\star) = -4.6877$ for $n = 5$, and $f(\boldsymbol{x}^\star) = -9.6602$ for $n = 10$.

13. **Test 13** (*Rastrigin problem*):

    The expression of the Rastrigin function (see [129] and [134], for example) has the following definition,

$$f(\boldsymbol{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right),$$

    where $x_i \in [-5.12, 5.12]$, $i = 1, \ldots, n$. The global minimum is located at $\boldsymbol{x}^\star = (0, \ldots, 0)$ and the objective function at this point is $f(\boldsymbol{x}^\star) = 0$. In our tests we consider $n = 100$ and $n = 400$.

14. **Test 14** (*Generalized Rosenbrock problem*):

    The Rosenbrock's function (see [81]), also known as Rosenbrock valley, banana function or the *second function of De Jong*, has the following expression,

$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i)^2 + (1 - x_i)^2\right],$$

with $x_i \in [-2.048, 2.048]$, $i = 1, \ldots, n$. The global minimum is located at $\boldsymbol{x}^\star = (1, \ldots, 1)$ with the function value $f(\boldsymbol{x}^\star) = 0$. In our tests we consider $n = 4$.

15. **Test 15** (*Salomon problem*):

    The Salomon function (see [125]) has the following definition,

$$f(\boldsymbol{x}) = 1 - \cos(2\pi||\boldsymbol{x}||_2) + 0.1||\boldsymbol{x}||_2,$$

    where $||\boldsymbol{x}||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$, and $x_i \in [-100, 100]$, $i = 1, \ldots, n$. In the general case (n) the number of local minima is not known. This function has a global minima located at $\boldsymbol{x}^\star = (0, \ldots, 0)$ with $f(\boldsymbol{x}^\star) = 0$. For our tests we consider $n = 10$.

16. **Test 16** (*Six-Hump Camel Back problem*):

    The expression of the Six-Hump Camel Back function (see [35]) is the following,

$$f(\boldsymbol{x}) = \left(4 - 2.1x_1^2 + \frac{1}{3}x_1^4\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2,$$

    with $x_1 \in [-3, 3]$ and $x_2 \in [-2, 2]$. This function has two global minima equal to $f(\boldsymbol{x}^\star) = -1.0316$, located at $\boldsymbol{x}^\star = (-0.0898, 0.7126)$ and $\boldsymbol{x}^\star = (0.0898, -0.7126)$.

17. **Test 17** (*Shubert problem*):

    The Shubert function (see [99]) has the following definition

$$f(\boldsymbol{x}) = \prod_{i=1}^{n}\left(\sum_{j=1}^{5} j\cos((j+1)x_i + j)\right),$$

    subject $x_i \in [-10, 10]$, $i = 1, \ldots, n$. For the n-dimensional case the number of local minima is unknown, however for $n = 2$, the function has 760 local minima, where 18 of them are global with $f(\boldsymbol{x}^\star) \approx -186.7309$. We have performed the tests for $n = 2$. For this case, the global optimizers are $(-7.0835, 4.8580)$,

$(-7.0835, -7.7083)$, $(-1.4251, -7.0835)$, $(5.4828, 4.8580)$, $(-1.4251, -0.8003)$, $(4.8580, 5.4828)$, $(-7.7083, -7.0835)$, $(-7.0835, -1.4251)$, $(-7.7083, -0.8003)$, $(-7.7083, 5.4828)$, $(-0.8003, -7.7083)$, $(-0.8003, -1.4251)$, $(-0.8003, 4.8580)$, $(-1.4251, 5.4828)$, $(5.4828, -7.7083)$, $(4.8580, -7.0835)$, $(5.4828, -1.4251)$ and $(4.850, -0.8003)$.

18. **Test 18** (*Shekel problem*):

    The expression of the Shekel function (see [11]) is

$$f(\boldsymbol{x}) = -\sum_{i=1}^{m} \frac{1}{\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i},$$

    where the matrix $A = (a_{ij})$ and the vector $\boldsymbol{c} = (c_i)$ are the following,

$$\boldsymbol{c} = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}, \quad A = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix}.$$

    The search domain is $x_i \in [0, 10]$, $i = 1, \ldots, 4$. The global optimum is $\boldsymbol{x}^\star = (4, 4, 4, 4)$ and the function value at this point is $f(\boldsymbol{x}^\star) = -10.1532$ for $m = 5$, $f(\boldsymbol{x}^\star) = -10.4029$ for $m = 7$ and $f(\boldsymbol{x}^\star) = -10.5364$ for $m = 10$.

19. **Test 19** (*Modified Shekel Foxholes problem*):

    The expression of the Modified Shekel Foxholes function (see [11]) is

$$f(\boldsymbol{x}) = -\sum_{i=1}^{30} \frac{1}{\sum_{j=1}^{n}(x_j - a_{ij})^2 + c_i},$$

where the matrix $A = (a_{ij})$ and the vector $\boldsymbol{c} = (c_i)$ are the following,

$$\boldsymbol{c} = \begin{pmatrix} 0.806 \\ 0.517 \\ 0.100 \\ 0.908 \\ 0.965 \\ 0.669 \\ 0.524 \\ 0.902 \\ 0.531 \\ 0.876 \\ 0.462 \\ 0.491 \\ 0.463 \\ 0.714 \\ 0.352 \\ 0.869 \\ 0.813 \\ 0.811 \\ 0.828 \\ 0.964 \\ 0.789 \\ 0.360 \\ 0.369 \\ 0.992 \\ 0.332 \\ 0.817 \\ 0.632 \\ 0.883 \\ 0.608 \\ 0.326 \end{pmatrix}, \quad A = \begin{pmatrix} 9.681 & 0.667 & 4.783 & 9.095 & 3.517 & 9.325 & 6.544 & 0.211 & 5.122 & 2.020 \\ 9.400 & 2.041 & 3.788 & 7.931 & 2.882 & 2.672 & 3.568 & 1.284 & 7.033 & 7.374 \\ 8.025 & 9.152 & 5.114 & 7.621 & 4.564 & 4.711 & 2.996 & 6.126 & 0.734 & 4.982 \\ 2.196 & 0.415 & 5.649 & 6.979 & 9.510 & 9.166 & 6.304 & 6.054 & 9.377 & 1.426 \\ 8.074 & 8.777 & 3.467 & 1.863 & 6.708 & 6.349 & 4.534 & 0.276 & 7.633 & 1.567 \\ 7.650 & 5.658 & 0.720 & 2.764 & 3.278 & 5.283 & 7.474 & 6.274 & 1.409 & 8.208 \\ 1.256 & 3.605 & 8.623 & 6.905 & 4.584 & 8.133 & 6.071 & 6.888 & 4.187 & 5.448 \\ 8.314 & 2.261 & 4.224 & 1.781 & 4.124 & 0.932 & 8.129 & 8.658 & 1.208 & 5.762 \\ 0.226 & 8.858 & 1.420 & 0.945 & 1.622 & 4.698 & 6.228 & 9.096 & 0.972 & 7.637 \\ 7.305 & 2.228 & 1.242 & 5.928 & 9.133 & 1.826 & 4.060 & 5.204 & 8.713 & 8.247 \\ 0.652 & 7.027 & 0.508 & 4.876 & 8.807 & 4.632 & 5.808 & 6.937 & 3.291 & 7.016 \\ 2.699 & 3.516 & 5.874 & 4.119 & 4.461 & 7.496 & 8.817 & 0.690 & 6.593 & 9.789 \\ 8.327 & 3.897 & 2.017 & 9.570 & 9.825 & 1.150 & 1.395 & 3.885 & 6.354 & 0.109 \\ 2.132 & 7.006 & 7.136 & 2.641 & 1.882 & 5.943 & 7.273 & 7.691 & 2.880 & 0.564 \\ 4.707 & 5.579 & 4.080 & 0.581 & 9.698 & 8.542 & 8.077 & 8.515 & 9.231 & 4.670 \\ 8.304 & 7.559 & 8.567 & 0.322 & 7.128 & 8.392 & 1.472 & 8.524 & 2.277 & 7.826 \\ 8.632 & 4.409 & 4.832 & 5.768 & 7.050 & 6.715 & 1.711 & 4.323 & 4.405 & 4.591 \\ 4.887 & 9.112 & 0.170 & 8.967 & 9.693 & 9.867 & 7.508 & 7.770 & 8.382 & 6.740 \\ 2.440 & 6.686 & 4.299 & 1.007 & 7.008 & 1.427 & 9.398 & 8.480 & 9.950 & 1.675 \\ 6.306 & 8.583 & 6.084 & 1.138 & 4.350 & 3.134 & 7.853 & 6.061 & 7.457 & 2.258 \\ 0.652 & 2.343 & 1.370 & 0.821 & 1.310 & 1.063 & 0.689 & 8.819 & 8.833 & 9.070 \\ 5.558 & 1.272 & 5.756 & 9.857 & 2.279 & 2.764 & 1.284 & 1.677 & 1.244 & 1.234 \\ 3.352 & 7.549 & 9.817 & 9.437 & 8.687 & 4.167 & 2.570 & 6.540 & 0.228 & 0.027 \\ 8.798 & 0.880 & 2.370 & 0.168 & 1.701 & 3.680 & 1.231 & 2.390 & 2.499 & 0.064 \\ 1.460 & 8.057 & 1.336 & 7.217 & 7.914 & 3.615 & 9.981 & 9.198 & 5.292 & 1.224 \\ 0.432 & 8.645 & 8.774 & 0.249 & 8.081 & 7.461 & 4.416 & 0.652 & 4.002 & 4.644 \\ 0.679 & 2.800 & 5.523 & 3.049 & 2.968 & 7.225 & 6.730 & 4.199 & 9.614 & 9.229 \\ 4.263 & 1.074 & 7.286 & 5.599 & 8.291 & 5.200 & 9.214 & 8.272 & 4.398 & 4.506 \\ 9.496 & 4.830 & 3.150 & 8.270 & 5.079 & 1.231 & 5.731 & 9.494 & 1.883 & 9.732 \\ 4.138 & 2.562 & 2.532 & 9.661 & 5.611 & 5.500 & 6.886 & 2.341 & 9.699 & 6.500 \end{pmatrix}.$$

The search domain is $x_i \in [-5, 15]$. For this function the number of local minima is unknown. For $n = 2$ the global minimum is located at the point $\boldsymbol{x}^\star = (8.024, 9.146)$ with $f(\boldsymbol{x}^\star) = -12.1190$. For $n = 5$ the global minima is $\boldsymbol{x}^\star = (8.025, 9.152, 5.114, 7.621, 4.564)$ with $f(\boldsymbol{x}^\star) = -10.4056$.

Appendix B

SABR equity

In this appendix we present the expressions of the implied volatility in the general case and the market data employed in Chapter 2.

B.1 Expression of implied volatility in the general case

For the **Case II** of Section 2.2.2 (general case), using Mathematica, the functions ν_1^2, ν_2^2, η_1 and η_2^2 given by (2.10) have the following expressions:

$$\begin{aligned}
\nu_1^2(T) = \; & \frac{1}{4b^5T^3}\Bigg[9q_\nu^2 + 6b^4\nu_0(4d_\nu+\nu_0)T^2 + 4b^5d_\nu^2T^3 + 9bq_\nu(16d_\nu+\nu_0-q_\nu T) \\
& +6b^3T(-\nu_0(8d_\nu+\nu_0)+(4d_\nu+\nu_0)q_\nu T) + 3b^2\left(\nu_0(16d_\nu+\nu_0) - 4(8d_\nu+\nu_0)q_\nu T + q_\nu^2T^2\right) \\
& -3e^{-2bT}\Big(3q_\nu^2 + 3bq_\nu\Big(16d_\nu e^{bT}+\nu_0+q_\nu T\Big) + b^2(\nu_0+q_\nu T)\Big(16d_\nu e^{bT}+\nu_0+q_\nu T\Big)\Big)\Bigg],
\end{aligned}$$

$$\begin{aligned}
\nu_2^2(T) = \; & \frac{1}{4b^5T^3}e^{-2bT}\Bigg\{18q_\nu^2 + 6b^3T(\nu_0+q_\nu T)^2 + 6b^2(\nu_0+q_\nu T)(\nu_0+3q_\nu T) \\
& +9bq_\nu(2\nu_0+3q_\nu T) + e^{2bT}\Big[-18q_\nu^2 + 6b^3\nu_0(8d_\nu+\nu_0)T + 4b^5d_\nu^2T^3 \\
& +9bq_\nu(-32d_\nu-2\nu_0+q_\nu T) + 6b^2\big(-\nu_0(16d_\nu+\nu_0)+2(8d_\nu+\nu_0)q_\nu T\big)\Big] \\
& +48bd_\nu e^{bT}\Big(6q_\nu + b\big(\nu_0(2+bT)+q_\nu T(4+bT)\big)\Big)\Bigg\},
\end{aligned}$$

$$
\begin{aligned}
\eta_1(T) = \; & \frac{2}{T^2}\Bigg\{ -\frac{2d_\nu q_\rho}{a^3} - \frac{2d_\rho q_\nu}{b^3} - \frac{6q_\rho q_\nu}{(a+b)^4} + \frac{d_\rho \nu_0 T}{b} + \frac{d_\nu \rho_0 T}{a} + \frac{a^3\nu_0\rho_0 T}{(a+b)^4} + \frac{b^3\nu_0\rho_0 T}{(a+b)^4} \\
& + \frac{d_\nu(-\rho_0 + q_\rho T)}{a^2} + \frac{d_\rho(-\nu_0 + q_\nu T)}{b^2} + \frac{a^2(-\nu_0\rho_0 + \nu_0 q_\rho T + q_\nu \rho_0 T)}{(a+b)^4} \\
& - \frac{2a\big(\nu_0 q_\rho + q_\nu(\rho_0 - q_\rho T)\big)}{(a+b)^4} \\
& + \frac{b\big[-2\nu_0(q_\rho + a\rho_0) + a\nu_0(2q_\rho + 3a\rho_0)T + 2q_\nu\big(q_\rho T + \rho_0(-1+aT)\big)\big]}{(a+b)^4} \\
& + \frac{b^2\big[q_\nu\rho_0 T + \nu_0\big(q_\rho T + \rho_0(-1+3aT)\big)\big]}{(a+b)^4} \\
& + \frac{1}{2a^3b^3(a+b)^4} e^{-(a+b)T}\Bigg\{ 4b^7 d_\nu e^{bT} q_\rho + 8a^2b^5 d_\nu e^{bT}\big(b\rho_0 + q_\rho(3+bT)\big) \\
& + 2ab^6 d_\nu e^{bT}\big(b\rho_0 + q_\rho(8+bT)\big) + a^7 d_\rho e^{aT}\Big(4q_\nu + b^3 d_\nu e^{bT}T^2 + 2b(\nu_0 + q_\nu T)\Big) \\
& + 4a^6 b d_\rho e^{aT}\Big(4q_\nu + b^3 d_\nu e^{bT}T^2 + 2b(\nu_0 + q_\nu T)\Big) \\
& + 2a^5b^2\Big[12 d_\rho e^{aT} q_\nu + 3b^3 d_\rho d_\nu e^{(a+b)T}T^2 + 6b d_\rho e^{aT}(\nu_0 + q_\nu T) \\
& + b(\rho_0 + q_\rho T)\Big(d_\nu e^{bT} + \nu_0 + q_\nu T\Big)\Big] + 4a^4b^3\Big[d_\nu e^{bT} q_\rho + \nu_0 q_\rho + 4d_\rho e^{aT} q_\nu + q_\nu\rho_0 + 2q_\rho q_\nu T \\
& + b^3 d_\rho d_\nu e^{(a+b)T}T^2 + 2b d_\rho e^{aT}(\nu_0 + q_\nu T) + b(\rho_0 + q_\rho T)\Big(2d_\nu e^{bT} + \nu_0 + q_\nu T\Big)\Big] \\
& + a^3b^3\Big[12 q_\rho q_\nu + b^4 d_\rho d_\nu e^{(a+b)T}T^2 + 4b\Big(4d_\nu e^{bT} q_\rho + \nu_0 q_\rho + q_\nu\big(d_\rho e^{aT} + \rho_0 + 2q_\rho T\big)\Big) \\
& + 2b^2\Big[d_\rho e^{aT}(\nu_0 + q_\nu T) + (\rho_0 + q_\rho T)\Big(6d_\nu e^{bT} + \nu_0 + q_\nu T\Big)\Big]\Big]\Bigg\}\Bigg\},
\end{aligned}
$$

$$
\begin{aligned}
\eta_2^2(T) = \; & \frac{12}{T^4}\int_0^T\int_0^t \Bigg\{ \frac{1}{a^2b^2(a+b)^3} e^{-(a+b)s}\Bigg\{ b^5 d_\nu e^{bs}\left(-1 + e^{as}\right) q_\rho 6 \\
& - ab^4 d_\nu e^{bs}\big(3q_\rho + b\rho_0 - e^{as}(3q_\rho + b\rho_0) + bq_\rho s\big) \\
& + a^5 d_\rho e^{as}\Big(-q_\nu - b(\nu_0 + q_\nu s) + e^{bs}\big(q_\nu + b(\nu_0 + bd_\nu s)\big)\Big) + a^3b^2\left[-q_\nu\left(3d_\rho e^{as} + \rho_0\right)\right. \\
& + e^{(a+b)s}\left((d_\nu + \nu_0)q_\rho + q_\nu(3d_\rho + \rho_0) + b(3d_\rho\nu_0 + 3d_\nu\rho_0 + 2\nu_0\rho_0) + 3b^2 d_\rho d_\nu s\right) \\
& \left. - d_\nu e^{bs}(q_\rho + 3b\rho_0 + 3bq_\rho s) - q_\rho(\nu_0 + 2q_\nu s) - b(\nu_0 + q_\nu s)\big(3d_\rho e^{as} + 2(\rho_0 + q_\rho s)\big)\right] \\
& + a^2b^2\Big[-2q_\rho q_\nu + e^{(a+b)s}\Big(2q_\rho q_\nu + b^2\big(3d_\nu\rho_0 + \nu_0(d_\rho + \rho_0)\big) + b\big((3d_\nu + \nu_0)q_\rho \\
& + q_\nu(d_\rho + \rho_0)\big) + b^3 d_\rho d_\nu s\Big) - 3bd_\nu e^{bs}(q_\rho + b\rho_0 + bq_\rho s) - b^2\left(d_\rho e^{as} + \rho_0 + q_\rho s\right)(\nu_0 + q_\nu s) \\
& - b\big(\nu_0 q_\rho + q_\nu\left(d_\rho e^{as} + \rho_0 + 2q_\rho s\right)\big)\Big] + a^4 b\Big[e^{(a+b)s}\big(3bd_\rho\nu_0 + 3d_\rho q_\nu + b(d_\nu + \nu_0)\rho_0 \\
& + 3b^2 d_\rho d_\nu s\big) - b(\rho_0 + q_\rho s)\Big(d_\nu e^{bs} + \nu_0 + q_\nu s\Big) - 3d_\rho e^{as}\big(q_\nu + b(\nu_0 + q_\nu s)\big)\Big]\Bigg\}\Bigg\}^2 ds dt.
\end{aligned}
$$

Note that for this general case, it is not possible to obtain an explicit expression

for $\eta_2^2(T)$, so that it must be approximated using an appropriate numerical quadrature formula.

B.2 Market data

Time	**Year fraction,** T	**Interest rate,** r	**Dividend yield,** y
3 months	0.2438	1.4198 %	1.5620 %
6 months	0.4959	1.2413 %	2.9769 %
12 months	1	1.0832 %	1.9317 %
24 months	2	1.0394 %	1.8610 %

Table B.1: EURO STOXX 50 (Dec. 2011). Spot value $S_0 = 2311.1$ €. Interest rates and dividend yields.

K (% of S_0)	3 months	6 months	12 months	24 months
80%	33.90%	33.81%	31.38%	29.25%
82%	33.47%	33.37%	31.06%	28.98%
84%	33.05%	32.93%	30.75%	28.70%
86%	32.62%	32.49%	30.43%	28.43%
88%	32.21%	32.07%	30.11%	28.16%
90%	31.79%	31.64%	29.79%	27.89%
92%	31.38%	31.23%	29.48%	27.62%
94%	30.98%	30.82%	29.16%	27.34%
96%	30.58%	30.42%	28.85%	27.08%
98%	30.18%	30.02%	28.53%	26.81%
100%	29.79%	29.63%	28.22%	26.54%
102%	29.40%	29.24%	27.90%	26.27%
104%	29.01%	28.87%	27.59%	26.00%
106%	28.63%	28.49%	27.28%	25.74%
108%	28.26%	28.13%	26.96%	25.47%
110%	27.89%	27.77%	26.65%	25.21%
112%	27.52%	27.42%	26.34%	24.95%
114%	27.16%	27.07%	26.03%	24.68%
116%	26.80%	26.73%	25.72%	24.42%
118%	26.45%	26.40%	25.41%	24.16%
120%	26.10%	26.07%	25.10%	23.90%

Table B.2: EURO STOXX 50 (Dec. 2011). Implied volatilities for each maturity with different strikes K (% of the spot S_0).

Time	**Year fraction,** T	**Interest rate,** r	**Dividend yield,** y
3 months	0.2528	1.3696 %	0.5894 %
6 months	0.5083	1.2110 %	0.6185 %
12 months	1	1.0832 %	0.6907 %
24 months	2	1.0394 %	0.7438 %

Table B.3: EUR/USD (Dec. 2011). Spot value $S_0 = 1.2939$ US dollars. Interest rates and dividend yields.

3 **months**		6 **months**		12 **months**		24 **months**	
K	σ_{market}	K	σ_{market}	K	σ_{market}	K	σ_{market}
1.1075	19.27%	1.0241	20.75%	0.9217	21.80%	0.8245	21.01%
1.1516	18.58%	1.0877	19.90%	1.0084	20.82%	0.9302	20.11%
1.1817	17.93%	1.1316	19.11%	1.0693	19.91%	1.0063	19.27%
1.2075	16.85%	1.1700	17.71%	1.1240	18.22%	1.0746	17.75%
1.2262	16.40%	1.1972	17.19%	1.1621	17.64%	1.1244	17.21%
1.2425	16.00%	1.2210	16.70%	1.1956	17.09%	1.1686	16.70%
1.2570	15.62%	1.2423	16.26%	1.2257	16.60%	1.2089	16.24%
1.2704	15.28%	1.2618	15.85%	1.2534	16.15%	1.2463	15.82%
1.2830	14.97%	1.2801	15.49%	1.2794	15.74%	1.2818	15.44%
1.2950	14.70%	1.2975	15.17%	1.3043	15.39%	1.3161	15.11%
1.3066	14.46%	1.3145	14.88%	1.3286	15.07%	1.3500	14.81%
1.3183	14.25%	1.3315	14.64%	1.3530	14.80%	1.3843	14.55%
1.3302	14.07%	1.3489	14.44%	1.3781	14.58%	1.4199	14.34%
1.3427	13.93%	1.3673	14.28%	1.4047	14.41%	1.4579	14.16%
1.3563	13.83%	1.3872	14.15%	1.4339	14.28%	1.5000	14.03%
1.3715	13.75%	1.4099	14.07%	1.4673	14.19%	1.5485	13.94%
1.3899	13.74%	1.4370	14.02%	1.5080	14.16%	1.6074	13.88%
1.4140	13.80%	1.4733	14.09%	1.5635	14.26%	1.6890	13.94%
1.4510	13.89%	1.5300	14.21%	1.6514	14.42%	1.8203	14.06%

Table B.4: EUR/USD (Dec. 2011). Implied volatilities for each maturity with different strikes K.

Appendix C

BSDEs

In this appendix the proof Proposition 5.2.1 and the stability results for discrete BSDE are shown.

C.1 Proof of Proposition 5.2.1

It is known from [58, Proposition 3.1] that it is sufficient to show that there is a continuous $C_\rho : \mathbb{R} \to [1, \infty)$ such that, for all $\Lambda \geq 0$, $\lambda \in [0, \Lambda]$, and $y \in \mathbb{R}^d$,

$$\frac{p^{(\mu)}_{\text{logis.}}(y)}{C_\rho(\Lambda)} \leq \int_{\mathbb{R}^d} p^{(\mu)}_{\text{logis.}}(y + z\sqrt{\lambda}) \exp(-\frac{|z|^2}{2}) \mathrm{d}z \leq C_\rho(\Lambda) p^{(\mu)}_{\text{logis.}}(y). \tag{C.1}$$

The proof is given for $d = 1$; generalization to higher dimensions is obvious because the multidimensional density is just the product of the one-dimensional densities over the components. Moreover, for simplicity the proof is given for $\mu = 1$, as generality in this parameter does not change the proof. For simplicity, we will write $p^{(\mu)}_{\text{logis.}}(x) = p(x)$ in what follows.

In terms of the hyperbolic cosine function, the density can be expressed as

$$p(x) = \frac{\exp(-x)}{\big(1 + \exp(-x)\big)^2} = \Big(\exp(\frac{x}{2}) + \exp(-\frac{x}{2})\Big)^{-1} = \Big(2\cosh(\frac{x}{2})\Big)^{-1}.$$

We define $I(y, \lambda) := 2 \int_{\mathbb{R}} p(y + z\sqrt{\lambda}) \exp(-\frac{z^2}{2}) \mathrm{d}z$, so that from the relation $\cosh(x +$

$y) = \cosh(x)\cosh(y) + \sinh(x)\sinh(y)$, we have that

$$I(y,\lambda) = \int_{\mathbb{R}} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\cosh(\frac{z\sqrt{\lambda}}{2}) + \sinh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2})}\mathrm{d}z := I_+(y,\lambda) + I_-(y,\lambda)$$

where $I_{+,-}$ denotes respectively the integral on $\mathbb{R}^+$ and $\mathbb{R}^-$.

Upper bound Suppose that $y \geq 0$. Then, if $z \geq 0$, it follows that $\sinh(y/2)\sinh(z\sqrt{\lambda}/2) \geq 0$, whence

$$I_+(y,\lambda) \leq \int_{\mathbb{R}_+} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\cosh(\frac{z\sqrt{\lambda}}{2})}\mathrm{d}z = 2\int_{\mathbb{R}_+} e^{-\frac{z^2}{2}}\mathrm{d}z \times p(y).$$

On the other hand, if $z \leq 0$, then $\sinh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2}) \geq \cosh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2})$, therefore

$$I_-(y,\lambda) \leq \int_{\mathbb{R}_-} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\{\cosh(\frac{z\sqrt{\lambda}}{2}) + \sinh(\frac{z\sqrt{\lambda}}{2})\}}\mathrm{d}z = 2\int_{\mathbb{R}_-} \exp\left(-\frac{z^2}{2} - \frac{z\sqrt{\lambda}}{2}\right)\mathrm{d}z \times p(y).$$

Therefore, if $y \geq 0$ then $I(y,\lambda) \leq 2\int_{\mathbb{R}} \exp(\frac{-z^2+(z)_-\sqrt{\Lambda}}{2})\mathrm{d}z \times p(y)$. Observing that $I(y,\lambda)$ is symmetric in y, thus the upper bound (C.1) is proved.

Lower bound Suppose that $y \geq 0$. For $z \leq 0$, observe that $\sinh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2}) \leq 0$, whence

$$I_-(y,\lambda) \geq \int_{\mathbb{R}_-} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\cosh(\frac{z\sqrt{\lambda}}{2})}\mathrm{d}z \geq 2\int_{\mathbb{R}_-} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{z\sqrt{\Lambda}}{2})}\mathrm{d}z \times p(y).$$

For $z \geq 0$, we use that $\sinh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2}) \leq \cosh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2})$ to obtain

$$\begin{aligned} I_+(y,\lambda) &\geq \int_{\mathbb{R}_+} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\{\cosh(\frac{z\sqrt{\lambda}}{2}) + \sinh(\frac{z\sqrt{\lambda}}{2})\}}\mathrm{d}z \\ &\geq 2\int_{\mathbb{R}_+} \exp\left(-\frac{z^2}{2} - \frac{z\sqrt{\Lambda}}{2}\right)\mathrm{d}z \times p(y). \end{aligned}$$

The result on $y < 0$ follows again from the symmetry of $I(y,\lambda)$. □

C.2 Stability results for discrete BSDE

We recall standard results borrowed to [60] and adapted to our setting, they are aimed at comparing two solutions of discrete BSDEs of the form (5.5) with different data. Namely, consider two discrete BSDEs, $(Y_{1,i}, Z_{1,i})_{0\leq i<N}$ and $(Y_{2,i}, Z_{2,i})_{0\leq i<N}$, given by

$$Y_{l,i} = \mathbb{E}_i\left(g(X_N) + \sum_{j=i}^{N-1} f_{l,j}(X_j, Y_{l,j+1}, Z_{l,j})\Delta_t\right),$$
$$\Delta_t Z_{l,i} = \mathbb{E}_i\left((g(X_N) + \sum_{j=i+1}^{N-1} f_{l,j}(X_j, Y_{l,j+1}, Z_{l,j})\Delta_t)\Delta W_i\right),$$

for $i \in \{0, \dots, N-1\}$, $l \in \{1, 2\}$.

To allow the driver $f_{1,i}$ to depend on the clouds of simulations (necessary in the analysis), we require that it is measurable w.r.t. $\mathcal{F}_T$ instead of $\mathcal{F}_{t_i}$ as usually.

Proposition C.2.1. *Assume that* $(\mathbf{A_g})$ *and* $(\mathbf{A_X})$*hold. Moreover, for each* $i \in \{0, \dots, N-1\}$*, assume that* $f_{1,i}(X_i, Y_{1,i+1}, Z_{1,i}) \in \mathbf{L}_2(\mathcal{F}_T)$ *and* f_2 *satisfies* $(\mathbf{A_f})$ *with constants* L_{f_2} *and* C_{f_2}*. Then, for any* $\gamma \in (0, +\infty)$ *satisfying* $6q(\Delta_t + \frac{1}{\gamma})L_{f_2}^2 \leq 1$*, we have for* $0 \leq i < N$

$$\begin{aligned}
&|Y_{1,i} - Y_{2,i}|^2\Gamma_i + \sum_{j=i}^{N-1} \Delta_t \mathbb{E}_i\left(|Z_{1,j} - Z_{2,j}|^2\right)\Gamma_j \\
&\leq 3C_{C.2.1}\left(\frac{1}{\gamma} + \Delta_t\right)\sum_{j=i}^{N-1} \Delta_t \mathbb{E}_i\left(|f_{1,j}(X_j, Y_{1,j+1}, Z_{1,j}) - f_{2,j}(X_j, Y_{1,j+1}, Z_{1,j})|^2\right)\Gamma_j,
\end{aligned} \tag{C.2}$$

where $\Gamma_i := (1 + \gamma\Delta_t)^i$ *and* $C_{C.2.1} := 2q + (1+T)e^{T/2}$.

Conclusions

The main objectives of this thesis have been to propose efficient procedures to calibrate and price market derivatives under different SABR-like stochastic volatility models, not only in equity/foreign exchange markets but also in interest rate markets. Furthermore, we have developed an original algorithm to efficiently approximate the solution of Forward-Backward Stochastic Differential Equations.

For these purposes, we have proposed a highly optimized version of a Simulated Annealing algorithm adapted to the more recently developed hardware tools of Graphic Processor Units (GPUs). The programming has been carried out with CUDA toolkit, specially designed for Nvidia GPUs. Efficient versions of Simulated Annealing have been first analyzed and adapted to GPUs. Thus, an appropriate sequential Simulated Annealing algorithm has been developed as starting point. Next, a straightforward asynchronous parallel version has been implemented and then a specific and more efficient synchronous version has been developed. A wide appropriate benchmark to illustrate the performance properties of the implementation has been considered. Among all tests, a classical sample problem provided by the minimization of the normalized Schwefel function has been selected to compare the behavior of the sequential, asynchronous and synchronous versions, the last one being more advantageous in terms of balance between convergence, accuracy and computational cost. Also the implementation of a hybrid method combining Simulated Annealing with a local minimizer method has been developed. Note that the generic feature of the Simulated Annealing algorithm allows its application in a wide set of real problems arising in a large variety of fields, such as biology, physics, engineering, finance and

industrial processes.

In the more classical models for equities and interest rates evolution, constant volatility is usually assumed. However, in practice the volatilities are not constant in financial markets and different models allowing a varying local or stochastic volatility also appear in the literature. Particularly, here we have considered the SABR model that has been first introduced in a paper by Hagan and coworkers, where an asymptotic closed-form formula for the implied volatility of European plain-vanilla options with short maturities is proposed. For the calibration of the parameters in static and dynamic SABR stochastic volatility models, we have proposed the application of the GPU technology to the Simulated Annealing global optimization algorithm and to the Monte Carlo simulation. This calibration has been performed for EURO STOXX 50 index and EUR/USD exchange rate with an asymptotic formula for volatility or Monte Carlo simulation. Moreover, in the dynamic model we have proposed an original more general expression for the functional parameters, specially well suited for the EUR/USD exchange rate case. Numerical results illustrate both the expected behavior of SABR models and the accuracy of the calibration. In terms of computational time, when the asymptotic formula for volatility is used the speedup with respect to CPU computation is around 200 with one GPU.

More recently, different works (Hagan-Lesniewski, Mercurio-Morini and Rebonato) have extended the use of SABR model to the context of LIBOR market models for the evolution of forward rates (SABR-LMM). One drawback of these models in practice comes from the increase of computational cost, mainly due to the growth of model parameters to be calibrated. Additionally, sometimes either it is not always possible to compute an analytical approximation for the implied volatility or its expression results to be very complex, so that numerical methods (for example, Monte Carlo in the calibration process) need to be used. The numerical results have clearly illustrated the advantages of using the proposed multi-GPUs tools when applied to real market data and popular SABR/LIBOR models.

In order to be able to overcome the drawbacks of the Monte Carlo simulation of the

studied SABR/LIBOR market models, we have also posed original partial differential equations associated to the studied systems of stochastic differential equations. The resulting partial differential equations are high dimensional in space, therefore we have implemented the sparse grid combination technique to cope with the curse of dimensionality to some extent. Indeed, we have efficiently priced derivatives with up to eight underlying LIBORs and their stochastic volatility.

Finally, we have designed a novel algorithm based on Least-Squares Monte Carlo (LSMC) in order to approximate the solution of discrete time Backward Stochastic Differential Equations (BSDEs). As a first step, we have drastically reworked an algorithm by Gobet and Turkedjiev to first minimize the exposure to the memory due to the storage of the simulations. This has allowed the computations in significantly larger dimensions. In this way, the algorithm could be implemented in parallel on GPU processors to optimize computational time. Numerical results up to dimension 19 have been illustrated. Moreover, the error analysis of the algorithm has been addressed.

Resumen extenso

En esta tesis hemos analizado la valoración de derivados financieros empleando determinados modelos matemáticos. Nuestro objetivo ha sido ilustrar el uso de estos modelos, poniendo énfasis en su implementación y calibración.

Un derivado financiero es un contrato cuyo valor depende de uno o más activos, denominados activos subyacentes. Normalmente, el activo subyacente es una acción, un tipo de intercambio de divisas, el precio de mercado de determinadas materias primas como el aceite o el trigo, o un bono (tipo de interés). Entre la gran variedad de derivados financieros que se comercializan hoy en día, una opción es el ejemplo más sencillo. Una opción es un contrato que da a su poseedor el derecho (pero no la obligación) de comprar o vender el activo subyacente a un precio prefijado en una fecha futura. Una opción de *compra* da el derecho a comprar, mientras que una opción de *venta* da el derecho a vender. Una opción se denomina *europea* si el derecho a comprar o vender puede ser ejercido solamente en la fecha de vencimiento, y se conoce como *americana* si puede ejercerse en cualquier instante anterior al vencimiento. Las opciones de compra y venta son los instrumentos derivados básicos y por ello normalmente se conocen como opciones vainilla. Sin embargo, existe una gran cantidad de derivados, normalmente conocidos como *exóticos*, cuya estructura entraña más complejidad. Valorar estos instrumentos derivados no es trivial, debido a que se desconoce el modo en que evolucionarán en el futuro los precios de los activos subyacentes.

La primera vez que se comercializaron estos derivados financieros en mercados organizados fue el 26 de abril de 1973 en el CBOE (*Chicago Board Options Exchange*). Primeramente simplemente se operaba con opciones de compra sobre 16 acciones, las

opciones de venta no fueron introducidas hasta 1977. En el año 1973, Merton [104] y Black y Scholes [13] publicaron por separado la teoría básica de valoración de opciones, la cobertura dinámica y la teoría de no arbitraje. Empleando estas estrategias, los autores obtuvieron la celebrada ecuación en derivadas parciales (EDP) de Black-Scholes y la fórmula de Black-Scholes para valorar opciones vainilla europeas. A pesar del gran éxito de esta fórmula, después de la quiebra del mercado de acciones en octubre de 1987, se reveló que el hecho de asumir en el modelo de Black-Scholes [13]

$$dS(t) = rS(t)dt + \sigma S(t)dW(t), \tag{1}$$

que la volatilidad σ del activo subyacente S era constante originaba limitaciones significativas en la valoración de las opciones. En (1) estamos considerando la medida de probabilidad riesgo neutro, donde r representa el tipo de interés libre de riesgo y W es un movimiento Browniano. Con el objetivo de fijar ideas, consideremos el precio en tiempo t de una opción europea de compra con precio de ejercicio K. Este contrato paga la cantidad

$$\max(S(T) - K, 0) = (S(T) - K)^+,$$

en su fecha de vencimiento T. Su valor a tiempo $t < T$ viene dado por la fórmula de Black

$$V^{\text{Black}}(S, t, \sigma, r, K, T) = S\Phi(d_1) - K\mathrm{e}^{-r(T-t)}\Phi(d_2),$$

donde Φ es la función de distribución acumulada de la distribución normal estándar y

$$d_1 = \frac{\log(S/K) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},$$
$$d_2 = \frac{\log(S/K) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}.$$

Dicha fórmula de Black es el método estándar de valoración en el mercado de opciones europeas de compra. Si conocemos el valor de la volatilidad σ y el resto de los parámetros podemos calcular el precio de la opción y viceversa, si disponemos de la cotización del precio de la opción $V^{\text{Black}}(S, t, \sigma, r, K, T)$ podemos deducir el valor de

la volatilidad σ, que se denomina volatilidad implícita. A continuación consideramos dos precios de ejercicio diferentes K_1 y K_2. Supongamos que en el mercado cotizan los precios de las dos opciones de compra correspondientes a los mencionados precios de ejercicio, $V^{\text{Black}}(S, t, \sigma, r, K_1, T)$ y $V^{\text{Black}}(S, t, \sigma, r, K_2, T)$. Nótese que ambas opciones de compra tienen el mismo subyacente S y el mismo vencimiento T. La cuestión clave es que no existe un único parámetro de volatilidad σ tal que

$$V^{\text{Market}}(S, t, K_1, T) = V^{\text{Black}}(S, t, \sigma, r, K_1, T),$$

y

$$V^{\text{Market}}(S, t, K_2, T) = V^{\text{Black}}(S, t, \sigma, r, K_2, T),$$

es decir, se necesitan dos volatilidades diferentes $\sigma(T, K_1)$ y $\sigma(T, K_2)$ para recuperar los precios de mercados observados:

$$V^{\text{Market}}(S, t, K_1, T) = V^{\text{Black}}(S, t, \sigma(T, K_1), r, K_1, T),$$

$$V^{\text{Market}}(S, t, K_2, T) = V^{\text{Black}}(S, t, \sigma(T, K_2), r, K_2, T).$$

Se podría seguir un argumento análogo fijando el precio de ejercicio K y considerando dos fechas de vencimiento T_1 y T_2. Por tanto, cada precio de mercado de la opción de compra precisa su propia volatilidad de Black $\sigma(T, K)$ dependiendo del precio de ejercicio K y del vencimiento T de la opción de compra. La forma de la gráfica de la volatilidad implícita frente al precio de ejercicio normalmente presenta formas de sonrisa (ver Figura 1), por tanto se conoce como sonrisa de volatilidad (*volatility smile* o *skew*). En algunos mercados muestra una asimetría considerable.

Teniendo en cuenta que las dinámicas descritas por la ecuación diferencial estocástica (1) no son capaces de capturar adecuadamente las volatilidades implícitas de mercado, los investigadores han tratado de encontrar modelos alternativos que sean adecuados para este propósito. A continuación revisamos brevemente los principales enfoques propuestos en la literatura.

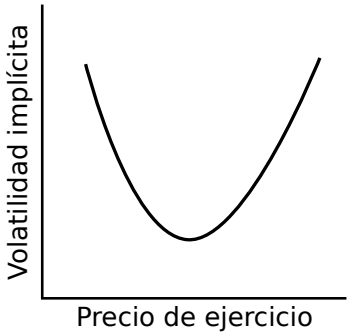


Figure 1: Forma común de la sonrisa de volatilidad.

Modelos de volatilidad local Estos modelos son extensiones analíticas directas de un movimiento Browniano geométrico, que permiten hasta cierto punto capturar sonrisas en la volatilidad implícita. Los principales ejemplos son los siguientes. En el año 1976 Cox y Ross [30] propusieron los procesos conocidos como CEV, *constant-elasticity-of-variance*. Los autores consideraron ecuaciones diferenciales estocásticas de la forma

$$dS(t) = rS(t)dt + \sigma S(t)^{\beta} dW(t), \quad \beta \in (0,1),$$

donde el parámetro β es el que añade al modelo la capacidad de capturar las sonrisas de volatilidad del mercado. En los años 1994 y 1997 Dupire [37, 38] sugiere el modelo

$$dS(t) = rS(t)dt + \sigma(S,t)S(t)dW(t),$$

donde la volatilidad instantánea σ es una función determinista del precio del activo S y del tiempo t.

Modelos de difusión con saltos Estos modelos han sido introducidos con el objetivo de modelar discontinuidades en el proceso estocástico subyacente. En los mercados de acciones estos modelos fueron incorporados en el año 1976 por Merton [105] y se emplean normalmente con el objetivo de calibrar sonrisas de volatilidad.

Modelos de volatilidad estocástica Estos modelos han sido diseñados para reproducir las sonrisas de volatilidad en el mercado y para capturar la naturaleza estocástica de la volatilidad. Los principales ejemplos son los modelos de Hull y White (1987), Heston (1993) y Hagan, Kumar, Lesniewski y Woodward (2002). Son modelos más realistas, pero tienen la contrapartida de ser más complejos en términos de la valoración de opciones.

En el modelo de Hull y White [77] el activo y su volatilidad se modelan de la siguiente forma

$$\begin{aligned} dS(t) &= rS(t)dt + \sqrt{\sigma(t)}S(t)dW(t), \quad S(0) = S_0, \\ d\sigma(t) &= \kappa\sigma(t)dt + \zeta\sigma(t)dZ(t), \quad \sigma(0) = \sigma_0, \end{aligned}$$

donde dW y dZ tienen coeficiente de correlación ρ. Los otros parámetros son el valor de retorno de la volatilidad κ, la volatilidad de la volatilidad ζ y el valor inicial de la volatilidad σ_0.

El modelo de Heston [73] está definido por

$$\begin{aligned} dS(t) &= rS(t)dt + \sqrt{\sigma(t)}S(t)dW(t), \quad S(0) = S_0, \\ d\sigma(t) &= \kappa(\theta - \sigma(t))dt + \zeta\sqrt{\sigma(t)}dZ(t), \quad \sigma(0) = \sigma_0, \end{aligned}$$

donde $dW(t)dZ(t) = \rho dt$. Los otros parámetros del modelo son la velocidad de reversión a la media κ, la volatilidad a largo plazo θ, la volatilidad de la volatilidad ζ y el valor inicial de la volatilidad σ_0. Este modelo es muy popular entre los operadores de mercado debido a la existencia de fórmulas cerradas para valorar las opciones europeas, lo cual resulta particularmente útil durante el proceso de calibración a los datos de mercado. Dichas fórmulas analíticas se obtienen empleando la función característica, que se calcula resolviendo la EDP correspondiente al modelo de Heston y mediante la inversión de una transformada de Fourier.

En [67] Hagan, Kumar, Lesniewski y Woodward han propuesto el llamado modelo SABR, que es la extensión natural del modelo CEV clásico a la volatilidad estocástica. El nombre SABR es un acrónimo de *Stochastic*, Alpha, Beta y Rho, tres de los cuatro

parámetros del modelo. Las dinámicas del precio *forward* $F(t) = \mathrm{e}^{(r-d)(T-t)}S(t)$ están determinadas por el siguiente sistema de ecuaciones diferenciales estocásticas

$$dF(t) = \alpha(t)F(t)^{\beta}dW(t), \quad F(0) = F_0,$$
$$d\alpha(t) = \nu\alpha(t)dZ(t), \quad \alpha(0) = \alpha_0,$$

donde (W, Z) es un movimiento Browniano bidimensional con correlación constante ρ. Los otros parámetros del modelo son la elasticidad de la varianza $\beta \in [0, 1]$, la volatilidad de la volatilidad ν y el nivel de referencia de la volatilidad α_0. El hecho de que el modelo SABR se haya convertido en el estándar de mercado para reproducir los precios de opciones europeas hace que este modelo destaque sobre todos los demás propuestos en la literatura hasta el momento. Aunque los modelos de volatilidad local podrían ajustarse al mercado incluso mejor que el modelo SABR, estos modelos predicen evoluciones para el subyacente que son poco realistas. De hecho el modelo SABR refleja cambios en los precios de las opciones con respecto al precio de ejercicio, a diferencia de los modelos de volatilidad local que capturan variaciones en los precios de las opciones con respecto a la evolución del subyacente.

Entre la gran cantidad de derivados financieros comercializados hoy en día, cuando el derivado es un tipo de interés o un conjunto de ellos, aparece la clase de derivados de tipos de interés. En este trabajo hemos considerado principalmente bonos, *caplets*, *caps*, *swaps* y *swaptions*. Un bono es un contrato que paga periódicamente cupones dependiendo de la evolución de ciertos tipos de interés. Un *caplet* es una opción de compra que paga la diferencia positiva entre un tipo de interés variable y otro fijo (*strike*). Un contrato *cap* es un conjunto de *caplets* asociados con varias fechas de vencimiento. Un *swap* es un contrato que intercambia dos tipos de interés diferentes. Un *swaption* es una opción que da el derecho a entrar en un *swap* en una fecha futura dada. En el libro de Brigo y Mercurio [19] puede encontrarse una descripción detallada sobre estos y otros derivados de tipos de interés. A diferencia del caso de mercados de acciones/divisas, en los mercados de tipos de interés la larga duración de los contratos y el comportamiento de los tipos de interés ha originado la consideración de modelos de tipos de interés estocásticos. Estos modelos pueden clasificarse en dos

categorías, modelos de tipo a corto plazo (*short rate models*) y modelos de mercado (*market models*).

Los modelos de tipo a corto plazo especifican las dinámicas de la evolución de un sólo tipo de interés, y a partir de ellas se determina la evolución futura de toda la curva de factores de descuento. Los populares modelos de Vasicek (1977) [136] y Cox, Ingersoll y Ross (1985) [29] pertenecen a esta categoría. El principal inconveniente de los modelos de tipos de interés a corto plazo es la imposibilidad de calibrar sus parámetros a la curva inicial de los factores de descuento, especialmente para aquellos modelos en los que no hay disponibles fórmulas de valoración analíticas.

En 1986, Ho y Lee [75] propusieron la primera alternativa a los modelos de tipos de interés a corto plazo, lo que supuso el primer trabajo en la categoría de los modelos de mercado. Los autores modelaron la evolución de toda la curva de factores de descuento empleando un árbol binomial. Más tarde, en el año 1992, Heath, Jarrow y Morton [69] trasladaron a tiempo continuo la hipótesis básica del modelo de Ho y Lee. Su modelo HJM se convirtió en el estándar de mercado para los tipos de interés a principios de los noventa. Sin embargo, el principal inconveniente del modelo HJM era su incompatibilidad con el uso en el mercado de las fórmulas de Black para valorar *caplets* y *swaptions*.

Con el objetivo de superar el principal obstáculo del modelo HJM, en el año 1999, Miltersen, Sandmann y Sondermann [108] publicaron un método basado en EDPs para derivar la fórmula de Black de valoración de *caplets* dentro del marco libre de arbitraje ofrecido por el modelo HJM. Teniendo en cuenta dicho método, Brace, Gatarek y Musiela [16] derivaron el llamado modelo BGM, también conocido como modelo de mercado LIBOR (LMM), ya que modela la evolución de los tipos LIBOR futuros empleando una distribución lognormal bajo determinadas medidas relevantes. En el año 1997, Jamshidian [79] también contribuyó significativamente a su desarrollo. Los tipos de interés de referencia más importantes son los *London Interbank Offered Rates* o LIBORs, que se calculan diariamente a partir de una media de los tipos de interés ofrecidos por bancos en Londres. El modelo LMM se ha convertido en el

modelo de tipos de interés más empleado. La principal razón es la consonancia entre este modelo y las fórmulas de Black. De hecho, el modelo LIBOR valora *caps* con la fórmula de Black para *caps*, que es la fórmula estándar empleada en el mercado de *caps*. Además, el modelo *Swap market model* (SMM) valora *swaptions* con la fórmula de Black para *swaptions*, que de nuevo es la fórmula estándar empleada en el mercado de *swaptions*. Teniendo en cuenta que los *caps* y los *swaptions* son los derivados de tipos de interés más comercializados, es muy importante que un modelo de mercado sea compatible con tales fórmulas de mercado. Además, los parámetros de estos modelos pueden calibrarse fácilmente a los precios de mercado empleando productos que cotizan diariamente.

El modelo de mercado LIBOR estándar considera volatilidades constantes para los tipos futuros. Sin embargo, esta es una hipótesis muy limitada, debido a que es imposible reproducir las sonrisas de volatilidad de los mercados. El modelo SABR no puede ser empleado para valorar derivados cuyas funciones de pago dependan de varios tipos futuros. De hecho, el modelo SABR trabaja en la medida terminal, bajo la cual tanto el tipo futuro como su volatilidad son martingalas. Esto siempre puede hacerse si trabajamos con un único tipo futuro aislado en cada tiempo. Sin embargo, bajo esta misma medida los procesos para otro tipo futuro y su volatilidad tendrían derivas. Para permitir que el modelo LMM capture las sonrisas de volatilidad del mercado, se han propuesto diferentes extensiones del modelo LMM que incorporaron la sonrisa de volatilidad por medio del modelo SABR. En este trabajo hemos estudiado los modelos propuestos por Hagan [68], Mercurio y Morini [103] y Rebonato [122]. El modelo de Hagan es la fusión natural entre los modelos de mercado SABR y LIBOR. En el modelo de Mercurio & Morini se asume la existencia de un único proceso de volatilidad lognormal que es común para todos los tipos *forward*. El modelo de Rebonato es análogo al modelo de Hagan, excepto en las dinámicas de las volatilidades.

En esta tesis nos hemos centrado en los modelos de tipo SABR debido a que son ampliamente empleados en la práctica por varios motivos. En primer lugar, haciendo

uso de técnicas de perturbación singulares es posible derivar fórmulas de aproximación para la volatilidad implícita bajo el modelo SABR. En segundo lugar, el modelo es relativamente simple y manejable. En tercer lugar, sus parámetros, que desempeñan roles específicos en la generación de sonrisas de volatilidad, tienen un significado intuitivo. Finalmente, se ha convertido en el estándar de mercado para interpolar y extrapolar precios de *caplets* y *swaptions* vainilla.

Desde el punto de vista numérico, en el marco de trabajo ofrecido por el modelo de mercado LIBOR la valoración de derivados de tipos de interés se realiza principalmente empleando simulación de Monte Carlo [49]. Sin embargo, teniendo en cuenta que la simulación de Monte Carlo tiene un elevado coste computacional, en este trabajo también hemos abordado, por primera vez en la literatura, el enfoque de valoración alternativo ofrecido por las EDPs. Así pues hemos planteado la original formulación en EDPs asociada a los tres modelos SABR/LIBOR propuestos por Hagan, Mercurio & Morini y Rebonato. No obstante, las EDPs asociadas a los modelos SABR/LIBOR tienen alta dimensión en espacio. Por tanto, los métodos tradicionales de mallas completas, como los métodos estándar de diferencias finitas o elementos finitos, no serán capaces de valorar derivados sobre más de tres o cuatro tipos de interés subyacentes, debido a la denominada maldición de la dimensión [7]. Con el propósito de vencer la maldición de la dimensión, el método de *sparse grid combination technique* propuesto originalmente por Zenger, Griebel y Schneider [63] ha sido analizado.

Los modelos estudiados en la primera parte de la tesis tienen un número elevado de parámetros. Calibrar estos parámetros a los datos de mercado es un objetivo real en la práctica. La calibración es el procedimiento de calcular los parámetros de un modelo ajustándolo a los precios cotizados de las opciones en el mercado. El enfoque de calibración estándar minimiza la distancia entre los precios del modelo, V^{model}, y los precios de mercado, V^{market}. Una medida común del error es el error cuadrático

$$SE = \sum_{k=1}^{N} \left(V_k^{\text{market}} - V_k^{\text{model}}(\boldsymbol{x}) \right)^2,$$

donde N denota el número de precios de opciones a los que se desea calibrar el modelo y $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ es el vector de los parámetros del modelo. La medida del error es una función $f : \mathbb{R}^n \to \mathbb{R}$ de los parámetros del modelo $\boldsymbol{x}$. Debido a que estamos buscando el vector de parámetros $\boldsymbol{x}^\star$ que permita el mejor ajuste del modelo a los precios de mercado disponibles, el procedimiento de calibración puede ser interpretado como un problema de optimización de la forma

$$\min_{\boldsymbol{x} \in I} f(\boldsymbol{x}),$$

donde $I \subseteq \mathbb{R}^n$ es el conjunto admisible de los parámetros del modelo $\boldsymbol{x}$, $I = I_1 \times \ldots \times I_n$, $I_k = [l_k, u_k]$, con l_k, $u_k \in \mathbb{R}$ para $k = 1, ..., n$. Es deseable emplear un algoritmo de minimización que no haga uso de las derivadas de la función de coste debido a que para el tipo de funciones de coste con las que vamos a tratar no disponemos de fórmulas analíticas. Además, en este caso la derivación numérica no es una alternativa debido a que es computacionalmente costosa. En este trabajo nos hemos centrado en la optimización estocástica, en particular en el conocido algoritmo de *Simulated Annealing* [88]. Con el objetivo de que las calibraciones de los modelos se realicen en el menor tiempo posible hemos empleado técnicas de computación de altas prestaciones.

En la segunda parte de la tesis hemos diseñado un nuevo algoritmo basado en Monte Carlo de Mínimos Cuadrados (*Least-Squares Monte Carlo*, LSMC) para aproximar los componentes (Y, Z) de la solución de la siguiente ecuación diferencial estocástica *forward-backward* (FBSDE),

$$\begin{aligned} Y_t &= g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)\mathrm{d}s - \int_t^T Z_s \mathrm{d}W_s, \\ X_t &= x + \int_0^t b(s, X_s)\mathrm{d}s + \int_0^t \sigma(s, X_s)\mathrm{d}W_s, \end{aligned}$$

donde W es un movimiento Browniano q-dimensional ($q \geq 1$). Además el algoritmo también ha aproximado la solución de la EDP parabólica semilineal asociada a dicha FBSDE.

En los últimos tiempos ha ido aumentando el interés de disponer de algoritmos que sean capaces de trabajar de manera eficiente cuando la dimensión d del espacio

ocupado por el proceso X es alta. Este interés ha sido propiciado principalmente por la comunidad de matemática financiera, en la que las reglas de valoración no lineales están siendo cada vez más importantes. Los algoritmos disponibles hasta el momento [18, 54, 59, 60] no eran capaces de manejar los casos en los que la dimensión era mayor que 8. El principal inconveniente no era simplemente el tiempo computacional necesario, sino principalmente el elevado uso de memoria principal de los citados algoritmos.

El objetivo de esta segunda parte de la tesis ha sido modificar totalmente el algoritmo propuesto en [60] para en primer lugar minimizar el uso de memoria principal debido al almacenamiento de las simulaciones. Esto nos ha permitido resolver el problema en dimensiones d más altas. De este modo, en segundo lugar el algoritmo ha podido ser implementado de forma paralela en unidades de procesamiento gráfico (GPUs), lo que nos ha permitido obtener substanciales aceleraciones con respecto a implementaciones tradicionales en CPU. Por ejemplo, hemos podido resolver problemas en dimensión $d = 11$ en ocho segundos empleando 2000 simulaciones por hipercubo. Con el objetivo de ilustrar el rendimiento del esquema propuesto hemos presentado varios experimentos numéricos, llegando a resolver problemas en dimensión $d = 19$. Además, se ha llevado a cabo el análisis del error de aproximación del algoritmo propuesto.

El esquema de esta memoria es el siguiente.

En la Parte I, que consta de cuatro capítulos, hemos trabajado con modelos de volatilidad estocástica de tipo SABR tanto en mercados de acciones/divisas como en mercados de tipos de interés. Nos hemos centrado en la valoración eficiente de distintos derivados financieros, así como en la calibración eficiente de los modelos estudiados a precios reales cotizados en los mercados.

El Capítulo 1 se ha dedicado a la presentación del algoritmo de optimización global estocástica *Simulated Annealing*. Este algoritmo se ha empleado posteriormente en las calibraciones de los modelos estudiados en los Capítulos 2 y 3. Hemos implementado los algoritmos haciendo uso de técnicas de computación de altas prestaciones (HPC)

debido a que en el mundo financiero las calibraciones deben hacerse en el menor tiempo posible.

En el Capítulo 2 hemos estudiado el modelo de volatilidad estocástica SABR en mercados de acciones y de divisas. Hemos analizado el modelo SABR clásico, llamado SABR estático, y otras extensiones de este modelo conocidas como SABR dinámico. Para el modelo SABR dinámico hemos propuesto una expresión original y más general para los parámetros funcionales. Posteriormente, hemos calibrado los modelos al índice EURO STOXX 50 y al tipo de cambio EUR/USD. Finalmente, hemos valorado un opción *cliquet* sobre el tipo de intercambio EUR/USD.

En el Capítulo 3 hemos presentado los modelos de mercado SABR/LIBOR, propuestos por Hagan, Mercurio & Morini y Rebonato. El principal objetivo de este capítulo ha sido calibrar eficientemente estos modelos a precios de mercado reales de *caplets* y *swaptions*. Hemos construido un conjunto de algoritmos, implementados haciendo uso de varias GPUs, que nos han permitido calibrar estos modelos empleando simulación de Monte Carlo. Este enfoque es particularmente útil cuando consideramos productos y modelos en los que no hay disponibles fórmulas de valoración, o bien estas no son suficientemente precisas.

En el Capítulo 4, al igual que en el capítulo anterior, hemos trabajado con los citados modelos de mercado SABR/LIBOR. Sin embargo, hemos seguido el enfoque alternativo ofrecido por las EDPs con el objetivo de intentar superar las limitaciones de la simulación de Monte Carlo, ver [139], a saber, convergencia muy lenta, la valoración de opciones con ejercicio anticipado y el cálculo de las denominadas "griegas". Hemos formulado los correspondientes modelos de EDPs para los modelos de mercado SABR/LIBOR introducidos en el Capítulo 3. Las EDPs resultantes tienen alta dimensión en espacio. Con el propósito de vencer la maldición de la dimensión hemos empleado la denominada *sparse grid combination technique*.

La Parte II, constituida por un único capítulo, se ha centrado en el estudio de las ecuaciones diferenciales estocásticas hacia atrás, BSDEs.

En el Capítulo 5 hemos diseñado un nuevo algoritmo basado en Monte Carlo de

Mínimos Cuadrados para aproximar la solución de *Backward Stochastic Differential Equations* discretas en tiempo. Nuestro algoritmo permite una paralelización masiva de las computaciones en procesadores multinúcleo tales como GPUs. El enfoque propuesto ha consistido en un nuevo método de estratificación, que ha sido crucial para permitir una paralelización altamente escalable. De este modo, hemos minimizado el consumo de memoria principal debido al almacenamiento de las simulaciones. De hecho, cabe destacar el menor consumo de memoria principal del algoritmo propuesto en comparación con el de los trabajos previos.

El Apéndice A contiene una breve descripción de los diferentes problemas de optimización que han sido testeados empleando la implementación propuesta del algoritmo *Simulated Annealing*. En el Apéndice B se ha obtenido la expresión de la volatilidad implícita del modelo SABR dinámico general. Además se han mostrado los datos de mercado empleados en el correspondiente Capítulo 2. El Apéndice C contiene dos resultados matemáticos relativos al algoritmo propuesto en el Capítulo 5 para resolver BSDEs.

Finalizamos con una breve sección que resume las principales conclusiones de este trabajo.

Resumo extenso

Nesta tese analizamos a valoración de derivados financeiros empregando determinados modelos matemáticos. O noso obxectivo foi ilustrar o uso destes modelos, poñendo énfase na súa implementación e calibración.

Un derivado financeiro é un contrato cuxo valor depende de un ou máis activos, denominados activos subxacentes. Normalmente, o activo subxacente é unha acción, un tipo de intercambio de moedas, o prezo de mercado de certas materias primas como o aceite ou o trigo, ou un bono (tipo de xuro). Entre a gran variedade de derivados financeiros que se comercializan hoxe en día, unha opción é o exemplo máis sinxelo. Unha opción é un contrato que dá ao seu posuidor o dereito (pero non a obrigación) de comprar ou vender o activo subxacente a un prezo prefixado nunha data futura. Unha opción de *compra* dá o dereito a comprar, mentres que unha opción de *venda* dá o dereito a vender. Unha opción denomínase *europea* se o dereito para comprar ou vender pode ser exercido soamente na data de vencemento, e coñécese como *americana* se pode exercerse en calquer instante anterior ao vencemento. As opcións de compra e venda son os instrumentos derivados básicos e por iso normalmente coñécense como opcións vainilla. Con todo, existe unha gran cantidade de derivados, normalmente coñecidos como *exóticos*, cuxa estrutura entraña máis complexidade. Valorar estes instrumentos derivados non é trivial, debido a que se descoñece o modo en que evolucionarán no futuro os prezos dos activos subxacentes.

A primeira vez que se comercializaron estes derivados financeiros en mercados organizados foi o 26 de abril do 1973 no CBOE (*Chicago Board Options Exchange*). Primeiramente operábase simplemente con opcións de compra sobre 16 accións, as

opcións de venda non foron introducidas ata o ano 1977. No ano 1973, Merton [104] e Black e Scholes [13] publicaron por separado a teoría básica de valoración de opcións, a cobertura dinámica e a teoría de non arbitraxe. Empregando estas estratexias, os autores obtiveron a celebrada ecuación en derivadas parciais (EDP) de Black-Scholes e a fórmula de Black-Scholes para valorar opcións vainilla europeas. A pesar do gran éxito desta fórmula, despois da quebra do mercado de accións en outubro de 1987, revelouse que o feito de asumir no modelo de Black-Scholes [13]

$$dS(t) = rS(t)dt + \sigma S(t)dW(t), \tag{1}$$

que a volatilidade σ do activo subxacente S era constante orixinaba limitacións significativas na valoración das opcións. En (1) estamos a considerar a medida de probabilidade risco neutro, onde r representa o tipo de xuro libre de risco e W é un movemento Browniano. Co obxectivo de fixar ideas, consideremos o prezo en tempo t dunha opción europea de compra con prezo de exercicio K. Este contrato paga a cantidade

$$\max(S(T) - K, 0) = (S(T) - K)^+,$$

na súa data de vencemento T. O seu valor a tempo $t < T$ ven dado pola fórmula de Black

$$V^{\text{Black}}(S, t, \sigma, r, K, T) = S\Phi(d_1) - K\mathrm{e}^{-r(T-t)}\Phi(d_2),$$

onde Φ é a función de distribución acumulada da distribución normal estándar e

$$d_1 = \frac{\log(S/K) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},$$

$$d_2 = \frac{\log(S/K) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}.$$

Dita fórmula de Black é o método estándar de valoración no mercado de opcións europeas de compra. Se coñecemos o valor da volatilidade σ e o resto dos parámetros podemos calcular o prezo da opción e viceversa, se dispoñemos da cotización do prezo da opción $V^{\text{Black}}(S, t, \sigma, r, K, T)$ poderemos deducir o valor da volatilidade σ, que se denomina volatilidade implícita. A continuación consideramos dous prezos de exercicio

diferentes K_1 e K_2. Supoñamos que no mercado cotizan os prezos das dúas opcións de compra correspondentes aos mencionados prezos de exercicio, $V^{\text{Black}}(S, t, \sigma, r, K_1, T)$ e $V^{\text{Black}}(S, t, \sigma, r, K_2, T)$. Nótese que ámbalas dúas opcións de compra teñen o mesmo subxacente S e o mesmo vencemento T. A cuestión clave é que non existe un único parámetro de volatilidade σ tal que

$$V^{\text{Market}}(S, t, K_1, T) = V^{\text{Black}}(S, t, \sigma, r, K_1, T),$$

e

$$V^{\text{Market}}(S, t, K_2, T) = V^{\text{Black}}(S, t, \sigma, r, K_2, T),$$

é dicir, precísanse dúas volatilidades diferentes $\sigma(T, K_1)$ e $\sigma(T, K_2)$ para recuperar os prezos de mercados observados:

$$V^{\text{Market}}(S, t, K_1, T) = V^{\text{Black}}(S, t, \sigma(T, K_1), r, K_1, T),$$

$$V^{\text{Market}}(S, t, K_2, T) = V^{\text{Black}}(S, t, \sigma(T, K_2), r, K_2, T).$$

Poderíase seguir un argumento análogo fixando o prezo de exercicio K e considerando dúas datas de vencemento T_1 e T_2. Por tanto, cada prezo de mercado da opción de compra precisa a súa propia volatilidade de Black $\sigma(T, K)$ dependendo do prezo de exercicio K e do vencemento T da opción de compra. A forma da gráfica da volatilidade implícita fronte ao prezo de exercicio normalmente presenta formas de sorriso (ver Figura 1), de ahí que se coñeza como sorriso de volatilidade (*volatility smile* ou *skew*). Nalgúns mercados mostra unha asimetría considerable. Se debuxamos as volatilidades implícitas fronte aos prezos de exercicio e vencementos, obsérvanse normalmente estructuras non planas, non só nos mercados de accións ou de divisas, senón tamén nos mercados de tipos de xuro. O feito de ignorar os sorrisos de volatilidade pode levar a unha incorrecta valoración das opcións.

Tendo en conta que as dinámicas descritas pola ecuación diferencial estocástica (1) non son capaces de capturar axeitadamente as volatilidades implícitas de mercado, os investigadores trataron de atopar modelos alternativos que fosen apropiados para este propósito. A continuación revisamos brevemente os principais enfoques propostos na literatura.

Figure 1: Forma común do sorriso de volatilidade.

Modelos de volatilidade local Estes modelos son extensións analíticas directas dun movemento Browniano xeométrico, que permiten ata certo punto capturar sorrisos na volatilidade implícita. Os principais exemplos son os seguintes. No ano 1976 Cox e Ross [30] propuxeron os procesos coñecidos como CEV, *constant-elasticity-of-variance*. Os autores consideraron ecuacións diferenciais estocásticas da forma

$$dS(t) = rS(t)dt + \sigma S(t)^{\beta} dW(t), \quad \beta \in (0,1),$$

onde o parámetro β é o que engade ao modelo a capacidade de capturar os sorrisos de volatilidade de mercado. Nos anos 1994 e 1997 Dupire [37, 38] suxire o modelo

$$dS(t) = rS(t)dt + \sigma(S,t)S(t)dW(t),$$

onde a volatilidade instantánea σ é unha función determinista do prezo do activo S e do tempo t.

Modelos de difusión con saltos Estes modelos foron introducidos co obxectivo de modelar discontinuidades no proceso estocástico subxacente. Nos mercados de accións estes modelos foron incorporados no ano 1976 por Merton [105] e empréganse normalmente co obxectivo de calibrar sorrisos de volatilidade.

Modelos de volatilidade estocástica Estes modelos foron deseñados para reproducir os sorrisos de volatilidade do mercado e para capturar a natureza estocástica da

volatilidade. Os principais exemplos son os modelos de Hull e White (1987), Heston (1993) e Hagan, Kumar, Lesniewski y Woodward (2002). Son modelos máis realistas, pero teñen o inconvinte de ser máis complexos en termos da valoración de opcións.

No modelo de Hull e White [77] o activo e a súa volatilidade modélanse do seguinte xeito

$$\begin{aligned} dS(t) &= rS(t)dt + \sqrt{\sigma(t)}S(t)dW(t), \quad S(0) = S_0, \\ d\sigma(t) &= \kappa\sigma(t)dt + \zeta\sigma(t)dZ(t), \quad \sigma(0) = \sigma_0, \end{aligned}$$

onde dW e dZ teñen coeficiente de correlación ρ. Os outros parámetros son o valor de retorno da volatilidade κ, a volatilidade da volatilidade ζ e o valor inicial da volatilidade σ_0.

O modelo de Heston [73] ven definido por

$$\begin{aligned} dS(t) &= rS(t)dt + \sqrt{\sigma(t)}S(t)dW(t), \quad S(0) = S_0, \\ d\sigma(t) &= \kappa(\theta - \sigma(t))dt + \zeta\sqrt{\sigma(t)}dZ(t), \quad \sigma(0) = \sigma_0, \end{aligned}$$

onde $dW(t)dZ(t) = \rho dt$. Os outros parámetros do modelo son a velocidade de reversión á media κ, a volatilidade a longo prazo θ, a volatilidade da volatilidade ζ e o valor inicial da volatilidade σ_0. Este modelo é moi popular entre os operadores de mercado debido á existencia de fórmulas pechadas para valorar as opcións europeas, o cal resulta particularmente útil durante o proceso de calibración aos datos de mercado. Ditas fórmulas analíticas obtéñense empregando a función característica, que se calcula resolvendo a EDP correspondente ao modelo de Heston e mediante a inversión dunha transformada de Fourier.

En [67] Hagan, Kumar, Lesniewski e Woodward propuxeron o chamado modelo SABR, que é a extensión natural do modelo CEV clásico á volatilidade estocástica. O nome SABR é un acrónimo de *Stochastic*, Alpha, Beta e Rho, tres dos catro parámetros do modelo. As dinámicas do prezo *forward* $F(t) = \mathrm{e}^{(r-d)(T-t)}S(t)$ veñen

determinadas polo seguinte sistema de ecuacións diferenciais estocásticas

$$dF(t) = \alpha(t)F(t)^{\beta}dW(t), \quad F(0) = F_0,$$
$$d\alpha(t) = \nu\alpha(t)dZ(t), \quad \alpha(0) = \alpha_0,$$

onde (W, Z) é un movemento Browniano bidimensional con correlación constante ρ. Os outros parámetros do modelo son a elasticidade da varianza $\beta \in [0, 1]$, a volatilidade da volatilidade ν e o nivel de referencia da volatilidade α_0. O feito de que o modelo SABR se convertese no estándar de mercado para reproducir os prezos de opcións europeas fai que este modelo destaque sobre todos os demais propostos na literatura ata o momento. Aínda que os modelos de volatilidade local poderían axustarse ao mercado mesmo mellor que o modelo SABR, estes modelos predín evolucións para o subxacente que son pouco realistas. De feito o modelo SABR reflicte cambios nos prezos das opcións con respecto ao prezo de exercicio, a diferenza dos modelos de volatilidade local que capturan variacións nos prezos das opcións con respecto á evolución do subxacente.

Entre a gran cantidade de derivados financeiros comercializados hoxe en día, cando o derivado é un tipo de xuro ou un conxunto deles, aparece a clase de derivados de tipos de xuro. Neste traballo consideramos principalmente bonos, *caplets*, *caps*, *swaps* e *swaptions*. Un bono é un contrato que paga periodicamente cupóns dependendo da evolución de certos tipos de xuro. Un *caplet* é unha opción de compra que paga a diferenza positiva entre un tipo de xuro variable e outro fixo (*strike*). Un contrato *cap* é un conxunto de *caplets* asociados con varias datas de vencemento. Un *swap* é un contrato que intercambia dous tipos de xuro diferentes. Un *swaption* é unha opción que dá o dereito a entrar nun *swap* nunha data futura dada. No libro de Brigo e Mercurio [19] pode atoparse unha descrición detallada sobre estes e outros derivados de tipos de xuro. A diferenza do caso dos mercados de accións/divisas, nos mercados de tipos de xuro a longa duración dos contratos e o comportamento dos tipos de xuro orixinou a consideración de modelos de tipos de xuro estocásticos. Estes modelos poden clasificarse en dúas categorías, modelos de tipo a curto prazo (*short rate models*) e modelos de mercado (*market models*).

Os modelos de tipo a curto prazo especifican as dinámicas da evolución dun só tipo de xuro, e a partir delas determínase a evolución futura de toda a curva de factores de desconto. Os populares modelos de Vasicek (1977) [136] e Cox, Ingersoll e Ross (1985) [29] pertencen a esta categoría. O principal inconveniente dos modelos de tipos de xuro a curto prazo é a imposibilidade de calibrar os seus parámetros á curva inicial dos factores de desconto, especialmente para aqueles modelos nos que non hai dispoñibles fórmulas de valoración analíticas.

En 1986, Ho e Lee [75] propuxeron a primeira alternativa aos modelos de tipos de xuro a curto prazo, o que supuxo o primeiro traballo na categoría dos modelos de mercado. Os autores modelaron a evolución de toda a curva de factores de desconto empregando unha árbore binomial. Máis tarde, no ano 1992, Heath, Jarrow e Morton [69] trasladaron a tempo continuo a hipótese básica do modelo de Ho e Lee. O seu modelo HJM converteuse no estándar de mercado para os tipos de xuro a principios dos noventa. Con todo, o principal inconveniente do modelo HJM era a súa incompatibilidade co uso no mercado das fórmulas de Black para valorar *caplets* e *swaptions*.

Co obxectivo de superar o principal obstáculo do modelo HJM, no ano 1999, Miltersen, Sandmann e Sondermann [108] publicaron un método baseado en EDPs para derivar a fórmula de Black de valoración de *caplets* dentro do marco libre de arbitraxe ofrecido polo modelo HJM. Tendo en conta devandito método, Brace, Gatarek e Musiela [16] derivaron o chamado modelo BGM, tamén coñecido como modelo de mercado LIBOR (LMM), xa que modela a evolución dos tipos LIBOR futuros empregando unha distribución lognormal baixo determinadas medidas relevantes. No ano 1997, Jamshidian [79] tamén contribuíu significativamente ao seu desenvolvemento. Os tipos de xuro de referencia máis importantes son os *London Interbank Offered Rates* ou LIBORs, que se calculan diariamente a partir dunha media dos tipos de xuro ofrecidos por bancos en Londres. O modelo LMM converteuse no modelo de tipos de xuro máis empregado. A principal razón é a consonancia entre este modelo e as fórmulas de Black. De feito, o modelo LIBOR valora *caps* coa fórmula de

Black para *caps*, que é a fórmula estándar empregada no mercado de *caps*. Ademais, o modelo *Swap market model* (SMM) valora *swaptions* coa fórmula de Black para *swaptions*, que de novo é a fórmula estándar empregada no mercado de *swaptions*. Tendo en conta que os *caps* e os *swaptions* son os derivados de tipos de xuro máis comercializados, é moi importante que un modelo de mercado sexa compatible con tales fórmulas de mercado. Ademais, os parámetros destes modelos poden calibrarse facilmente aos prezos de mercado empregando produtos que cotizan diariamente.

O modelo de mercado LIBOR estándar considera volatilidades constantes para os tipos futuros. Con todo, esta é unha hipótese moi limitada, debido a que é imposible reproducir os sorrisos de volatilidade dos mercados. O modelo SABR non pode ser empregado para valorar derivados cuxas funcións de pago dependan de varios tipos futuros. De feito, o modelo SABR traballa na medida terminal, baixo a cal tanto o tipo futuro como a súa volatilidade son martingalas. Isto sempre pode facerse se traballamos cun único tipo futuro illado en cada tempo. Non obstante, baixo esta mesma medida os procesos para outro tipo futuro e a súa volatilidade terían derivas. Para permitir que o modelo LMM capture os sorrisos de volatilidade do mercado, propuxéronse diferentes extensións do modelo LMM que incorporaron o sorriso de volatilidade por medio do modelo SABR. Neste traballo estudamos os modelos propostos por Hagan [68], Mercurio e Morini [103] e Rebonato [122]. O modelo de Hagan é a fusión natural entre os modelos de mercado SABR e LIBOR. O modelo de Mercurio & Morini asume a existencia dun único proceso de volatilidade lognormal que é común para todos os tipos *forward*. O modelo de Rebonato é análogo ao modelo de Hagan, excepto nas dinámicas das volatilidades.

Nesta tese centrámonos nos modelos de tipo SABR debido a que son amplamente empregados na práctica por varios motivos. En primeiro lugar, facendo uso de técnicas de perturbación singulares é posible derivar fórmulas de aproximación para a volatilidade implícita baixo o modelo SABR. En segundo lugar, o modelo é relativamente sinxelo e manexable. En terceiro lugar, os seus parámetros, que desempeñan roles

específicos na xeración de sorrisos de volatilidade, teñen un significado intuitivo. Finalmente, converteuse no estándar de mercado para interpolar e extrapolar prezos de *caplets* e *swaptions* vainilla.

Desde o punto de vista numérico, no marco de traballo ofrecido polo modelo de mercado LIBOR a valoración de derivados de tipos de xuro realízase principalmente empregando simulación de Monte Carlo [49]. Con todo, tendo en conta que a simulación de Monte Carlo ten un elevado custo computacional, neste traballo tamén abordamos, por primeira vez na literatura, o enfoque de valoración alternativo ofrecido polas EDPs. Así pois, propuxemos a orixinal formulación en EDPs asociada aos tres modelos SABR/LIBOR propostos por Hagan, Mercurio & Morini e Rebonato. Non obstante, as EDPs asociadas aos modelos SABR/LIBOR teñen alta dimensión en espazo. Por tanto, os métodos tradicionais de mallas completas, como os métodos estándar de diferenzas finitas ou elementos finitos, non serán capaces de valorar derivados sobre máis de tres ou catro tipos de xuro subxacentes, debido á denominada maldición da dimensión [7]. Co propósito de vencer a maldición da dimensión, o método de *sparse grid combination technique* proposto orixinalmente por Zenger, Griebel e Schneider [63] foi analizado.

Os modelos estudados na primeira parte da tese teñen un número elevado de parámetros. Calibrar estes parámetros aos datos de mercado é un obxectivo real na práctica. A calibración é o procedemento de calcular os parámetros dun modelo axustándoo aos prezos cotizados das opcións no mercado. O enfoque de calibración estándar minimiza a distancia entre os prezos do modelo, V^{model}, e os prezos de mercado, V^{market}. Unha medida común do error é o error cadrático

$$SE = \sum_{k=1}^{N} \left(V_k^{\text{market}} - V_k^{\text{model}}(\boldsymbol{x}) \right)^2,$$

onde N denota o número de prezos de opcións aos que se desexa calibrar o modelo e $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ é o vector dos parámetros do modelo. A medida do error é unha función $f : \mathbb{R}^n \to \mathbb{R}$ dos parámetros do modelo $\boldsymbol{x}$. Debido a que estamos a buscar o vector de parámetros $\boldsymbol{x}^\star$ que permita o mellor axuste do modelo aos prezos

de mercado dispoñibles, o procedemento de calibración pode ser interpretado como un problema de optimización da forma

$$\min_{\boldsymbol{x}\in I} f(\boldsymbol{x}),$$

onde $I \subseteq \mathbb{R}^n$ é o conxunto admisible dos parámetros do modelo $\boldsymbol{x}$, $I = I_1 \times \ldots \times I_n$, $I_k = [l_k, u_k]$, con l_k, $u_k \in \mathbb{R}$ para $k = 1, ..., n$. É desexable empregar un algoritmo de minimización que non faga uso das derivadas da función de custo debido a que para o tipo de funcións de custo coas que vamos a tratar non dispoñemos de fórmulas analíticas. Ademais, no caso que nos ocupa a derivación numérica non é unha alternativa debido a que é computacionalmente costosa. Neste traballo centrámonos na optimización estocástica, en particular no coñecido algoritmo de *Simulated Annealing* [88]. Co obxectivo de que as calibracións dos modelos se leven a cabo no menor tempo posible empregamos técnicas de computación de altas prestacións.

Na segunda parte da tese deseñamos un novo algoritmo baseado en Monte Carlo de Mínimos Cadrados (*Least-Squares Monte Carlo*, LSMC) para aproximar as compoñentes (Y, Z) da solución da seguinte ecuación diferencial estocástica *forward-backward* (FBSDE),

$$\begin{aligned} Y_t &= g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)\mathrm{d}s - \int_t^T Z_s \mathrm{d}W_s, \\ X_t &= x + \int_0^t b(s, X_s)\mathrm{d}s + \int_0^t \sigma(s, X_s)\mathrm{d}W_s, \end{aligned}$$

onde W é un movemento Browniano q-dimensional ($q \geq 1$). Ademais o algoritmo tamén aproximou a solución da EDP parabólica semilineal asociada a dita FBSDE.

Nos últimos tempos foi aumentando a interese por dispoñer de algoritmos que fosen capaces de traballar de maneira eficiente cando a dimensión d do espazo ocupado polo proceso X é alta. Esta interese foi propiciada principalmente pola comunidade de matemática financeira, na que as regras de valoración non lineais veñen sendo cada vez máis importantes. Os algoritmos dispoñibles ata o momento [18, 54, 59, 60] non eran capaces de manexar os casos nos que a dimensión era maior que 8. O

principal impedimento non era simplemente o tempo computacional necesario, senón principalmente o elevado uso de memoria principal dos citados algoritmos.

O obxectivo desta segunda parte da tese foi modificar totalmente o algoritmo proposto en [60] para en primeiro lugar minimizar o uso de memoria principal debido ao almacenamento das simulacións. Esto permitiunos resolver o problema en dimensións d máis altas. Deste xeito, en segundo lugar o algoritmo puido ser implementado de forma paralela en unidades de procesamento gráfico (GPUs), o que nos permitiu obter aceleracións substanciais con respecto a implementacións tradicionais en CPU. Por exemplo, puidemos resolver problemas en dimensión $d = 11$ en oito segundos empregando 2000 simulacións por hipercubo. Co obxectivo de ilustrar o rendemento do esquema proposto presentamos varios experimentos numéricos, chegando a resolver problemas en dimensión $d = 19$. Ademais, levouse a cabo a análise do erro de aproximación do algoritmo proposto.

O esquema desta memoria é o seguiente.

Na Parte I, que consta de catro capítulos, traballamos con modelos de volatilidade estocástica de tipo SABR tanto en mercados de accións/divisas como en mercados de tipos de xuro. Centrámonos na valoración eficiente de distintos derivados financeiros, así como na calibración eficiente dos modelos estudados a prezos reales cotizados nos mercados.

O Capítulo 1 dedicouse á presentación do algoritmo de optimización global estocástica *Simulated Annealing*. Este algoritmo foi empregado posteriormente nas calibracións dos modelos estudados nos Capítulos 2 e 3. Implementamos os algoritmos facendo uso de técnicas de computación de altas prestacións (HPC) debido a que no mundo financeiro as calibracións deben facerse no menor tempo posible.

No Capítulo 2 estudamos o modelo de volatilidade estocástica SABR en mercados de accións e de divisas. Analizamos o modelo SABR clásico, chamado SABR estático, e outras extensións deste modelo coñecidas como SABR dinámico. Para o modelo SABR dinámico propuxemos unha expresión orixinal e máis xeral para os parámetros funcionais. Posteriormente, calibramos os modelos ao índice EURO STOXX 50 e ao

tipo de cambio EUR/USD. Finalmente, valoramos unha opción *cliquet* sobre o tipo de intercambio EUR/USD.

No Capítulo 3 presentamos os modelos de mercado SABR/LIBOR propostos por Hagan, Mercurio & Morini e Rebonato. O principal obxectivo deste capítulo foi calibrar eficientemente estes modelos aos prezos de mercado reais de *caplets* e *swaptions*. Construimos un conxunto de algoritmos, implementados facendo uso de varias GPUs, que nos permitiron calibrar estes modelos empregando simulación de Monte Carlo. Este enfoque é particularmente útil cando consideramos produtos e modelos nos que non hay dispoñibles fórmulas de valoración, ou ben estas non son suficientemente precisas.

No Capítulo 4, ao igual que no capítulo anterior, traballamos cos citados modelos de mercado SABR/LIBOR. Sen embargo, seguimos o enfoque alternativo ofrecido polas EDPs co obxectivo de intentar superar as limitacións da simulación de Monte Carlo, ver [139], a saber, converxencia moi lenta, a valoración de opcións con exercicio anticipado e o cálculo das denominadas "gregas". Formulamos os correspondentes modelos de EDPs para os modelos de mercado SABR/LIBOR introducidos no Capítulo 3. As EDPs resultantes teñen alta dimensión en espazo. Co propósito de vencer a maldición da dimensión empregamos a denominada *sparse grid combination technique*.

A Parte II, constituída por un único capítulo, centrouse no estudo das ecuacións diferenciais estocásticas cara atrás, BSDEs.

No Capítulo 5 deseñamos un novo algoritmo baseado en Monte Carlo de Mínimos Cadrados para aproximar a solución de *Backward Stochastic Differential Equations* discretas en tempo. O noso algoritmo permite unha paralelización masiva das computacións en procesadores multinúcleo tales como GPUs. O enfoque proposto consiste nun novo método de estratificación, que foi crucial para permitir unha paralelización altamente escalable. Deste xeito, minimizamos o consumo de memoria principal debido ao almacenamento das simulacións. De feito, cabe destacar o menor consumo de memoria principal do algoritmo proposto en comparación co dos traballos previos.

O Apéndice A contén unha breve descrición dos diferentes problemas de optimización que foron testeados empregando a implementación proposta do algoritmo *Simulated Annealing*. No Apéndice B obtívose a expresión da volatilidade implícita do modelo SABR dinámico xeral. Ademais mostráronse os datos de mercado empregados no correspondente Capítulo 2. O Apéndice C contén dous resultados matemáticos relativos ao algoritmo proposto no Capítulo 5 para resolver BSDEs.

Finalizamos cunha breve sección que resume as principais conclusións deste traballo.

Bibliography

[1] S. Achatz. Higher order sparse grid methods for elliptic partial differential equations with variable coefficients. *Computing*, 71(1):1–15, 2003.

[2] D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer, Boston, 1987.

[3] L. Andersen and J. Andreasen. Jump diffusion processes: volatility smile fitting and numerical methods for option pricing. *Review of Derivatives Research*, 4:231–262, 2000.

[4] L. Andersen and J. Andreasen. Volatility skews and extension of the Libor Market Model. *Applied Mathematical Finance*, 7:1–32, 2000.

[5] L. Andersen and R. Brotherton-Ratcliffe. Extended LIBOR market models with stochastic volatility. *Journal of Computational Finance*, 9(1):1–40, 2005.

[6] A. Arciniega and E. Allen. Extrapolation of difference methods in option valuation. *Applied Mathematics and Computation*, 135:165–186, 2004.

[7] R. Bellmann. Adaptive control processes: A guided tour. *Princenton University Press*, 1961.

[8] D. Belomestny and J. G. M. Schoenmakers. A jump-diffusion Libor model and its robust calibration. *Quantitative Finance*, 11:529–546, 2011.

[9] C. Bender and J. Steiner. Least-squares Monte Carlo for BSDEs. In R. Carmona, P. Del Moral, P. Hu, and N. Oudjane, editors, *Numerical Methods in Finance*, volume 12, pages 257–289. Springer Proceedings in Mathematics, 2012.

[10] A. Bernemann and R. Schereyer. Accelerating Exotic Option Pricing and Model Calibration Using GPUs. *Social Science Research Network*, pages 1–19, 2011.

[11] H. Bersini, M. Dorigo, S. Langerman, G. Seront, and L. M. Gambardella. Results of the first international contest on evolutionary optimisation (1st ICEO). In *Proceedings of IEEE International Conference on Evolutionary Computation*, IEEE-EC 96, pages 611–615, New York, 1996. IEEE Press.

[12] G. Beylkin and M. J. Mohlenkamp. Algorithms for numerical analysis in high dimensions. *SIAM Journal on Scientific Computing*, 26(6):2133–2159, 2005.

[13] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81:637–654, 1973.

[14] W. M. Bolstad. *Understanding Computational Bayesian Statistics*. Wiley, New York, 2001.

[15] B. Bouchard and X. Warin. Monte-Carlo valuation of American options: facts and new algorithms to improve existing methods. In R. Carmona, P. Del Moral, P. Hu, and N. Oudjane, editors, *Numerical Methods in Finance*, volume 12, pages 215–255. Springer Proceedings in Mathematics, 2012.

[16] A. Brace, D. Gatarek, and M. Musiela. The Market model of interest rate dynamics. *Mathematical Finance*, 7(2):127–155, 1997.

[17] L. Breiman and A. Cutler. A deterministic algorithm for global optimization. *Mathematical Programming*, 58(1):179–199, 1993.

[18] P. Briand and C. Labart. Simulation of BSDEs by Wiener Chaos Expansion. *Annals of Applied Probability*, 24:1129–1171, 2014.

[19] D. Brigo and F. Mercurio. *Interest Rate Models - Theory and Practice. With Smile, Inflation and Credit*. Springer, second edition, 2007.

[20] S. B. Brooks and B. J. T. Morgan. Optimization using simulated annealing. *The Statistician*, 44(2):241–257, 1995.

[21] H. J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004.

[22] H. J. Bungartz, M. Griebel, D. Röschke, and C. Zenger. Pointwise convergence of the combination technique for the Laplace equation. *East-West Journal of Numerical Mathematics*, 2:21–45, 1994.

[23] H. J. Bungartz, M. Griebel, D. Röschke, and C. Zenger. A proof of convergence for the combination technique for the Laplace equation using tools of symbolic computation. *Mathematics and Computers in Simulation*, 41:595–605, 1996.

[24] T. Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press, 1996.

[25] B. Chen, L. A. Grzelak, and C. W. Oosterlee. Calibration and Monte Carlo Pricing of the SABR-Hull-White Model for Long-Maturity Equity Derivatives. *Journal of Computational Finance*, 15:79–113, 2012.

[26] B. Chen, C. W. Oosterlee, and H. van der Weide. A low-bias simulation scheme for the SABR stochastic volatility model. *International Journal of Theoretical and Applied Finance*, 15:1–27, 2012.

[27] D. J. Chen, C. Y. Lee, C. H. Park, and P. Mendes. Parallelizing simulated annealing algorithms based on high-performance computer. *Journal of Global Optimization*, 39:261–289, 2007.

[28] H. Chen, N. S. Flann, and D. W. Watson. Parallel genetic simulated annealing: A massively parallel SIMD algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9:126–136, 1998.

[29] J. C. Cox, J. E. Ingersoll, and S. A. Ross. A Theory of the Term Structure of Interest Rates. *Econometrica*, 53:385–407, 1985.

[30] J. C. Cox and S. A. Ross. The Valuation of Options for Alternative Stochastic Processes. *Journal of Financial Economics*, 3:145–166, 1976.

[31] S. Crépey. *Financial Modeling: A Backward Stochastic Differential Equations Perspective*. Springer, 2013.

[32] A. Debudaj-Grabysz and R. Rabenseifner. Nesting OpenMP in MPI to Implement a Hybrid Communication Method of Parallel Simulated Annealing on a Cluster of SMP Nodes. In *EuroPVM/MPI, Sorrento, September 19*, 2005.

[33] A. Dekkers and E. Aarts. Global optimization and simulated annealing. *Mathematical Programming*, 50(1):367–393, 1991.

[34] E. Derman and I. Kani. Riding on a smile. *Risk*, 7(2):32–39, 1994.

[35] L. Dixon and G. Szegö. *Towards Global Optimization*. North Holland, New York, 1978.

[36] D. J. Duffy. *Finite Difference methods in financial engineering. A Partial Differential Equation Approach*. Wiley Finance Series, 2006.

[37] B. Dupire. Pricing with a smile. *Risk*, 7:18–20, 1994.

[38] B. Dupire. Pricing and Hedging with Smiles. In M.A.H. Dempster and S.R. Pliska, editors, *Mathematics of Derivative Securities*, pages 103–111. Cambridge University Press, Cambridge, 1997.

[39] E. Eberlein, U. Keller, and K. Prause. New insights into smile, mispricing, and value at risk: the hyperbolic model. *Journal of Business*, 71(3):371–405, 1998.

[40] E. Eberlein and F. Ozkan. The Lèvy LIBOR model. *Finance and Stochastics*, 9:327–348, 2005.

[41] M. Fatica and E. Phillips. Pricing American options with least squares Monte Carlo on GPUs. In *Proceeding of the 6th Workshop on High Performance Computational Finance*, ACM, 2013.

[42] J. L. Fernández, A. M. Ferreiro, J. A. García, A. Leitao, J. G. López-Salas, and C. Vázquez. Static and dynamic SABR stochastic volatility models: calibration and option pricing using GPUs. *Mathematics and Computers in Simulation*, 94:55–75, 2013.

[43] A. M. Ferreiro, J. A. García, J. G. López-Salas, and C. Vázquez. An efficient implementation of parallel simulated annealing algorithm in GPUs. *Journal of Global Optimization*, 57(3):863–890, 2013.

[44] A. M. Ferreiro, J. A. García, J. G. López-Salas, and C. Vázquez. SABR/LIBOR market models: Pricing and calibration for some interest rate derivatives. *Applied Mathematics and Computation*, 242:65–89, 2014.

[45] D. Funaro. Polynomial approximation of differential equations. In *Lecture Notes in Physics, New Series m: Monographs*, volume 8. Springer-Verlag, Berlin, 1992.

[46] A. Gaikwad and I. M. Toke. GPU Based Sparse Grid Technique for Solving Multidimensional Options Pricing PDEs. In *Proceedings of the 2nd Workshop on High Performance Computational Finance*, WHPCF '09, pages 6:1–6:9, New York, NY, USA, 2009. ACM.

[47] J. Gatheral. *The Volatility Surface: A Practitioner's Guide*. Wiley Finance, 2006.

[48] T. Gerstner and P. Kloeden, editors. *Recent Developments in Computational Finance. Foundations, Algorithms and Applications*. World Scientific Publishers, Interdisciplinary Mathematical Science, 2013.

[49] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York, 2003.

[50] P. Glasserman and S. Kou. The term structure of simple forward rates with jump risk. *Mathematical Finance*, 13(3):383–410, 2003.

[51] P. Glasserman and N. Merener. Numerical solution of jump-diffusion LIBOR market models. *Journal of Computational Finance*, 7(1):1–27, 2003.

[52] P. Glasserman and Q. Wu. Forward and Future Implied Volatility. *International Journal of Theoretical and Applied Finance*, 14:407–432, 2011.

[53] E. Gobet. Advanced Monte Carlo methods for barrier and related exotic options. *Mathematical Modeling and Numerical Methods in Finance*, pages 497–528, 2009.

[54] E. Gobet and C. Labart. Solving BSDE with adaptive control variate. *SIAM Numerical Analysis*, 48:257–277, 2010.

[55] E. Gobet, J. P. Lemor, and X. Warin. A regression-based Monte Carlo method to solve backward stochastic differential equations. *Annals of Applied Probability*, 15:2172–2202, 2005.

[56] E. Gobet, J. G. López-Salas, P. Turkedjiev, and C. Vázquez. Stratified Regression Monte-Carlo Scheme for Semilinear PDEs and BSDEs with Large Scale Parallelization on GPUs. *Accepted for publication in SIAM Journal on Scientific Computing*, 2016.

[57] E. Gobet and A. Makhlouf. L_2-time regularity of BSDEs with irregular terminal functions. *Stochastic Processes and their Applications*, 120:1105–1132, 2010.

[58] E. Gobet and P. Turkedjiev. Adaptive importance sampling in least-squares Monte Carlo algorithms for backward stochastic differential equations. *Accepted for publication in Stochastic Processes and Applications*, 2015.

[59] E. Gobet and P. Turkedjiev. Approximation of BSDEs using Malliavin weights and least-squares regression. *Bernoulli*, 22(1):530–562, 2016.

[60] E. Gobet and P. Turkedjiev. Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions. *Mathematics of Computation*, 85(299):1359–1391, 2016.

[61] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[62] M. Griebel. The combination technique for sparse grids solution of PDEs on multiprocessor machines. *Parallel Processing Letters*, 3:66–71, 1992.

[63] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *Proceedings of the IMACS International Symposium on Iterative Methods in Linear Albegra*, pages 263–281, Brussels, April 1991. Elsevier, Amsterdam, 1992.

[64] M. Griebel and V. Thurner. The efficient solution of fluid dynamics problems by the combination technique. *International Journal of Numerical Methods for Heat and Fluid Flow*, 5(3):251–269, 1995.

[65] A. O. Griewank. General descent for global optimization. *Journal Optimization Theory Applications*, 34:11–39, 1981.

[66] L. Györfi, M. Kohler, A. Krzyżak, and H. Walk. *A distribution-free theory of nonparametric regression*. Springer Series in Statistics, Springer-Verlag, New York, 2002.

[67] P. Hagan, D. Kumar, A. Lesniewski, and D. Woodward. Managing Smile Risk. *Wilmott Magazine*, pages 84–108, 2002.

[68] P. Hagan and A. Lesniewski. LIBOR market model with SABR style stochastic volatility. *Working paper, available at* `http://lesniewski.us/papers/working/SABRLMM.pdf` *(2008)*, 2008.

[69] D. Heath, R. Jarrow, and A. Morton. Bond Pricing and the Term Structure of Interest Rates: A New Methodology. *Econometrica*, 60:77–105, 1992.

[70] P. Henry-Labordère. Unifying the BGM and SABR Models: A Short ride in Hyperbolic Geometry. *SSRN, available at* `http://papers.ssrn.com/sol3/papers.cfm?abstract_id=877762` *(2006)*, 2006.

[71] P. Henry-Labordère. Combining the SABR and LMM models. *Risk (2007)*, 2007.

[72] P. Henry-Labordère. *Analysis, Geometry and Modeling in Finance: Advanced Methods in Option Pricing*. Financial Mathematics Series, Boca Raton, FL, New York, Chapman & Hall, 2008.

[73] S. Heston. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *The Review of Financial Studies*, 6:327–343, 1993.

[74] D. M. HimmelBlau. *Applied Linear Programming*. McGraw-Hill, New York, 1972.

[75] T. S. Y. Ho and S. B. Lee. Term Structure Movements and the Pricing of Interest Rate Contingent Claims. *The Journal of Finance*, 41:1011–1029, 1986.

[76] C. Homescu. Implied volatility surface: construction methodologies and characteristics. `http://arxiv.org/abs/1107.1834`, 2011.

[77] J. Hull and A. White. The Pricing of Options on Assets with Stochastic Volatilities. *The Journal of Finance*, 42(2):281–300, 1987.

[78] J. Hull and A. White. An Analysis of the Bias in Option Pricing Caused by a Stochastic Volatility. *Advances in Futures and Options Research*, 3:27–61, 1988.

[79] F. Jamshidian. LIBOR and swap market models and measures. *Finance and Stochastic*, 1:293–330, 1997.

[80] F. Jamshidian. Libor market model with semimartigales. In E. Jouini, J. Cvitanić, and Marek Musiela, editors, *Option pricing, interest rates and risk management*. Cambridge University Press, 2001.

[81] K. A. De Jong. *An analysis of the behavior of a glass of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, 1975.

[82] M. Joshi and R. Rebonato. A displaced-diffusion stochastic volatility LIBOR market model: motivation, definition and implementation. *Quantitative Finance*, 3(6):458–469, 2003.

[83] M. S. Joshi. Graphical Asian Options. *Wilmott Journal*, 2(2):97–107, 2010.

[84] P. Jäckel. *Monte Carlo Methods in Finance*. Wiley Finance, 2002.

[85] L. Kaisajuntti and J. Kennedy. Stochastic Volatility for Interest Rate Derivatives. *http://papers.ssrn.com/sol3/papers.cfm?abstract_id=189488*, 2011.

[86] N. El Karoui, S. Peng, and M. C. Quenez. Backward stochastic differential equations in finance. *Mathematical Finance*, 7(1):1–71, 1997.

[87] J. Kienitz and M. Wittke. Option Valuation in Multivariate SABR Models. *Research Paper Quantitative Finance Research Centre, University of Technology Sydney*, 272:1–24, 2010.

[88] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[89] A. Kliewer and S. Tschöke. A General Parallel Simulated Annealing Library and Its Applications in Industry. *Working paper, Department of Mathematics and Computer Science, University of Paderborn, Germany*, 1998.

[90] L. Abbas-Turki and B. Lapeyre. American options pricing on multi-core graphic cards. In *Business Intelligence and Financial Engineering*, BIFE'09, pages 307–311. IEEE, 2009.

[91] L. Abbas-Turki and S. Vialle and B. Lapeyre and P. Mercier. High dimensional pricing of exotic European contracts on a GPU cluster, and comparison to a CPU cluster. In *Parallel & Distributed Processing*, IPDPS 2009, pages 1–8. IEEE, 2009.

[92] C. Labart and J. Lelong. A parallel algorithm for solving BSDEs. *Monte Carlo Methods and Applications*, 19:11–39, 2013.

[93] J. W. Larson, M. Hegland, B. Harding, S. Roberts, L. Stals, A. P. Rendell, P. Strazdins, M. M. Ali, C. Kowitz, R. Nobes, J. Southern, N. Wilson, M. Li, and Y. Oishi. 2013 International Conference on Computational Science: Fault-Tolerant Grid-Based Solvers: Combining Concepts from Sparse Grids and MapReduce. *Procedia Computer Science*, 18:130–139, 2013.

[94] K. Larsson. Dynamic extensions and probabilistic expansions of the SABR model. *http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1536471*, 2010.

[95] A. Lee, C. Yau, M. B. Giles, A. Doucet, and Ch. C. Holmes. On the utility of graphic cards to perform massively parallel simulation of advanced Monte Carlo methods. *Journal of Computational and Graphical Statistics*, 19(4):769–789, 2010.

[96] S. Y. Lee and K. G. Lee. Synchronous and asynchronous parallel simulated annealing with multiple Markov Chains. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):993–1008, 1996.

[97] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing

on CPU and GPU. *Newsletter ACM SIGARCH Computer Architecture News - ISCA '10*, 38(3):451–460, 2010.

[98] 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.

[99] A. V. Levy and A. Montalvo. The tunneling algorithm for the global minimization of functions. *SIAM Journal on Scientific and Statistical Computing*, 6:15–29, 1985.

[100] F. A. Longstaff and E. S. Schwartz. Valuing american options by simulation: A simple least-squares approach. *The Review of Financial Studies*, 14(1):113–147, 2001.

[101] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(4):1–6, 2003.

[102] D. L. McLeish. *Monte Carlo Simulation and Finance*. Wiley, 2005.

[103] F. Mercurio and M. Morini. No-Arbitrage dynamics for a tractable SABR term structure Libor Model. *Modeling Interest Rates: Advances in Derivatives Pricing, Risk Books (2009)*, 2009.

[104] R. C. Merton. Theory of rational option pricing. *Bell Journal of Economics and Management Science*, 4:141–183, 1973.

[105] R. C. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, pages 125–144, 1976.

[106] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[107] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1998.

[108] K. Miltersen, K. Sandmann, and D. Sondermann. Closed-form solutions for term structure derivatives with lognormal interest rates. *Journal of Finance*, 52(1):409–430, 1997.

[109] Nvidia Corporation. *CUDA C Programming guide*.

[110] Nvidia Corporation. *CURAND Library*.

[111] Nvidia Corporation. *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210*.

[112] Nvidia Corporation. *Whitepaper. NVIDIA's Next Generation CUDA Compute Architecture: Fermi*.

[113] J. Oblój. Fine-Tune your smile: Correction to Hagan et al. *Wilmott Magazine*, 2008.

[114] E. Onbasoglu and L. Özdamar. Parallel Simulated Annealing Algorithms in Global Optimization. *Journal of Global Optimization*, 19:27–50, 2001.

[115] Y. Osajima. The asymptotic expansion formula of implied volatility for dynamic SABR Model and FX Hybrid Model. *Capital markets: asset pricing and valuation e-Journal, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=965265*, 2007.

[116] L. Paulot. Asymptotic implied volatility at second order with application to the SABR model. *http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1413649*, 2009.

[117] V. Piterbarg. Pricing and Hedging Callable Libor Exotics in Forward Libor Models. *The Journal of Computational Finance*, 8(2):65–119, 2005.

[118] V. Piterbarg. Stochastic Volatility Model with Time-dependent Skew. *Applied Mathematical Finance*, 12(2):147–185, 2005.

[119] R. Rebonato. *Modern pricing of interest-rate derivatives: the Libor market model and beyond*. Princeton University Press, 2002.

[120] R. Rebonato. A time-homogeneous SABR-consistent extension of the LMM. *Risk (2008)*, 2008.

[121] R. Rebonato, K. Mckay, and R. White. *The SABR/LIBOR Market Model. Pricing, Calibration and Hedging for Complex Interest-Rate Derivatives*. John Wiley & Sons, first edition, 2009.

[122] R. Rebonato and R. White. Linking caplets and swaptions prices in the LMM-SABR model. *The Journal of Computational Finance*, 13(2):19–45, 2009.

[123] C. Reisinger. Analysis of linear difference schemes in the sparse grid combination technique. *IMA Journal of Numerical Analysis*, 33(2):544–581, 2013.

[124] M. J. Ruijter and C. W. Oosterlee. A Fourier Cosine Method for an Efficient Computation of Solutions to BSDEs. *SIAM Journal on Scientific Computing*, 37(2):A859–A889, 2015.

[125] R. Salomon. Reevaluating genetic algorithms performance under coordinate rotation of benchmark functions. *BioSystems*, 39(3):263–278, 1995.

[126] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley, Michigan, 2011.

[127] S. E. Shreve. *Stochastic Calculus for Finance*. Springer, 2004.

[128] S. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Dokl. Akad. Nauk SSR*, 148:1042–1045, 1963.

[129] R. Storn and K. Price. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[130] A. Takahashi, A. Takehara, and M. Toda. Computation in an asymptotic expansion method. *http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1413924*, 2009.

[131] Y. Tian, Z. Zhu, F. C. Klebaner, and K. Hamza. Option pricing with the SABR model on the GPU. *High Performance Computational Finance (WHPCF) IEEE Workshop*, 14:1–8, 2010.

[132] Y. Tian, Z. Zhu, F. C. Klebaner, and K. Hamza. Pricing barrier and American options under the SABR model on the graphics processing unit. *Concurrency and Computation: Practice and Experience*, pages 1–13, 2011.

[133] P. Turkedjiev. Two algorithms for the discrete time approximation of Markovian backward stochastic differential equations under local conditions. *Electronic Journal of Probability*, 20, 2015.

[134] A. Törn and A. Ẑilinskas. Global Optimization. In *Lecture Notes in Computer Science*, volume 350, Berlin, 1996. Springer.

[135] P. J. M van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer, Dordrecht, 1987.

[136] O. Vasicek. An Equilibrium Characterization of the Term Structure. *Journal of Financial Economics*, 5:177–188, 1977.

[137] G. West. Calibration of the SABR Model in Illiquid Markets. *Applied Mathematical Finance*, 12:371–385, 2005.

[138] P. Wilmott. Cliquet Options and Volatility Models. *Wilmott Magazine*, pages 78–83, 2002.

[139] P. Wilmott. *Paul Wilmott on Quantitative Finance*. John Wiley & Sons, second edition, 2006.

[140] L. Wu and F. Zhang. LIBOR market model with stochastic volatility. *Journal of Industrial and Management Optimization*, 2:199–207, 2006.

[141] H. Yserentant. On the multi-level splitting of finite element spaces. *Numerische Mathematik*, 49:379–412, 1986.

[142] H. Yserentant. Hierarchical bases. In *Proceedings of the Second International Conference on Industrial and Applied Mathematics*, ICIAM 91, pages 256–276, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

[143] A. Zeiser. Fast Matrix-Vector Multiplication in the Sparse-Grid Galerkin Method. *Journal of Scientific Computing*, 47(3):328–346, 2010.

[144] C. Zenger. Sparse grids. In W. Hackbusch, editor, *Parallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar*, volume 31, pages 241–251, Kiel, Germany, 1990. Vieweg-Verlag, 1991.

[145] J. Zhang. A numerical scheme for BSDEs. *The Annals of Applied Probability*, 14:459–488, 2004.

[146] W. Zhao, L. Chen, and S. Peng. A new kind of accurate numerical method for backward stochastic differential equations. *SIAM Journal on Scientific Computing*, 28(4):1563–1581, 2006.

[147] Thrust Library web page: `https://github.com/thrust/thrust/wiki/Documentation`.

[148] MPI web page: `http://www.mpi-forum.org`.

[149] OpenMP web page: `http://www.openmp.org`.

[150] `http://www.it.lut.fi/ip/evo/functions/node10.html`.

[151] `http://www.top500.org/list/2012/06/100`.

[152] CUSIMANN web pages: http://gforge.i-math.cesga.es/projects/cusimann/ or http://code.google.com/p/cusimann/.