Efficient calibration
of the dynamic SABR model

using multi-GPU

José Antonio García Rodríguez

Joint work with: A. Ferreiro, A. Leitao, J.G. Salas, C. Vázquez

Dpt. of Mathematics
Group M²NICA
U. A Coruña

Conference and Training Event on
Computational Methods and Technologies for Finance
[HPCFINANCE, TAMPERE 2013]
STRIKE

Novel Methods in Computational Finance

An Introduction to the Marie Curie Multi-ITN STRIKE Network

www.itn-strike.eu

José Antonio García
University of A Coruña, Spain
STRIKE – Overview

- maximum EU contribution of 3,582,470.98 € and 492 researcher months
  - 12 ESRs (PhD students, 36 months funding each)
  - 5 ERs (PostDocs, 12 months funding each)
- 11 Beneficiaries, 9 Associate Partners (3 universities, 3 SMEs, 3 Banks)
- many conferences, schools, compact courses, workshops, etc.
Scientific Work Packages 1-4 + Transferable Skills

Strike

Novel Methods in Computational Finance

- Novel nonlinear and kinetic Models of Financial Markets
- Modern Techniques of Discretization and Numerical Analysis
- Innovative Ideas in Scientific Computing
- State-of-the-art Calibration Methods
- Transferable Skills and Social Awareness

Close Academic and Private Sector Partnership

Dissemination

Research

Training

Impact

May 15, 2013

An Introduction to the Marie Curie Multi-ITN STRIKE Network
STRIKE – Structured Research Training

Trans-national training for the ESRs is based on **4 complementary elements**

1. **A Research Project**, involving original research (modelling, analysis, simulation, validation)

2. **Structured Training** in mathematical finance, analysis and modern numerical methods

3. **Scientific Computing (SC) Training** in the design, programming and operation of High Performance Computing resources like GPUs and established CFD libraries like OpenFOAM

4. **Transferable Skills Training** involving scientific, management and communication skills

**Training of ERs** will follow a similar pattern, with an independent research programme replacing the research project and less time spent on structured research training elements.

The ERs should assist the WP leaders and will be trained in the organization of the STRIKE network events.

May 15, 2013
An Introduction to the Marie Curie Multi-ITN STRIKE Network

May 15, 2013
STRIKE – Upcoming Events

- Summer School *Numerical methods for stochastic differential equations*, TU Vienna, Austria, September 2–4, 2013
- LMS Minisymposium *Advanced Decomposition Methods for Partial Differential Equations*, Kingston University, September 2–4, 2013
- Special Session *Computational Finance*, 13th Mathematical Modelling in Engineering & Human Behaviour, Universitat Politecnica de Valencia, Spain, September 3–6, 2013
- Winter School, University of Greenwich, December 2013.

more events on www.itn-stripe.eu
GOAL

Efficient SABR model calibration
GOAL

Efficient SABR model calibration

- Simulated annealing: global optimization
  Stochastic methaeuristic algorithm ⇒ High computational cost
Efficient SABR model calibration

- **Simulated annealing**: global optimization
  
  Stochastic metaheuristic algorithm ⇒ **High computational cost**

- **GPUs**: parallel SA
  
  The model must be calibrated/adjusted to market data very fast

  We will use GPUs for calibration
1. SABR MODEL

2. SIMULATED ANNEALING (SA)

3. PARALLEL IMPLEMENTATION OF SA

4. SABR CALIBRATION
1. SABR MODEL

2. SIMULATED ANNEALING (SA)

3. PARALLEL IMPLEMENTATION OF SA

4. SABR CALIBRATION
The (static) SABR model

SABR model ($\sigma, \alpha, \beta, \rho$ model):

- Stochastic volatility model
- Proposed by Hagan et al. in

**REF:**


- Local volatility models (Dupire, for example) were not able to reproduce market volatility smiles
The (static) SABR model

Stochastic differential equations for the dynamics of the forward price and volatility

\[
\begin{align*}
    dF_t &= \alpha_t F_t^\beta dW_1^t, \quad F_0 = \hat{f}, \\
    d\alpha_t &= \nu \alpha_t dW_2^t, \quad \alpha_0 = \alpha,
\end{align*}
\]

- \(F_t = S_t e^{(r-q)(T-t)}\) the forward price of the underlying asset \(S\)
  - \(r\): constant interest rate
  - \(q\): constant dividend yield
- \(\alpha_t\): asset volatility process
- \(dW_1, dW_2\): correlated Brownian motions \((dW_1^t dW_2^t = \rho dt)\)
- \(S_0\): spot price of the asset
The (static) SABR model

Stochastic differential equations for the dynamics of the forward price and volatility

\[
\begin{align*}
    dF_t &= \alpha_t F_t^\beta dW_1^t, \\
    d\alpha_t &= \nu \alpha_t dW_2^t,
\end{align*}
\]

\( F_0 = \hat{f}, \quad \alpha_0 = \alpha, \)

- \( F_t = S_t e^{(r-q)(T-t)} \) the forward price of the underlying asset \( S \)
  - \( r \): constant interest rate
  - \( q \): constant dividend yield
- \( \alpha_t \): asset volatility process
- \( dW_1, dW_2 \): correlated Brownian motions \( (dW_1^t dW_2^t = \rho dt) \)
- \( S_0 \): spot price of the asset

Parameters of the model:

- \( \alpha > 0 \): the volatility’s reference level
- \( 0 \leq \beta \leq 1 \): the variance elasticity
- \( \nu > 0 \): the volatility of the volatility
- \( \rho \): the correlation coefficient
The static SABR model - Implied volatility formula

Asymptotic expansion formula (correction proposed by Oblôj-2008)

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

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

where the following new expression for \( z \) is considered:

\[
z = \frac{\nu (\hat{f}^{1-\beta} - K^{1-\beta})}{\alpha (1 - \beta)}
\]

and \( x(z) \) is given

\[
x(z) = \ln \frac{\sqrt{1 - 2\rho z + z^2} + z - \rho}{1 - \rho}
\]
The (static) SABR model - Implied volatility formula

Neglecting the terms after \( + \cdots \):

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

where the coefficients \( A_1, A_2 \) and \( B \) are given by

\[
A_1 = -\frac{1}{2} (1 - \beta - \rho \nu \omega)
\]

\[
A_2 = \frac{1}{12} \left( (1 - \beta)^2 + 3((1 - \beta) - \rho \nu \omega) + (2 - 3 \rho^2) \nu^2 \omega^2 \right)
\]

\[
B = \frac{(1 - \beta)^2}{24} \frac{1}{\omega^2} + \frac{\beta \rho \nu}{4} \frac{1}{\omega} + \frac{2 - 3 \rho^2}{24} \nu^2
\]

and the value of \( \omega \) is given by

\[
\omega = \frac{\hat{\hat{f}}^{1-\beta}}{\alpha}
\]

Two special cases: \( \beta = 1 \), corresponding to a lognormal model, and \( \beta = 0 \), leading to a normal model.
Dynamic SABR- Functional parameters

- Drawback of the static SABR model: large errors when calibrating options with several maturity dates

- Dynamic SABR model: allows time dependency in some parameters

\[
dF_t = \alpha_t F_t^\beta dW_t^1, \quad F_0 = \hat{f},
\]

\[
d\alpha_t = \nu(t) \alpha_t dW_t^2, \quad \alpha_0 = \alpha,
\]

- The correlation coefficient is also time dependent:

\[
dW_t^1 dW_t^2 = \rho(t) dt
\]
Dynamic SABR- Functional parameters

- Similarly to the static SABR model, the dynamic one also provides an asymptotic expression to approximate the implied volatility (Osajima-2007):

\[
\sigma_B(K, \hat{f}, T) = \frac{1}{\omega} \left[ 1 + A_1(T) \ln(K/\hat{f}) + A_2(T) \ln^2(K/\hat{f}) + B(T)T \right],
\]

where

\[
A_1(T) = \frac{\beta - 1}{2} + \frac{\eta_1(T) \omega}{2},
\]

\[
A_2(T) = \frac{(1 - \beta)^2}{12} + \frac{1 - \beta - \eta_1(T) \omega}{4} + \frac{4\nu_1^2(T) + 3(\eta_2^2(T) - 3\eta_1^2(T))}{24} \omega^2,
\]

\[
B(T) = \frac{1}{\omega^2} \left[ \frac{(1 - \beta)^2}{24} + \frac{\omega \beta \eta_1(T)}{4} + \frac{2\nu_2^2(T) - 3\eta_2^2(T)}{24} \omega^2 \right],
\]

with

\[
\nu_1^2(T) = \frac{3}{T^3} \int_0^T (T - t)^2 \nu^2(t) dt
\]

\[
\nu_2^2(T) = \frac{6}{T^3} \int_0^T (T - t)t \nu^2(t) dt
\]

\[
\eta_1(T) = \frac{2}{T^2} \int_0^T (T - t) \nu(t) \rho(t) dt
\]

\[
\eta_2^2(T) = \frac{12}{T^4} \int_0^T \int_0^t \left( \int_0^s \nu(u) \rho(u) du \right)^2 ds dt
\]
1. SABR MODEL

2. SIMULATED ANNEALING (SA)

3. PARALLEL IMPLEMENTATION OF SA

4. SABR CALIBRATION
Simulated annealing sequential algorithm

Global optimization:

\[ \min f(x), \quad x \in V \]

\( f \) is the cost function
Simulated annealing sequential algorithm

Global optimization:

\[
\min f(x), \quad x \in V
\]

\(f\) is the cost function

Simulated annealing steps:

- **Step 1**: Start with the given temperature, \(T_0\), and the initial point, \(x_0\), with energy of configuration \(E_0 = f(x_0)\)
Simulated annealing sequential algorithm

Global optimization:

$$\min f(x), \quad x \in V$$

$f$ is the cost function

Simulated annealing steps:

- **Step 1**: Start with the given temperature, $T_0$, and the initial point, $x_0$, with energy of configuration $E_0 = f(x_0)$

- **Step 2**: Select a random coordinate of $x_0$ and a random number to modify the selected coordinate to obtain another point $x_1 \in V$ in the neighborhood of $x_0$. 
Simulated annealing sequential algorithm

Global optimization:

\[ \min f(\mathbf{x}), \quad \mathbf{x} \in V \]

\( f \) is the cost function

Simulated annealing steps:

- **Step 1**: Start with the given temperature, \( T_0 \), and the initial point, \( \mathbf{x}_0 \), with energy of configuration \( E_0 = f(\mathbf{x}_0) \)

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

- **Step 3**: Compare the function value at the two previous points using the Metropolis criterion:

Let \( E_1 = f(\mathbf{x}_1) \) 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.
Simulated annealing sequential algorithm

Global optimization:

$$\min f(\mathbf{x}), \quad \mathbf{x} \in V$$

$f$ is the cost function

Simulated annealing steps:

▶ **Step 1**: Start with the given temperature, $T_0$, and the initial point, $\mathbf{x}_0$, with energy of configuration $E_0 = f(\mathbf{x}_0)$

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

▶ **Step 3**: Compare the function value at the two previous points using the Metropolis criterion:
Let $E_1 = f(\mathbf{x}_1)$ 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

▶ **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
Simulated annealing sequential algorithm

Global optimization:

\[ \min f(x), \quad x \in V \]

\( f \) is the cost function

Simulated annealing steps:

- **Step 1**: Start with the given temperature, \( T_0 \), and the initial point, \( x_0 \), with energy of configuration \( E_0 = f(x_0) \)

- **Step 2**: Select a random coordinate of \( x_0 \) and a random number to modify the selected coordinate to obtain another point \( x_1 \in V \) in the neighborhood of \( x_0 \).

- **Step 3**: Compare the function value at the two previous points using the Metropolis criterion:
  Let \( E_1 = f(x_1) \) 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

- **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 following annealing schedule:

\[ T_1 = \lambda T_0, \quad 0 < \lambda < 1, \text{usually } \lambda \approx 1 \]
Simulated annealing sequential algorithm

- Pseudocode of the sequential algorithm

```plaintext
\[ \mathbf{x} = \mathbf{x}_0 ; \; T = T_0 ; \]
for i = 1 to NumberOfTemperatureReduction do
  for j = 1 to N do
    \[ \mathbf{x}' = \text{ComputeNeighbour}( ) ; \]
    \[ \Delta E = f(\mathbf{x}') - f(\mathbf{x}) \; \text{//Energy increment} \]
    if (\( \Delta E < 0 \) or \( \text{AcceptWithProbability} \; P(\Delta E, T) ) \)
      \[ \mathbf{x} = \mathbf{x}' \; \text{//The trial is accepted} \]
  end for
  \[ T = \lambda T \; \text{with} \; \lambda < 1 \]
end for;
```
Simulated annealing sequential algorithm

- Pseudocode of the sequential algorithm

```plaintext
x = x_0 ; T = T_0;
for i = 1 to NumberofTemperatureReduction do
  for j = 1 to N do
    x' = ComputeNeighbour( );
    ΔE = f(x') - f(x); //Energy increment
    if (ΔE < 0 or AcceptWithProbability P(ΔE, T))
      x = x'; //The trial is accepted
  end for
  T = λT ; with λ < 1
end for;
```

- Remark 1: this algorithm is intrinsically sequential
Simulated annealing sequential algorithm

- Pseudocode of the sequential algorithm

\[ \mathbf{x} = \mathbf{x}_0; \quad T = T_0; \]
for \( i = 1 \) to \( \text{NumberOfTemperatureReduction} \) do
  for \( j = 1 \) to \( N \) do
    \( \mathbf{x}' = \text{ComputeNeighbour}(\ ); \)
    \( \Delta E = f(\mathbf{x}') - f(\mathbf{x}); \) //Energy increment
    if \( (\Delta E < 0 \text{ or AcceptWithProbability } P(\Delta E, T)) \)
      \( \mathbf{x} = \mathbf{x}'; \) //The trial is accepted
  end for
  \( T = \lambda T; \) with \( \lambda < 1 \)
end for;

- Remark 1: this algorithm is intrinsically sequential

- Remark 2: the computational cost of the algorithm depends on the number of function evaluations
1. SABR MODEL

2. SIMULATED ANNEALING (SA)

3. PARALLEL IMPLEMENTATION OF SA

4. SABR CALIBRATION
Parallel implementation of simulated annealing

GPUs have a double advantage

Execution parallelism

- A GPU is based in a Many-core architecture with a huge number of cores
  - Up to 512 in the Nvidia Fermi architecture (for example, GTX580)
  - Up to 2688 in Nvidia Kepler architecture!!! (for example Nvidia Titan)

- The GPU can execute many tasks (execution threads) at a time

- SIMD paradigm (Single Instruction Multiple Data)
  - The same instruction is executed asynchronously by all the computing cores, but over different data

  We only need to code the set of instructions once, and this computing kernel is executed by all the cores
Parallel implementation of simulated annealing

GPUs have a double advantage

Execution parallelism

- A GPU is based in a **Many-core** architecture with a huge number of cores
  - Up to 512 in the Nvidia Fermi architecture (for example, GTX580)
  - Up to 2688 in Nvidia Kepler architecture!!! (for example Nvidia Titan)
- The GPU can execute many tasks (execution threads) at a time

- **SIMD paradigm** (Single Instruction Multiple Data)
  - The same instruction is executed asynchronously by all the computing cores, but over different data
  - We only need to code the set of instructions once, and this computing kernel is executed by all the cores

Faster memory access

- Much bigger memory bandwidth than a CPU (up to 288 GB/s)
Parallel implementation of simulated annealing

GPUs have a double advantage

Execution parallelism

- A GPU is based in a Many-core architecture with a huge number of cores
  - Up to 512 in the Nvidia Fermi architecture (for example, GTX580)
  - Up to 2688 in Nvidia Kepler architecture!!! (for example Nvidia Titan)
- The GPU can execute many tasks (execution threads) at a time

- **SIMD paradigm** (Single Instruction Multiple Data)
  - The same instruction is executed asynchronously by all the computing cores, but over different data
  
  We only need to code the set of instructions once, and this computing kernel is executed by all the cores

Faster memory access

- Much bigger memory bandwidth than a CPU (up to 288 GB/s) ...but...
Parallel implementation of simulated annealing

GPUs have a double advantage

Execution parallelism

- A GPU is based in a **Many-core** architecture with a huge number of cores
  - Up to 512 in the Nvidia Fermi architecture (for example, GTX580)
  - Up to **2688** in Nvidia Kepler architecture!!! (for example Nvidia Titan)
- The GPU can execute many tasks (execution threads) at a time
  **SIMD paradigm** (Single Instruction Multiple Data)
- The same instruction is executed asynchronously by all the computing cores, but over different data
  We only need to code the set of instructions once, and this **computing kernel** is executed by all the cores

Faster memory access

- Much bigger memory bandwidth than a CPU (up to 288 GB/s)
  **...but...**
- **Lunch is not for free:** To benefit from this, the programmer is responsible of doing a good job when accessing memory
Parallel implementation of simulated annealing

Asynchronous implementation (Mono-GPU)

- Each thread performs a whole Simulated Annealing process
- When all the threads have finished, we perform a reduction operation
- Random numbers are computed in place using Nvidia CURAND library
Each thread performs a Markov chain at each temperature level

Before advancing to the next temperature level, the best minimum is reduced from all threads and used as starting point for the next temperature step (for all threads)
Parallel implementation of simulated annealing

Synchronous (Multi-GPU)

CPU threads are created with OpenMP
Parallel implementation of simulated annealing

A typical benchmark for optimization techniques is the Normalized Schwefel function:

\[
f(\mathbf{x}) = -\frac{1}{n} \sum_{i=1}^{n} x_i \cdot \sin(\sqrt{|x_i|}), \quad -512 \leq x_i \leq 512, \quad \mathbf{x} = (x_1, \ldots, x_n)
\]

For any \( n \), the global minimum is achieved at \( \bar{x}_i = 420,969, \ i = 1, \ldots, n \), and \( f(\bar{x}) = -418,983 \).
Convergence: dimension $n = 64$
Convergence: dimension $n = 128$

![Graph showing convergence for $n = 128$](image)
Convergence: dimension $n = 256$

![Convergence plot for SABR model calibration with GPUs](image)

- **$n = 256$**: The plot shows the convergence of the calibration process for different values of $n$.
- **Relative error**: The relative error decreases as the number of function evaluations increases.
- **Number of function evaluations**: The x-axis represents the number of function evaluations required to achieve convergence.

The plot illustrates the effectiveness of the PARALLEL SA method in calibrating the SABR model with GPUs, demonstrating rapid convergence and efficient use of computational resources.
Parallel implementation of simulated annealing

Table: Accuracy and performance of CUDA version vs sequential version with one CPU core. Relative error is measured in $\| \cdot \|_2$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Sequential</th>
<th></th>
<th>Parallel (GPU)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Relative error</td>
<td>Time</td>
<td>Relative error</td>
</tr>
<tr>
<td>8</td>
<td>1493,7686</td>
<td>$2,4283 \times 10^{-3}$</td>
<td>5,6859</td>
<td>$4,1656 \times 10^{-5}$</td>
</tr>
<tr>
<td>32</td>
<td>4618,5820</td>
<td>$3,8852 \times 10^{-3}$</td>
<td>60,1882</td>
<td>$6,0577 \times 10^{-5}$</td>
</tr>
<tr>
<td>128</td>
<td>16479,27613</td>
<td>$9,2648 \times 10^{-3}$</td>
<td>215,5416</td>
<td>$1,5304 \times 10^{-4}$</td>
</tr>
<tr>
<td>512</td>
<td>68134,5760</td>
<td>$1,8097 \times 10^{-2}$</td>
<td>893,7668</td>
<td>$4,5503 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

- With the used configuration, the algorithm performs $1,8776 \times 10^9$ function evaluations in all cases
- Increasing the problem dimension, the speedup decreases: speed up is limited by memory bandwidth
- Hardware configuration:  
  GPU: Nvidia GTX580  
  CPU: Dual Quad Core Intel Xeon E5630
Parallel implementation of simulated annealing

- Details of the GPU parallelization of SA available at:

**REF:**


- The multi-GPU code (C/C++ and CUDA) is open-source and freely available at:

  [http://code.google.com/cusimann](http://code.google.com/cusimann)

- Documentation and several classical optimization tests are included
1. SABR MODEL

2. SIMULATED ANNEALING (SA)

3. PARALLEL IMPLEMENTATION OF SA

4. SABR CALIBRATION
SABR calibration

The calibration process tries to obtain a set of model parameters that makes model values as close as possible to market ones, i.e.:

\[ \text{Data}_{\text{market}}(K_j, \hat{f}, T_i) \approx \text{Data}_{\text{model}}(K_j, \hat{f}, T_i). \]

In order to achieve this target we must follow several steps:

- Decide how to perform the calibration process, *(prices or volatilities)*
- Choose market data that should be highly representative of the market’s situation
- Choose the *error measure* will be used to compare model and market volatilities (or prices)
- Choose the (local or global) optimization or minimization algorithm to reduce as much as possible the obtained 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.
SABR calibration

Cost functions:

- Individual calibration: the calibration is made for one maturity $T_i$ the

$$f_{i,E}(\mathbf{x}) = \sum_{j=1}^{m_i} \left( \text{Data}_{\text{market}}(K_j, \hat{f}, T_i) - \text{Data}_{\text{model}}(K_j, \hat{f}, T_i) \right)^2(\mathbf{x}),$$

or

$$f_{i,E}(\mathbf{x}) = \sum_{j=1}^{m_i} \left( \frac{\text{Data}_{\text{market}}(K_j, \hat{f}, T_i) - \text{Data}_{\text{model}}(K_j, \hat{f}, T_i)}{\text{Data}_{\text{market}}(K_j, \hat{f}, T_i)} \right)^2(\mathbf{x}),$$

where $\mathbf{x}$ denotes the parameters to calibrate.

- “All-at-once” calibration: the calibration is made for a set of maturity dates (we calibrate the whole vol surface)

$$f_E(\mathbf{x}) = \sum_{i=1}^{N} \sum_{j=1}^{m_i} \left( \text{Data}_{\text{market}}(K_j, \hat{f}, T_i) - \text{Data}_{\text{model}}(K_j, \hat{f}, T_i) \right)^2(\mathbf{x}),$$

or

$$f_E(\mathbf{x}) = \sum_{i=1}^{N} \sum_{j=1}^{m_i} \left( \frac{\text{Data}_{\text{market}}(K_j, \hat{f}, T_i) - \text{Data}_{\text{model}}(K_j, \hat{f}, T_i)}{\text{Data}_{\text{market}}(K_j, \hat{f}, T_i)} \right)^2(\mathbf{x}).$$
Calibration: EURUSD

EURUSD. Market volatility surface (December 2011)
EURUSD. Calibrated parameters for each maturity when considering the SABR model

<table>
<thead>
<tr>
<th></th>
<th>3 months</th>
<th>6 months</th>
<th>12 months</th>
<th>24 months</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.146859</td>
<td>0.152825</td>
<td>0.158210</td>
<td>0.154572</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1.0</td>
<td>0.990518</td>
<td>0.945088</td>
<td>0.999993</td>
</tr>
<tr>
<td>$\nu$</td>
<td>0.911966</td>
<td>0.675457</td>
<td>0.491647</td>
<td>0.328907</td>
</tr>
<tr>
<td>$\rho$</td>
<td>−0.447718</td>
<td>−0.490521</td>
<td>−0.511180</td>
<td>−0.560022</td>
</tr>
</tbody>
</table>
EURUSD. Calibration performance for $T = 24$ months: OpenMP vs GPU in single precision

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (1 thread)</td>
<td>4198,03</td>
<td>-</td>
</tr>
<tr>
<td>2 threads</td>
<td>2109,56</td>
<td>1,99</td>
</tr>
<tr>
<td>4 threads</td>
<td>1062,79</td>
<td>3,95</td>
</tr>
<tr>
<td>8 threads</td>
<td>548,04</td>
<td>7,66</td>
</tr>
<tr>
<td>GPU</td>
<td>21,84</td>
<td>192,19</td>
</tr>
<tr>
<td>2 GPU</td>
<td>12,58</td>
<td>333,52</td>
</tr>
</tbody>
</table>
Calibration: EURUSD

(a) 3 months maturity.

(b) 6 months maturity.

(c) 12 months maturity.

(d) 24 months maturity.
Calibration: EUROSTOXX50

EUROSTOXX50. Market Volatility surface (December 2011)
Calibration: EUROSTOXX50

Using the asymptotic expression of the implied volatility as the cost function

Individual calibration (all strikes and one maturity)

<table>
<thead>
<tr>
<th></th>
<th>3 months</th>
<th>6 months</th>
<th>12 months</th>
<th>24 months</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.298999</td>
<td>0.302060</td>
<td>0.289271</td>
<td>0.277844</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$\nu$</td>
<td>0.382558</td>
<td>0.381724</td>
<td>0.308560</td>
<td>0.264178</td>
</tr>
<tr>
<td>$\rho$</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-0.999729</td>
<td>-1.0</td>
</tr>
</tbody>
</table>

EUROSTOXX50. Calibrated parameters considering the static SABR model
**Calibration: EUROSTOXX50**

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CPU</td>
<td>4172.746</td>
<td>-</td>
</tr>
<tr>
<td>2 CPU</td>
<td>2096.857</td>
<td>1.99</td>
</tr>
<tr>
<td>4 CPU</td>
<td>1056.391</td>
<td>3.95</td>
</tr>
<tr>
<td>8 CPU</td>
<td>545.456</td>
<td>7.65</td>
</tr>
<tr>
<td>GPU</td>
<td>21.955</td>
<td>190.06</td>
</tr>
<tr>
<td>2 GPU</td>
<td>12.609</td>
<td>330.93</td>
</tr>
</tbody>
</table>

Calibration: EUROSTOXX50

(e) 3 months maturity.

(f) 6 months maturity.

(g) 12 months maturity.

(h) 24 months maturity.

Figura: EUROSTOXX50. Static SABR: $\sigma_{\text{model}}$ vs. $\sigma_{\text{market}}$ at maturities 3, 6, 12 and 24 months.
Dynamic SABR- Functional parameters

- Similarly to the static SABR model, the dynamic one also provides an asymptotic expression to approximate the implied volatility (Osajima-2007):

\[
\sigma_B(K, \hat{f}, T) = \frac{1}{\omega} \left( 1 + A_1(T) \ln(K/\hat{f}) + A_2(T) \ln^2(K/\hat{f}) + B(T)T \right),
\]

where

\[
A_1(T) = \frac{\beta - 1}{2} + \frac{\eta_1(T)\omega}{2}
\]

\[
A_2(T) = \frac{(1 - \beta)^2}{12} + \frac{1 - \beta - \eta_1(T)\omega}{4} + \frac{4\nu_1^2(T) + 3(\eta_2^2(T) - 3\eta_1^2(T))}{24} \omega^2
\]

\[
B(T) = \frac{1}{\omega^2} \left( \frac{(1 - \beta)^2}{24} + \frac{\omega \beta \eta_1(T)}{4} + \frac{2\nu_2(T) - 3\eta_2^2(T)}{24} \omega^2 \right)
\]

with

\[
\nu_1^2(T) = \frac{3}{T^3} \int_0^T (T - t)^2 \nu^2(t)dt
\]

\[
\nu_2^2(T) = \frac{6}{T^3} \int_0^T (T - t)t \nu^2(t)dt
\]

\[
\eta_1(T) = \frac{2}{T^2} \int_0^T (T - t)\nu(t)\rho(t)dt
\]

\[
\eta_2^2(T) = \frac{12}{T^4} \int_0^T \int_0^t \left( \int_0^s \nu(u)\rho(u)du \right)^2 dsdt
\]
Dynamic SABR- Functional parameters

Choice of the functions \( \rho \) and \( \nu \) is a very important decision: the volatility approximation depends on this choice.

The expressions of \( \rho(t) \) and \( \nu(t) \) have to be smaller for long terms \( (t \text{ large}) \) rather than for short terms \( (t \text{ small}) \).

We consider the following functions with exponential decay (classical choice)

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

an thus

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

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

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

\[
\eta_2^2(T) = \frac{3\nu_0^2 \rho_0^2}{T^4(a + b)^4} \left[1 - 8e^{-(a+b)T} + (7 + 2(a + b)T(-3 + (a + b)T))\right]
\]
Parallel implementation of simulated annealing

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>0,155464</th>
<th>$\beta$</th>
<th>0,971908</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_0$</td>
<td>-0,642617</td>
<td>$\nu_0$</td>
<td>0,800275</td>
</tr>
<tr>
<td>$a$</td>
<td>0,001000</td>
<td>$b$</td>
<td>2,609300</td>
</tr>
</tbody>
</table>

EURUSD. Dynamic SABR model: Calibrated parameters

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (1 thread)</td>
<td>16793,41</td>
<td>-</td>
</tr>
<tr>
<td>2 threads</td>
<td>8389,14</td>
<td>2,00</td>
</tr>
<tr>
<td>4 threads</td>
<td>4240,11</td>
<td>3,96</td>
</tr>
<tr>
<td>8 threads</td>
<td>2204,20</td>
<td>7,62</td>
</tr>
<tr>
<td>GPU</td>
<td>69,73</td>
<td>240,83</td>
</tr>
<tr>
<td>2 GPUs</td>
<td>37,16</td>
<td>451,92</td>
</tr>
</tbody>
</table>

EURUSD. Dynamic SABR model: Performance of OpenMP vs. GPU versions, in single precision
Dynamic SABR - Functional parameters

We also propose the original and more general choice

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

\[ \nu_1^2, \nu_2^2 \text{ and } \eta_1 \text{ (computed with Mathematica)} \]

\[ \eta_2^2: \text{ Must be approximated by quadrature formula} \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>0.154037</td>
</tr>
<tr>
<td>( \beta )</td>
<td>1.000000</td>
</tr>
<tr>
<td>( \rho_0 )</td>
<td>-0.693682</td>
</tr>
<tr>
<td>( \nu_0 )</td>
<td>7.541424</td>
</tr>
<tr>
<td>( a )</td>
<td>0.000000</td>
</tr>
<tr>
<td>( b )</td>
<td>150.000000</td>
</tr>
<tr>
<td>( d_\rho )</td>
<td>-0.200342</td>
</tr>
<tr>
<td>( d_\nu )</td>
<td>0.339807</td>
</tr>
<tr>
<td>( q_\rho )</td>
<td>0.345973</td>
</tr>
<tr>
<td>( q_\nu )</td>
<td>-0.992551</td>
</tr>
</tbody>
</table>

EURUSD. Dynamic SABR model: Calibrated parameters
Hibrid synchronous algorithm

- $T_0$
- $\lambda T_0$
- $\lambda^{-1} T_0 \leq T_{\text{min}}$

**CPU Threads**

- Randomly:
  - $\mathbf{x}_0^0 \rightarrow \mathbf{x}_0^1 \rightarrow \mathbf{x}_1^0 \rightarrow \mathbf{x}_0^w-2 \rightarrow \mathbf{x}_0^{w-1}$
  - $\mathbf{x}_1^0 \rightarrow \mathbf{x}_1^1 \rightarrow \mathbf{x}_1^{w-2} \rightarrow \mathbf{x}_1^{w-1}$
- Reduce min in CPU:
  - $\mathbf{x}_0^\text{min} \rightarrow \mathbf{x}_1^\text{min} \rightarrow \mathbf{x}_2^\text{min} \rightarrow \mathbf{x}_{\text{r-1}}^\text{min} \rightarrow \mathbf{x}_{\text{r}}^\text{min}$
- $\mathbf{x}_0^0 \rightarrow \mathbf{x}_1^0 \rightarrow \mathbf{x}_2^0 \rightarrow \mathbf{x}_{\text{r}}^0$
- Reduce min:
  - $\mathbf{x}_0^\text{min} \rightarrow \mathbf{x}_1^\text{min} \rightarrow \mathbf{x}_2^\text{min} \rightarrow \mathbf{x}_{\text{r-1}}^\text{min} \rightarrow \mathbf{x}_{\text{r}}^\text{min}$
- $\mathbf{x}_0^0 \rightarrow \mathbf{x}_1^0 \rightarrow \mathbf{x}_2^0 \rightarrow \mathbf{x}_{\text{r}}^0$

**CPU**

**SABR CALIBRATION**
Parallel implementation of simulated annealing

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>35033,72</td>
<td>—</td>
</tr>
<tr>
<td>2 GPUs</td>
<td>19143,35</td>
<td>1,83</td>
</tr>
</tbody>
</table>