COMPUTATIONAL FINANCE
- ON THE SEARCH FOR PERFORMANCE.

PhD Thesis

LYKKE RASMUSSEN

This thesis has been submitted to the PhD School of The Faculty of Science, University of Copenhagen.

This dissertation has been submitted in partial fulfilment of the requirements of the degree of Doctor of Philosophy at the Ph.D. School of The Faculty of Science, University of Copenhagen.

Department	The Department of Mathematical Sciences, University of Copenhagen.
Research group	Insurance and Economics, connected to the HIPERFIT project.
Author	Lykke Rasmussen, rasmussen.lykke@gmail.com.
Supervisor	Prof. Rolf Poulsen, University of Copenhagen.
Submission date	January 18th, 2016.
Assessment Committee	David Skovmand, University of Copenhagen, Prof. Natalie Packham, Berlin School of Economics and Law, Antoine Savine, Danske Bank Copenhagen.
ISBN	978-87-7078-943-1.

The code and corresponding output data for the papers can be found at the Electronic Research Data Archive for University of Copenhagen.

www.erda.dk/public/archives/YXJjaGl2ZS1iRVhMYUs=/published-archive.html.
www.erda.dk/public/archives/YXJjaGl2ZS1jcGhvM3E=/published-archive.html.

Paper I : Efficient Calibration of the Local Volatility Model

Abstract

Calibration of Dupire's deterministic local volatility function has been extensively researched over the years, and numerous papers on this topic has been presented. Yet the question of whether one of these methods are superior to the rest still remains unanswered. This paper examines the performance for five different methods for calibrating the local volatility function. The methods are thoroughly assessed in a uniform testing framework where they are compared in terms of accuracy, smoothness, speed and robustness.

Resumé

Kalibreringen af Dupires determiniske lokale volatilitets funktion er blevet udforsket i stort omfang i løbet af årerne, talrige artikler omkring emnet er blevet skrevet. Alligevel er spørgsmålet om hvorvidt en af disse metoder er bedre end de andre stadig ubesvaret. Denne artikel undersøger egenskaberne for fem af disse forskellige metoder til at kalibrere den lokale volatilitetsfunktion. Metoderne bliver grundigt undersøgt indenfor en ensartet testramme, hvor de sammenlignes på nøjagtighed, glathed, hastighed og robusthed.

Paper II : Efficient Calculation of Sensitivities

Abstract

One of the major challenges in todays post-crisis finance environment is calculating the sensitivities of complex products for hedging and risk management. Historically, these derivatives have been determined using *bump-and-revalue*, but due to the increasing magnitude of these computations does this get increasingly difficult on available hardware. In this paper three alternative methods for evaluating derivatives are compared: the *complex-step derivative approximation*, the *algorithmic forward mode* and the *algorithmic backward mode*. These are applied to the price of the Credit Value Adjustment for an interest rate swap, and subsequently assessed in terms of accuracy, stability and run time. Hands-on details are provided for the implementation along with a thorough validation framework.

Resumé

En af de største udfordringer indenfor finansiering efter finanskrisen, er at beregne følsomhederne for komplekse produkter med henblik på hedging og risiko styring. Historisk set er disse afledte blevet beregnet vha. *bump-and-revalue*, men grundet det stigende omfang af disse beregninger, bliver dette stadigt vanskeligere på tilgængeligt hardware. I dette papir sammenlignes tre alternative metoder til at beregne afledte: *complex-step derivative approksimation*, *algorithmic forward mode* og *algorithmic backward mode*. Disse benyttes på Credit Value Adjustment prisen for en rente swap og undersøges derefter i forhold til nøjagtighed, stabilittet og køretid. Praktiske detaljer omkring implementeringen er angivet sammen med en grundig validerings fremgangsmåde.

Preface

The financial crisis of '08 disrupted the financial environment. Within the field of mathematical finance this led to an increased focus on stability and robustness of the calibrated models as well as extensive risk management requiring both speed and accuracy of the sensitivities. This dissertation *Computational Finance - on the search for performance* contributes to this endeavour by assessing methods seeking to fulfill these goals.

Stability and robustness of the local volatility model gained renewed interest with the award winning paper Andreasen & Huge (2011). This paper does, as many others, describe its unique contribution to the discipline of calibrating the local volatility function, but omits to bring its contribution into context with other methods already published within the field. The first paper presented in this dissertation fills this gap by presenting an extensive comparison of the five most promising methods on the subject.

The second paper in this dissertation is concerned with the performance of alternative computer algorithms used for calculating the sensitivities of complex financial products. The game changer within this line of research was the award winning article Giles & Glasserman (2006) on algorithmic differentiation. This method is gaining grounds within the finance industry and there is therefore an increased need for an exhaustive assessment of the alternative methods available for calculating the sensitivities. The second paper provides this assessment.

In the spring of 2014 I was fortunate enough to visit the author Mike Giles. He was working on a book script at the time regarding algorithmic differentiation. During my review of the script I suggested - as a true rookie - that he separated the math from the implementation in order to simplify things. He answered *that for this very topic the two are closely related and cannot be separated* - once you understand this, it becomes clear why and how to utilize this technique. I suggest the reader bears this in mind when reading the second paper.

Acknowledgments

I wish to thank my PhD advisor, Professor Rolf Poulsen, for his support, guidance and participation during the PhD program. A special thanks goes to Professor Brian Vinter for taking an interest in me and my HPC upbringing. I also owe Brian great thanks for helping me arrange my stay abroad. My gratitudes extends to Mike Giles for his hospitality and patience during my stay at Oxford-Man Institute at University of Oxford and for introducing me to the world of adjoints. I would also like to thank Andreas Winther Jessen for input on the practical issues regarding CVA. Also thanks to Jacques du Toit for helping me shape the format of the first paper by inputs and fruitful discussions.

Lykke Rasmussen,
Copenhagen, 2016.

Bibliography

Andreasen, Jesper and Huge, Brian (2011). Volatility interpolation. *Risk Magazine*, March, 76–79.

Giles, Mike and Glasserman, Paul (2006). Smoking adjoints : fast Monte Carlo Greeks. *Risk Magazine*, (3), 88–92.

Contents

Introduction 1

I Efficient Calibration of the Local Volatility Model 5

1 Motivation 7
2 Setting the Scene 10
2.1 Normed call prices 10
2.2 The local volatility model 12
2.3 Absence of arbitrage 13
3 Presentation of the methods 19
3.1 Benko 21
3.2 Fengler 28
3.3 AndreasenHuge 35
3.4 GlaserHeider 42
3.5 FenglerHin 49
4 Empirical work 58
4.1 Data 58
4.2 Filtering 58
4.3 Quantitative Comparison of the Methods 64
5 Did we reach our goals? 89
A Appendix 91
A.1 Glaser & Heider quadratic programming problem. 91
A.2 Fengler & Hin linear inequality constraints. 91
References 94

II Efficient Calculation of Sentivitities 97

1 Motivation 99
2 Financial Framework 101
2.1 CVA approximation formula. 102
2.2 Interest Rate Swap 103

2.3 Short rate model - Hull White 106
2.4 Intensity model - CIR 107
2.5 Calibrating the model framework 109
2.6 Discretization 114
2.7 The derivatives 117
3 Derivatives Framework 119
3.1 Finite Difference Approximation 123
3.2 Complex-Step Derivative Approximation 126
3.3 Algorithmic Derivatives 131
4 Implementation 142
4.1 Program Design 142
4.2 Active and passive variables 144
4.3 Recording 145
4.4 Validation Checks 147
4.5 Automatic Differentiation 149
5 Results 151
5.1 Accuracy 151
5.2 Stability 153
5.3 Run Time 155
5.4 Automatic tool 159
6 Wrapping it up 161
6.1 When to use What? 161
6.2 Before Flying the Nest 162
A Appendix : Source Code 164
A.1 Object Oriented Design 164
A.2 Code formatting 171
B Appendix : Data 172
References 182

Paper I

Efficient Calibration of the Local Volatility Model

1 Motivation

The calibration of Dupires deterministic local volatility function has been extensively researched over the years, and numerous papers on the topic has been presented. The problem gained renewed interest in 2012 when Andreasen & Huge was awarded 'Quants of the Year' by Risk Magazine for their article 'Volatility Interpolation'. To this day the question of whether one of these numerous methods are superior to the rest still remains unanswered. We will in this paper try to get closer to an answer by assessing 5 of the currently most promising methods.

The local volatility model (see e.g. Dupire (1994)) is the simplest expansion to the Black-Scholes model which captures the implied volatility smile - and term structure, which has been evident in the market since the crash of 1987.The dynamics of this model resembles that in the Black-Scholes model except that the volatility function is not a constant, but a deterministic function of time and the value of the underlying, as can be seen in eqn. (3).

Today this model is widely used in many institutions and across many asset classes, as described by Fengler & Hin (2013) in the quote given below. It is also used as a first step in calibrating the more advanced stochastic local volatility (SLV) models, which have become the standard models in areas such as FX and exotic equity derivatives.

> *Nowadays, local volatility models are widely used in the practice of quantitative finance for pricing and hedging mildly path-dependent derivatives. This is because these models neatly embed in a portfolio context as they provide an almost perfect fit to observed option prices; at the same time option pricing is fast and efficient.* (Fengler & Hin (2013))

Dupire shows in his paper, *Pricing with a Smile*, from 1994 how the local volatility function can be derived from the derivatives of a continuous surface of arbitrage-free call prices: $c(K,\tau)$. This formula has afterwards become known as Dupire's formula:

$$\sigma^{loc}(K,\tau) = \sqrt{\frac{2\,\frac{\partial c(K,\tau)}{\partial \tau}}{K^2\,\frac{\partial^2 c(K,\tau)}{\partial K^2}}}.$$

Using this formula, the calibration of the local volatility model boils down to the problem of generating a *full continuum in expiry and strike of arbitrage-consistent*

European-style option prices from a discrete set of observed option quotes, as Andreasen & Huge (2011) put it. This is a non-trivial problem consisting of interpolating, or approximating, this scattered set of observations while preserving, or establishing, absence of arbitrage.

Numerous procedures for overcoming this difficulty has been presented over the years. In this paper the focus has been on some of the more recent methods listed in figure 1.

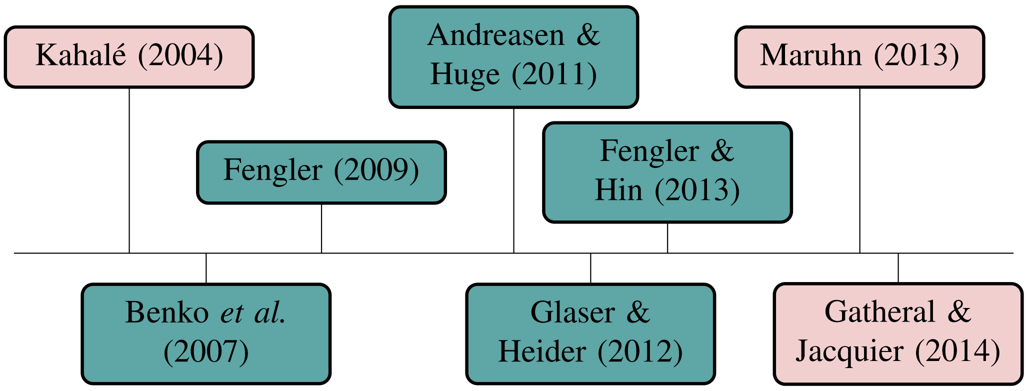


Figure 1: Articles presenting methods for calibrating the local volatility function.

Three of these methods have been filtered out beforehand:

▷ Kahalé (2004) has been disregarded, as it assumes arbitrage-free observations as input. The discussion in Hentschel (2003) indicates, that this is definitely not a given, rather the reverse. This method is therefore not robust enough to be considered here.

▷ Maruhn (2013) describes a method for fitting the call surface in his presentation notes from *the Global derivatives conference* in 2013. Unfortunately, too many details have been left out in order to apply it in practice. Hopefully, a more detailed paper will be published in the future.

▷ Gatheral & Jacquier (2014) presents a calibration method based on Gatheral's SVI model. The model and several parameterizations hereof is presented in detail in the paper. The actual calibration of the call surface is carried out in a slice-by-slice manor where arbitrage is eliminated using penalization, which is very vaguely described. Hence, this method too has been disqualified due to insufficient details.

Common to all of the methods is that the authors claim that their specific method is particularly fast, accurate and fits the market well. But experience reveals that the exact choice of method can have severe implications for the shape of the resulting local volatility function. Thus, the aim of this paper is to examine the methods listed above in detail in order to assess their qualities and weaknesses by considering the following parameters:

- ▷ **Accuracy** Ability to fit observed market prices. There is some discussion in the literature on whether an exact fit should be preferred to an approximated, see Andreasen & Huge (2011) and Coleman *et al.* (1999).
- ▷ **Smoothness** The shape of the local- and implied volatility surface is ideally smooth and continuous, without any odd spikes or areas of missing data.
- ▷ **Speed** *Greed for speed* is one of the predominant factors in financial applications.
- ▷ **Robustness** The stability of the surfaces towards changes in the input data and across time.

From this thorough assessment it should be possible to conclude whether or not one of the methods stands out from the rest as being superior. Note, that the methods are not enhanced in this setup, but only evaluated in the form proposed by the authors. Furthermore, the assessment is limited to the *interpolation* of the observed data, as extrapolation of the wings is a different research topic.

2 Setting the Scene

The comparison of the calibration methods presented in this paper, builds on the empirical evidence presented in later sections. Hence, as the theoretical aspect is not the main focus here and the details presented in this section are therefore kept at a minimum. For an elaborate description the reader is referred to other sources such as Björk (2009).

2.1 Normed call prices

The methods to be compared vary with respect to the parameters they take as input. Some methods allow for deterministic interest rates and dividend yields, some for constant rates, while others only allow for one or no parameters, see the overview in table 1.

Method	r_τ	δ_τ
Benko *et al.* (2007)	Deterministic	0
Fengler (2009)	Deterministic	Deterministic
Glaser & Heider (2012)	Constant	Constant
Andreasen & Huge (2011)	0	0
Fengler & Hin (2013)	Deterministic	Deterministic

Table 1: Interest rates and dividend yield formats for the methods.

Thus, in order to compare these methods on *common grounds* we here specify a unified theoretical framework consisting of *normed call prices* [1] and transform our datasets for the empirical work accordingly. This transformation of the framework allow us to treat the data as if we were in a scenario with zero interest rate, $r_\tau = 0$, and dividends, $\delta_\tau = 0$.

Please note that this *normalization* of the framework eliminates the possibility of evaluating the methods' ability to handle interest rates and dividend yields. Although this is not optimal, the alternative solution would have been to start adjusting some of the methods to allow for deterministic rates. Such adjustments has been deemed outside the scope this paper.

Normed call prices is just another way of quoting prices, similar to using implied volatilities, both quotes are derived from observed call prices using the

[1] The term *normed call prices* has been picked up from Gope & Fries (2011) who describes this as a *more natural coordinate system.*

Black-Scholes formula. This transformation from ordinary European call prices to normed prices is here carried out along the lines of Gope & Fries (2011, sec. 2).

We start by reformulating the Black-Scholes formula into terms of the forward-moneyness: $\kappa = \frac{K}{F_0^\tau}$, where F_0^τ is the forward price given by: $F_0^\tau = se^{(r_\tau - \delta_\tau)\tau}$. The Black-Scholes formula for positive interest rates and dividend yields can then be reformulated as:

$$\begin{aligned}
& C_s(K,\tau) && = se^{-\delta_\tau \tau}\mathcal{N}(d_1) - Ke^{-r_\tau \tau}\mathcal{N}(d_2) \\
\Leftrightarrow\quad & \frac{C_s(K,\tau)}{se^{-\delta_\tau \tau}} && = \mathcal{N}(d_1) - \frac{Ke^{-r_\tau \tau}}{se^{-\delta_\tau \tau}}\mathcal{N}(d_2) \\
\Leftrightarrow\quad & \frac{C_s(K,\tau)e^{r_\tau \tau}}{F_0^\tau} && = \mathcal{N}(d_1) - \frac{K}{F_0^\tau}\mathcal{N}(d_2) \\
\Leftrightarrow\quad & \frac{C_s(K,\tau)e^{r_\tau \tau}}{F_0^\tau} && = \mathcal{N}(d_1) - \kappa\mathcal{N}(d_2)\,. \qquad (1)
\end{aligned}$$

where

$$\begin{aligned}
d_1 &= \frac{\ln\left(\frac{s}{K}\right) + \left(r_\tau - \delta_\tau + \frac{1}{2}\sigma^2\right)\tau}{\sigma\sqrt{\tau}} \\
&= \frac{\ln\left(\frac{s}{K}\right) + \ln\left(e^{(r_\tau - \delta_\tau)\tau}\right) + \frac{1}{2}\sigma^2\tau}{\sigma\sqrt{\tau}} = \frac{\ln\left(\frac{se^{(r_\tau-\delta_\tau)\tau}}{K}\right) + \frac{1}{2}\sigma^2\tau}{\sigma\sqrt{\tau}} \\
&= \frac{\ln\left(\frac{1}{\frac{K}{se^{(r_\tau-\delta_\tau)\tau}}}\right) + \frac{1}{2}\sigma^2\tau}{\sigma\sqrt{\tau}} = \frac{\ln\left(\frac{1}{\kappa}\right) + \frac{1}{2}\sigma^2\tau}{\sigma\sqrt{\tau}}, \\
d_2 &= d_1 - \sigma\sqrt{\tau}.
\end{aligned}$$

The right-hand-side of eqn. (1) is equivalent to the Black-Scholes formula for a European call option with strike price κ, maturity τ and a value of the underlying equal to 1. Hence, this relation defines the transformation from the original call prices to normed call prices:

$$C_1(\kappa,\tau) = \frac{C_s(K,\tau)e^{r_\tau \tau}}{F_0^\tau}. \qquad (2)$$

This transformation *incorporates* the interest rate and dividend yield into the strike price of the option, κ, making the normed call prices equivalent to zero interest rate, zero dividend yield products. Hence, these normed prices are ideal for our comparison analysis, as they are independent of the parameter assumptions in the various methods.

2.2 The local volatility model

The local volatility model, also termed the generalized Black-Scholes model, was developed to fit the implied volatility smile and the implied volatility term structure evident in the market after the stock crash of October 1987, see (Rasmussen, 2012, sec. 2.1) for details. The model is characterized by having a deterministic volatility function dependent on the time and the value of the underlying at that given time.

The local volatility is a non-parametric, deterministic function depending on the asset price and the time. A priori unknown, it must be computed numerically from option prices, or equivalently, from the implied volatility surface. (Fengler (2009))

In Dupire (1994)[2] a functional form for this local volatility function is presented such that the risk-neutral diffusion process given by:

$$\begin{aligned} dS_u &= (r_u - \delta_u) S_u du + \sigma(S_u, u) S_u dW^Q(u) \qquad && u \in (t, T] \\ S_t &= S > 0 && 0 \leq t. \end{aligned} \tag{3}$$

is a Q-martingale consistent with a continuum of observed arbitrage-free call prices:

$$c(K,u) = \mathbb{E}^{\mathbb{Q}}\left[(S_u - K)^+ \,|S_0 = s\right].$$

The local volatility function are according to Dupire (1994) uniquely given using the derivatives of these observed call prices:

$$\sigma^{loc}(K,\tau) = \sqrt{\frac{2\left(\frac{\partial c(K,\tau)}{\partial \tau} + \delta_\tau c\,(K,\tau) + (r_\tau - \delta_\tau) K \frac{\partial c(K,\tau)}{\partial K}\right)}{K^2 \frac{\partial^2 c(K,\tau)}{\partial K^2}}}. \tag{4}$$

The local volatility formula presented by Dupire (1994) must, like the call prices themselves, be transformed in order to fit our *normed* framework described previously. This transformation takes the relation given in eqn. (2) as a starting point,

[2] Here we present a version which allows for a deterministic interest rate and dividend yield which differs slightly from the original representation.

and can be found (Gope & Fries, 2011, sec. 2.8) in full detail. Here, we just repeat the intermediate results from this derivation:

$$\begin{aligned}
\frac{\partial c\left(K,\tau\right)}{\partial \tau} =& se^{-\delta_\tau \tau}\left[-\delta_\tau c_1(\kappa,\tau) - \kappa\left(r_\tau - \delta_\tau\right)\frac{\partial c_1(\kappa,\tau)}{\partial \kappa} + \frac{\partial c_1(\kappa,\tau)}{\partial \tau}\right],\\
\frac{\partial c\left(K,\tau\right)}{\partial K} =& \frac{se^{-\delta_\tau \tau}}{F_0^\tau}\frac{\partial c_1(\kappa,\tau)}{\partial \kappa},\\
\frac{\partial^2 c\left(K,\tau\right)}{\partial K^2} =& \frac{se^{-\delta_\tau \tau}}{\left(F_0^\tau\right)^2}\frac{\partial^2 c_1(\kappa,\tau)}{\partial \kappa^2},
\end{aligned} \tag{5}$$

Inserting these equations into Dupires formula in eqn. (4), this gives us the *normed* local volatility function:

$$\sigma^{loc}(K,\tau) = \sqrt{\frac{2\frac{\partial c_1(\kappa,\tau)}{\partial \tau}}{\kappa^2\frac{\partial^2 c_1(\kappa,\tau)}{\partial \kappa^2}}} = \sigma_1^{loc}(\kappa,\tau). \tag{6}$$

Perhaps not surprisingly, this transformation eliminate the interest rate and dividend yield by incorporating these amounts into the strike price, as were the case for the call price. What is more interesting though, is the relation: $\sigma^{loc}(K,\tau) = \sigma_1^{loc}(\kappa,\tau)$, stating how the normed local volatility surface corresponds to the original framework.

This normed local volatility function can then be be used to derive the local volatility surface using a suitable grid of fitted call prices. Only, one of the methods does not approximate the call surface, but the implied volatility surface, see table 2 below. In order to utilize the derivatives of the implied volatility surface, this method provide as a bi-product to the optimization, an alternative version of Dupire's formula given in (Gatheral, 2006, eqn. 1.10) is provided here:

$$\sigma_1^{loc}(\kappa,\tau) = \sqrt{\frac{\frac{\partial w}{\partial \tau}}{1 - \frac{\log\kappa}{w}\frac{\partial w}{\partial \log\kappa} + \frac{1}{4}\left(-\frac{1}{4} - \frac{1}{w} + \frac{(\log\kappa)^2}{w^2}\right)\left(\frac{\partial w}{\partial \log\kappa}\right)^2 + \frac{1}{2}\frac{\partial^2 w}{\partial(\log\kappa)^2}}} \tag{7}$$

where w is the total variance: $w_1(\kappa,\tau) = \sigma_1^{imp}(\kappa,\tau)^2\tau$.

2.3 ABSENCE OF ARBITRAGE

The basic requirement on the generated surfaces is absence of arbitrage opportunities. It is well known that if the surface fails to fulfill this property, it could lead to mispricings and false greeks, see the quote by Fengler (2009) below. Thus, absence of arbitrage is one of the pillars in mathematical finance.

The pricing accuracy and pricing performance of local volatility models depends on the absence of arbitrage in the implied volatility surface...surface that is not arbitrage-free can result in negative transition probabilities and consequently mispricings and false greeks. (Fengler (2009))

Carr & Madan (2005) point out that determining whether a given set of option prices is free of arbitrage, and thus whether a continuous time martingale exists[3], can prove difficult in practice as one must specify the structure of all possible price paths. That is, one must beforehand determine which arbitrage-free continuous time model the observed call prices stem from. Trivially, this model is unknown as it is only possible to observe past prices quoted in discrete time.

This problem can be turned upside down such that one instead would ask whether an arbitrage-free continuous time model consistent with the observed call prices exists? That is, whether a model consistent with a given non-negative martingale, S, exists and generates the European call prices observed in the market:

$$c_1(\kappa,\tau) = \mathbb{E}^{\mathbb{Q}}\left[(S_\tau - \kappa)^+ \,|S_0 = 1\right].$$

Carr & Madan name this alternative form of arbitrage *static*. The term refers to a trading strategy where the position in the underlying is only allowed to depend on the current time and the corresponding value of the underlying itself at this time.

This definition of an arbitrage opportunity has afterwards become the standard in the literature and in 2010 Roper formalizes this for the call surface:

...a call option price surface $(K,\tau) \to C_s(K,\tau)$ is free from static arbitrage if and only if there exists a nonnegative Markov martingale, say S_τ, such that $C_s(K,\tau) = \mathbb{E}^{\mathbb{Q}}\left[(S_\tau - K)^+\right]$ for every $K,\tau \geq 0$. (Roper (2010, def. 2.1))

Here, we more specifically want to determine whether an arbitrage-free local volatility model, with diffusion process given in eqn. (3), can be determined. That is, we want to find an answer to the same question as Dupire, see quote below.

...given the arbitrage-free prices $C(K,T)$ of European calls of all strikes K and maturities T, is it possible to find a risk-neutral

[3] *First Fundamental Theorem of Finance*, see for instance Björk (2009, Thm. 10.22)

process for the spot in the form of a diffusion:

$$\frac{dS}{S} = r(t)dt + \sigma(S,t)dW$$

where the instantaneous volatility s is a deterministic function of the spot and of the time? (Dupire (1994))

The answer to this question, for the normed framework, was given in the previous section by the local volatility function in eqn.'s 6 and 7. Hence, we need a set of constraints on the surfaces generated by the various methods in order to secure that they are indeed free of static arbitrage.

Initially, we would like to get an overview of the surfaces generated by the respective methods and hence the *versions* of no-arbitrage conditions needed for evaluating these. This overview is given in table 2.

Method	Surface for which the no-arbitrage constraints is applied
Benko *et al.* (2007)	Implied volatility (state price density)
Fengler (2009)	Call price
Andreasen & Huge (2011)	Call price
Glaser & Heider (2012)	Call price
Fengler & Hin (2013)	Call price

Table 2: Overview of the no-arbitrage constraints needed to evaluate the collection of methods.

It can be noted from table 2 that the majority of the methods generate a surface of approximated, or interpolated, call prices for which the no-arbitrage constraints are applied. The method by Benko *et al.* stands out by instead approximating the implied volatility surface and by using a mix of constraints partly on the state price density - or implied risk-neutral density - and partly the implied volatility function itself. The reason for this mix will later become apparent.

In correspondence with table 2, the no-arbitrage conditions for the normed call surface, the implied vol surface and the surface of implied risk-neutral densities, will be presented in the following.

- **Normed European call prices** We here repeat the sufficient conditions for absence of *static* arbitrage in the normed call surface presented in Fengler

& Hin (2013). These conditions are a slightly less general version of the originals given in Roper (2010), further simplified by the transformation to the normed framework.

Proposition 1. *Define the function* $C_1 : [0,\infty) \times [0,\infty) \to \mathbb{R}$ *such that* $C_1(\kappa, \tau) \ldots$

(C1) *is convex in* κ *for all* $\tau \geq 0$*,*

(C2) *is bounded according to*

$$(1-\kappa)^+ \leq C_1(\kappa,\tau) \leq 1, \qquad \forall \kappa \geq 0,\ \tau \geq 0.$$

(C3) *obeys* $C_1(\kappa, 0) = (1-\kappa)^+$ *for all* κ*,*

(C4) *has* $\lim_{\kappa\to\infty} C_1(\kappa,\tau) = 0$ *for all* τ*,*

(C5) *and is non-decreasing in* τ *for all* $\kappa \geq 0$*.*

Then there exists a non-negative Markov martingale M_τ *such that*

$$C_1(\kappa,\tau) = \mathbb{E}^{Q}\left[(M_\tau - \kappa)^+ | \mathcal{F}_0\right], \qquad \forall \kappa, \tau \geq 0.$$

■ **Implied volatility**

The sufficient conditions for absence of *static* arbitrage in the implied volatility surface are originally given in Roper (2010). Here, they are repeated in a slightly less general version along the lines of the conditions presented above.

Define the surface of *time scaled implied volatilities (in log-moneyness form)*:

$$\begin{aligned} v : \mathbb{R}\times[0,\infty]) &\to [0,\infty) \\ (\tilde{\kappa},\tau) &\mapsto \sqrt{\tau}\sigma_1^{imp}(\exp(\tilde{\kappa})), \qquad \tilde{\kappa} = \ln\kappa. \end{aligned} \tag{8}$$

Proposition 2 (Roper (2010))**.** *Define the function* $v : \mathbb{R}\times[0,\infty) \to \mathbb{R}$ *such that* $v(\tilde{\kappa},\tau)$

(IV1) *is twice differentiable in* $\tilde{\kappa}$ *for all* $\tau > 0$*,*

(IV2) *is bounded according to*

$$0 < v(\tilde{\kappa},\tau) \qquad \forall \tilde{\kappa} \in \mathbb{R},\ \tau > 0,$$

(IV3) fulfills the so-called Durrleman's condition

$$0 \leq \left(1 - \frac{\tilde{\kappa}}{\nu(\tilde{\kappa},\tau)}\frac{\partial \nu(\tilde{\kappa},\tau)}{\partial \tilde{\kappa}}\right)^2 - \frac{1}{4}\nu(\tilde{\kappa},\tau)^2\left(\frac{\partial \nu(\tilde{\kappa},\tau)}{\partial \tilde{\kappa}}\right)^2 + \nu(\tilde{\kappa},\tau)\frac{\partial^2 \nu(\tilde{\kappa},\tau)}{\partial \tilde{\kappa}^2}$$

for all $\tilde{\kappa} \in \mathbb{R}$ and $\tau > 0$,

(IV4) obeys $\nu(\tilde{\kappa}, 0) = 0$ for all $\tilde{\kappa} \in \mathbb{R}$,

(IV5) has $\lim_{\tilde{\kappa}\to\infty}\left(\frac{\nu(\tilde{\kappa},\tau))}{2} - \frac{\tilde{\kappa}}{\nu(\tilde{\kappa},\tau)}\right) = -\infty$ for all $\tau > 0$,

(IV6) and is non-decreasing in τ for all $\tilde{\kappa} \in \mathbb{R}$.

Then there exists a non-negative Markov martingale M_τ such that

$$C_1(\kappa,\tau) = \mathbb{E}^{Q}\left[(M_\tau - \kappa)^+ | \mathcal{F}_t\right], \qquad \forall \kappa, \tau \geq 0.$$

It is evident that the constraints regarding the implied volatility surface are far more complicated to work with in practice than the corresponding set for the normed call surface. Thus, proposition 2 is mainly stated here as an illustration of why Benko *et al.* instead choose to work with a set of constraints on the implied risk-neutral density function[4] for the forward-moneyness dimension. These text-book conditions for the risk-neutral density function are given below.

■ Implied risk-neutral density

- ▫ Nonnegativity property:

$$\varphi_1^{imp}(\kappa;\tau) \geq 0, \qquad \kappa \geq 0,\ \tau \geq 0. \tag{9}$$

- ▫ Integrability property:

$$\int_0^\infty \varphi_1^{imp}(\kappa;\tau)d\kappa = 1 \qquad \tau \geq 0. \tag{10}$$

Note the conditions listed above are sufficient to exclude arbitrage opportunities for various types of surfaces. Unfortunately, our limited framework only allows us to assess some of them. As stated earlier, we have limited our analysis to *interpolation* of surfaces. Hence, we are in this framework not able to assess the conditions given above relying on *extrapolation*: prop. 1 (C3,C4), prop. 2 (IV4, IV5) and the conditions in eqn. (10)).

[4]The implied risk-neutral density function can be derived from the implied volatility surface and it's derivatives, see eqn (14).

Method	Condition(s) Forward-moneyness	Condition(s) Time to maturity
Benko *et al.* (2007)	eqn.(9)	prop. 2: IV6
Fengler (2009)	prop. 1: C1, C2	prop. 1: C5
Andreasen & Huge (2011)	prop. 1: C1	prop. 1: C5
Glaser & Heider (2012)	prop. 1: C1	prop. 1: C5
Fengler & Hin (2013)	prop. 1: C1, C2	prop. 1: C5

Table 3: Overview of arbitrage conditions applied to the various methods.

In sections titled '*No-Arbitrage Conditions*' we give a comprehensive evaluation of the arbitrage conditions applied for each method and relate this to the conditions presented in this section. As a preview, table 3 summarizes which of the sufficient arbitrage conditions have been applied to the respective methods.

3 Presentation of the methods

In this section we give the reader an overview and a solid basic understanding of the various methods assessed, and their characteristics. The descriptions do however not give a complete review of all details and theoretical aspects of the techniques encountered, for this the reader is referred to the original articles and their respective references.

The main features which sets the methods apart are summarized in table 4. Here it can be observed that the calibrated surfaces, consisting of either call prices or implied volatilities, have all been either interpolation or approximated. If the surface is approximated by a spline, the methods can furthermore differ with respect to the type and/ or representation hereof. The last column indicates whether the surface has been fitted *slice-by-slice* (by a function for each maturity), *locally* (by functions covering specific grid-points), or *globally* (by a function covering the entire grid).

Method	Construction	Scope
Benko *et al.* (2007)	Approx ∼ quadratic polynomial	Local
Fengler (2009)	Approx ∼ natural cubic spline	Slice-by-slice
Andreasen & Huge (2011)	Interpolation ∼ PDE	Slice-by-slice
Glaser & Heider (2012)	Approx ∼ quadratic polynomial	Local
Fengler & Hin (2013)	Approx ∼ tensor-product B-spline	Global

Table 4: Construction methods for the surfaces.

The descriptions given in this section have for clarity been fitted into a unified framework consisting of 5 subsections:

- ▷ *The general idea* - a brief summary of the method.
- ▷ *Details of the method* - technical details of the approximation/ interpolation procedure are described in this part.
- ▷ *No-arbitrage conditions* - the arbitrage conditions applied for the generated surface are compared to the sufficient conditions given in sec. 2.3.
- ▷ *Conversion to the Local Volatility surface* - the computation of the local volatility surface are given in terms of the variables generated by the method.
- ▷ *Tweaking the method including an algorithm* - description of how the parameters for the given method are adjusted. Please not that *tweaking* the

methods is here a subjective process where smoothness versus loss of information needs to be balanced. This has been carried out by a combination *eyeballing* the *goodness of fit* to the data and the shape of the output surfaces.

In addition to these, a pseudo-code algorithm is provided for each method, describing the calibration of the local volatility function on a user-defined grid: $\hat{\boldsymbol{\kappa}} \times \hat{\boldsymbol{\tau}}$. More details on how this grid is defined in practice are given in section 4.3.1.

3.1 Benko

3.1.1 The general idea

Benko *et al.* (2007) propose to estimate the implied volatility surface directly from market quotes assuming that the surface follows the regression model:

$$\sigma_1^{imp}(\kappa_i, \tau_i) = \hat{\sigma}_1^{imp}(\kappa_i, \tau_i) + \varepsilon_i, \qquad i = 1, \ldots, N. \tag{11}$$

where $\sigma_1^{imp}(\kappa_i, \tau_i)$ is the observed implied volatilities, ε_i represents the *market noise* and N is the total number of observations.

This model is estimated as a local polynomial regression problem across a range of user-specified maturities, $\tau_1, \ldots, \tau_L$, for one strike-level, κ, at a time. This estimation across maturity levels allows Benko *et al.* to impose the no-arbitrage conditions for the implied risk-neutral density in both directions. In the strike-direction a non-negativity constraint on the implied risk-neutral density is applied, and in the maturity-direction a constraint on the total-variance: $w_1(\kappa, \tau) = \sigma_1^{imp}(\kappa, \tau)^2 \tau$, is applied to ensure a strict increase.

3.1.2 Details of the method

The implied volatility surface is, for a given point (κ_i, τ_i), near (κ, τ), approximated using a two-dimensional local polynomial quadratic in κ and linear in τ:

$$\begin{aligned}\hat{\sigma}_1^{imp}(\kappa_i, \tau_i) &\overset{taylor}{\approx} \hat{\sigma}_1^{imp}(\kappa, \tau) + \frac{\partial \hat{\sigma}_1^{imp}(\kappa, \tau_l)}{\partial \kappa}(\kappa_i - \kappa) + \frac{\partial \hat{\sigma}_1^{imp}(\kappa, \tau)}{\partial \tau}(\tau_i - \tau) \\ &\quad + \frac{1}{2!}\frac{\partial^2 \hat{\sigma}_1^{imp}(\kappa, \tau)}{\partial \kappa^2}(\kappa_i - \kappa)^2 + \frac{\partial \hat{\sigma}_1^{imp}(\kappa, \tau_l)}{\partial \kappa \partial \tau}(\kappa_i - \kappa)(\tau_i - \tau) \\ &= \alpha_{0,l} + \alpha_{1,l}(\kappa_i - \kappa) + \alpha_{2,l}(\tau_i - \tau) + \alpha_{3,l}(\kappa_i - \kappa)^2 \\ &\quad + \alpha_{4,l}(\kappa_i - \kappa)(\tau_i - \tau).\end{aligned} \tag{12}$$

Let in the following this polynomial be denoted $p_{\kappa,\tau}(\kappa_i, \tau_i)$.

This polynomial is then fitted to the observed input data for a given strike-level, κ, and a whole range of user-specified maturities, $\tau_1, \ldots, \tau_L$, at a time in order to apply the cross-maturity no-arbitrage conditions (see sec. 3.1.3). In practice this is carried out by solving the weighted least squares problem:

$$\min_{\alpha} \sum_{l=1}^{L} \sum_{i=1}^{N} \mathcal{K}_H(\kappa - \kappa_i, \tau_l - \tau_i)\left(\sigma_1^{imp}(\kappa_i, \tau_i) - p_{\kappa,\tau_l}(\kappa_i, \tau_i; \alpha)\right)^2, \tag{13}$$

where $\boldsymbol{\alpha}$ is a $L \times 5$ matrix of estimated coefficients for all the maturities in the grid.

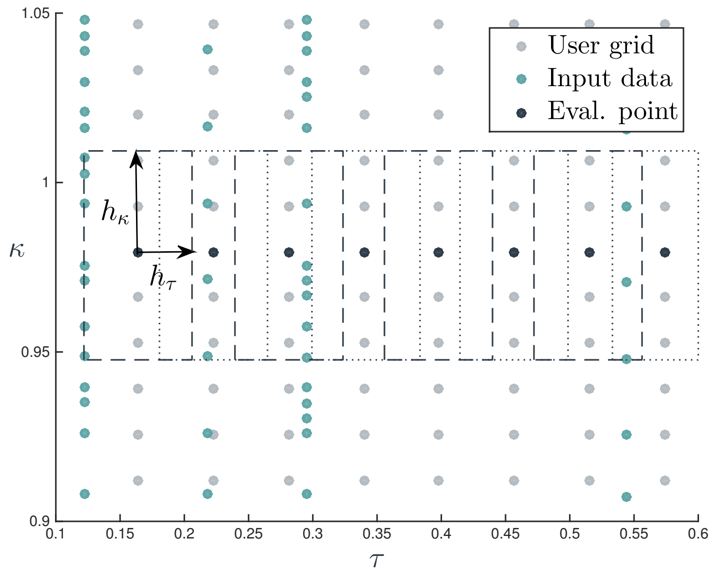


Figure 2: Grid-points for all maturities in the user-grid are fitted *simultaneously* based on a *local* subset of the observed data points.

The kernel function is given as the product of two *Epanechnikov kernels:*

$$\mathcal{K}_H(\kappa - \kappa_i, \tau_l - \tau_i) = \frac{1}{h_\kappa}\mathcal{K}\left(\frac{\kappa - \kappa_i}{h_\kappa}\right)\frac{1}{h_\tau}\mathcal{K}\left(\frac{\tau_l - \tau_i}{h_\tau}\right), \quad \mathcal{K}(u) = \frac{3}{4}\left(1 - u^2\right)\mathbf{1}_{|u|\leq 1}.$$

where the parameters: h_κ, h_τ, determine the *locality* of the estimation, also termed the *bandwidth*. Points outside the area: $[\kappa - h_\kappa; \kappa + h_\kappa] \times [\tau_l - h_\tau; \tau_l + h_\tau]$, will be disregarded from the estimation, while points inside will be assigned a given weight according to the kernel. This rectangle of data defined for each point, (κ, τ), is illustrated in figure 2. The parameters h_κ and h_τ needs to be chosen carefully, otherwise the method will tend to either under- or oversmooth the data.

3.1.3 No-arbitrage conditions

The sufficient no-arbitrage conditions on the time scaled implied volatility surface are given in proposition 2, where it can be noted that the cross-strike conditions are rather complex to impose in practice. Benko *et al.* proposes a workaround to avoid these constraints by switching to the implied risk-neutral density using the formula[5]:

$$\begin{aligned}\varphi_1^{imp}(\kappa;\tau) = \sqrt{\tau}N'(d_1)\Bigg(&\frac{1}{\kappa^2\tau\sigma_1^{imp}(\kappa,\tau)} + 2\frac{d_1}{\kappa\sqrt{\tau}\sigma_1^{imp}(\kappa,\tau)}\frac{\partial\sigma_1^{imp}(\kappa,\tau)}{\partial\kappa}\\ &+\left(\frac{d_1(d_1-\sigma_1^{imp}(\kappa,\tau)\sqrt{\tau})}{\sigma_1^{imp}(\kappa,\tau)}\right)\left(\frac{\partial\sigma_1^{imp}(\kappa,\tau)}{\partial\kappa}\right)^2\\ &+\frac{\partial^2\sigma^{imp}(\kappa,\tau)}{\partial\kappa^2}\Bigg),\quad d_1 = \frac{\frac{1}{2}\sigma_1^{imp}(\kappa,\tau)^2\tau - \ln(\kappa)}{\sigma_1^{imp}(\kappa,\tau)\sqrt{\tau}}.\end{aligned} \tag{14}$$

The constraints on this surface consist of eqn.'s (9) and (10) where the authors neglect to consider the latter. The global condition given by eqn. (10) requires knowledge of the density in the tails, and as this method does not handle extrapolation Benko *et al.* have to disregard this condition as well. Hence, the no-arbitrage condition for the strike-direction is given by the non-negativity condition:

$$\varphi_1^{imp}(\kappa;\tau) \geq 0.$$

The cross-maturity condition, given in prop. 2 (IV6) for the time scaled implied volatility, is on the other hand simple to impose, as it only consists of a non-decreasing constraint. In the paper, this constraint is rephrased in terms of total variance which must be strictly increasing:

$$\frac{\partial\sigma_1^{imp}(\kappa,\tau)^2\tau}{\partial\tau} > 0 \quad\Leftrightarrow\quad 2\alpha_0\alpha_1\tau > 0,$$

The derivative has in the last equation been rewritten in terms of regression coefficients, details on this are given below in sec. 3.1.4. As the estimation is carried out for a range of maturities, this constraint is extended to also include: $\sigma_1^{imp}(\kappa,\tau_l)^2\tau_l \leq \sigma_1^{imp}(\kappa,\tau_{l'})^2\tau_{l'}$, for each maturity-pair: $\tau_l < \tau_{l'}$.

The final method is summarized in algorithm 1.

3.1.4 Conversion to the Local Volatility Surface

Recall Dupire's local volatility function, $\hat{\sigma}_1^L(\kappa,\tau)$, given in terms of the implied volatility surface in eqn. (7):

$$\hat{\sigma}_1^L(\kappa,\tau) = \sqrt{\frac{\frac{\partial\hat{w}}{\partial\tau}}{1-\frac{\log\kappa}{\hat{w}}\frac{\partial\hat{w}}{\partial\log\kappa}+\frac{1}{4}\left(-\frac{1}{4}-\frac{1}{\hat{w}}+\frac{(\log\kappa)^2}{\hat{w}^2}\right)\left(\frac{\partial\hat{w}}{\partial\log\kappa}\right)^2+\frac{1}{2}\frac{\partial^2\hat{w}}{\partial(\log\kappa)^2}}}$$

[5]Eqn. (4) in Benko *et al.* (2007) adjusted to the normed framework.

with $\hat{w}$ being the approximated total variance: $\hat{w}_1(\kappa,\tau) = \hat{\sigma}_1^{imp}(\kappa,\tau)^2\tau$. This can be rewritten using the chain-rule for the derivatives wrt. $\log\kappa$:

$$\frac{\partial w}{\partial \log\kappa}\frac{\partial \log\kappa}{\partial\kappa} = \frac{\partial w}{\partial\kappa} \Leftrightarrow \frac{\partial w}{\partial\log\kappa} = \frac{\partial w}{\partial\kappa}\frac{1}{\frac{\partial\log\kappa}{\partial\kappa}} = \frac{\partial w}{\partial\kappa}\kappa,$$

$$\begin{aligned}\frac{\partial^2 w}{\partial(\log\kappa)^2} &= \frac{\partial\frac{\partial w}{\partial\kappa}\kappa}{\partial\log\kappa} = \frac{\partial\frac{\partial w}{\partial\kappa}}{\partial\log\kappa}\kappa + \frac{\partial w}{\partial\kappa}\frac{\partial\kappa}{\partial\log\kappa}\\ &= \frac{\partial\frac{\partial w}{\partial\kappa}}{\partial\kappa}\kappa^2 + \frac{\partial w}{\partial\kappa}\kappa = \frac{\partial^2 w}{\partial\kappa^2}\kappa^2 + \frac{\partial w}{\partial\kappa}\kappa.\end{aligned}$$

Using the relations above gives the local volatility function in terms of the partial derivatives wrt. τ and κ:

$$\begin{aligned}\sigma_1^L(\kappa,\tau) &= \sqrt{\frac{\frac{\partial\hat{w}}{\partial\tau}}{1 - \frac{\log\kappa}{\hat{w}}\frac{\partial\hat{w}}{\partial\kappa}\kappa + \frac{1}{4}\left(-\frac{1}{4} - \frac{1}{\hat{w}} + \frac{(\log\kappa)^2}{\hat{w}^2}\right)\left(\frac{\partial\hat{w}}{\partial\kappa}\kappa\right)^2 + \frac{1}{2}\left(\frac{\partial^2\hat{w}}{\partial\kappa^2}\kappa^2 + \frac{\partial\hat{w}}{\partial\kappa}\kappa\right)}}\\ &= \sqrt{\frac{\frac{\partial\hat{w}}{\partial\tau}}{1 + \kappa\left(\frac{1}{2} - \frac{\log\kappa}{\hat{w}}\right)\frac{\partial\hat{w}}{\partial\kappa} + \frac{1}{4}\kappa^2\left(-\frac{1}{4} - \frac{1}{\hat{w}} + \frac{(\log\kappa)^2}{\hat{w}^2}\right)\left(\frac{\partial\hat{w}}{\partial\kappa}\right)^2 + \frac{1}{2}\frac{\partial^2\hat{w}}{\partial\kappa^2}\kappa^2}}\end{aligned} \tag{15}$$

Using the regression coefficients defined in eqn. (12), the derivatives in eqn. (15) can now be formulated in terms of the output from the optimization procedure:

$$\begin{aligned}\frac{\partial\hat{w}}{\partial\tau} &= \frac{\partial(\hat{\sigma}_1^{imp}(\kappa,\tau)^2\tau)}{\partial\tau} = \frac{\partial\hat{\sigma}_1^{imp}(\kappa,\tau)^2}{\partial\tau}\tau + \hat{\sigma}_1^{imp}(\kappa,\tau)^2\\ &= 2\hat{\sigma}_1^{imp}(\kappa,\tau)\frac{\partial\hat{\sigma}_1^{imp}(\kappa,\tau)}{\partial\tau}\tau + \hat{\sigma}_1^{imp}(\kappa,\tau)^2\\ &= 2\hat{\alpha}_0\hat{\alpha}_2\tau + \hat{\alpha}_0^2.\\ \frac{\partial\hat{w}}{\partial\kappa} &= \frac{\partial(\hat{\sigma}_1^{imp}(\kappa,\tau)^2\tau)}{\partial\kappa} = \frac{\partial\hat{\sigma}_1^{imp}(\kappa,\tau)^2}{\partial\kappa}\tau\\ &= 2\hat{\sigma}_1^{imp}(\kappa,\tau)\frac{\partial\hat{\sigma}_1^{imp}(\kappa,\tau)}{\partial\kappa}\tau\\ &= 2\hat{\alpha}_0\hat{\alpha}_1\tau.\\ \frac{\partial^2\hat{w}}{\partial\kappa^2} &= \frac{\partial\frac{\partial\hat{w}}{\partial\kappa}}{\partial\kappa} = \frac{\partial\left(2\hat{\sigma}_1^{imp}(\kappa,\tau)\frac{\partial\hat{\sigma}_1^{imp}(\kappa,\tau)}{\partial\kappa}\tau\right)}{\partial\kappa}\\ &= 2\left(\frac{\partial\hat{\sigma}_1^{imp}(\kappa,\tau)}{\partial\kappa}\frac{\partial\hat{\sigma}_1^{imp}(\kappa,\tau)}{\partial\kappa} + \hat{\sigma}_1^{imp}(\kappa,\tau)\frac{\partial^2\hat{\sigma}_1^{imp}(\kappa,\tau)}{\partial\kappa^2}\right)\tau\\ &= 2\left(\hat{\alpha}_1^2 + \hat{\alpha}_0 2\hat{\alpha}_3\right)\tau.\end{aligned}$$

3.1.5 Tweaking the method

Evaluate surface in specified point $(\bar{\kappa}, \bar{\tau})$: The method approximates values for a user-specified grid of points:

$$\hat{\boldsymbol{\kappa}} \times [\hat{\tau}_1, \hat{\tau}_L] = \hat{\boldsymbol{\kappa}} \times \hat{\boldsymbol{\tau}} \subseteq \{\kappa_i \times \tau_i\}_{i=1,\dots,N}$$

These grid-points are allowed to be non-uniformly spaced in both directions, but must be contained within the rectangle defined by the original grid. Thus, to obtain a value for a point $(\bar{\kappa}, \bar{\tau})$ is must be contained in this user-specified grid given as input to the optimization method.

▷ For all methods we wish to obtain estimates of the surfaces for a grid consisting of all observed points, (κ_i, τ_i), as well as the user-specified points, $(\hat{\kappa}, \hat{\tau})$.

  But the optimization problem (13) approximates the surface simultaneously across all the given maturity levels. Hence, this method presented by Benko *et al.* is extremely expensive in the number of user-supplied maturity levels.

  Here, we are therefore forced only to fit the surfaces for the observed maturity levels, $\{\tau_i\}_{i=1,\dots,N}$, and afterwards use linear interpolation to obtain values for the user-specified grid.

▷ Similar to the method by Glaser & Heider described below in sec. 3.4, the main tweaking for this method consists of setting parameters: h_κ, h_τ, determining the *locality* of the approximation for a given evaluation point.

  The examinations in this paper take the values suggested by Benko *et al.* (2007) as a starting point:

$$h_\kappa = 0.05, \quad h_\tau = \begin{cases} 0.2 & 0 < \tau \leq \frac{1}{3} \\ 0.3 & \frac{1}{3} < \tau \leq \frac{2}{3} \\ 0.4 & \frac{2}{3} < \tau \leq 1. \end{cases}$$

h_τ Our first aim is to adjust the parameter h_τ such that the number of data points with positive weight, given by the kernel for each grid point across the user-defined surface, is as evenly distributed as possible. The investigations do however indicate, that this is not possible using a globally set parameter, the observed data points are simply too unevenly distributed across the surface. In particular, it can be noted that it does not seem possible to increase the number of data points involved in approximating the largest maturities. These are therefore left with only $\frac{1}{2} - \frac{1}{3}$ of the points involved for the remaining surface. An alternative to using globally set parameters, is given below in section 3.4.5 for the Glaser & Heider (2012) method.

We do, however, see that reducing the size of h_τ for all maturities by 0.1 improves the fit and the quality of the local volatility surface. This improvement seems to be caused by a change in the weight that the kernel ascribes each data point, rather than by a change in the number of points involved in each local approximation. Especially for small forward-moneyness- and maturity levels, this change has a significant positive effect on the smoothness of the local volatility surface.

The varying levels across the surface seems fitting for our dataset as well, hence we end up using a parameter h_τ with levels:

$$h_\tau = \begin{cases} 0.1 & 0 < \tau \leq \frac{1}{3} \\ 0.2 & \frac{1}{3} < \tau \leq \frac{2}{3} \\ 0.3 & \frac{2}{3} < \tau \leq 1. \end{cases}$$

h_κ The distribution of the observed points across forward-moneyness levels is clearly *bell*-shaped with the highest density near at-the-money level. One could try to differ the parameter value for different forward-moneyness levels, as for the maturity direction. But as there is no issues regarding smoothness or fit, it is left as a constant.

Adjusting this parameter does not have the same significant, positive impact on the smoothness of the volatility surface, as were the case for the maturity-parameter. Trying different levels ranging from 0.04 to 0.07, it is however evident that a value of $h_\kappa = 0.06$ results in the smoothest local volatility surface, while keeping the relative distance to the observations at a reasonable level, so we choose this level for the empirical comparison.

The quality of the implied volatility surface does not change visibly when adjusting the parameters: h_κ, h_τ. But one can see that the approximation is rather poor for values outside the *cone*, defined by the data observations. Hence, this method is not suited for even the slightest amount of extrapolation. Later we will observed that this is also the case for the 'local' method by Glaser & Heider in section 3.5.5, while the other methods does not seem to experience this level of difficulty.

Algorithm 1: Local polynomial smoothing of the implied volatility.

```
input: $\tau_i$, $i=1,\ldots,N$                      ▷ observed maturity levels.
       $\kappa_i$, , $i=1,\ldots,N$                  ▷ observed forward-moneyness levels.
       $\sigma_1^{imp}(\kappa_i,\tau_i)$             ▷ observed implied volatilities.
       $h_\kappa, h_\tau$                            ▷ Parameters determining the locality.

Def: $\hat{\boldsymbol{\kappa}} \times [\hat{\tau}_1, \hat{\tau}_L] = \hat{\boldsymbol{\kappa}} \times \hat{\boldsymbol{\tau}} \subseteq \{\kappa_i \times \tau_i\}_{i=1,\ldots,N}$
  ▷ Grid-points for which the local approximation of the implied
    volatility surface is derived.  Contained in the rectangle
    defined by the input grid.  Allowed to be non-uniform.

Def: $\mathcal{K}_H(\kappa-\kappa_i, \tau-\tau_i) = \frac{1}{h_\kappa}\mathcal{K}\left(\frac{\kappa-\kappa_i}{h_\kappa}\right)\frac{1}{h_\tau}\mathcal{K}\left(\frac{\tau-\tau_i}{h_\tau}\right)$.
  ▷ Bivariate kernel function given by the Epanechnikov kernel:
    $\mathcal{K}(u) = \frac{3}{4}\left(1-u^2\right)\mathbf{1}\{|u| \le 1\}$, and the parameters:  $h_\kappa$, $h_\tau$.  The kernel
    determines the 'locality' of the approximation.

Def: $p_{\hat{\kappa},\hat{\tau}_l}(\kappa_i,\tau_i) = \alpha_{0,l} + \alpha_{1,l}(\kappa_i-\hat{\kappa}) + \alpha_{2,l}(\tau_i-\hat{\tau}_l)$
                          $+\alpha_{3,l}(\kappa_i-\hat{\kappa})^2 + \alpha_{4,l}(\kappa_i-\hat{\kappa})(\tau_i-\hat{\tau}_l)$
  ▷ Local polynomial quadratic in $\kappa$, linear in $\tau$ where:
    $\hat{\sigma}_1^{imp}(\hat{\kappa},\hat{\tau}_l) = \alpha_{0,l}$;  $\frac{\partial\hat{\sigma}_1^{imp}(\hat{\kappa},\hat{\tau}_l)}{\partial\kappa} = \alpha_{1,l}$;  $\frac{\partial\hat{\sigma}_1^{imp}(\hat{\kappa},\hat{\tau}_l)}{\partial\tau} = \alpha_{2,l}$;
    $\frac{\partial^2\hat{\sigma}_1^{imp}(\hat{\kappa},\hat{\tau}_l)}{\partial\kappa^2} = 2\alpha_{3,l}$;  $\frac{\partial^2\hat{\sigma}_1^{imp}(\hat{\kappa},\hat{\tau}_l)}{\partial\kappa\partial\tau} = \alpha_{4,l}$.

for $\hat{\kappa} \in \hat{\boldsymbol{\kappa}}$ do
|   Solve:  $\min_{\boldsymbol{\alpha}} \sum_{l=1}^{L}\sum_{i=1}^{N} \mathcal{K}_H(\hat{\kappa}-\kappa_i, \hat{\tau}_l-\tau_i)\left(\sigma_1^{imp}(\kappa_i,\tau_i) - p_{\hat{\kappa},\hat{\tau}_l}(\kappa_i,\tau_i)\right)^2$
|           s.t.                                                                                 ▷ where
|             $\sqrt{\hat{\tau}_l}\varphi(d_1(l))\left\{\frac{1}{\hat{\kappa}^2\alpha_{0,l}\hat{\tau}_l} + \frac{2d_1(l)}{\hat{\kappa}\alpha_{0,l}\sqrt{\hat{\tau}_l}}\alpha_{1,l} + \frac{d_1(l)d_2(l)}{\alpha_{0,l}}\alpha_{1,l}^2 + 2\alpha_{3,l}\right\} \ge 0$
|                  $2\tau_l\alpha_{0,l}\alpha_{2,l} + \alpha_{0,l}^2 > 0$;   $\alpha_{0,l}^2\hat{\tau}_l < \alpha_{0,l'}^2\hat{\tau}_{l'}$, $\hat{\tau}_l < \hat{\tau}'_l$.
|     $d_1(l) = \frac{\frac{\alpha_{0,l}^2\tau_l}{2} - \ln\kappa}{\alpha_{0,l}\sqrt{\tau_l}}$,   $d_2(l) = d_1(l) - \alpha_{0,l}\sqrt{\tau_l}$. Remark, the parameter $\boldsymbol{\alpha} \in \mathbb{R}^{L\times 5}$
|     contains the parameters for all maturities in the grid.
|
|   for $\tau_l \in \boldsymbol{\tau}$ do
|   |   $\hat{\sigma}_1^{imp}(\hat{\kappa},\hat{\tau}_l) = \alpha_{0,l}^*$;   $\frac{\partial w}{\partial\tau} = 2\alpha_{0,l}^*\alpha_{2,l}^*\hat{\tau}_l + \left(\alpha_{0,l}^*\right)^2$;
|   |   $\frac{\partial w}{\partial\kappa} = 2\alpha_{0,l}^*\alpha_{1,l}^*\hat{\tau}_l$;   $\frac{\partial^2 w}{\partial\kappa^2} = 2\left(\left(\alpha_{1,l}^*\right)^2 + \alpha_{0,l}^* 2\alpha_{3,l}^*\right)\hat{\tau}_l$;
|   |   $y = \log\hat{\kappa}$;   $w = \hat{\sigma}_1^{imp}(\kappa,\tau_l)^2\hat{\tau}_l$;
|   |   $\sigma_1^L(\hat{\kappa},\tau_l) = \sqrt{\frac{\frac{\partial w}{\partial\tau}}{1 + \hat{\kappa} + \frac{\partial w}{\partial\kappa}\left(\frac{1}{2}-\frac{y}{w}\right) + \frac{1}{2}\hat{\kappa}^2\frac{\partial^2 w}{\partial\kappa^2} - \frac{1}{4}\hat{\kappa}^2\left(\frac{\partial w}{\partial\kappa}\right)^2\left(\frac{1}{2}+\frac{1}{w}-\frac{y^2}{w^2}\right)}}$
```

3.2 Fengler

3.2.1 The general idea

Fengler (2009) assumes implicitly that the call prices observed in the market follows the regression model:

$$c_1(\kappa_i, \tau_i) = \hat{c}_1(\kappa_i, \tau_i) + \varepsilon_i \qquad i = 1, \ldots, N \tag{16}$$

where $c_1(\kappa_i, \tau_i)$ is the normed, observed call prices and ε_i is the error term representing the market noise. Thus, in order to fit the market, Fengler proposes an approximation method rather than an interpolation method.

Fengler (2009) proposes to approximate this unknown surface of call prices, $\hat{c}_1(\kappa_i, \tau_i)$, by a series of natural cubic splines, one for each *maturity-slice*. These splines are fitted to the observed data backwards in time, starting with the largest maturity, continuing backwards one slice at a time.

The optimization problem is formulated as minimizing a penalized sum of squares subject to a range of linear no-arbitrage constraints for both the strike- and maturity direction. Fengler exploits the linearity of these conditions to reformulate the optimization problem as a quadratic program.

3.2.2 Details of the method

The natural cubic splines used for the approximation are given by:

$$\begin{aligned} g_\tau(x) &= \textstyle\sum_{m=1}^{M} \mathbb{1}_{[\kappa_m, \kappa_{m+1})}(x) s_{m,\tau}(x), \\ s_{m,\tau}(x) &= a_{m,\tau} + b_{m,\tau}(x - \kappa_m) + c_{m,\tau}(x - \kappa_m)^2 + d_{m,\tau}(x - \kappa_m)^3. \end{aligned} \tag{17}$$

where $\kappa_1, \ldots, \kappa_M$ are a given sequence of spline knots.

These splines are for a given maturity, τ, fitted to the observed data by solving a penalized minimum least squares problem:

$$\min_{g_\tau} \quad \sum_{m=1}^{M} w_m \{c_1(\kappa_m, \tau) - g_\tau(\kappa_m)\}^2 + \lambda \int_{\kappa_1}^{\kappa_M} \{g_\tau''(x)\}^2 dx \tag{18}$$

with weights $w_m > 0$ and smoothing parameter $\lambda > 0$. The parameter λ determines the impact of the penalty term, given by the local variation, and thus the smoothness of the curve.

The optimization problem, (18), is solved with respect to a range of linear no-arbitrage constraints given in section 3.2.3. These are both cross-strike and cross-maturity constraints, where the latter relies on an underlying *regular* (in the sense of not being scattered) grid of observations.

The method is therefore initialized by *pre-smoothening* the scattered input data on a predefined regular grid: $[\hat{\kappa}_1; \hat{\kappa}_M] \times [\hat{\tau}_1; \hat{\tau}_L]$, using a two-dimensional non-parametric smoother. Fengler suggests that the smoothening is carried out in the implied volatility space and afterwards converted to call prices. These are, along with the knot sequence: $[\hat{\kappa}_1; \hat{\kappa}_M]$, given as input to the optimization problem (18).

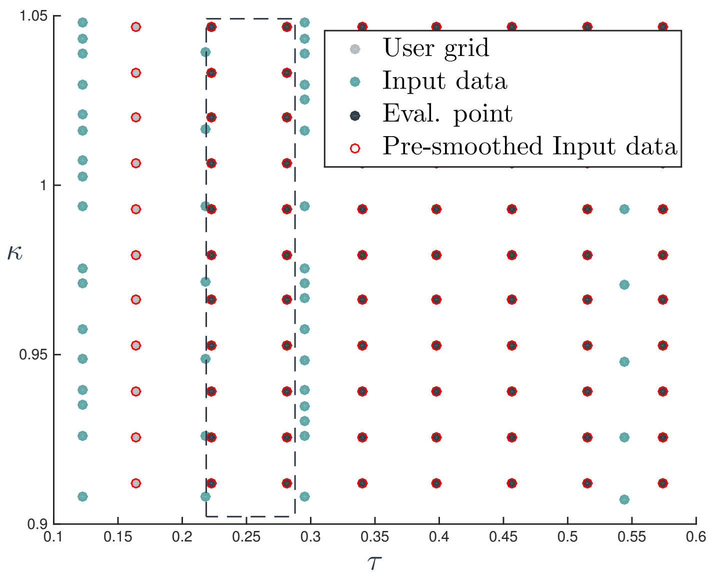


Figure 3: Grid-points for all forward-moneyness levels in the user-grid are fitted by a natural cubic spline to the pre-smoothed call surface.

3.2.3 No-arbitrage conditions

The no-arbitrage conditions Fengler applies to the minimization problem, (18), are given below in terms of the predefined grid $[\hat{\kappa}_1; \hat{\kappa}_M] \times [\hat{\tau}_1; \hat{\tau}_L]$:

$$g_{\hat{\tau}_l}{}''\left([\hat{\kappa}_2; \hat{\kappa}_{M-1}]\right) \geq 0, \tag{19}$$

$$\frac{g_{\hat{\tau}_l}(\hat{\kappa}_2) - g_{\hat{\tau}_l}(\hat{\kappa}_1)}{h_1} - \frac{h_1}{6} g_{\hat{\tau}_l}{}''(\hat{\kappa}_2) \geq -1 \tag{20}$$

$$-\frac{g_{\hat{\tau}_l}(\hat{\kappa}_M) - g_{\hat{\tau}_l}(\hat{\kappa}_{M-1})}{h_{M-1}} - \frac{h_{M-1}}{6} g_{\hat{\tau}_l}{}''(\hat{\kappa}_{M-1}) \geq 0 \tag{21}$$

$$g_{\hat{\tau}_l}(\hat{\boldsymbol{\kappa}}) \leq g_{\hat{\tau}_{l+1}}(\hat{\boldsymbol{\kappa}}) \quad \text{if } l < L \tag{22}$$

$$g_{\hat{\tau}_l}(\hat{\kappa}_1) \leq 1 \quad \text{if } l = L \tag{23}$$

$$g_{\hat{\tau}_l}(\hat{\kappa}_1) \geq 1 - \hat{\kappa}_1 \tag{24}$$

$$g_{\hat{\tau}_l}(\hat{\kappa}_M) \geq 0. \tag{25}$$

Comparing these to the sufficient no-arbitrage conditions in prop. 1, one can see that the conditions evaluated in this framework: *(C1), (C2), (C5)*, are all satisfied by eqn. (19)-(25):

(C1) The 2nd order derivative of a natural cubic spline is linear and zero at the end-knots. Thus, it is sufficient to apply the convexity constraint at the *internal* knot points, as in condition (19).

(C2) Fengler starts by applying a non-increasing constraint in the strike-direction given by eqn. (20) and (21). This is only applied to the end segments of the spline, as the convexity constraint insures a non-decreasing slope.

This monotonicity property reduces the boundary conditions to a few constraints on the end-segments: the lower boundary constraint is given as a combination of eqn. (24) and (25), while the upper boundary is given by eqn. (23).

(C5) The non-decreasing property for the maturity direction is for all maturities $< \hat{\tau}_L$ given directly by eqn. (22).

Note that the complexity of this constraint is significantly reduced compared to the original paper due to the *normed* framework applied here.

In the introduction it was briefly mentioned that the linearity of constraints (19) -(25) makes it possible to reformulate the optimization problem, as a quadratic program using the *value-second derivative representation* of the spline. The final method is listed in algorithms 2 and 3.

3.2.4 Conversion to the Local Volatility Surface

Dupire's local volatility function given in eqn. (6), $\hat{\sigma}_1^L(\kappa,\tau)$, is for this method given as a *hybrid* estimate, consisting of a central finite difference estimate replacing the derivative wrt. maturity:

$$\frac{\partial c_1(\kappa,\tau_l)}{\partial\tau} \approx \frac{c_1(\kappa,\tau_{l+1}) - c_1(\kappa,\tau_{l-1})}{\tau_{l+1}-\tau_{l-1}}. \tag{26}$$

The second order derivative wrt. forward-moneyness are given directly by the output of the quadratic program due to the *value-second derivative representation* of the spline, see algorithm 3.

Hence, the final estimate of the local volatility function is given by:

$$\hat{\sigma}_1^L(\kappa,\tau_l) = \sqrt{\frac{2\frac{\hat{c}_1(\kappa,\tau_{l+1})-\hat{c}_1(\kappa,\tau_{l-1})}{\tau_{l+1}-\tau_{l-1}}}{\kappa^2\frac{\partial^2\hat{c}_1(\kappa,\tau_l)}{\partial\kappa^2}}}.$$

Algorithm 2: Part I: Fengler (2009) - approximation of the call surface.

```
input: $\tau_i,\ i=1,\dots,N$                    ▷ observed maturity levels.
       $\kappa_i,\ ,\ i=1,\dots,N$              ▷ observed forward-moneyness levels.
       $\sigma_1^{imp}(\kappa_i,\tau_i)$          ▷ observed implied volatilities.
       $\lambda$                                ▷ smoothening parameter.

Def: $[\hat{\kappa}_1;\hat{\kappa}_M]\times[\hat{\tau}_1;\hat{\tau}_L] = \hat{\boldsymbol{\kappa}}\times\hat{\boldsymbol{\tau}} \subseteq \{\kappa_i\times\tau_i\}_{i=1,\dots,N}$
  ▷ Regular (non-scattered) grid for which the approximation of the
    call surface is derived.  Contained in the rectangle defined by
    the input grid.  Allowed to be non-uniform.

$\tilde{\sigma}_1^{imp}(\hat{\boldsymbol{\kappa}},\hat{\boldsymbol{\tau}}) = \textbf{Interpolate}\Big(\big\{\sigma_1^{imp}(\kappa_i,\tau_i)\big\}_{i=1,\dots,N}\Big).$
  ▷ Apply two-dimensional non-parametric smoother to obtain
    pre-estimate of the implied volatility surface.

$\tilde{c}_1(\hat{\boldsymbol{\kappa}},\hat{\boldsymbol{\tau}}) = \textbf{BS\_Call}\big(1,\hat{\boldsymbol{\kappa}},\hat{\boldsymbol{\tau}},\tilde{\sigma}_1^{imp}(\hat{\boldsymbol{\kappa}},\hat{\boldsymbol{\tau}})\big)$
  ▷ Convert pre-smoothed implied volatility surface to call surface.
```

3.2.5 Tweaking the method

Evaluate surface in specified point $(\bar{\kappa},\bar{\tau})$: The method pre-smoothes the input data on a user-specified grid:

$$\hat{\boldsymbol{\kappa}}\times[\hat{\tau}_1,\hat{\tau}_L] = \hat{\boldsymbol{\kappa}}\times\hat{\boldsymbol{\tau}} \subseteq \{\kappa_i\times\tau_i\}_{i=1,\dots,N}\,.$$

These grid-points are allowed to be non-uniformly spaced in both directions, but must form a regular - non scattered - grid and be contained within the rectangle defined by the observed data points.

These pre-smoothed values are then used to fit a natural cubic spline for each maturity level, τ_l. Thus, it should be possible to evaluate the surface for arbitrary forward-moneyness levels as long as the corresponding maturity is contained in the user-defined grid above. But this evaluations is not described in Fengler (2009) and the reader is therefore referred to Green & Silverman (1993) for details on spline evaluation when using the *value-second derivative representation*

- ▷ For all methods we wish to obtain estimates of the surfaces for a grid consisting of all observed points, (κ_i, τ_i), as well as the user-specified points, $(\hat{\kappa}, \hat{\tau})$.

  As mentioned in the paragraph above, this can in theory be achieved by pre-smoothening the data on a regular grid consisting of a reasonable number of forward-moneyness levels, and afterwards evaluate the splines for a much denser grid of forward-moneyness levels. This approach would in theory save us some computational time, although this gain could in practice be canceled out by an increase in the computational effort required for the optimization procedure.

  This does however not seem to be the approach used by Fengler (2009), so here we will be content with obtaining values for the combined grid (consisting of both the observed- and the user-defined grid-points) by including all grid-points in the user-specified grid, used in the pre-smoothening procedure and throughout the method.

- ▷ The pre-smoother used in these calculation is the *thin-plate-spline*, which is one of the methods suggested in the article.

- ▷ This method distinguishes itself by only having one *tweak-able* parameter given by the smoothening parameter, λ, determining the weight of the *roughness penalty*, given in terms of the second order derivative. The weight-values, w_i, could in theory also be altered, but this is rarely seen in the literature and this is therefore ignored here.

  Fengler finds that a value of $\lambda = 1e - 7$ is suitable for the DAX dataset he uses for the empirical demonstration. Here, we have tested different values: $\lambda = \{1e{-}5,\ 1e{-}7,\ 1e{-}9,\ 1e{-}11,\ 1e{-}13\}$, on a subpart of our dataset. From these samples, it is evident that a smoothing parameter $< 1e - 9$ smoothes the data too much. This can be seen on the implied volatility surface for the lowest maturity levels, as the surface here behaves irregularly, deviating from the observed data points for large and small forward-moneyness levels. For values $\geq 1e-9$ the implied volatility surface fits the observed data points nicely for all maturity levels.

Further looking at the local volatility surface, it can be seen that as λ increases, the number of *bumps and folds* in the surface increases as well. For $\lambda > 1e - 9$ this does not look like additional information being captured, but rather as noise. Hence, for the calculations used in this paper we have chosen the smoothing parameter $\lambda = 1e - 9$.

Algorithm 3: Part II: Fengler (2009) - approximation of the call surface.

```
Def: $\mathbf{g}_{\hat{\tau}_l} = (g_{\hat{\tau}_l}(\hat{\kappa}_1), \dots, g_{\hat{\tau}_l}(\hat{\kappa}_M))^\top \quad \gamma_{\hat{\tau}_l} = (g_{\hat{\tau}_l}{}''(\hat{\kappa}_2), \dots, g_{\hat{\tau}_l}{}''(\hat{\kappa}_{M-1}))^\top$
  ▷ The value-second derivative representation of the natural cubic
    spline, $g_{\hat{\tau}_l}(\cdot)$, approximating the call surface for a given maturity
    $\hat{\tau}_l \in \hat{\boldsymbol{\tau}}$.

Set:
$\mathbf{Q} = \begin{pmatrix} \frac{1}{h_1} & 0 & 0 & 0 & 0 \\ \left(-\frac{1}{h_1} - \frac{1}{h_2}\right) & \frac{1}{h_2} & 0 & 0 & 0 \\ \frac{1}{h_2} & \left(-\frac{1}{h_2} - \frac{1}{h_3}\right) & \ddots & 0 & 0 \\ 0 & \frac{1}{h_3} & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \ddots & \frac{1}{h_{M-2}} \\ 0 & 0 & 0 & 0 & \left(-\frac{1}{h_{M-2}} - \frac{1}{h_{M-1}}\right) \\ 0 & 0 & 0 & 0 & \frac{1}{h_{M-1}} \end{pmatrix},$
$\mathbf{R} = \begin{pmatrix} \frac{1}{3}(h_1 + h_2) & \frac{1}{6}h_2 & 0 & 0 & 0 \\ \frac{1}{6}h_2 & \frac{1}{3}(h_2 + h_3) & \frac{1}{6}h_3 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \frac{1}{6}h_{M-3} & \frac{1}{3}(h_{M-3} + h_{M-2}) & \frac{1}{6}h_{M-2} \\ 0 & 0 & 0 & \frac{1}{6}h_{M-2} & \frac{1}{3}(h_{M-2} + h_{M-1}) \end{pmatrix},$
$\mathbf{A} = (\mathbf{Q}, -\mathbf{R}^\top), \qquad \mathbf{W}_M = \mathrm{diag}(w_1, \dots, w_M), \qquad \mathbf{B} = \begin{pmatrix} \mathbf{W}_M & \mathbf{0} \\ \mathbf{0} & \lambda\mathbf{R} \end{pmatrix},$
  ▷ where $h_m = \hat{\kappa}_{m+1} - \hat{\kappa}_m$ and $w_m$ are strictly positive
    weights.

for $l = L, \dots, 1$ do
    Set: $\mathbf{x} = \begin{pmatrix} \mathbf{g}_{\hat{\tau}_l}^\top \\ \gamma_{\hat{\tau}_l}^\top \end{pmatrix}$, $\mathbf{y} = (w_1\tilde{c}_1(\hat{\kappa}_1, \hat{\tau}_l), \dots, w_M\tilde{c}_1(\hat{\kappa}_M, \hat{\tau}_l), 0, \dots, 0)^\top$.
    Solve  $\min_{\mathbf{x}} -\mathbf{y}^\top\mathbf{x} + \frac{1}{2}\mathbf{x}^\top\mathbf{B}\mathbf{x}$
           $s.t.\ \mathbf{A}^\top\mathbf{x} = 0, \quad (19) - (25).$
      ▷ the first condition ensures that the optimal
        solution is a natural cubic spline.  For the other
        conditions see sec. 3.2.3.
    Set: $\hat{c}_1(\hat{\boldsymbol{\kappa}}, \hat{\tau}_l) = \mathbf{g}^*_{\hat{\tau}_l}$
for $l = 2, \dots, L-1$ do
    Set: $\hat{\sigma}_1^L(\hat{\boldsymbol{\kappa}}, \hat{\tau}_l) = \sqrt{\dfrac{2\frac{\hat{c}_1(\hat{\boldsymbol{\kappa}}, \hat{\tau}_{l+1}) - \hat{c}_1(\hat{\boldsymbol{\kappa}}, \hat{\tau}_{l-1})}{\hat{\tau}_{l+1} - \hat{\tau}_{l-1}}}{\hat{\boldsymbol{\kappa}}^2 \frac{\partial^2 \hat{c}_1(\hat{\boldsymbol{\kappa}}, \hat{\tau}_l)}{\partial \kappa^2}}}$
```

3.3 AndreasenHuge

3.3.1 The general idea

Unlike the other methods examined in this paper, Andreasen & Huge (2011) does not take a regression model, and the corresponding assumption of the observed market prices being inflicted with noise, as a starting point. Instead the authors have derived a method which focuses on matching the observed data points, and which is arbitrage free, not only in the *limit*, as the step-size of the grid tends to zero, but for a grid of arbitrary step-size.

Thus, the method presented by Andreasen & Huge mainly differs from the others by interpolating the observed prices, rather than approximating them. This is done using a 1-step implicit finite difference approximation of Dupire's forward PDE, using a pre-fitted, piecewise linear local volatility function for each observed maturity level.

3.3.2 Details of the method

The method relies on the *forward pricing PDE*, derived by Dupire (1994), for calculation of the interpolated call surface. This is a boundary value problem which for the *normed* framework is given by:

$$\begin{aligned} -\frac{\partial c_1(\kappa,\tau)}{\partial\tau} + \frac{1}{2}\left(\sigma_1^L(\kappa,\tau)\right)^2 \kappa^2 \frac{\partial^2 c_1(\kappa,\tau)}{\partial\kappa^2} &= 0 \\ c_1(\kappa,0) &= (1-\kappa)^+ \end{aligned} \tag{27}$$

where the still unknown local volatility function, $\sigma_1^L(\kappa,\tau)$, is replaced by proxy functions: $\vartheta_j(\kappa)$, for each maturity level in the observed data set:

$$\left\{\kappa_{ij},\tau_j\right\}_{ij=1,\ldots,I_j,\ j=1,\ldots,J.}$$

Note that the notation used for the observed data points in this method, differs from the notation used elsewhere in this paper.

The proxy volatility functions are formed as piecewise constant functions for a grid of user-defined equidistant forward-moneyness levels $\hat{\kappa}$:

$$\vartheta_j(\hat{\kappa}) = \begin{cases} a_1 & \hat{\kappa} \le b_1 \\ a_{ij} & b_{ij-1} < \hat{\kappa} \le b_{ij} \quad ij = 2,\ldots I_j - 1 \\ a_{I_j} & b_{I_j-1} < \hat{\kappa} \end{cases}$$

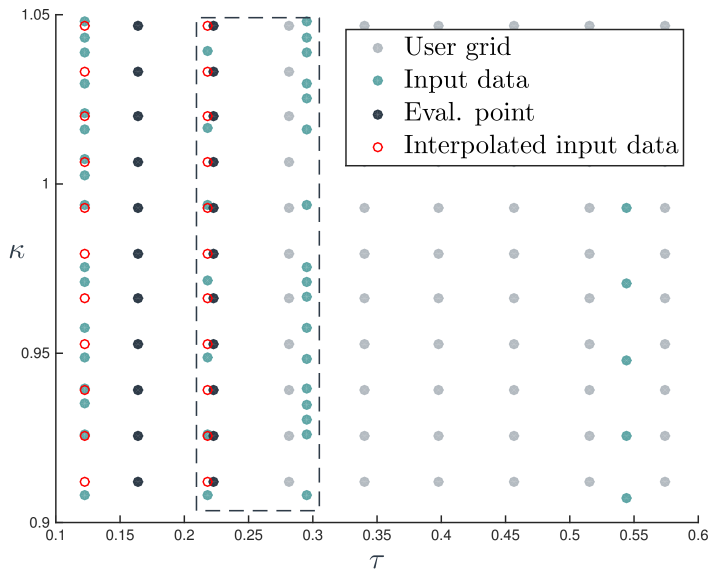


FIGURE 4: A subset of the call surface between maturity τ_{j-1} and τ_j is fitted by implicit PDE interpolation and the calibrated volatility function $\vartheta_j(\kappa)$.

where the b_{ij}'s are given as the mid-points between the observed strike levels for the given maturity:

$$b_{ij} = \frac{\kappa_{ij+1} - \kappa_{ij}}{2} \qquad ij = 1, \ldots, I_j - 1.$$

The volatility proxy's, $\vartheta_j(\cdot)$, are calibrated to the observed call prices for one observed maturity at a time - starting with the smallest, τ_0, working its way up to τ_J - as the solution to the least squares minimization problem:

$$\min_{\vartheta_j(\hat{\boldsymbol{\kappa}}))} \sum_{ij=1}^{I_j} \left(\frac{c_1(\kappa_{ij}, \tau_j) - \hat{c}_1(\kappa_{ij}, \tau_j; \vartheta_j(\hat{\boldsymbol{\kappa}}))}{w_{ij}} \right)^2, w_{ij} = \frac{\partial c_1(\kappa_{ij}, \tau_j)}{\partial \sigma_1^{imp}} \tag{28}$$

where the interpolated call values, $\hat{c}_1(\kappa_{ij}, \tau_j; \vartheta_j(\hat{\boldsymbol{\kappa}}))$, are given as the solution to the implicit finite difference approximation of the forward PDE given in eqn. (27):

$$\begin{aligned} \left[1 - \frac{1}{2}\Delta_{\tau_j}\vartheta_j(\hat{\boldsymbol{\kappa}})^2(\hat{\boldsymbol{\kappa}})^2\delta_{\kappa\kappa}\right]\hat{c}_1(\hat{\boldsymbol{\kappa}}, \tau_j) &= \hat{c}_1(\hat{\boldsymbol{\kappa}}, \tau_{j-1}) \\ \hat{c}_1(\hat{\boldsymbol{\kappa}}, 0) &= (1 - \hat{\boldsymbol{\kappa}})^+ \end{aligned} \tag{29}$$

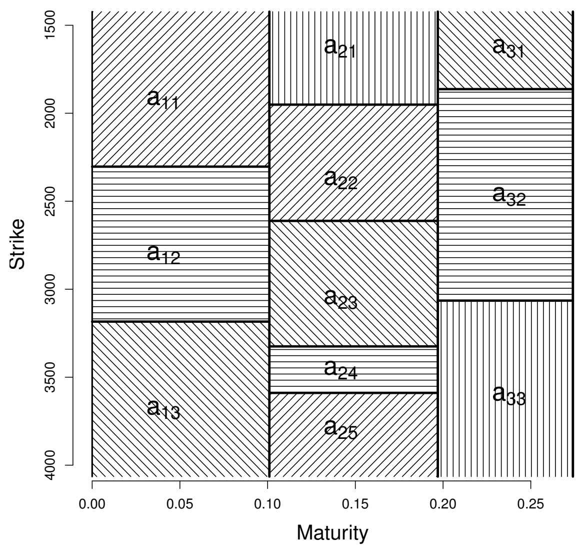


Figure 5: A set of piecewise constant volatility proxy's, one for each of the three sub-periods defined by the observed maturity levels.

which is evaluated in κ_{ij} using linear interpolation. Here, $\Delta_{\tau_j} = \tau_{j+1} - \tau_j$ and $\delta_{\kappa\kappa} f(\kappa)$ denotes the central finite difference approximation of the second order derivative:

$$\delta_{\kappa\kappa} f(\kappa) = \frac{f(\kappa + \Delta_\kappa) - 2f(\kappa) + f(\kappa + \Delta_\kappa)}{\Delta_\kappa^2}.$$

The final step of the method consists of calculating the call prices for maturities in between the observed levels, τ_j, by solving an implicit finite-difference scheme similar to the one in eqn. (29):

$$\left[1 - \frac{1}{2}\left(\hat{\tau} - \tau_j\right) \vartheta_j(\hat{\kappa})^2 (\hat{\kappa})^2 \delta_{\kappa\kappa}\right] \hat{c}_1(\hat{\kappa}, \hat{\tau}) = \hat{c}_1(\hat{\kappa}, \tau_j), \quad \hat{\tau} \in]\tau_{j-1}, \tau_j[\tag{30}$$

The complete interpolation procedure is given in algorithm 4 and 5, where the implicit finite difference schemes, (29) and (30), are expressed using a tri-diagonal matrix with absorbing boundary conditions. This representation of the problem reduces the computational cost significantly.

3.3.3 No-arbitrage conditions

This method is quite unique in terms of the no-arbitrage conditions. The optimization problems for all other methods assessed in this paper are subject to a number of no-arbitrage constraints. But the optimization problem in Andreasen & Huge's method, (28), are not subject to any constraints, as the no-arbitrage conditions are *built* into the calibration procedure itself.

Andreasen & Huge prove that the surface of interpolated call prices, produced by the respective implicit finite difference steps, are arbitrage free by showing that they are convex in forward-moneyness and non-decreasing in maturity. This arbitrage-free'ness stems from the structure of the tri-diagonal matrix with absorbing boundaries representing the implicit finite difference approximation.

An exhaustive documentation of this proof can be found in Rasmussen (2012) from which it is clear, that these conditions are fullfilled for a grid of arbitrary step-size. In terms of arbitrage, this would in theory allow for a stable result for a much coarser grid, which reduces the computational cost of the calibration.

Comparing the arbitrage conditions considered in this paper with the conditions in proposition 1, it becomes evident that the authors are omitting to explicitly account for the boundary conditions given in *(C2)*.

3.3.4 Conversion to the Local Volatility Surface

Dupire's local volatility function, $\hat{\sigma}_1^L(\boldsymbol{\kappa}, \boldsymbol{\tau})$, given in eqn. (6) can for this function only be determined by continuing the use of finite difference approximations.

The first order derivative wrt. maturity is replaced by the central approximation given in eqn. (26) as in the local volatility function associated with Fengler's method, described in section 3.2.4. The second order derivative wrt. forward-moneyness is likewise replaced with a central finite difference approximation:

$$\frac{\partial^2 c_1(\kappa, \tau)}{\partial \kappa^2} \approx \frac{c_1(\kappa_{m+1}, \tau) - 2c_1(\kappa_m, \tau) + c_1(\kappa_{m-1}, \tau)}{(\kappa_m - \kappa_{m-1})(\kappa_{m+1} - \kappa_m)} \tag{31}$$

Hence, the final estimate of the local volatility function is given by:

$$\hat{\sigma}_1^L(\kappa_m, \tau_l) = \sqrt{\frac{2\frac{\hat{c}_1(\kappa_m,\tau_{l+1})-\hat{c}_1(\kappa_m,\tau_{l-1})}{\tau_{l+1}-\tau_{l-1}}}{\kappa_m^2\frac{\hat{c}_1(\kappa_{m+1},\tau_l)-2\hat{c}_1(\kappa_m,\tau_l)+\hat{c}_1(\kappa_{m-1},\tau_l)}{(\kappa_m-\kappa_{m-1})(\kappa_{m+1}-\kappa_m)}}}.$$

3.3.5 Tweaking the method

Evaluate surface in specified point $(\bar{\kappa}, \bar{\tau})$: The method approximates values for a user-specified grid in forward-moneyness and the observed maturity levels:

$$\hat{\boldsymbol{\kappa}} \times \boldsymbol{\tau} \supseteq \{\kappa_i \times \tau_i\}_{i=1,\dots,N}$$

Algorithm 4: Part I - Andreasen & Huge (2011) - interpolation of the call surface.

```
input: $\tau_j$, $_{j=1,\dots,J}$                          ▹ observed time to maturities,
       $\kappa_{ij}$, $_{ij=1,\dots,I_j}$                     ▹ observed forward-moneyness,
       $c_1(\kappa_{ij}, \tau_j)$                  ▹ observed European forward call prices,
       $\sigma_1^{imp}(\kappa_{ij}, \tau_j)$                        ▹ implied volatilities,
       $\nu(\kappa_{ij}, \tau_j)$:                                         ▹ vegas.

Def: $[\hat{\kappa}_1; \hat{\kappa}_M] \times [\hat{\tau}_1; \hat{\tau}_L] = \hat{\boldsymbol{\kappa}} \times \hat{\boldsymbol{\tau}} \supseteq \{\kappa_i \times \tau_i\}_{i=1,\dots,N}$
  ▹ Regular grid for which the approximation of the call surface is
    derived.  Uniform in forward-moneyness with an additional margin
    compared to the interval defined by the observed levels.
    Maturity levels are allowed to be non-uniform, but must as a
    minimum include all observed levels:  $\{\tau_j\}_{j=0}^J$.

Def: $\vartheta_j(\hat{\boldsymbol{\kappa}}) := \begin{cases} a_1 & \hat{\boldsymbol{\kappa}} \leq b_1 \\ a_{ij} & b_{ij-1} < \hat{\boldsymbol{\kappa}} \leq b_{ij},\ {}_{ij=2,\dots I_j-1} \\ a_{I_j} & b_{I_j-1} < \hat{\boldsymbol{\kappa}} \end{cases}$
  ▹ Piecewise constant volatility Proxy defined for each $\tau_j$ where
    $b_{ij} = \kappa_{ij} + \frac{\kappa_{ij+1} - \kappa_{ij}}{2}$, $_{ij=1,\dots,I_j-1}$ are mid-points between the observed strike
    levels for the given maturity.

Set: $\mathbf{A}_{j,l} = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ -z_2^j & 1+2z_2^j & -z_2^j & 0 & \cdots & \vdots \\ 0 & -z_3^j & 1+2z_3^j & -z_3^j & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & -z_{M-1}^j & 1+2z_{M-1}^j & -z_{M-1}^j \\ 0 & \cdots & \cdots & 0 & 0 & 1 \end{pmatrix}$
  ▹ with coefficients are given by:  $z_m^j = \frac{1}{2} \frac{\Delta_{\tau_{j,l}}}{\Delta_{\hat{\kappa}}^2} \vartheta_j(\hat{\kappa}_m)^2 \hat{\kappa}_m^2$,  $\Delta_{\hat{\kappa}} = \hat{\kappa}_2 - \hat{\kappa}_1$,
    $\Delta_{\tau_{j,l}} = \tau_l - \tau_j$.
```

After calibrating the local volatility proxys the values for a given maturity level, $(\hat{\boldsymbol{\kappa}}, \bar{\tau})$, are determined by solving the 1-step implicit finite difference step given in eqn.(30). The forward-moneyness grid is equidistant, hence one is forced to interpolate in this dimension to obtain the value for a user-specified grid-point, $(\bar{\kappa}, \bar{\tau})$.

▹ This interpolation difficulty for the forward-moneyness grid is eliminated here, as the user-defined grid is equidistant and can therefore be implemented as a subpart of the grid used for the calibration procedure.

Algorithm 5: Part II - Andreasen & Huge (2011) - interpolation of the call surface.

```
Set: $\hat{c}_1(\hat{\boldsymbol{\kappa}}, 0; \vartheta_j) = \max(1 - \hat{\boldsymbol{\kappa}}, 0)$
  ▷ Initialize rectangular grid of approximated call prices.

for $j = 1$ to $J$ do
    $\min_{\vartheta_j(\boldsymbol{\kappa}))} \sum_{ij=1}^{I_j} \left( \frac{c_1(\kappa_{ij},\tau_j) - \hat{c}_1(\kappa_{ij},\tau_j;\vartheta_j(\hat{\boldsymbol{\kappa}}))}{w_{ij}} \right)^2, \; w_{ij} = \frac{\partial c_1(\kappa_{ij},\tau_j)}{\partial \sigma imp}$
      ▷ Calibrate volatility proxys $\vartheta_j(\cdot)$ using a nonlinear least
        squares optimization procedure with initial guess: $\sigma_1^{imp}(\cdot, j)$.
        The call price estimate, $\hat{c}_1(\boldsymbol{\kappa}, \tau_j)$, is given as the solution to
        a 1-step implicit finite difference solver:
        $\mathbf{A}_{j-1,j} \cdot \hat{c}_1(\hat{\boldsymbol{\kappa}}, \tau_j) = \hat{c}_1(\hat{\boldsymbol{\kappa}}, \tau_{j-1})$, with initial condition
        $\hat{c}_1(\hat{\boldsymbol{\kappa}}, 0) = (1 - \hat{\boldsymbol{\kappa}})^+$.

    for $\hat{\tau}_l \in (\tau_{j-1}, \tau_j]$ do
        Set: $\hat{c}_1(\hat{\boldsymbol{\kappa}}, \hat{\tau}_l; \vartheta_j) :=$
                implicit_FD $\left(\mathbf{A}_{j-1,l} \cdot \hat{c}_1(\hat{\boldsymbol{\kappa}}, \tau_l) = \hat{c}_1(\hat{\boldsymbol{\kappa}}, \tau_{j-1})\right)$
          ▷ Fill the rectangular sub-grid defined by $\hat{\boldsymbol{\kappa}}$ and all
            $\hat{\tau}_l \in (\tau_{j-1}, \tau_j]$ with values interpolated using a 1-step
            implicit finite difference method.

for $m = 2, \ldots, M-1$, $l = 2, \ldots, L-1$ do
    Set: $\hat{\sigma}_1^L(\kappa_m, \tau_l) = \sqrt{\frac{2 \frac{\hat{c}_1(\kappa_m,\tau_{l+1}) - \hat{c}_1(\kappa_m,\tau_{l-1})}{\tau_{l+1} - \tau_{l-1}}}{\kappa_m^2 \frac{\hat{c}_1(\kappa_{m+1},\tau_l) - 2\hat{c}_1(\kappa_m,\tau_l) + \hat{c}_1(\kappa_{m-1},\tau_l)}{(\kappa_m - \kappa_{m-1})(\kappa_{m+1} - \kappa_m)}}}$.
```

Here we let the calibration grid in forward-moneyness be given by the user-defined grid with additional points uniformly placed in each grid-gap, and with extra *padding* at the ends, see discussion of the padding under the next bullet.

The number of additional data points needed for each gap depends on the construction of the user-grid and the input data available. Here, the method has been run with: $\{0, 1, 2, 3, 5\}$ additional points in each gap. We found that the best fit in our implementation was obtained using 2 additional points between each pair of user-defined. Looking at the the local volatility surface, it can be seen that it fluctuates a lot and sudden high values can be observed. This behavior is probably due to the finite difference evaluation which do not seem stable for the call surfaces produced by this method. Unfortunately, these extreme fluctuations do not seem to improve with the number of additional points added.

▷ The *padding* added to the calibration grid in the forward-moneyness dimension, is necessary due to the boundary conditions involved when solving a finite difference method. If the padding is too sparse, this will result in a corrupted surface near the edges. If there on the contrary is too much padding, this will unnecessarily increase the computational time. Here, we have tried to add 25%, 50% and 75% of the distance from the smallest forward-moneyness level observed to the highest. Looking at the local volatility surface, it is evident that 25% is not enough as erroneous variations can be seen for small values of the forward-moneyness. A padding of additional 50% seems sufficient, this could perhaps even have been reduced a bit, but in this framework a reasonable fit for all dates in the dataset is more important.

- ▷ Finally, it can be mentioned that the algorithm used for the optimization problem, given in eqn. (28), is implemented using the *Levenberg-Marquardt algorithm* as specified in (Rasmussen (2012)).

3.4 GLASERHEIDER

3.4.1 THE GENERAL IDEA

Glaser & Heider (2012) propose a *local* calibration method similar to that by Benko *et al.* (2007), but for estimation of the call surface instead of the implied volatility surface. The regression model for the call quotes observed in the market are given by:

$$c_1(\kappa_i, \tau_i) = \hat{c}_1(\kappa_i, \tau_i) + \varepsilon_i \qquad i = 1, \ldots, N \tag{32}$$

where the noise term, ε_i, represents the *input data risk.*

This is estimated using the *moving least-squares* method for one grid-point in the surface, (κ, τ), at a time. The method is *local* in the sense that a weight function evaluates which input data points to include in the approximation for a given point in the surface. This method is in the statistical literature known as local (polynomial) regression, i.e. the same method used by Benko *et al.* in sec. 3.1. Working directly with the call surface implies linear no-arbitrage constraints which, along with the structure of the problem, allows Glaser & Heider to rephrase the optimization problem to linear least squares form.

3.4.2 DETAILS OF THE METHOD

The moving least squares method is, as mentioned above, equivalent to the local polynomial regression method used by Benko *et al.* (2007). Thus, in order to keep things consistent, the framework for this method is described using the notation from sec. 3.1.

The call surface is for a given point (κ_i, τ_i) near (κ, τ) approximated using a two-dimensional local polynomial quadratic in κ and linear in τ:

$$\begin{aligned}
\hat{c}_1(\kappa_i, \tau_i) &\overset{taylor}{\approx} \hat{c}_1(\kappa, \tau) + \frac{\partial \hat{c}_1(\kappa, \tau)}{\partial \kappa}(\kappa_i - \kappa) + \frac{\partial \hat{c}_1(\kappa, \tau)}{\partial \tau}(\tau_i - \tau) \\
&\quad + \frac{1}{2!}\frac{\partial^2 \hat{c}_1(\kappa, \tau)}{\partial \kappa^2}(\kappa_i - \kappa)^2 \\
&= \alpha^0_{\kappa,\tau} + \alpha^1_{\kappa,\tau}(\kappa_i - \kappa) + \alpha^2_{\kappa,\tau}(\tau_i - \tau) + \frac{\alpha^3_{\kappa,\tau}}{2}(\kappa_i - \kappa)^2 .
\end{aligned} \tag{33}$$

Let in the following this polynomial be denoted $p_{\kappa,\tau}(\kappa_i, \tau_i)$.

The polynomial representing the call surface is fitted for one grid-point, (κ, τ), by minimizing the weighted least-squares error:

$$\min_{\alpha} \frac{1}{2} \sum_{i} \left(p_{\kappa,\tau}(\kappa_i, \tau_i) - c_1(\kappa_i, \tau_i)\right)^2 \Phi_1(\kappa_i - \kappa, \tau_i - \tau) \tag{34}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^4$ is the vector of polynomial coefficients.

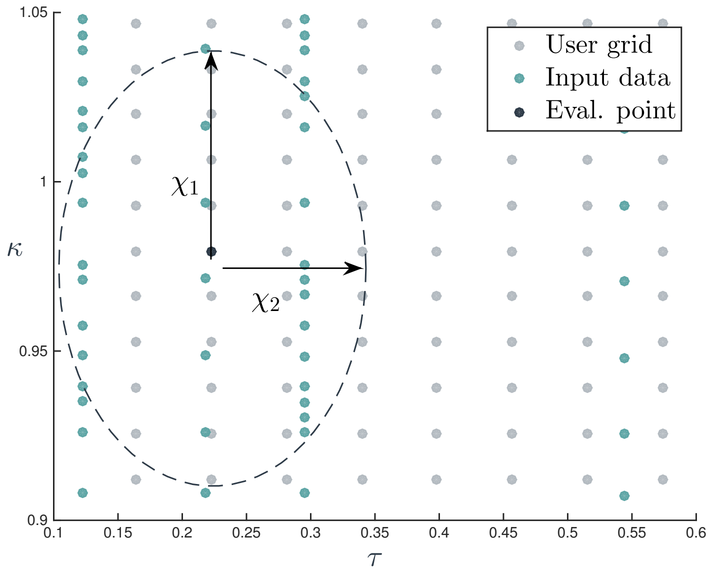


Figure 6: Each grid-point is fitted individually based on a *local* subset of the observed data points.

The kernel function consists in this setup of two parts. The first part is the index set, $I(\cdot)$, determining which observations are influencing the estimation near (κ, τ) in eqn. (34). Here, data points lying within an ellipse with radius': χ_1, χ_2, are included in the approximation:

$$I(\kappa, \tau) = \left\{ i \in \{1, \ldots, N\} \middle| \left(\frac{(\kappa_i - \kappa)}{\chi_1(\kappa, \tau)_2}\right)^2 + \left(\frac{(\tau_i - \tau)}{\chi_2(\kappa, \tau)}\right)^2 < 1 \right\}. \tag{35}$$

In figure 7 is given an illustration of this ellipse of points. The second part is the weight function: Φ_1, defined for all points in the index set, $I(\cdot)$, given by *exponential weighing*:

$$\Phi_1(\kappa_l - \kappa, \tau_l - \tau) = \exp\left(-\left[\left(\frac{(\kappa_l - \kappa)}{\chi_1(\kappa, \tau)}\right)^2 + \left(\frac{(\tau_l - \tau)}{\chi_2(\kappa, \tau)}\right)^2\right]\right) \quad \text{for } l \in I(\kappa, \tau).$$

The distance in strike-direction, χ_1, and distance in the maturity-direction, χ_2, should according to Glaser & Heider (2012) be determined for each grid point, (κ, τ), such that the index set in eqn. (35) contains data points for 3 maturity levels, each with 10 strike levels. This can be a challenge in practice if the data is unevenly distributed, cf. discussion in sec. 3.4.5.

3.4.3 No-arbitrage conditions

The no-arbitrage conditions imposed on the call surface are given below in an adjusted version fitting the normed framework used in this paper.

$$\begin{array}{ll} \mathrm{I}: \frac{\partial c_1(\kappa,\tau)}{\partial \kappa} \leq 0; & \mathrm{II}: \frac{\partial^2 c_1(\kappa,\tau)}{\partial \kappa^2} \geq 0; \\ \mathrm{III}: \frac{\partial c_1(\kappa,\tau)}{\partial \tau} \geq 0; & \mathrm{IV}: c_1(\kappa,\tau) \geq 0; \end{array}$$

Comparing these to the sufficient no-arbitrage conditions for the normed call surface in prop. 1, it becomes evident that condition (III) is too weak while (I) is redundant.
Condition I-IV are added to the optimization problem in eqn. (34) as a set of linear constraints[6]:

$$B\boldsymbol{\alpha}_{\kappa,\tau} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \boldsymbol{\alpha}_{\kappa,\tau} \leq 0. \tag{36}$$

$$\tag{37}$$

where $\boldsymbol{\alpha}_{\kappa,\tau}$ is the vector of coefficients given in eqn. (33).

The actual optimization problem given in eqn. (34), can also be rewritten to matrix-form and can thus be rephrased as a linear least squares problem. The final method is summarized in algorithms 6 and 7.

3.4.4 Conversion to the Local Volatility Surface

Dupire's local volatility function given in eqn. (6), $\hat{\sigma}_1^L(\kappa, \tau)$, can for this method be obtained directly from the estimated coefficients, $\boldsymbol{\alpha}_{\kappa,\tau}$, using Dupire's formula:

$$\hat{\sigma}_1^L(\kappa,\tau) = \sqrt{\frac{2\frac{\partial \hat{c}_1(\kappa,\tau)}{\partial \tau}}{\kappa^2 \frac{\partial^2 \hat{c}_1(\kappa,\tau)}{\partial \kappa^2}}} = \sqrt{\frac{2\alpha^2_{\kappa,\tau}}{\kappa^2 \alpha^3_{\kappa,\tau}}}.$$

[6]Remark, the rows have been rearranged compared to (Glaser & Heider, 2012, eqn. 15).

Algorithm 6: Part I - Glaser & Heider (2012) - approximation of the call surface.

```
input: $\tau_i$, $i=1,\dots,N$                        ▹ observed maturity levels.
       $\kappa_i$, , $i=1,\dots,N$                    ▹ observed forward-moneyness levels.
       $c_1(\kappa_i, \tau_i)$                        ▹ observed European forward call prices

Def: $\hat{\kappa} \times \hat{\tau} \subseteq \{\kappa_i \times \tau_i\}_{i=1,\dots,N}$
  ▹ Grid-points for which the local approximation of the call surface
    is derived.  Contained in the rectangle defined by the input
    grid.  Allowed to be non-uniform.

Def: $p_{\hat{\kappa},\hat{\tau}}(\kappa_i, \tau_i) = \alpha^0_{\hat{\kappa},\hat{\tau}} + \alpha^1_{\hat{\kappa},\hat{\tau}}(\kappa_i - \hat{\kappa}) + \alpha^2_{\hat{\kappa},\hat{\tau}}(\tau_i - \hat{\tau}) + \frac{\alpha^3_{\hat{\kappa},\hat{\tau}}}{2}(\kappa_i - \hat{\kappa})^2$
  ▹ Local polynomial quadratic in $\hat{\kappa}$, linear in $\hat{\tau}$ where:
    $\hat{c}_1(\hat{\kappa}, \hat{\tau}) = \alpha^0_{\hat{\kappa},\hat{\tau}}$; $\frac{\partial \hat{c}_1(\hat{\kappa},\hat{\tau}}{\partial \kappa} = \alpha^1_{\hat{\kappa},\hat{\tau}}$; $\frac{\partial \hat{c}_1(\hat{\kappa},\hat{\tau}}{\partial \tau} = \alpha^2_{\hat{\kappa},\hat{\tau}}$; $\frac{\partial^2 \hat{c}_1(\hat{\kappa},\hat{\tau}}{\partial \kappa^2} = \alpha^3_{\hat{\kappa},\hat{\tau}}$;

Set $B = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}$.
  ▹ Matrix-form of the linear no-arbitrage constraints on the
    coefficients vector $\alpha_{\hat{\kappa},\hat{\tau}}$.
```

3.4.5 Tweaking the method

Evaluate surface in specified point $(\bar{\kappa}, \bar{\tau})$: The method approximates values for one user-specified grid-point:

$$(\hat{\kappa}, \hat{\tau}) \subseteq \{\kappa_i \times \tau_i\}_{i=1,\dots,N}$$

at a time. This grid-point must be contained within the rectangle defined by the original grid.

Thus, to obtain a value for a given point: $(\bar{\kappa}, \bar{\tau})$, the calibration procedure must be applied for this point.

▹ Applying this method for a user-specified grid is straight forward. As can be seen above the method simply has to be repeated for each grid point.

▹ The optimization procedure on the other hand, does cause a bit of trouble. Matlab's function for constrained least squares problems, `lsqlin`, does not obtain sufficient accuracy for this problem. The problem given in al-

Algorithm 7: Part II - Glaser & Heider (2012) - approximation of the call surface.

```
for $\hat{\kappa} \in \hat{\boldsymbol{\kappa}}$, $\hat{\tau} \in \hat{\boldsymbol{\tau}}$ do
    Set $\chi_1(\hat{\kappa}, \hat{\tau})$, $\chi_2(\hat{\kappa}, \hat{\tau})$
      ▹ Parameters determining the 'locality' of the approximation.

    for $i = 1, \dots, N$ do
        $I(\hat{\kappa}, \hat{\tau}) = I(\hat{\kappa}, \hat{\tau}) \cup \left\{ i \,\middle|\, \frac{(\kappa_i - \hat{\kappa})^2}{\chi_1^2} + \frac{(\tau_i - \hat{\tau})^2}{\chi_2^2} < 1 \right\}$
      ▹ Determine for each data point whether it lies within the
        ellipse surrounding $(\hat{\kappa}, \hat{\tau})$ and thus belongs to the index set.

    Set $A = \begin{pmatrix} 1 & \Delta^1_{\hat{\kappa}} & \Delta^1_{\hat{\tau}} & \frac{1}{2}\left(\Delta^1_{\hat{\kappa}}\right)^2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \Delta^{|I(\hat{\kappa},\hat{\tau})|}_{\hat{\kappa}} & \Delta^{|I(\hat{\kappa},\hat{\tau})|}_{\hat{\tau}} & \frac{1}{2}\left(\Delta^{|I(\hat{\kappa},\hat{\tau})|}_{\hat{\kappa}}\right)^2 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} c_1(\kappa_1, \tau_1) \\ \vdots \\ c_1\left(\kappa_{|I(\hat{\kappa},\hat{\tau})|}, \tau_{|I(\hat{\kappa},\hat{\tau})|}\right) \end{pmatrix}$,
      ▹ where $\Delta^l_{\hat{\kappa}} = \kappa_l - \hat{\kappa}$, $\Delta^l_{\hat{\tau}} = \tau_l - \hat{\tau}$ for $l \in I(\hat{\kappa}, \hat{\tau})$.

    $\Phi = \operatorname{diag}\left( \exp\left( -\left[ \left( \frac{(\kappa_l - \hat{\kappa})}{\chi_1(\hat{\kappa},\hat{\tau})} \right)^2 + \left( \frac{(\tau_l - \hat{\tau})}{\chi_2(\hat{\kappa},\hat{\tau})} \right)^2 \right] \right) \middle| \, l \in I(\hat{\kappa}, \hat{\tau}) \right)$.

    Solve:  $\hat{\boldsymbol{\alpha}}_{\hat{\kappa},\hat{\tau}} = \underset{\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}}{\arg\min} \, \frac{1}{2} \left\| \Phi^{1/2} \left( A \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \mathbf{b} \right) \right\|_2^2$
            s.t. $B \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} \leq 0$.

    Set $\hat{c}_1(\hat{\kappa}, \hat{\tau}) = \hat{\alpha}^0_{\hat{\kappa},\hat{\tau}}$;     $\hat{\sigma}^L_1(\hat{\kappa}, \hat{\tau}) = \sqrt{\frac{2\hat{\alpha}^2_{\hat{\kappa},\hat{\tau}}}{\hat{\kappa}^2 \hat{\alpha}^3_{\hat{\kappa},\hat{\tau}}}}$;
      ▹ Estimated value of the call- and local volatility surface in
        point $(\hat{\kappa}, \hat{\tau})$.
```

gorithm 6 is therefore translated to a quadratic programming problem:

$$\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2}\, \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}^{\top}\, (A^{\top}\Phi A)\, \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - (A^{\top}\Phi \mathbf{b})^{\top}\, \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}},$$

similar to that used by Fengler & Hin (2013), which can be be solved using Matlab function `quadprog`. Details are given in appendix A.1.

$\triangleright$ The major challenge for this method is to find an *automatic* procedure for choosing parameters: χ_1, χ_2, such that each *ellipse* roughly contains the same amount of observed data points.

Here, an iterative-style algorithm has been used, which sets χ_2 according to the requested number of maturity levels and afterwards searches for a value of χ_1, such that the ellipse contains a number of data points within given limits, to the extend possible.

The maturity levels is chosen as the observed maturity levels closest to the user-specified grid-points. One could have specified that the levels should lie on each side of the user-defined level. However, this does not improve the fit, rather it can for some grid-points worsen it. We have observed that the surface can be severely disrupted in certain areas if this strategy is conducted, as the algorithm is then forced to include information from data-points far away from the point of evaluation.

χ_2 As mentioned above, we start by setting χ_2 such that the required number of maturity levels closest to the evaluation point is contained in the ellipse.

Now, we examine the surfaces generated using ellipses with a target of $30 - 32$ grid-points distributed across $\{2, 3, 4\}$ maturity levels, respectively. Glaser & Heider (2012, remark 1) states that there should at least be 2 maturity levels in the ellipse in order for the optimization problem to have a unique solution, and more than 4 levels can for some datasets approximate the total number of observed maturity levels.

These surfaces suggest that the fewer maturity levels, the smoother is the local volatility surface. This result is quite intuitive as more levels is tantamount to additional information needed to be captured by the local polynomial which is not that flexible in the maturity direction. Thus subsets of the input data containing more than 3 maturity levels are disregarded.

This examination unveils further that by using only 2 maturity levels the observed data can be distributed in such a way, that the band of points has to be quite wide. For the example here the limits had to be set to: $25 - 45$, while the target number of points could easily be fit into a band of: $25 - 35$, when using 3 maturity levels.

Hence, total number of data points in the ellipse is here spread across 3 maturity levels, as suggested by Glaser & Heider (2012).

χ_1 This parameter determines the width of the ellipse in the forward-moneyness dimension, and is set by the iterative algorithm such that the total number of points is contained within this band.

Our examinations show, contrary to the number of maturity levels, that an increase in the number of forward-moneyness levels increases the smoothness of the surface. The only problem is now to determine when the surface is smooth enough without becoming too smooth.

We have examined surfaces for 3 maturity levels and a target of: $\{10, 15, 25, 25\}$, forward-moneyness levels for each, with the total number of points within band: $3 \cdot \#\text{levels} \pm 5$. As one might expect, there is a clear trade-off between smoothness and the fit to the input values.

Here, we choose a target of 15 levels for each maturity, as this generates surfaces with a reasonable smoothness while keeping a good fit to the input values.

Taking a closer look at the implied volatility surfaces generated by this methods, one can see that the surfaces are close to zero in the corner for large forward-moneyness levels and small maturity levels. Generally there are no observations for this area, thus the method will have to extrapolate here. It is not possible to tweak the parameters in such a way that this area will get a better fit. Hence, this seem to be one of the biggest weaknesses of the method.

3.5 FENGLERHIN

3.5.1 THE GENERAL IDEA

Fengler & Hin (2013) assume, like Fengler (2009) and Glaser & Heider (2012), that the call prices observed in the market follows the regression model:

$$c_1(\kappa_i, \tau_i) = \hat{c}_1(\kappa_i, \tau_i) + \varepsilon_i \qquad i = 1, \ldots, N \tag{38}$$

where $c_1(\kappa_i, \tau_i)$ is the normed, observed call prices and ε_i is the error term representing the market noise.

The method presented here extends the work from Fengler's previous article, Fengler (2009), by not only fitting the unknown surface of call prices, $\hat{c}_1(\kappa, \tau)$, with a 1-dimensional spline for each maturity level, but instead fitting a 2-dimensional spline to the entire surface. This 2-dimensional spline is constructed as the tensor product of two univariate polynomial splines, both represented as a linear combination of B-splines, defined by a set of coefficients and a sequence of knots.

This particular type of parameterization allows for a fast evaluation of the surface at a given point using the same order of operations as for the univariate spline. Thus, using tensor-product B-splines to represent the surface makes it practically possible to perform a global approximation with the no-arbitrage constraints translated to constraints on the coefficients.

3.5.2 DETAILS OF THE METHOD

Let the two-dimensional polynomial spline approximating the call surface be of degree: p_1, p_2, given for a regular grid of knots: $\boldsymbol{\xi} \times \boldsymbol{\nu}$. The knot sequence for the *forward-moneyness*-direction is given by:

$$\kappa_1 = \xi_0 = \cdots = \xi_{p_1} < \xi_{p_1+1} < \cdots < \xi_{q_1} = \xi_{q_1+1} = \cdots = \xi_{q_1+p_1+1} = \kappa_N, \tag{39}$$

where the boundary knots have multiplicity $p_1 + 1$. The sequence for the *maturity*-direction are similarly given by:

$$\tau_1 = \nu_0 = \cdots = \nu_{p_2} < \nu_{p_2+1} < \cdots < \nu_{q_2} = \nu_{q_2+1} = \cdots = \nu_{q_2+p_2+1} = \tau_N. \tag{40}$$

where the boundary knots have multiplicity $p_2 + 1$.

As mentioned in the introduction, the spline surface is here constructed as the tensor product of two polynomial splines given by their B-spline representation. Hence, the approximation for the call surface is given as a linear combination of B-splines:

$$s(\kappa,\tau) = \sum_{j_1=0}^{q_1}\sum_{j_2=0}^{q_2}\theta_{j_1,j_2}B_{j_1,p_1}(\kappa)B_{j_2,p_2}(\tau) \tag{41}$$

where $\theta_{j_1,j_2} \in \mathbb{R}$ is the coefficients.

The B-splines are defined as in the univariate case, thus for a general non-decreasing knot sequence: $\mathbf{t}$, they are determined using the recursive formula:

$$\begin{aligned} B_{j,p}(x) &= \frac{x-t_j}{t_{j+p}-t_j}B_{j,p-1}(x) + \frac{t_{j+1+p}-x}{t_{j+1+p}-t_{j+1}}B_{j+1,p-1}(x), \\ B_{j,0}(x) &= \begin{cases} 1 & t_j \le x < t_{j+1}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{42}$$

where the convention: $\frac{0}{0} = 0$ is applied in case of multiple knots: $t_j = t_{j+r}$.

The approximation for the call surface is fitted to the observed call prices using a penalized least squares estimator:

$$\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=1}^{N}(c_1(\kappa_i,\tau_i) - s(\kappa_i,\tau_i))^2 + \lambda_N|\boldsymbol{\theta}|^2 \tag{43}$$

where $|\cdot|$ is the Euclidian vector norm and $\boldsymbol{\theta}$ is the $(q_1+1)(q_2+1)\times 1$ vector of B-spline coefficients:

$$\begin{aligned} \boldsymbol{\theta}^\top &= \left\{\theta_{j_1,j_2}\right\}_{j_1,j_2=0}^{q_1,q_2} \\ &= \left(\theta_{0,0},\theta_{0,1},\ldots,\theta_{0,q_2},\theta_{1,0},\theta_{1,1},\ldots,\theta_{1,q_2},\theta_{2,0},\ldots,\theta_{q1,q2}\right). \end{aligned}$$

Before the optimization is executed, the knot sequences in eqn.'s (39) and (40) must be set. Fengler & Hin notes that the distribution of the knots in the maturity-direction seems to be of less significance, and is therefore statically defined.

The distribution of the knots across the forward-moneyness range, $[\kappa_1;\kappa_N]$, has on the other hand significant influence on the quality of the final estimate, see Fengler & Hin (2013) for further details on this topic. Hence, an initial search for an optimal *knot placement* is necessary in the forward-moneyness direction. This search consists of two steps:

(1) Adding additional knot to the initial sequence $\kappa_1 = \xi_0 = \xi_1 < \xi_2 = \xi_3 = \kappa_N$ that minimizes the unconstrained *Akaike Information Criterion* for a linear $(p_1 = p_2 = 1)$ spline surface.

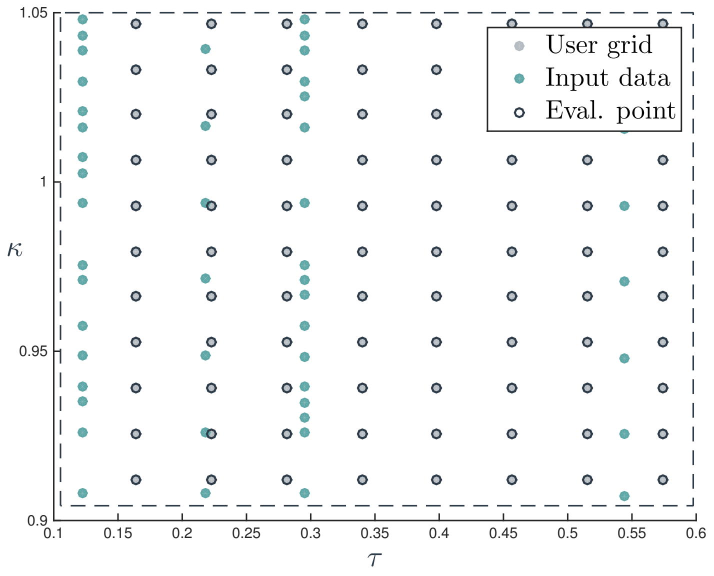


Figure 7: All grid-points are fitted by a tensor-product B-spline.

(2) Relocating or deleting knots from the sequence obtained in step (1) according to the constrained version of the selection criterion, now using the desired degrees $p_1,\ p_2$ for the surface.

The complete algorithms for performing this *knot placement* are given in algorithms 8 and 9.

3.5.3 No-arbitrage conditions

The no-arbitrage conditions applied by Fengler & Hin (2013) are listed in prop. 1. In this framework these conditions are applied as linear constraints on the control points of the approximated surface: $\left(\xi^*_{j_1}, \nu^*_{j_2}, \theta_{j_1,j_2}\right)_{j_1,j_2=0}^{q_1,q_2}$, where

$$\xi^*_{j_1} = \frac{\xi_{j_1+1} + \cdots + \xi_{j_1+p_1}}{p_1} \qquad \nu^*_{j_2} = \frac{\nu_{j_2+1} + \cdots + \nu_{j_2+p_2}}{p_2}$$

are the knot averages.

(C1) Convexity of the spline surface in the forward-moneyness direction is given by constraints on the coefficients and the knot averages in this dimension:

$$-1 \le \frac{\theta_{j_1+1,j_2+1} - \theta_{j_1,j_2}}{\xi^*_{j_1+1} - \xi^*_{j_1}} \le \frac{\theta_{j_1+2,j_2} - \theta_{j_1+1,j_2}}{\xi^*_{j_1+2} - \xi^*_{j_1+1}} \le 0,$$
$$j_1 = 0, \dots, q_1 - 2, \; j_2 = 0, \dots, q_2.$$

(C2) The lower- and upper bounds on the normed call prices are solely given by conditions on the coefficients:

$$\theta_{j_1,j_2} \ge 0 \quad \textit{and} \quad \theta_{j_1,j_2} \le 1, \quad j_1 = 0, \dots, q_1, \; j_2 = 0, \dots, q_2.$$

(C5) The non-decreasing call surface in the maturity-direction is obtained by constraints solely on the coefficients as well:

$$\theta_{j_1,j_2+1} - \theta_{j_1,j_2} \ge 0, \quad j_1 = 0, \dots, q_1, \; j_2 = 0, \dots, q_2 - 1.$$

These conditions are, as stated in the proposition, sufficient. Furthermore, these are used as the reference point for the no-arbitrage evaluations in the other methods assessed, and will therefore not be discussed further here.

Like for the method in sec. 3.2 by Fengler, it is here possible to exploit the linearity of the no-arbitrage constraints to reformulate the optimization problem into a constrained quadratic program. The complete method for fitting the surface is given in algorithms 10 and 11.

3.5.4 Conversion to the Local Volatility Surface

Dupire's local volatility function given in eqn. (6), $\hat{\sigma}^L_1(\kappa, \tau)$, can for this method be derived analytically from the definition of the tensor-product B-spline.

This is a bit messy notation-wise though, so here we have chosen to approximate the local volatility surface by replacing the derivatives in eqn. (6) with their numerical estimates. For the B-spline representation these numerical derivatives can easily be determined using the Matlab function intended for this purpose[7].

3.5.5 Tweaking the method

Evaluate surface in specified point $(\bar{\kappa}, \bar{\tau})$: This method fits the tensor product B-spline to the observed data points as described in section 3.5.2. Once the coefficients have been fitted it is possible to evaluate the call surface in any given point,

[7]See the Matlab function *fnder* for further details.

Algorithm 8: PartI: Fengler & Hin (2013) - Knot placement.

```
KnotSearch($\boldsymbol{\kappa}, \boldsymbol{\nu}, \Delta_\Xi, \Delta_\xi$)
    ▷ Use a linear TP B-spline:  $p_1 = p_2 = 1$.
    Set $\boldsymbol{\xi} = \{\xi_0, \xi_1, \xi_2, \xi_3\} = \{\min \boldsymbol{\kappa}, \min \boldsymbol{\kappa}, \max \boldsymbol{\kappa}, \max \boldsymbol{\kappa}\}$;
    Set $\boldsymbol{\nu} = \{\nu_{\min}, \boldsymbol{\nu}, \nu_{\max}\}$                    ▷ Add boundary knots.

    $\boldsymbol{\xi^u} = \boldsymbol{\xi}$                     ▷ Initialize the book-keeping knot sequence.
    while knots can be added to $\boldsymbol{\xi}$ do
        Set $q_1$ =length ($\boldsymbol{\xi}$) − ($p_1 + 2$);             ▷ Using eqn.(39).
        for $i = 1, \dots, q_1$ do
            $\xi^0 = \arg\min_{\xi \in [\xi_i, \xi_{i+1}]} \Xi_{mAIC} (\xi \cup \boldsymbol{\xi}, \boldsymbol{\nu})$
            ▷ Additional knot which added to the working knot
              sequence minimizes the selection criterion.

            if $\Xi_{mAIC} (\boldsymbol{\xi}, \boldsymbol{\nu}) - \Xi_{mAIC} \left(\xi^0 \cup \boldsymbol{\xi}, \boldsymbol{\nu}\right) > \Delta_\Xi$ then
                if $\xi^0 - \xi_i > \Delta_\xi \wedge \xi_{i+1} - \xi^0 > \Delta_\xi$ then
                    $\boldsymbol{\xi^u} = \boldsymbol{\xi^u} \cup \xi^0$.
                    ▷ Add the knot to the working sequence.
        $\boldsymbol{\xi} = \boldsymbol{\xi^u}$.
    return $\boldsymbol{\xi}$
```

$(\bar{\kappa}, \bar{\tau})$:

$$\hat{c}_1(\bar{\kappa}, \bar{\tau}) = \sum_{j_1=0}^{q_1} \sum_{j_2=0}^{q_2} \theta_{j_1,j_2} B_{j_1,p_1}(\bar{\kappa}) B_{j_2,p_2}(\bar{\tau})$$

This flexibility significantly distinguishes this method from the other methods presented in this paper.

▷ The order of the tensor product B-spline is given by the degree of the univariate splines. As a point of reference, note that a cubic spline is of order 4, equivalent to a degree of 3, referring to the number of coefficients.

In table 5 different values for the mean and standard deviation of the relative distance from the observed- to the fitted call prices is given. These numbers indicate that a degree of $p1 = 5$ gives the best fit to the observed call prices. Looking at the local volatility surfaces, it can be seen that $p1 = 3$ gives a *noisy* surface, this improves for surfaces of higher degree. These surfaces of degree above $p1 = 3$ resembles each other more or less, though a tendency of too much smoothing starts to show for $p1 = 7$.

Algorithm 9: PartII: Fengler & Hin (2013) - Knot placement.

```
KnotRelocateDelete($\boldsymbol{\xi}, \boldsymbol{\nu}, \Delta_\xi, p_1, p_2$)
    ▹ Use a TP B-spline of desired degree.
    Set $q_1$ = length ($\boldsymbol{\xi}$) − ($p_1$ + 2);                    ▹ Using eqn.(39).
    Set $\boldsymbol{\nu} = \{\{\nu_{\min}\}_{\times p_2}, \boldsymbol{\nu}, \{\nu_{\max}\}_{\times p_2}\}$        ▹ Add boundary knots.

    $\boldsymbol{\xi}^{delete}$, $\boldsymbol{\xi}^{add}$              ▹ Initialize the book-keeping sequences.
    for $i = p_1, \ldots, (q_1 - 1)$ do
        $\boldsymbol{\xi}_i = \boldsymbol{\xi} \backslash \xi_{i+1}$.              ▹ Sequence excluding the $i+1$'th knot.
        $\xi^0_{i+1} = \underset{\xi \in [\xi_i, \xi_{i+2}]}{\arg\min}\, \Xi^C_{mAIC}(\xi \cup \boldsymbol{\xi}_i, \boldsymbol{\nu})$
        s.t. $\xi^0_{i+1} - \xi_i > \Delta_\xi \wedge \xi_{i+2} - \xi^0_{i+1} > \Delta_\xi$
          ▹ Additional knot which added to $\boldsymbol{\xi}_i$ minimizes the selection
            criterion.

        if $\Xi^C_{mAIC}(\boldsymbol{\xi}_i, \boldsymbol{\nu}) < \Xi^C_{mAIC}\left(\xi^0_{i+1} \cup \boldsymbol{\xi}_i, \boldsymbol{\nu}\right)$ then
            if $\Xi^C_{mAIC}(\boldsymbol{\xi}_i, \boldsymbol{\nu}) < \Xi^C_{mAIC}(\boldsymbol{\xi}, \boldsymbol{\nu})$ then
                $\boldsymbol{\xi}^{delete} = \boldsymbol{\xi}^{delete} \cup \xi_{i+1}$.          ▹ Delete $i+1$'th knot.

        if $\Xi^C_{mAIC}\left(\xi^0_{i+1} \cup \boldsymbol{\xi}_i, \boldsymbol{\nu}\right) < \Xi^C_{mAIC}(\boldsymbol{\xi}_i, \boldsymbol{\nu})$ then
            if $\Xi^C_{mAIC}\left(\xi^0_{i+1} \cup \boldsymbol{\xi}_i, \boldsymbol{\nu}\right) < \Xi^C_{mAIC}(\boldsymbol{\xi}, \boldsymbol{\nu})$ then
                $\boldsymbol{\xi}^{delete} = \boldsymbol{\xi}^{delete} \cup \xi_{i+1}$.
                $\boldsymbol{\xi}^{add} = \boldsymbol{\xi}^{add} \cup \xi^0_{i+1}$.            ▹ Replace $i+1$'th knot.

    return $\boldsymbol{\xi} = \left(\boldsymbol{\xi} \backslash \boldsymbol{\xi}^{delete}\right) \cup \boldsymbol{\xi}^{add}$.
```

For the maturity direction, the degree of the spline have been tested for levels between 2 and 4 - the best fit is achieved for $p2 < 4$, but for $p2 = 2$ the surface is too *noisy*, just as we observed for $p1 = 3$.

Hence, like in Fengler & Hin it seems that degrees of $p1 = 5,\ p2 = 3$ gives the best fit of the surface for the dataset at hand.

▹ Fengler & Hin (2013) find that the quality of the output surfaces to a great extent depends on the distribution of knot sequence in the forward-moneyness dimension, while the distribution for the maturity dimension seems to be of less importance. Thus, the majority of the computational time is spend on optimizing the forward-moneyness knot sequence by applying algorithms 8 and 9.

The parameters used for this *knot sequence optimization* are given by: Δ_Ξ, Δ_ξ. The authors suggest to use $\Delta_\Xi = 10^{-5}$ and $\Delta_\xi = 10^{-4}$.

Algorithm 10: PartI: Fengler & Hin (2013) - approximation of the call surface.

```
input: $\tau_i$, $i=1,\ldots,N$                         ▷ observed maturity levels.
       $\kappa_i$, , $i=1,\ldots,N$                     ▷ observed forward-moneyness levels.
       $c_1(\kappa_i,\tau_i)$                           ▷ observed normed call prices.
       $\lambda$                                        ▷ smoothening parameter.

Set: $\Delta_\Xi$, $\Delta_\xi$, $\boldsymbol{\nu} = \{\tilde{\tau}_1,\ldots,\tilde{\tau}_J\}$.    ▷ Initialize parameters

Set: $\mathbf{B} = \begin{pmatrix} \left(B_{0,p_1}(\kappa_1),\ldots,B_{q_1,p_1}(\kappa_1)\right)^\intercal \otimes \left(B_{0,p_2}(\tau_1),\ldots,B_{q_1,p_1}(\tau_1)\right)^\intercal \\ \vdots \\ \left(B_{0,p_1}(\kappa_N),\ldots,B_{q_1,p_1}(\kappa_N)\right)^\intercal \otimes \left(B_{0,p_2}(\tau_N),\ldots,B_{q_1,p_1}(\tau_N)\right)^\intercal \end{pmatrix}$
  ▷ where the B-splines are given according to eqn. (42).

Def: $\Xi^{(C)}_{mAIC}(\boldsymbol{\xi},\boldsymbol{\nu}) = \log\left(\sum_{i=1}^{N} \frac{(c_1(\kappa_i,\tau_i)-\hat{c}_1(\kappa_i,\tau_i))^2}{N}\right) + 1 + \frac{2(tr(\mathbf{S})+1)}{N-tr(\mathbf{S})-2}$.
  ▷ Akaike Information Criterion with linear smoother matrix:
    $\mathbf{S} = \mathbf{B}\left(\mathbf{B}^\intercal\mathbf{B} + \lambda\mathbf{I}\right)^{-1}\mathbf{B}^\intercal$ and estimate of the call surface:
    $\hat{c}_1(\boldsymbol{\kappa},\boldsymbol{\tau}) = \mathbf{B}\boldsymbol{\theta}^*$, where $\boldsymbol{\theta}^*$ is obtained by the quadratic program given
    below.  The Q.P problem is unconstrained for $\Xi_{mAIC}$ and
    constrained for $\Xi^C_{mAIC}$.

Calculate:
  $\boldsymbol{\xi}_{tmp}$ = KnotSearch($\boldsymbol{\kappa}, \boldsymbol{\nu}, \Delta_\Xi, \Delta_\xi$);
  $\boldsymbol{\xi}$ = KnotRelocateDelete($\boldsymbol{\xi}_{tmp}, \boldsymbol{\nu}, \Delta_\xi, p_1, p_2$);
```

Here, the method has been run for a month of the Sep0405 data[8] using values: $\Delta_\Xi = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$. Looking at the mean and standard deviation of the relative distance to the observed call prices, one can see that $\Delta_\Xi = \{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$ produces the same knot sequence and hence the same fit to the data. $\Delta_\Xi = 10^{-3}$ gives the best fit by reducing the length of the knot sequence with a single knot for 2 out of the 22 dates observed. $\Delta_\Xi = 10^{-2}$ worsens the fit a bit by reducing the number of knots for the majority of the observed dates, but in return reduces the computational time significantly. Here, the primary focus is on the quality of the surface and parameter value $\Delta_\Xi = 10^{-3}$ is therefore chosen for the calculations.

Adjusting the parameter Δ_ξ either up or down worsens the fit to the observed data, both in terms of the mean as well as standard deviation of the relative distance. Hence, here the value is kept at $\Delta_\xi = 10^{-4}$, as suggested by Fengler

[8]See section 4.1 for details on the data used.

Algorithm 11: PartII: Fengler & Hin (2013) - approximation of the call surface.

```
Set: $\mathbf{D} = \mathbf{B}^\top\mathbf{B} + \lambda\mathbf{I}$;   $\mathbf{d} = \mathbf{B}^\top \begin{pmatrix} c_1(\kappa_1, \tau_1) \\ \vdots \\ c_1(\kappa_N, \tau_N) \end{pmatrix}$
  ▷ where I is the unit matrix.

Solve $\min_{\boldsymbol{\theta}} \frac{1}{2}\boldsymbol{\theta}^\top\mathbf{D}\boldsymbol{\theta} - \boldsymbol{\theta}^\top\mathbf{d}$
$s.t.$ $\left[\mathbf{C}^{(1)}\ \mathbf{C}^{(2)}\ \mathbf{C}^{(3)}\ \mathbf{C}^{(4)}\right]^\top \leq [\mathbf{0}\ \mathbf{1}\ \mathbf{0}\ \mathbf{0}]^\top$.

  ▷ Solve the quadratic program subject to the set of
    linear constraints given in sec. 3.5.3, see
    appendix A.2 for details.

Def: $[\hat{\kappa}_1; \hat{\kappa}_M] \times [\hat{\tau}_1; \hat{\tau}_L] = \hat{\boldsymbol{\kappa}} \times \hat{\boldsymbol{\tau}} \subseteq \{\kappa_i \times \tau_i\}_{i=1,\dots,N}$
  ▷ Regular (non-scattered) grid for which the
    approximation of the call surface is derived.
    Contained in the rectangle defined by the input grid.
    Allowed to be non-uniform.

Set: $\hat{c}_1(\hat{\boldsymbol{\kappa}}, \hat{\boldsymbol{\tau}}) = \sum_{j_1=0}^{q_1} \sum_{j_2=0}^{q_2} \theta_{j_1,j_2} B_{j_1,p_1}(\hat{\boldsymbol{\kappa}}) B_{j_2,p_2}(\hat{\boldsymbol{\tau}})\ \hat{\sigma}_1^L(\hat{\boldsymbol{\kappa}}, \hat{\boldsymbol{\tau}}) \overset{*}{\approx} \sqrt{\frac{2\frac{\partial^2 \hat{c}_1(\hat{\boldsymbol{\kappa}},\hat{\boldsymbol{\tau}})}{\partial \tau}}{\hat{\boldsymbol{\kappa}}^2 \frac{\partial^2 \hat{c}_1(\hat{\boldsymbol{\kappa}},\hat{\boldsymbol{\tau}})}{\partial \kappa^2}}}$.
  ▷ Calculate values of the call- and the local volatility
    surface in the user-defined grid-points.
  ▷ $*$: approximate the derivatives involved using Matlab's
    built-in function for numerical evaluation of
    B-splines.
```

& Hin (2013).

It is unclear from the article whether the end-knots should be set equal to the minimum and maximum observed forward-moneyness levels, respectively, or if they should be *padded*. Here, a small padding of -0.01 and $+0.01$ have been chosen.

▷ As mentioned under the previous bullet point, Fengler & Hin (2013) states that the output is not particularly dependent on the distribution of the knot

p1	$\mathbb{E}$(relative distance)	**Std**(relative distance)
3	12.92	5.86
4	12.99	6.80
5	12.36	4.77
6	13.79	5.36
7	14.15	5.94

Table 5: Goodness of fit associated with varies spline degrees for the forward-moneyness direction. Measured for a subset of the Sep0405 dataset.

sequence for the maturity dimension.

This theory have been tested by partly by altering the number of knots and partly by adjusting the position of these. The experiments indicate that the best fit can be obtained when the number of knots equals the order of the spline. We determined above that the spline degree should be set to: $p_2 = 3$, which sets the order, and hence the number of knots, equal to 4. The exact position of these knots do not seem to have a significant impact on the result, as stated in the original paper. Here we have used quantiles of the maturities available for a given date to set the knot sequence: $\boldsymbol{\nu}_1 = \{Q_0(\boldsymbol{\tau}), Q_{1/3}(\boldsymbol{\tau}), Q_{2/3}(\boldsymbol{\tau}), Q_1(\boldsymbol{\tau})\}$ and $\boldsymbol{\nu}_2 = \{Q_0(\boldsymbol{\tau}), Q_{2/4}(\boldsymbol{\tau}), Q_{3/4}(\boldsymbol{\tau}), Q_1(\boldsymbol{\tau})\}$. The sequence $\boldsymbol{\nu}_2$ seems to give a slightly better fit and a less *noisy* surface at the margins, thus we have used this sequence for the maturity dimension.

▷ Like for Fengler (2009) described in section 3.2, this method too involves a smoothing parameter λ. Adjusting this parameter, we observe that values: $\lambda = 1e-5,\ 1e-11$ produced *noisy* surfaces that do not fit data well enough. Values $\lambda = 1e-7, 1e-9$ produces very similar results and we therefore choose $\lambda = 1e-9$, as for the method by Fengler.

From this extensive description of the amount of *tweaking* necessary it is evident, that just as flexible as the method is to evaluate, just as cumbersome is it to set up.

4 EMPIRICAL WORK

4.1 DATA

The empirical work are carried out for daily observations of European style options written on the S&P500 index. The end-of-day quotes, provided by the OptionMetrics database, are retrieved for two periods:

- Sep.'04-Sep.'05, pre-crisis period where the volatility was unusually low.
- Sep.'08-Sep.'09, the crisis period where the turbulence and the volatility in the financial markets were at its peak.

These periods have been chosen such that the algorithms are tested for a *baseline* period, Sep. '04-'05, and a *stress-testing* period, Sep.'08-'09.

...while fitting algorithms cope well in providing interpolations on market data (which mostly exhibit only slight arbitrage violations), artificially created stress scenarios exhibit strong arbitrage violations, and fitting algorithms may take much longer, or even fail to find reasonable fits that ensure smoothness and lack of arbitrage. (Gope & Fries (2011))

In accordance with the quote above, any given method is required to perform well for the *baseline* period in order to be considered applicable in practice. It is expected that the performance for the *stress-testing* period is what will distinguish the quality of the algorithms.

4.2 FILTERING

The market data quoted in the OptionMetrics database are contaminated with a number of measurement errors as pointed out by Hentschel (2003) in the quote below.

...prices are observed with errors stemming from finite quote precision, bid-ask spreads, non-synchronous observations...

Hence, errors in the quoted prices stem from rounding to quote precision, the uncertainty regarding the *true* price and the non-synchronous logging of closing prices for the options and the value of their underlying - often measured 15 min. apart according to Hentschel.

We therefore refine the raw datasets obtained from OptionMetrics database by applying a number of filters to obtain a reasonable dataset that can be used as input for the algorithms described in section 3. These filters are mainly inspired by Constantinides *et al.* (2013) who also work with S&P500 datasets.

> *...the estimation or calibration process must be robust against noise present in price observations due to market micro-structure effects, such as bid-ask spreads, discrete ticks in prices or quotes, non-synchronous trading, effects due to the auction mechanism itself, or simply to misprints (for a detailed analysis of errors in IV data we refer to Hentschel 2003)...* (Benko *et al.* (2007))

Constantinides *et al.* apply very restrictive filtering which for the most part will be followed here. Although, a lot of *stress-testing* (see quote by Benko *et al.* above) is left-out on this account it is important that we have a regular grid of data, and that we do not end up testing the methods for the *wings* which is a topic outside the scope of this paper.

Also, as we discussed in section 2.1, the dataset must be *normalized* before being provided as input. The interest rates and dividend yields quoted in the OptionMetrics database are unsuited for this purpose - which Constantinides *et al.* (2013) also brings to attention during their filtering process. Here, we obtain these interest rates and dividend yields by backing them out from the filtered data using a simple linear regression model for the put-call parity.

4.2.1 Raw data

▷ **Identical filter** Duplicate entries with identical terms (option type, strike, expiration date) and identical price are removed. For entries with identical terms but different prices, the quote with the widest bid-ask spread is removed.

- *Sep '04-'05:* 0 removed.
- *Sep '08-'09:* 0 removed.

▷ **Zero bid** Quotes with zero bids are removed to avoid low valued options and negative bids (which this could indicate). According to practitioners this might occur if a trader uses the Black-Scholes formula to derive a price and afterwards adds a bid-ask spread to that.

- *Sep '04-'05:* 11623 quotes removed.

- *Sep '08-'09:* 41332 quotes removed.

▷ **Days to maturity** The observed data are truncated to only include data with more than 7 days to maturity. According to Constantinides *et al.* (2013) quotes with shorter maturities will tend to move *erratically*, which tend to cause troubles in the evaluation of the implied volatility.

In the opposite end, data is truncated to quotes with at most one year to maturity. This filtering deviates from Constantinides *et al.* (2013) who proposes a more restrictive filtering of at most 0.5 years to maturity, but for the $S\&P500$ options it is not uncommon with longer durations according to practitioners.

- *Sep '04-'05:* 30133 quotes removed.
- *Sep '08-'09:* 68844 quotes removed.

▷ **Moneyness** Constantinides *et al.* (2013) proposes to remove all quotes with moneyness, $\frac{K}{s}$, below 0.8 or above 1.2. This rather restrictive filter is applied to avoid working in the wings. This filter could be *loosened* by for instance letting these moneyness boundaries depend on the time to maturity, such that for short maturities the range starts at $[0.8, 1, 2]$ and ends at $[0.4, 1.8]$ for expiries tending towards 5 years. Instead of a rectangle, this will then leave give a cone of data.

- *Sep '04-'05:* 25673 quotes removed.
- *Sep '08-'09:* 155.718 quotes removed.

▷ **Implied volatility** *Remark: this filter and the next is applied after the interest rates and dividend yields have been backed out from the data using a simple linear regression, see section 4.2.2.*

Quotes with calculated implied volatilities lower than 5% or higher than 100% indicate quotation problems and are therefore removed according to Constantinides *et al.*. One could perhaps discuss whether the lower boundary of 5% unambiguously imply quotations problems. However none of the computed volatility levels hits these boundaries in either of two periods.

We do though see NaN's for some of the quotes stemming from troubles in the implied volatility solver provided by Matlab.

- *Sep '04-'05:* 47 quotes removed.
- *Sep '08-'09:* 6 quotes removed.

- ▷ **Mertons Tunnel** The final filter is given by the fundamental no-arbitrage boundaries, also termed *Merton's tunnel*, which for a given maturity reads:

$$\left(se^{-q_j\tau_j} - e^{-r_j\tau_j}K_{ij}\right)^+ \leq c_s(K_{ij}, \tau_j) \leq se^{-q_j\tau_j} \qquad \forall \tau_j.$$

  Thus, quotes violating this basic condition contain arbitrage opportunities and are therefore removed from the dataset.

  - ∘ *Sep '04-'05:* 0 removed.
  - ∘ *Sep '08-'09:* 0 removed.

4.2.2 Extracting interest rate and dividend yield

In order to transform the data to fit the normed framework, see sec. 2.1, we need to obtain the interest rates and dividend yields. Unfortunately these are not available in a usable format in the OptionMetrics database. The zero rate is known to be imprecise, see the discussion in Constantinides *et al.* (2013), and there is only one dividend quote available per date. These rates are therefore extracted from the filtered data by fitting a simple regression model of the put-call parity for each *(date, expiry)*-combination.

Initially, assume that the interest rates and dividend yields are deterministic functions of *date* and *expiry*. Thus, for a given *(date, expiry)*-combination, represented by τ, these rates are constant across strike levels. Thus, for each *(date, expiry)*-combination the put-call parity can be fitted to the data by a simple linear regression model:

$$\begin{aligned} & c_s(K_i, \tau) - P_s(K_i, \tau) = se^{-q_\tau\tau} - e^{-r_\tau\tau}K_i + \varepsilon_i, \\ \Rightarrow \quad & y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \end{aligned} \tag{44}$$

such that the the implied interest rates and dividend yields can be *backed out* from the regression coefficients:

$$\begin{aligned} \hat{\beta}_0 &= se^{-q_\tau\tau} & \Leftrightarrow \quad q_\tau &= \frac{-\ln\left(\frac{\hat{\beta}_0}{s}\right)}{\tau} \\ \hat{\beta}_1 &= -e^{-r_\tau\tau} & \Leftrightarrow \quad r_\tau &= \frac{-\ln\left(-\hat{\beta}_1\right)}{\tau}. \end{aligned}$$

Note that this regression can only be applied if there is sufficient data available for the given *(date,expiry)*-combination. Hence, we have to leave out the *(date,expiry)*-pairs with insufficient data.

- *Sep '04-'05:* 0 removed.
- *Sep '08-'09:* 66 *(date,expiry)*-pairs removed.

It can further be noted, that these *(date,expiry)*-pairs are evenly distributed over the entire year.

The call- and put- quotes used as input to this regression needs to be calculated from their respective bid-ask prices listed in the OptionMetrics database. In the financial literature it is standard practice to compute these final quotes, or mid-quotes, by calculating the average of their bid-ask spread. This average can be presented as the affine combination of the bid- and ask prices for $\lambda_c = \lambda_p = 0.5$:

$$c_1(\cdot) = \lambda_c c_1^{bid}(\cdot) + (1-\lambda_c)c_1^{ask}(\cdot), \qquad p_1(\cdot) = \lambda_p p_1^{bid}(\cdot) + (1-\lambda_p)p_1^{ask}(\cdot)$$

Though this is not necessarily the *true* price, as mentioned in the beginning of this section. Thus, here we will try to adjust the parameters: λ_c, λ_p, when some of the estimates from the regression in eqn. (44) turns out *badly* - interest rates or dividend yields outside some pre-defined bands, where the lower bound on the interest rate given by zero[9]. In practice, the sample mean of these parameters over the two periods lies closely around 0.5 as can be seen below:

- *Sep '04-'05:* $\mu_{\lambda_c} = 0.4910, \mu_{\lambda_p} = 0.4604$.
- *Sep '08-'09:* $\mu_{\lambda_c} = 0.4936, \mu_{\lambda_p} = 0.5112$.

If for a given *(date,expiry)*-combination it is not possible to adjust the mid-quote such that the interest rate and dividend yield estimates turn out reasonable, all quotes are disregarded for that combination.

- *Sep '04-'05:* 36 *(date,expiry)*-pairs removed.
- *Sep '08-'09:* 61 *(date,expiry)*-pairs removed.

An alternative to taking the average of the spread as a starting point, as done here, is to use the median of the spread, as suggested in Glaser & Heider (2012).

To summarize we give some sample statistics of the estimated interest rate and dividend yield below:

[9]This definition of *badly* is based on Matlabs implied volatility solver which does not accept negative interest rates.

- *Sep '04-'05:*

$$\mu_{r_\tau} = 0.0298,\ s_{r_\tau} = 0.0095 \quad \min_{r_\tau} = 3.9901e-04,\ \max_{r_\tau} = 0.0616,$$
$$\mu_{\delta_\tau} = 0.0178,\ s_{\delta_\tau} = 0.0024 \quad \min_{\delta_\tau} = 0.0115, \max_{\delta_\tau} = 0.0206.$$
$$\text{band}_{r_\tau} \overset{*}{=} [0.0180; 0.0618] \approx [0.00; 0.0618].$$
$$\text{band}_{\delta_\tau} \overset{*}{=} [0.0115; 0.0206].$$

- *Sep '08-'09:*

$$\mu_{r_\tau} = 0.0132,\ s_{r_\tau} = 0.0099 \quad \min_{r_\tau} = 4.8960e-06,\ \max_{r_\tau} = 0.0598,$$
$$\mu_{\delta_\tau} = 0.0260,\ s_{\delta_\tau} = 0.0067 \quad \min_{\delta_\tau} = 0.0052, \max_{\delta_\tau} = 0.0386.$$
$$\text{band}_{r_\tau} \overset{*}{=} [-0.0094; 0.0602] \approx [0.00; 0.0602].$$
$$\text{band}_{\delta_\tau} \overset{*}{=} [0.0051; 0.0386].$$

The bands are given by: mean $\pm\, 3 \cdot$ std, of the OptionMetrics provided quotes for the interest rate and dividend yield, respectively.

4.2.3 Final dataset

The final set of *normed* option quotes are generated through eqn. (2) using the interest rates, dividend yields and their corresponding mid-quotes.

As for the filtering process we here have a trade-off between standardizing the data to ease implementation and the ability to test the methods. In this case, the possibility of testing how the algorithms handle interest rates and dividends is left out in favor of a unified framework across all methods.

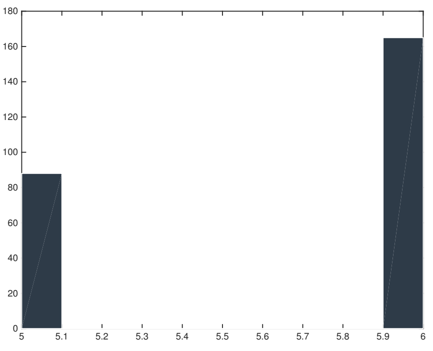
We have tried to illustrate the *distribution* of the final quotes by determining:

- the number of distinct expiries available for each observed date.
- the number of quotes available for each *(date,expiry)*-pair.

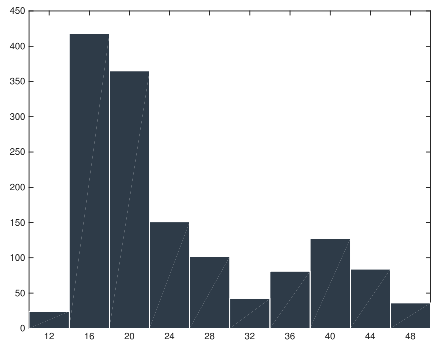
In the *Sep'04-'05* dataset, 1 date with only 1 maturities exists, and in the *Sep'08-'09* dataset, two dates with only 3 maturities exist. These have been removed from the datasets. The final data then consist of:

- *Sep '04-'05:* 36326 quotes distributed over 253 dates with an average of 5.7 maturities per date.

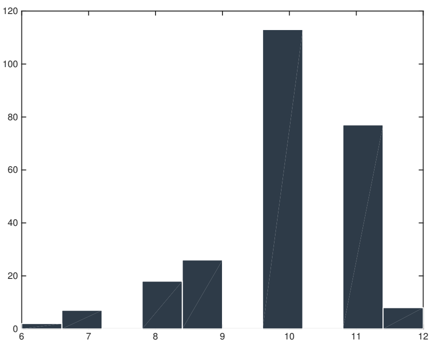
Figure 8: Histograms of *quote distribution*.



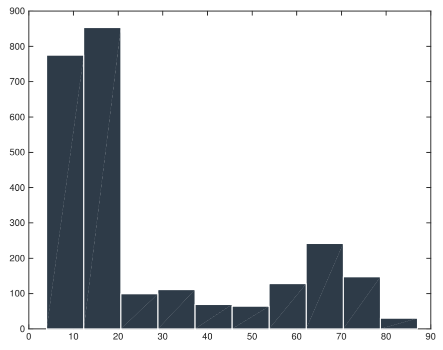
Sep’0405: # expiries per date.



’Sep’0405: # quotes per (date, expiry)-pair.



Sep’0809: # expiries per date.



Sep’0809: # quotes per (date, expiry)-pair.

- *Sep ’08-’09:* 68594 quotes distributed over 251 dates with an average of 10 maturities per date.

Hence, we end up having almost twice as much data available for the post-crisis period after our filtering has been applied which is also illustrated in figure 8.

4.3 Quantitative Comparison of the Methods

4.3.1 Output grid

The output surfaces - call, implied volatility, local volatility - are for each date in a given dataset derived for a user-defined grid and an input grid, determined by the observed grid-points for that given date. The user-defined grid is set separately for each dataset: Sep’0405, Sep’0809, in accordance with algorithm 12.

Algorithm 12: Setting the user-defined input grid.

```
input:
    $(t_j, T_{ji}, \kappa_{ji}),\ _{j=1,\dots,J,\ ji=1,\dots,I_j}$
                ▷ dataset: date, forward-moneyness, expiry.
    $M$         ▷ number of grid-points, forward-moneyness.
    $L$         ▷ number of grid-points, maturity.

for $j = 1 \to J$ do
    $\min_{\kappa,j} = \min\left(\kappa_1, \dots, \kappa_{I_j}\right);\ \max_{\kappa,j} = \max\left(\kappa_1, \dots, \kappa_{I_j}\right);$
    $\min_{\tau,j} = \min\left(\tau_1, \dots, \tau_{I_j}\right);,\ \max_{\tau,j} = \max\left(\tau_1, \dots, \tau_{I_j}\right);$

$\min_\kappa = \max\left(\min_{\kappa,1}, \dots, \min_{\kappa,J}\right);\ \max_\kappa = \min\left(\max_{\kappa,1}, \dots, \max_{\kappa,J}\right);$
$\min_\tau = \max\left(\min_{\tau,1}, \dots, \min_{\tau,J}\right);\ \max_\tau = \min\left(\max_{\tau,1}, \dots, \max_{\tau,J}\right);$
$\Delta_\kappa = \frac{\max_\kappa - \min_\kappa}{M};\quad \Delta_\tau = \frac{\max_\tau - \min_\tau}{L};$
return
    $[\kappa_1; \kappa_M]$ = seq($\min_\kappa : \Delta_\kappa : \max_\kappa$); $[\tau_1; \tau_L]$ = seq($\min_\tau : \Delta_\tau : \max_\tau$);
```

Interpolation in the output surfaces can introduce arbitrage if not conducted carefully. Thus, the calibration has for as many methods as possible been carried out for a grid given as the combination of the user-defined- and the input grid. In practice this has been possible for all methods but the one given by Benko *et al.* in sec. 3.1. See sections *tweaking* for details on the specific methods.

4.3.2 COMPUTATIONAL TIME

The run time - quoted in seconds - for the respective methods has been measured for each observed date separately. This measure consists solely of the pure computation time for a given algorithm. Hence, all data manipulation, graph plotting, etc. are left out. Furthermore, it should be noted that all methods have been implemented using Matlabs library functions to the extend possible. This means that the speed and the quality of all optimizations are determined by Matlabs build-in functionality.

The run times for the various methods are depicted in figure 9. It can be noted from these that Benko *et al.* (2007) has a run time which by far exceeds the others. This is caused by the fact, that this algorithm runs simultaneous optimizations for all maturities in the calibration grid at once, see algorithm 1 for details.

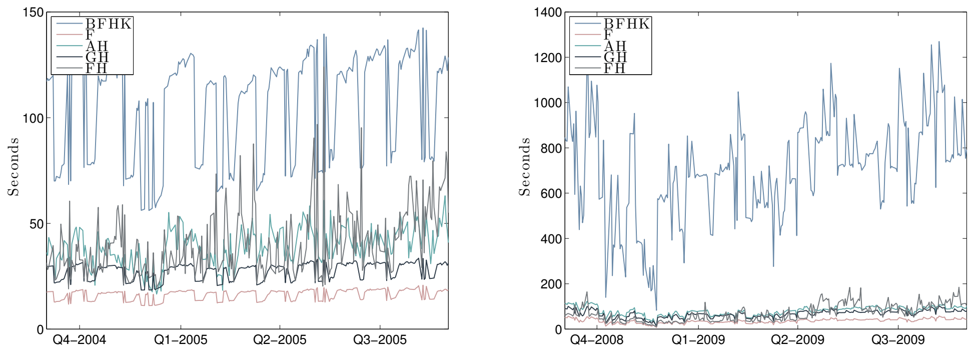


Figure 9: Execution run time in seconds.

In table 6, the run time for each dataset are summarized using the sample mean and standard deviation. The multiplier indicates the factor by which the run time for the *stress-testing* year, Sep'08-'09, is greater than for the baseline year, Sep'04-'05.

Table 6: Sample mean and standard deviation of the *run time (seconds)*.

Method	Sep'04-Sep'05		Sep'08-Sep'09		Multiplier
	Mean	Std.	Mean	Std.	
Benko *et al.* (2007)	106.02	25.06	696.50	220.08	6.57
Fengler (2009)	16.34	2.48	35.79	9.68	2.19
Andreasen & Huge (2011)	39.11	8.84	79.93	18.79	2.04
Glaser & Heider (2012)	27.38	4.01	66.89	16.51	2.44
Fengler & Hin (2013)	41.09	16.30	70.85	38.32	1.72

Here it is again evident that Benko *et al.* (2007) is a very expensive method, especially during Sep'08-'09 where it on average consumes 6.57 times as much computation time, than for the previous period Sep'04-'05. For the rest of these methods this number lies around 2.

Comparing the run time for the different methods one can see that the fastest method, Fengler (2009), is 6.5 times faster than the slowest, Benko *et al.* (2007), in Sep'04-'05, and 19.5 times faster in Sep'08-'09. Generally, Fengler (2009) is around twice as fast compared to the other methods: Andreasen & Huge (2011),Glaser & Heider (2012) and Fengler & Hin (2013), which all lie around the same level. In order to conclude further from these levels, they must be weighed up against the quality of their respective fits - which is assessed in the following sections.

Concluding: From these experiments there do appear to be a significant difference in the relative computational time for the respective methods. We saw that Benko *et al.* (2007) stood out as the by far most expensive method, and Fengler (2009) as the cheapest, while the others lay at similar levels. Furthermore, it was noticed that the computational time on average increased by a factor 2 when going from the *base*-period to the *stress-testing*-period. This can either be a sign of increased difficulties in fitting the more volatile data for this period, or a sign of the additional effort needed to fit the additional quotes (twice as many) for this period.

4.3.3 The call surface - fit to observed quotes

The majority of the algorithms approximates, or interpolates, a call surface from the observed data, see table 2, and afterwards derive the implied- and local volatility surface from this. The exception being Benko *et al.* (2007) which instead approximates the implied volatility surface.

The enclosed animations of the call surfaces across different dates for each of the two datasets, indicate that regardless of which surface has been fitted, the generated call surface turns out regular, smooth, and closely fitted to the market quotes for all methods.

This *goodness* of fit for the various call surfaces is further investigated by measuring the *distance* between the approximated values, $\hat{c}_1(\kappa_i, \tau_i)$, and the observed values, $c_1(\kappa_i, \tau_i)$. This distance is given by the euclidean norm of the relative distance:

$$\text{Relative Distance} = \sqrt{\sum_{i=1}^{N} \left(\frac{c_1(\kappa_i, \tau_i) - \hat{c}_1(\kappa_i, \tau_i)}{c_1(\kappa_i, \tau_i)} \right)^2}.$$

In table 7 this distance is reported by its sample mean and standard deviation across each period for all methods.

One would intuitively expect that the distance for the *stress-testing* period would be greater compared to the *baseline* period, due to the additional efforts needed in order to fit these more volatile observations. But in practice table 7 shows mixed results when comparing these two periods. The mean distance turns out to be reduced for the *stress-testing* period for all methods except for Andreasen & Huge (2011), while the standard deviation is increased, except for Fengler & Hin (2013).

TABLE 7: Call price: Sample mean and standard deviation of the *relative distance*.

	Sep'04-Sep'05		Sep'08-Sep'09	
Method	Mean	Std.	Mean	Std.
Benko *et al.* (2007)	0.89	0.52	0.87	0.69
Fengler (2009)	1.88	0.56	1.05	0.71
Andreasen & Huge (2011)	0.31	0.25	0.55	1.02
Glaser & Heider (2012)	4.14	0.54	2.94	1.21
Fengler & Hin (2013)	16.12	9.68	5.66	7.08

This decrease in the mean distance could be attributed to the increased amount of observations available for the dataset: Sep'08-'09, see sec. 4.2.3 for details. In which case it would be concluded that the mean distance does not seem to be affected by the volatility level in the market for any of the methods. The amount by which the standard deviation is increased differs for the various methods and seems to be significantly larger for Andreasen & Huge (2011) and Glaser & Heider (2012).

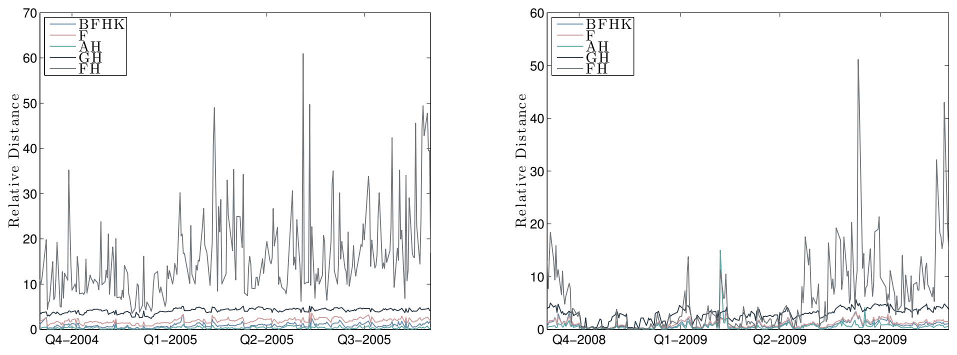


FIGURE 10: *Relative distance* between the fitted call surface and the observed values.

The mean and standard deviation of the distance also differs among the methods themselves. Especially Fengler & Hin (2013) stands out with a remarkably bad fit compared to the other methods, this is also easily seen in figure 10, where the relative distance has been depicted as a function of time for each method. The best fit to the data is obtained by Andreasen & Huge (2011), this is to be expected as this method is the only one utilizing interpolation rather than approximation.

Another interesting comparison between the distance and run time are given in

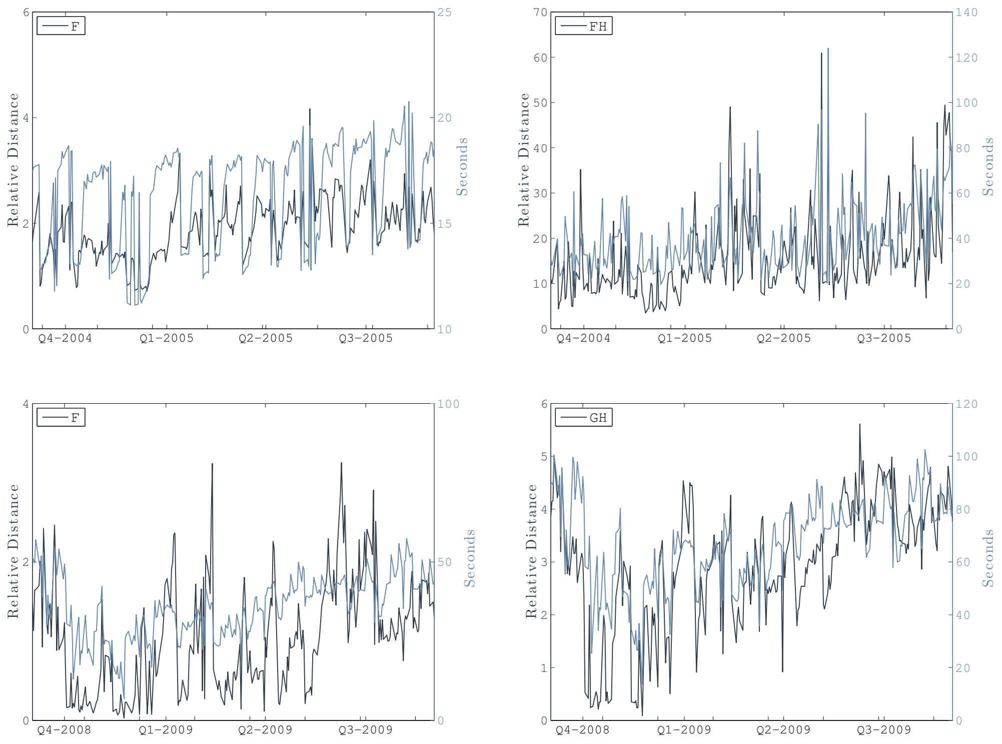


Figure 11: Relative distance vs Execution time in seconds.

figure 11 for a couple of the methods. For Sep'04-'05 there seems to be a relation between the *goodness of fit* and the run time for Fengler & Hin (2013), while the plot for Fengler (2009) is more ambiguous. For Sep'08-'09 the relation for Fengler (2009) though becomes more pronounced, and also the plot for Glaser & Heider (2012) indicates that there might be a correlation between the two measures. This hypothesis is quantified by table 8 where the sample correlations are reported.

Table 8 gives an indication of a small positive correlation between the distance and the run time, thus when the distance increases a corresponding increase in the run time can be observed. This indicates that the algorithms produce a larger distance for dates where the optimization routines has spend more time on fitting the data. This could suggest that the optimization routine for these dates experience some kind of difficulty, exceeding the maximum number of iterations, or for some other reason not being able to find an optimal fit. Hence, this correlation probably states more about Matlabs built-in functions than it does about the performance of the methods.

Table 8: Call price: Sample correlations between the *relative distance* and the run time.

	Corr	
Method	Sep'04-'05	Sep'08-'09
Benko *et al.* (2007)	0.64	0.51
Fengler (2009)	0.74	0.66
Andreasen & Huge (2011)	0.44	0.19
Glaser & Heider (2012)	0.69	0.78
Fengler & Hin (2013)	0.63	0.57

Concluding: In the 3D graphs of the call surfaces depicted over time it was observed that these all were regular, smooth and of a similar shape. When taking a closer look at the relative distances to the observed data points, there were some differences though. Fengler & Hin (2013) as well as Glaser & Heider (2012) stood out with the worst fit, especially the former, while our only interpolation method by Andreasen & Huge (2011) obtained the best fit, as expected. It was also noticed that the fit in general improved when the number of available quotes increased. Just as it was noted that a positive correlation between the relative distance and the computational time might indicate shortcommings in the optimization procedure.

4.3.4 The implied volatility surface

The implied volatility surface are for all methods, except Benko *et al.* (2007), derived from the fitted call surface using Matlabs built-in implied volatility function. This built-in function seems to be of limited functionality, as it for some of the options is not able to determine the implied volatility and instead returns a NaN. But for Benko *et al.* (2007) this source of error does not influence the implied volatility surface, which is directly fitted to the observed market quotes, see algorithm 1 for details. This difference could be contributing to the fact that Benko *et al.* (2007), as one of the only methods, is able to fit the implied volatility smile for small maturities when this becomes pronounced. The other method is Andreasen & Huge (2011) which interpolates the observations and therefore are able to give a better fit to the observed quotes, as were seen in table 7.

Below is a brief summary of the characteristics of the enclosed animations for the implied volatility surfaces across different dates for each of the two datasets:

- **Sep'04-'05:** the implied volatility surface for this period lies on a stable level with a volatility smile for the shortest maturities varying in level of *pronouncement* over the period.

- ○ Benko *et al.* (2007) are, as already mentioned, able to fit the smile for the smallest maturity level when this becomes pronounced. This fitted smile does, however, have dents and far OTM these become severe. Hence, the region defined by a large forward-moneyness and a small maturity level must be used with great care.

- ○ Fengler (2009) generates a very smooth surface which fits the observed levels nicely, except for the smallest maturity level. Here the method experiences problems when the smile becomes too pronounced - the fitted surface are simply not able to capture these high levels.

- ○ Andreasen & Huge (2011) is the second method that is able to fit the implied volatility smile when this becomes pronounced for small maturities. Hence, this surface gives the best fit to the observations while still being smooth, the only issue is a couple of small *curles* at the edge for the smallest forward-moneyness level.

- ○ Glaser & Heider (2012) generates a surface which for small maturity levels experience severe problems: For ITM options the surface has significant dents, and parts of the surface are missing (NaN) for $\kappa < 0.9$. For OTM options there is an entire region where the surface have a value of zero.

- ○ Fengler & Hin (2013) also experience problems with missing values far ITM and the smallest maturity level. Hence, this method is also not able to fit the high values when the implied volatility becomes pronounced. The remaining surface fits the observations smoothly.

• **Sep'08-'09:** the implied volatility surface for this period is quite flat for all maturities, where the general level is varying over the period.

- ○ Benko *et al.* (2007) the surface is smooth and fits the data nicely, except for rare spikes (both up and down) occurring in the corners of the surface.

- ○ Fengler (2009) as for the previous period, this surface fits the observed data points smoothly, except when the implied volatility values become too high far ITM or far OTM for the shortest maturity.

- Andreasen & Huge (2011) generally gives a very good fit to the quoted implied volatilities, even for high levels. Occasionally, a spike or a dent becomes visible far ITM for the shortest maturity.

- Glaser & Heider (2012) continues to experience problems with a region consisting of zero valued implied volatilities, though for a few dates this region is replaced by nicely fitted values. The troubles of fitting the smile for the shortest maturity persists, however, this is now reflected in spikes and dents rather than missing values (NaN's).

- Fengler & Hin (2013) are experiencing downward spikes far ITM for short maturities, for the majority of the dates in this period. Thus, whenever levels rise by a small amount compared to the rest of the surface.

From the analysis above two things can generally be concluded. First, the main feature that distinguishes these methods, in relation to this surface, is whether or not they are able to fit the implied volatility smile along with their *stability* at the edges. Second, the *local* methods, Benko *et al.* (2007) as well as Glaser & Heider (2012), seem very sensitive towards the distribution of the observed data-points. In *Sep'04-'05* both methods experience difficulties in a region where no observations are available[10] - ITM options for short maturities are illiquid - and in *Sep'08-'09*, where more data-points are available, these problems are limited to Glaser & Heider (2012) and they are not as severe, as for the previous period.

The summary of the fit for the various implied volatility surfaces above thus shows, that for this type of surface it does have an impact on the result which method is chosen for the approximation/interpolation. Below in table 9 pair-wise comparisons of the mean levels of the implied volatility surfaces are given, to see whether these differences in fit have an impact on the overall implied volatility level for the various methods.

In table 9 the pattern from table 7 repeats itself, as the mean of the pair-wise distances are smaller for the latter period, *Sep'08-'09*. Here also the standard deviation decreases which can be interpreted as an increased *closeness* in surface values for a larger part of the surface. Again this can be attributed to the increased number of observations for this period.

[10]Although extrapolation is not considered in this paper, there are corner-regions for the shortest maturity levels without observations due to the definition of the user-defined output grid, see algorithm 12 for details.

Table 9: Implied Volatility: Sample mean and standard deviation of the absolute difference in levels accross dates.

		F		AH		GH		FH	
	Sep	Mean	Std	Mean	Std	Mean	Std	Mean	Std
BFHK	'0405	0.0034	0.0047	0.0025	0.0026	0.0140	0.0247	0.0070	0.0123
	'0809	0.0013	0.0020	0.0021	0.0025	0.0035	0.0038	0.0015	0.0021
F	0405			0.0020	0.0043	0.0131	0.0219	0.0081	0.0159
	'0809			0.0016	0.0020	0.0032	0.0031	0.0011	0.0010
AH	'0405					0.0139	0.0239	0.0072	0.0128
	'0809					0.0043	0.0041	0.0019	0.0023
GH	'0405							0.0192	0.0356
	'0809							0.0032	0.0033

The difficulties described above for the surface by Glaser & Heider (2012) here manifest itself by having the largest distance to the other surfaces for both periods. Thus, the method by Glaser & Heider (2012) can be regarded as the *outsider* in the context of implied volatility surfaces.

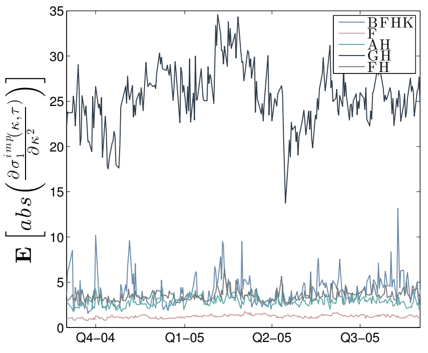
Though the *eyeball*-smoothness have already been discussed above for the methods, this can be formally quantified by taking a look at the second order derivative for both dimensions of the surface. The smaller the derivative, the smoother the surface is. The second order derivatives are calculated using the central finite difference approximation given eqn. (31) for the uniform user-defined grid of implied volatility values:

$$\begin{aligned}\frac{\partial^2\sigma_1^{imp}(\kappa_m,\tau_l)}{\partial\kappa^2} &\approx \frac{\sigma_1^{imp}(\kappa_{m+1},\tau_l)-2\sigma_1^{imp}(\kappa_m,\tau_l)+\sigma_1^{imp}(\kappa_{m-1},\tau_l)}{\Delta_\kappa},\\ \frac{\partial^2\sigma_1^{imp}(\kappa_m,\tau_l)}{\partial\tau^2} &\approx \frac{\sigma_1^{imp}(\kappa_m,\tau_{l+1})-2\sigma_1^{imp}(\kappa_m,\tau_l)+\sigma_1^{imp}(\kappa_m,\tau_{l-1})}{\Delta_\tau}.\end{aligned} \tag{45}$$

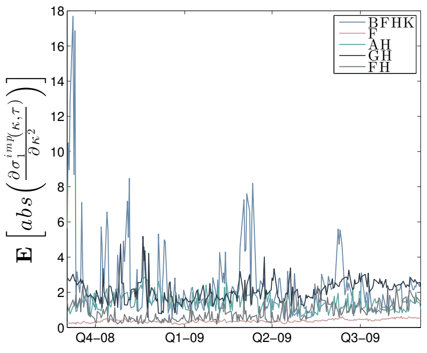
As mentioned previously, the 2nd order derivative wrt. forward-moneyness is the bi-product for some of the optimization algorithms. But none of these have the 2nd order derivative wrt. maturity as an output value. Thus, in order to standardize this *smoothness*-measure across methods and across variables, the finite difference approximation is used for all. The results are given in figure 12. Although the overall *smoothness*-level is determined by the observed quotes, it is here possible to see whether some of the methods deviate from this:

(12a) When observing the *smoothness* in the forward-moneyness direction, two

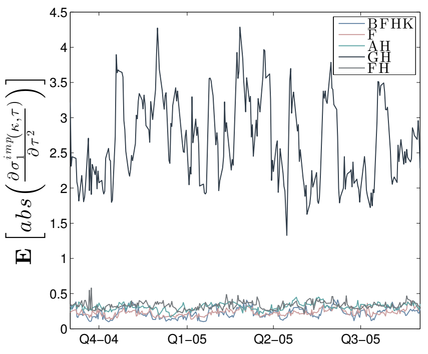
Figure 12: Sample mean of the absolute finite difference approximations of the 2nd order derivatives of the implied volatility surface.



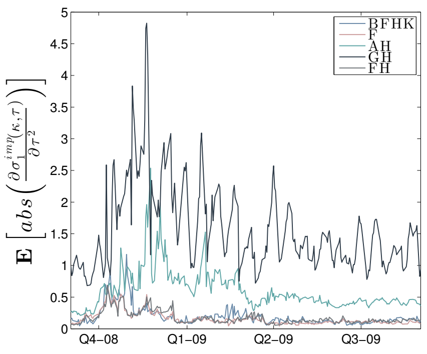
Sep'04-'05: Forward-moneyness.



Sep'08-'09: Forward-moneyness.



Sep'04-'05: Maturity.



Sep'08-'09: Maturity.

methods clearly stands out: Glaser & Heider (2012) and Fengler (2009). Glaser & Heider (2012) stands out as the mean of its 2nd order derivative clearly lie way higher than for the rest of the methods, thus it is by far the least *smooth* surface in this direction, in line with our previous observations. Fengler (2009) stands out with a stable and low mean value of its 2nd order derivative, and therefore must be considered the *smoothest*.

The rest of the methods lie around the same level, though the curve for Benko *et al.* (2007) are experiencing some spikes indicating *smoothness* problems for some of the dates in the period. This is, just as for Glaser & Heider (2012), in accordance with earlier observations.

(12b) The 2nd order derivatives for the period *Sep'08-'09* all lie at lower levels and thereby are *smoother* compared to the previous period. Hence, although still being the *smoothest*, Fengler (2009) is no longer superior.

Benko *et al.* (2007) and Glaser & Heider (2012) are, as expected, more well-behaved for this period especially Glaser & Heider (2012). For Benko *et al.* (2007) some spikes in the beginning of the period, corresponding to a few corner-spikes in the implied volatility surface, are visible.

(12c) This picture for the *smoothness* in the maturity direction resembles the graph for the forward-moneyness direction where Glaser & Heider (2012) has 2nd derivatives that significantly exceeds the level for the other methods. The level for the other methods lie collectively near a steady low level. Note that the scale on the y-axis for this dimension, is considerably lower than for the forward-moneyness direction.

(12d) Unlike the forward-moneyness direction, the 2nd order derivatives in the maturity-direction seem to increase in the 4th quarter of *Sep'08*. After which they decrease to a level a under that seen in the previous period. This is due to the structure of the implied volatility surface which has a steeper curve for short maturities during this quarter.

Here, the methods Glaser & Heider (2012) and Andreasen & Huge (2011) stand out as the least smooth methods, although Glaser & Heider (2012) is closer to the overall level than in the previous period.

Concluding: Observing the graphs of the fitted implied volatility functions over time, it could be seen that there were significant differences in their ability to fit the observed data and in the regularity of their surfaces. Benko *et al.* (2007), which calibrates the implied volatility surface directly, and Andreasen & Huge (2011), which interpolates the call data, gave the best fit to the volatility smile when this became pronounced for small maturities. While Fengler (2009) and Fengler & Hin (2013) were not able to capture these high implied volatility levels.

The regularity of the surfaces must generally be regarded as of erratic quality, as the corners of the surfaces seem prone to spikes or missing values for a majority of the methods. Especially for the local calibration method by Glaser & Heider (2012) which were experiencing severe problems for small maturities. This tendency was also evident when comparing the average level of the implied volatility surfaces in table 9.

The smoothness in Sep'0405 lies for all methods, except Glaser & Heider (2012), at a similar low level for both the forward-moneyness and the maturity. This picture becomes blurry for Sep'0809 where spikes also appeared for the other local calibration method Benko *et al.* (2007) for the forward-moneyness. The smoothest of the methods appear to be Fengler (2009), this held together with the lacking ability of capturing the volatility smile could indicate a method, which is too smooth to capture the structure and fluctuations in the market.

4.3.5 THE LOCAL VOLATILITY SURFACE

Assessing the local volatility differs from both the call- and the implied volatility surface, as there are no observed values to compare it up against. The local volatility surface are for all methods given by evaluating the local volatility function, (6) or (7), in the user-defined grid points. How the derivatives are determined do, however, differ and the reader is referred to the descriptions of the respective methods in sec. 3 for further details.

The enclosed animations of the local volatility surfaces across different dates for each of the two datasets are recorded from a different viewpoint compared to the animations for the implied volatility surfaces. Below a brief summary of the characteristics of the respective surfaces are given.

...the second derivative $\frac{\partial c^2}{\partial \kappa^2}$ becomes small for far-out-of-the-money and far-in-the-money options. While for far-out-of-the-money options the multiplication by κ^2 regulates the small second derivative, this effect is not so pronounced for far-in-the-money options and we expect numerical difficulties for these options.
(Glaser & Heider (2012))

- **SEP'04-'05:**
  - Benko *et al.* (2007) derive the local volatility surface from their fitted implied volatility surface. Thus, one would expect that the difficulties for the far OTM and the shortest maturity translates to this surface.

    Indeed, this region in the local volatility surface do seem to experience some difficulties which manifest itself as small spikes in the outermost corner. Also, for the opposite corner, far ITM, does there seem to be some irregularities materializing in occasional spikes.

    Other than this, the surface appears smooth with occasional *waves* or curvatures near the ATM level.

  - Fengler (2009) derives the local volatility surface from the call surface. The call surface does not exhibit any irregularities for any of the methods, including this one. Hence, there exists no problem areas which could be transferred to the local volatility surface through the local volatility function.

The derivatives featured in the local volatility function, (6), are given as output from the optimization procedure. Hence, theoretically there are no numerical instabilities present which could affect the surface.

As expected, the local volatility surface do indeed look very smooth, curvatures are only visible for a few of the dates in the period.

- Andreasen & Huge (2011) derive, as Fengler (2009), the local volatility surface from the surface of fitted call prices. Only for this method, there are no derivatives available, neither as output nor as an analytical expression. These must therefore be approximated using finite differences which introduced numerical inaccuracies.

  The local volatility surface is obviously far more angular than the others due to the finite difference approximation, and huge spikes appear frequently, obstructing parts of the surface.

  Deriving this surface for a coarser grid reduces the appearance of the spikes. This discovery indicates that the finite difference approximation is unstable for a step-size corresponding to the user-defined grid.

- Glaser & Heider (2012) derive the local volatility from the fitted call surface using the derivatives which - as for Fengler (2009) - are given by the output from the optimization procedure.

  This local volatility surface has more structure compared to Benko *et al.* (2007) and Fengler (2009) above, this can probably be attributed to the *local* element of the method. For the majority of the period the surface is relatively smooth and stable, though ,for a range of dates towards the end the period, irregularities and spikes appear far ITM for the short maturity. Glaser & Heider (2012) mention this in their paper, see the quote above, and suggest as a solution to bound the second order derivative below, by a small positive number $\varepsilon = 0.00025$.

- Fengler & Hin (2013) derive the local volatility from the call surface which has been approximated by a TP B-spline. Though, analytical expressions are available for the derivatives in this case, MatLabs built-in function for calculating the numerical derivatives of a B-spline have been chosen here, see sec. 3.5.4.

  When assessing the local volatility surface one immediately notice that this numerical approximation of the derivatives does not have the same negative impact as were the case for Andreasen & Huge (2011). This surface has the same *smoothness level* as Benko *et al.* (2007) and Fengler (2009), but is more *wavy* compared to the others. Towards the end of the period these waves become more like curls, especially around the ATM level. This is not a behavior seen for any of the other methods.

- **Sep'08-'09:**
  - Benko *et al.* (2007) have an implied volatility surface for this period which appears smooth and only rarely experiences spikes for the corner-regions. Hence, one would expect that the local volatility surface would be more stable for this data period.

    This does, however, not seem to be the case though these instabilities probably stem from other sources. For about half of the dates (evenly distributed across the period) the surface looks stabil and reasonably smooth. For the other half the surface experiences problems for OTM options, materializing as either the shape of the Sydney Opera House or as an unnaturally large downwards bend for far OTM options.
  - Fengler (2009) generates a local volatility surface which for this period, as for the previous, is stable and very smooth compared to some of the others. Small curvatures occurs in varying regions of the surface during this period. There are no spikes or other large irregularities to be spotted for this period neither.
  - Andreasen & Huge (2011) presented a local volatility surface for the previous period where *parts* of the surface were obstructed due to spikes. For this period this is not only a region, but rather the *entire* surface that is obstructed by spikes for the majority of the year.
  - Glaser & Heider (2012) presented an implied volatility surface which had been improved by the increased number of observations for this period of time. The same effect seems to be in evidence for the local volatility surface. This surface is stable during the entire period, even for shorter maturity levels does it only experience some *furrows* along the forward-moneyness dimension. These do become pronounced for some dates, but nothing severe.

- Fengler & Hin (2013) present a *wavy* local volatility surface as for the previous period. The waves or curls, parallel to the maturity dimension, become, for a significant number of dates, large in size and thereby seem more like irregularities than structures reflecting the market.

As briefly mentioned above, this is not in the same favorable situation, as for the implied volatility, where there exists a set of observations the surface levels can be compared to. Hence, it is not possible to assess whether or not these surfaces actually *hit the market*. Instead a thorough study of the local volatility levels can be conducted by making pair-wise comparisons of the mean levels, see table 10, and by observing the local volatility level across dates for each method individually, see figure 13.

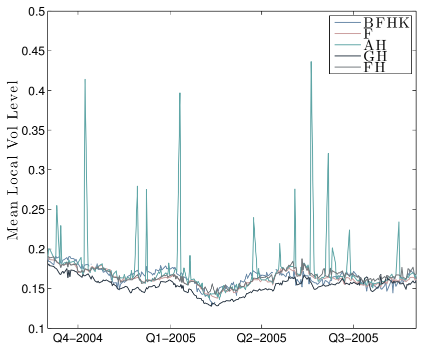
Table 10: Local Volatility: Sample mean and standard deviation of the absolute difference in pair-wise surface levels.

		F		AH		GH		FH	
	Sep	Mean	Std	Mean	Std	Mean	Std	Mean	Std
BFHK	’0405	0.0122	0.0169	0.0254	0.2887	0.0262	0.0319	0.0194	0.0223
	’0809	0.0209	0.0288	0.6223	3.6246	0.0300	0.0322	0.0310	0.0337
F	0405			0.0214	0.2886	0.0230	0.0279	0.0180	0.0184
	’0809			0.6176	3.6320	0.0171	0.0167	0.0211	0.0218
AH	’0405					0.0340	0.2909	0.0300	0.2882
	’0809					0.6237	3.6327	0.6255	3.6327
GH	’0405							0.0262	0.0260
	’0809							0.0271	0.0249

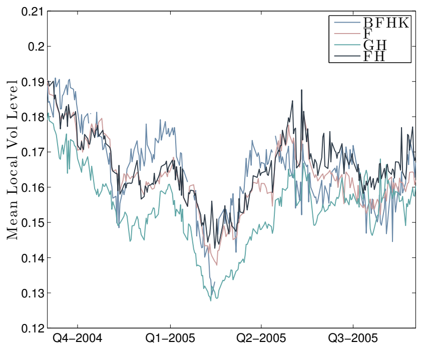
The pair-wise comparison in table 10 are given by the sample mean of the absolute differences in local volatility levels across dates. The general picture here, is that the difference in mean levels lie somewhere between 0.02 and 0.03 in *Sep’04-’05*, with a single exception given by Andreasen & Huge (2011) which will be discussed further below. For all comparisons, except those involving Andreasen & Huge (2011), the standard deviations also lies at a steady low level.

With only a few exceptions these mean and standard deviation levels for the difference rise for *Sep’08-’09* - opposed to what was seen for the implied volatility surface. One reason for this could be that the general local volatility level seems to be changing a lot over this period, see figure 13d, compared to what could be observed for the implied volatility. Hence, slight delays in a surface compared to the general level will result in a higher numerical value in this case.

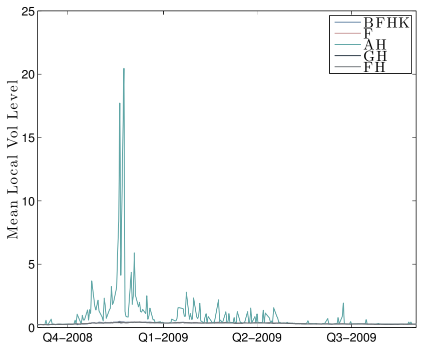
Now, lets get back to the *black sheep* - Andreasen & Huge (2011) -this method, which did not do very well in generating regular local volatility surfaces have a mean level matching the general picture for *Sep'04-'05*, but for *Sep'08-'09* this difference compared to the other surfaces takes a drastic jump upwards. Especially the standard deviation, which were high for *Sep '04-'05*, goes through the roof which is not a big surprise considering the animated local volatility surfaces. This deviation from the other surfaces is clearly illustrated in figures 13a and 13c.



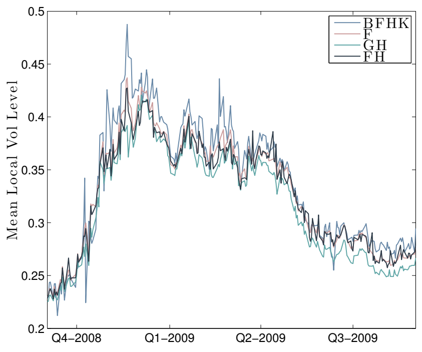
a *Sep'04-'05* - all methods.



b *Sep'04-'05* - all methods except Andreasen & Huge (2011).



c *Sep'08-'09* - all methods.



d *Sep'08-'09* - all methods except Andreasen & Huge (2011).

Figure 13: Sample mean of local volatility level.

The overall trend in the local volatility level can - as briefly mentioned - be found in figure 13. The spikes cause by Andreasen & Huge (2011) obstructs the notion of this trend and have therefore been removed in subfigure 13b and 13d. Here, it is evident that the market, and hence the mean local volatility level, is a lot more volatile in our *stress-testing* period, *Sep'08-'09*, than for our *baseline* period, *Sep'04-'05* as claimed in section 4.1.

The *distribution* of the standard deviation, corresponding to the mean levels seen in 13, are elaborated in figures 14 and 15. These plots display the sample mean of the local volatility for each date and method as a solid line with symmetrical vertical bars in each direction of length equal to the sample standard deviation for that given date.

In figure 14, depicting the majority of the methods, one can see that in *Sep'04-'05* the methods generally lie around a stable local volatility level with a stable standard deviation across time. This distribution seems to be a bit different for Glaser & Heider (2012), where the standard deviation *band* is more narrow, except for the end of the period where some oscillations are visible. This agrees with the observations made earlier when *eyeballing* the local volatility surface across time in the enclosed animated graphs.

The *band*-width of the standard deviations are more distinguished for the *stress-testing* period *Sep'08-'09*. For Benko *et al.* (2007) the first half is characterized by oscillating band-size, afterwards the width seems to settle at a stable level comparable to the previous period. Also, the distribution for Fengler (2009) changes by narrowing the band-width of standard deviations compared to the previous period. Hence, it seems that regardless of the market turbulence, this method experience an increased stability, explained by the increased number of observations available.

The distribution for Andreasen & Huge (2011) is not less interesting but appears on a different scale and have therefore been depicted by itself in figure 15. For *Sep'04-'05* notice that the overall stable look of the other methods also applies here, except for 14-16 troublesome dates. These occasional spikes for the standard deviation does, however, go trough the roof in *Sep'08-'09* and are especially persistent during the 4th quarter of 2008. This is also the period with the highest local volatility levels according to figure 13d, thus the market turbulence does seem to be a factor for this method contrary to the method by Fengler (2009).

Like for the implied volatility surfaces, the *smoothness* of the surfaces can be further quantified by looking at the second order derivatives - given by finite difference approximations equivalent to eqn. (45) - for both dimensions. The results are given in figures 16 and 17.

From figures 16a, 16c, 17a and 17c it is apparent that the *smoothness* is far worse

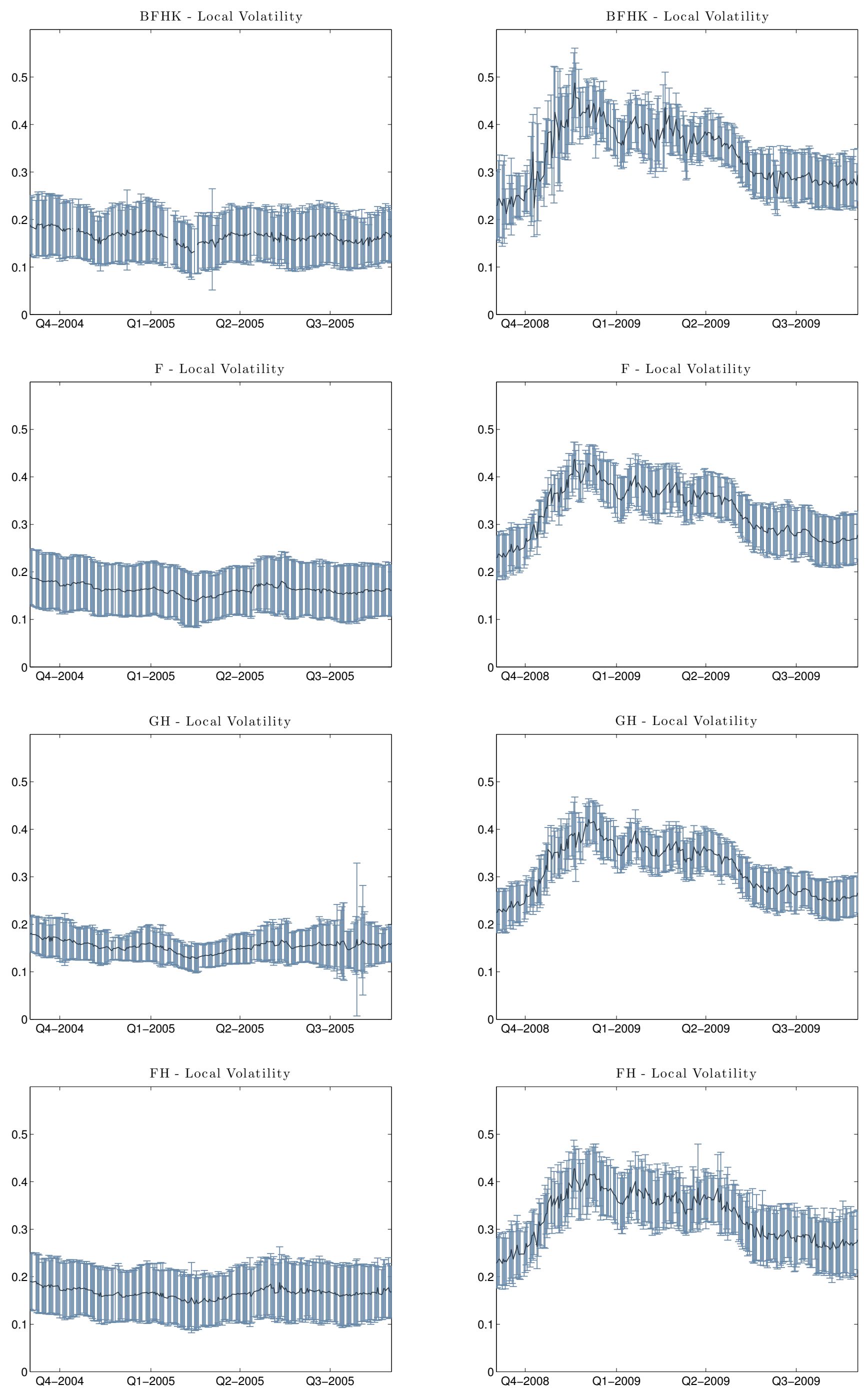


Figure 14: Sample mean of local volatility with symmetric error bars of total size 2 x sample standard deviations.

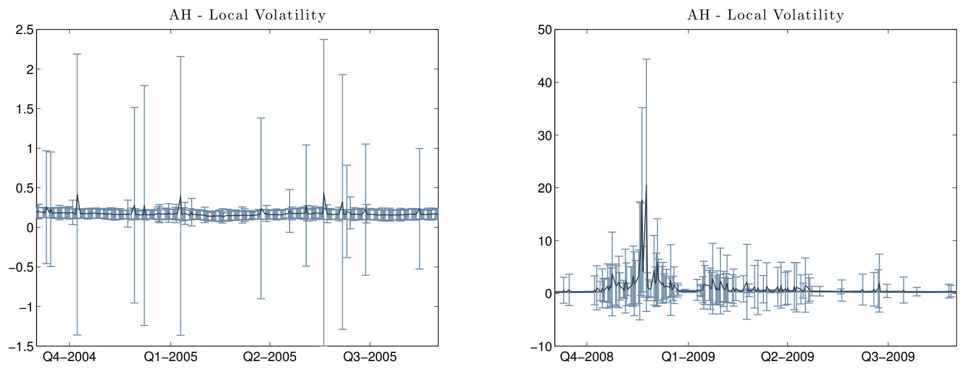


Figure 15: Sample mean of local volatility with symmetric error bars of total size 2 x sample standard deviations.

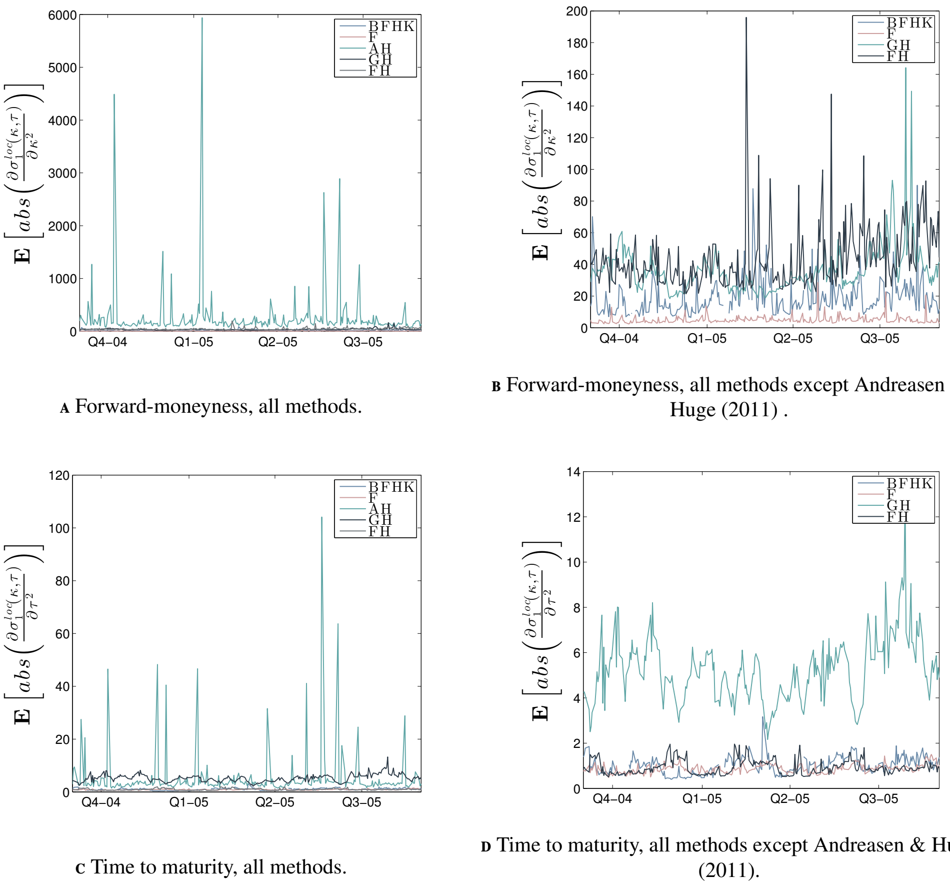
for Andreasen & Huge (2011) compared to the others, just as anticipated. It is can further be observed that, similar to the other methods, the *smoothness* is higher in the maturity direction than in the forward-moneyness direction. For period *Sep'08-'09* the concept *smoothness* is pretty much *non-existent*. Thus, like for figure 13, the method by Andreasen & Huge (2011) is removed in order to assess the other methods.

(16b) Like for the implied volatility surface, one here see that Fengler (2009) has the lowest values for the 2nd order derivative and therefore the highest *smoothness*-level. Benko *et al.* (2007) get second place, while the remaining two is a bit harder to differentiate though Fengler & Hin (2013) clearly obtains the highest and most frequent spikes. This is consistent with the observations made earlier regarding the waves, or curls, in the surface that became more distinct towards the end of the period.

Generally the *smoothness*-level seems stable for the first half of the period *Sep'04-'05* while spikes occur more or less often spikes in the second half for methods: Fengler & Hin (2013), Glaser & Heider (2012) and Benko *et al.* (2007).

(16d) The *smoothness*-levels are more homogeneous for the maturity dimension. Here, only one method, Glaser & Heider (2012), stands out by having a significantly higher 2nd order derivative, and thus lower level of *smoothness*. This is probably related to the *locality* of the method, creating a more structured surface as noted above.

Figure 16: *Sep'04-'05*: Sample means of the absolute value taken of the finite difference approximations of the 2nd order derivative wrt. forward-moneyness and maturity for the local volatility surface.



a Forward-moneyness, all methods.

b Forward-moneyness, all methods except Andreasen & Huge (2011) .

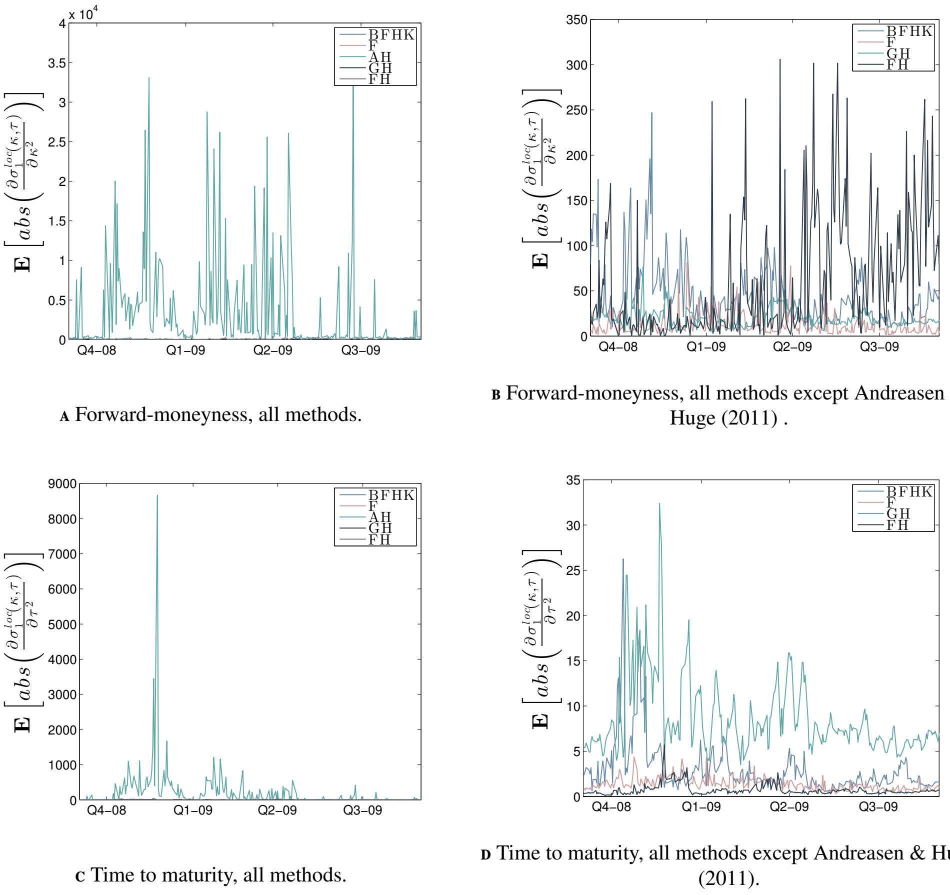
c Time to maturity, all methods.

d Time to maturity, all methods except Andreasen & Huge (2011).

(17b) The general trend for *Sep'08-'09* is more diffuse than for the previous period. Here, none of the methods seem to be doing significantly better than the others. Fengler & Hin (2013) still seem to be having the lowest level of *smoothness* and for the first three quarters of '09 it experiences some large spikes.Benko *et al.* (2007) on the other hand experiences spikes for the 4th quarter of *'08*, though these are not as large in size. Glaser & Heider (2012) and Fengler (2009) both lie at fluctuating levels in the bottom of the graph and hence are the *smoothest* of the methods assessed.

This general trend of fluctuating levels can for this surface probably, in part, be attributed to the fluctuating overall level for the local volatility surface ifself, see subfigure (13d).

Figure 17: Sep0809: Sample means of the absolute value taken of the finite difference approximations of the 2nd order derivative wrt. forward-moneyness and maturity for the local volatility surface.



a Forward-moneyness, all methods.

b Forward-moneyness, all methods except Andreasen & Huge (2011) .

c Time to maturity, all methods.

d Time to maturity, all methods except Andreasen & Huge (2011).

(17d) The increase in fluctuations also translates to the 2nd order derivatives for the maturity direction. Though Glaser & Heider (2012) still has the overall lowest *smoothness* levels, one can see that the difficulties for Benko *et al.* (2007) in the 4th quarter *'08* stands out here as well - recall that this is accordance with our *eyeball* observations above. Fengler & Hin (2013) as well as Fengler (2009) obtains the smallest derivatives - thus highest *smoothness* - which still lie at a higher level than for the *baseline* year.

Concluding: Looking at the enclosed 3D graphs of the local volatility surfaces it becomes evident that the majority of the methods are experiencing regularity problems in the corner regions where spikes and missing data appears. These irregularities become more pronounced when fitted to the data for the *stress-testing*-

period. The surface by Andreasen & Huge (2011) is by far the worst and for the latter period as good as useless. As in the case for the implied volatility surface, Fengler (2009) presents a nice smooth surface without any irregularities. Also, Fengler & Hin (2013) produced a local volatility surface without spikes or missing values, but an irregular pattern of waves and curls that become particular pronounced for Sep'0809.

Comparing the mean levels of these local volatility surfaces - after removing Andreasen & Huge (2011) - it appears that they all lie at similar levels and thus generally produce reliable surface values. Taking a closer look at the mean levels compared to the corresponding standard deviations for each method, it appears that Fengler (2009) along with Glaser & Heider (2012) is the most stable method in this respect.

The smoothness for this surface is, opposed to the implied volatility surface, experiencing spikes for both Fengler & Hin (2013) and Glaser & Heider (2012) in Sep'0405 for the forward-moneyness which becomes more pronounced in Sep'0809. While for the maturity only Glaser & Heider (2012) deviates significantly from the general level. Hence, it is evident that a great part of the methods are challenged with respect to smoothness in terms of the local volatility surface. Here it must be taken into considerations that a lot of the tricky areas has been filtered out during the data refinement.

4.3.6 Stability across time

The robustness across time can be quantified by testing how the local volatility function fitted for a given date performs if used X days later for pricing the surface of call options. If the computed surface deviates heavily from the call surface fitted from the data at this specific date, this would indicate that the local volatility function is changing rapidly over time. Such an instability in the local voltility function across time is not desirable, as it often leads to mispricings according to practitioners.

The exact procedure for this *out-of-sample* testing is described in algorithm 13. Here, Monte Carlo evaluation have been used, but it could just as well have been solved using a finite difference solver applied to the PDE for the European call price.

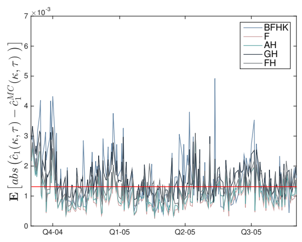
The mean of the absolute difference between the call surface computed with the $\{1, 5, 20\}$-day old local volatility function and the call surface fitted directly to the observed quotes are for all methods given in figure 18.

Algorithm 13: Monte carlo simulation of European call price in the local volatility model.

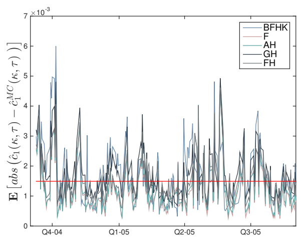
```
input: $s$                                                        ▹ initial price underlying,
       $t_i, t_{i+1}$                                             ▹ daily observation dates,
       $\sigma_1^{loc}(\hat{\kappa}_m, \hat{\tau}_l; t_i)$, $m=1,\dots,M, l=1,\dots,L$   ▹ local volatility surface,
       $\hat{c}_1(\hat{\kappa}_m, \hat{\tau}_l; t_{i+1})$, $m=1,\dots,M, l=1,\dots,L$   ▹ surface of call prices,
       $N_{MC}$                                                   ▹ number of simulations paths.

for $sim = 1$ to $N_{MC}$ do
    Set: $S^{prev} = s$;   $\hat{\tau}^{prev} = 0$.

    for $l = 1$ to $L$ do
        Set: $\Delta_\tau = \hat{\tau}_l - \hat{\tau}^{prev}$.

        if $S^{prev} \notin [\hat{\kappa}_1; \hat{\kappa}_M]$ then
            PrintOut('Not optimal')

        $\hat{\kappa}_m$=FindClosest($S^{prev}$, $\hat{\boldsymbol{\kappa}}$),
        $\sigma = \sigma_1^{loc}(\hat{\kappa}_m, \hat{\tau}_l; t_i)$.

        $Z \sim \mathcal{N}(0, 1)$,
        $S = S^{prev} \exp\left(\frac{1}{2}\sigma^2\Delta_\tau + \sigma\sqrt{\Delta_\tau}Z\right)$.

        $\text{Payoff}_{sim}(\hat{\boldsymbol{\kappa}}, \hat{\tau}_l) = \max(S - \hat{\boldsymbol{\kappa}}, 0)$.

        Set: $S^{prev} = S$;   $\hat{\tau}^{prev} = \hat{\tau}_l$.

$\hat{c}_1^{MC}(\hat{\boldsymbol{\kappa}}, \hat{\boldsymbol{\tau}}; t_{i+1}) = \frac{1}{N_{MC}} \sum_{sim=1}^{N_{MC}} \text{Payoff}_{sim}(\hat{\boldsymbol{\kappa}}, \hat{\boldsymbol{\tau}})$.

Diff_measure= $\mathbb{E}\left[\text{abs}\left(\hat{c}_1(\hat{\boldsymbol{\kappa}}, \hat{\boldsymbol{\tau}}; t_{i+1}) - \hat{c}_1^{MC}(\hat{\boldsymbol{\kappa}}, \hat{\boldsymbol{\tau}}; t_{i+1})\right)\right]$.
```

In figure 13b one could see that the general level for the local volatility surface was quite stable across Sep'0405. The absolute error in figures 18a-18c are small enough (compared to the call prices) to indicate that the local volatility surfaces

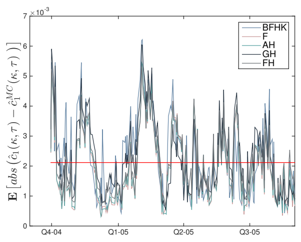
Figure 18: Sample mean of absolute difference in the surfaces calibrated and simulated, respectively.



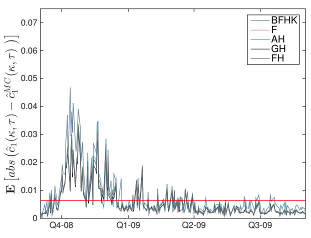
A 1d. lag - Sep'0405.



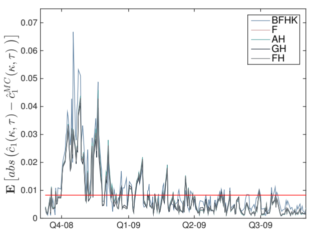
B 5d. lag - Sep'0405.



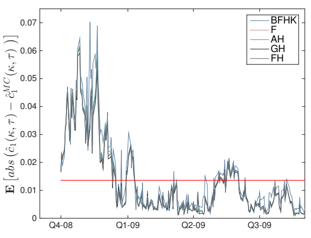
C 20d. lag - Sep'0405.



D 1d. lag - Sep'0809.



E 5d. lag - Sep'0809.



F 20d. lag - Sep'0809.

fitted for this period can be declared stable across time.

Likewise, one could see in figure 13d that the overall level for the local volatility surface was fluctuating across the period Sep'0809 and especially for the last quarter of '08 it experienced a steep increase. Hence, the quite large errors for this quarter it is seen in figures 18d-18f when increasing the lag from 1 to 20 are anticipated. The errors do, however, stabilize for the remaining part of the period.

This *out-of-sample* testing does therefore not seem to distinguish any of the methods. Based on the periods with a stable local volatility level, it can therefore be concluded that all methods seem suitably stable across time.

5 Did we reach our goals?

Initially the investigations carried out in this paper were motivated with the desire to examine whether their performance are as suggested by the respective authors. A number of criteria were therefore outlined according to which the methods were evaluated. Below is a summary of the results obtained through the empirical experiments for each of these criteria and at the end it is concluded whether an *optimal* method exists among the participants.

- ▷ **Accuracy:** The *goodness of fit* have been measured by the relative distance from the observed call prices to the corresponding grid-points on the calibrated surface. These numbers revealed that the best fit is obtained by Andreasen & Huge (2011) which was anticipated, as this is the only interpolation method. In the other end of the scale, with the worst fit, is the method by Fengler & Hin (2013) which is the only global approximation method.

  The calibrated implied volatility surfaces' *goodness of fit* to the observed quotes have also been eyeballed. Here, it appeared that the methods could be divided in two: those who were able to capture the volatility smile for the short maturities and those who were not. Benko *et al.* (2007) and Andreasen & Huge (2011) were as the only two methods able to fit the smile for dates when this became pronounced. This could be attributed to the fact that Benko *et al.* (2007) calibrates the implied volatility surface directly and Andreasen & Huge (2011) is an interpolation method.

- ▷ **Smoothness:** The smoothness level has been quantified in each direction by a finite difference approximation of the 2nd order derivative.

  For the implied volatility surface it was noted how Fengler (2009) was the smoothest method in forward-moneyness, while Glaser & Heider (2012) was the least smooth method for Sep'0405 in both directions. The general *smoothness*-levels in Sep'0809 seemed more aligned across the various methods.

  For the local volatility surface it was noted how Andreasen & Huge (2011) had severe problems in fitting the surface and this was especially evident in its *smoothness*-curve which lay at a sky-high level compared to the other methods. For this surface the smoothness levels fluctuates across time, this trend seems particularly spikey for Fengler & Hin (2013) in forward-moneyness while Glaser & Heider (2012) experience the least smoothness for maturity.

One should note that this smoothness measure does not capture the effect of missing values in the surface. *Eyeballing* of the respective surfaces reveals that especially for the local volatility surfaces this is a significant issue for the corner regions for all methods except Fengler (2009) and Fengler & Hin (2013). For the local calibration methods Benko *et al.* (2007) and Glaser & Heider (2012) it is evident that these corner areas coincides with areas of the surface where no input data are available.

▷ **SPEED:** The speed is measured as the calibration time for a single date. It was noticed how Benko *et al.* (2007) has a relative speed compared to the others which is inexcusably high. Fengler (2009) on the other hand spend half as much computational time on average compared with the other methods.

▷ **ROBUSTNESS:** Robustness was defined as the ability to cope with changing input data and stability across time. The robustness with respect to changing input data was hard to isolate due to other factors, such as the number of input quotes which also played a role. For the local volatility surface there were a significant trend for all the methods consisting of increased difficulty with fitting the local volatility surface in the *stress-testing*-period compared to the *basis*-period. Though only Andreasen & Huge (2011) seemed to become unusable as a consequence hereof.

The stability across time was quantified by the *out-of-sample* tests conducted using Monte Carlo simulation. Here it was observed that a lag corresponding to 20 business days only resulted in a small absolute error for all the methods. Thus the local volatility surfaces all seem stable across time.

Our final verdict over methods are therefore as follows:

- Benko *et al.* (2007) - too slow to be used in practice,
- Fengler (2009) - too smooth to capture the market structure,
- Andreasen & Huge (2011) - the local volatility surface is useless - at least in this setup,
- Glaser & Heider (2012) - large parts of the surface is obstructed due to the local nature of the method,
- Fengler & Hin (2013) - bad fit to data and an irregular structure of the local volatility surface.

We can from this conclude that the ultimate method is yet to come...

A Appendix

A.1 Glaser & Heider quadratic programming problem.

$$
\begin{aligned}
&\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2} \left\| \Phi^{1/2} \left(A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \mathbf{b} \right) \right\|_2^2 \\
= &\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2} \left\| \Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \Phi^{1/2}\mathbf{b} \right\|_2^2 \\
= &\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2} \left(\Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \Phi^{1/2}\mathbf{b} \right)^\intercal \left(\Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \Phi^{1/2}\mathbf{b} \right) \\
= &\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2} \left(\left(\Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} \right)^\intercal \Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \left(\Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} \right)^\intercal \Phi^{1/2}\mathbf{b} - \left(\Phi^{1/2}\mathbf{b} \right)^\intercal \Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} + \left(\Phi^{1/2}\mathbf{b} \right)^\intercal \Phi^{1/2}\mathbf{b} \right) \\
= &\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2} \left(\left(\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}^\intercal A^\intercal \Phi^{1/2\intercal} \right) \Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \left(\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}^\intercal A^\intercal \Phi^{1/2\intercal} \right) \Phi^{1/2}\mathbf{b} - \left(\mathbf{b}^\intercal \Phi^{1/2\intercal} \right) \Phi^{1/2} A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} + \left(\mathbf{b}^\intercal \Phi^{1/2\intercal} \right) \Phi^{1/2}\mathbf{b} \right) \\
= &\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2} \left(\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}^\intercal A^\intercal \Phi A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}^\intercal A^\intercal \Phi\mathbf{b} - \mathbf{b}^\intercal \Phi A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} + \mathbf{b}^\intercal \Phi\mathbf{b} \right) \\
\overset{*}{=} &\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2} \left(\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}^\intercal A^\intercal \Phi A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}^\intercal A^\intercal \Phi\mathbf{b} - \mathbf{b}^\intercal \Phi A\boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} \right)
\end{aligned}
$$

$*$: constant term removed.
This can be solved through the quadratic program:

$$
\min_{\alpha_{\hat{\kappa},\hat{\tau}}} \frac{1}{2}\, \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}^\intercal \left(A^\intercal \Phi A \right) \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}} - \left(A^\intercal \Phi \mathbf{b} \right)^\intercal \boldsymbol{\alpha}_{\hat{\kappa},\hat{\tau}}.
$$

A.2 Fengler & Hin linear inequality constraints.

• Convexity: $\frac{\partial^2 z}{\partial \kappa^2}(\kappa, \tau) \geq 0$:

$$
\begin{aligned}
&\frac{\theta_{j_1+1,j_2} - \theta_{j_1,j_2}}{\xi^*_{j_1+1} - \xi^*_{j_1}} \leq \frac{\theta_{j_1+2,j_2} - \theta_{j_1+1,j_2}}{\xi^*_{j_1+2} - \xi^*_{j_1+1}} \qquad j_1 = 0, \ldots, q_1 - 2,\ j_2 = 0, \ldots, q_2, \\
\Leftrightarrow &\frac{\theta_{j_1+1,j_2}}{\xi^*_{j_1+1} - \xi^*_{j_1}} - \frac{\theta_{j_1,j_2}}{\xi^*_{j_1+1} - \xi^*_{j_1}} - \frac{\theta_{j_1+2,j_2}}{\xi^*_{j_1+2} - \xi^*_{j_1+1}} + \frac{\theta_{j_1+1,j_2}}{\xi^*_{j_1+2} - \xi^*_{j_1+1}} \leq 0 \\
\Leftrightarrow &\mathbf{C}^{(1)} \cdot \boldsymbol{\theta} \leq \mathbf{0}, \quad \text{for } \mathbf{C}^{(1)} = \left[\mathbf{C}^{(1)}_{j_1} \right]_{j_1=0,\ldots q_1-2}. \qquad (46)
\end{aligned}
$$

Using notation: $\Delta\xi_i^* = \xi_i^* - \xi_{i-1}^*$, the $\mathbf{C}_{j_1}^{(1)}$'s are defined as:

$\mathbf{C}_{j_1}^{(1)} =$

$$\begin{array}{c|ccccccccc|}
 & j_1,0 & \cdots & j_1,q_2 & j_1+1,0 & \cdots & j_1+1,q_2 & j_1+2,0 & \cdots & j_1+2,q_2 \\
j_1,0 & \frac{-1}{\Delta\xi_{j_1+1}^*} & 0 & 0 & \frac{1}{\Delta\xi_{j_1+1}^*}+\frac{1}{\Delta\xi_{j_1+2}^*} & 0 & 0 & \frac{-1}{\Delta\xi_{j_1+2}^*} & 0 & 0 \\
\vdots & 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\
j_1,q_2 & 0 & 0 & \frac{-1}{\Delta\xi_{j_1+1}^*} & 0 & 0 & \frac{1}{\Delta\xi_{j_1+1}^*}+\frac{1}{\Delta\xi_{j_1+2}^*} & 0 & 0 & \frac{-1}{\Delta\xi_{j_1+2}^*}
\end{array}$$

• Lower bounds on the first derivative: $-1 \le \frac{\partial z}{\partial \kappa}(\kappa,\tau)$:

$$\begin{aligned}
& -1 \le \frac{\theta_{j_1+1,j_2} - \theta_{j_1,j_2}}{\xi_{j_1+1}^* - \xi_{j_1}^*}, \qquad j_1 = 0\ldots,q_1-2,\ j_2 = 0,\ldots,q_2, \\
\overset{*}{\Leftrightarrow} & -1 \le \frac{\theta_{1,j_2} - \theta_{0,j_2}}{\xi_1^* - \xi_0^*}, \qquad j_2 = 0,\ldots,q_2, \\
\Leftrightarrow & \mathbf{C}^{(2)} \cdot \theta \le \mathbf{1}
\end{aligned}$$

where

$$\mathbf{C}^{(2)} = \begin{array}{c|ccccccccc|}
 & 0,0 & \cdots & 0,q_2 & 1,0 & \cdots & 1,q_2 & 2,0 & \cdots & q_1,q_2 \\
0 & \frac{1}{\Delta\xi_1^*} & 0 & 0 & \frac{-1}{\Delta\xi_1^*} & 0 & 0 & 0 & 0 & 0 \\
\vdots & 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\
q_2 & 0 & 0 & \frac{1}{\Delta\xi_1^*} & 0 & 0 & \frac{-1}{\Delta\xi_1^*} & 0 & 0 & 0
\end{array}$$

* : Due to the convexity constraint in eqn. (46).

• Upper bounds on the first derivative: $\frac{\partial z}{\partial \kappa}(\kappa,\tau) \le 0$:

$$\begin{aligned}
& \frac{\theta_{j_1+2,j_2} - \theta_{j_1+1,j_2}}{\xi_{j_1+2}^* - \xi_{j_1+1}^*} \le 0, \qquad j_1 = 0,\ldots,q_1-2,\ j_2 = 0,\ldots,q_2, \\
\overset{*}{\Leftrightarrow} & \frac{\theta_{q_1,j_2} - \theta_{q_1-1,j_2}}{\xi_{q_1}^* - \xi_{q_1-1}^*} \le 0, \qquad j_2 = 0,\ldots,q_2, \\
\Leftrightarrow & \mathbf{C}^{(3)} \cdot \theta \le \mathbf{0}
\end{aligned}$$

where

$$\mathbf{C}^{(3)} = \begin{array}{c|ccccccccc|}
 & 0,0 & \cdots & q_1-2,q_2 & q_1-1,0 & \cdots & q_1-1,q_2 & q_1,0 & \cdots & q_1,q_2 \\
0 & 0 & 0 & 0 & \frac{-1}{\Delta\xi_{q_1}^*} & 0 & 0 & \frac{1}{\Delta\xi_{q_1}^*} & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
q_2 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{\Delta\xi_{q_1}^*} & 0 & 0 & \frac{1}{\Delta\xi_{q_1}^*}
\end{array}$$

* : Due to the convexity constraint in eqn. (46).

• Monotonicity: $z(\kappa, \tau_2) \geq z(\kappa, \tau_1)$, for $\tau_2 \geq \tau_1$:

$$\begin{aligned} &\theta_{j_1,j_2+1} - \theta_{j_1,j_2} \geq 0, \qquad j_1 = 0,\ldots,q_1,\ j_2 = 0,\ldots,q_2-1, \\ \Leftrightarrow &\mathbf{C^{(4)}} \cdot \boldsymbol{\theta} \leq \mathbf{0}, \quad \text{for } \mathbf{C}^{(4)} = \left[\mathbf{C}^{(4)}_{j_1}\right]_{j_1=0,\ldots q_1} \end{aligned}$$

where

$$\mathbf{C}^{(4)}_{j_1} = \begin{array}{c} \\ {\scriptstyle j_1,0} \\ {\scriptstyle j_1,1} \\ \vdots \\ {\scriptstyle j_1,q_2-1} \end{array} \begin{array}{c} \begin{array}{ccccc} {\scriptstyle j_1,0} & {\scriptstyle j_1,1} & {\scriptstyle \cdots} & {\scriptstyle j_1,q_2-1} & {\scriptstyle j_1,q_2} \end{array} \\ \begin{bmatrix} 1 & -1 & \cdots & 0 & 0 \\ 0 & 1 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -1 \end{bmatrix} \end{array}.$$

Bibliography

Andreasen, Jesper and Huge, Brian (2011). Volatility interpolation. *Risk Magazine*, March, 76–79.

Benko, M. and Fengler, M. and Härdle, W. and Kopa, M. (2007). On extracting information implied in options. *Computational Statistics*, 22(4), 543–553.

Björk, Tomas (2009). *Arbitrage Theory in Continuous time*. Oxford University Press, 3rd edition.

Carr, Peter and Madan, Dilip B. (2005). A note on sufficient conditions for no arbitrage. *Finance Research Letters*, 2, 125–130.

Coleman, Thomas F. and Li, Yuying and Verma, Arun (1999). Reconstructing the unknown volatility function. *Journal of Computational Finance*, 3(2), 77–102.

Constantinides, G. M. and Jackwerth, J. C. and Savov, A. (2013). The Puzzle of Index Option Returns. *Review of Asset Pricing Studies*, 3(2), 229–257.

Dupire, Bruno (1994). Pricing with a Smile. *Risk Magazine*, 7(1), 1–10.

Fengler, Matthias R (2009). Arbitrage-Free Smoothing of the Implied Volatility Surface. *Quantitative Finance*, 9(4), 417–428.

Fengler, Matthias R and Hin, Lin-yee (2013). Semi-nonparametric estimation of the call price surface under strike and time-to-expiry no-arbitrage constraints.

Gatheral, Jim (2006). *The Volatility Surface - a practitioner's guide*. John Wiley & Sons, 1st edition.

Gatheral, Jim and Jacquier, Antoine (2014). Arbitrage-free SVI volatility surfaces. *Quantitative Finance*, 14(1), 59–71.

Glaser, Judith and Heider, Pascal (2012). Arbitrage-free approximation of call price surfaces and input data risk. *Quantitative Finance*, 12(1), 61–73.

Gope, Pijush and Fries, C (2011). Arbitrage-free Asset Class Independent Volatility Surface Interpolation on Probability Space using Normed Call Prices.

Green, Peter J. and Silverman, Bernard W. (1993). *Nonparametric Regression and Generalized Linear Models: A roughness penalty approach*. Taylor & Francis Ltd.

Hentschel, Ludger (2003). Errors in Implied Volatility Estimation. *The Journal of Financial and Quantitative Analysis*, 38(4), 779.

Kahalé, Nabil (2004). An arbitrage-free interpolation of volatilities. *Risk Magazine*, 17(5), 102–106.

Maruhn, Jan (2013). Techniques for Instantaneous Arbitrage-Free Fitting of Bid & Ask Quotes. Conference: Global Derivatives Trading & Risk Management - Amsterdam.

Rasmussen, Lykke (2012). Calibrating the Local Volatility Model - an implicit approach. Master's thesis, University of Copenhagen.

Roper, Michael (2010). *Arbitrage free implied volatility surfaces*. PhD thesis, The University of Sydney.

Paper II

Efficient Calculation of Sentivitities

1 Motivation

One of the major challenges in todays post-crisis environment is calculating the sensitivities of various complex products for hedging and risk management. For instance describes Savickas (2011) how *financial institutions need frequent assessment of a substantial set of sensitivities to manage the complex nature of CVA*, while Davidson (2015) explains how *calculating a products' sensitivity to a huge variety of risk factors, commonly known as Greeks, is placing an ever-increasing strain on the modelling and computational demands of issuers' pricing and risk models*.

These sensitivities have historically been determined using finite difference approximation - in this context better known as *bumping* - for which the original calculation is repeated multiple times to determine each sensitivity. However, in the current climate, the amount of risk factors and the computational cost for the original valuations, are in practice making it impossible to determine all sensitivities within a reasonable amount of time.

The financial industry is therefore starving for alternative approaches and has increasingly been replacing bumping with *algorithmic differentiation*. This method provides machine precision accuracy and a significant reduction in computational time. An example of this is the CVA case presented in this paper for which the speedup turns out to be in the range of 70-90% compared to bumping. This technique have been widely used within engineering disciplines for the last two decades, but was first introduced to the finance community in 2006 with the award winning article *Smoking Adjoints* by Giles & Glasserman.

Depending on the context, speed might not be the only objective worth considering when looking for alternatives to bumping. Development costs, for instance, are rather high for algorithmic differentiation. Under some circumstances it might therefore be reasonable to consider alternatives which offer significantly lower development costs in return for smaller speed gains. These objectives are fulfilled by the *complex-step derivative approximation (CSDA)* that also, like the algorithmic method, gives derivatives of machine precision. The author became acquainted with this method in connection with setting up a validation framework for the algorithmic implementation. But, as it will be argued later, this method turned out to be a genuine alternative for bumping as well.

The methods: bumping, CSDA and algorithmic differentiation, have not yet been compared in a formalized framework in academia. This paper intends to fill this gap by providing a thorough and transparent comparison of the alternatives for deriving a set of sensitivities. This includes a validation framework which can be used in practice to guarantee the correctness of the algorithmic implementation.

Evaluating the sensitivities of the CVA price is a case for which tremendous speedups can be obtained using algorithmic differentiation. Several larger international banks have already implemented the method for this area according to Davidson (2015). Thus, in order to assess the power of algorithmic differentiation, the comparison presented in this paper is based on sensitivity calculations for the CVA price of an 10Y20Y receiver interest rate swap entered with Adidas AG as counterparty. The CVA calculation can be implemented with various levels of sophistication, but for this academic exercise the complexity is kept at a minimum to keep the framework transparent and the focus on the methods compared.

Quantification of the methods is given in terms of accuracy- and run times measurements. Reports of numbers like these given in the current financial literature does frankly seem like the wild west, especially when it comes to speedups. Unless the reader is provided a description of how the results have been obtained, these are in the authors opinion not of much practical use.

The results presented here are therefore accompanied by hands-on details of the implementation. The *size* of the calculation are documented by the enclosed source code, as the machine precision accuracy depends on the number of floating point operations performed. Information on how the actual run time has been measured is provided, as this can have a large impact on the speedup results. And the memory management has been described, as memory consumption too can have a significant impact on the performance, especially for the algorithmic method. Furthermore, does this documentation hopefully give a sense of the development costs connected to each of the methods assessed.

2 Financial Framework

Counterparty credit risk is the risk associated with OTC derivatives which can be assigned to the possible default of a counterparty and the associated losses. Here, the default event occurs at a random time τ and is defined as the event where the given counterparty "...cannot face its obligation on the payments owed to some entity", see Brigo *et al.* (2013, p.47).

The price of counterparty credit risk is called Credit Value Adjustment (CVA) which intuitively can be described as below by Brigo *et al.*:

> ...the reduction in price an investor requires in order to trade a product with a default-risky counterparty as opposed to a default-free one, with which the investor would pay full price.
>
> (Brigo *et al.* (2013))

In the case considered here, one of the two counterparties entering the financial contract is assumed default-free while the other is assumed to have a positive probability of default. This assumption of a default free counterparty was widely accepted before the financial crisis '07 were the majority of financial institutions where perceived as *too-big-to-fail*. Although this assumption no longer is realistic, see Brigo *et al.* (2013, Remark 4.1.4), it is used here - along with the list of simplifications given below - in order to keep the theoretical framework as transparent as possible.

The complete list of simplifications are:

- ▷ One default-free and one default-risky counterparty. The corresponding CVA of the financial contract is thus unilateral.
- ▷ The exposure is independent of counterparty credit quality. The wrong- or right way risk is therefore not accounted for.
- ▷ The exposure is assessed at contract level, netting is therefore not applied as it only applies to counterparty-level portfolios.
- ▷ Collateral is not taken into account and neither are the funding costs.

As indicated by this list can the calculation of CVA be done with increasing levels of sophistication and associated complexity as described in Brigo *et al.* (2013). In this paper the focus is on implementing the calculation of sensitivities using the four methods presented below and the extensions is therefore left for future work.

Counterparty credit risk management requires evaluation of the sensitivities:

$$\frac{\partial CVA(t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

where $\boldsymbol{\theta}$ is a vector of market parameters. Currently, the most time consuming sensitivities to calculate are for $\boldsymbol{\theta}$ given by the points on the zero coupon yield curve. This curve not only affect the exposure calculation but also the initial calibration of the short rate- and intensity models. Today these sensitivities are derived using bumping, or bump-and-revalue, which require two additional calculations for each point on the curve. The framework presented here describes alternative solutions for calculating these derivatives using complex-step derivative approximation and algorithmic differentiation.

The financial product and corresponding models used for the CVA valuation has been broadly adapted from Hansen & Glibstrup (2014) with further details and adjustments added whenever needed. In the article by Hansen & Glibstrup the CVA price is calculated across time, but in this paper will the valuation be limited to the present time in order to keep the complexity at a minimum.

Interest rate swaps constitute the majority of instruments to which counterparty risk pricing is typically applied.
(Brigo *et al.* (2013))

2.1 CVA approximation formula.

Sources: Brigo *et al.* (2013).

The value of a financial contract between two counterparties is in the unilateral case calculated from the point of view of the default-free party. Let therefore $\Pi(t, T)$ be the net cash-flows of the contract between time t and expiry of the contract T, as seen from the default-free party. The corresponding *exposure* at time t is the amount the default-free party stands to loose in the event of a default with zero recovery:

$$Ex(t) = \left(\mathbb{E}_t^{\mathbb{Q}}\left[\Pi(t, T)\right]\right)^+ \tag{1}$$

which is the positive part of the market value at time t calculated under a risk-neutral $\mathbb{Q}$-measure.

The CVA price of the contract can then be written in terms of the exposure at the time of default, τ, under the assumptions listed in the previous section:

$$CVA_0 = (1 - \mathrm{REC})\mathbb{Q}_0\{\tau < T\}\,\mathbb{E}_0^{\mathbb{Q}}\left[D(0,\tau)Ex(\tau)\right]. \tag{2}$$

Here, $D(0,\tau)$ is the default-free stochastic discount factor at time 0 for the maturity τ and $(1 - \mathrm{REC})$ is the *loss given default rate* which is a constant fraction of the exposure that will be lost in the event of default.

This price can for practical purposes be approximated by *bucketing* the time of default, τ, into a set of intervals:

$$(0 = t_0, t_1], (t_1, t_2], \ldots, (t_{n-2}, t_{n-1}], (t_{n-1}, t_n = T]. \tag{3}$$

The time of default, τ, in eqn. (2) is then replaced by t_i whenever $\tau \in (t_{i-1}, t_i]$ which gives the *bucketed* approximation of the CVA price:

$$CVA_0 = (1 - \mathrm{REC})\sum_{i=1}^{n}\mathbb{Q}_0\{\tau \in (t_{i-1}, t_i]\}\,\mathbb{E}_0^{\mathbb{Q}}\left[D(0,t_i)Ex(t_i)\right]. \tag{4}$$

In the case presented here, the time buckets are of equal length: $t_i - t_{i-1} = \delta^\tau$, and each corresponds to a quarter: $\delta^\tau = 0.25$.

2.1.1 The model framework

The evaluation of the CVA approximation given in eqn. (4) has, as previously mentioned, been widely adopted from Hansen & Glibstrup (2014). There are two main components in the formula that must be modeled:

- $\mathbb{Q}_0\{\tau \in (t_{i-1}, t_i]\}$: the probability of the counterparty's default in time bucket $(t_{i-1}, t_i]$. The default of the counterparty is here modeled by the CIR intensity model.
- $\mathbb{E}_0^{\mathbb{Q}}\left[D(0,t_i)Ex(t_i)\right]$: the time $t = 0$ value of the expected exposure. The financial contract here is an interest rate swap which is evaluated using a Hull-White short rate model.

2.2 Interest Rate Swap

Sources: Hansen & Glibstrup (2014), Linderstroem (2013), Munk (2011).

The financial contract under investigation is the forward starting interest rate swap specified in Hansen & Glibstrup (2014). The interest rate swap is an ATM 10Y20Y receiver which the default-free party have entered with 'Adidas AG' as the default-risky counterparty.
The interest rate swap (IRS) contract is characterized by:

- ▷ *Position: Receiver.* The position is given relative to the fixed leg, the default-free party will thus receive a fixed rate and pay a floating rate.
- ▷ $N = 100.0$. The notional for the interest rate swap is the amount of which the interest rates are calculated.
- ▷ $T_S = 10Y$. The start date of the IRS refers to the date where the first floating rate payment is fixed.
- ▷ $T_E = 30Y$. The maturity date of the IRS refers to the date where the last payment is transfered between the counterparties.
- ▷ $K = R(0, T_S, T_E)$. The fixed rate is set to the *par swap rate* which ensures that the present value of the fixed leg and the floating are equal, see (6). The swap is thus ATM.
- ▷ $\delta^{float} = 0.5$. Year fraction between the floating rate payments is semiannual as for standard EUR interest swaps.
- ▷ $\delta^{fix} = 1.0$. Year fraction between the fixed rate payments is annual as for standard EUR interest swaps.

For each of the fixed leg payment dates:

$$\left\{T_S + \delta^{fix}, T_S + 2 \cdot \delta^{fix}, \dots, T_S + M^{fix} \cdot \delta^{fix}\right\}, \quad M^{fix} = \frac{T_E - T_S}{\delta^{fix}},$$

the default-risky counterparty pays the fixed amount:

$$N \cdot \delta^{fix} \cdot K = N \cdot \delta^{fix} \cdot R(0, T_S, T_E),$$

to the default-free counterparty. The time $t \leq T_S + \delta^{fix}$ value of the fixed leg is then given by:

$$\Pi_t^{fix} = N \sum_{i=1}^{M^{fix}} P\left(t, T_S + i \cdot \delta^{fix}\right) \delta^{fix} R(0, T_S, T_E),$$

where the discount factor, $P(t, T)$, is the time t value of a zero coupon bond with maturity T.

For each of the floating leg payment dates:

$$\left\{T_S + \delta^{float}, T_S + 2 \cdot \delta^{float}, \ldots, T_S + M^{float} \cdot \delta^{float}\right\}, \quad M^{float} = \frac{T_E - T_S}{\delta^{float}},$$

the default-free counterparty pays the floating amount:

$$N \cdot \delta^{float} L\left(T_S + (i-1) \cdot \delta^{float}, T_S + i \cdot \delta^{float}\right), \qquad i = 1, \ldots, M^{float},$$

to the default-risky counterparty. Here, $L(T_{i-1}, T_i)$ is the spot EURIBOR rate between time T_{i-1} and T_i, fixed at time T_{i-1}. The time $t < T_S$ value of the floating leg is then given by:

$$\begin{aligned}\Pi_t^{float} =& N \sum_{i=1}^{M^{float}} \delta^{float} F\left(t, T_S + (i-1) \cdot \delta^{float}, T_S + i \cdot \delta^{float}\right) \cdot P\left(t, T_S + i \cdot \delta^{float}\right) \\ \overset{\triangle}{=}& N\left(P\left(t, T_S\right) - P\left(t, T_E\right)\right),\end{aligned}$$

$\triangle$: reduced by the definition of the forward EURIBOR rates, $F(t, T_{i-1}, T_i)$, and the equally spaced time points $T_i - T_{i-1} = \delta^{float}$, see Linderstroem (2013, p. 18) for details.

For $T_S \leq t < T_S + \delta^{float}$ the value of the EURIBOR rate, $L\left(T_S, T_S + \delta^{float}\right)$, has already been fixed and the first payment in the floating leg is therefore known:

$$\Pi_t^{float} = N\left[\delta^{float} L\left(T_S, T_S + \delta^{float}\right) P\left(t, T_S + \delta^{float}\right) + P\left(t, T_S + \delta^{float}\right) - P\left(t, T_E\right)\right],$$

where the EURIBOR rate according to Brigo & Mercurio (2006, p.7) can be determined as:

$$L\left(T_S, T_S + \delta^{float}\right) = \frac{1 - P\left(T_S, T_S + \delta^{float}\right)}{\delta^{float} P\left(T_S, T_S + \delta^{float}\right)}.$$

The time $t \leq T_S$ value of the receiver IRS, as seen from the default-free party, is then given by:

$$\Pi_t^{receiver} = \Pi_t^{fixed} - \Pi_t^{float}. \tag{5}$$

The value for $T_S < t$ is determined using the formulas above with adjusted start date, $T_S + \delta$, determined for each leg individually.

The fixed rate, given by the *par swap rate* $R(0, T_S, T_E)$, is set at time $t = 0$ such that $\Pi_0^{fix} = \Pi_0^{float}$:

$$R(0, T_S, T_E) = \frac{P(0, T_S) - P(0, T_E)}{\sum_{i=1}^{M^{fix}} \delta^{fix} P\left(0, T_S + i \cdot \delta^{fix}\right)}. \tag{6}$$

2.3 Short rate model - Hull White

Sources: Hansen & Glibstrup (2014), Munk (2011).

The expected discounted exposure term in the CVA approximation, (4), for an interest rate swap with value given by eqn. (5):

$$\mathbb{E}_0^{\mathbb{Q}}\left[D(0,t_i)Ex(t_i)\right] = \mathbb{E}_0^{\mathbb{Q}}\left[D(0,t_i)\left(\Pi_{t_i}^{receiver}\right)^+\right],$$

can be evaluated using a model for the underlying stochastic short rate, $r(t)$. Here, the Hull-White model has been chosen due to its time varying parameter, $\theta_{hw}(t)$, which allows for a perfect fit to the observed term structure of zero coupon bond prices, $T \mapsto P^*(0,T)$. This feature is not available in simpler models with constant parameters and it is particularly important in connection with pricing of derivatives as pointed out by Munk.

> If the model is not able to price the underlying securities (that is the zero-coupon bonds) correctly, why trust the model prices for derivative securities? (Munk (2011))

The short rate dynamics under the risk-neutral $\mathbb{Q}$-measure in the Hull White model are given by:

$$dr(t) = \left[\theta_{hw}(t) - a_{hw}\cdot r(t)\right]dt + \sigma_{hw}dW^{\mathbb{Q}}(t), \qquad r(0) = r_0, \tag{7}$$

where a_{hw}, σ_{hw} are positive constants and the time-varying function, $\theta_{hw}(t)$, is chosen such that the model matches the observed zero coupon yield curve as described below.

Let the observed zero coupon bond prices be given in terms of the corresponding instantaneous forward rates:

$$f^*(0,T) = -\frac{\partial \log P^*(0,T)}{\partial T}, \qquad f^*(0,0) = r(0).$$

The model will then match these prices if $\theta_{hw}(t)$ is chosen according to Munk (2011, theorem. 9.3):

$$\theta_{hw}(t) = \frac{\partial f^*(0,t)}{\partial T} + a_{hw}f^*(0,t) + \frac{\sigma_{hw}^2}{2a_{hw}}\left(1 - e^{-2a_{hw}t}\right), \qquad \theta_{hw}(0) = a_{hw}\cdot r(0). \tag{8}$$

Notice that the zero coupon bond prices, $P^*(0,T)$, are not directly observable in the market and must therefore be estimated from other more liquid quotes. Thus, before the parameters of the Hull-White model can determined, the zero coupon yield curve must initially be fitted to the market as described in section 2.5.1.

The theoretical zero coupon bond price at time t for the stochastic short rate, r, is given by:

$$P(t,T) = \mathbb{E}_t^{\mathbb{Q}}\left[e^{-\int_t^T r(u)du}\right]. \tag{9}$$

This price can in the Hull-White model, with θ chosen according to (8), be explicitly determined as:

$$\begin{aligned} P(t,T) &= e^{-A(t,T)-B(t,T)r(t)}, \\ B(t,T) &= \frac{1}{a}\left(1-e^{-a(T-t)}\right), \\ A(t,T) &= -\ln\left(\frac{P^*(0,T)}{P^*(0,t)}\right) - B(t,T)f^*(0,t) + \frac{\sigma^2}{4a}B(t,T)^2\left(1-e^{-2at}\right). \end{aligned} \tag{10}$$

Hence, at any given future time, t_i, the entire term structure: $T \mapsto P(t_i,T)$, can be generated from the short rate value $r(t_i)$. The valuation of the interest rate swap at time t_i given by eqn. (5) is thus reduced to simulation of the underlying short rate $r(t_i)$.

The calibration of the Hull-White model is described in section 2.5.2 and involves the valuation of a European put option written on a zero coupon bond. In the Hull-White model the current time $t = 0$ price of this option with strike K, maturity T_{i-1} and underlying ZCB with maturity $T_i > T_{i-1}$ is given by:

$$\begin{aligned} \Pi_0^{ZCB,PUT}(T_{i-1},T_i,K) &= KP(0,T_{i-1})\Phi\left(\sigma_P - d\right) - P(0,T_i)\Phi(-d), \\ d &= \frac{1}{\sigma_P}\log\left(\frac{P(0,T_i)}{P(0,T_{i-1})K}\right) + \frac{1}{2}\sigma_P, \\ \sigma_P &= \sigma_{hw}\sqrt{\frac{1-e^{-2a_{hw}T_{i-1}}}{2a_{hw}}}B(T_{i-1},T_i), \end{aligned} \tag{11}$$

where Φ is the standard normal distribution function.

2.4 Intensity model - CIR

Sources: Brigo & Mercurio (2006), Hansen & Glibstrup (2014).

The counterparty's default probability within a time bucket $(t_{i-1},t_i]$ in the CVA approximation, (4), can be rewritten in terms of survival probabilities:

$$\mathbb{Q}_0\left\{\tau \in (t_{i-1},t_i]\right\} = \mathbb{Q}_0\left(\tau > t_{i-1}\right) - \mathbb{Q}_0\left(\tau > t_i\right) \tag{12}$$

In this framework the default time, τ, is modeled as the time for the first jump of a Poisson process with stochastic intensity rate $\lambda(t)$. The formula for the survival probabilities is in this case given by:

$$\mathbb{Q}_0\left(\tau > t\right) = \mathbb{E}_0^{\mathbb{Q}}\left[e^{-\int_0^t \lambda(u)du}\right]$$

which is equivalent to the zero coupon bond price with stochastic short rate λ. This therm can thus be evaluated using a short rate model for the intensity.

The CIR model is here chosen for the intensity rate, λ, which are then governed by the $\mathbb{Q}$-dynamics:

$$d\lambda(t) = a_{cir}\left[b_{cir} - \lambda(t)\right]dt + \sigma_{cir}\sqrt{\lambda(t)}dW^{Q}(t), \quad \lambda(0) = \lambda_0 \tag{13}$$

with positive constants $\lambda_0, a_{cir}, b_{cir}, \sigma_{cir}$. The condition $2a_{cir}b_{cir} > \sigma_{cir}^2$ ensures that the intensity remains positive.

The survival probabilities are then explicitly given by the CIR formula for the price of a zero coupon bond with intensity process λ:

$$\begin{aligned}
&\mathbb{Q}_t\left(\tau > T\right) = A(t,T)e^{-B(t,T)\lambda(t)},\\
&A(t,T) = \left[\frac{2\gamma e^{(a_{cir}+\gamma)\left(\frac{T-t}{2}\right)}}{\left(a_{cir}+\gamma\right)\left(e^{\gamma(T-t)}-1\right)+2\gamma}\right]^{\frac{2a_{cir}b_{cir}}{\sigma^2}}\\
&B(t,T) = \frac{2\left(e^{\gamma(T-t)}-1\right)}{\left(a_{cir}+\gamma\right)\left(e^{\gamma(T-t)}-1\right)+2\gamma}\\
&\gamma = \sqrt{a_{cir}^2 + 2\sigma_{cir}^2}.
\end{aligned} \tag{14}$$

Since the CVA approximation, (4), is only evaluated at time $t = 0$ in this framework, the corresponding survival probabilities can thus be evaluated using only the initial intensity rate λ_0. Hence, the dynamics given in eqn. (13) are not used here for simulating the stochastic intensity rate at future time points t.

If CDS quotes for maturities up to $T = 30Y$, matching the horizon of the interest rate swap, had been available, the survival probabilities could have been directly stripped from these without assuming a model, as described in Brigo *et al.* (2013, p.68).

But, as seen in table 3, only quotes with maturities up to $T = 10Y$ are available here. The CIR model is therefore used in this context as a sort of *extrapolation* method through the survival probability formula given in eqn.(14).

2.5 Calibrating the model framework

Calibration of the Hull-White and the CIR model to market data are implemented as least squares optimization. Hence, nonlinear least squares problems are formulated for each set of parameters and are then solved using the numerical routine *Levenberg-Marquardt* given in Press *et al.* (2007).

2.5.1 Calibration zero coupon yield curve

Sources: Linderstroem (2013), Hansen & Glibstrup (2014).

The Hull White model is calibrated such that the theoretical zero coupon bond prices, $P(0,T)$, matches the corresponding prices observed in the market, $P^*(0,T)$, by means of the function θ_{cir}. Unfortunately these prices are not directly observable in the market and an initial calibration to observed swap rates is therefore required.

Let the zero coupon bond prices, $P(0,T)$, be expressed in terms of their corresponding continuously compounded zero coupon rate, $r_{cont}(0,T)$, so that:

$$P(0,T) = \exp\left(-r_{cont}(0,T) \cdot T\right).$$

The zero coupon yield curve is then defined by the mapping: $T \mapsto r_{cont}(0,T)$, and is here represented by a natural cubic spline with knot points: $r^*_{cont}(0,T)$ for $T \in Z_1 = \{1,2,3,4,5,7,10,15,20,25,30\}$ corresponding to each observed maturity given in table 1.

The zero coupon rates, $r^*_{cont}(0,T)$, are calibrated by calculating the theoretical swap rates according to eqn. (6) for spot starting EUR interest rate swaps with annual fixed payments:

$$R(0,0,T) = \frac{1.0 - P(0,T)}{\sum_{i=1}^{T} P(0,i)} = \frac{1.0 - \exp\left(-r_{cont}(0,T) \cdot T\right)}{\sum_{i=1}^{T} \exp\left(-r_{cont}(0,i) \cdot i\right)}.$$

The calibration of the zero coupon yield curve is then given by the least squares optimization problem:

$$\min_{\mathbf{r}_{cont}} \sum_{T \in Z_1} \left(R(0,0,T) - R^*(0,0,T)\right)^2, \qquad \mathbf{r}_{cont} = \{r_{cont}(0,T)\}_{T \in Z_1}.$$

where the observed quotes, $R^*(0,0,T)$, are given in table 1.

Maturity	Swap Rate
1	0.3680 %
2	0.3860 %
3	0.4780 %
4	0.6235 %
5	0.7915 %
7	1.1430 %
10	1.5890 %
15	2.0397 %
20	2.2131 %
25	2.2704 %
30	2.2820 %

Table 1: Swap Rates - EUR IRS.
Source: Hansen & Glibstrup (2014, table 1).

The 1-day quote given by the EONIA-rate of 0.1690%[11] can afterwards be added to the collection of fitted knot points: $r^*_{cont}(0, T)$, $T \in Z^*_1 = \{1/365, Z_1\}$.

The calculation of the CVA price in eqn. (4) for the IRS specified in section 2.2 includes zero coupon bond prices with maturities $T \in \{0.25, 0.5, \ldots, 30\}$. Hence, the full zero coupon yield curve are here obtained directly from the market quotes and extrapolation of the cubic spline of zero coupon rates is therefore not needed.

2.5.2 Calibration Hull-White

Sources: Hansen & Glibstrup (2014), Munk (2011).

The parameters to be calibrated in the Hull-White model, for the short rate dynamics given in eqn. (7), are the initial value r_0, the mean-reversion speed a_{hw} and the volatility σ_{hw}.

The initial value of the short rate process, r_0, is set to the 1-day EONIA rate of 0.00169. The estimate $a_{hw} = 0.1$ is fixed in advance to avoid unrealistically low levels as described in Hansen & Glibstrup (2014, p. 40). The volatility σ_{hw} is calibrated to a set of observed cap quotes using the theoretical cap prices given by the Hull-White model.

Structurally an interest rate cap is identical to the floating leg in an IRS, as it also has a series of payments based on the floating rate, $L(T_{i-1}, T_i)$, fixed at time T_{i-1}

[11] See Hansen & Glibstrup (2014, p. 39).

and payed at time T_i. These individual payments are called caplets and they are given as call options on the floating rate with strike K and expiry T_i. Thus, for each payment date:

$$\left\{T_S + \delta^{float}, T_S + 2 \cdot \delta^{float}, \dots, T_S + M^{float} \cdot \delta^{float}\right\}, \quad M^{float} = \frac{T_E - T_S}{\delta^{float}},$$

the caplet is given by the payoff:

$$N \cdot \delta^{float}\left(L\left(T_S + (i-1)\cdot\delta^{float}, T_S + i \cdot \delta^{float}\right) - K\right)^+, \qquad i = 1, \dots, M^{float}.$$

These caplets are reformulated in various textbooks[12] as the price of a European put option with expiry $T_S + (i-1)\cdot\delta^{float}$ and strike $\frac{1}{1+\delta^{float}K}$, written on a zero-coupon bond with maturity $T_S + i\cdot\delta^{float}$. Thus, the current time $t = 0$ value of the caplets are given by:

$$\begin{aligned}&\Pi_0^{caplet}(T_S + (i-1)\cdot\delta^{float}, T_S + i\cdot\delta^{float}, K)\\&\qquad = N(1+\delta^{float}K)\cdot\Pi_0^{ZCB,PUT}\left(T_S + (i-1)\cdot\delta^{float}, T_S + i\cdot\delta^{float}, \frac{1}{1+\delta^{float}K}\right),\end{aligned}$$

which can be evaluated in the Hull-White model using the explicit formula in eqn. (11).

The cap is then given by a sum of the caplets and has time $t = 0$ value given by:

$$\begin{aligned}\Pi_0^{cap}(T_S, T_E, K) &= \sum_{i=1}^{M^{float}} \Pi_0^{caplet}(T_S + (i-1)\cdot\delta^{float}, T_S + i\cdot\delta^{float}, K) \qquad (15)\\&= N(1+\delta^{float}K)\sum_{i=1}^{M^{float}}\Pi_0^{ZCB,PUT}\left(T_S + (i-1)\cdot\delta^{float}, T_S + i\cdot\delta^{float}, \frac{1}{1+\delta^{float}K}\right).\end{aligned}$$

The observed quotes used for the calibration, given in table 2, are spot starting caps with semi-annual payments, $\delta^{float} = 0.5$, on the EURIBOR rate. By convention, the first caplet is disregarded for a spot starting cap and the start date is thus set to $T_S = 0.5$. The quotes are given in terms of their ATM strike level which is defined as:

$$K_0^{ATM}(T_S, T_E) = \frac{P(0,T_S) - P(0,T_E)}{\sum_{i=1}^{M^{float}}\delta^{float}P(0, T_S + i\cdot\delta^{float})}.$$

[12] See for instance Munk (2011, p.165).

Maturity	ATM Strike	Price (bps)
3	0.49 %	44
4	0.65 %	95
5	0.83 %	165
6	1.01 %	249
7	1.19 %	343
8	1.36 %	443
9	1.50 %	545
10	1.64 %	649
12	1.85 %	854
15	2.07 %	1140
20	2.23 %	1556
25	2.28 %	1913
30	2.29 %	2233

TABLE 2: Observed ATM Strikes and Prices for spot starting EUR caps.
Source: Hansen & Glibstrup (2014, table 2).

The calibration of the volatility parameter in the Hull-White model, σ_{hw}, is then given by the least squares optimization problem:

$$\min_{\sigma} \sum_{T \in Z_2} \left(\Pi_0^{cap} \left(0.5, T, K_0^{ATM} (0.5, T) \right) - \Pi_0^{*\ cap} \left(0.5, T, K_0^{*\ ATM} \right) \right)^2 ,$$

where Z_2 is the set of maturities for the observed quotes, $\Pi_0^{*\ cap} \left(0.5, T, K_0^{*\ ATM} \right)$, given in table 2. The volatility parameter obtained as the solution to this optimization problem is given by:

$$\sigma_{hw} = 0.0112472.$$

2.5.3 CALIBRATION CIR

Sources: Brigo *et al.* (2013), Hansen & Glibstrup (2014).

The parameters to be calibrated in the CIR model for the intensity rate dynamics, given in eqn. (13), are the initial value λ_0 and the parameters a_{cir}, b_{cir}, σ_{cir}.

The volatility parameter, σ_{cir}, is in Hansen & Glibstrup (2014) shown to have a minimal effect on the survival probability. The value has therefore been estimated from S&P-data[13] in order to achieve a stabile result: $\tilde{\sigma}_{cir} = 0,0923$. The remaining parameters λ_0, a_{cir}, b_{cir} are calibrated to a set of observed CDS quotes, with

[13]Details on the estimation is given in Hansen & Glibstrup (2014, sec. 5.4.2).

the defaultable counterparty *Adidas AG* as reference entity, using the theoretical *fair CDS premium* evaluated in the CIR intensity model.

Credit Default Swaps (CDS) protects the buyer against default of the reference entity between time T_S and T_E. The protection buyer will in the event of default receive the Loss Given Default, $(1-REC)\cdot N$, from the protection seller. In return the buyer pays periodically a fixed amount, $N \cdot C$, to the seller until either default, τ, or expiry, T_E.

Let in the following $T_S = 0$, as in table 3, so that the payment dates for the fixed payments are given by: $\left\{\delta, 2\cdot\delta, \dots, M^{CDS}\cdot\delta\right\}$ with $M^{CDS} = T_E/\delta$. Assume that if the default occurs between to successive payment dates, $\tau \in ((i-1)\cdot\delta, i\cdot\delta]$, the protection buyer must still pay the premium for the entire period at time $i\cdot\delta$. The value of the *premium leg* can then be derived as:

$$\Pi_0^{prem}(T_E) = N \cdot C \cdot \delta \sum_{i=1}^{M^{CDS}} P(0, i\cdot\delta)\mathbb{Q}_0(\tau > i\cdot\delta),$$

The *protection leg* has one payment at the event of default, τ, if this occurs before maturity, T_E. If the default occurs between to payment dates, $\tau \in ((i-1)\cdot\delta, i\cdot\delta]$, the LGD amount is assumed not to be paid until the end of the period $i\cdot\delta$. The value of the *protection leg* can then be derived as:

$$\Pi_0^{protect}(T_E) = N \cdot (1 - REC) \sum_{i=1}^{M^{CDS}} P(0, i\cdot\delta)\left(\mathbb{Q}_0(\tau > (i-1)\cdot\delta) - \mathbb{Q}_0(\tau > i\cdot\delta)\right).$$

The CDS quotes are given in terms of their *fair premium*, $C(0, T_E)$, which similar to the *par swap rate* ensures that present value of the two legs are equal:

$$C(0, T_E) = \frac{(1-REC)}{\delta} \frac{\sum_{i=1}^{M^{CDS}} P(0, i\cdot\delta)\left(\mathbb{Q}_0(\tau > (i-1)\cdot\delta) - \mathbb{Q}_0(\tau > i\cdot\delta)\right)}{\sum_{i=1}^{M^{CDS}} P(0, i\cdot\delta)\mathbb{Q}_0(\tau > i\cdot\delta)} \tag{16}$$

The observed quotes used for the calibration are in table 3. The payment periods for these are quarterly, $\delta = 0.25$, and the recovery rate is set to a constant $REC = 0.4$.

The calibration of the parameters λ_0, a_{cir}, b_{cir} in the CIR model is then given by the least squares optimization problem:

$$\min_{\lambda_0, a_{cir}, b_{cir}} \sum_{T\in Z_3} \left[C(0,0,T) - C^*(0,T)\right],$$

Maturity	Annual CDS premium (bps)
1	104.1
2	129.1
3	154.3
4	191.5
5	229.5
7	253.1
10	271.1

Table 3: Observed CDS (fair) premiums with Adidas AG as reference entity. Source: Hansen & Glibstrup (2014, table 3).

where Z_3 is the set of maturities for the observed premiums, $C^*(0,T)$, given in table 3. The parameters obtained as the solution to this optimization problem is given by:

$$\lambda_0 = 0.00848787, \qquad a_{cir} = 0.250464, \qquad b_{cir} = 0.0753663.$$

2.6 Discretization

In order to evaluate the CVA approximation given in eqn. (4), the underlying short rate must be simulated and the expectation of the discounted exposure estimated. The scheme and Monte Carlo estimate used for this evaluation are described in this section.

2.6.1 Hull-White: Simulating the Short Rate Process

Sources: Brigo & Mercurio (2006), Glasserman (2004).

Let the short rate process, r, be modeled by the Hull-White $\mathbb{Q}$-dynamics given in eqn. (7) with function θ_{hw} defined in eqn. (8). Conditional on the value at time t_{i-1}, the value at time $t_i > t_{i-1}$ is normally distributed with mean and variance given by[14]:

$$\begin{aligned}
\mathbb{E}^{\mathbb{Q}}_{t_{i-1}}[r(t_i)] &= r(t_{i-1})e^{-a_{hw}\delta} + \alpha(t_i) - \alpha(t_{i-1})e^{-a_{hw}\delta}, \\
\mathbb{V}ar^{\mathbb{Q}}_{t_{i-1}}[r(t_i)] &= \frac{\sigma^2_{hw}}{2a_{hw}}\left(1 - e^{-2a_{hw}\delta}\right), \\
\alpha(t_i) &= f^*(0,t_i) + \frac{\sigma^2_{hw}}{2a^2_{hw}}\left(1 - e^{-a_{hw}t_i}\right)^2, \qquad \delta = t_i - t_{i-1}.
\end{aligned}$$

[14]See Brigo & Mercurio (2006, eqn. 3.37) for details on the conditional distribution for $r(t)$.

An *exact* simulation scheme can be obtained by inserting these explicit expressions in Glasserman (2004, eqn. (3.45)):

$$r(t_i) = r(t_{i-1})e^{-a_{hw}\cdot\delta} + \alpha(t_i) - \alpha(t_{i-1})e^{-a_{hw}\cdot\delta} + \sigma_{hw}\sqrt{\frac{1-e^{-2a_{hw}\cdot\delta}}{2a_{hw}}}\cdot Z_i, \qquad (17)$$

where $Z_i \sim \mathcal{N}(0,1)$ and $\delta = t_i - t_{i-1}$ is the constant step size. This scheme is *exact* as it does not entail a discretization error, opposed to the simpler Euler scheme used in Hansen & Glibstrup (2014):

$$r(t_i) = r(t_{i-1}) + [\theta_{hw}(t_{i-1}) - a_{hw}\cdot r(t_{i-1})]\,\delta + \sigma_{hw}\sqrt{\delta}Z_i.$$

2.6.2 CVA: Simulating

The valuation of the CVA approximation, (4), is to a great extend dependent on the preliminary assumption of independence between the discounted exposure and the credit quality of the counterparty. This is the exact assumption that allows for a separate valuation of the two terms marked below:

$$CVA_0 = (1-\text{REC})\sum_{i=1}^{n}\underbrace{\mathbb{Q}_0\{\tau\in(t_{i-1},t_i]\}}_{\text{① Credit model}}\underbrace{\mathbb{E}_0^{\mathbb{Q}}[D(0,t_i)Ex(t_i)]}_{\text{② Short rate model}}.$$

Here, the RHS in the time bucket, $(t_{i-1}, t_i]$, is given by: $t_i = i\cdot\delta^\tau$ for $i = 1,\dots,n$ where $t_0 = 0$, $t_n = T$ and δ^τ is the constant step size set to 0.25. The maturity, T, is for the 10Y20Y interest rate swap given by $T = 30Y$.

① This default probability is evaluated as the difference between the corresponding survival probabilities as described in eqn. (12).

Each of these survival probabilities, $\mathbb{Q}_0(\tau > t)$, are evaluated according to a credit model for the counterparty, here given by the CIR model calibrated to CDS quotes with *Adidas AG* as reference entity.

Thus, for each time bucket this term can be evaluated using the explicit formula for the survival probability given in eqn. (14.)

② The expection of the discounted exposure is evaluated using a model for the stochastic short rate, here given by the Hull-White model calibrated to a set of cap quotes.

The exposure defined in eqn. (1) is for the interest rate swap given by:

$$Ex(t_i) = \left(\mathbb{E}_t^{\mathbb{Q}}[\Pi(t_i,T)]\right)^+ = \left(\Pi_{t_i}^{receiver}\right)^+$$

The time t_i price of the receiver swap can be evaluated by the explicit formula given in eqn. (5), with $r(t_i)$ provided as input. The fixed rate for the interest rate swap is given by the par swap rate, calculated according to eqn. (6) for time $t = 0$.

The discount factor, $D(0, t_i)$, is given in terms of the short rate:

$$D(0, t_i) = e^{-\int_0^{t_i} r(s)ds} \overset{\triangle}{\approx} e^{-\sum_{j=1}^{i} \frac{r(t_j)+r(t_{j-i})}{2}\delta^\tau} \tag{18}$$

$\triangle$: the integral $\int_0^{t_i} r(s)ds$ is approximated by a Riemann sum using the midpoint of the short rates.

The total expected discounted exposure term can then be estimated using Monte Carlo simulation:

$$\mathbb{E}_0^{\mathbb{Q}}\left[D(0, t_i)Ex(t_i)\right] \approx \frac{1}{M}\sum_{m=1}^{M} e^{-\sum_{j=1}^{i} \frac{r^{(m)}(t_j)+r^{(m)}(t_{j-i})}{2}\delta^\tau} \left(\Pi_{t_i}^{(m),receiver}\right)^+ . \tag{19}$$

where M is the number of sample paths. See Glasserman (2004) for details on Monte Carlo estimation.

The CVA *evaluation flow* is illustrated in figure 1 where each of the short rate sample paths are depicted as a function of t_i for $i = 1, \ldots, n$. The vertical dashed lines indicate that for each time bucket, the terms ① and ② must be evaluated. Pseudocode for the CVA calculation are given in algorithm 1.

Algorithm 1: Pseudocode for the CVA calculation for an IRS.

```
input: $n$: no. of time buckets,
       $N$: no. of sample paths,
$CVA_0 := 0.0$;
for $t_i := t_1$ to $t_n$ $(= T)$ do
    $sum := 0.0$;
    for $m := 1$ to $N$ do
        $r^{(m)}(t_i) \sim$ simulate according to eqn. (17).
        $D^{(m)}(0, t_i) \sim$ update the Riemann sum according to eqn. (18).
        $\Pi_{t_i}^{(m),receiver} \sim$ calculate according to eqn. (5).
        $sum := sum + D^{(m)}(0, t_i)\left(\Pi_{t_i}^{(m),receiver}\right)^+$
    ①: $\mathbb{E}_0^{\mathbb{Q}}\left[D(0, t_i)Ex(t_i)\right] := \frac{sum}{M}$.
    ②: $\mathbb{Q}_0\{\tau \in (t_{i-1}, t_i]\} \sim$ calculate according to eqn. (12) and eqn. (14).
    $CVA_0 := CVA_0 + \mathbb{Q}_0\{\tau \in (t_{i-1}, t_i]\} \cdot \mathbb{E}_0^{\mathbb{Q}}\left[D(0, t_i)Ex(t_i)\right]$.
$CVA_0 := (1 - REC) \cdot CVA_0$.
```

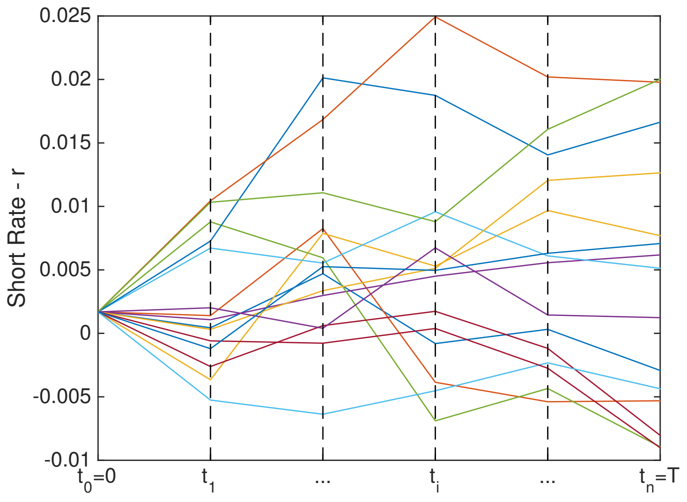


Figure 1: Illustration of CVA evaluation flow.

2.7 The derivatives

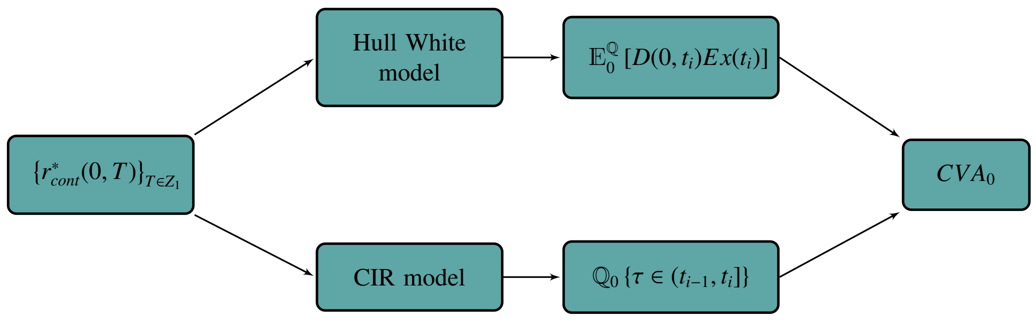
Counterparty credit risk management requires knowledge of the sensitivities wrt. market factors, $\boldsymbol{\theta}$, as mentioned in the beginning of sec. 2. Here, the sensitivities of interest are given by the derivatives of the CVA price wrt. points on the zero coupon yield curve.

Recall from sec. 2.5.1 that the zero coupon yield curve is given in terms of the continuously compounded zero coupon rates, $T \mapsto r^*_{cont}(0,T)$, fitted to observed swap rates for maturities $T \in Z^*_1 = \{1/360, 1, 2, 3, 4, 5, 7, 10, 15, 20, 25, 30\}$. This curve is given as input to both the Hull-White and the CIR model, which in turn are used for calculating the CVA price. This data flow is illustrated in figure 2.

Hence, the sensitivities of the CVA price for $\theta = \{r^*_{cont}(0,T)\}_{T\in Z^*_1}$ are in this setting given by:

$$\begin{aligned}\frac{\partial CVA_0(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = (1-\mathrm{REC})\sum_{i=1}^{n}\Bigg(&\underbrace{\frac{\partial\left(\mathbb{Q}_0\left\{\tau\in(t_{i-1},t_i]\right\}\right)}{\partial\boldsymbol{\theta}}}_{①\text{ CIR model}}\cdot\mathbb{E}_0^{\mathbb{Q}}\left[D(0,t_i)Ex(t_i)\right] \\ &+\underbrace{\frac{\partial\left(\mathbb{E}_0^{\mathbb{Q}}\left[D(0,t_i)Ex(t_i)\right]\right)}{\partial\boldsymbol{\theta}}}_{②\text{ Hull-White model}}\cdot\mathbb{Q}_0\left\{\tau\in(t_{i-1},t_i]\right\}\Bigg).\end{aligned} \tag{20}$$

The results presented in section 5 are given for two cases: one where ① and ② in-



$\{r^*_{cont}(0,T)\}_{T\in Z_1}$
Hull White model
CIR model
$\mathbb{E}^{\mathbb{Q}}_0\left[D(0,t_i)Ex(t_i)\right]$
$\mathbb{Q}_0\{\tau \in (t_{i-1},t_i]\}$
CVA_0

FIGURE 2: Structure of the dependence between input factors, $\theta = \{r^*_{cont}(0,T)\}_{T\in Z^*_1}$, and the two terms used for the calculation of the CVA_0 price.

clude the calibration process for the respective models, and one where they do not.

The derivative of the expected discounted exposure, ②, is here given by the derivative of the Monte Carlo estimator given in eqn. (19). In order to later evaluate this term using algorithmic differentiation, it is initially reformulated using the *pathwise sensitivity estimator*:

$$\frac{\partial \mathbb{E}^{\mathbb{Q}}_0\left[D(0,t_i)Ex(t_i)\right]}{\partial \boldsymbol{\theta}} \approx \frac{1}{M}\sum_{m=1}^{M} \frac{\partial\left[e^{-\sum_{j=1}^{i}\frac{r^{(m)}(t_j)+r^{(m)}(t_{j-i})}{2}\delta^{\tau}}\left(\Pi^{(m),receiver}_{t_i}\right)^{+}\right]}{\partial \boldsymbol{\theta}}. \tag{21}$$

This estimator stems from interchanging the differentiation and the (implicit) integral on the LHS under Glasserman's rule of thumb: *the pathwise method applies when the payoff is continuous in the parameter of interest.* This condition is fulfilled in this basic setup, as the zero coupon rates are represented by a cubic spline continuous in the first order derivatives.

3 Derivatives Framework

Sources: Griewank & Walther (2008), Homescu (2011).

There are several ways for obtaining the first order derivatives of a function in practice. Assume in the following that the original function F has already been translated into an algorithm that can be implemented as a computer program. The aim is then to determine the derivatives of function F just as efficiently and accurately as this original algorithm.

In this section an overview of three main approaches for obtaining derivatives: *symbolic*, *numerical* and *algorithmic*, are provided along with a list of pros and cons. Subsequently, three specific methods: *finite-difference approximation*, *complex-step derivative approximation* and *algorithmic differentiation*, are described including both theoretical and practical details.

Let in the following F be a vector-valued function, $F : \mathcal{D} \subset \mathbb{R}^n \longmapsto \mathbb{R}^m$, for which the Jacobian is a well-defined matrix-valued function on $\mathcal{D}$:

$$F'(\mathbf{x}) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial F_m}{\partial x_1} & \frac{\partial F_m}{\partial x_2} & \cdots & \frac{\partial F_m}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{m \times n}. \tag{22}$$

Here, F is defined of as a subpart of the overall CVA algorithm, as illustrated by figure 3, for input parameters $\boldsymbol{\theta}$, intermediate variables $\mathbf{x}$, $\mathbf{y}$ and final outputs $\boldsymbol{\Pi}$.

$$\begin{tikzcd} \boldsymbol{\theta} \arrow[r] & \cdots \arrow[r] & \mathbf{x} \arrow[r] & \boxed{F} \arrow[r] & \mathbf{y} \arrow[r] & \cdots \arrow[r] & \boldsymbol{\Pi} \end{tikzcd}$$

Figure 3: Structure of algorithm.

- ▷ **Symbolic** evaluation is the classical method known from differential calculus where explicit formulas for the derivatives are formulated using the elementary rules of differentiation. Thus, by definition this derivative is *accurate* opposed to the numerical approximations discussed below.

  Regarding financial products, the model dynamics and/or the payoff functions are in general too complex for nice, closed-form expressions to be derived by hand within a reasonable amount of time. Further, this method does not take advantage of the algorithm already translated for the original calculation, and is not in itself designed for an efficient execution.

This method, in its original form, is therefore disregarded for this setup.

▷ **Numerical** evaluation is here defined as approximation of derivatives using Taylor expansion. In the following sections two different methods for numerical evaluation is presented: the well-known *finite difference approximation* and *the complex-step derivative approximation* described in Homescu (2011) and Martins *et al.* (2003).

Both these approximations are subject to truncation error due to their construction. But only the finite difference method is also subject to cancellation error[15] which limits the overall accuracy that can be obtained using this method.

The implementation for both methods consists of no, or minor, adjustments to the original algorithm. In return, the corresponding computational costs for these numerical methods are relatively high.

▷ **Algorithmic** evaluation consists of applying the chain rule to a series of derivatives by propagating the original algorithm either *forward* or in *reverse*.

These derivatives are determined for each statement, or intermediate variable, in the original algorithm using the elemental rules of differentiation. This is similar to the procedure described for the symbolic method, only here the expressions being differentiated are much simpler fractions of the overall calculation.

> ...we may view symbolic differentiation as a fairly close relative with whom we ought to interact more than we usually do... (Griewank & Walther (2008, p.33))

The symbolic expression for each of the derivatives are immediately evaluated as any other intermediate variable in a computer program. Thus, the chain rule is in practice applied to a series of numerical values rather than symbolic expressions. This yields derivatives with working accuracy as only the floating point operations induce roundoff errors.

For the *forward* mode, an additional line of code, representing the intermediate derivative, are added before every *active* variable[16]. For the reverse,

[15]Cancellation error is defined as the round-off error occurring when two finite-precision values of comparable size are subtracted.

[16]The concept of *active* variables are described in detail in section 4.2.

or adjoint, mode a *return sweep* is added at the end of the original evaluation, in which the derivatives are evaluated by backward propagation of the original code.

The computational cost for the forward mode increases with the number of parameters, $\boldsymbol{\theta}$, as for the numerical methods above. The adjoint mode on the other hand, increases with the number of final output values, $\boldsymbol{\Pi}$.

A summary of the features for the methods mentioned above are given in table 4. Remark that both the numerical- and the algorithmic methods *calculate the greeks as a by-product of the original valuation*, as Davidson (2015) beautifully puts it. Thus, for these methods an additional evaluation to obtain the original values is not necessary.

Method	Pros	Cons
Finite Difference Approximation:	Intuitive.	Accuracy depends on the bump-size,
		- Truncation errors.
		- Cancellation errors.
	No additional programming.	Computationally expensive,
		- dependent on #inputs.
Complex-Step Approximation:	Machine precision.	Non-intuitive.
	Handles discontinuous functions.	
	Minimum of additional programming.	Computationally expensive,
		- dependent on #inputs.
Algorithmic Differentiation	Machine Precision.	Considerable amount of additional programming.
-Forward:	Intuitive.	
	Comp. cost independent of #outputs	Comp. cost dependent on #inputs.
-Adjoint:		
	Comp. cost independent of #inputs.	Comp. cost dependent on #outputs.
		Non-intuitive.

Table 4: Summary of features for alternative derivatives methods (1st order derivatives).

Throughout this section, examples of how these methods are applied in practice will be provided for a function, F, defined as the 1-step simulation of the short rate given in eqn. (17). The examples determine the derivative of the simulated short rate, $r(t_i)$, wrt. the Hull-White parameter σ_{hw}.

Compared to figure 3, the input parameter for the overall CVA calculation, $\boldsymbol{\theta}$, is for this example given by the vector of zero coupon rates, $\{r^*_{cont}\}$. The input vector to F of intermediate variables, $\mathbf{x}$, is given by: the Hull White parameters, σ_{hw}, a_{hw}, the previous short rate, $r(t_{i-1})$, and the zero coupon rates, $\{r^*_{cont}\}$. The output vector for F of intermediate variables, $\mathbf{y}$, is given by the simulated short rate, $r(t_i)$. Last, the final output, $\boldsymbol{\Pi}$, is a scalar given by the CVA price.

The pseudo code for the function is provided in algorithm 2 below. Remark, the simulation calls a sub-function, $\alpha(\cdot)$, which is considered as input to the function. Hence, this sub-function and its derivatives are assumed to be handled elsewhere in the code.

Algorithm 2: 1-step simulation of the short rate in the Hull White model.

```
input:
    $\sigma_{hw},\ a_{hw}$,              ▷ Hull White model parameters,
    $\{r^*_{cont}\}$,                    ▷ zero coupon rates,
    $r(t_{i-1})$,                        ▷ Short rate at the previous time $t_{i-1}$.
    $\Delta_\sigma$,                     ▷ Bump size.
    $\alpha(\cdot)$,                     ▷ Sub-function.

$v_1 := \alpha\left(t_{i-1}, \sigma_{hw},\ a_{hw}, \{r^*_{cont}\}\right);$
$v_2 := \alpha\left(t_i, \sigma_{hw},\ a_{hw}, \{r^*_{cont}\}\right);$

$v_3 := \mathcal{N}(0,1);$
$v_4 := e^{-a\cdot(t_i - t_{i-1})};$
$v_5 := \dfrac{1 - e^{-2a\cdot(t_i - t_{i-1})}}{2a};$
$r(t_i) := r(t_{i-1}) \cdot v_4 + v_2 - v_1 \cdot v_4 + \sigma_{hw}\sqrt{v_5} \cdot v_3;$
```

3.1 Finite Difference Approximation

Sources: Glasserman (2004), Griewank & Walther (2008).

Finite difference methods are heavily used in the finance industry, both for solving pricing PDE's and, as here, for determining a set of sensitivities, *the Greeks*. In connection with the latter, this method is often referred to as *bumping* or *bump-and-revalue*. The methods widespread use for determining derivatives of financial products can mainly be ascribed its simplicity, and the fact that no additional development is needed for the implementation. The downside to this, is a poor accuracy of the approximated values which are subject to both truncation- and cancellation errors.

> We can construct finite difference approximations of the derivatives of a function by expressing the derivative as a linear combination of the function evaluated at a number of adjacent points. By expanding the function evaluations in Taylor series, we can determine the coefficients of the linear combination such that the derivative we seek is expressed with any desired order of accuracy. (Tavella & Randall (2000, sec.3))

The finite difference approximation of a derivative is determined by a *scheme*, this is a specification of the linear combination of function values referred to in the quote by Tavella & Randall. In this context, the main difference between the schemes is the accuracy of the generated derivative approximation.

The scheme most commonly used for first order derivatives is the *central difference scheme*. Let in the following $\mathbf{x}(\boldsymbol{\theta})$ be notation for the dependence between input $\mathbf{x}$ and the vector of parameters $\boldsymbol{\theta}$. Then the derivative of function F wrt. a parameter of interest, θ_i, is for this scheme given as the divided difference:

$$\frac{\partial F(\mathbf{x}(\boldsymbol{\theta}))}{\partial \theta_i} \approx \frac{F(\mathbf{x}(\boldsymbol{\theta} + \Delta_\theta \mathbf{e}_i)) - F(\mathbf{x}(\boldsymbol{\theta} - \Delta_\theta \mathbf{e}_i))}{2\Delta_\theta}. \tag{23}$$

where Δ_θ is the *bump* and $\mathbf{e}_i$ is the i'th unit vector such that only the i'th parameter in $\boldsymbol{\theta}$ is bumped. Hence, in practice the derivative is approximated by running the original calculation two additional times, one for each of the *bumped* set of parameters. An example of this is given in algorithm 3.

The finite difference approximation can thus be obtained using only the original evaluation procedure already implemented and therefore has no additional development cost. But, as one might imagine, this procedure becomes computationally costly when there are p parameters of interest instead of just one:

$$TIME\left\{F(\mathbf{x}(\boldsymbol{\theta}))\,;p\times\frac{\partial F(\mathbf{x}(\boldsymbol{\theta}))}{\partial\theta}\right\}=(1+2p)\cdot TIME\{F(\mathbf{x}(\boldsymbol{\theta}))\},\tag{24}$$

where $TIME$ is a relative measure for the run time.

The computational cost could be reduced to $(1+p)\cdot TIME\{F(\mathbf{x}(\theta))\}$ if a one-sided difference scheme was chosen instead. But Taylor expansion show[17] that the accuracy of a one-sided scheme is only $O(\Delta_\theta)$, opposed to $O\left(\Delta_\theta^2\right)$ for the central scheme. This significant difference in accuracy is the reason why the central difference scheme, in spite of its computational cost, is still preferred.

3.1.1 Working the method

Evaluating the derivatives of the CVA calculation presented in this framework requires consideration of two practical issues:

▷ **The bump-size $\boldsymbol{\Delta_\theta}$** - the size of the bump has a significant impact on the accuracy of the generated finite difference estimate. The size has opposing effects on the *truncation error* and the *cancellation error*, respectively.

The accuracy of order $O\left(\Delta_\theta^2\right)$ suggests to choose Δ_θ as small as possible in order to reduce the *truncation error* to a minimum. But when Δ_θ becomes too small, the values subtracted become too close, $F(\mathbf{x}(\boldsymbol{\theta}+\Delta_\theta\mathbf{e}_i)))\approx F(\mathbf{x}(\boldsymbol{\theta}-\Delta_\theta\mathbf{e}_i)))$, in which case the computers finite-precision representation results in *roundoff errors*, or in this context *cancellation errors*. Hence, the step size cannot be too small, as the cancellation error then becomes dominant.

In practice, the *optimal* step size is often approximated by running a series of initial tests or, as here, fixed in advance at the commonly used level $\Delta_\theta = 10^{-4}$.

▷ **Monte Carlo estimate** - the original CVA calculation includes the Monte Carlo simulated values defined in eqn. (19). In this case, the variance of the finite difference approximation can be reduced significantly by using the same random variables, $Z_i^{(m)}$, for both calculations: $CVA_0(\boldsymbol{\theta}+\Delta_\theta\mathbf{e}_i))$ and $CVA_0(\boldsymbol{\theta}-\Delta_\theta\mathbf{e}_i))$. Further details on this subject are given in Glasserman (2004, sec 7.1.1).

[17]See Glasserman (2004, eqn. (7.6)) for details.

Algorithm 3: Bumping of 1-step simulation of the short rate.

```
input: $\sigma_{hw}$, $a_{hw}$,                    ▷ Hull White model parameters,
       $\{r^*_{cont}\}$,                            ▷ zero coupon rates,
       $r(t_{i-1})$,           ▷ Short rate at the previous time $t_{i-1}$.
       $\Delta_\sigma$,                                       ▷ Bump size.
       $\alpha(\cdot)$,                                    ▷ Sub-function.

for $\sigma^{bump} := \{\sigma_{hw} \pm \Delta_\sigma\}$ do
    $v_1 := \alpha\left(t_{i-1}, \sigma^{bump},\ a_{hw}, \{r^*_{cont}\}\right)$;
    $v_2 := \alpha\left(t_i, \sigma^{bump},\ a_{hw}, \{r^*_{cont}\}\right)$;

    $v_3 := \mathcal{N}(0, 1)$;
    $v_4 := e^{-a\cdot(t_i - t_{i-1})}$;
    $v_5 := \dfrac{1 - e^{-2a\cdot(t_i - t_{i-1})}}{2a}$;

    $r(t_i)^{\pm\Delta_\sigma} := r(t_{i-1}) \cdot v_4 + v_2 - v_1 \cdot v_4 + \sigma^{bump} \sqrt{v_5} \cdot v_3$;
$\dfrac{\partial r(t_i)}{\partial \sigma_{hw}} := \dfrac{r(t_i)^{+\Delta_\sigma} - r(t_i)^{-\Delta_\sigma}}{2\Delta_\sigma}$;     ▷ F.D. approx of the derivative.
```

3.2 Complex-Step Derivative Approximation

Sources: Martins *et al.* (2003).

The complex-step estimate can be thought of as a relative to the finite difference estimate, for which second order accuracy and increased robustness can be achieved while eliminating the cancellation error. The implementation requires a minimal amount of development effort and in return reduces the computational cost compared to bumping.

Despite the obvious advantage of eliminating the cancellation error, this method is not really used within the field of financial mathematics, and some might not even be familiar with the technique. The estimate can be derived either using Taylor expansion, as in Homescu (2011), or using logical reasoning, as in Martins *et al.* (2003, sec. 2.1). Here, the latter is repeated for completeness.

Let $F(\theta)$ be shorthand notation for $F(\mathbf{x}(\theta))$ where θ for ease of notation is given as a scalar. Let furthermore F^{cplx} be a complex-valued function of complex variables, then F^{cplx} can by definition be written as:

$$F^{cplx}\left(\theta^{cplx}\right) = F^{real}\left(\theta^{cplx}\right) + iF^{imag}\left(\theta^{cplx}\right), \qquad \theta^{cplx} = \theta^{real} + i\theta^{imag},$$

where F^{real} and F^{imag} both are real-valued functions. Assume that $F^{imag}\left(\theta^{cplx}\right) = 0$ for complex variables with $\theta^{imag} = 0$ such that $F(\theta)$ can be represented using this function.

Now, if F^{cplx} is complex differentiable the *Cauchy-Riemann* equations reads:

$$\frac{\partial F^{real}}{\partial \theta^{real}} = \frac{\partial F^{imag}}{\partial \theta^{imag}} \qquad \frac{\partial F^{real}}{\partial \theta^{imag}} = -\frac{\partial F^{imag}}{\partial \theta^{real}}.$$

The first of these equations can be rewritten using the *definition of a derivative* for the RHS:

$$\frac{\partial F^{real}}{\partial \theta^{real}} = \lim_{\varepsilon \to 0} \frac{F^{imag}\left(\theta^{real} + i\left(\theta^{imag} + \varepsilon\right)\right) - F^{\ imag}\left(\theta^{real} + i\theta^{imag}\right)}{\varepsilon},$$

where ε is a real number. Here, $\theta^{imag} = 0$ by definition as θ^{cplx} is a representation of the real-valued θ. Then according to the assumption above $F^{\ imag}\left(\theta^{real} + i\theta^{imag}\right) = 0$. Thus, the expression can be reduced to:

$$\frac{\partial F}{\partial \theta} \approx \frac{F^{imag}\left(\theta^{real} + i\varepsilon\right)}{\varepsilon} = \frac{\text{Im}\left[F^{cplx}\left(\theta^{real} + i\varepsilon\right)\right]}{\varepsilon}, \tag{25}$$

which is the *complex-step derivative approximation*. As the finite difference approximation in eqn. (23), this method can too be used to determine the derivative for a vector parameter, $\boldsymbol{\theta}$, wrt. one of the elements, θ_i.

The complex-step estimator is implemented by initially identifying all *active* variables throughout the algorithm. Active variables are defined as intermediate variables directly derived from the input parameter of interest, θ_i, or derived from other intermediate variables dependent on this parameter[18]. These active variables are then replaced by their complex counterparts and, if necessary, the corresponding operators are replaced as well. The computation can then be initialized by bumping the imaginary part of the parameter of interest: $\theta_i^{imag} = \varepsilon$. Once the evaluation has terminated, the derivatives, along with the original values, can be retrieved from the output using eqn.(25):

$$F\left(\mathbf{x}\left(\boldsymbol{\theta}\right)\right) = \text{Re}\left[F^{cplx}\left(\mathbf{x}\left(\boldsymbol{\theta}^{real} + i\varepsilon\mathbf{e}_i\right)\right)\right], \qquad \frac{\partial F\left(\mathbf{x}\left(\boldsymbol{\theta}\right)\right)}{\partial \theta_i} = \frac{\text{Im}\left[F^{cplx}\left(\boldsymbol{\theta}^{real} + i\varepsilon\mathbf{e}_i\right)\right]}{\varepsilon}.$$

An example of this application is given in algorithm 4 at the end of this section.

> ...the complex-step method carries the derivative information in the imaginary part of the variables.
>
> (Martins *et al.* (2003, sec.2.4))

The assumption, that F^{cplx} is *complex differentiable*, is ensured in practice by redefinition of essential operators:

- ▷ Relational operators, $\geq > \leq <$, used for conditional statements are in this context defined by comparing only the real part of the complex variables. This ensures that the complex computation follows the same evaluation trace as the original calculation.

- ▷ Minimum and maximum functions relies on the relational operators and must therefore also be redefined such that only the real part of the arguments are compared.

[18] See section 4.2 for further details on active and passive variables.

- ▷ The absolute value are redefined such that the sign of the imaginary part only depends on the sign of the real part[19]:

$$abs(x+iy) = \begin{cases} -x - iy, & \text{if } x < 0 \\ x + iy, & \text{if } x \geq 0. \end{cases}$$

Like for the finite difference approximation, the accuracy of this estimator is determined using Taylor expansion. The derivation in Martins *et al.* (2003, sec. 2.1) show that the estimate is second order accurate, $O\left(\varepsilon^2\right)$, as the central finite difference scheme.

The computational cost of the derivatives, compared to the original calculation, is difficult to put on a general formula like the one in eqn.(24), as it depends on the number of complex operations needed. The numbers in Martins *et al.* (2003, table II) suggests that this cost could potentially be significantly higher than for the finite difference approximation. However, the cost increases linearly with the number of parameters as for the finite difference approximation.

Robustness was also mentioned in the introduction as a characteristic of the complex-step approximation. This statement refers to the processing of functions containing a discontinuity point. The finite-difference method gives incorrect estimates if the point of evaluation are within a Δ_θ-distance of the discontinuity location. The Complex-step method, on the other hand, is correct up to the discontinuity location and even at the discontinuity returns a valid one-sided derivative, see Martins *et al.* (2003, sec. 2.3).

3.2.1 Working the method

- ▷ **The bump-size** ε - the size of the imaginary bump is what really sets this method apart from the finite difference approximation.

> ...the main advantage of this method in comparison with finite differencing: there is no need to compute a given sensitivity repeatedly to find out the optimal step that yields the minimum error in the approximation.
>
> (Martins *et al.* (2003, sec.5))

[19]The argument for this re-definition can be found in Martins *et al.* (2003, sec. 2.3).

Remark that the complex-step derivative in eqn. (25) does not contain a difference and is therefore not subject to cancellation errors. Thus, the truncation error can be limited to a minimum by choosing the bump-size, ε, sufficiently small. There is a lower limit though, as ε must be big enough to be represented by a finite-precision double.

In the following computations the imaginary step size is set to the commonly used constant $\varepsilon = 10^{-20}$.

▷ **Implementation** - the development effort regarding the implementation is limited to changing the data type for all *active* variables and if necessary the corresponding operators listed above.

As mentioned above, *Active* variables are defined as intermediate variables directly derived from the input parameter of interest, θ, or derived from other intermediate variables dependent on this parameter. *Passive* variables are intermediate variables which are independent of the parameter of interest.

One could be tempted implement the method by brute force, i.e. to change all doubles in the algorithm to complex counterparts. But, as complex operations are more expensive than regular, this would induce an increase in computational cost.

Algorithm 4: CSDA of 1-step simulation of the short rate.

```
input: $\sigma_{hw}$, $a_{hw}$,              ▷ Hull White model parameters,
       $\{r^*_{cont}\}$,                     ▷ zero coupon rates,
       $r(t_{i-1})$,         ▷ Short rate at the previous time $t_{i-1}$.
       $\varepsilon$,                                  ▷ Bump size.
       $\alpha(\cdot)$,                                ▷ Sub-function.

Set $\sigma^{cplx}_{hw} = \sigma_{hw} + i\varepsilon$.
  ▷ Set active variable to a complex number with bumped
    imaginary part.

$v^{cplx}_1 := \alpha\left(t_{i-1}, \sigma^{cplx}_{hw},\ a_{hw}, \{r^*_{cont}\}\right);$
$v^{cplx}_2 := \alpha\left(t_i, \sigma^{cplx}_{hw},\ a_{hw}, \{r^*_{cont}\}\right);$

▷ Passive, not complex.
  $v_3 := \mathcal{N}(0,1);$
  $v_4 := e^{-a\cdot(t_i - t_{i-1})};$
  $v_5 := \dfrac{1 - e^{-2a\cdot(t_i - t_{i-1})}}{2a};$

$r(t_i)^{cplx} := r(t_{i-1}) \cdot v_4 + v_2 - v_1 \cdot v_4 + \sigma^{cplx}_{hw} \sqrt{v_5} \cdot v_3;$

$\dfrac{\partial r(t_i)}{\partial \sigma_{hw}} := \dfrac{\mathrm{Im}\left[r(t_i)^{cplx}\right]}{\varepsilon}.$                ▷ CSDA
```

3.3 Algorithmic Derivatives

Sources: Griewank & Walther (2008, ch. 2).

Algorithmic differentiation takes a function, F, formulated as an algorithm as starting point for the evaluation of the derivatives. This is corner stone of algorithmic differentiation, it cannot be described from an isolated mathematical viewpoint, one must take the implementation details into account simultaneously.

An *algorithm* is defined as a *recipe*, specifying the explicit steps involved in evaluating the formula for F as a computer program, see algorithm 2 for an example. Translating a formula into an algorithm involves turning it *into a sequence of assignments that must be executed in a certain order*, as Griewank & Walther phrases it. Let the sequence of intermediate variables for the evaluation of F be given by:

$$[\underbrace{v_{1-n}, \ldots, v_0}_{\mathbf{x}}, v_1, v_2, \ldots, v_{l-m-1}, v_{l-m}, \underbrace{v_{l-m+1}, \ldots, v_l}_{\mathbf{y}}]$$

where $\mathbf{x} \in \mathbb{R}^n$ is the vector of input variables and $\mathbf{y} \in \mathbb{R}^m$ is the vector of output variables.

All computer programs are essentially build from a composition of *elemental functions* such as the basic arithmetic operations: $+$, $-$, $*$, $\backslash$, and the intrinsic functions: $\exp(\cdot)$, $\log(\cdot), \mathrm{sqrt}(\cdot)$, etc. Assuming that F is formulated as an algorithm is therefore equivalent to assuming that F can be decomposed into a sequence of elemental functions. Let therefore each of the l intermediate variables $\{v_i\}_{i=1,\ldots,l}$ be calculated using elemental functions φ_i:

$$v_i = \varphi_i(v_j)_{j \prec i} = \varphi_i(\mathbf{u}_i), \qquad \mathbf{u}_i \equiv \left(v_j\right)_{j \prec i} \in \mathbb{R}^{n_i}, \qquad \text{for } i = 1, \ldots, l,$$

where $j \prec i$ is the precedence relation, stating that v_j is an input to the elemental function φ_i, in other words, that the variable v_i depends directly on v_j. Remark, the calculation is assumed to be *sequential* and $j < i$ must therefore hold.

The complete algorithm, or evaluation procedure, for function F is outlined in table 5. The first row corresponds to the input arguments being *loaded* into a set of intermediate variables. The middle row consists of the functionality in F, divided into a sequence of elemental functions. The last row corresponds to the results being *unloaded* to a set of output variables.

In articles such as Giles (2007), the algorithmic differentiation techniques is introduced from a *vectorized* perspective. This notation turns out quite useful when

v_{i-n}	$= x_i$	$i = 1, \ldots, n.$
v_i	$= \varphi_i(v_j)_{j \prec i} = \varphi_i(\mathbf{u}_i)$	$i = 1, \ldots, l.$
y_{m-i}	$= v_{l-i}$	$i = m-1, \ldots, 0.$

Table 5: General Evaluation Procedure, Griewank & Walther (2008, table 2.2).

introducing the application of the chain rule in *forward-* and *reverse-* mode. Let therefore the *state transformation*, $\mathbf{\Phi}_i$, associated with φ_i be given by:

$$\mathbf{v}_i = \mathbf{\Phi}_i(\mathbf{v}_{i-1}), \qquad \mathbf{\Phi}_i : \mathbb{R}^{n+l} \mapsto \mathbb{R}^{n+l}$$
$$\mathbf{v}_i \equiv \left(v_{1-n}, \ldots, v_i, 0, \ldots, 0\right)^{\intercal} \in \mathbb{R}^{n+l} \qquad \text{for } i = 1, \ldots, l.$$

Here, $\mathbf{\Phi}_i$ sets v_i to $\varphi_i(\mathbf{u}_i)$ while keeping all input values, $v_1 - n, \ldots, v_0$, and all previously calculated variables, $v_1, \ldots, v_{i-1}$, unchanged.

The function F can now be expressed as the composition of all state transformations:

$$F(\mathbf{x}) = \mathbf{Q}_m \mathbf{\Phi}_l \circ \mathbf{\Phi}_{l-1} \circ \cdots \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1 \left(\mathbf{P}_n^{\intercal} \mathbf{x}\right), \tag{26}$$

where $\mathbf{P}_n = \left(\mathbf{I}_{n \times n}, 0, \ldots, 0\right) \in \mathbb{R}^{n \times (n+l)}$ maps the n input values to a $(n+l)$-vector and $\mathbf{Q}_m = \left(0, \ldots, 0, \mathbf{I}_{m \times l}\right) \in \mathbb{R}^{n \times (n+l)}$ extracts the m output values.

The elemental functions, φ_i, associated with each of the state transformations, have well known derivatives qua their definition. These are determined using the elementary rules of differentiation. The derivative of the i'th state transformation - the i'th *state Jacobian* - can thus be evaluated as:

$$\mathbf{A}_i \equiv \mathbf{\Phi}_i'(\mathbf{v}_{i-1}) = \begin{pmatrix} 1 & 0 & \cdots & 0 & \cdot & \cdot & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdot & \cdot & \cdots & 0 \\ 0 & 0 & \cdots & 1 & \cdot & \cdot & \cdots & 0 \\ c_{i,1-n} & c_{i,2-n} & \cdots & c_{i,i-1} & 0 & \cdot & \cdots & 0 \\ 0 & 0 & \cdots & 0 & \cdot & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdot & \cdot & \cdot & \cdots & 1 \end{pmatrix} \in \mathbb{R}^{(n+l) \times (n+l)}, \tag{27}$$

where c_{ij} are the *elemental partials*:

$$c_{ij} = \begin{cases} \frac{\partial \varphi_i(\mathbf{u}_i)}{\partial v_j}, & j \prec i, \\ 0 & \text{otherwise.} \end{cases}$$

The Jacobian given in eqn. (22) can now be reformulated in terms of the state Jacobians by applying the chain rule to the decomposed representation in (26):

$$F'(\mathbf{x}) = \mathbf{Q}_m \mathbf{A}_l \mathbf{A}_{l-1} \dots \mathbf{A}_2 \mathbf{A}_1 \mathbf{P}_n^\intercal. \tag{28}$$

During the evaluation, the chain rule is not applied to symbolic expressions but to actual numerical values, as the elemental partials are evaluated instantly as any other intermediate variable in the program. This is the main difference between algorithmic- and symbolic differentiation, as Griewank & Walther (2008) phrase it in the quote given below.

> The fact that the elemental partials $c_{i,j}$ can be evaluated as real numbers at the current point before they enter into the multiplication prescribed by the chain rule is a key distinction from fully symbolic differentiation.
>
> (Griewank & Walther (2008, p.33))

In terms of accuracy, the algorithmic differentiation method differs substantially from the numerical finite difference method presented in section 3.1. The algorithmic derivatives are *analytical down to machine precision*, which means that the estimates are not subject to truncation- or cancellation errors. But the application of the chain rule to floating point numbers causes a small roundoff error.

3.3.1 Forward Mode

The *forward mode* of algorithmic differentiation corresponds the *intuitive*, or *classical*, way of applying the chain-rule forward through the calculation. The forward derivatives are given as *tangents* where the derivative of each element in the output vector is determined wrt. a parameter of interest, θ_i. In this section the forward mode is derived using the *vectorized* presentation which is afterwards unwounded as an evaluation procedure. This complicated presentation of a rather simple concept will make more sense when deriving the reverse mode in the following section.

Let $\mathbf{y} = F(\mathbf{x})$ and $F'(\mathbf{x})$ be the Jacobian defined in eqn. (22), then the *forward derivatives* of the output vector, $\mathbf{y}$, wrt. a specific parameter, θ_i, is given by:

$$\dot{\mathbf{y}} = F'(\mathbf{x})\dot{\mathbf{x}} = \left(\frac{\partial F_1}{\partial \theta_i}, \frac{\partial F_2}{\partial \theta_i}, \dots, \frac{\partial F_m}{\partial \theta_i}\right)^\intercal, \qquad \dot{\mathbf{x}} \in \mathbb{R}^n,\ \dot{\mathbf{y}} \in \mathbb{R}^m, \tag{29}$$

where $\dot{\mathbf{y}}$, $\dot{\mathbf{x}}$ is notation for $\frac{\partial \mathbf{y}}{\partial \theta_i}$, $\frac{\partial \mathbf{x}}{\partial \theta_i}$, the latter of which is called the *seed direction*.

Replacing the Jacobian in the expression above with its decomposed representation in eqn. (28) yields:

$$\dot{\mathbf{y}} = \mathbf{Q}_m \mathbf{A}_l \mathbf{A}_{l-1} \dots \mathbf{A}_2 \mathbf{A}_1 \mathbf{P}_n^\intercal \dot{\mathbf{x}}.$$

The vector of forward derivatives can thus be evaluated using the chain-rule starting from the input, $\mathbf{P}_n^\intercal \dot{\mathbf{x}}$, and continuing outwards by adding one transition, $\mathbf{A}_i$, at a time. This calculation flow is illustrated below terms of the state Jacobians:

$$\boxed{\dot{\mathbf{y}} = \mathbf{Q}_m \Bigg(\mathbf{\Phi}_l'(\mathbf{v}_{l-1}) \Bigg(\mathbf{\Phi}_{l-1}'(\mathbf{v}_{l-2}) \Bigg(\dots \Big(\mathbf{\Phi}_2'(\mathbf{v}_1) (\mathbf{\Phi}_1'(\mathbf{v}_0) \, (\mathbf{P}_n^\intercal \dot{\mathbf{x}})) \Big) \Bigg) \Bigg) \Bigg).}$$

Translating this result to an evaluation procedure is done by unwinding the i'th bracket in this expression, this yields the forward derivative of v_i:

$$\dot{\mathbf{v}}_i = \mathbf{\Phi}_i'(\mathbf{v}_{i-1}) \dot{\mathbf{v}}_{i-1} = \sum_{j=1-n}^{i-1} c_{i,j} \dot{v}_{i-1} \quad \Leftrightarrow \dot{v}_i = \sum_{j<i} \frac{\partial \varphi_i(\mathbf{u}_i)}{\partial v_j} \dot{v}_j, \qquad i = 1, \dots, l.$$

Thus, in order to evaluate $\dot{v}_i$ all inputs to the i'th assignment, $\mathbf{u}_i$, and the previous derivatives, $\dot{v}_j$, must already have been calculated. The values v_i and $\dot{v}_i$ must therefore be calculated at the same point in the evaluation procedure as described in table 6. Remark that if overwriting of variables is allowed, the calculation of the derivatives must be implemented *before* the original assignment. An example of this forward procedure is given in algorithm 5.

$[v_{i-n}, \dot{v}_{i-n}]$	$= [x_i, \dot{x}_i]$	$i = 1, \dots, n.$
$\dot{v}_i$ v_i	$\sum_{j<i} \frac{\partial \varphi_i(\mathbf{u}_i)}{\partial v_j} \dot{v}_j$ $\varphi_i(\mathbf{u}_i)$	$i = 1, \dots, l.$
$[y_{m-i}, \dot{y}_{m-i}]$	$= [v_{l-i}, \dot{v}_{l-i}]$	$i = m-1, \dots, 0.$

Table 6: Forward evaluation procedure with overwrites, Griewank & Walther (2008, table 3.4).

The length of the *forward evaluation procedure* is approximately twice as long as the original procedure, since a derivative has been added for each original assignment. The computational cost for evaluating the forward derivatives of output vector, $\mathbf{y}$, wrt. a scalar parameter, θ_i, is therefore given by[20]:

$$TIME\,\{F(\mathbf{x}); F'(\mathbf{x})\dot{\mathbf{x}}\} \leq \omega_{forward} \cdot TIME\,\{F(\mathbf{x})\}\,, \qquad \omega_{forward} \in [2; 5/2]. \tag{30}$$

The forward derivative wrt. a vector of p parameters, $\boldsymbol{\theta}$, are calculated for each element separately. This is equivalent to replacing the seed vector, $\dot{\mathbf{x}}$, in eqn. (29) with a $n \times p$-matrix, $\dot{\mathbf{X}}$, where the i'th column contains the seed direction corresponding to θ_i:

$$\dot{\mathbf{Y}} = F'(\mathbf{x})\dot{\mathbf{X}}, \qquad \dot{\mathbf{Y}} \in \mathbb{R}^{m \times p},\ \dot{\mathbf{X}} \in \mathbb{R}^{n \times p}, \tag{31}$$

then forward derivative wrt. θ_i is given by the i'th column of $\dot{\mathbf{Y}}$. Hence, the computational cost increases linearly with the number of parameters p:

$$TIME\left\{F(\mathbf{x}); F'(\mathbf{x})\dot{\mathbf{X}}\right\} \leq \omega^{p}_{fwd} \cdot TIME\,\{F(\mathbf{x})\}\,, \quad \omega^{p}_{fwd} \in [1+p; 1+1.5p]. \tag{32}$$

To summarize, this is a more involved way of introducing the forward mode which is basically a systematic use of the chain-rule, continually applied to each assignment in the evaluation procedure. Here, the individual derivatives are determined as numerical values, rather than symbolic expressions, using the intermediate variables already calculated in the original statements.

Algorithm 5: Forward derivative of 1-step simulation of the short rate.

```
input: $\sigma_{hw}$, $a_{hw}$,                ▷ Hull White model parameters,
       $\dot{\sigma}_{hw}$,                     ▷ Seed direction for active parameter,
       $\{r^*_{cont}\}$,                        ▷ zero coupon rates,
       $r(t_{i-1})$,                            ▷ Short rate at the previous time $t_{i-1}$.
       $\Delta^{(imag)}_{\sigma}$,              ▷ Bump size.
       $\boldsymbol{\alpha}(\cdot)$,            ▷ Sub-function.

▷ Active functions.
  $[v_1, \dot{v}_1] := \boldsymbol{\alpha}\left(t_{i-1}, \sigma_{hw}, \dot{\sigma}_{hw},\ a_{hw}, \{r^*_{cont}\}\right)$;
  $[v_2, \dot{v}_2] := \boldsymbol{\alpha}\left(t_i, \sigma_{hw}, \dot{\sigma}_{hw},\ a_{hw}, \{r^*_{cont}\}\right)$;

▷ Passive, no derivatives determined.
  $v_3 := \mathcal{N}(0,1)$;
  $v_4 := e^{-a\cdot(t_i - t_{i-1})}$;
  $v_5 := \dfrac{1 - e^{-2a\cdot(t_i - t_{i-1})}}{2a}$;

$\dot{r}(t_i) := \dot{v}_2 - \dot{v}_1 \cdot v_4 + \sigma_{hw}\sqrt{v_5} \cdot v_3$;          ▷ Forward derivative
$r(t_i) := r(t_{i-1}) \cdot v_4 + v_2 - v_1 \cdot v_4 + \sigma_{hw}\sqrt{v_5} \cdot v_3$;
```

[20] See Griewank & Walther (2008, sec. 4.5) for details on the derivation.

3.3.2 Adjoint Mode

The adjoint, or *reverse*, mode of algorithmic differentiation consists of applying the chain-rule *backwards* through the calculation. The adjoint derivatives are given as *gradients* where the derivative of an output value of interest, Π_i, wrt. each element in the input vector is determined. The following derivation of this adjoint mode is, contrary to the forward mode, quite unintuitive.

> The learning curve is steep and it can take considerable time to get comfortable with the approach even for experienced mathematicians,...the math involved in AAD is not difficult, but it's not very intuitive... (Mike Giles, Davidson (2015))

Again, let $\mathbf{y} = F(\mathbf{x})$ and $F'(\mathbf{x})$ be the Jacobian defined in eqn. (22), then the *adjoint derivatives* of the output of interest, Π_i, wrt. the input vector $\mathbf{x}$ is given by:

$$\bar{\mathbf{x}}^\top = \bar{\mathbf{y}}^\top F'(\mathbf{x}) \;\Leftrightarrow\; \bar{\mathbf{x}} = F'(\mathbf{x})^\top \bar{\mathbf{y}} = \left(\frac{\partial \Pi_i}{\partial x_1}, \frac{\partial \Pi_i}{\partial x_2}, \dots, \frac{\partial \Pi_i}{\partial x_n}\right)^\top, \quad \bar{\mathbf{x}} \in \mathbb{R}^n,\ \bar{\mathbf{y}} \in \mathbb{R}^m, \tag{33}$$

where $\bar{\mathbf{x}}$, $\bar{\mathbf{y}}$ is notation for $\frac{\partial \Pi_i}{\partial \mathbf{x}}$, $\frac{\partial \Pi_i}{\partial \mathbf{y}}$, the latter of which is called the *weight functional*.

Replacing the Jacobian in the expression above with its decomposed representation in eqn. (28) yields:

$$\bar{\mathbf{x}} = \mathbf{P}_n \mathbf{A}_1^\top \mathbf{A}_2^\top \dots \mathbf{A}_{l-1}^\top \mathbf{A}_l^\top \mathbf{Q}_m^\top \bar{\mathbf{y}}.$$

The vector of adjoint derivatives can thus be evaluated using the chain-rule starting from the output, $\mathbf{Q}_m^\top \bar{\mathbf{y}}$, and continuing outwards by propagating the state transitions, $\mathbf{A}_i$, in reverse order. This calculation flow is illustrated below in terms of the state Jacobians:

$$\bar{\mathbf{x}} = \mathbf{P}_n \Bigg(\mathbf{\Phi}_1'(\mathbf{v}_0)^\top \Bigg(\mathbf{\Phi}_2'(\mathbf{v}_1)^\top \Big(\dots \Big(\mathbf{\Phi}_{l-1}'(\mathbf{v}_{l-2})^\top \big(\mathbf{\Phi}_l'(\mathbf{v}_{l-1})^\top \left(\mathbf{Q}_m^\top \bar{\mathbf{y}} \right) \big) \Big) \Big) \Bigg) \Bigg).$$

Translating this result back to an evaluation procedure is done by unwinding the i'th bracket, this yields the adjoint derivatives $\bar{v}_j$ for $j \prec i$:

$$\bar{\mathbf{v}}_{i-1} = \Phi_i'(\mathbf{v}_{i-1})^\top \, \bar{\mathbf{v}}_i \Leftrightarrow \begin{bmatrix} \bar{v}_{1-n} \\ \bar{v}_{2-n} \\ \vdots \\ \bar{v}_{i-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \bar{v}_{1-n} + c_{i,1-n}\bar{v}_i \\ \bar{v}_{2-n} + c_{i,2-n}\bar{v}_i \\ \vdots \\ \bar{v}_{i-1} + c_{i,i-1}\bar{v}_i \\ 0 \\ \vdots \\ 0 \end{bmatrix} \Leftrightarrow \bar{v}_j += \frac{\partial \varphi_i(\mathbf{u}_i)}{\partial v_j}\bar{v}_i, \quad \forall j \prec i,$$

$$i = 1, \dots, l.$$

Thus, for the i'th original assignment all adjoint derivatives, corresponding to each of its inputs $(v_j)_{j\prec i}$, must be incremented by the amount $\frac{\partial \varphi_i(\mathbf{u}_i)}{\partial v_j}\bar{v}_i$.

In order to evaluate $\bar{\mathbf{v}}_{i-1}$, all inputs to the i'th assignment, $\mathbf{u}_i$, as well as the derivatives corresponding to the $(i+1)$'th assignment, $\bar{\mathbf{v}}_i$, must already have been calculated. The derivatives for the $(i+1)$'th assignment can in turn only be evaluated using the corresponding inputs, $\mathbf{u}_{i+1}$, and so on. The adjoint derivatives can therefore not be evaluated before all intermediate variables, $\mathbf{v}_l$, have been determined through a *forward sweep* of the original calculation.

$\bar{v}_i$	$= 0$	$i = 1-n, \dots, l.$
v_{i-n}	$= x_i$	$i = 1, \dots, n.$
v_i	$= \varphi_i(v_j)_{j\prec i} = \varphi_i(\mathbf{u}_i)$	$i = 1, \dots, l.$
y_{m-i}	$= v_{l-i}$	$i = m-1, \dots, 0.$
$\bar{v}_{l-i}$	$= \bar{y}_{m-i}$	$i = 0, \dots, m-1.$
$\bar{v}_j$	$+= \bar{v}_i \frac{\partial}{\partial v_j}\varphi_i(\mathbf{u}_i)$ for $j \prec i$	$i = l, \dots, 1.$
$\bar{x}_i$	$= \bar{v}_{i-n}$	$i = n, \dots, 1.$

TABLE 7: Adjoint evaluation procedure, Griewank & Walther (2008, table 3.5)

This adjoint evaluation procedure is given in table 7. The first half of the procedure is the *forward sweep*, corresponding to the original evaluation procedure. The second half is the *return sweep*, here the adjoint derivatives are calculated using the intermediate values saved in memory during the forward sweep. An example of the adjoint evaluation procedure is given in algorithm 6 at the end of this section.

The forward evaluation procedure given in table 6 allowed for overwrites of the variables. This is not allowed in the adjoint evaluation procedure given in table 7, as overwrites of the intermediate variables in the forward sweep, would cause these values to be missing for the return sweep.

In practice, overwriting in the original evaluation is a useful technique for bringing down the total amount of memory consumed to a minimum. Overwriting must therefore be handled explicitly by *saving* the intermediate values needed in the return sweep. This will also handle the case, where some of the intermediate values are calculated as local variables in a sub-function, and therefore goes out of scope[21] before the return sweep is initiated.

Griewank & Walther suggest using a *LIFO* - last in first out - structure as *the data are needed on the way back in exactly the same order they where generated on the way forward*, as illustrated by table 7. In this implementation a *stack* of doubles is used for the *recording* of intermediate values generated in the forward sweep. In order for the procedure to record the correct values, one condition for the overwriting in the forward sweep must hold. Assume v_j is an argument of v_i, then v_j may not be overwritten by any v_k for $k \leq i$. This is logical for $k < i$, but for $k = i$ one must be aware that incremental statements like: $\pm =, * =$, which implicitly overwrite v_i, are not allowed.

The adjoint evaluation procedure that accounts for overwrites are given in table 8. Here, the symbol $\odot\!\!\rightarrow$ indicates that the variable on the left-hand side is being *pushed* onto the stack, while $\leftarrow\!\!\odot$ indicates that a value is being *popped* off the stack and assigned to the variable on the left-hand side.

The adjoint evaluation procedure given in table 8 differs from the one suggested by Griewank & Walther. Here, the values of the arguments, $\mathbf{u}_i$, are stored before an assignment, whereas Griewank & Walther instead store the prevalue, v_i, in order to save memory. In this implementation assignments are allowed to be evaluated using sub-functions where the local variables will go out of scope, as mentioned above. Hence, this distinction ensures that all intermediate values can be retrieved in the return sweep.

[21] Local variables in a function f are only *alive* while f is being evaluated. Once f terminates the variables are deleted.

$\bar{v}_i$	$= 0$	$i = 1, \dots, l.$
$[v_{i-n}, \bar{v}_{i-n}]$	$= [x_i, \bar{x}_i]$	$i = 1, \dots, n.$
$\mathbf{u}_i$	$\odot\!\!\rightarrow$	$i = 1, \dots, l.$
v_i	$= \varphi_i(\mathbf{u}_i)$	
y_{m-i}	$= v_{l-i}$	$i = m-1, \dots, 0.$
$\bar{v}_{l-i}$	$\bar{y}_{m-i}$	$i = 0, \dots, m-1.$
$\mathbf{u}_i$	$\leftarrow\!\!\odot$	
$\bar{v}_j$	$+ = \bar{v}_i \frac{\partial}{\partial v_j} \varphi_i(\mathbf{u}_i) \quad \forall j \prec i$	$i = l, \dots, 1.$
$\bar{v}_i$	$= 0$	
$\bar{x}_i$	$= \bar{v}_{i-n}$	$i = n, \dots, 1.$

Table 8: Adjoint evaluation procedure with overwrites, Griewank & Walther (2008, table 4.4 - v_i in line 3, 6 changed to u_i.).

In practice, the structure of the evaluation procedure can be taken into account to save space on the recorded stack, as discussed below in section 4.3.

The length of the *backward evaluation procedure* is approximately twice as long as the original procedure, as this is propagated once forward and once in reverse. Taking the memory handling into account increases this further. The total computational cost for evaluating the adjoint derivatives of a scalar output of interest, Π_i, can thus be shown to satisfy the *cheap gradient principle* given in Griewank & Walther (2008, eqn. (3.14)):

$$TIME\,\{F(\mathbf{x}); \bar{\mathbf{y}}^\intercal F'(\mathbf{x})\} \leq \omega_{adjoint} \cdot TIME\,\{F(\mathbf{x})\}, \qquad \omega_{adjoint} \in [3; 4]. \tag{34}$$

Remark that this computational cost is independent of the number of elements in the input vector, $\mathbf{x}$. This allows for a remarkable reduction in computational costs for evaluation procedures, with a large amounts of input arguments which is often the case in finance for interest rate products.

The adjoint derivatives wrt. a vector of output values of interest, $\mathbf{\Pi} \in \mathbb{R}^q$, are calculated for each element separately. In practice, this is equivalent to replacing the weight functional vector, $\bar{\mathbf{y}}$, in eqn. (33) with a $m \times q$-matrix, $\bar{\mathbf{Y}}$, where the i'th column contains the weight functional corresponding to the i'th final output of interest, Π_i:

$$\bar{\mathbf{X}}^\intercal = \bar{\mathbf{Y}}^\intercal F'(\mathbf{x}), \qquad \bar{\mathbf{X}} \in \mathbb{R}^{n \times q},\ \bar{\mathbf{Y}} \in \mathbb{R}^{m \times q}, \tag{35}$$

then the adjoint derivative of Π_i is given by the i'th column of $\bar{\mathbf{X}}$. Hence, the computational cost increases linearly with the number of output values q:

$$TIME\left\{F(\mathbf{x}); \bar{\mathbf{Y}}^{\intercal} F'(\mathbf{x})\right\} \leq \omega^q_{adj} \cdot TIME\left\{F(\mathbf{x})\right\}, \; \omega^q_{adj} \in [1 + 2q; 1.5 + 2.5q]. \quad (36)$$

To summarize, using the chain-rule in reverse is a quite unintuitive way of calculating the derivatives of a function, F. The practical implementation is though not much more complicated than for the forward mode, as long as the recipe given in table 8 is followed rigorously. The challenges arise when the method is applied to a collection of functions, spread across multiple object classes. These implementation issues, and how they can be handled, are further described in section 4.

Algorithm 6: Adjoint derivative of 1-step simulation of the short rate.

```
input: $\sigma_{hw}$, $a_{hw}$,                ▹ Hull White model parameters,
       $\{r^*_{cont}\}$,                        ▹ zero coupon rates,
       $r(t_{i-1})$,            ▹ Short rate at the previous time $t_{i-1}$.
       $\bar{r}(t_i)$,              ▹ Weight functional for active output,
       $\alpha(\cdot)$,                  ▹ Sub-function, forward sweep.
       $\bar{\alpha}(\cdot)$,                 ▹ Sub-function, return sweep.

▹ Forward Sweep
  $v_1 := \alpha\left(t_{i-1}, \sigma_{hw}, a_{hw}, \{r^*_{cont}\}\right);$
  $v_2 := \alpha\left(t_i, \sigma_{hw}, a_{hw}, \{r^*_{cont}\}\right);$

  $v_3 := \mathcal{N}(0,1);$
  $v_4 := e^{-a\cdot(t_i - t_{i-1})};$
  $v_5 := \dfrac{1 - e^{-2a\cdot(t_i - t_{i-1})}}{2a};$
  $r(t_i) := r(t_{i-1}) \cdot v_4 + v_2 - v_1 \cdot v_4 + \sigma_{hw}\sqrt{v_5} \cdot v_3;$

▹ Return Sweep
  $\bar{v}_1 := \bar{v}_1 + \bar{r}(t_i) \cdot (-v_4);$
  $\bar{v}_2 := \bar{v}_2 + \bar{r}(t_i);$
  $\bar{\sigma}_{hw} := \bar{\sigma}_{hw} + \bar{r}(t_i)\sqrt{v_5} \cdot v_3;$

  $\bar{\sigma}_{hw} := \bar{\sigma}_{hw} + \bar{\alpha}\left(\bar{v}_2\right);$
  $\bar{\sigma}_{hw} := \bar{\sigma}_{hw} + \bar{\alpha}\left(\bar{v}_1\right);$         ▹ Adjoint derivative.
```

3.3.3 WORKING THE METHOD

In practice, the individual assignments are not expanded to a series single elemental functions, but they are assumed to be sufficiently simple such that the derivatives are straight forward to determined by the *elementary rules of differentiation*.

- ▷ **Implementation : Forward** - the evaluation procedure in table 6 are implemented by adding a forward derivative, $\dot{v}_i$, in connection with each of the *active* variables, v_i, defined in section 3.2.1.

- ▷ **Implementation : Adjoint** - the implementation of the forward sweep in table 8 is given by the original evaluation procedure, with intermediate values being pushed onto the record if they are included in the calculation of an *active* variable. Later, in section 4.3, rules-of-thumb are given on how to save space on this record.

  The return sweep is implemented by traversing the original calculation in reverse order, while popping *active* variables, v_i, off the record as described in table 8.

  Special attention must be paid in order to restore the original control flow in the reverse sweep. For instance must all variables entering conditional statements be pushed onto the record *after* the statement has terminated.

4 IMPLEMENTATION

The methods are in this setup implemented using C++11. The primary reason for this choice of language is the selection of automatic differentiation tools available. Further details on this topic are given below in section 4.5. The secondary advantage is the object orientation available for C++ opposed to other languages popular for financial research such as MATLAB.

This section begins with a description and a motivation of the object oriented design chosen for the CVA solution. The challenges regarding the practical implementation of the three methods: CSDA, forward- and adjoint mode, are covered afterwards. The finite difference approximation is, as mentioned in section 3.1, implemented by running the original calculation twice for bumped inputs. The only challenge connected to the implementation of this method is thus choosing the bump size and it has therefore been left out of the discussion below.

4.1 PROGRAM DESIGN

The implementation of the CVA calculation is, as mentioned above, based on object oriented design. The calculation has therefore been split into a set of intuitive and well defined computational blocks. These are implemented by *classes*, each managing the functionality and data corresponding to their area of responsibility. The overall structure of the object oriented design for the CVA calculation is given in figure 4 and consists of classes:

- ▷ `IRS` : class representing the interest rate swap product. Objects for the short rate- and the intensity rate model are given as data members. The member functions include the time t value of the receiver swap and the time $t = 0$ value of the corresponding CVA approximation. The latter could with advantage be moved outside this class if the framework were expanded to handle a portfolio of products. Details on the functionality for this class are given in figure 13.

- ▷ `ShortRateModels, IntensityModel` : abstract classes containing interfaces for the models used in the CVA calculation. Adding interfaces allows for easy replacement of the models with other, perhaps more sophisticated, ones as long as they implement the virtual functions defined in the corresponding interface.

- ▷ `Hull_White` : concrete class representing the Hull White model and implementing the abstract class for short rate model. Object for the zero coupon

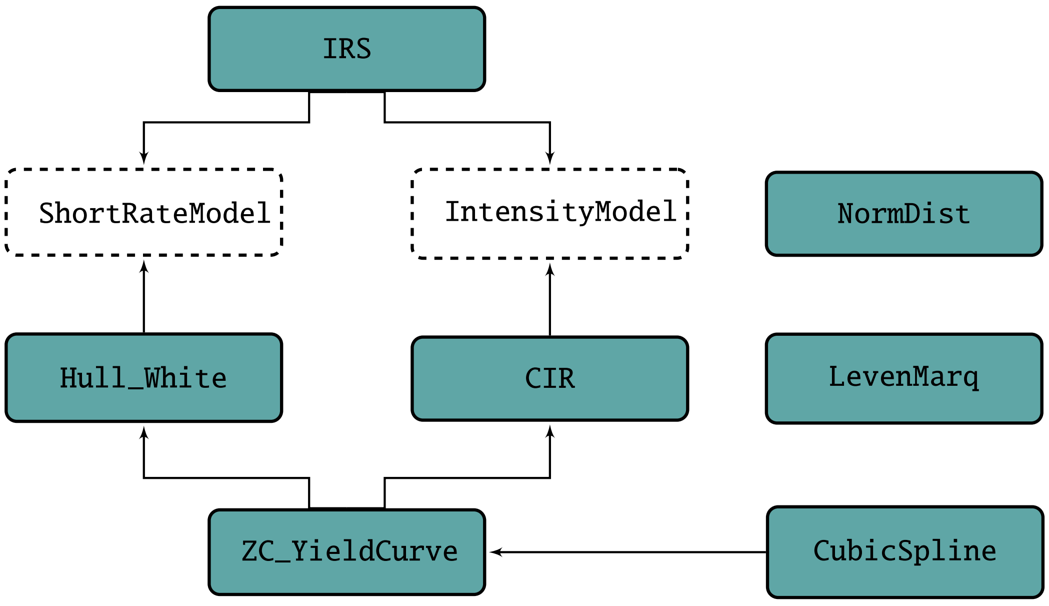


Figure 4: Object oriented design for the CVA setup.

yield curve is given as a data member. The member functions include: calibration, the time t price of a zero coupon bond and the 1-step simulation scheme for the short rate process. Details on the functionality for this class are given in figure 14.

▷ `CIR` : concrete class representing the CIR model and implementing the abstract class for the intensity model. Object for the zero coupon yield curve is given as a data member. The member functions include calibration and the time $t = 0$ probability of default within a given time-bucket. Details on the functionality for this class are given in figure 15.

▷ `ZC_YieldCurve` : class representing the zero coupon yield curve. Object for the natural cubic spline, containing the continuously compounded zero coupon rates, are given as a data member. The member functions include: calibration, the time $t = 0$ price of a zero coupon bond and the instantaneous forward rate. Details on the functionality for this class are given in figure 16.

▷ `NormDist, LevenMarq, CubicSpline` : collection of classes containing generic algorithms used in various parts of the CVA solution. Details on the functionality for these are given in figures 17, 18 and 19, respectively.

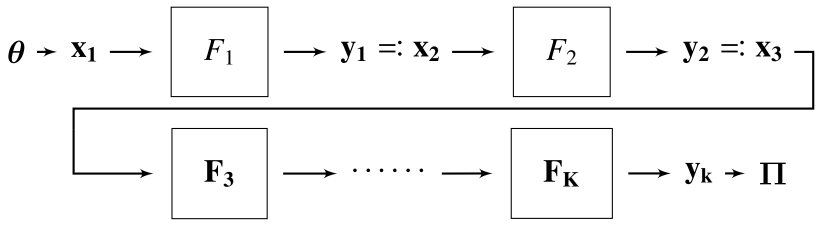
Notice that these classes and their corresponding member functionality have been re-implemented, instead of importing an optimized library, in order to hand-code the CSDA and the algorithmic derivatives. Compared to library routines, this will slow down the computations, but it has been tolerated here due to the academic purpose of this CVA solution.

As an alternative one could instead have used one of the simpler open source libraries, such as AlgLib[22], and applied an automatic differentiation tool on these specific functions.

4.2 Active and passive variables

Sections 3.2.1 and 3.3.3 went over specific implementation details for each of the respective methods. Common to all methods were the initial process of identifying the *active* and *passive* variables.

In order to identify these two types of variables, it is convenient to have a complete outline of the functions and their corresponding in- and output variables. Assume therefore that a given algorithm consists of K functions. These can then be ordered according to the overall *computational flow* of the algorithm as illustrated in figure 5. In this extension of the single-function-view in figure 3, functions are allowed to be called multiple times and the $F_i's$ are therefore not unique.



$\boldsymbol{\theta} \to \mathbf{x_1} \to F_1 \to \mathbf{y_1} =: \mathbf{x_2} \to F_2 \to \mathbf{y_2} =: \mathbf{x_3}$
$\to \mathbf{F_3} \to \cdots\cdots \to \mathbf{F_K} \to \mathbf{y_k} \to \boldsymbol{\Pi}$

Figure 5: Structure of an algorithm with vector of parameters, $\boldsymbol{\theta}$, as input and vector of values, $\boldsymbol{\Pi}$, as output. The functions F_i are not assumed to be unique, as a given function can be called multiple times.

This computational flow is not only useful for determining the active- and passive variables, it is also very useful as a *road-map* for propagating through the original algorithm, implementing and testing the derivatives function-by-function.

Returning to the *active* variables, recall that these were defined as intermediate variables directly derived from the input parameter(s) of interest[23], $\boldsymbol{\theta}$, or derived from other intermediate variables dependent on this parameter. The remaining

[22]AlgLib can be found at http://www.alglib.net/.

[23]The parameter of interest is a scalar for the CSDA- and forward method while it can be a vector for the adjoint mode.

intermediate variables are then independent of $\boldsymbol{\theta}$ and were termed *passive*. An example of this can be seen in figure 6 for the Hull-White ZCB price function given in eqn. (10).

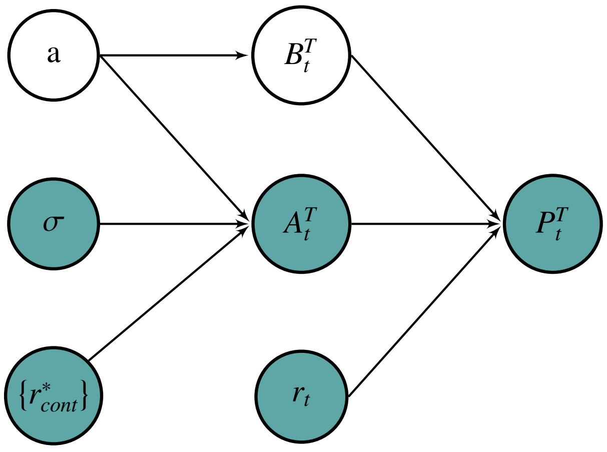


Figure 6: Computational graph for the ZCB price function, (10), where the inputs: t, T, have been suppressed. The inputs of interest are given by the volatility parameter, σ_{hw}, and the collection of zero coupon rates, $\{r^*_{cont}\}$. Colored nodes indicate *active* variables while the white nodes are *passive*.

Assuming that the original algorithm has already been implemented, the active variables can then be identified:

- ▷ Start : identify the parameter(s) of interest in the parameter vector, $\boldsymbol{\theta}$.
- ▷ For $k = 1, \ldots, K$: identify all active intermediate variables occuring in F_k starting from, and including, $\mathbf{x_k}$ up till, and including, $\mathbf{y_k}$.
- ▷ End : identify the active output variables in the final output vector $\boldsymbol{\Pi}$.

The active variables for the CVA calculation are marked in the comments of the enclosed source code for modules: CSDA, forward- and adjoint mode.

4.3 Recording

The computational cost function $TIME$, used in section 3 to describe the relative run time for the methods, does not take the memory consumption into account. But memory is, just as flops[24], a limited resource which can slow down execution time significantly.

[24]Floating point operations per second.

...the speed of processors (measured in floating-point operations per second, or FLOPS) is more than adequate. They can work so fast that FLOPS are virtually free. The problem is that they are not capable of operating to their capacity because they are starved for local access to memory...

(The Kavli Foundation)

This is especially a challenge when it comes to the adjoint mode for the CVA calculation, as the memory consumption for the *records* here depends on the number of simulation paths. Hence, the *computation may become memory bound* as Griewank & Walther (2008) points out. Roughly speaking, this means that the CPU might be available for processing the next operation, but it has to wait for the variables to be fetched from memory.

The following *rule-of-thumbs* have therefore been applied for the implementation of the adjoint mode to cut down the overall memory consumption.

- MULTIPLE USAGE : if the intermediate value is included in multiple calculations, it is not pushed onto the record until right before the last appearance.
- LINEAR OPERATION : if a variable only appears in linear operations, its value will not enter the derivative operations during the return sweep. The value is therefore not pushed onto the record.
- SUB-FUNCTIONS : in this framework all functions are themselves responsible for the bookkeeping of their intermediate variables. Hence, if a value enters the calculation of an *active* variable in the form of an argument to a subfunction, it is not pushed onto the record.
- RE-CALCULATION : in practice it can be more efficient to re-calculate an intermediate variable during the return sweep, than to retrieve it from the record, cf. discussion above.

  A simple case is when a variable can be recalculated using values already - or about to be - pushed onto the record[25].

  On a larger scale, memory management of the records can be conducted using *Checkpointing*[26] This is a systematic technique where chunks of the original algorithm are recalculated during the return sweep. Each chunk is

[25]An example of this can be found in function: `Hull_White_backward::ZcbPutZero`, contained in the source code.

[26]Checkpointing are described in detail in Griewank & Walther (2008, ch.12).

reproduced from a limited set of carefully chosen variables pushed onto the record during the forward sweep.

4.4 Validation Checks

Testing tends to be an overlooked virtue within the field of mathematical finance. But especially in this context, a good testing framework is crucial in order to obtain derivatives of machine precision accuracy. This is especially the case when working with the hand-coded versions, opposed to the automatic software generated versions discussed below in section 4.5.

Assume the derivatives of a function F have been derived using all four methods presented in section 2.7. Here, the function F either represents a single function, F_k, or the overall algorithm given as the composition of all K functions depicted in figure 5. In the later case the input vector is given by the parameter, $\mathbf{x} = \boldsymbol{\theta}$, and the output vector is given by the final output, $\mathbf{y} = \boldsymbol{\Pi}$.

$$\begin{tikzcd} \mathbf{x} \arrow[r] & \boxed{F} \arrow[r] & \mathbf{y} \end{tikzcd}$$

- ▷ The overall *level* of the values produced by CSDA, forward- and adjoint mode are checked by comparison to the finite difference approximation.

  This approximation is ideal as an initial check because it can be implemented without any changes to the original algorithm. The risk of introducing an error to the code is therefore minimal.

  The accuracy of this approximation is on the other hand far from the machine precision obtained by the others, and it can therefore not be used for an elaborate comparison.

- ▷ The CSDA is similar to the forward mode in that both methods propagate forward through the function evaluation, determining the derivatives of vector output, $\mathbf{y}$, according to the seed direction $\dot{\mathbf{x}}$. The corresponding derivatives can therefore easily be checked against each other using the identity given in Giles *et al.* (2008):

  $$\dot{\mathbf{y}} \approx \frac{\mathcal{I}\{F(\mathbf{x} + i\varepsilon\dot{\mathbf{x}})\}}{\varepsilon}, \qquad \varepsilon = 10^{-20}. \tag{37}$$

  This can either be checked for random seed directions $\dot{\mathbf{x}}$ or, as in this setup, for a sequence of unit vectors $\dot{\mathbf{x}} = \{\mathbf{e}_i\}_{i=1,\ldots,n}$, resulting in a comparison of all columns in the Jacobian (22). The latter allows each row of the Jacobian to be compared afterwards to the output of the adjoint method, $\bar{\mathbf{x}}$, likewise called for a sequence of unit vectors $\bar{\mathbf{y}} = \{\mathbf{e}_i\}_{i=1,\ldots,m}$.

▷ An alternative to comparing the complete Jacobian's is defined by the following identity suggested by Mike Giles[27]:

$$\mathrm{tr}\left(\bar{\mathbf{Y}}^{\intercal}\dot{\mathbf{Y}}\right) = \mathrm{tr}\left(\bar{\mathbf{X}}^{\intercal}\dot{\mathbf{X}}\right), \tag{38}$$

where the matrices are defined in eqn. (31) and (35), respectively. This identity should be satisfied for all randomly generated matrices $\dot{\mathbf{X}}$, $\bar{\mathbf{Y}}$.

Here, the code for the CVA calculation has not been vectorized. The derivatives are therefore evaluated for one parameter of interest at a time for the forward mode, and for one final output of interest at a time for the adjoint mode. The identity above can thus be checked by generating random vectors $\dot{\mathbf{x}}$, $\bar{\mathbf{y}}$, after which the LHS can be evaluated using the forward mode:

$$\underbrace{\bar{\mathbf{y}}^{\intercal}}_{\text{random}}\ \underbrace{\dot{\mathbf{y}}}_{\text{calculated}} = \bar{\mathbf{x}}^{\intercal}\ \underbrace{\dot{\mathbf{x}}}_{\text{input}}.$$

and the RHS can be evaluated using the backward mode:

$$\underbrace{\bar{\mathbf{y}}^{\intercal}}_{\text{input}}\ \dot{\mathbf{y}} = \underbrace{\bar{\mathbf{x}}^{\intercal}}_{\text{calculated}}\ \underbrace{\dot{\mathbf{x}}}_{\text{random}}.$$

The identities (37) and (38) both checks the derivatives down to machine precision. Thus, it would be convenient to set up a unit testing framework which automatically flagged, whenever the absolute difference between the sides in either of the identities exceeded a certain threshold. Unfortunately, this threshold depends on the number of floating point operations as each elemental operation introduces an error. There is no automatic tool available for counting the number of floating point operations and it is practically impossible to do by hand. For instance, how many operations does the computer use for evaluating `sqrt` or `exp`?

In practice, one must therefore settle for *eyeballing* the validation checks written to a .csv file. The dependence on the number of floating point operations becomes evident for the calibration functions where the Levenberg-Marquardt solver runs multiple iterations. The relative error of the derivatives compared to the CSDA can be seen in figure7.

[27]To the authors knowledge this identity has not been publish, but was suggested by Mike Giles during a discussion.

The combination of the four methods presented in section 3, thus forms the optimal testing framework presented above. But in real life applications one would be forced to target the testing due to time constraints. Giles *et al.* (2008) suggest to distinguish between cores parts of the functionality and the algorithm as a whole.

4.5 Automatic Differentiation

The practical implementation of the algorithmic methods is to a large extent given by mechanically applying the elementary rules of differentiation to the original code. This process can be automated using a more or less sophisticated software tool.

The sophistication level lies within the AD tool's ability to analyze and exploit the structure of the original code to generate efficient code for the forward- and adjoint mode. Several general-purpose tools are available, as will be discussed below, but special-purpose tools must generally be constructed from scratch as described in Griewank & Walther (2008) sections 6.1 and 6.2.

> ...the best results will be obtained when AD takes advantage of the user's insight into the structure underlying the program, rather than by the blind application of AD to existing code.
> (Griewank & Walther (2008, p.4))

The main advantages of using automatic tools includes the obvious lack of hand-coding all derivatives line-by-line. But also maintaining the code are eased, as it becomes simpler to make adjustments and replace models or products. The downside are an increased computational cost, as the tools are not able to optimize the code to the same extent as if it was derived by hand.

The automatic AD tools can be divided into two main categories:

▷ Source code transformation tools : take the original algorithm as input, and returns the algorithm with additional lines of code that have been automatically generated for the forward- or adjoint mode derivatives. Capriotti (2011) describes how the tool typically breaks each instruction down into a composition of elemental functions for which the derivatives are known. According to Martins *et al.* (2003), the source code returned by such a source transformation tool is greatly enlarged which complicates debugging and future alterations.

Examples of source transformation tools for C++ code is the Tapanade and TAC++ packages.

- ▷ OPERATOR OVERLOADING TOOLS : does not change the original code, instead they provide new data types for the active variables, containing both the original value and the derivative. Along with the data types are a set of redefined (overloaded) operators which evaluate both the original value and the corresponding derivative. In order to apply this tool, the data type must therefore be altered for all active variables. This maintains the readability of the code, but in return all functionality are hidden *under the hood.* According to Capriotti (2011), this lack of transparency ultimately leads to slower execution compared to the transformed code due to the missing compiler optimization.

  Examples of operator overloading tools for C++ code is the ADOL-C and FADBAD++ packages.

In this setup, the automatic tool FADBAD++ has been applied in order to compare the performance with the hand-coded versions. This tool is open source and can be downloaded from the website `http://www.fadbad.com/fadbad.html`.

The implementation of either mode using FADBAD++ is super simple and quite similar to that described in section 3.2.1 for the CSDA method. Hence, the data type for all active variables is changed to `F<double>` for the forward mode and `B<double>` for the backward mode. In addition to this, three header files must be included in the code. After these simple alternations, the derivatives of a given function with respect to its explicitly given arguments can be determined[28].

[28]The syntax for extracting the derivatives can be seen in functions `CVA_noCalibration` and `CVA_Calibration` included in the source code file `CVA_Calculation.cpp`.

5 Results

The C++ code for the CVA price and the corresponding sensitivities have been compiled using clang with optimization flag -O3 and run on a *late 2012* iMac with *3.4 GHz Intel Core i7* processor and *8GB 1600 MHz DDR3* RAM. A short introduction to the overall structure of the enclosed source code can be found in appendix A. Documentation on the identification of active variables, the derivatives calculated, and the handling of the record for the adjoint mode are given in the comments of the source code.

The results presented in this section have been derived for two test cases. The main case consists of calculating the derivatives of the CVA price wrt. parameters, $\boldsymbol{\theta}_1$, given by the 12 zero coupon rates used as input for both the model calibrations and the CVA calculation as illustrated in figure2. The second case is mainly included to compare run times for varying numbers of sensitivities. In this case the dependence between the zero coupon rates and the model parameters have been left out by setting the parameters to pre-calibrated values, giving a total of 16 sensitivities.

$$\boldsymbol{\theta}_1 = \left[\{r^*_{cont}(0,T)\}_{T\in Z^*_1}\right], \qquad \boldsymbol{\theta}_2 = \left[\sigma_{hw}, \lambda_0, a_{cir}, b_{cir}, \{r^*_{cont}(0,T)\}_{T\in Z^*_1}\right],$$

where $Z^*_1 = \{1/365, 1, 2, 3, 4, 5, 7, 10, 15, 20, 25, 30\}$ were defined in section 2.5.1.

The complete dataset can be found in appendix B, and the details regarding the data collection are listed below.

- ▷ The data has been derived for $M = \{5.000, 10.000, 50.000, 100.000\}$ simulation paths. The adjoint mode fails for 150.000 paths due to memory constraints and results have therefore not been obtained for 250.000 paths as in Hansen & Glibstrup (2014). These memory issues for the adjoint mode are discussed in detail later in this section.
- ▷ The data has been calculated using the same set of random variables $Z^{(m)}_i$ for $i = 1, \dots, n$ and $m = 1, \dots, M$ for all methods in order to compare the accuracy of the methods.

5.1 Accuracy

The numerical values produced by the CSDA, forward- and adjoint mode, should in theory be equal in size down to machine precision times a factor dependent on the number of floating points cf. discussion in section 4.4. While the finite difference approximation should give significantly larger errors due to the combination of truncation- and cancellation errors.

The absolute value of the relative error, compared to the CSDA method, are depicted in figure 7. Based on the order of the errors it is safe to assume that all methods have been implemented correctly. The errors for the sensitivities calculated using bumping are as expected significantly larger compared to the forward- and adjoint mode, especially for zero coupon rates associated with large maturities.

Zooming in on the errors for the algorithmic methods gives the graphs in figure 8. Here it can be seen how the errors are larger for the first case where the calibration, and thereby additional floating point operations, are included. For the second case a difference in accuracy becomes visible. This is probably because of the additional floating point operations involved in the recording of the intermediate variables, which have a negative influence on the accuracy for the adjoint mode.

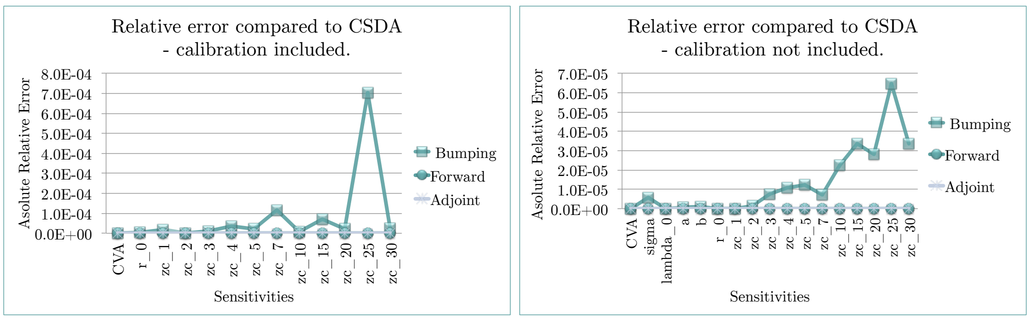


Figure 7: Absolute value of the relative error, compared to the CSDA method, for bumping and both algorithmic modes, calculated using 100.000 simulation paths.

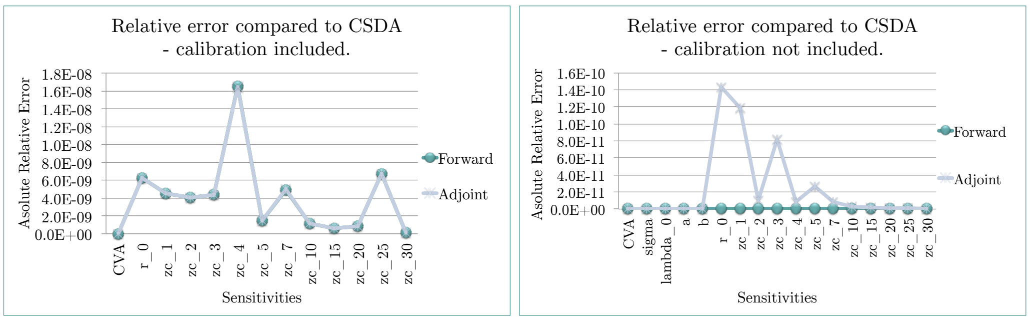


Figure 8: Absolute value of the relative error, compared to the CSDA method, for the forward- and adjoint mode, calculated using 100.000 simulation paths.

5.2 STABILITY

The mean and standard deviation of the CVA price and corresponding sensitivities based on 500 calculations are given in tables 9 and 10. These values have been calculated using the adjoint mode as this method has the smallest computational cost, as will be seen in the following section. The run time for the single CVA calculation, given in table 9, is an exception though, as this measure cannot be extracted from the adjoint computation and therefore has been obtained by an additional run of the original algorithm.

Case 1								
# Sim. paths	5.000		10.000		50.000		100.000	
	Mean	RSD(%)	Mean	RSD(%)	Mean	RSD(%)	Mean	RSD(%)
Run time(s) : CVA_0	2.4537	0.39	4.9095	0.3597	24.6555	0.1670	49.5300	0.3693
Run time(s) : $\frac{\partial CVA_0}{\partial \theta_1}$	4.4335	0.17	11.4519	1.54	190.0406	1.2752	456.3290	15.4092
CVA_0	2.1457	1.83	2.1438	1.36	2.1448	0.5875	2.1460	0.4210
$\frac{\partial CVA_0}{\partial r_0}$	0.2361	2.83	0.2358	2.14	0.2359	0.9251	0.2362	0.6495
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	1.5703	2.61	1.5680	1.97	1.5690	0.8501	1.5706	0.5979
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	2.6889	2.91	2.6848	2.19	2.6869	0.9499	2.6900	0.6664
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	5.4853	2.49	5.4776	1.88	5.4811	0.8111	5.4862	0.5714
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	1.1392	7.87	1.1359	5.90	1.1386	2.5600	1.1425	1.7932
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	13.6154	2.00	13.5986	1.50	13.6047	0.6434	13.6134	0.4581
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-5.1009	2.34	-5.0991	1.71	-5.0961	0.7163	-5.0922	0.5332
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-63.1896	1.60	-63.1686	1.18	-63.1817	0.5148	-63.2105	0.3700
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	35.4903	1.65	35.5296	1.22	35.5037	0.5228	35.4900	0.3745
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	17.2466	3.34	17.3087	2.46	17.2867	1.0708	17.2654	0.7583
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	3.1864	19.90	3.1989	15.00	3.1875	6.5709	3.1587	4.7746
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-142.2900	1.53	-142.2222	1.12	-142.2498	0.4906	-142.3034	0.3491

TABLE 9: Mean and relative standard deviation for samples of size 500 calculated using the adjoint mode for *Case 1*.

Case 2								
# Sim. paths	5.000		10.000		50.000		100.000	
	Mean	RSD(%)	Mean	RSD(%)	Mean	RSD(%)	Mean	RSD(%)
Run time(s) : $\frac{\partial CVA_0}{\partial \boldsymbol{\theta_2}}$	4.4042	0.29	11.2811	1.78	189.0374	1.2223	444.3145	1.5873
CVA_0	2.1429	1.79	2.1461	1.31	2.1457	0.5876	2.1461	0.4182
$\frac{\partial CVA_0}{\partial \sigma_{hw}}$	168.2628	2.65	168.6710	1.95	168.5504	0.8615	168.6283	0.6167
$\frac{\partial CVA_0}{\partial \lambda_0}$	-3.1110	3.15	-3.1183	2.25	-3.1182	1.0218	-3.1208	0.7085
$\frac{\partial CVA_0}{\partial a_{cir}}$	0.2013	7.73	0.2018	5.58	0.2012	2.4417	0.2007	1.7589
$\frac{\partial CVA_0}{\partial b_{cir}}$	9.2729	2.10	9.2921	1.53	9.2861	0.6811	9.2851	0.4911
$\frac{\partial CVA_0}{\partial r_0}$	-0.0362	1.63	-0.0363	1.18	-0.0363	0.5351	-0.0363	0.3795
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	0.0874	1.46	0.0874	1.04	0.0875	0.4815	0.0875	0.3386
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	-0.2196	1.40	-0.2198	0.97	-0.2199	0.4656	-0.2199	0.3236
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	0.7913	1.40	0.7916	0.97	0.7922	0.4663	0.7920	0.3236
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	-2.9453	1.40	-2.9467	0.97	-2.9488	0.4664	-2.9481	0.3236
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	5.8633	1.40	5.8661	0.97	5.8701	0.4664	5.8688	0.3236
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-10.7163	1.40	-10.7213	0.97	-10.7288	0.4664	-10.7263	0.3236
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-47.9467	1.32	-47.9926	0.96	-48.0006	0.4283	-48.0061	0.3070
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	60.8093	1.27	60.8526	0.91	60.8878	0.4159	60.8882	0.2983
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	42.3539	1.44	42.3808	1.05	42.3828	0.4532	42.3889	0.3217
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	30.0097	1.45	30.0211	1.06	30.0181	0.4794	30.0199	0.3351
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-114.6223	1.31	-114.7100	0.96	-114.7244	0.4257	-114.7390	0.3026

Table 10: Mean and relative standard deviation for samples of size 500 using the adjoint mode for *Case 2*.

The first point to notice here is the stability of the run times, the standard deviation for these are small though increasing with the number of simulations. Thus, the run times examined in the remaining section obtained using one sample can be assumed representative.

The validity of these results can be assessed by comparing the CVA_0 price with results obtained in Hansen & Glibstrup (2014, table 4) which are repeated here in table 13 found in appendix B. The values and the standard deviations are roughly equal if the difference in sample size and simulation scheme are taken into account. The run time for a single CVA evaluation are on the other hand not, the C++ implementation are observed to be slower by a factor 10 for 5.000 paths and a factor 15 for 10.000 paths and a factor 19 for 100.000 paths. This increasing difference in run times stems from the vectorization of the MatLab implementation used by Hansen & Glibstrup. Unfortunately, vectorization is not simple to implement in C++ and substantial developer effort would had to be invested in order to get the same speed up.

The mean value of the derivatives naturally differ for the two cases, but for both it can be observed how the sensitivities for the zero coupon rates increase with the years to maturity. Comparing the standard deviations of the sensitivities in the two cases, it becomes evident that these seem remarkably high for the first case where the calibration is included. These high values could stem from the gradient of the cap values and CDS premiums, respectively, used by the Levenberg-Marquardt algorithm in connection with the calibration of the two models. These gradients are calculated using bumping in the original algorithm. Thus, when either of algorithmic modes are applied, this results in the derivative of a bumped derivative which could cause inaccuracies.

In the description of the methods in section 3 emphasis were put on the accuracy of the methods. Comparing the relative errors given in figure 7 and the standard deviations in tables 9 and 10, it can be observed that the errors regarding accuracy are insignificant compared to the errors for the Monte Carlo estimate. Thus, in this case the accuracy is not a deciding factor.

5.3 Run Time

The run times reported here include: initializing the calculation, evaluating the original CVA price and the set of derivatives. For the first case initializing the calculation consists of setting the zero coupon rates to the pre-calibrated values and afterwards calibrating both models. For the second case it consist of setting the zero coupon rates and the parameters for the two models to the pre-calibrated values.

Based on the theoretical run times listed in eqn.'s (24), (30) and (34), bumping is expected to be the computationally most expensive method followed by the CSDA. The algorithmic methods are expected to be cheaper than the first two,

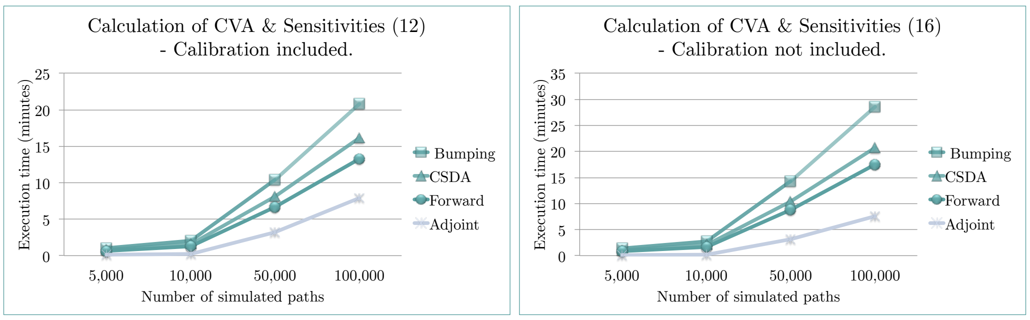


Figure 9: Run times measured in minutes for each of the four methods.

and in this case where the number of inputs greatly exceeds the number of outputs, the adjoint mode is expected to be significantly cheaper than the forward mode.

Figure 9 shows the run time in minutes for each of the number of simulated paths. Here, it can be seen that for 5.000 simulation paths the difference in computational cost between the methods are insignificant. But for numbers of simulation paths above 5.000, the run time for the different methods are ranked as expected and the difference seem to increase with the number of simulation paths. This dependence on the number of simulation paths will explained in connection to the *normalized* numbers derived below.

For both cases it can be observed that the difference in cost between the CSDA and the forward mode is relatively small. According to table 11, the run time for the CSDA method is only 15-20% higher per sensitivity compared to the forward mode. This observation is consistent with the comment given in section 4.4 regarding the connection between these two methods.

	Bumping		Forward		Adjoint	
# Sim. paths	Case 1	Case 2	Case 1	Case 2	Case 1	Case 2
5.000	27.77%	30.18%	-19.20%	-16.51%	-85.38%	-88.87%
10.000	28.16%	30.04%	-19.03%	-16.53%	-85.38%	-88.87%
50.000	29.07%	38.32%	-17.83%	-15.32%	-60.52%	-69.32%
100.000	29.11%	38.41%	-17.78%	-15.27%	-51.26%	-63.37%

Table 11: Relative change in run time per sensitivity compared to the CSDA method.

Looking further into the relative differences between the methods in figure 9, these seem to increase for 100.000 simulation paths for the when going from the first to the second case. Observing the relative difference per sensitivity in table 11, one can see that these are approximately equal in two cases for the first two methods. The adjoint mode becomes relatively cheaper compared to the other methods when the number of sensitivities increase, which can also visible in this table. Thus, the increased dispersion in the computational cost is only caused by the increase in the number of sensitivities.

The computational cost for the adjoint mode is observed to be constant across the two cases in line with the methods independence of the number of inputs. But the relative difference reported in table 11 are, contrary to the other methods, not stable across the different numbers of simulation paths. There seem to be a significant jump for 50.000 simulation paths, this will become more evident in the following examinations.

Bounds on the run time for the forward- and adjoint mode were defined in eqn. (32) and eqn. (36) for multiple in- and outputs, respectively. In order to visually assess whether these bounds are respected in practice, the run times given in figure 9 have initially been normalized as described below.

The CVA calculation consists of two nested loops: an outer loop propagating all n time buckets, given in eqn. (3), and an inner loop iterating for all m simulation paths, cf. algorithm 1. Asymptotic analysis of the run time for the CVA calculation then (with slight abuse of notation) gives:

$$\mathcal{O}(CVA_0) = \mathcal{O}(n) + \mathcal{O}(n \cdot m) \quad \Leftrightarrow \quad \mathcal{O}\Big(\frac{CVA_0}{m}\Big) = \mathcal{O}\Big(\frac{n}{m}\Big) + \mathcal{O}(n)\,. \tag{39}$$

In this setup the number of time buckets, n, is a constant of 120. Hence, the second term, $\mathcal{O}(n)$, is likewise a constant, while the first term, $\mathcal{O}\big(\frac{n}{m}\big)$, fast becomes very small.

Hence, the run times normalized wrt. to the number of simulation paths should theoretically be constant across the various numbers of simulation paths. The normalized run times for the forward- and adjoint mode are depicted in figure 10 along with the theoretical bounds. Note that only the first case are reported here as the figure for the second case is equivalent.

In figure 10 it can be observed that the normed run times for the forward mode lies well within the bounds and is, as predicted, constant. The run times for the adjoint mode are on the other hand not constant across the various number simulation paths, and the trend seen in table 11 are thereby repeated here. Furthermore, notice that the adjoint mode *over-performs* for the first two observations which lie

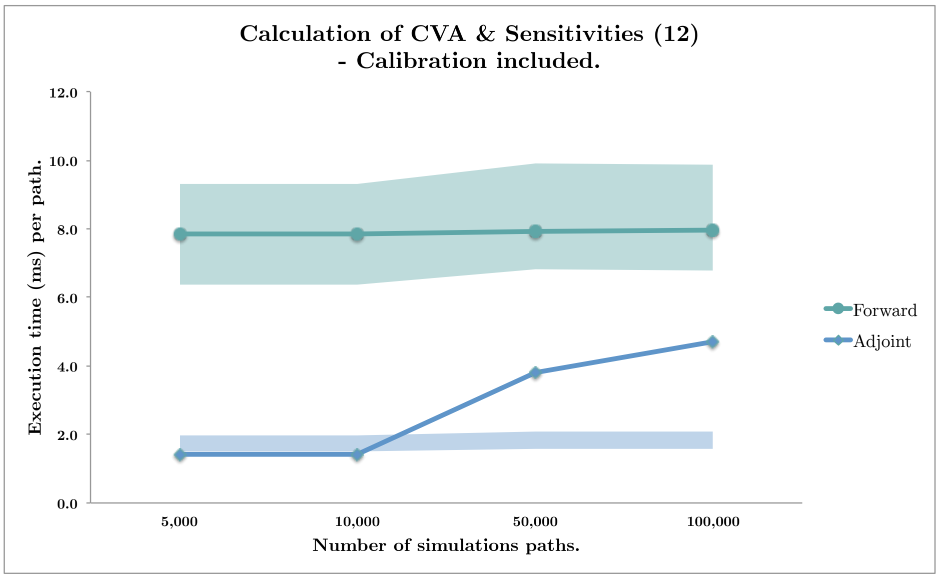


Figure 10: Run time normed wrt. the number of simulation paths for case 1.

below the theoretical lower bound.

Neither the time bounds given in eqn. (36), nor the asymptotic result in eqn. (39), accounts for the memory cost of the algorithm. Thus, even though the record have been managed according to the rules-of-thumb given in section 4.3, the algorithm seems to be memory bound for 50.000 simulation paths and above. This observation is consistent with the preliminary remark, stating that the adjoint mode runs out of memory around 150.000 simulation paths.

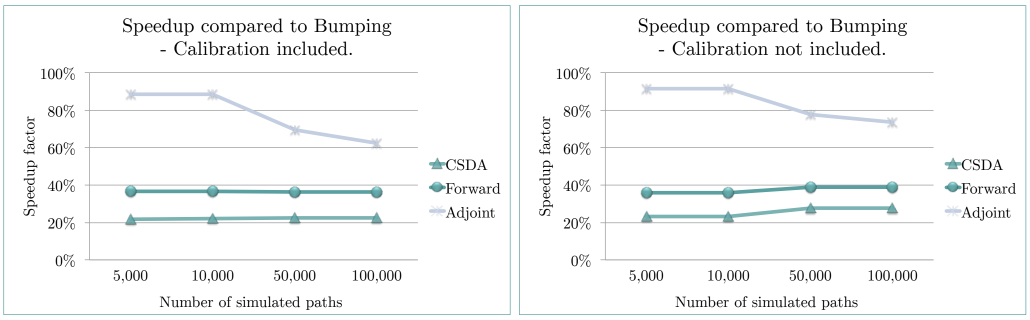


Figure 11: Speedup for the three different methods compared to Bumping.

The last comparison of the run times consists of determining the speedup that could be gained by replacing bumping with one of the other methods. Figure 11 reports these speedups for both cases. As expected is the highest speedup obtained for the adjoint mode, as this method is superior to the forward mode when the number of inputs greatly exceeds the number of outputs, cf. table 4. The speedup factor for the CSDA and forward mode are a good deal lower, but still significant.

5.4 Automatic tool

The automatic AD tool FADBAD++ has been applied to the CVA algorithm for both the forward and adjoint mode. As mentioned in section 4.5, the implementation is fairly easy, especially for this case where the CSDA version was already available. Hence, here the implementation consisted of a find-and-replace of the data type `complex<double>` with `F<double>` or `B<double>`, and adding a few alterations to the structure of the function call. The estimated developer time from beginning to extracting of the correct results at the end, was limited to a maximum of four hours per mode. This is a tremendous reduction in the development cost compared to the hand-coded versions.

There are two main trade-offs though to this reduction in developer cost. The first is the inefficient memory consumption for the adjoint mode. This stems from the automatic tools inability to make intelligent decisions when it comes to the recording of variables during the forward sweep. This manifest itself in case 2 where the transformed code are unable to execute for numbers of simulations paths above 10.000, which is significantly worse than the limit of 100.000 for the hand-coded version. Unfortunately, the code is not able to execute at all for case 1 where the calibration process is included in the derivative calculation. This issue could be related to the complex control flow for the Levenberg-Marquardt algorithm.

The second trade-off comes in the form of increased computational cost, as seen in figure 12. In both cases the run time for the automatic forward mode, FaDiff, lies well above the bumping method which had the worst performance out of the four methods assessed. Surprisingly, the run time for the automatic adjoint mode, BaDiff, actually lies above this level, and thereby has the worst performance of all.

According to the quote given below, the run times of the automatically generated forward- and adjoint implementations should be a constant factor times the run time for the hand-coded version.

> ... an unsophisticated approach suffices to produce AD code that is within a constant factor of the optimal performance bounds ... subsequent effort is devoted just to reducing the value of this constant (Griewank & Walther (2008, p.107))

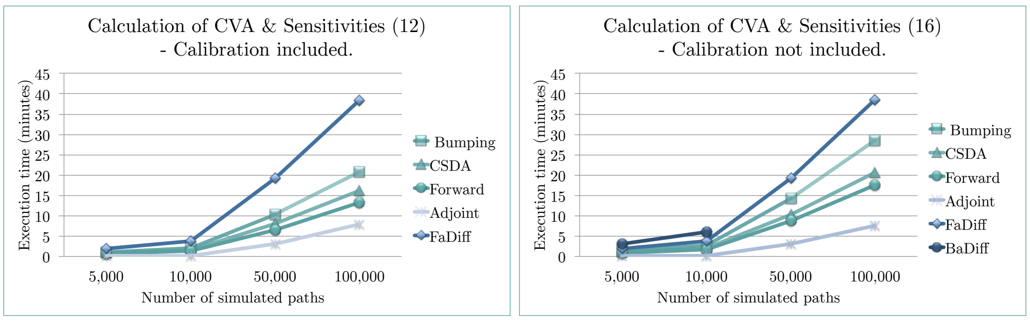


Figure 12: Absolute value of the relative error compared to the CSDA method, calculated using 100.000 simulation paths.

There does indeed seem be such constant factors when looking at table 12. The constant factor for the automatically generated adjoint mode is a remarkable factor of 26.3 times higher than the hand-coded version in case 2.

	Case 1	Case 2	
	FaDiff : Forward	FaDiff : Forward	BaDiff : Adjoint
5000	2.9440	2.2242	26.2752
10000	2.9353	2.2187	26.3253
50000	2.9064	2.2066	
100000	2.9000	2.1974	

Table 12: The ratio between the automatic forward- and adjoint mode and the hand-coded forward- and adjoint mode, respectively.

The automatic AD modes could also have been implemented in a brute force manor where all `doubles` are replaced by either `F<double>` or `B<double>`. This way one would have avoided the process of identifying all active variables. But, concluding from the execution times reported above, this would have resulted in a pretty much useless implementation for this case.

6 WRAPPING IT UP

The objective of this paper was to assess the CSDA and algorithmic differentiation methods as alternatives to bumping which currently rules the financial industry. The performance of the method were in this paper examined in terms of accuracy and run time. Both are important objects when it comes to calculating the CVA price which involves Monte Carlo simulations for a series of time buckets.

6.1 WHEN TO USE WHAT?

The accuracy gains by replacing the bumping method with either of the three machine precision methods: CSDA, forward- and adjoint mode, was illustrated by figure 7. Here it was evident that especially for factors with a large impact on the CVA price this gain in accuracy could be significant.

The possible speedups related to replacing the bumping method was reported in table 11. These should be seen in relation with the developer cost associated with the implementation for each of the alternative methods. This is a trade-off which should be weighed against the purpose of the algorithm: which is more important: low initial costs i.e. *developer costs*, or low long term use costs i.e. *computational costs*?

The CSDA method is by far the cheapest to implement as the only alterations to the original algorithm needed is changing the data type and conditional operations for all active variables. In return, the speedup gained is on average only 24% compare to bumping, which is significantly lower than for both of the algorithmic methods. This method could also have been implemented by brute force, replacing all doubles with their complex counterparts. This would have further reduced development costs but would also have resulted in an even lower speedup.

The development cost takes a jump upwards when going from the CSDA method to the algorithmic forward mode. In addition to identifying all active variables, the symbolic derivative for each line of code must now be derived and added accordingly. The corresponding speedup obtained, when going from the CSDA to the forward mode, does in return almost double to an average of 37%.

The by far most expensive method in terms of development is given by the algorithmic adjoint mode. Here, not only must the derivatives be determined but the calculation must also be reversed which is a major source of error. Additionally, there is the whole memory issue which becomes urgent for numbers of simulation paths above 50.000 and critical above 100.000 . This can be resolved either by porting the solution to hardware containing more RAM, or through a more active memory management strategy. The reward for this increase in development effort comes in the form of a speed gain of 90% for numbers of simulation paths up to and including 50.000. Above this level of paths, the algorithm becomes memory bound and the gain is reduced to an average of 70%.

As an attempt to overcome these rather large development costs associated with the two algorithmic modes, the automatic AD tool FADBAD++ was applied to the algorithm. Unfortunately, this general-purpose tool was not sophisticated enough to obtain any speedups for this framework . In fact, the speed was decreased by an average of 85% for case 1 and 39% for case 2, when replacing bumping with the automatic forward mode. The backward mode was hardly able to run, and for the two calculations in case 1 for which it did run, the decrease in run time compared to bumping were 125%.

Even though the automatic tool did not do any wonders for the computational costs in this framework the concept is still very useful in practice, especially when considering long term maintenance costs. The derivatives for algorithms, which are object to continuous improvements or alterations, can be easily maintained using either an automatic AD tool or the CSDA method, as neither require anything else than management of the data types. This crucial factor should also be taken into account when comparing the pros and cons for the various methods.

Instead of applying just one of the methods for the entire algorithm, it is also possible to combine their qualities into a *hybrid* mode, where the methods are applied locally to subparts of the algorithm.

This could be a purely algorithmic hybrid where the optimal mode are determined and applied for parts of the code instead of the algorithm as a whole. Though for this framework it can be observed in the figures given in appendix A.1, that all functions have more inputs than outputs. Hence, the adjoint mode is the optimal choice for all functions and introducing a hybrid mode would not lead to any further speedups.

In practice a hybrid could be any combination of the methods presented in this paper, which applied to an algorithm, would lead to reduced development cost, increased accuracy or additional speedups.

6.2 Before Flying the Nest

As mentioned in the beginning of this section, the purpose of this framework was a purely academic comparison of alternative methods for deriving sensitivities.

Hence, the complexity level of the financial CVA framework should be increased before being applied in practice. This can be obtained by replacing one or both of the models described in sections 2.3 and 2.4, and by removing some of the simplifying assumptions listed in the beginning of section 2. Likewise, the setup should be expanded in order to handle portfolios on counterparty level.

On the computational side the run time for each of the algorithms should be brought down before being used in practice. There are various possibilities for achieving this, the most obvious for this type of application is vectorization which would increase the speed by several magnitudes. As briefly mentioned, this is a demanding task for a C++ implementation, but as a side benefit it will become easier afterwards to transfer parts of the computation to a GPGPU for additional speedups. Parallelizing parts of the code could furthermore be used in connection with multicore processors or computer clusters, as described in Griewank & Walther (2008, ch. 6.3). The latter of these could also solve some of the memory issues regarding the adjoint mode implementation previously discussed.

A APPENDIX : SOURCE CODE

The source code is divided in 6 main modules. `Basic` contains the original CVA algorithm, reused for calculating the finite difference derivatives. `Complex, Forward, Backward` contains code for calculation of the CSDA, forward- and adjoint mode derivatives, respectively. `AutoDiff` contains both the forward and the adjoint mode implemented using FADBAD++. The last module `Testing` contains classes testing the derivatives, for each of the functions displayed below, for all methods.

The main function for calculating the CVA for the interest rate swap is placed in the `IRS` class as function `CVA_Zero`. The results displayed in section 5 have been obtained by running the two functions `CVA_noCalibration` and `CVA_Calibration`, members of the class `CVA_Calculation`.

A.1 OBJECT ORIENTED DESIGN

This sections contains figures for each class depicted in figure 4, providing a detailed description of the functionality. The notation used in the figures are given by:

- active variables.
- - - - -➤ inputted (outputted) implicitly from (to) a data member.

The following details have been left out:

- Read-access to private data members not depicted.
- Simple 'transport' functions not depicted.
- Passive functions not depicted.
- Specific data types left out.

FIGURE 13: IRS OBJECT.

Class responsible for calculating the value and CVA price of an ATM, forward starting, receiver interest rate swap. Functions include valuation and simulation of the CVA price.

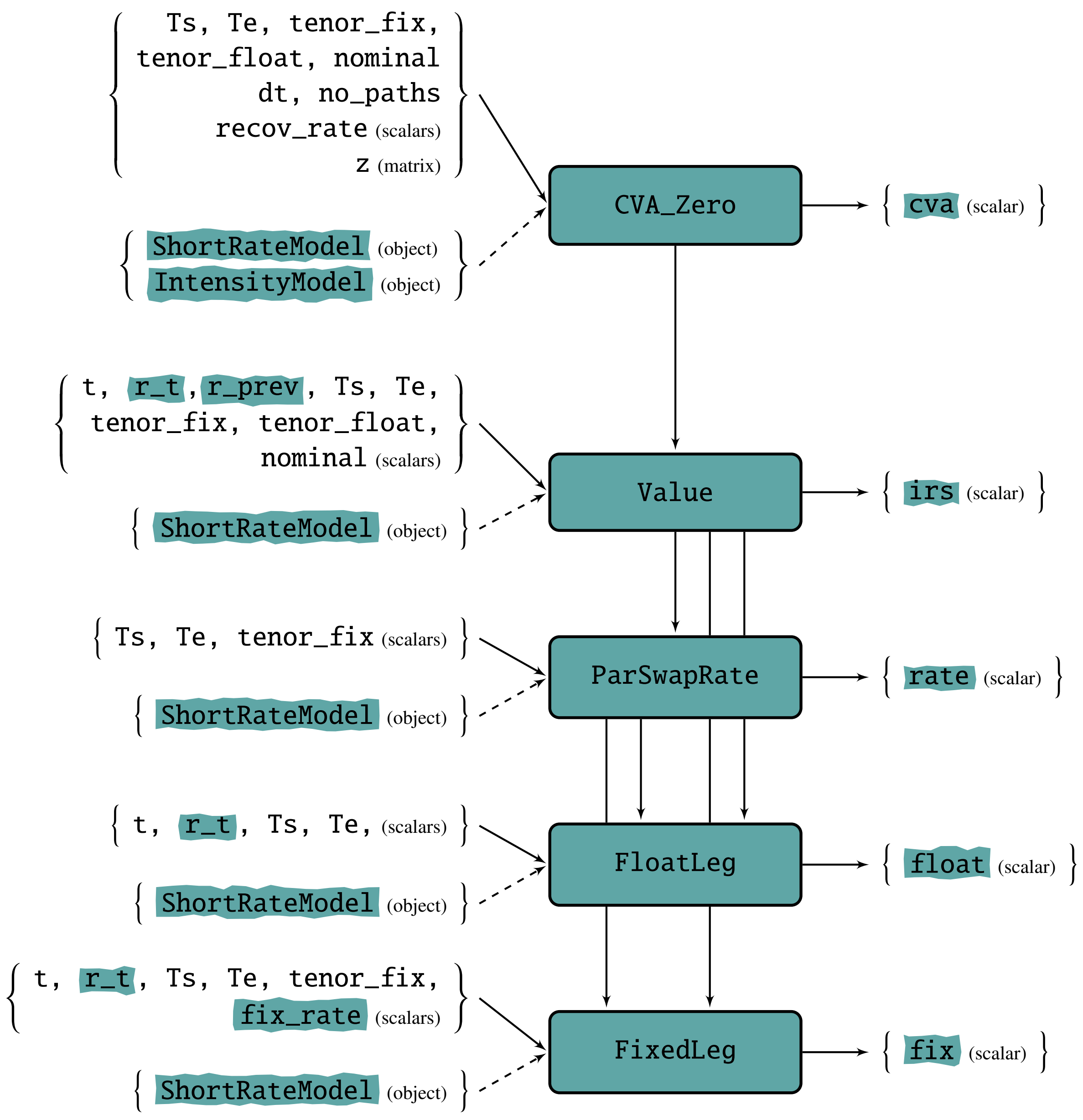


FIGURE 14: HULL_WHITE OBJECT.

Class implementing abstract class 'ShortRateModel' - responsible for modeling the stochastic short rate process according to the Hull White model. Functions include simulating the short rate, calculation of the zero coupon bond price, calculation of the price of a European Put Option (for calibration purposes).

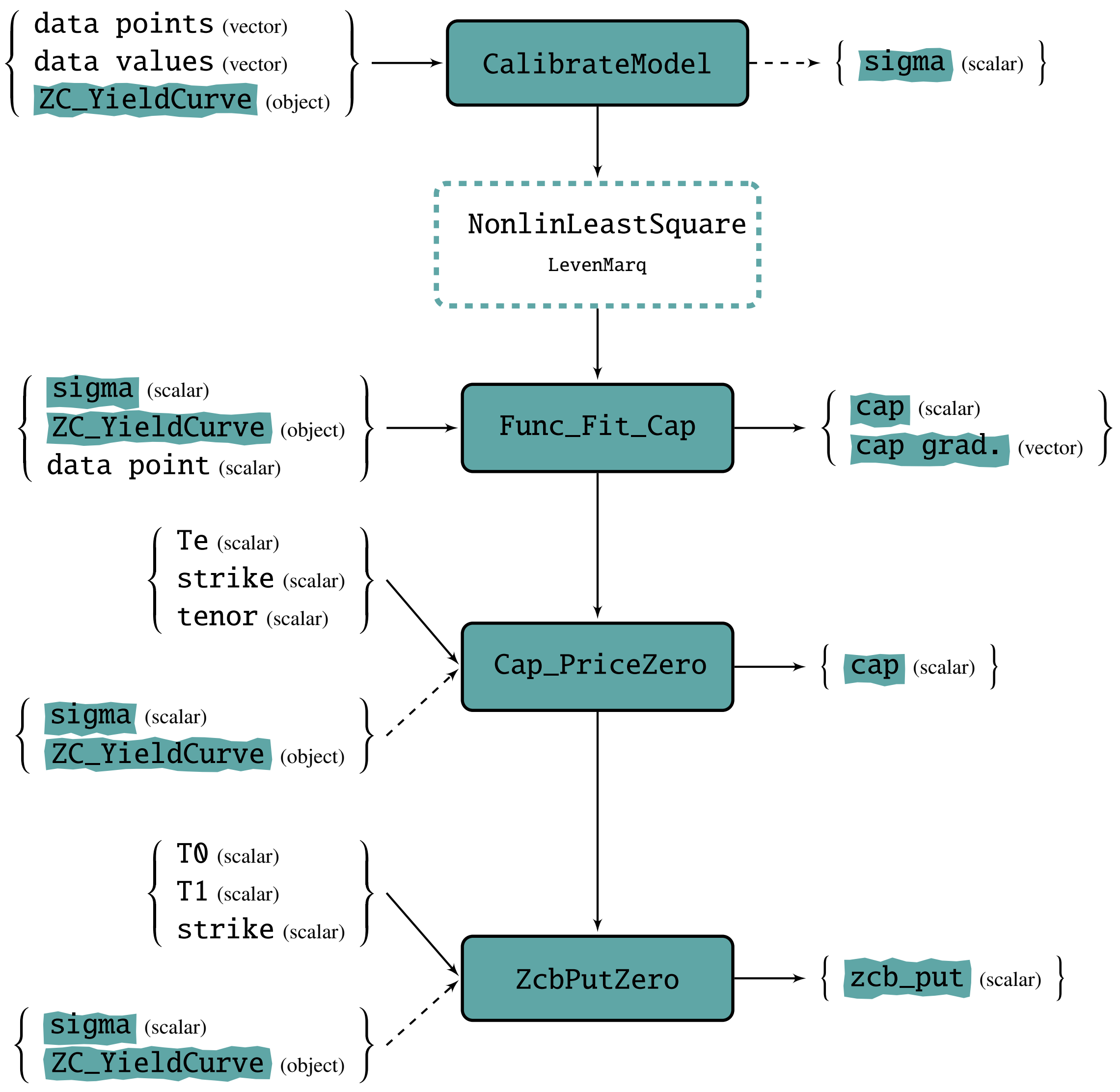
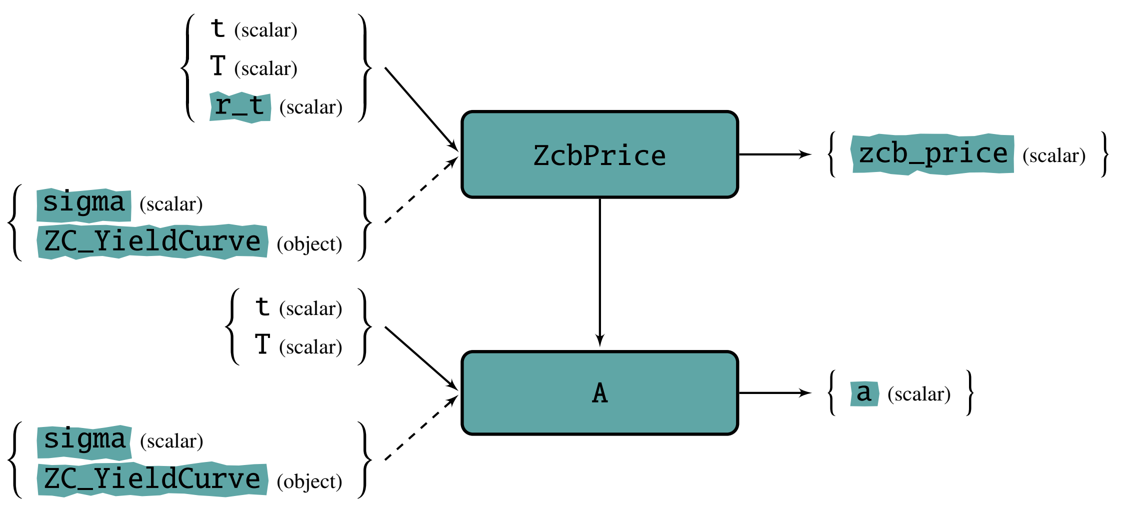


FIGURE 14: Continued...



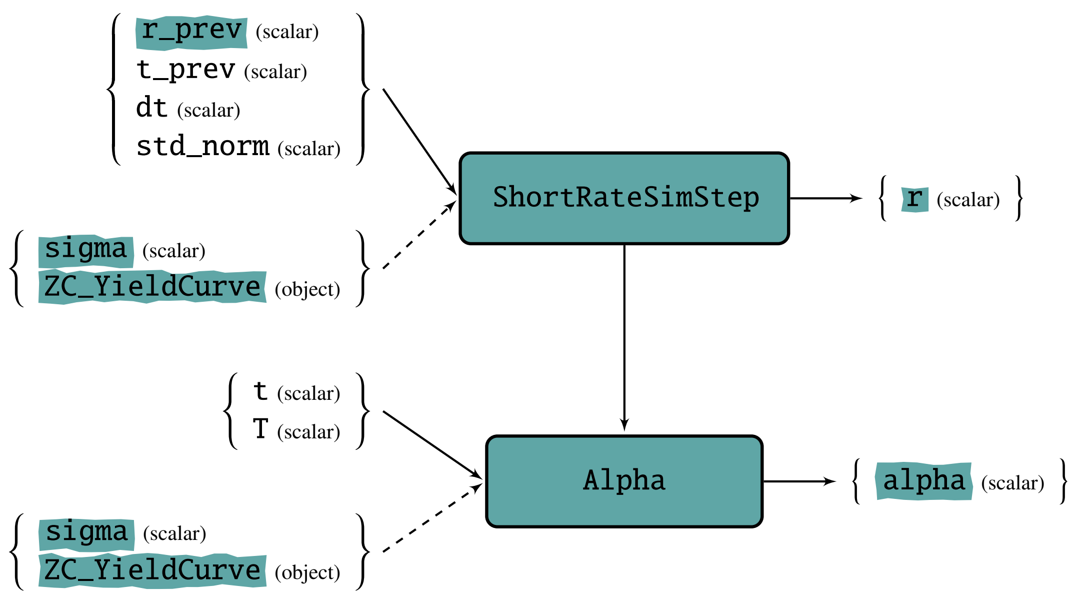


FIGURE 15: CIR OBJECT.

Class implementing abstract class ’IntensityModel’ - responsible for modeling the stochastic short rate process according to the CIR model. Functions include calculating the survival- and default probabilities once the parameters have been calibrated to a given counterparty.

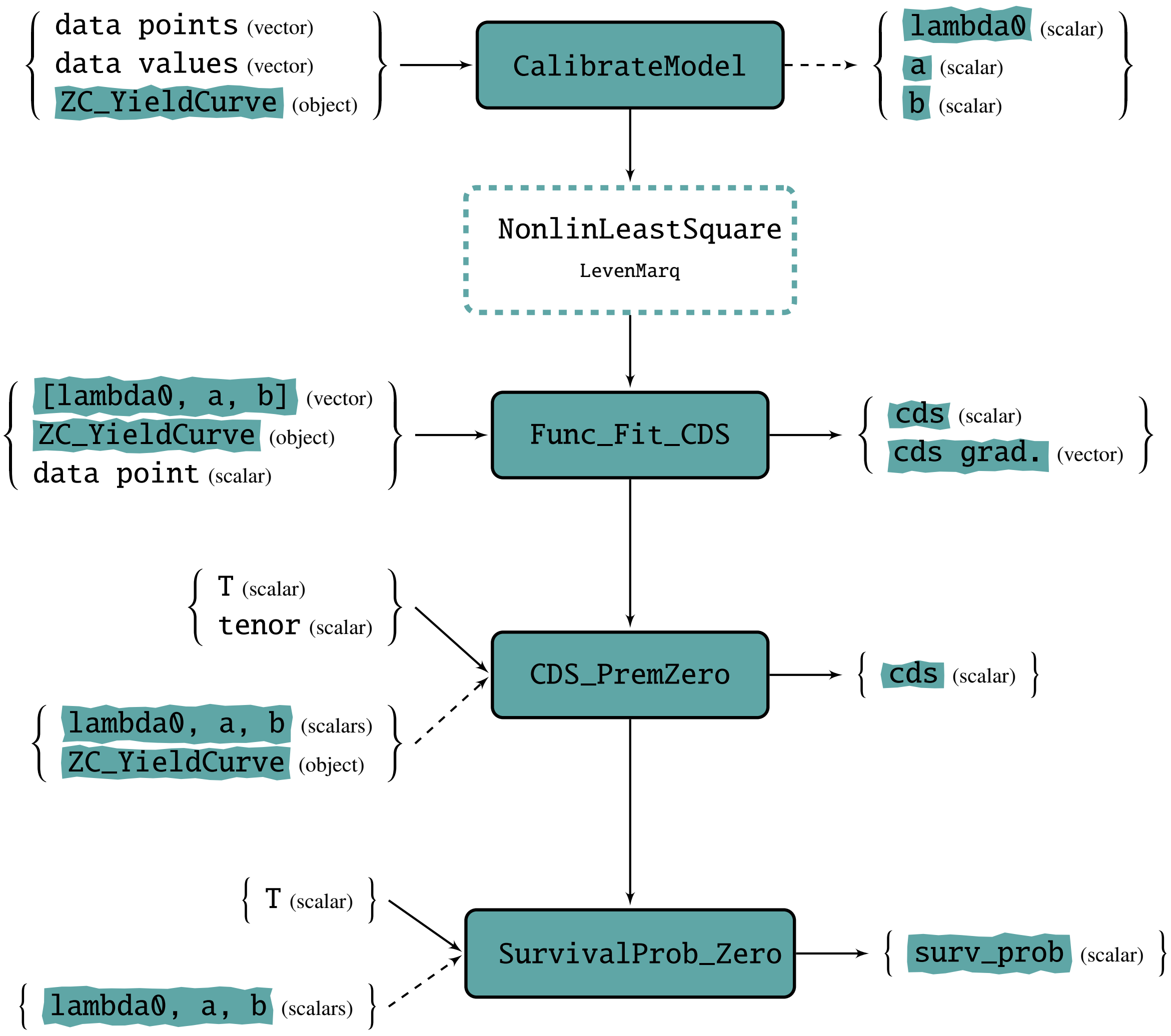


FIGURE 16: ZC_YIELDCURVE OBJECT.

Class responsible for the Zero Coupon Yield Curve given by a CubicSpline object fitted to a set of observed swaprates. Functions for interpolating the Zero Coupon- rate and price are available.

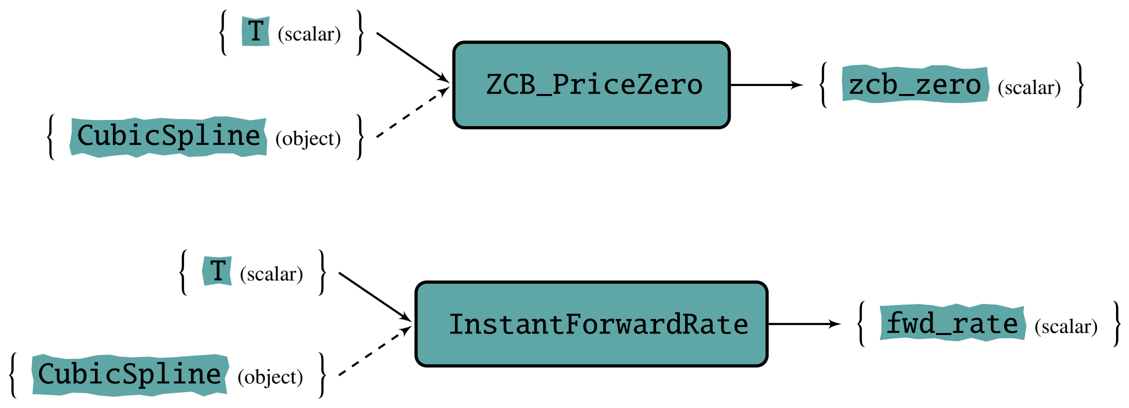


FIGURE 17: NORMDIST OBJECT.

Class responsible for calculating the normal cumulative distribution function.

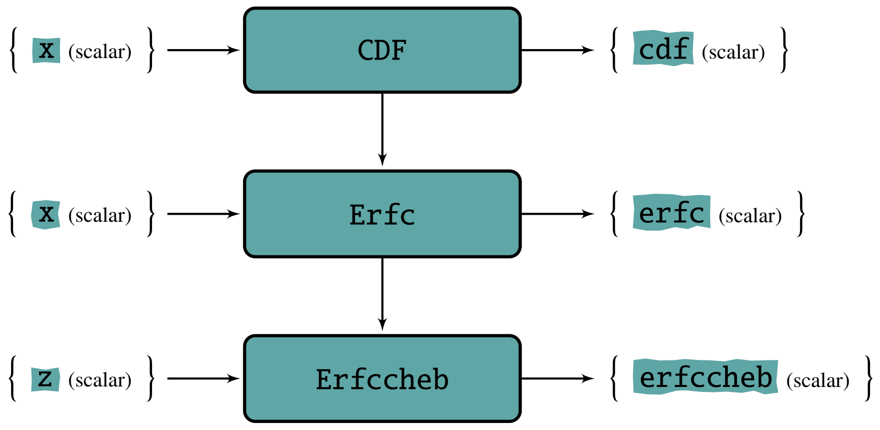


FIGURE 18: LEVENMARQ OBJECT.

Class responsible for the non-linear least square optimization 'Levenberg Marquardt'. Calculations include solution of a linear equation, $RHS = X \cdot LHS$, using Gauss-Jordan elimination. The calculations are given in Numerical Recipes 3rd edition §15.5.2 and §2.1.2.

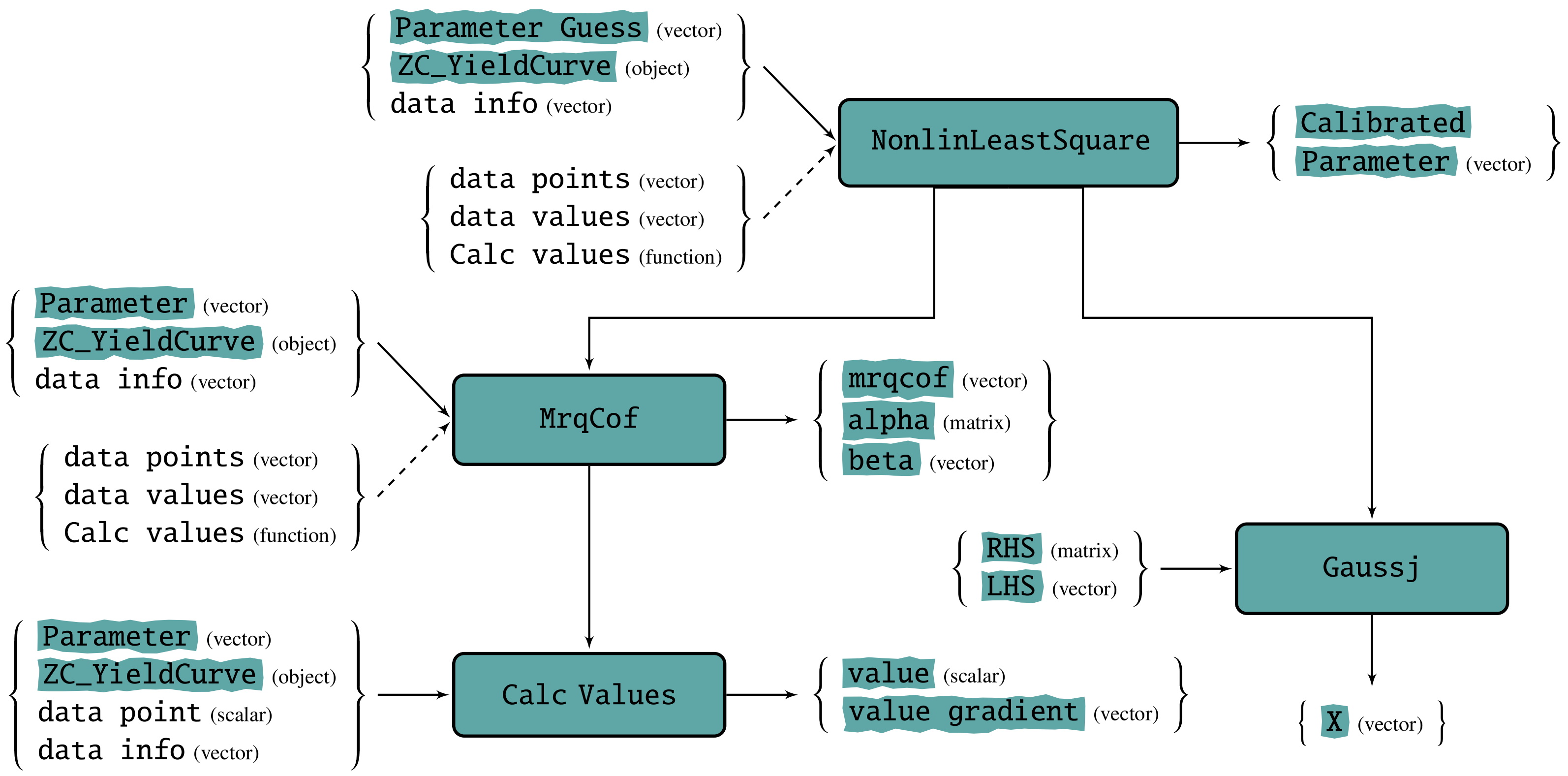
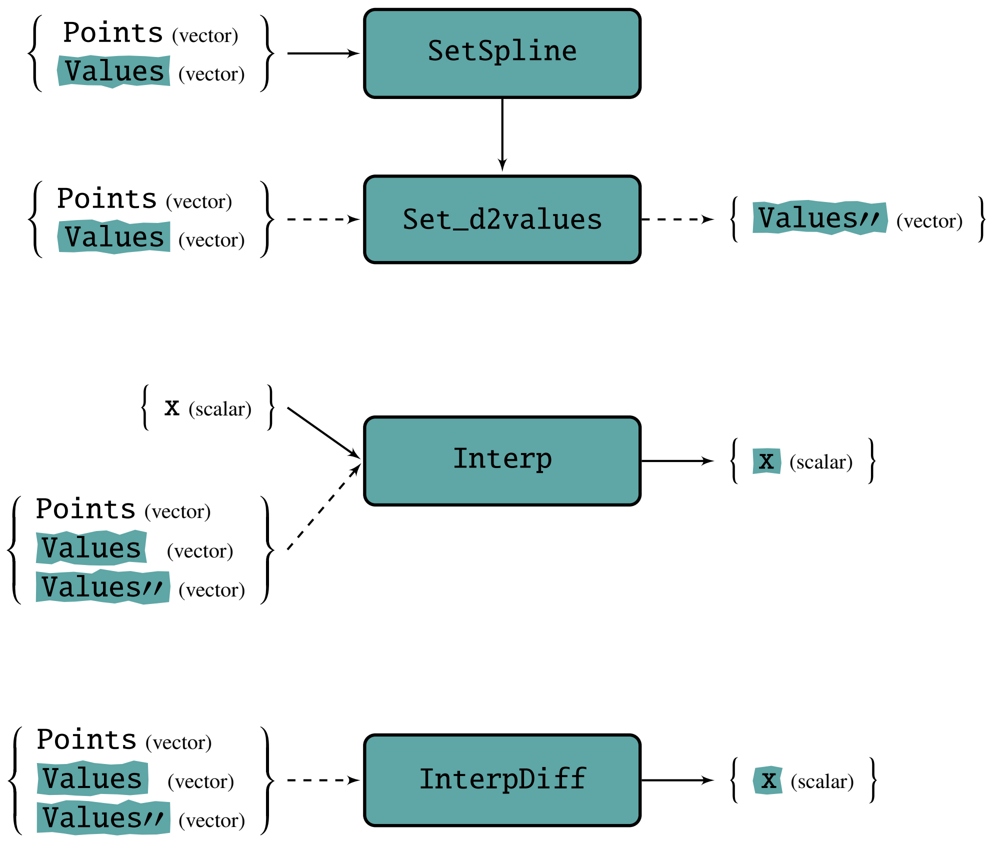


FIGURE 19: CUBICSPLINE OBJECT.

Class responsible for calculating and storing a natural cubic spline interpolation object given by its: points, values and values”. Functions for interpolation and interpolation of 1st derivative are available.



A.2 CODE FORMATTING

Below are descriptions of the function formats for the various derivatives modules. Here, the input, `x`, and output, `y`, can each take the form of a scalar, multiple scalars, vectors or other objects. In the code keyword: `const`, have been used to indicate which functions do not alter the input and/or data members of the corresponding object.

▷ `Basic` : this code is re-used for evaluating the finite difference derivatives.

```
eval(x,y).
```

- ▷ `Complex` : the code is given by the original algorithm where all active variables and functions: >= and max(), have been replaced with their complex counterparts.

  ```
  eval(x,y).
  ```

- ▷ `Forward` : the code is given by the original algorithm with each statement augmented by a derivative counterpart. The seed direction is given as input, `x_dot`, and the output, `y_dot`, is the forward derivative of `y`.

  ```
  eval(x,x_dot, y, y_dot).
  ```

- ▷ `Backward` : the code consists of approximately twice the amount of functions compared to the original algorithm. One half of these are given by the original evaluation where variables are pushed onto the tape according to the rules-of-thumb in section 4.3. The other half consists of the reverse propagation of the original evaluation trace where intermediate variables are popped off the tape. The input, `y_bar`, is the weight functional set to zero after the evaluation in accordance with the scheme given in table 8. The calculated derivative is added to, and saved in, the input variable `x_bar`.

  ```
  eval(x,y)    eval_bar(x_bar,y_bar)
  ```

In practice some of the inputs `x`, `x_dot`, `x_bar`, `y_bar`, and some of the outputs `y`, `y_dot`, `x_bar`, are given or saved implicitly as data members in the various classes. This eases the handling of variables and objects included in multiple functions, but unfortunately also decreases the transparency.

B Appendix : Data

# Sim. paths	Mean CVA_0	RSD(%) CVA_0	95% Confidence Interval	Run time(s)
5.000	2.1444	1.86	[2.073; 2.2211]	0.24
10.000	2.1438	1.34	[2.0837; 2.2005]	0.33
100.000	2.1452	1.43	[2.1276; 2.1637]	2.67

Table 13: Table 4 from Hansen & Glibstrup (2014) repeated for validation of the computed CVA_0 values. The values have been obtained for a sample of 2.000 calculations.

TABLE 14: 5.000 SIMULATION PATHS - CASE 1.

Run time(ms) CVA | 2490	Bumping	CSDA	Forward	Adjoint	FaDiff
Run time(ms) : $\frac{\partial CVA_0}{\partial \boldsymbol{\theta_2}}$	62079	48586	39258	7102	115574
CVA_0	2.1313	2.1313	2.1313	2.1313	2.1313
$\frac{\partial CVA_0}{\partial r_0}$	0.2346	0.2346	0.2346	0.2346	0.2346
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	1.5589	1.5589	1.5589	1.5589	1.5589
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	2.6733	2.6734	2.6734	2.6734	2.6734
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	5.4421	5.4422	5.4422	5.4422	5.4422
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	1.1597	1.1600	1.1600	1.1600	1.1600
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	13.4697	13.4695	13.4695	13.4695	13.4695
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-4.9693	-4.9680	-4.9680	-4.9680	-4.9680
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-62.6081	-62.6064	-62.6064	-62.6064	-62.6064
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	34.6099	34.6055	34.6055	34.6055	34.6055
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	17.3917	17.3896	17.3896	17.3896	17.3896
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	3.2096	3.2239	3.2239	3.2239	3.2239
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-141.2222	-141.2317	-141.2317	-141.2317	-141.2317

TABLE 15: 10.000 SIMULATION PATHS - CASE 1.

Run time(ms) CVA | 4981	Bumping	CSDA	Forward	Adjoint	FaDiff
Run time(ms) : $\frac{\partial CVA_0}{\partial \theta_2}$	124398	97061	78586	14188	230672
CVA_0	2.0854	2.0854	2.0854	2.0854	2.0854
$\frac{\partial CVA_0}{\partial r_0}$	0.2258	0.2258	0.2258	0.2258	0.2258
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	1.5083	1.5083	1.5083	1.5083	1.5083
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	2.5648	2.5648	2.5648	2.5648	2.5648
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	5.2800	5.2801	5.2801	5.2801	5.2801
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	0.9781	0.9779	0.9779	0.9779	0.9779
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	13.2674	13.2682	13.2682	13.2682	13.2682
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-5.2688	-5.2694	-5.2694	-5.2694	-5.2694
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-61.7091	-61.7111	-61.7111	-61.7111	-61.7111
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	35.7269	35.7325	35.7325	35.7325	35.7325
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	17.8675	17.8737	17.8737	17.8737	17.8737
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	4.2691	4.2690	4.2690	4.2690	4.2690
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-139.6577	-139.6622	-139.6622	-139.6622	-139.6622

TABLE 16: 50.000 SIMULATION PATHS - CASE 1.

Run time(ms) CVA | 24949	Bumping	CSDA	Forward	Adjoint	FaDiff
Run time(ms) : $\frac{\partial CVA_0}{\partial \theta_2}$	623141	482787	396715	190620	1153013
CVA_0	2.1535	2.1535	2.1535	2.1535	2.1535
$\frac{\partial CVA_0}{\partial r_0}$	0.2372	0.2372	0.2372	0.2372	0.2372
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	1.5769	1.5769	1.5769	1.5769	1.5769
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	2.7027	2.7027	2.7027	2.7027	2.7027
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	5.5015	5.5016	5.5016	5.5016	5.5016
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	1.1687	1.1687	1.1687	1.1687	1.1687
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	13.6201	13.6205	13.6205	13.6205	13.6205
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-5.0365	-5.0358	-5.0358	-5.0358	-5.0358
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-63.5462	-63.5452	-63.5452	-63.5452	-63.5452
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	35.5133	35.5113	35.5113	35.5113	35.5113
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	17.6747	17.6760	17.6760	17.6760	17.6760
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	3.3578	3.3598	3.3598	3.3598	3.3598
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-143.2753	-143.2779	-143.2779	-143.2779	-143.2779

TABLE 17: 100.000 SIMULATION PATHS - CASE 1.

Run time(ms) CVA | 49919	Bumping	CSDA	Forward	Adjoint	FaDiff
Run time(ms) : $\frac{\partial CVA_0}{\partial \boldsymbol{\theta_2}}$	1248044	966659	794771	471161	2304841
CVA_0	2.1395	2.1395	2.1395	2.1395	2.1395
$\frac{\partial CVA_0}{\partial r_0}$	0.2345	0.2345	0.2345	0.2345	0.2345
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	1.5609	1.5609	1.5609	1.5609	1.5609
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	2.6694	2.6694	2.6694	2.6694	2.6694
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	5.4531	5.4531	5.4531	5.4531	5.4531
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	1.1129	1.1130	1.1130	1.1130	1.1130
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	13.5626	13.5629	13.5629	13.5629	13.5629
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-5.1297	-5.1291	-5.1291	-5.1291	-5.1291
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-63.1128	-63.1123	-63.1123	-63.1123	-63.1123
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	35.7172	35.7147	35.7147	35.7147	35.7147
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	17.5281	17.5277	17.5277	17.5277	17.5277
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	3.3693	3.3716	3.3716	3.3716	3.3716
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-142.1769	-142.1803	-142.1803	-142.1803	-142.1803

TABLE 18: 5.000 SIMULATION PATHS - CASE 2.

Run time(ms) CVA | 2450	Bumping	CSDA	Forward	Adjoint	FaDiff	BaDiff
Run time(ms) : $\frac{\partial CVA_0}{\partial \theta_2}$	80758	62035	51790	6904	115192	181404
CVA_0	2.0905	2.0905	2.0905	2.0905	2.0905	2.0905
$\frac{\partial CVA_0}{\partial \sigma_{hw}}$	161.9774	161.9811	161.9811	161.9811	161.9811	161.9811
$\frac{\partial CVA_0}{\partial \lambda_0}$	-3.0499	-3.0499	-3.0499	-3.0499	-3.0499	-3.0499
$\frac{\partial CVA_0}{\partial a_{cir}}$	0.1890	0.1890	0.1890	0.1890	0.1890	0.1890
$\frac{\partial CVA_0}{\partial b_{cir}}$	8.9637	8.9637	8.9637	8.9637	8.9637	8.9637
$\frac{\partial CVA_0}{\partial r_0}$	-0.0355	-0.0355	-0.0355	-0.0355	-0.0355	-0.0355
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	0.0860	0.0860	0.0860	0.0860	0.0860	0.0860
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	-0.2174	-0.2174	-0.2174	-0.2174	-0.2174	-0.2174
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	0.7838	0.7838	0.7838	0.7838	0.7838	0.7838
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	-2.9178	-2.9178	-2.9178	-2.9178	-2.9178	-2.9178
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	5.8084	5.8086	5.8086	5.8086	5.8086	5.8086
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-10.6155	-10.6162	-10.6162	-10.6162	-10.6162	-10.6162
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-47.3404	-47.3348	-47.3348	-47.3348	-47.3348	-47.3348
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	60.3694	60.3568	60.3568	60.3568	60.3568	60.3568
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	41.8763	41.8704	41.8704	41.8704	41.8704	41.8704
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	29.7769	29.7844	29.7844	29.7844	29.7844	29.7844
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-113.2407	-113.2385	-113.2385	-113.2385	-113.2385	-113.2385

Table 19: 10.000 simulation paths - Case 2.

Run time(ms) CVA |4893	Bumping	CSDA	Forward	Adjoint	FaDiff	BaDiff
Run time(ms) : $\frac{\partial CVA_0}{\partial \theta_2}$	161405	124118	103597	13810	229852	363552
CVA_0	2.1329	2.1329	2.1329	2.1329	2.1329	2.1329
$\frac{\partial CVA_0}{\partial \sigma_{hw}}$	167.0035	167.0036	167.0036	167.0036	167.0036	167.0036
$\frac{\partial CVA_0}{\partial \lambda_0}$	-3.0613	-3.0613	-3.0613	-3.0613	-3.0613	-3.0613
$\frac{\partial CVA_0}{\partial a_{cir}}$	0.2029	0.2029	0.2029	0.2029	0.2029	0.2029
$\frac{\partial CVA_0}{\partial b_{cir}}$	9.1953	9.1953	9.1953	9.1953	9.1953	9.1953
$\frac{\partial CVA_0}{\partial r_0}$	-0.0362	-0.0362	-0.0362	-0.0362	-0.0362	-0.0362
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	0.0880	0.0880	0.0880	0.0880	0.0880	0.0880
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	-0.2226	-0.2226	-0.2226	-0.2226	-0.2226	-0.2226
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	0.8026	0.8026	0.8026	0.8026	0.8026	0.8026
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	-2.9878	-2.9878	-2.9878	-2.9878	-2.9878	-2.9878
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	5.9476	5.9479	5.9479	5.9479	5.9479	5.9479
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-10.8701	-10.8709	-10.8709	-10.8709	-10.8709	-10.8709
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-47.8398	-47.8388	-47.8388	-47.8388	-47.8388	-47.8388
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	61.1165	61.1177	61.1177	61.1177	61.1177	61.1177
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	42.2135	42.2131	42.2131	42.2131	42.2131	42.2131
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	29.8508	29.8496	29.8496	29.8496	29.8496	29.8496
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-114.4785	-114.4726	-114.4726	-114.4726	-114.4726	-114.4726

TABLE 20: 50.000 SIMULATION PATHS - CASE 2.

Run time(ms) CVA | 26123	Bumping	CSDA	Forward	Adjoint	FaDiff	BaDiff
Run time(ms) : $\frac{\partial CVA_0}{\partial \theta_2}$	856225	619024	524203	189932	1156699	0.0
CVA_0	2.1446	2.1446	2.1446	2.1446	2.1446	0.0
$\frac{\partial CVA_0}{\partial \sigma_{hw}}$	168.5107	168.5110	168.5110	168.5110	168.5110	0.0
$\frac{\partial CVA_0}{\partial \lambda_0}$	-3.0899	-3.0899	-3.0899	-3.0899	-3.0899	0.0
$\frac{\partial CVA_0}{\partial a_{cir}}$	0.2084	0.2084	0.2084	0.2084	0.2084	0.0
$\frac{\partial CVA_0}{\partial b_{cir}}$	9.3248	9.3247	9.3247	9.3247	9.3247	0.0
$\frac{\partial CVA_0}{\partial r_0}$	-0.0362	-0.0362	-0.0362	-0.0362	-0.0362	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	0.0873	0.0873	0.0873	0.0873	0.0873	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	-0.2193	-0.2193	-0.2193	-0.2193	-0.2193	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	0.7899	0.7899	0.7899	0.7899	0.7899	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	-2.9403	-2.9403	-2.9403	-2.9403	-2.9403	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	5.8533	5.8532	5.8532	5.8532	5.8532	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-10.6983	-10.6978	-10.6978	-10.6978	-10.6978	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-47.9091	-47.9083	-47.9083	-47.9083	-47.9083	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	60.7049	60.7020	60.7020	60.7020	60.7020	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	42.3127	42.3137	42.3137	42.3137	42.3137	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	29.8809	29.8839	29.8839	29.8839	29.8839	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-114.4099	-114.4121	-114.4121	-114.4121	-114.4121	0.0

TABLE 21: 100.000 SIMULATION PATHS - CASE 2.

Run time(ms) CVA | 51964	Bumping	CSDA	Forward	Adjoint	FaDiff	BaDiff
Run time(ms) : $\frac{\partial CVA_0}{\partial \theta_2}$	1715183	1239182	1049976	453942	2307244	0.0
CVA_0	2.1456	2.1456	2.1456	2.1456	2.1456	0.0
$\frac{\partial CVA_0}{\partial \sigma_{hw}}$	168.5919	168.5929	168.5929	168.5929	168.5929	0.0
$\frac{\partial CVA_0}{\partial \lambda_0}$	-3.1253	-3.1253	-3.1253	-3.1253	-3.1253	0.0
$\frac{\partial CVA_0}{\partial a_{cir}}$	0.2010	0.2010	0.2010	0.2010	0.2010	0.0
$\frac{\partial CVA_0}{\partial b_{cir}}$	9.2929	9.2929	9.2929	9.2929	9.2929	0.0
$\frac{\partial CVA_0}{\partial r_0}$	-0.0363	-0.0363	-0.0363	-0.0363	-0.0363	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,1)}$	0.0876	0.0876	0.0876	0.0876	0.0876	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,2)}$	-0.2205	-0.2205	-0.2205	-0.2205	-0.2205	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,3)}$	0.7944	0.7944	0.7944	0.7944	0.7944	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,4)}$	-2.9569	-2.9569	-2.9569	-2.9569	-2.9569	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,5)}$	5.8863	5.8862	5.8862	5.8862	5.8862	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,7)}$	-10.7583	-10.7582	-10.7582	-10.7582	-10.7582	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,10)}$	-47.9872	-47.9862	-47.9862	-47.9862	-47.9862	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,15)}$	60.9790	60.9770	60.9770	60.9770	60.9770	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,20)}$	42.2713	42.2701	42.2701	42.2701	42.2701	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,25)}$	29.9424	29.9404	29.9404	29.9404	29.9404	0.0
$\frac{\partial CVA_0}{\partial r^*_{cont}(0,30)}$	-114.6113	-114.6074	-114.6074	-114.6074	-114.6074	0.0

BIBLIOGRAPHY

BRIGO, DAMIANO AND MERCURIO, FABIO (2006). *Interest Rate Models - Theory and Practice with Smile, Inflation and Credit*. Springer, Berlin, 2nd edition.

BRIGO, DAMIANO AND MORINI, MASSIMO AND PALLAVICINI, ANDREA (2013). *Counterparty Credit Risk, Collateral and Funding*. John Wiley & Sons, 1st edition.

CAPRIOTTI, LUCA (2011). Fast Greeks by algorithmic differentiation. *The Journal of Computational Finance*, 14(3), 3–35.

DAVIDSON, CLIVE (2015). Structured products desks join the AAD revolution.

GILES, MICHAEL (2007). Monte Carlo evaluation of sensitivities in computational finance.

GILES, MIKE AND GLASSERMAN, PAUL (2006). Smoking adjoints : fast Monte Carlo Greeks. *Risk Magazine*, (3), 88–92.

GILES, MICHAEL B. AND GHATE, DEVENDRA P AND DUTA, MIHAI C. (2008). Using Automatic Differentiation for Adjoint CFD Code Development. *Computational Fluid Dynamics Journal*, 16(4), 434–443.

GLASSERMAN, PAUL (2004). *Monte Carlo Methods in Financial Engineering*. Springer.

GRIEWANK, ANDREAS AND WALTHER, ANDREA (2008). *Evaluating Derivatives - Principles and Techniques of Algorithmic Differentiation*. SiAM, 2nd edition.

HANSEN, LEA NØHR HJELMAGER AND GLIBSTRUP, ANDERS KJÆR (2014). CVA med Wrong Way risiko. Master's thesis, University of Copenhagen.

HOMESCU, CRISTIAN (2011). Generic computing alternatives for better Greeks.

LINDERSTROEM, MARTIN DALSKOV (2013). Fixed Income Derivatives. Course notes for the M.Sc. course 'Fixed Income Derivatives: Risk Management and Financial Institutions' at the Department of Economics, University of Copenhagen.

MARTINS, JOAQUIM R. R. A. AND STURDZA, PETER AND ALONSO, JUAN J. (2003). The complex-step derivative approximation. *ACM Transactions on Mathematical Software*, 29(3), 245–262.

MUNK, CLAUS (2011). *Fixed income modelling*. Oxford University Press.

PRESS, WILLIAM H. AND TEUKOLSKY, SAUL A. AND FLANNERY, BRIAN P. (2007). *Numerical Recipes*. Cambridge University Press, 3rd edition.

SAVICKAS, VYTAUTAS (2011). *Fast Greeks : Case of Credit Valuation Adjustments*. PhD thesis, Utrecht University.

TAVELLA, DOMIGO AND RANDALL, CURT (2000). *Pricing Financial Instruments*. John Wiley & Sons.

THE KAVLI FOUNDATION (2016). Extreme machines: What science needs from computers. [Online; accessed 8-January-2016].