

Static and dynamic SABR stochastic volatility models: calibration and option pricing using GPUs *

J.L. Fernández^a, A.M. Ferreiro^b, J.A. García^b, A. Leitao^b, J.G. López-Salas^b, C. Vázquez^b

^a*Department of Mathematics, Universidad Autónoma de Madrid,
Francisco Tomás y Valiente, 7, 28049 – Madrid (Spain)*

^b*Departament of Mathematics, University of A Coruña,
Campus Elviña s/n, 15071 – A Coruña (Spain)*

Abstract

For the calibration of the parameters in static and dynamic SABR stochastic volatility models, we propose 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 propose an original more general expression for the functional parameters, specially well suited for the EUR/USD exchange rate case. Numerical results illustrate the expected behavior of both 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. Furthermore, GPU technology allows the use of Monte Carlo simulation for calibration purposes, the computational time with CPU being prohibitive.

Keywords: Parallel Simulated Annealing, SABR volatility model, Calibration, GPUs, CUDA.

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 [2], 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

*Partially financed by MCINN (Project MTM2010–21135–C02-01) and Ayuda CN2011/004 partially funded with FEDER funds). Also the authors would like to thank M. Menéndez, from Banesto Bank, for providing the market data.

Email addresses: joseluis.fernandez@uam.es (J.L. Fernández), aferreiro@udc.es (A.M. Ferreiro), jagrodriguez@udc.es (J.A. García), alvaro.leitao@udc.es (A. Leitao), jose.lsalamas@udc.es (J.G. López-Salas), carlosv@udc.es (C. Vázquez)

and strike (volatility surface). In order to overcome this problem, different local and stochastic volatility models have been introduced (see [6, 7, 12, 14, 16], for example). Here we consider the SABR model first proposed in [12], where a first order approximation formula for the implied volatility of European plain-vanilla options with short maturities is obtained. In [24] this formula is improved. In [23] 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 [27] a fifth order asymptotic expansion is proposed for λ -SABR model; thus providing an extension with a mean-reversion term in [13].

The existence of 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 [12, 30], 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 [9, 15], for example), in [12] a SABR model with time dependent parameters (dynamic SABR) is introduced and in [25] an asymptotic expression for the implied volatility is obtained. Also by means of piecewise constant parameters, in [11] the static SABR model is extended. In [20] 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 [19] the SABR model with a time-dependent volatility function and a mean reverting volatility process is obtained. In [4] 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 [8]. Particularly, we consider Nvidia Fermi GPUs and the API for its programming, named CUDA (Compute Unified Device Architecture), see [26, 34]. 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 [10, 17]). 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 paper, Monte Carlo simulation is suitable for pricing options on GPUs [21].

In this work, 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 [18] Asian options pricing with Black-Scholes model by a quasi-Monte Carlo method is considered, thus getting a speedup factor up to 150. In [1] 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 [28], 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 [29] 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 [28]. They obtain speedups up to 134 in the case of barrier options and around 22 in American ones. In our present work, the random number generation is performed “on the fly” by using the Nvidia CURAND library (see [34], 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 paper is as follows. In Section 2 the static and dynamic SABR models are described, with a new proposal for functional parameters in the dynamic case. In Section 3 the calibration to market data is detailed. In Section 4, the Monte Carlo method and its parallel implementation in CUDA is described. In this work 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 5 we discuss different techniques to parallelize the SA algorithm. In Section 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.

2. The SABR model

The SABR (*Stochastic α, β, ρ*) model was introduced in [12], arguing that local volatility models could not reproduce market volatility smiles and that their predicted volatility dynamics contradicts market smiles and skews. The main advantage of the SABR model comes from its great simplicity compared to alternative stochastic volatility models [12]. The dynamics of the forward price and its volatility satisfy the system of stochastic differential equations

$$dF_t = \alpha_t F_t^\beta dW_t^1, \quad F_0 = \hat{f}, \quad (1)$$

$$d\alpha_t = \nu \alpha_t dW_t^2, \quad \alpha_0 = \alpha, \quad (2)$$

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, α_t denotes the asset volatility process, dW^1 and dW^2 are two correlated Brownian motions with constant correlation coefficient ρ (i.e. $dW_t^1 dW_t^2 = \rho dt$) 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).

2.1. 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 [12]. The great advantage is that

the following asymptotically approximated explicit formula for the implied Black-Scholes volatility can be obtained:

$$\begin{aligned}\sigma_{model}(K, \hat{f}, T) = & \frac{\alpha}{(K\hat{f})^{(1-\beta)/2} \left[1 + \frac{(1-\beta)^2}{24} \ln^2 \left(\frac{\hat{f}}{K} \right) + \frac{(1-\beta)^4}{1920} \ln^4 \left(\frac{\hat{f}}{K} \right) + \dots \right]} \cdot \left(\frac{z}{x(z)} \right). \\ & \left\{ 1 + \left[\frac{(1-\beta)^2}{24} \frac{\alpha^2}{(K\hat{f})^{1-\beta}} + \frac{1}{4} \frac{\rho\beta\nu\alpha}{(K\hat{f})^{(1-\beta)/2}} + \frac{2-3\rho^2}{24} \nu^2 \right] \cdot T + \dots \right\},\end{aligned}\quad (3)$$

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

$$x(z) = \ln \left(\frac{\sqrt{1-2\rho z+z^2}+z-\rho}{1-\rho} \right). \quad (4)$$

In this work, we consider the following correction to (3) proposed by Obłój in [24],

$$\begin{aligned}\sigma_{model}(K, \hat{f}, T) = & \frac{1}{\left[1 + \frac{(1-\beta)^2}{24} \ln^2 \left(\frac{\hat{f}}{K} \right) + \frac{(1-\beta)^4}{1920} \ln^4 \left(\frac{\hat{f}}{K} \right) + \dots \right]} \cdot \left(\frac{\nu \ln \left(\frac{\hat{f}}{K} \right)}{x(z)} \right). \\ & \left\{ 1 + \left[\frac{(1-\beta)^2}{24} \frac{\alpha^2}{(K\hat{f})^{1-\beta}} + \frac{1}{4} \frac{\rho\beta\nu\alpha}{(K\hat{f})^{(1-\beta)/2}} + \frac{2-3\rho^2}{24} \nu^2 \right] \cdot T + \dots \right\},\end{aligned}\quad (5)$$

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 (4). The omitted terms after $+ \dots$ can be neglected, so that (5) turns to

$$\sigma_{model}(K, \hat{f}, T) = \frac{1}{\omega} \left(1 + A_1 \ln \left(\frac{K}{\hat{f}} \right) + A_2 \ln^2 \left(\frac{K}{\hat{f}} \right) + BT \right), \quad (6)$$

where the coefficients A_1 , A_2 and B are given by

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

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

2.2. 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 can appear. In order to overcome this problem, the following dynamic SABR model allows time dependency in some parameters [12]:

$$dF_t = \alpha_t F_t^\beta dW_t^1, \quad F_0 = \hat{f}, \quad (7)$$

$$d\alpha_t = \nu(t) \alpha_t dW_t^2, \quad \alpha_0 = \alpha, \quad (8)$$

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 [25]:

$$\sigma_{model}(K, \hat{f}, T) = \frac{1}{\omega} \left(1 + A_1(T) \ln \left(\frac{K}{\hat{f}} \right) + A_2(T) \ln^2 \left(\frac{K}{\hat{f}} \right) + B(T)T \right), \quad (9)$$

where

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

with

$$\begin{aligned} \nu_1^2(T) &= \frac{3}{T^3} \int_0^T (T-t)^2 \nu^2(t) dt, & \nu_2^2(T) &= \frac{6}{T^3} \int_0^T (T-t)t \nu^2(t) dt, \\ \eta_1(T) &= \frac{2}{T^2} \int_0^T (T-t)\nu(t)\rho(t) dt, & \eta_2^2(T) &= \frac{12}{T^4} \int_0^T \int_0^t \left(\int_0^s \nu(u)\rho(u) du \right)^2 ds dt. \end{aligned} \quad (10)$$

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 (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 work we consider two possibilities with exponential decay:

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

$$\rho(t) = \rho_0 e^{-at}, \quad \nu(t) = \nu_0 e^{-bt}, \quad (11)$$

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 (10), can be exactly calculated and are given by:

$$\begin{aligned} \nu_1^2(T) &= \frac{6\nu_0^2}{(2bT)^3} \left[((2bT)^2/2 - 2bT + 1) - e^{-2bT} \right], \\ \nu_2^2(T) &= \frac{6\nu_0^2}{(2bT)^3} \left[2(e^{-2bT} - 1) + 2bT(e^{-2bT} + 1) \right], \\ \eta_1(T) &= \frac{2\nu_0\rho_0}{T^2(a+b)^2} \left[e^{-(a+b)T} - (1 - (a+b)T) \right], \\ \eta_2^2(T) &= \frac{3\nu_0^2\rho_0^2}{T^4(a+b)^4} \left[e^{-2(a+b)T} - 8e^{-(a+b)T} + \left(7 + 2(a+b)T(-3 + (a+b)T) \right) \right]. \end{aligned} \quad (12)$$

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

$$\rho(t) = (\rho_0 + q_\rho t)e^{-at} + d_\rho, \quad \nu(t) = (\nu_0 + q_\nu t)e^{-bt} + d_\nu. \quad (13)$$

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 [Appendix A](#)). 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.

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 ($i = 1, \dots, n$) and the strike K_j ($j = 1, \dots, m_i$), 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

$$f_{i,E}(\boldsymbol{x}) = \sum_{j=1}^{m_i} \left(\frac{Data_{market}(K_j, \hat{f}, T_i) - Data_{model}(K_j, \hat{f}, T_i)}{Data_{market}(K_j, \hat{f}, T_i)} \right)^2 (\boldsymbol{x}), \quad (14)$$

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

$$f_E(\boldsymbol{x}) = \sum_{i=1}^n \sum_{j=1}^{m_i} \left(\frac{Data_{market}(K_j, \hat{f}, T_i) - Data_{model}(K_j, \hat{f}, T_i)}{Data_{market}(K_j, \hat{f}, T_i)} \right)^2 (\boldsymbol{x}). \quad (15)$$

- 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 [12, 30]. 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 (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 [30], for details). Once β has been fixed and the term BT in (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 [12]. Thus, by fixing $\beta = 1$ and α , the (possibly large) calibration cost is reduced: only ρ and ν are calibrated.

In this work 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.

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. 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 $V(S_0, K) = e^{-rT} \mathbb{E}(V(S_T, K))$, 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:

$$\begin{aligned} \alpha_{i+1} &= \alpha_i e^{\nu(t_i) Z_i^1 \sqrt{\Delta t} - \nu^2(t_i) \Delta t / 2}, \\ \hat{\nu} &= \alpha_i F_i^{\beta-1}, \\ F_{i+1} &= F_i e^{\hat{\nu} (\rho(t_i) Z_i^1 + Z_i^2 \sqrt{1 - \rho^2(t_i)}) \sqrt{\Delta t} - \hat{\nu}^2 \Delta t / 2}, \end{aligned} \tag{16}$$

where Z_i^1 and Z_i^2 denote two independent standard normal distribution samples. In [5] 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 instability, this alternative low-bias simulation scheme in [5] could be adapted.

Note that $S_T = F_T$. If N denotes the number of simulated paths, then the option price is given by

$$\widehat{V}(S_0, K) = e^{-rT} \frac{1}{N} \sum_{j=1}^N V(S_T^j, K). \quad (17)$$

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 [22].

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 [36] (included in the Nvidia toolkit since its version 4.0). With this function we apply kernel fusion reduction kernels. Then, the sum in (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.

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 work, the calibration of the parameters has been done with a Simulated Annealing (SA) stochastic global optimization method. The SA algorithm consists of a temperature loop (see [3] for details), where the equilibrium state at each temperature is computed using the Metropolis algorithm.

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

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 (14) and (15), respectively, both for individual and joint calibrations. Note that σ_{market} denotes the market quoted implied volatilities and σ_{model} the model ones, (6) and (9) for the static and dynamic models, respectively.

As it is well known in the literature, SA involves a great computational cost. For this reason, the calibration of the parameters has been done using the CUSIMANN library [32], that is a parallel implementation of the SA algorithm proposed in [8]; specifically the so called synchronous implementation. In [8], authors have analyzed different parallel SA implementations and have concluded that the synchronous version is the most appropriated for the GPU technology.

The idea of this synchronous parallel version of SA is the following: the algorithm starts from a fixed initial solution \mathbf{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 \mathbf{x}^p and the value $f(\mathbf{x}^p)$, $p = 0, \dots, w - 1$ (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^* then \mathbf{x}^{p^*} is used as starting point for all threads at the following temperature level. In [8] the implementation on CUDA of this parallel version is detailed.

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 [35] 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 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 \mathbf{x}_*^{\min} and \mathbf{y}_*^{\min} , and in the CPU another reduction operation is done to get the minimum point \mathbf{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 the MPI [33] for intercommunication between the nodes of a GPU cluster and while to use OpenMP inside the node to control several graphic cards. 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.

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

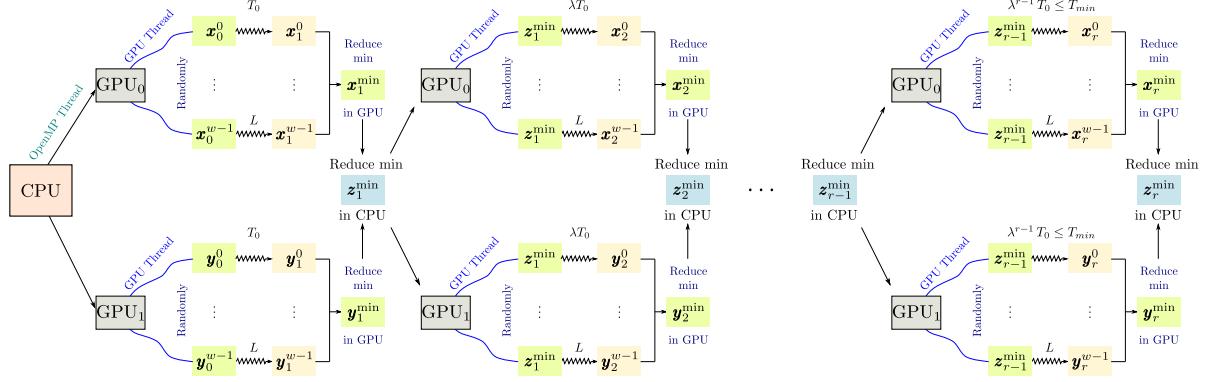


Figure 1: Sketch of the parallel SA algorithm using two GPUs and OpenMP.

model when considering the **Case II** (see Section 2.2). For the joint calibration, in the cost function (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 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 $\nu(t) > 0, \forall t 0 < t \leq T$.

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 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 (11).
- DSabr_II: dynamic SABR model in the **Case II** (general case), where ρ and ν are given by (13).

Moreover, depending on the technique used to calibrate the model (see Section 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_III: **Technique II**, where the cost function is evaluated by the efficient implementation in GPUs of the Monte Carlo pricing method, as detailed in Section 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.

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 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 the execution times for CPU and 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 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 1: Pricing results for European options. RE denotes the relative error with respect to the reference value 225.887329.

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.

6.2.1. EURO STOXX 50 index

The data corresponds to EURO STOXX 50 quotes of December of 2011. The asset spot value is 2311.1 €. In Tables B.18 and B.19 of the Appendix [Appendix B](#), 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 (6) in the cost function, the individual calibration (all strikes and one maturity) of the SSabr model has been carried out. In Table 4 the

SSabr						
Paths	Single			Double		
	CPU	GPU	Speedup	CPU	GPU	Speedup
2^{16}	2.890	0.142	$\times 20.313$	2.762	0.175	$\times 15.783$
2^{20}	46.287	0.206	$\times 223.726$	44.142	0.723	$\times 61.054$
2^{24}	721.592	1.271	$\times 567.373$	704.617	9.474	$\times 74.373$
DSabr_I						
Paths	Single			Double		
	CPU	GPU	Speedup	CPU	GPU	Speedup
2^{16}	3.638	0.144	$\times 25.112$	3.358	0.179	$\times 18.759$
2^{20}	58.228	0.255	$\times 227.674$	53.663	0.840	$\times 63.884$
2^{24}	929.254	2.047	$\times 453.918$	860.556	11.515	$\times 74.733$
DSabr_II						
Paths	Single			Double		
	CPU	GPU	Speedup	CPU	GPU	Speedup
2^{16}	3.694	0.145	$\times 25.388$	3.392	0.182	$\times 18.637$
2^{20}	59.790	0.287	$\times 207.762$	54.144	0.853	$\times 64.474$
2^{24}	949.459	2.658	$\times 357.095$	868.156	11.590	$\times 74.905$

Table 2: Pricing European options. Execution times for CPU and GPU calibration (in seconds), considering single and double precision with $\Delta t = 1/250$.

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 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 $\{80, 81 \dots, 119, 120\}$, respectively.

	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 4: EURO STOXX 50. SSabr model: Calibrated parameters for each maturity.

calibrated parameters are detailed. For each maturity (3, 6, 12 and 24 months), Figure 2 shows market and model volatilities with the parameters of Table 4.

Table 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 5: EURO STOXX 50. SSabr model: Performance of OpenMP vs. GPU versions, in single precision for $T = 24$ months.

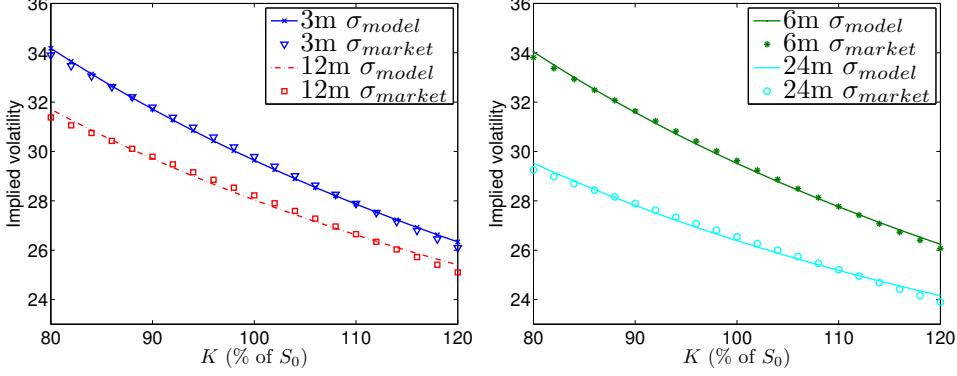


Figure 2: EURO STOXX 50. SSabre model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

2. Calibration of the DSabre-I model using the technique T-I

The joint calibration for all strikes and maturities, using the DSabre-I model is made with the GPU version of SA and using the asymptotic formula (9), when ρ and ν are given by (11). In Table 6 the calibrated parameters are detailed. In Figure 3, the whole volatility surface for all maturities is shown. By using the parameters in Table 6, for several strikes. Table 7 shows the market volatilities (σ_{market}) vs. the model ones (σ_{model} , computed with formulas (9), (11) and (12)). For single precision, the maximum relative error is 7.608205×10^{-2} and the mean relative error is 2.073025×10^{-2} .

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

Table 6: EURO STOXX 50. DSabre-I model: Calibrated parameters.

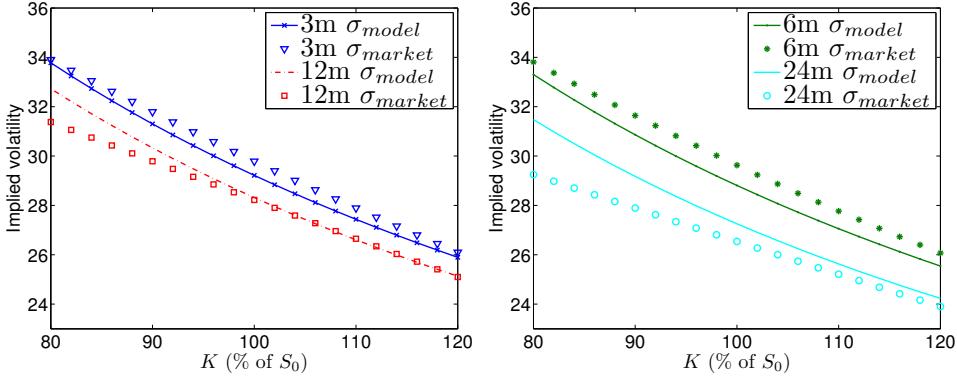


Figure 3: EURO STOXX 50. DSabre-I model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

In Table 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 DSabre-II model using the technique T-II

K (% of S_0)	3 months			6 months		
	σ_{market}	σ_{model}	$\frac{ \sigma_{market} - \sigma_{model} }{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{ \sigma_{market} - \sigma_{model} }{\sigma_{market}}$
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}	$\frac{ \sigma_{market} - \sigma_{model} }{\sigma_{market}}$	σ_{market}	σ_{model}	$\frac{ \sigma_{market} - \sigma_{model} }{\sigma_{market}}$
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 7: EURO STOXX 50. DSabr_I model: σ_{market} vs. σ_{model} .

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

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 5). The calibration process is performed in prices. For the Monte Carlo pricing method we have considered 2^{20} paths with $\Delta t = 1/250$. In Table 9 the calibrated parameters are detailed. In this case, double precision computations have been used to ensure the convergence to the real minimum.

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

Table 9: EURO STOXX 50. DSabr_II model: Calibrated parameters.

In Figure 4, the whole prices surface at maturities 3, 6, 12 and 24 months is shown. In Table 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} .

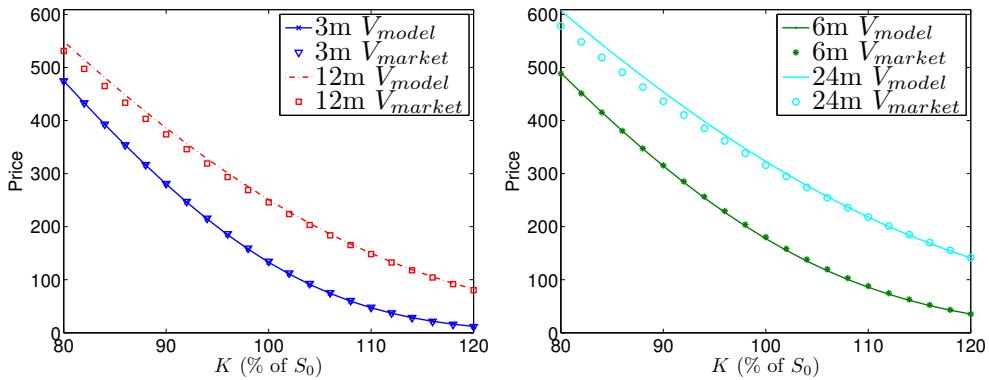


Figure 4: EURO STOXX 50. DSabr_II model: V_{model} vs. V_{market} for the whole prices surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

K (% of S_0)	3 months			6 months		
	V_{market}	V_{model}	$\frac{ V_{market} - V_{model} }{V_{market}}$	V_{market}	V_{model}	$\frac{ V_{market} - V_{model} }{V_{market}}$
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}	$\frac{ V_{market} - V_{model} }{V_{market}}$	V_{market}	V_{model}	$\frac{ V_{market} - V_{model} }{V_{market}}$
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 10: EURO STOXX 50. `DSabr_II` model: V_{market} vs. V_{model} .

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 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.

6.2.2. 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.20 and B.21 of the Appendix [Appendix B](#) 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 (6), the individual calibration of the `SSabr` model has been carried out. The parameters in Table 11 have been obtained. For them, in Figure 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 11: EURUSD. `SSabr` model: Calibrated parameters for each maturity.

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

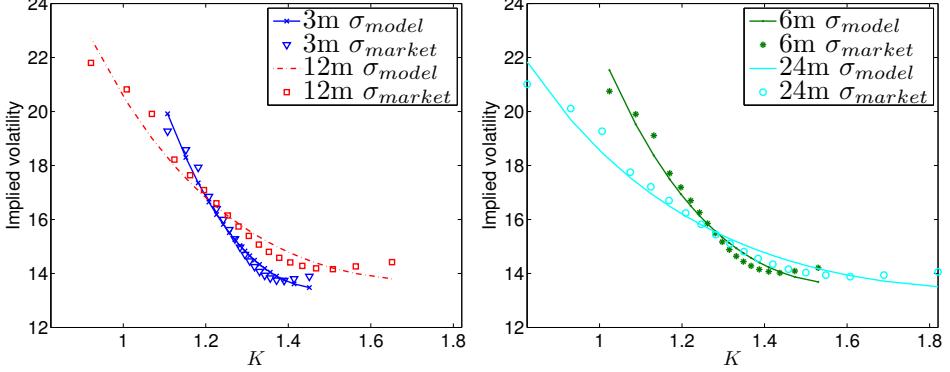


Figure 5: EURUSD. SSabre model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

	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 12: EURUSD. SSabre model: Performance of OpenMP vs. GPU versions, in single precision for $T = 24$ months.

2. Calibration of the DSabre_I model using the technique T_I

DSabre_I model calibration to all strikes and maturities has been performed with the SA GPU version and formula (9), with ρ and ν given by (11). In Table 13 the calibrated parameters are detailed. In Figure 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 14, the market volatilities vs. the model ones (9) are shown. The mean relative error is 2.441714×10^{-2} and the maximum relative error is 6.954307×10^{-2} . In Table 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.

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

Table 13: EURUSD. DSabre_I model: Calibrated parameters.

3 months				6 months			
K	σ_{market}	σ_{model}	$\frac{ \sigma_{market} - \sigma_{model} }{\sigma_{market}}$	K	σ_{market}	σ_{model}	$\frac{ \sigma_{market} - \sigma_{model} }{\sigma_{market}}$
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}	$\frac{ \sigma_{market} - \sigma_{model} }{\sigma_{market}}$	K	σ_{market}	σ_{model}	$\frac{ \sigma_{market} - \sigma_{model} }{\sigma_{market}}$
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 14: EURUSD. DSabre_I model: σ_{market} vs. σ_{model} .

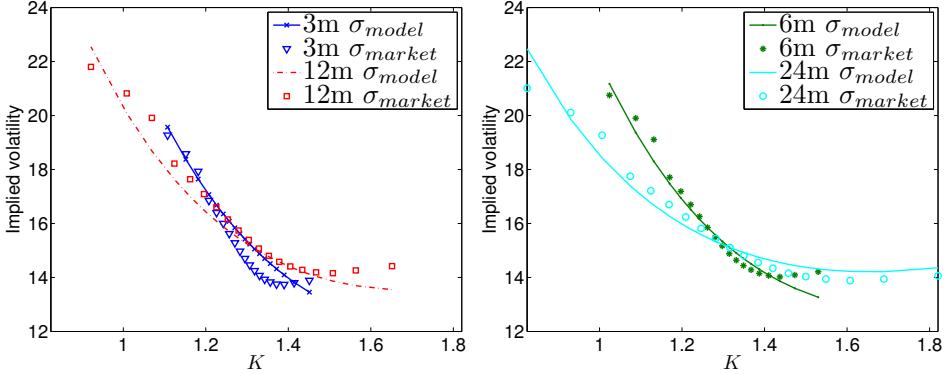


Figure 6: EURUSD. DSabr_I model: σ_{model} vs. σ_{market} for the whole volatility surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

	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 15: EURUSD. DSabr_I model: Performance of OpenMP vs. GPU versions, in single precision.

3. Calibration of the DSabr_II model using technique T-II

Analogously to the previous Section 6.2.1, an asymptotic expression for implied volatility in the DSabr_II model is not available. The calibration has been carried with Technique II (see Section 5). In this case, calibration is performed in prices and using double precision. The set of calibrated parameters is detailed in Table 16.

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

Table 16: EURUSD. DSabr_II model: Calibrated parameters.

Figure 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 17 the market and model prices for some strikes are shown.

3 months				6 months			
K	V_{market}	V_{model}	$\frac{ V_{\text{market}} - V_{\text{model}} }{V_{\text{market}}}$	K	V_{market}	V_{model}	$\frac{ V_{\text{market}} - V_{\text{model}} }{V_{\text{market}}}$
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}	$\frac{ V_{\text{market}} - V_{\text{model}} }{V_{\text{market}}}$	K	V_{market}	V_{model}	$\frac{ V_{\text{market}} - V_{\text{model}} }{V_{\text{market}}}$
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 17: EURUSD. DSabr_II model: V_{market} vs. V_{model} .

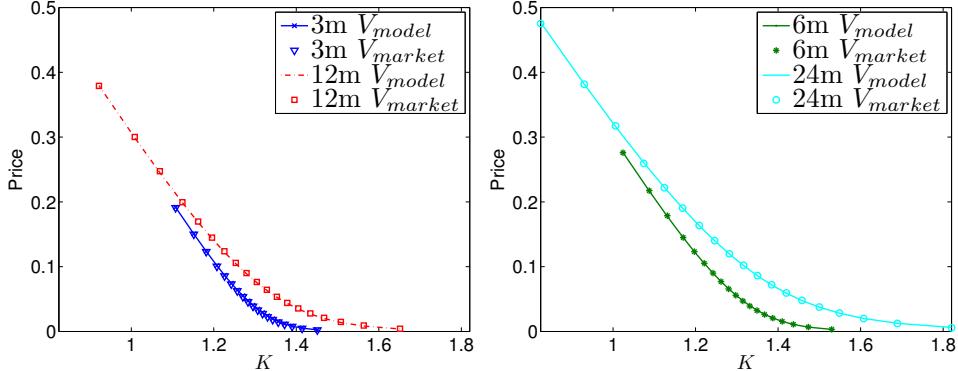


Figure 7: EURUSD. `DSabr_II` model: V_{model} vs. V_{market} for the whole prices surface. Maturities: 3 and 12 months (left), 6 and 24 months (right).

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

6.2.3. 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 calibration is also carried out with the same Monte Carlo method.

EURO STOXX 50: In Figure 8, the comparison between market prices (calculated with Black-Scholes formula and market volatilities) and model prices (calculated with the expression (9) and parameters of Table 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 of the EURO STOXX 50 case, in Figure 9, the market prices vs. the model ones are shown, using the parameters of Table 13. Furthermore, relative errors are shown. The mean relative error is 3.577857×10^{-2} and the maximum relative error is 2.582023×10^{-1} . Pricing all options in GPU takes 0.248013 seconds.

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 13 and for the second one those in Table 16.

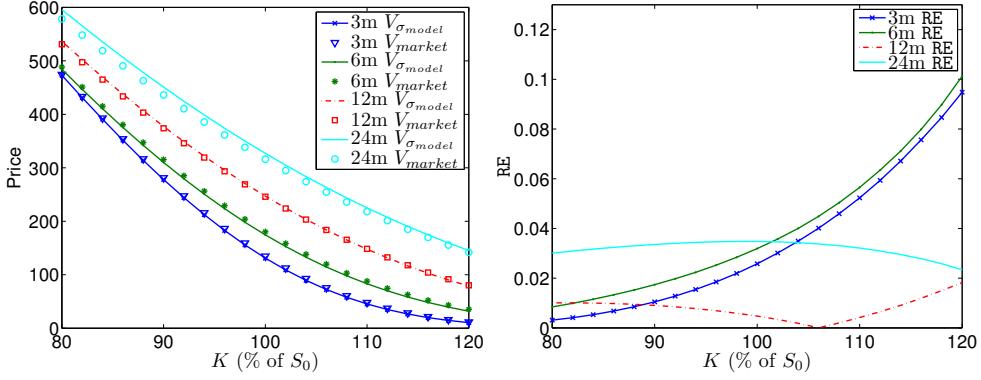


Figure 8: EURO STOXX 50. DSabr_I model for pricing European options. Prices (left) and relative errors (right).

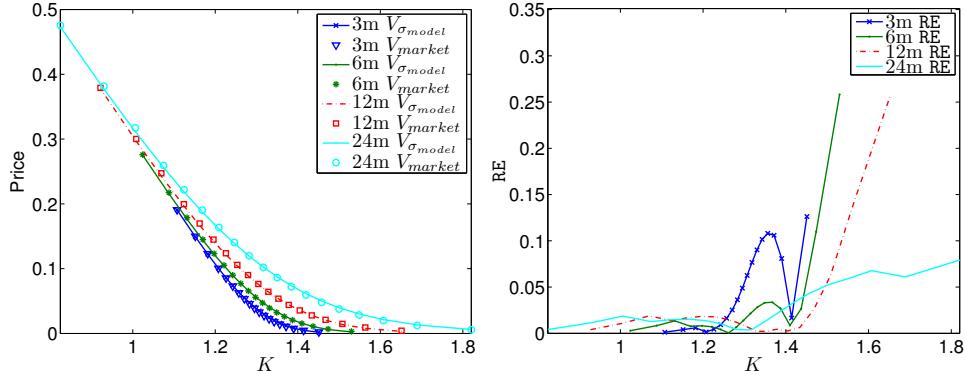


Figure 9: EURUSD. DSabr_I model for pricing European options. Prices (left) and relative errors (right).

Cliquet options are options where the strike price is periodically reset several times before final expiration date [31]. 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 series of forward-starting at-the-money options with local limits, cap and floor. The option payoff function is:

$$Clique(F_l, C_l, \{d_i\}_{i=1...D}, \{S_{d_i}\}_{i=1...D}) = \sum_{i=2}^D \max \left(\min \left(\frac{S_{d_i} - S_{d_{i-1}}}{S_{d_{i-1}}}, C_l \right), F_l \right), \quad (18)$$

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:

$$Clique_{global}(C_g, F_g, Clique) = \max(C_g, \min(F_g, Clique)), \quad (19)$$

where C_g is global cap, F_g global floor and *Cliquet* is given by (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 (19). We price the cliquet option with DSabr_I and DSabr_II models, in single precision. By using the DSabr_I model and the calibrated parameters in Table 13, the simulated cliquet option price is $V_{\sigma_{model}} = 0.098289$, while when using DSabr_II model and the parameters in Table 16 then 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.

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 paper 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 1 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.

Appendix

Appendix A. Expression of implied volatility in the general case

For the **Case II** of Section 2.2 (general case), using Mathematica, the functions ν_1^2 , ν_2^2 , η_1 and η_2^2 given by (10) have the following expressions:

$$\begin{aligned}
\nu_1^2(T) = & \frac{1}{4b^5 T^3} \left[9q_\nu^2 + 6b^4 \nu_0 (4d_\nu + \nu_0) T^2 + 4b^5 d_\nu^2 T^3 + 9bq_\nu (16d_\nu + \nu_0 - q_\nu T) \right. \\
& + 6b^3 T (-\nu_0 (8d_\nu + \nu_0) + (4d_\nu + \nu_0) q_\nu T) + 3b^2 (\nu_0 (16d_\nu + \nu_0) - 4(8d_\nu + \nu_0) q_\nu T + q_\nu^2 T^2) \\
& \left. - 3e^{-2bT} \left(3q_\nu^2 + 3bq_\nu (16d_\nu e^{bT} + \nu_0 + q_\nu T) + b^2 (\nu_0 + q_\nu T) (16d_\nu e^{bT} + \nu_0 + q_\nu T) \right) \right], \\
\nu_2^2(T) = & \frac{1}{4b^5 T^3} e^{-2bT} \left\{ 18q_\nu^2 + 6b^3 T (\nu_0 + q_\nu T)^2 + 6b^2 (\nu_0 + q_\nu T) (\nu_0 + 3q_\nu T) \right. \\
& + 9bq_\nu (2\nu_0 + 3q_\nu T) + e^{2bT} \left[-18q_\nu^2 + 6b^3 \nu_0 (8d_\nu + \nu_0) T + 4b^5 d_\nu^2 T^3 \right. \\
& + 9bq_\nu (-32d_\nu - 2\nu_0 + q_\nu T) + 6b^2 (-\nu_0 (16d_\nu + \nu_0) + 2(8d_\nu + \nu_0) q_\nu T) \left. \right] \\
& \left. + 48bd_\nu e^{bT} (6q_\nu + b(\nu_0(2+bT) + q_\nu T(4+bT))) \right\}, \\
\eta_1(T) = & \frac{2}{T^2} \left\{ -\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} \right. \\
& + \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 (\nu_0 q_\rho + q_\nu (\rho_0 - q_\rho T))}{(a+b)^4} \\
& + \frac{b [-2\nu_0 (q_\rho + a\rho_0) + a\nu_0 (2q_\rho + 3a\rho_0) T + 2q_\nu (q_\rho T + \rho_0 (-1 + aT))] }{(a+b)^4} \\
& + \frac{b^2 [q_\nu \rho_0 T + \nu_0 (q_\rho T + \rho_0 (-1 + 3aT))] }{(a+b)^4} \\
& + \frac{1}{2a^3 b^3 (a+b)^4} e^{-(a+b)T} \left\{ 4b^7 d_\nu e^{bT} q_\rho + 8a^2 b^5 d_\nu e^{bT} (b\rho_0 + q_\rho (3 + bT)) \right. \\
& + 2ab^6 d_\nu e^{bT} (b\rho_0 + q_\rho (8 + bT)) + a^7 d_\rho e^{aT} (4q_\nu + b^3 d_\nu e^{bT} T^2 + 2b(\nu_0 + q_\nu T)) \\
& + 4a^6 bd_\rho e^{aT} (4q_\nu + b^3 d_\nu e^{bT} T^2 + 2b(\nu_0 + q_\nu T)) \\
& + 2a^5 b^2 [12d_\rho e^{aT} q_\nu + 3b^3 d_\rho d_\nu e^{(a+b)T} T^2 + 6bd_\rho e^{aT} (\nu_0 + q_\nu T) \\
& + b(\rho_0 + q_\rho T) (d_\nu e^{bT} + \nu_0 + q_\nu T)] + 4a^4 b^3 [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 + 2bd_\rho e^{aT} (\nu_0 + q_\nu T) + b(\rho_0 + q_\rho T) (2d_\nu e^{bT} + \nu_0 + q_\nu T)] \\
& + a^3 b^3 [12q_\rho q_\nu + b^4 d_\rho d_\nu e^{(a+b)T} T^2 + 4b (4d_\nu e^{bT} q_\rho + \nu_0 q_\rho + q_\nu (d_\rho e^{aT} + \rho_0 + 2q_\rho T))] \\
& \left. + 2b^2 [d_\rho e^{aT} (\nu_0 + q_\nu T) + (\rho_0 + q_\rho T) (6d_\nu e^{bT} + \nu_0 + q_\nu T)] \right\} \right\},
\end{aligned}$$

$$\eta_2^2(T) = \frac{12}{T^4} \int_0^T \int_0^t \left\{ \frac{1}{a^2 b^2 (a+b)^3} e^{-(a+b)s} \left\{ b^5 d_\nu e^{bs} (-1 + e^{as}) q_\rho 6 \right. \right. \\ - ab^4 d_\nu e^{bs} (3q_\rho + b\rho_0 - e^{as} (3q_\rho + b\rho_0) + bq_\rho s) \\ + a^5 d_\rho e^{as} (-q_\nu - b(\nu_0 + q_\nu s) + e^{bs} (q_\nu + b(\nu_0 + bd_\nu s))) + a^3 b^2 [-q_\nu (3d_\rho e^{as} + \rho_0) \\ + e^{(a+b)s} ((d_\nu + \nu_0)q_\rho + q_\nu (3d_\rho + \rho_0) + b(3d_\rho \nu_0 + 3d_\nu \rho_0 + 2\nu_0 \rho_0) + 3b^2 d_\rho d_\nu s) \\ - d_\nu e^{bs} (q_\rho + 3b\rho_0 + 3bq_\rho s) - q_\rho (\nu_0 + 2q_\nu s) - b(\nu_0 + q_\nu s) (3d_\rho e^{as} + 2(\rho_0 + q_\rho s))] \\ + a^2 b^2 \left[-2q_\rho q_\nu + e^{(a+b)s} (2q_\rho q_\nu + b^2 (3d_\nu \rho_0 + \nu_0 (d_\rho + \rho_0)) + b((3d_\nu + \nu_0)q_\rho \right. \\ \left. \left. + q_\nu (d_\rho + \rho_0)) + b^3 d_\rho d_\nu s \right) - 3bd_\nu e^{bs} (q_\rho + b\rho_0 + bq_\rho s) - b^2 (d_\rho e^{as} + \rho_0 + q_\rho s) (\nu_0 + q_\nu s) \right. \\ \left. - b(\nu_0 q_\rho + q_\nu (d_\rho e^{as} + \rho_0 + 2q_\rho s)) \right] + a^4 b \left[e^{(a+b)s} (3bd_\rho \nu_0 + 3d_\rho q_\nu + b(d_\nu + \nu_0) \rho_0 \right. \\ \left. + 3b^2 d_\rho d_\nu s) - b(\rho_0 + q_\rho s) (d_\nu e^{bs} + \nu_0 + q_\nu s) - 3d_\rho e^{as} (q_\nu + b(\nu_0 + q_\nu s)) \right] \left. \right\}^2 ds dt. \right\}$$

Note that for this general case, it is not possible to obtain an explicit expression for $\eta_2^2(T)$, so that it can be approximated using an appropriate numerical quadrature formula.

Appendix B. 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.18: EURO STOXX 50 (Dec. 2011). Spot value $S_0 = 2311.1$ €. Interest rates and dividend yields.

References

- [1] A. Bernemann, R. Schereyer: Accelerating Exotic Option Pricing and Model Calibration Using GPUs. *Social Science Research Network*, (2011), 1-19.
- [2] F. Black, M. Scholes: The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81 (1973), 637-654.
- [3] S. B. Brooks, B. J. T. Morgan: Optimization using simulated annealing. *The Statistician*, 44 (2) (1995), 241-257.
- [4] B. Chen, L.A. Grzelak, C. W. Oosterlee: Calibration and Monte Carlo Pricing of the SABR-Hull-White Model for Long-Maturity Equity Derivatives, *Journal of Computational Finance*, 15 (2012), 79-113.
- [5] B. Chen, C. W. Oosterlee, H. van der Weide: A low-bias simulation scheme for the SABR stochastic volatility model. *Int. Journal of Theoretical & Applied Finance*, 15 (2012), 1-27.

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.19: 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.20: EUR/USD (Dec. 2011). Spot value $S_0 = 1.2939$ US dollars. Interest rates and dividend yields.

- [6] B. Dupire: Pricing with a smile. *Risk*, 7 (1994), 18-20.
- [7] B. Dupire: Pricing and hedging with a smile. *Mathematics of Derivative Securities*, Cambridge University Press, 1997.
- [8] A. M. Ferreiro, J. A. García, J. G. López-Salas, C. Vázquez: An efficient implementation of parallel simulated annealing algorithm in GPUs. *Journal of Global Optimization*, (2012). Available at <http://dx.doi.org/10.1007/s10898-012-9979-z>.
- [9] J. Gatheral: *The Volatility Surface: A Practitioner's Guide*. Wiley Finance, 2006.
- [10] P. Glasserman: *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York, 2003.
- [11] P. Glasserman, Q. Wu: Forward and Future Implied Volatility. *International Journal of Theoretical and Applied Finance*, 14 (2011), 407-432.
- [12] P. Hagan, D. Kumar, A. Lesniewski, and D. Woodward: Managing smile risk. *Wilmott Magazine*, (2002) 84-108.

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.21: EUR/USD (Dec. 2011). Implied volatilities for each maturity with different strikes K .

- [13] P. Henry-Labordere: *Analysis, Geometry and Modeling in Finance: Advanced Methods in Option Pricing*. Financial Mathematics Series, Boca Raton, FL, New York, Chapman & Hall, 2008.
- [14] S. Heston: A closed form solution for Options with Stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, 6 (1993), 327-343.
- [15] C. Homescu: Implied volatility surface: construction methodologies and characteristics. Available at <http://arxiv.org/abs/1107.1834>, (2011).
- [16] J. Hull, A. White: An Analysis of the Bias in Option Pricing Caused by a Stochastic Volatility. *Advances in Futures and Options Research*, 3 (1988), 27-61.
- [17] P. Jäckel: *Monte Carlo Methods in Finance*, Wiley Finance, 2002.
- [18] M. S. Joshi: Graphical Asian Options. *Wilmott Journal*, 2 (2) (2010), 97-107.
- [19] L. Kaisajuntti, J. Kennedy: Stochastic Volatility for Interest Rate Derivatives. *Working paper available at* http://papers.ssrn.com/sol3/papers.cfm?abstract_id=189488, (2011).
- [20] K. Larsson: Dynamic extensions and probabilistic expansions of the SABR model. Available at http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1536471, work in progress (2010).
- [21] A. Lee, C. Yau, M.B. Giles, A. Doucet, 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) (2010), 769-789.

- [22] G. Marsaglia: Xorshift RNGs. *Journal of Statistical Software*, 8 (4) (2003), 1-6.
- [23] L. Paulot: Asymptotic implied volatility at second order with application to the SABR model. Available at http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1413649, (2009).
- [24] J. Oblój: Fine-Tune your smile: Correction to Hagan et al. *Wilmott Magazine*, May 2008.
- [25] Osajima, Y: The asymptotic expansion formula of implied volatility for dynamic SABR Model and FX Hybrid Model. *Capital markets: asset pricing and valuation e-Journal*, Available at http://papers.ssrn.com/sol3/papers.cfm?abstract_id=965265, (2007).
- [26] J. Sanders, E. Kandrot: CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison Wesley Professional. 2011
- [27] A. Takahashi, A. Takehara, M. Toda: Computation in an asymptotic expansion method. Available at http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1413924, (2009).
- [28] Y. Tian, Z. Zhu, F. C. Klebaner, K. Hamza: Option pricing with the SABR model on the GPU. *High Performance Computational Finance (WHPCF) IEEE Workshop*, 14 (2010), 1-8.
- [29] Y. Tian, Z. Zhu, F. C. Klebaner, K. Hamza: Pricing barrier and American options under the SABR model on the graphics processing unit. *Concurrency and Computation: Practice and Experience*, (2011), 1-13.
- [30] G. West: Calibration of the SABR Model in Illiquid Markets. *Applied Mathematical Finance*, 12 (2005), 371-385.
- [31] P. Wilmott: Cliquet Options and Volatility Models. *Wilmott Magazine*, (2002), 78-83.
- [32] CUSIMANN web pages: <http://gforge.i-math.cesga.es/projects/cusimann/> or <http://code.google.com/p/cusimann/>
- [33] MPI web page: <http://www.mpi-forum.org>
- [34] Nvidia Corp.: CUDA 4.0 Programming guide. (2011).
- [35] OpenMP web page: <http://www.openmp.org>
- [36] Thrust web page: <http://thrust.github.com/>