Adjoint and PDE methods for pricing and risk management of exotic interest rate derivatives

Nicholas Andrew Denson

Submitted in total fulfilment of the requirements of the degree of Doctor of Philosophy

November 2010
Centre for Actuarial Studies
The University of Melbourne
Abstract

This thesis demonstrates how to compute Greeks accurately and efficiently for the displaced-diffusion LIBOR market model and the separable LIBOR Markov-functional model.

Chapter 1 introduces the problem and discusses the three main classes of interest rate models. The LIBOR market model is introduced in detail in Chapter 2, where we also provide information for an efficient model implementation.

In Chapter 3 we extend the pathwise adjoint method to the displaced-diffusion LIBOR market model and present a 20 percent speed improvement. This improvement is achieved without additional approximations to those of Giles and Glasserman, [34]. We also derive the pathwise adjoint method for the predictor-corrector drift approximation. We compare the deltas and vegas computed using log-Euler and predictor-corrector, showing both methods have the same computational order but the latter is more accurate.

In Chapter 4 we introduce the separable LIBOR Markov-functional model. We present a new formulation in the spot measure and find that for a Monte Carlo implementation the spot measure performs better than the terminal measure, which is consistent with well-known results for the LIBOR market model. For a PDE implementation, however, we find that the terminal measure performs better than the spot measure. These results are demonstrated by pricing interest rate caps and double-digital options.

Chapter 5 presents the separable LIBOR Markov-functional model as a control variate for the LIBOR market model. This method has been previously suggested in the literature, however, it only works for prices and deltas, not vegas. We present a new technique that is effective for all three. Our approach introduces many innovations, but the essential improvement is an accurate, efficient and stable calibration procedure. It achieves a standard error reduction by a factor of 8 and 10 for the prices of a five-factor, 20-year Bermudan swaption and cancellable inverse floater. For the Bermudan swaption a reduction of 5 is achieved for the vega.

In Chapter 6 we introduce the adjoint PDE method for Markov-functional models. This is an accurate and efficient way to compute Greeks, where most of the
model sensitivities can be calculated in approximately the same time as it takes to
determine the price. We demonstrate the speed and accuracy of the method using
the separable LIBOR Markov-functional model, also showing how the model Greeks
can be converted into market Greeks.
Declaration

This is to certify that:

1. the thesis comprises only my original work towards the PhD except where indicated in the Preface,

2. due acknowledgement has been made in the text to all other material used,

3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Signed,

Nicholas Andrew Denson
Preface

This thesis was completed under the supervision of Associate Professor Mark Joshi in the Centre for Actuarial Studies at The University of Melbourne. Chapters 3-6 contain the original research of this thesis, except as otherwise noted.

Chapter 3 is based on two papers “Flaming logs”, co-authored by Mark Joshi and published in Wilmott Journal, Volume 1, 2009. The second paper “Fast and accurate Greeks for the LIBOR market model” was also co-authored by Mark Joshi and is to appear in the Journal of Computational Finance. The research for both papers was carried out by Nicholas Denson under the supervision of Mark Joshi. Both papers were written by Nicholas Denson, with proofreading and editing by Mark Joshi.

A subset of Chapter 5 is based on the paper “Smooth Calibration of Markov-functional models for pricing exotic interest rate derivatives” co-authored by Mark Joshi to appear in Risk. The research was carried out and written by Nicholas Denson with supervision and editing by Mark Joshi.

Chapter 6 is based on the paper “Fast Greeks for Markov-functional models using adjoint PDE methods” co-authored by Mark Joshi and available on the SSRN eLibrary. The research for this paper was carried out by Nicholas Denson under the supervision of Mark Joshi. The paper was written and figures produced by Nicholas Denson, with proofreading by Mark Joshi.

None of the work appearing here has been submitted for any other qualifications, nor was it carried out prior to PhD candidature enrollment.
Acknowledgements

Many people have contributed to making my PhD an enriching experience. First and foremost I would like to thank my supervisor Associate Professor Mark Joshi. His teaching, assistance and practical knowledge has been invaluable. Working with him has been an enjoyable and thought provoking experience.

I am grateful to the Department of Economics and the Centre for Actuarial Studies at the University of Melbourne for providing a stimulating environment within which to complete my PhD. I would also like to thank them for their financial support.

Thanks to my fellow students in the Centre for Actuarial Studies and research fellow Will Wright, for their interesting discussions.

Finally, I would like to sincerely thank my family and friends, in particular Mum, Dad, Jo and Kim.
Contents

List of Figures xiii
List of Tables xiv
Notation xx

1 Introduction 1
  1.1 Interest rate derivatives ............................................. 1
    1.1.1 Short-rate models ............................................. 2
    1.1.2 Market models ................................................. 2
    1.1.3 Markov-functional models ...................................... 4
  1.2 The structure of the thesis ........................................ 5

2 The displaced-diffusion LIBOR market model 7
  2.1 Model dynamics ..................................................... 7
  2.2 Drift approximations .............................................. 10
    2.2.1 Log-Euler ...................................................... 10
    2.2.2 Predictor-corrector ........................................... 10
    2.2.3 Glasserman and Zhao .......................................... 11
    2.2.4 Which drift approximation to use? ............................ 12
  2.3 Products .......................................................... 12
  2.4 Calibration ........................................................ 13
    2.4.1 Volatility and correlation calibration ......................... 13
    2.4.2 Swaption approximation ....................................... 15
    2.4.3 Skew calibration .............................................. 15
  2.5 Simulation .......................................................... 17
  2.6 Early exercise ..................................................... 18

3 LIBOR market model Greeks 21
  3.1 Pathwise adjoint method .......................................... 22
  3.2 Working with the logarithm ....................................... 23
6.1.5 Markov Greeks .............................................. 85
6.2 Markov-functional model derivatives .......................... 86
  6.2.1 Markov-functional model delta .......................... 87
  6.2.2 Markov-functional model mapping vegas ................. 87
  6.2.3 Markov-functional model Markov vegas ................. 89
  6.2.4 Markov-functional model skew sensitivity .......... 90
6.3 Numerical results ............................................ 91
6.4 Market Greeks .............................................. 94
  6.4.1 Market vega ........................................... 95
  6.4.2 Market skew ......................................... 100

Bibliography .................................................. 107
List of Figures

3.1 Delta errors for out-of-the-money caplets in the spot measure using log-Euler (LE) and predictor-corrector (PC). .............................................. 41
3.2 Vega errors for out-of-the-money caplets in the spot measure using log-Euler (LE) and predictor-corrector (PC). .............................................. 41
3.3 Delta errors for out-of-the-money caplets in the terminal measure using log-Euler (LE) and predictor-corrector (PC). .............................................. 42
3.4 Vega errors for out-of-the-money caplets in the terminal measure using log-Euler (LE) and predictor-corrector (PC). .............................................. 45

4.1 Markov-functional model pricing errors (in basis points) for out-of-the-money caplets using a PDE method. .............................................. 59
4.2 Markov-functional model pricing errors and 2 standard errors (in basis points) for out-of-the-money caplets using a quasi-Monte Carlo method. .............................................. 60
4.3 Markov-functional model pricing errors (in basis points) for double-digital options using a PDE method. .............................................. 61
4.4 Markov-functional model pricing errors and 2 standard errors (in basis points) for double-digital options using a quasi-Monte Carlo method. .............................................. 62

6.1 Sensitivity of the cancellable inverse floater swap with respect to caplet volatilities (scaled by a factor of 100). .............................................. 97
6.2 Sensitivity of the cancellable inverse floater swap with respect to coterminal swaption volatilities (scaled by a factor of 100). .............................................. 98
6.3 Sensitivity of the cancellable inverse floater swap with respect to caplet and coterminal swaption volatilities (scaled by a factor of 100). .............................................. 100
6.4 Sensitivity of the cancellable inverse floater swap (with strike = 10%) with respect to caplet and coterminal swaption volatilities, showing the difference caused by including the Markov vegas (scaled by a factor of 100). .............................................. 101
6.5 Sensitivity of the cancellable inverse floater swap (with strike = 18\%) with respect to caplet and coterminal swaption volatilities (scaled by a factor of 100). \hspace{1cm} 102

6.6 Sensitivity of the callable Bermudan swaption with respect to caplet and coterminal swaption volatilities, showing the difference of including the Markov vegas (scaled by a factor of 100). \hspace{1cm} 103
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Average multiple of time taken (and standard error) by working with the forward rates instead of the log forward rates</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Analytic Black-76 deltas and vegas for the ten caplets</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Relative error for deltas and vegas for the ten caplets in the spot measure using log-Euler (LE) and predictor-corrector (PC)</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Relative error for deltas and vegas for the ten caplets in the terminal measure using log-Euler (LE) and predictor-corrector (PC)</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>Prices of the lockout Bermudan swaptions</td>
<td>43</td>
</tr>
<tr>
<td>3.6</td>
<td>Relative error for the deltas of the different length lockout Bermudan swaptions in the spot measure using log-Euler (LE) and predictor-corrector (PC)</td>
<td>43</td>
</tr>
<tr>
<td>3.7</td>
<td>Relative error for the vegas of the different length lockout Bermudan swaptions in the spot measure using log-Euler (LE) and predictor-corrector (PC)</td>
<td>44</td>
</tr>
<tr>
<td>3.8</td>
<td>Time taken (in seconds) to compute price and deltas and vegas for the 5-year lockout Bermudan swaption with $2^{14}$ paths</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison of absolute price errors (in basis points) for ITM (strike 3%), ATM (strike 5%) and OTM (strike 7%) caps using PC and IPC drift approximations. Prices were computed using a PDE method.</td>
<td>58</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of absolute price errors (in basis points) for ITM (strike 3%), ATM (strike 5%) and OTM (strike 7%) Caps using PC and IPC drift approximations. Prices were computed using randomised quasi-Monte Carlo simulation.</td>
<td>58</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of the errors (in basis points) for the caplets and double-digital options, computed using equation (4.16)</td>
<td>61</td>
</tr>
<tr>
<td>5.1</td>
<td>Calibration accuracy range for a variety of volatility scenarios using the statistic (5.2) with a two-factor LMM</td>
<td>69</td>
</tr>
</tbody>
</table>
5.2 Calibration accuracy range for a variety of volatility scenarios using the statistic (5.2) with a three-factor LMM. ............................ 69
5.3 Calibration accuracy range for a variety of volatility scenarios using the statistic (5.2) with a five-factor LMM. ............................ 69
5.4 Pseudo-square roots exactly fitted corresponding to the two-factor calibration in Table 5.1 ............................................................ 70
5.5 Prices and standard errors in basis points for different tenor Bermudan swaptions. 2-, 3- and 5-factor LMMs were used with the control variate. ......................................................... 71
5.6 Standard errors in basis points for different tenor Bermudan swaptions using both terminal and spot measures. 2-, 3- and 5-factor LMMs were used with the control variate. ......................... 72
5.7 Standard errors in basis points for different tenor cancellable inverse floater swaps using both terminal and spot measures. 2-, 3- and 5-factor LMMs were used with the control variate. ......................... 72
5.8 Model vegas and standard errors in basis points for the Bermudan swaptions priced in Table 5.5. All caplet volatilities were increased by 1%.............................................................. 74
5.9 Model vegas and standard errors in basis points for the Bermudan swaptions priced in Table 5.5. Three caplet volatilities in the middle of the product’s term structure were perturbed. ....................... 74
5.10 Model vegas and standard errors in basis points for the Bermudan swaptions priced in Table 5.5. The middle caplet volatilities were perturbed i.e. those corresponding to the forward rates resetting at times $T_5$, $T_7$, $T_{10}$ and $T_{20}$ for the 5Y, 7Y, 10Y and 20Y products respectively. ......................................................... 75
5.11 Standard error reductions when computing the vega in Table 5.10. Method 1 uses the calibration defined by equation (5.3), while method 2 uses (5.4). ................................................................. 76
5.12 Estimate and standard error of $\zeta$ in equation (5.5), corresponding to the pricing Table 5.5 ............................................................. 77
5.13 Estimate and standard error of $\zeta$ in equation (5.5), corresponding to the vega Table 5.8 ................................................................. 77
5.14 Estimate and standard error of $\zeta$ in equation (5.5), corresponding to the vega Table 5.9 ................................................................. 77
5.15 Estimate and standard error of $\zeta$ in equation (5.5), corresponding to the vega Table 5.10 ................................................................. 78
6.1 Deltas, vegas and skew sensitivities for a cap, strike = 7%. A 51×51 spatial grid was used. .......... 93
6.2 Deltas, vegas and skew sensitivities for a cap, strike = 7%. A 101×101 spatial grid was used. .......... 94
6.3 Markov vegas for a cap, strike = 7%. A 101×101 spatial grid was used. 95
6.4 Deltas, vegas and skew sensitivities for a cancellable inverse floater swap, strike = 10%. A 51×51 spatial grid was used for the finite difference (FD) and adjoint Greeks. .......... 96
6.5 Market volatilities used in Figure 6.3 .......... 99
6.6 Deltas, vegas and skew sensitivities for a callable Bermudan swaption, strike = 7%. A 101×101 spatial grid was used. .......... 102
Notation

The following is a list of notation used throughout the thesis, listed in order of appearance.

- $P(t, T_k)$  The price at time $t$ of a zero-coupon bond paying one at its maturity, $T_k$
- $\text{SR}_i(T_k)$  The coterminal swap rate from time $T_i$ to $T_n$, evaluated at time $T_k$
- $\text{Ann}_i(T_k)$  The annuity of the swap from time $T_i$ to $T_n$, evaluated at time $T_k$
- $n$  The number of forward rates
- $f_i(T_k)$  The forward rate for the interval $[T_i, T_{i+1})$ at time $T_k$
- $T_k$  Tenor date, $k = 0, 1, \ldots, n$
- $\tau_k$  Accrual factor, equal to $T_{k+1} - T_k$
- $\alpha_i$  Displacement coefficient for $f_i$
- $\mu$  Drift terms
- $\sigma_i(t)$  Instantaneous volatility term for $f_i$ at time $t$
- $W_i(t)$  Brownian motion
- $\text{cov}_{ij}$  The covariance of $\log f_i$ and $\log f_j$
- $A(k)$  The pseudo-square root of the covariance matrix across time step $[T_{k-1}, T_k)$ for the LMM
- $D$  Number of factors used in the LMM
- $Z(k)$  A $D$-dimensional vector of independent Gaussian random variables across time step $[T_{k-1}, T_k)$
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\eta(t)$</td>
<td>The index of the next forward rate to reset at time $t$</td>
</tr>
<tr>
<td>$N(T_k)$</td>
<td>Numeraire value at time $T_k$</td>
</tr>
<tr>
<td>$K$</td>
<td>Option strike</td>
</tr>
<tr>
<td>$U$</td>
<td>Matrix to convert from LIBOR to coterminal swap rate volatility</td>
</tr>
<tr>
<td>$g$</td>
<td>Deflated payoff function</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Model parameter</td>
</tr>
<tr>
<td>$H_k$</td>
<td>Map from log forward rates at time $T_k$ to time $T_{k+1}$</td>
</tr>
<tr>
<td>$Y(k)$</td>
<td>Displaced log rates, equal to $\log(f(k) + \alpha)$</td>
</tr>
<tr>
<td>$V(k)$</td>
<td>Delta vector at time $T_k$</td>
</tr>
<tr>
<td>$\tilde{V}(k)$</td>
<td>Delta vector at time $T_k$, using log rates</td>
</tr>
<tr>
<td>$S_{ij}(k)$</td>
<td>Intermediate drift calculation</td>
</tr>
<tr>
<td>$F$</td>
<td>Number of factors used in the Markov-functional model</td>
</tr>
<tr>
<td>$X(T_k)$</td>
<td>$F$-dimensional vector of Markov factors across $[0, T_k)$</td>
</tr>
<tr>
<td>$u(T_k)$</td>
<td>Vector of discretised option values at time $T_k$</td>
</tr>
<tr>
<td>$h(u, T_k)$</td>
<td>PDE option value at time $T_k$</td>
</tr>
<tr>
<td>$\nu_i$</td>
<td>$F$-dimensional vector of forward rate specific volatility terms for $f_i$ in the Markov-functional model</td>
</tr>
<tr>
<td>$C(t)$</td>
<td>$F \times F$ matrix of time-dependent volatility terms for the Markov-functional model</td>
</tr>
<tr>
<td>$\zeta$</td>
<td>Variable in control variate equation</td>
</tr>
<tr>
<td>$B(k)$</td>
<td>The pseudo-square root over $[T_{k-1}, T_k)$ for the separable Markov-functional model</td>
</tr>
<tr>
<td>$\lambda(T_k)$</td>
<td>Adjoint PDE variable at time $T_k$</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Interest rate derivatives

Interest rate derivatives dwarf all other asset classes. According to the Bank for International Settlements, [1], the notional outstanding on interest rate swaps in December 2009 was $US 349 trillion. This, combined with the notional outstanding on forward rate agreements of $US 51 trillion and other interest rate contracts of $US 450 trillion, brings the total notional amount close to a quadrillion US dollars. To put this into perspective the notional outstanding on foreign exchange contracts was $US 49 trillion.

The sheer size of interest rate derivative portfolios pose significant risks to banks. It is therefore imperative that the mathematical models used to price and risk manage these contracts accurately capture the movements of interest rates. Three major classes of models are used for this purpose, short-rate models, market models and Markov-functional models. These are in general more complex than models used for foreign exchange or equity derivatives, because they need to capture the term structure of interest rates, instead of the price of a single asset. Accurately and efficiently modelling the interest rate term structure provides interesting mathematical challenges.

Consider a bank with a large interest rate derivatives portfolio. The trading desk and risk management systems need to continuously recalculate prices and sensitivities of these trades as the market changes. There have been instances of banks having to limit the number of trades, not from lack of investor interest, but from
the computational burden of the models. This is clearly not an ideal situation. The techniques presented in this thesis aim to reduce this computational burden, whilst retaining an acceptable level of accuracy.

1.1.1 Short-rate models

Short-rate models sparked mathematical curiosity in interest rate derivatives. Introduced in the late 1970s, they aim to model the entire yield curve in terms of an instantaneous short rate, \( r(t) \). Once the short rate is determined we can compute the prices of zero-coupon bonds and then value a product. In the case of the classical Vasicek model, \([86]\), a one-dimensional diffusion is used

\[
\frac{dr(t)}{dt} = k[\theta - r(t)]dt + \sigma dW(t),
\]

where \( k, \theta \) and \( \sigma \) are deterministic parameters and \( W(t) \) is a standard Brownian motion in the relevant pricing measure. The literature is crowded with many short-rate models, some common ones being Dothan, \([30]\), Brennan and Schwartz, \([18]\), Cox, Ingersoll and Ross, \([24]\), Hull and White, \([42]\), and Black and Karasinski, \([10]\).

One of the reasons for the popularity of these models is that they are Markovian in the short rate. Being Markovian means that the future value of the yield curve is determined by the current value of the short rate. This is a key advantage as it implies that prices of interest rate products can be computed using lattice methods, such as trees, PDEs or numerical integration. Since short-rate models typically use one or two factors this means that a lattice implementation will be computationally efficient and prices can be computed quickly.

There are, however, two major disadvantages with short-rate models. The main problem stems from modelling a quantity that is not observable in the market. Practitioners therefore need to work hard just to calibrate a model to basic market prices. Another issue is that one or two driving factors are not usually sufficient to price exotic interest rate derivatives.

For an in-depth coverage of short-rate models we refer the reader to the textbooks of Brigo and Mercurio, \([19]\), Cairns, \([20]\) and Wilmott, \([87]\).

1.1.2 Market models

The most frequently traded interest rate products are forward rate agreements (FRAs) and interest rate swaps. So instead of trying to calculate the value of these products by modelling an unobservable instantaneous rate of interest, why not model these quantities directly? This is the approach of market models. The LIBOR market model (LMM), introduced by Miltersen, Sandmann, and Sonder-
1.1. Interest rate derivatives

mann, [62], Brace, Gatarek and Musiela, [17], Musiela and Rutkowski, [63], and Jamshidian, [49], focuses on forward or LIBOR rates. An alternative approach, due to Jamshidian, [49], models swap rates.

The main benefit of modelling market observable rates is that we can automatically calibrate to the initial yield and volatility curves. Options traded on FRAs and swaps, known as caplets and swaptions, are quoted in terms of volatilities for the Black formula, [15]. If we assume that forward rates or swap rates have log-normal volatilities, we will be consistent with this formula. To fit the model to the market, we then directly use these volatilities.

Throughout this thesis we will focus on the LMM, where forward rates are the fundamental quantities. A forward rate is a future rate of interest between times $T_i$ and $T_{i+1}$ defined by no arbitrage arguments and expressed in terms of zero-coupon bonds as

$$f_i(t) = \frac{P(t, T_i) - P(t, T_{i+1})}{(T_{i+1} - T_i)P(t, T_{i+1})},$$

where $P(t, T_i)$ denotes the price at time $t$ of a zero-coupon bond paying one at its maturity, $T_i$. The term structure of interest rates, known as the yield curve, is then made up of $n$ forward rates which span the tenor dates $T_0, T_1, \ldots, T_n$.

We will need to price swaps within this model, so we define $SR_i(T_k)$ to denote the coterminal swap rate that runs from time $T_i$ to $T_n$, evaluated at time $T_k$. The swap rate is given by no arbitrage arguments as a weighted average of the forward rates

$$SR_i(T_k) = \sum_{j=i}^{n-1} w_j(T_k)f_j(T_k),$$

where the weights sum to one and are calculated using

$$w_j(T_k) = \frac{\tau_j P(T_k, T_{j+1})}{\text{Ann}_0(T_k)},$$

and $\text{Ann}_i(T_k)$ is the annuity of the swap

$$\text{Ann}_i(T_k) = \sum_{j=i}^{n-1} \tau_j P(T_k, T_{j+1}).$$

There is an inconsistency with the LMM and the market, the forward rates and swap rates cannot both have log-normal volatilities. This well-known problem is theoretically unpleasant, but is less of a practical issue. Fortunately there exist good approximations for swap rate volatilities, which we will meet in Section 2.4.2.
1.1.3 Markov-functional models

Markov-functional models are designed to strike a calibrational and computational medium. That is, they are constructed to automatically calibrate to the current yield curve and caplet volatilities, whilst remaining Markovian in a small number of factors and therefore being computationally efficient. It is often said that they try to capture the best features of the LMM and short-rate models.

A general procedure for creating Markovian interest rate models was first presented by Schmidt, [81], and Hagan and Woodward, [39]. Schmidt concentrated on a general formulation of a one-factor short-rate model. Hagan and Woodward, focused on models in the Heath, Jarrow and Morton framework, [40], with instantaneous forward rates. Both papers were more interested in the properties of the underlying Markov process than the functional form of the model.

The first Markov-functional model was introduced by Hunt, Kennedy and Pelsser, [45]. The model assumes discrete forward rates and that the corresponding discount factors are a function of some underlying Markov process. The functional form of the model is specified using no-arbitrage arguments and by fitting the parameters to the initial yield curve and caplet volatilities.

A similar model to that of Hunt, Kennedy and Pelsser is the separable LIBOR Markov-functional model of Pietersz, Pelsser and Van Regenmortel, [70]. The model assumes a log-normal functional form for the discrete forward rates in a similar way to the LMM. To keep the model Markovian, however, a restriction is made to the volatility structure and a modification to the forward rate drift calculation is also made.

Markovian models are driven by a low-factor Markov process which means they are typically implemented using a lattice method. Since lattice methods work backwards in time this makes them ideally suited to pricing early exercisable products as the exercise strategy is obtained without any additional work.

Markov-functional models do have some drawbacks. In the case of the separable LIBOR Markov-functional model, the modification made to ensure the Markovian condition holds means that it can become inaccurate. For longer dated products,
particularly those with a tenor of more than 10 years, this can be an issue. Another problem is that the restriction placed on the volatility structure makes calibration more difficult than for the LMM. If, however, the product to be priced is not highly exotic and has a tenor of 10 years or less, then there is a strong case to use a Markov-functional model instead of the LMM.

For more details on Markov-functional models we refer the reader to the textbooks of Fries, [32], Hunt and Kennedy, [44] and Pelsser, [68].

1.2 The structure of the thesis

In Chapter 2 we introduce the displaced-diffusion LIBOR market model. Whilst it is difficult to cover all aspects of the model in detail, we present sufficient material to gain a basic understanding and provide references to additional information as necessary.

The original research of this thesis is contained in Chapters 3, 4, 5 and 6. Chapter 3 presents the pathwise adjoint method for computing sensitivities or Greeks in the LMM. We derive a faster method for computing both deltas and vegas, which is approximately 20 percent more efficient than the standard method. This speed improvement is achieved without additional approximations. The method is also extended to the predictor-corrector discretisation scheme, allowing for accurate Greek calculations, as demonstrated by numerical results.

Chapter 4 introduces the separable LIBOR Markov-functional model as well as providing an overview for a practical implementation. We show that it is possible to formulate the model in the spot measure and make comparisons with the terminal measure implementation.

To improve the speed of the LMM we consider using a Markov-functional model as a control variate in Chapter 5. This method has been previously suggested in the literature, however, we make many improvements, the main one being an accurate, fast and stable calibration procedure. This calibration technique enables us to achieve significant variance reductions for both prices and vegas.

In Chapter 6 we introduce a new methodology for computing Greeks in Markov-functional models, the adjoint PDE method. This method allows one to compute all of the Greeks in a PDE model in approximately the same time as it takes to compute the price. Not only is this procedure very efficient, we also demonstrate that it is accurate.

We conclude with a brief summary of the results of this thesis.
Chapter 2

The displaced-diffusion LIBOR market model

The aim of this chapter is to introduce the displaced-diffusion LIBOR market model. Most of the details presented below are taken from the textbooks of Joshi, [50], Brace, [16], Glasserman, [35] and Jäckel, [47].

2.1 Model dynamics

The LIBOR market model is based on the idea of evolving discrete market-observable forward rates. One of the strengths of the model is its ease of calibration to market-quoted vanilla instruments. Assume we have \(n\) discrete forward rates \(f_0, \ldots, f_{n-1}\) and a corresponding tenor structure \(0 < T_0 < T_1 < \ldots < T_n\), with \(\tau_i = T_{i+1} - T_i\).

The forward rate \(f_i\) is for the interval \([T_i, T_{i+1})\). The forward rates are assumed to have dynamics

\[
\frac{d(f_i(t) + \alpha_i)}{f_i(t) + \alpha_i} = \mu_i(f, t)dt + \sigma_i(t)dW_i(t),
\]

(2.1)

where the \(\alpha_i\)'s are constant displacement coefficients, \(\mu_i(f, t)\) are the drift terms, \(\sigma_i(t)\) are deterministic volatility terms and \(W_i\) are standard Brownian motions under the relevant martingale measure. The displaced-diffusion model is an extension of the log-normal model, with the displacement coefficients allowing for skew in the caplet volatility surface, a persistent market feature, see Section 14.12 of Joshi, [50].
In any case, the model collapses to the log-normal version by setting all displacement coefficients to zero.

Forward rates are typically evolved from one tenor date to the next. At each tenor date a forward rate will reset, meaning its value remains fixed, i.e. \( f_i(T_i) = f_i(T_j) \) for \( T_j \geq T_i \). To evolve the forward rates from \( T_k \) to \( T_{k+1} \) we evolve the logarithm of the rates since this yields state-independent volatility. Using \( f(k) = f(T_k) \) for the sake of brevity, we have

\[
\log(f_i(k+1) + \alpha_i) = \log(f_i(k) + \alpha_i) + \mu_i(f, k) - \frac{\text{cov}_{ij}(k)}{2} + a_i(k)Z(k + 1), \quad (2.2)
\]

where \( \text{cov}_{ij}(k) \) is the covariance of \( \log f_i \) and \( \log f_j \) across the time step, \( a_i(k) \) is the \( i \)th row of the pseudo-square root matrix \( A(k) \) and \( Z(k + 1) \) is a \( D \)-dimensional vector of independent Gaussian random variables. We will discuss how to compute the covariance and pseudo-square root matrices in Section 2.4.1.

The methodology for approximating the state-dependent drift terms that arise in the evolution (2.2)

\[
\mu_i(f, k) = \int_{T_k}^{T_{k+1}} \mu_i(f, t)dt,
\]

defines our choice of numerical method. Throughout this thesis we will work in either the spot or terminal measure. The spot measure corresponds to using the discretely-compounded money market account as numeraire. This numeraire starts with a zero-coupon bond expiring at \( T_0 \), the proceeds of which are invested in bonds expiring at \( T_1 \), which are reinvested in bonds expiring at the next tenor date, up until the last reset date \( T_{n-1} \). If we let \( \eta(t) \) denote the unique integer which satisfies

\[
T_{\eta(t)-1} \leq t < T_{\eta(t)},
\]

and thus gives the index of the next forward rate to reset, then the numeraire value is equal to

\[
N(t) = P(t, T_{\eta(t)}) \prod_{i=0}^{\eta(t)-1} (1 + \tau_i f_i(T_i)). \quad (2.3)
\]

Cash flows generated in the model are accounted for by storing the ratio of their value with the value of the numeraire.

The terminal measure uses the terminal zero-coupon bond \( P(t, T_n) \) as numeraire.
and its value is equal to

\[ N(t) = P(t, T_{\eta(t)}) \prod_{i=\eta(t)}^{n-1} \frac{1}{1 + \tau_i f_i(t)}. \]

(2.4)

A benefit of this numeraire is that it does not change during simulation, unlike the discretely-compounded money market account. However, this numeraire can be problematic during a simulation, because it produces a blow-out in the sample standard deviation, see Example 5.1 in Brace, [16]. The spot measure, however, does not usually suffer from this problem. For more details see Beveridge, [11] and Chapter 5 of Brace, [16].

We denote the spot measure as \( Q^0 \) and the terminal measure as \( Q^n \). The well-known expression for the drift under the two different measures is

\[ \mu_i(k) = \begin{cases} 
\sum_{j=\eta(k)}^{i} \frac{(f_j(k) + \alpha_j)\tau_j}{1 + f_j(k)\tau_j} a_i(k)a_j(k)^T, & \text{under } Q^0, \\
-\sum_{j=i+1}^{n-1} \frac{(f_j(k) + \alpha_j)\tau_j}{1 + f_j(k)\tau_j} a_i(k)a_j(k)^T, & \text{under } Q^n,
\end{cases} \]

(2.5)

where the drift is computed from one evolution date to the next. For proof that this is the no arbitrage drift, see the derivation in Chapter 3 of Brace, [16]. The reason for restricting ourselves to these two measures, rather than considering an intermediate measure, is that the details below can easily be extended from these two cases. It also appears that the spot and terminal measure are most commonly used in practice.

Which probability measure should we use? From a theoretical viewpoint it makes no difference to the price, but from a practical viewpoint it matters. The drift under the spot measure is positive, which can lead to drift blow-up when the covariance is large, because of positive feedback. In the terminal measure all drifts are negative and this stops the feedback. The numeraire in the terminal measure causes problems in a Monte Carlo simulation, leading to large variance and therefore slow convergence, see Beveridge, [11]. The variance issue is more significant than the small chance of drift blow-up, so we therefore recommend using the spot measure for the LMM.

When calculating the drifts we use the rapid technique from Joshi, [51]. This method works by extracting \( a_i \) from the summation, for example, in the spot measure

\[ a_i(k) \sum_{j=\eta(k)}^{i} \frac{(f_j(k) + \alpha_j)\tau_j}{1 + f_j(k)\tau_j} a_j(k)^T. \]

(2.6)
The summation terms no longer contain $i$ and we can reuse them. This means that computing the drift over one time step will require order $nD$ operations and since it is the most computationally intensive term of equation (2.2), evolving the forward rates over one time step will be order $nD$.

2.2 Drift approximations

There is a wealth of literature focused on finding fast and accurate approximations for the state-dependent drift term, see for example, Balland, [8], Glasserman and Zhao, [37], Hunter, Jäckel and Joshi, [46], Joshi and Stacey, [53], and Pietersz, Pelsser and Van Regenmortel, [70]. Here we focus on the log-Euler, predictor-corrector, [46], and Glasserman and Zhao, [37], approximations because they appear to be the most commonly used methods in practice.

2.2.1 Log-Euler

The log-Euler approach works with the logarithm of the forward rates, calculating the drift term assuming that the rates are constant across the interval $[T_k, T_{k+1})$ and equal to their value at $T_k$. Under the spot measure, for example, this corresponds to using

$$
\mu_i(k) = \sum_{j=\eta(k)}^{i} \frac{(f_j(k) + \alpha_j)\tau_j a_i(k) a_j(k)^T}{1 + f_j(k)\tau_j}. \quad (2.7)
$$

This is one of the most basic drift approximations and has been shown to be inaccurate by many authors, see for example Beveridge, Denson and Joshi, [12].

2.2.2 Predictor-corrector

A more accurate alternative to log-Euler is to use predictor-corrector (PC), introduced by Hunter, Jäckel and Joshi, [46]. This is an extension of the scheme introduced in equation (15.5.4) of Kloeden and Platen, [56]. The algorithm, quoted directly from [46], is

1. Evolve the logarithms of the forward rates as if the drifts were constant and equal to their initial values according to the log-Euler scheme.

2. Compute the drifts at the terminal time with the so evolved forward rates.

3. Average the initially calculated drift coefficients with the newly computed ones.
4. Re-evolve using the same normal variates as initially but using the new predictor-corrector drift terms.

Let a hat above a forward rate denote that it has been estimated through an initial log-Euler evolution using (2.2) with (2.7) as the drift; we define for the spot measure

\[
\mu_{i}^{PC}(k) = \frac{1}{2}\left( \sum_{j=\eta(k)}^{i} \frac{(f_j(k) + \alpha_j)\tau_j}{1 + f_j(k)\tau_j} a_i(k)a_j(k)^T + \sum_{j=\eta(k)}^{i} \frac{(f_j(k) + \alpha_j)\tau_j}{1 + f_j(k)\tau_j} a_i(k)a_j(k)^T \right),
\]

where \( f_i(k) \) is the corrected forward rate which used \( \mu_{i}^{PC}(k) \) as its drift instead of \( \mu_{i}(k) \). This method is known as iterative predictor-corrector (IPC) and has been shown to be more accurate than non-iterative PC, see Joshi and Stacey, [53]. Another benefit of working in an iterative way is that we can efficiently calculate the drift, see Section 15.6.2.3 of Andersen and Piterbarg, [6].

IPC can also be used in the spot measure, where we start at time \( \eta(t) \) and the drift \( \mu_{i} \) is calculated using a combination of the corrected values \( \hat{f}_j \) for \( j < i \) and the predicted value \( \hat{f}_i \).

2.2.3 Glasserman and Zhao

The Glasserman and Zhao method, [37], uses a change of coordinates to evolve positive martingales instead of forward rates. The idea is that martingales are driftless and we can use log-Euler stepping to evolve these quantities preserving the martingale property. The forward rates can then be recovered from the simulated quantities. We consider evolving the logarithm of the difference between adjacent deflated bonds, i.e. \( \log((P(t,T_j) - P(t,T_{j+1}))/N(t)) \), with minor adjustments to ensure positivity of discount ratios and to take account of displacements. We evolve
the logs of

\[ V_j(t) = \frac{P(t, T_j)}{N(t)} - (1 - \tau_j \alpha_j) \frac{P(t, T_{j+1})}{N(t)}, \]

\[ = \tau_j (f_j(t) + \alpha_j) \prod_{k=0}^{j} \frac{1}{1 + \tau_k f_k(t)}. \]

This method has the theoretical virtue that deflated bond prices can be made to be martingales even in the discretised model. This is not the case for other discretisation methods such as predictor-corrector which provide a very good non-martingale approximation to a martingale measure.

One problem with this method is a non-zero probability of negative discount ratios, which arises because of the way forward rates are recovered from the evolved quantities. This is a substantial problem as a negative value for a zero-coupon bond implies that one can buy something for a negative amount that will have a guaranteed value of one in the future, which is clearly an arbitrage. See [12] for a further discussion of the issues with the Glasserman and Zhao method.

2.2.4 Which drift approximation to use?

We recommend predictor-corrector, because it strikes a balance between accuracy and speed. This assertion is based on many studies assessing drift approximations, in particular, Joshi and Stacey, [53], and Beveridge, Denson and Joshi, [12]. There exist more accurate approximations, such as CANI, [53], however the extra computational burden does not justify the increased accuracy.

2.3 Products

The most liquidly traded interest rate options are caps and swaptions. An interest rate cap pays the positive difference between the forward rate and strike over a collection of intervals. The discounted payoff is given by

\[ \sum_{i=0}^{n-1} (f_i(T_i) - K_i)^+ \tau_i P(t, T_{i+1}), \]

which is made up of the sum of \( n \) caplets. Each caplet can be analytically priced using the Black formula, so the price of the cap is then the sum of the caplets. Similarly to caps, floors pay the maximum of the strike minus the forward rate. The underlying put options that make up a floor are known as floorlets.
Interest rate swaptions pay the positive difference between a swap rate and strike. The discounted payoff of a payer swaption on a swap rate over the interval \([T_i, T_n]\) is given by

\[(SR_i(T_i) - K)^+ \text{Ann}_i(T_i),\]

with a receiver swaption paying the positive difference between the strike and swap rate.

A variation of a swaption that allows the holder the right to enter into a swap on a discrete set of dates is known as a Bermudan swaption. This product is also structured as the option to break a swap, known as a cancellable swap.

An example of a more exotic product is an inverse floater. This product pays the following coupon

\[(K - 2f_i(T_i))^+ \tau_i,\]

at each of the tenor dates, \(T_i\). As interest rates go up the coupon falls and so it gets its name from having an inverse relationship with the forward rates. Most inverse floaters can be cancelled at some future time, where coupons will cease to be paid.

For examples of other products, see Chapter 13 of Brigo and Mercurio, [19].

### 2.4 Calibration

The strength of the LIBOR market model lies in its flexibility to calibrate to a large number of market instruments. Calibration to the initial yield curve and the at-the-money (ATM) caplet curve is instantaneous, whilst there is sufficient flexibility to calibrate to other market instruments.

#### 2.4.1 Volatility and correlation calibration

Throughout this thesis we will use the \(abcd\) volatility form, which assumes the instantaneous caplet volatility is given by

\[\sigma_i(t) = [a + b(T_i - t)]e^{-c(T_i - t)} + d,\]  \hspace{1cm} (2.8)

for some constants, \(a, b, c, d \in \mathbb{R}\). The benefit of this function is that it allows for the humped shape characteristic seen in market caplet volatility curves. Another advantage is its time-homogeneity, i.e. future caplet volatility curves as a function of time to expiry will look the same as they do today. For more details see Section 6.4.2 of Brigo and Mercurio, [19].
More accurate calibration methods typically use a non-parametric approach. Popular techniques are the Pedersen method, \cite{67}, the cascade algorithm, see Section 7.7 of Brace, \cite{16} and a Markovian projection method, see Piterbarg, \cite{74}. One accurate technique is that of Ametrano and Joshi, \cite{2}. Their procedure allows for calibration to both ATM caplet and swaption volatilities using stable one-dimensional optimisations.

A successful calibration routine not only fits volatilities well, but also captures the association between forward rates. A commonly used parametric form for the correlation between forward rates $i$ and $j$ is

$$
\rho_{ij} = L + (1 - L)e^{-\beta|T_i - T_j|},
$$

where $L \in [0, 1)$ and $\beta \in \mathbb{R}^+$. This structure is usually flexible enough to fit the market and we will use it throughout this thesis. For details on how to calibrate a correlation structure to the market we refer the reader to Chapter 6 of Brace, \cite{16}.

Using the correlation and instantaneous volatility we compute covariance matrices for the log forward rates across each evolution step. The covariance matrix across $[T_k, T_{k+1})$ is computed using

$$
cov_{ij}(k) = \rho_{ij} \int_{T_k}^{T_{k+1}} \sigma_i(t)\sigma_j(t)dt,
$$

where the integral for the $abcd$ form can be found in Section 12.4 of Jäckel, \cite{47}.

In practice a reduced factor model is used, where the number of factors, $D$, is less than the number of forward rates, typically $D = 3, 4$ or $5$. To factor reduce our LMM we use the method in Section 14.11 of Joshi, \cite{50}. That is, we first construct a matrix consisting of the correlation between forward rates. We then calculate the eigenvalues and eigenvectors of this matrix. For a $D$-factor model we keep the $D$ largest eigenvalues and set the remaining ones to zero. This new pseudo-correlation matrix is scaled such that it has ones on the diagonal. These new correlation numbers are then used in (2.10) to compute the reduced factor covariance matrices.

To evolve the forward rates we need the pseudo-square roots of the covariance matrices. These can be computed using the spectral decomposition in the same way as the reduced factor correlation matrices were computed. Upon inspection of the forward rate evolution (2.2), we see that it is the pseudo-square roots which drive the forward rates. We therefore define a calibration as a vector of covariance matrices and the evolution is determined by a vector of pseudo-square root matrices.
2.4. Calibration

2.4.2 Swaption approximation

To calibrate the LMM to swaptions, we need to price them quickly. One approach is to use an approximation for the swap rate volatility such as Hull and White, [43]. This approximate volatility is then inserted into the Black formula to compute the price. Ignoring drifts we have that the evolution of the swap rates are

\[ d \log SR(T_i, T_k) = \sum_{j=1}^{n-1} \frac{\partial \log SR(T_i, T_k)}{\partial \log(f_j(T_k) + \alpha_j)} d \log(f_j(T_k) + \alpha_j), \]

which, after expanding the logarithms, gives

\[ d \log SR(T_i, T_k) = \sum_{j=1}^{n-1} \frac{f_j(T_k) + \alpha_j}{SR(T_i, T_k) \partial (f_j(T_k) + \alpha_j)} d \log(f_j(T_k) + \alpha_j). \]

The idea of the approximation is to use the time zero values of the forward rates in the term

\[ U_j := \frac{f_j(T_k) + \alpha_j}{SR(T_i, T_k) \partial (f_j(T_k) + \alpha_j)}. \] (2.11)

The variance of the swap rate \( SR(T_i, T_k) \) is then approximated by this matrix and the pseudo-square roots

\[ \sum_{k=0}^{i} \sum_{f=1}^{D} \left( \sum_{j=i}^{n-1} U_j a_{jf}(k) \right)^2. \]

2.4.3 Skew calibration

This section is principally taken from Section 7.1 of Brace, [16]. From the market we are given prices for \( m \) caplets expiring at time \( T_k \) with strikes \( K_j \). The idea is to find the displacement coefficient \( \alpha_k \), which best fits this data. That is, we want to minimise the difference between the market and model caplet prices

\[ \sum_{j=1}^{m} \left( \text{Cplt}^{\text{market}}(K_j, \sigma_k, \alpha_k) - \text{Cplt}^{\text{model}}(K_j, \sigma_k, \alpha_k) \right)^2, \]

by altering the caplet volatility, \( \sigma_k \), and the displacement coefficient, \( \alpha_k \). One problem with a displaced-diffusion model is that skew, but not smile affects can be captured. If both in-the-money (ITM) and out-of-the-money (OTM) options have
higher volatilities than ATM options, we will not be able to fully capture this smile surface. The reason for this is that increasing the displacement coefficients makes the process more Gaussian and Gaussian processes are skewed to OTM options, see Section 16.6 of Rebonato, [77].

To incorporate a smile we could use a stochastic volatility LMM, see Andersen and Brotherton-Ratcliffe, [5] and Piterbarg, [71]. Some stochastic volatility models start with a displaced-diffusion LMM, where the displacement terms are used to capture the skew and the stochastic part adds curvature to the wings of the volatility smile. It has been shown that simply allowing for skew produces a reasonable calibration to the market near the strike of the option, see Rebonato, [78].

The prices of caplets in the displaced-diffusion LIBOR market model are given by the Black formula with the displacement term added to the forward rate and strike, see Section 3.1 of [16]. There is no simple way to separate out the volatility and displacement parameters, so we use non-linear regression in our calibration. One useful routine is the Levenberg-Marquardt algorithm, which is implemented in Section 15.5 of Numerical Recipes, [75].

We perform the least-squares fit for each caplet giving $n$ displacement coefficients and $n$ volatilities. The volatilities are discarded, because we would typically use a separate calibration to fit ATM caplet and swaption volatilities with the computed displacement terms. The skew calibration routine is particularly fast. For example, 40 displacement parameters were calibrated to a volatility surface comprising 20 caplets per expiry in less than a tenth of a second in our test program on a standard desktop computer$^1$.

If instead of fitting to the caplet volatility surface we wanted to fit to the coterminous swaption surface, we find the swap rate displacements $\alpha_{k}^{SR}$ and swap rate volatilities $\sigma_{k}^{SR}$ such that

$$
\sum_{j=1}^{m} (\text{Swpn}_j^{\text{market}}(K_j, \sigma_{k}^{SR}, \alpha_{k}^{SR}) - \text{Swpn}_j^{\text{model}}(K_j, \sigma_{k}^{SR}, \alpha_{k}^{SR}))^2
$$

is minimised. We can use the same algorithm as the caplet calibration, employing the Black formula to price the swaptions. Once these swap rate displacement terms are found we now need to translate them back into forward rate displacements. Recalling that the swap rate is a weighted sum of forward rates from (1.1), we have that

$$
\alpha_{i}^{SR} = \sum_{j=1}^{n-1} U_{i,j}(0)\alpha_{j}.
$$

$^1$3.16Ghz Intel CPU, with 4GB of RAM.
2.5. Simulation

Doing this for each $i$, we have in matrix notation

$$\alpha^{SR} = U\alpha.$$  

The matrix $U$ is non-singular and upper triangular so we can compute its inverse giving

$$\alpha = U^{-1}\alpha^{SR},$$  \hspace{1cm} (2.12)

which converts swap rate displacements into forward rate displacements. Note that we only have one displacement term per maturity, so we can not fit both caplet and swaption volatility surfaces. We recommend fitting the surface which will be used to hedge the exotic option, as this will increase the accuracy of the hedge.

2.5 Simulation

The LMM is implemented using Monte Carlo simulation. This is because it is a high-dimensional model and simulation methods are best suited to these problems. Monte Carlo methods can be plagued by slow convergence, however, using the techniques below will improve the convergence rate.

To simulate forward rates we need to draw uniform random variates. One method that works well is the Mersenne twister, [61]. A problem with using random numbers is that the convergence rate is of the order $N^{1/2}$ where $N$ is the number of simulations. To improve this we employ quasi-Monte Carlo, which attempts to equally disperse variates in the unit hypercube.

A popular quasi random number generator is due to Sobol, [82]. When using this we need to run $2^x - 1$ simulations for some $x$, so that the moments of the sample are correct. To use Sobol effectively careful choice of the initialisation numbers is required, otherwise it can be less effective than random Monte Carlo, see Section 8.3.4 of Jäckel, [47]. It is difficult to quote convergence rates for quasi Monte Carlo because it depends on the dimension of the problem. As a guide, rates of $N^{0.7}$ can usually be achieved. One issue with Sobol sequences is that they work best in low-dimensions. We need techniques that maximise the impact of the earlier variates, with Brownian bridging being one effective method.

We consider drawing the path for a one-dimensional Brownian motion at $n$ points. The Brownian bridge uses the first Gaussian variate to draw the final value of the path. The next variate is then used to draw the middle point of the path and so on. This has the advantage that the earlier variates have the greatest impact on the path realisation.
An issue with quasi-Monte Carlo is that we are not drawing numbers randomly, so the standard error is no longer meaningful. To look at the simulation error we need to use randomised quasi-Monte Carlo simulation (RQMC). A simple way to implement this is to break the $N$ simulation paths into batches of $m$ paths. We draw a single random number and then add this to each of the quasi-random variates until we have drawn the first $m$ paths. For the second $m$ paths we draw a new random number and add this to the quasi-random variates. We are then left with $N/m$ independent samples, which can be used to estimate the standard error. For more details of RQMC, see Section 5.4 of Glasserman, [35].

For the LMM we recommend using RQMC because it achieves a higher rate of convergence than Monte Carlo and we are able to see the standard error, unlike quasi Monte Carlo.

2.6 Early exercise

Most exotic interest rate contracts have an early exercise ingredient. These products are known as Bermudans and exercise can occur once on a discrete set of dates. Since Monte Carlo simulation works in a forwards direction it is not naturally suited to Bermudan products. The issue arises on an exercise date, where one must choose between the value upon immediate exercise (exercise value) and the value of continuing the product (continuation value). The continuation value is not freely available because we have not simulated past the current date. In some cases the exercise value may not even be easy to compute as it can depend on the present value of a complex series of future cash flows.

To deal with Bermudans we make the distinction between callable products, which give the holder the right to receive future cash flows and cancellable products, which give the issuer the right to cancel future cash flows. The more frequently traded products are cancellables. One way to price a cancellable is to decompose it as a non-Bermudan product plus a callable to enter into the opposite product. However, this decomposition suffers from the problem that the exercise value of the callable product is often not easy to compute. A solution is to work with the cancellable product directly, as suggested by Amin, [3], so that the exercise value is zero or in some cases a deterministic rebate.

There is no simple way of avoiding the computation of the continuation value, but there are a range of methods to compute it. The most accurate and reliable being a least-squares regression approach introduced by Carrière, [22], Tsitsiklis and van Roy, [85], and Longstaff and Schwartz, [59]. The idea is to run a simulation where data is collected (the first pass), which is then used to estimate regression coefficients. These coefficients will predict the continuation value. We generate a
2.6. Early exercise

We assume that the product can be exercised on the tenor dates $T_0, T_1, \ldots, T_{n-1}$. Let $CF_k \in \mathbb{R}$ denote the deflated cash flows received between the interval $[T_{k-1}, T_k)$ and $R \in \mathbb{R}$ as the deterministic rebate amount received upon exercise. The continuation value of the product at time $T_k$ is given by

$$V(T_k) = N(T_k) \sup_{s \in \Gamma_{k+1}} \mathbb{E} \left( \sum_{j=k+1}^{s} CF_j + \frac{R}{N(s)} \mid \mathcal{F}_{T_k} \right),$$

where $\Gamma_k$ is the set of exercise dates taking values $\{T_k, T_{k+1}, \ldots, T_{n-1}\}$ and $\mathcal{F}_{T_k}$ is the information up to time $T_k$. To use the regression approach we simulate the forward rates to each exercise date and compute the cash flows, storing them as we go for each of the $N$ simulations. We also store the values of the vector of basis functions, $b(T_k)$, which will be used to estimate the regression coefficients.

For ease of exposition we assume the cash flows have a natural time lag, that is they are determined at time $T_k$ and paid at time $T_{k+1}$. At the last exercise date we know the continuation value because of this time lag. Using these values for each simulation path, we compute the least-squares regression coefficients, $\beta(T_{n-1})$, by solving the system

$$\sum_{j=1}^{N} (V_j(T_{n-1}) - \beta(T_{n-1})^T b_j(T_{n-1}))^2.$$

We move on to the second last exercise date, $T_{n-2}$. Using the computed coefficients $\beta(T_{n-1})$ and the current cash flow we update the continuation value

$$V_j(T_{n-2}) = \begin{cases} CF_j(T_{n-2}) + R, & \text{if } \beta(T_{n-1})^T b_j(T_{n-1}) \leq 0, \\ CF_j(T_{n-2}) + V_j(T_{n-1}), & \text{if } \beta(T_{n-1})^T b_j(T_{n-1}) > 0, \end{cases}$$

based on the product being cancelled or continuing. This data is then used to compute regression coefficients for time $T_{n-2}$. The process is repeated backwards until the first exercise date $T_0$. The result of the first pass is a set of regression coefficients for each exercise date.

We cannot use the average of the $V_j(0)$ values as the price of the product because we have introduced foresight bias, see Fries, [33]. Instead, we run another independent simulation using the $\beta(T_k)$ coefficients to compute the continuation value at each exercise date, $T_k$. We cease the evolution of each path when the product is exercised and the price is given by the average over all second pass paths.
One improvement to the regression approach is to use the Andersen method, [4], superimposed as suggested by Bender, Kolodko and Schoenmakers, [9]. We compute the regression coefficients as above and then perform a backwards optimisation to improve the exercise strategy. We introduce a variable $H$ at each exercise date and starting at time $T_{n-1}$ the product is cancelled if

$$\beta(T_{n-1})^T b(T_{n-1}) + H(T_{n-1}) \leq 0,$$

where we perform a one-dimensional optimisation to find the best $H(T_{n-1})$. This process is then repeated through all of the exercise dates and the variables $H(T_k)$ are stored in addition to the regression coefficients. Since one-dimensional optimisations are performed quickly the method is almost as fast as the regression approach, but with the benefit of the optimisation picking up some of the value missed by the chosen set of basis functions.

There have been numerous other improvements suggested in the literature, such as a double regression approach and a method to use a generic set of basis functions, see Beveridge and Joshi, [14] and [13].
Chapter 3

LIBOR market model Greeks

In this chapter we consider the pathwise adjoint method for computing Greeks in the displaced-diffusion LIBOR market model. We introduce a speed improvement by working with the log forward rates instead of the rates themselves. This results in time savings of approximately 20 percent.

We also extend the pathwise adjoint method to the predictor-corrector discretisation scheme. Under normal market conditions the simple log-Euler scheme will usually be accurate enough. However, errors may arise during a market scenario of high volatilities, such as during the Global Financial Crisis. In volatile times accurate Greeks are essential as the risks are changing quickly. This is the sort of scenario where the log-Euler scheme could break down, resulting in inaccurate hedges. It is therefore important to have a methodology that works not just under normal market conditions, but also under more extreme conditions, as this is where knowing the true risk parameters is crucial.

There is another benefit to using the predictor-corrector pathwise method when we are already using the predictor-corrector discretisation for pricing; we can compute the price and Greeks in the same simulation. If we were using predictor-corrector in the pricing simulation and we had the log-Euler pathwise method, then we would need to run another log-Euler simulation to compute the Greeks (in the case of the LMM it is not ideal to mix a pathwise method derived from one discretisation scheme with forward rates simulated using another scheme). Therefore, having the predictor-corrector pathwise method will save an extra simulation.

This chapter is based on the papers [26] and [27].
3.1 Pathwise adjoint method

We will provide a brief introduction to the pathwise method and refer the reader to Section 7.2 of Glasserman, [35], for a more in-depth introduction. The idea is to estimate the Greeks or sensitivities on each path within a simulation, rather than performing multiple simulations with different initial parameters. That is, we interchange differentiation and expectation

\[
\frac{\partial}{\partial \theta} \mathbb{E}[g(\theta)] = \mathbb{E} \left[ \frac{\partial}{\partial \theta} g(\theta) \right],
\]

for the deflated payoff function \( g \) and the parameter of interest \( \theta \), obtaining the price and sensitivities on each path. The conditions on the payoff \( g \) that allow the interchange of differentiation and expectation can be found in Glasserman, [35]. From a practitioner’s perspective, the above equation holds when \( g \) is Lipschitz continuous. We note that there are techniques to handle a discontinuity, see Chan and Joshi, [23].

One of the initial problems of using the pathwise method to compute a large number of sensitivities to a single payoff, for example sensitivities of multiple portions of the yield curve, is that it was considered slow. To see why, assume we have an \( n \)-dimensional vector of initial inputs \( f(0) \), which for our purposes, will be forward rates. We have a discretisation scheme, which evolves this initial vector through time until the expiry of the option. By holding the random variables fixed we can write the evolution from time \( T_k \) to \( T_{k+1} \) as

\[
\log f(k) \xrightarrow{H_k} \log f(k + 1),
\]

with \( H_k: \mathbb{R}^n \to \mathbb{R}^n \). There are many choices for the function \( H_k \) as discussed in Section 2.2. We will study two within this chapter, log-Euler and predictor-corrector.

We need to compute the option’s sensitivity to the initial inputs, that is

\[
\frac{\partial g(f(k))}{\partial f(0)}.
\]

From (3.2) we can see that \( f(k) \) is a function of \( f(k - 1) \) and \( f(k - 1) \) is a function of \( f(k - 2) \), etc. We rewrite (3.3) showing all of the state variables at the simulation dates

\[
\frac{\partial g}{\partial f(0)} = \frac{\partial g}{\partial f(k)} \frac{\partial f(k)}{\partial f(k-1)} \cdots \frac{\partial f(1)}{\partial f(0)}.
\]

To evaluate equation (3.4) and hence compute the delta, we could multiply from right to left and update a Jacobian matrix at each step. Since matrix multiplication
3.2 Working with the logarithm

Giles and Glasserman, [34], made the natural choice of using the forward rates as the state variables when carrying out the differentiation. However, since we evolve the LMM in log-space another logical choice is to use the logarithm of the forward
rates. Let the vector $Y(k) := \log(f(k) + \alpha)$ denote the logarithm of the rates plus displacements coefficients. We now demonstrate how working with the log rates speeds up the Greek computations. To introduce the method we present details using the spot measure, but note that the results are similar in the terminal measure, as we will see in Section 3.4.3.

### 3.2.1 Delta

The pathwise method estimates the delta using

$$E \left[ \frac{\partial g(f(k))}{\partial f(0)} \right].$$

Focusing on the partial derivative inside the expectation, we have from the chain rule

$$\frac{\partial g(f(k))}{\partial f(0)} = \frac{\partial g(f(k))}{\partial Y(0)} \frac{\partial Y(0)}{\partial f(0)}.$$

Using the recursive equation (3.4), we now express it in terms of the log rates

$$\frac{\partial g(f(k))}{\partial f(0)} = \frac{\partial g(f(k))}{\partial f(k)} \frac{\partial f(k)}{\partial Y(k)} \frac{\partial Y(k)}{\partial Y(k-1)} \cdots \frac{\partial Y(1)}{\partial Y(0)} \frac{\partial Y(0)}{\partial f(0)}.$$

It is easier to differentiate the payoff function with respect to the rates themselves, so we apply the chain rule again

$$\frac{\partial g(f(k))}{\partial f(0)} = \frac{\partial g(f(k))}{\partial f(k)} \frac{\partial f(k)}{\partial Y(k)} \frac{\partial Y(k)}{\partial Y(k-1)} \cdots \frac{\partial Y(1)}{\partial Y(0)} \frac{\partial Y(0)}{\partial f(0)}.$$

To simplify, we note that

$$\frac{\partial f(k)}{\partial Y(k)} = \frac{\partial f(k)}{\partial f(k)} = \frac{\partial f(k)}{\partial Y(k)}$$

and

$$\frac{\partial Y(0)}{\partial f(0)} = \begin{bmatrix} \frac{1}{f(0)+\alpha} & \cdots & \frac{1}{f(n-1)+\alpha} \end{bmatrix} = \text{diag} \left( \frac{1}{f(0)+\alpha} \right),$$

giving the final expression

$$\frac{\partial g}{\partial f(0)} = \frac{\partial g}{\partial f(k)} \text{diag}(f(k)+\alpha) \frac{\partial Y(k)}{\partial Y(k-1)} \cdots \frac{\partial Y(1)}{\partial Y(0)} \text{diag} \left( \frac{1}{f(0)+\alpha} \right). \quad (3.5)$$
Equation (3.5) is used to compute the delta, updating a vector from left to right. To do so we need to know the derivative of the log forward rates at time $T_{k+1}$ with respect to the log forward rates at time $T_k$. After differentiating the log evolution scheme (2.2) with the Euler drift approximation in the spot measure (2.7) we obtain

$$\frac{\partial \log(f_i(k+1) + \alpha_i)}{\partial \log(f_j(k) + \alpha_i)} = \begin{cases} 
1, & i = j < \eta(k), \\
1 + \frac{(f_i(k+1) + \alpha_i)(1 - \alpha_i \tau_i) a_i^T}{(1 + \tau_i f_i(k))^2}, & i = j \geq \eta(k), \\
\frac{(f_j(k) + \alpha_j)(1 - \alpha_j \tau_j) a_j^T}{(1 + \tau_j f_j(k))^2}, & i > j \geq \eta(k), \\
0, & \text{otherwise}.
\end{cases}$$

Using these derivatives we can develop a formula for the delta in the same way as Giles and Glasserman. We define

$$V(k) = \frac{\partial g(f(k))}{\partial f(k)},$$

and

$$\tilde{V}(k) = \text{diag}(f(k) + \alpha)V(k).$$

The delta can be computed recursively, starting at the final time $T_{n-1}$ and working backwards using

$$V(k) = \left( \frac{\partial f(k+1)}{\partial f(k)} \right) V(k+1),$$

or, in the case of log rates

$$\tilde{V}(k) = \left( \frac{\partial Y(k+1)}{\partial Y(k)} \right) \tilde{V}(k+1).$$

Using the rates themselves as state variables this means

$$V_i(k) = \frac{(f_i(k+1) + \alpha_i)V_i(k+1)}{f_i(k) + \alpha_i} + \frac{a_i \tau_i(1 - \alpha_i \tau_i)}{(1 + \tau_i f_i(k))^2} \sum_{j=i}^{n-1} (f_j(k+1) + \alpha_j)V_j(k+1)a_j^T,$$

(3.6)

compared to the simpler expression when working with the log-rates

$$\tilde{V}_i(k) = \tilde{V}_i(k+1) + \frac{(f_i(k) + \alpha_i)a_i \tau_i(1 - \alpha_i \tau_i)}{(1 + \tau_i f_i(k))^2} \sum_{j=i}^{n-1} \tilde{V}_j(k+1)a_j^T,$$

(3.7)
with the initial delta given by $V(0)$ and

$$V(0) = \text{diag} \left( \frac{1}{f(0) + \alpha} \right) \tilde{V}(0).$$

We can store the inverse values of the displaced rates $f(0) + \alpha$ initially and then only perform multiplication, since it is 10 to 20 times faster than division in a computer program, see [34].

The delta estimate using (3.7) is identical to (3.6) on every path (up to floating point error). We emphasise this as it means that working with the log-rates or the rates themselves gives exactly the same Greeks; we are not introducing any approximations.

Using the log-rates as the state variables instead of the rates produces a simpler and more efficient expression. This is important because we need to perform the recursive delta calculation on each simulation path. The summation term is calculated $nD$ times per evolution step and the term outside the summation $n$ times per step. We have to do the extra multiplication by the final rates and the inverse initial rates, however this is only required once for each path. Given that the number of forward rates can be greater than 40 and usually $D = 3$ to 5, simplifying a calculation that is performed $nD$ times for $n$ time steps will significantly outweigh the extra time spent multiplying by the final and initial rates.

### 3.2.2 Vega

We have seen how working with the log-rates simplifies the delta calculation, we now derive the method for vega. Assume we need the sensitivity to a model parameter, $\theta$. That is we want to compute

$$\frac{\partial g(f(k))}{\partial \theta} = \frac{\partial g(f(k))}{\partial f(k)} \frac{\partial f(k)}{\partial \theta}.$$  

Using the chain rule to write part of the expression in terms of log-rates gives

$$\frac{\partial f(k)}{\partial \theta} = \frac{\partial f(k)}{\partial Y(k)} \frac{\partial Y(k)}{\partial \theta}.$$  

Now taking the full expression for $Y(k)$ in terms of the map $H_k$ defined in (3.2) we have

$$\frac{\partial f(k)}{\partial \theta} = \frac{\partial f(k)}{\partial Y(k)} \left( \frac{\partial Y(k)}{\partial Y(k-1)} \frac{\partial Y(k-1)}{\partial \theta} + \frac{\partial H_{k-1}}{\partial \theta} \right).$$
We can use this recursively, arriving at a similar expression to that of Giles and Glasserman

\[
\frac{\partial g}{\partial \theta} = \sum_{k=0}^{T-1} \tilde{V}(k+1)^T B(k),
\]

(3.8)

where the \( \tilde{V}(k) \) are computed during the delta calculation by (3.7) and

For the purposes of the LMM our model volatility parameter \( \theta \) will generally be an element of a pseudo-square root matrix. These elementary vegas will need to be converted into vegas of market traded products using a method such as Joshi and Kwon, [52]. Differentiating the parts of \( H_k \) with respect to the pseudo-square root element \( a_{ij}(k) \) (the \( j \)th row of the \( f \)th column for time step \( T_k \)) gives

\[
\frac{\partial}{\partial a_{ij}(k)} \mu_i(k) = \begin{cases} a_{ij}(k) \frac{f_i(k)+\alpha_i}{1+\gamma_i f_i(k)} + \sum_{l=\eta(k)}^{i} a_{lj}(k) \frac{f_l(k)+\alpha_l}{1+\gamma_l f_l(k)}, & i = j \geq \eta(k), \\ a_{ij}(k) \frac{f_i(k)+\alpha_i}{1+\gamma_i f_i(k)}, & i > j \geq \eta(k), \\ 0, & \text{otherwise}, \end{cases}
\]

for the drift term. The derivative of the drift correction term is

\[
-\frac{1}{2} \frac{\partial}{\partial a_{ij}(k)} \text{cov}_{ii}(k) = -\frac{1}{2} \frac{\partial}{\partial a_{ij}(k)} \sum_{f=1}^{D} a_{if}(k)^2 = \begin{cases} -a_{ij}(k), & \text{if } i = j \geq \eta(k), \\ 0, & \text{otherwise}. \end{cases}
\]

Finally, the derivative of the random term is

\[
\frac{\partial}{\partial a_{ij}(k)} \sum_{f=1}^{D} a_{if}(k) Z_f(k+1) = \begin{cases} Z_f(k+1), & \text{if } i = j \geq \eta(k), \\ 0, & \text{otherwise}. \end{cases}
\]

These derivatives are the same when working with the rates themselves, except in that case the current displaced rates, \( f_i(k+1) + \alpha_i \), are multiplied through at each step. Therefore, the computational time for the vega calculations will be quicker when working with the log-rates as we avoid these multiplications at each step.

Using the above derivatives we can develop a formula for the vega calculation. The vega for row \( i \), column \( f \) of the pseudo-square root at time step \( T_k \) is computed
recursively using

$$\left(S_{ij}(k) - a_{ij}(k) + Z_f(k+1)\right)\tilde{V}_i(k + 1) + \left(f_i(k) + \alpha_i\right)\tau_i \sum_{j=i}^{n-1} \tilde{V}_j(k + 1)a_{jj}(k),$$

(3.9)

compared to working with the rates themselves

$$\left(S_{ij}(k) - a_{ij}(k) + Z_f(k+1)\right)\left(f_i(k + 1) + \alpha_i\right)V_i(k + 1) + \left(f_i(k) + \alpha_i\right)\tau_i \sum_{j=i}^{n-1} \left(f_j(k + 1) + \alpha_i\right)V_j(k + 1)a_{jj}(k),$$

(3.10)

where

$$S_{ij}(k) = \sum_{j=\eta(k)}^{i} \frac{\left(f_j(k) + \alpha_j\right)\tau_j}{1 + \tau_j f_j(k)} a_{jj}(k).$$

(3.11)

The vega estimate using (3.9) is identical to (3.10) on every path, but it is more computationally efficient as we will demonstrate in Section 3.2.4. We note that the matrices $S(k)$ are used to efficiently compute the drift in (2.6) and will therefore already be computed.

### 3.2.3 Skew

Giles and Glasserman did not consider the sensitivity to the skew parameters, $\alpha$, because they were not working with a displaced-diffusion model. When computing the displacement sensitivities it is easier to work with the rates themselves, instead of the log rates. If we look at (2.2) we have $\alpha_i$ on both sides of the equation, however, for the rates themselves we have

$$f_i(k + 1) = \left(f_i(k) + \alpha_i\right)\exp\left(\mu_i(f, k) - \frac{\text{cov}_{ii}(k)}{2} + a_i(k)Z(k + 1)\right) - \alpha_i.$$

To compute the sensitivities we can use the same equation used to derive the vega (3.8), where $\tilde{V}$ is replaced by $V$. Our parameter $\theta$ is now an $\alpha_j$, which on differentiation gives

$$\frac{\partial f_i(k + 1)}{\partial \alpha_j} = \begin{cases} \frac{\left(f_i(k+1)+\alpha_i\right)\tau_i}{1 + \tau_i f_i(k)} a_i(k)a_i(k)^T + \frac{f_i(k+1)+\alpha_i}{f_i(k)+\alpha_i} - 1, & i = j \geq \eta(k), \\ \frac{\left(f_i(k+1)+\alpha_i\right)\tau_j}{1 + \tau_j f_j(k)} a_i(k)a_j(k)^T, & i \geq j \geq \eta(k), \\ 0, & \text{otherwise}. \end{cases}$$
3.2. Working with the logarithm

Writing out the adjoint equation, the sensitivity of the \( i \)th displacement coefficient at step \( T_k \) is equal to

\[
\left( \frac{f_i(k + 1) + \alpha_i}{f_i(k) + \alpha_i} - 1 \right) V_i(k + 1) + \frac{\tau_i}{1 + \tau_i f_i(k)} a_i(k) \sum_{j=1}^{n} (f_j(k) + \alpha_j) a_j(k)^T V_j(k + 1),
\]

which can be computed in order \( nD \) operations per step.

3.2.4 Rates and log-rates comparison

Working with the log-rates produces simpler delta and vega expressions; here we demonstrate that the computational time is 20 percent quicker. Consider pricing five- and ten-year interest rate caps as well as a ten-year Bermudan swaption, where we simulate forward rates to each tenor date (see Section 3.3 on how to handle early exercise in the pathwise method). All products have quarterly forward rates and flat volatility. We use both three- and five-factor models. Pathwise delta and vega estimates were computed 5 times for each product using 1 million simulations each time.

Table 3.1 presents the average multiple of time taken when working with the rates themselves instead of the log-rates. We show results for the delta and vega calculations individually and combined. Also included are the standard errors of the 5 averages. Only the actual computation of the Greek estimates were clocked, other computations, such as the time taken to evolve the rates, were not included. That is, we timed the computation of (3.6) and (3.7) for the delta, and (3.9) and (3.10) for the vega. For example, in the case of a five-factor ten-year Bermudan swaption we see that working with the log-rates to compute the delta and vega is 17 percent quicker than working with the rates themselves.

<table>
<thead>
<tr>
<th>Product</th>
<th>factors</th>
<th>delta</th>
<th>vega</th>
<th>combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y Cap</td>
<td>3</td>
<td>1.27x (1.1E-02)</td>
<td>1.12x (3.5E-03)</td>
<td>1.19x (8.4E-03)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.21x (1.2E-02)</td>
<td>1.17x (1.5E-02)</td>
<td>1.20x (7.8E-03)</td>
</tr>
<tr>
<td>10Y Cap</td>
<td>3</td>
<td>1.25x (1.0E-03)</td>
<td>1.12x (3.5E-03)</td>
<td>1.18x (1.9E-03)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.47x (5.4E-03)</td>
<td>1.35x (5.6E-03)</td>
<td>1.10x (3.7E-03)</td>
</tr>
<tr>
<td>10Y Bermudan</td>
<td>3</td>
<td>1.19x (5.1E-03)</td>
<td>1.16x (1.0E-02)</td>
<td>1.17x (7.3E-03)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.10x (3.7E-03)</td>
<td>1.15x (6.7E-03)</td>
<td>1.17x (2.5E-03)</td>
</tr>
</tbody>
</table>

Table 3.1: Average multiple of time taken (and standard error) by working with the forward rates instead of the log forward rates

The speed improvements for the delta and vega combined are comparable for all of the products at approximately 20 percent. The log-rates algorithm is still order \( nD \) per step, but is clearer and more efficient. The time savings are achieved without using additional approximations as the Greeks are identical to those from the Glasserman and Giles method on every path.
3.3 Early exercise pathwise Greeks

Bermudan products present a problem for the pathwise method. The issue surrounds the sensitivity of the exercise strategy to the initial model parameters. Fortunately we are able to ignore the effect of the changed parameters on the time of exercise when computing first order Greeks, see Piterbarg, [72].

As we saw in Section 2.6, Bermudan products can be viewed in a number of different ways:

1. The right to enter into a contract.
2. The right to receive the present value of a contract.
3. The right to break a contract.

The first two are clearly equivalent, and the last can be transformed into the first simply by considering the unbreakable contract minus the breakable contract. These lead to three different ways of computing adjoint Greeks:

1. Using the method of Leclerc, Liang and Schneider, [57], where we enter into a product at the exercise time and continue simulating until the final coupon date, accumulating the differentiated coupons.
2. Working with the derivatives of the net present value of the remaining coupons at the exercise date.
3. Accumulating the derivatives of the coupons up to the exercise date.

The first method has the disadvantage that all paths must be simulated until the final tenor date and then the Greeks need to be backwards updated through all dates. Methods two and three, however, permit the cessation of a path at the time of exercise. Method two is useful when the net present value of the remaining contract is easily computable, which is the case for Bermudan swaptions (this is the method used in the previous section). The final method is preferable for complicated contracts where the net present value is not easily computed.

To obtain the adjoint delta for a product that is exercised at time $T$, we start by setting

$$V(r) = \frac{\partial g}{\partial f(r)},$$  \hspace{1cm} (3.12)

that is the derivative of the deflated payoff with respect to the vector of forward rates at the exercise time. We then update this $n$-dimensional vector at each simulation date back until the starting time, giving the initial delta, $V(0)$. 
3.4 Predictor-corrector discretisation

To derive the pathwise adjoint Greeks we needed to specify a discretisation scheme. Giles and Glasserman chose log-Euler and we have so far done the same. In this section we shift our attention to predictor-corrector, showing how to drive deltas and vegas and compare their accuracy with the log-Euler scheme.

3.4.1 Predictor-corrector delta

To derive the delta using log-Euler it is easy to consider the evolution of the forward rates as the single map
\[
\begin{align*}
\log(f(k) + \alpha) & \xrightarrow{H_k} \log(f(k + 1) + \alpha),
\end{align*}
\]

because of the simple drift approximation. However, for the predictor-corrector evolution, it will be easier to break the problem into four sub-maps, each sub-map updating a vector. Here we build on the ideas introduced by Joshi and Yang, [55]. We use the following sub-maps
\[
\begin{align*}
\log(f(k) + \alpha) & \xrightarrow{H_{k,0}} \begin{bmatrix} \log(f(k) + \alpha) \\ \mu(k) \end{bmatrix} \xrightarrow{H_{k,1}} \begin{bmatrix} \log(f(k) + \alpha) \\ \mu(k) \\ \hat{f}(k) \end{bmatrix}, \\
\log(f(k) + \alpha) & \xrightarrow{H_{k,2}} \begin{bmatrix} \log(f(k) + \alpha) \\ \mu(k) \\ \hat{\mu}(k) \end{bmatrix} \xrightarrow{H_{k,3}} \begin{bmatrix} \log(f(k) + 1) + \alpha \end{bmatrix},
\end{align*}
\]

for step \( T_k \). This is how forward rates would be evolved in a computer program, where we start with the vector of log forward rates, compute the drifts, evolve the rates, compute the corrected drifts and then re-evolve the rates, as discussed in the predictor-corrector algorithm of Section 2.2.2.

Due to the manageable form of log-Euler drift, we were easily able to differentiate the single map at each step, producing a recursive formula for the delta vector \( \tilde{V} \). The extra complexity of predictor-corrector requires us to differentiate each of the four sub-maps individually. This produces Jacobian matrices that can be used to update the vector \( \tilde{V} \). When we update \( \tilde{V} \) we will now need four sets of computations between each simulation date instead of one.

To derive the method we start with the last sub-map (as we are updating in a backwards direction) \( H_{k,3} : \mathbb{R}^{3n} \to \mathbb{R}^n \) and differentiate. This gives an \( n \times 3n \)
Jacobian matrix, with elements

\[
\begin{bmatrix}
1 & \frac{1}{2} & \frac{1}{2} \\
\vdots & \ddots & \ddots \\
1 & \frac{1}{2} & \frac{1}{2}
\end{bmatrix},
\]

where blanks are equal to zero. The fractions above originate from the predictor-corrector drift approximation being composed of half the log-Euler and predicted drift estimates. Differentiating the second last sub-map \(H_{k,2}: \mathbb{R}^{3n} \rightarrow \mathbb{R}^{3n}\) gives a \(3n \times 3n\) Jacobian matrix

\[
\begin{bmatrix}
1 & \cdots & \cdots & \cdots \\
\vdots & \ddots & \ddots & \cdots \\
1 & \cdots & \cdots & \cdots \\
\times & \cdots & \cdots & \cdots
\end{bmatrix},
\]

where the sub-matrix of \(\times\)'s is \(n \times n\) and lower triangular. It appears because \(H_{k,2}\) updates the predicted drift terms, \(\hat{\mu}(k)\) and its entries are

\[
\frac{\partial \hat{\mu}_i(k)}{\hat{f}_j(k)} = \begin{cases} 
\frac{\tau_i (1 - \alpha_i \tau_j)}{(1 + \tau_j f_j(k))^2} a_i(k) a_j(k)^T, & i \geq j \geq \eta(k), \\
0, & \text{otherwise}.
\end{cases}
\]

Differentiating the sub-map \(H_{k,1}: \mathbb{R}^{2n} \rightarrow \mathbb{R}^{3n}\), which updates the predicted rates \(\hat{f}(k)\), gives a \(3n \times 2n\) Jacobian matrix

\[
\begin{bmatrix}
1 & \cdots & \cdots & \cdots \\
\vdots & \ddots & \ddots & \cdots \\
\times & \cdots & \cdots & \cdots \\
\times & \cdots & \cdots & \cdots
\end{bmatrix},
\]

where the bottom left diagonal elements are

\[
\frac{\partial \hat{f}_i(k)}{\partial \log(f_j(k) + \alpha_j)} = \begin{cases} 
\hat{f}_i(k) + \alpha_i, & i = j \geq \eta(k), \\
0, & \text{otherwise},
\end{cases}
\]
and the bottom right diagonal elements are
\[
\frac{\partial \hat{f}_i(k)}{\partial \mu_j(k)} = \begin{cases} 
\hat{f}_i(k) + \alpha_i, & i = j \geq \eta(k), \\
0, & \text{otherwise}. 
\end{cases}
\]

Finally, differentiating the first sub-map $H_{k,0}$: $\mathbb{R}^n \to \mathbb{R}^{2n}$ gives a $2n \times n$ Jacobian matrix
\[
\begin{bmatrix}
1 \\
. & . \\
. \\
\times \\
. \\
\times & \ldots & \times
\end{bmatrix},
\]
with entries arising because $H_{k,0}$ updates the log-Euler drifts $\mu(k)$
\[
\frac{\partial \mu_i(k)}{\partial \log(f_j(k) + \alpha_j)} = \begin{cases} 
(f_j(k) + \alpha_j)^{\tau_j(1-\alpha_j\tau_j)/(1+\tau_jf_j(k))} a_i(k)a_j(k)^T, & i \geq j \geq \eta(k), \\
0, & \text{otherwise}. 
\end{cases}
\]

Now that we have differentiated each of the four sub-maps, we can compute the delta. One approach is to multiply the vector $\tilde{V}(k+1)$ by these four Jacobian matrices, however, given that the majority of entries are either zero or one, this will be inefficient. A better method is to only update the vector based on the elements that change between each sub-map, which we now explain.

Looking at the Jacobian matrix for $H_{k,3}$, we see that it contains three sub-matrices all of which are diagonal. It is straightforward to update the delta vector at this step and will only require order $n$ operations.

The next Jacobian matrix, $H_{k,2}$, contains approximately $n^2/2$ non-zero entries. Multiplying a vector by this matrix will be order $n^2$. However, given the anatomy of the lower-triangular sub-matrix, we can compute a summation term with order $nD$ computations, which makes the vector multiplication at this sub-step order $nD$.

The summation term we compute is the same term that makes the log-Euler delta (3.7) order $nD$. That is, for delta $i$ at step $k$ we compute
\[
\sum_{j=i}^{n-1} \tilde{V}_j(k+1)a_j(k)^T,
\]
where the summation is calculated recursively, starting with $V_{n-1}(k+1)a_{n-1}(k)^T$. 
We then use this sum multiplied by the coefficient
\[
\frac{\tau_i (1 - \alpha_i \tau_i)}{(1 + \tau_i f_i(k))^2} a_i(k),
\]
to obtain the updated delta vector.

The Jacobian matrix at \(H_{k,1}\), similar to that at \(H_{k,3}\), is made up of sub-matrices which are diagonal, so updating the vector at this step is elementary and will only require order \(n\) operations. At the step corresponding to \(H_{k,0}\) we need to compute another summation term to ensure the overall computational order is \(nD\). We compute
\[
\sum_{j=i}^{n-1} \tilde{V}_j(k + 1, 1) a_j(k)^T,
\]
where \(\tilde{V}_j(k+1,1)\) is the delta vector, backwards updated to the sub-map \(H_{k,1}\). Using this summation multiplied by
\[
(f_i(k) + \alpha_i) \frac{\tau_i (1 - \alpha_i \tau_i)}{(1 + \tau_i f_i(k))^2} a_i(k),
\]
gives the delta \(\tilde{V}_i(k)\). We simply repeat until we have \(V(0)\).

We have introduced a method which is order \(nD\) per step retaining the same computational order as log-Euler whilst achieving greater accuracy.

In presenting the above Jacobian matrices, we have not considered forward rates which have reset. These can be easily accommodated as the elements of the delta vector \(\tilde{V}\) which correspond to reset rates will not change across a time step and so do not need to be updated.

### 3.4.2 Predictor-corrector vega

We now derive the vega for the predictor-corrector discretisation scheme and we do so in a similar way as the delta. The pathwise estimate of the sensitivity to a pseudo-square root element at step \(T_k < T_r\) is given by
\[
\frac{\partial g(f(r))}{\partial a_{jj}(k)} = \frac{\partial g(f(r))}{\partial f(r)} \frac{\partial f(r)}{Y(r)} \frac{Y(r)}{\partial a_{jj}(k)} = \frac{\partial Y(r)}{\partial a_{jj}(k)} \frac{\partial Y(r)}{\partial Y(r-1)} \cdots \frac{\partial Y(k)}{\partial a_{jj}(k)} = \tilde{V}(k) \frac{\partial Y(k)}{\partial a_{jj}(k)}.
\]
To compute the partial derivatives of the forward rates with respect to the pseudo-square root elements, we include them in (3.13),

\[
\begin{align*}
  A(k) & \log(f(k) + \alpha) \\
  \log(f(k) + \alpha) & \mu(k) \\
  \mu(k) & \hat{f}(k)
\end{align*}
\]

\[
H_{k,0} \rightarrow \begin{bmatrix}
  A(k) \\
  \log(f(k) + \alpha) \\
  \mu(k)
\end{bmatrix}
\]

\[
H_{k,1} \rightarrow \begin{bmatrix}
  A(k) \\
  \log(f(k) + \alpha) \\
  \mu(k) \\
  \hat{f}(k)
\end{bmatrix}
\]

\[
H_{k,2} \rightarrow \begin{bmatrix}
  A(k) \\
  \log(f(k) + \alpha) \\
  \mu(k) \\
  \hat{\mu}(k)
\end{bmatrix}
\]

\[
H_{k,3} \rightarrow \begin{bmatrix}
  A(k) \\
  \log(f(k) + \alpha) \\
  \mu(k) \\
  \hat{\mu}(k)
\end{bmatrix}
\]

(3.14)

We proceed in the same manner as the delta computation, that is we multiply \( \hat{V}(k) \) backwards through the Jacobian matrices, this time updating the vector of vega sensitivities. The matrices will now have an additional \( nD \) more rows and columns to accommodate the \( nD \) pseudo-square root elements, with the bottom portion of the Jacobian matrices remaining unchanged from the delta section above.

The only extra calculations we need are the derivatives of the sub-maps with respect to the pseudo-square root entries. We do not graphically present the form of the Jacobian matrices, because they are easy to infer from the derivatives below.

Differentiating \( H_{k,3} : \mathbb{R}^{nD+3n} \rightarrow \mathbb{R}^{n} \) gives,

\[
\frac{\partial \log(f_i(k+1) + \alpha_i)}{\partial a_{ji}(k)} = \begin{cases} 
-a_{ij}(k) + Z_f(k+1), & i = j \geq \eta(k), \\
0, & \text{otherwise.}
\end{cases}
\]

For the previous sub-map \( H_{k,2} : \mathbb{R}^{nD+3n} \rightarrow \mathbb{R}^{nD+3n} \), we have

\[
\frac{\partial \hat{\mu}_i(k)}{\partial a_{ji}(k)} = \begin{cases} 
a_{ij}(k) \left( \frac{\hat{f}_i(k) + \alpha_i}{1+\tau_i f_i(k)} \right) + \hat{S}_{ij}(k), & i = j \geq \eta(k), \\
a_{ij}(k) \left( \frac{\hat{f}_j(k) + \alpha_j}{1+\tau_j f_j(k)} \right), & i > j \geq \eta(k), \\
0, & \text{otherwise.}
\end{cases}
\]

where \( \hat{S}_{ij}(k) \) is the same as (3.11), however the quantities \( f_i(k) \) are replaced with \( \hat{f}_i(k) \). These terms will have been computed during the predictor-corrector evolution so we can reuse them. For the sub-map \( H_{k,1} : \mathbb{R}^{nD+2n} \rightarrow \mathbb{R}^{nD+3n} \), we compute the partial derivatives to be

\[
\frac{\partial \hat{f}_i(k)}{\partial a_{ji}(k)} = \begin{cases} 
(\hat{f}_i(k) + \alpha_i)(-a_{ij}(k) + Z_f(k+1)), & i = j \geq \eta(k), \\
0, & \text{otherwise.}
\end{cases}
\]
Finally, for the first sub-map $H_{k,0}: \mathbb{R}^{nD+n} \to \mathbb{R}^{nD+2n}$, we have an expression which is very similar to the derivative of $H_{k,2}$, which we would expect given the resemblance of $\mu(k)$ to $\hat{\mu}(k)$:

$$
\frac{\partial \mu_i(k)}{\partial a_j f(k)} = \begin{cases} 
a_i f(k) \frac{(f_i(k)+\alpha_i) \tau_i}{1+\tau_i f_i(k)}, & i = j \geq \eta(k), \\
a_i f(k) \frac{(f_i(k)+\alpha_j) \tau_j}{1+\tau_j f_j(k)}, & i > j \geq \eta(k), \\
0, & \text{otherwise},
\end{cases}
$$

with $S_{if}(k)$ defined in (3.11). By computing the vegas in this way we also obtain the deltas at the same time. Analogously to updating the delta vector $\tilde{V}(k+1)$ we only compute products with non-zero elements when calculating the vega vector.

The overall computational order is the same as the log-Euler vegas, that is $nD$ per step. We will see in Section 3.4.6 that the predictor-corrector vegas are more accurate than the log-Euler vegas.

### 3.4.3 Deltas and vegas in the terminal measure

We have so far focused on the spot measure and for completeness we briefly show how to compute deltas and vegas in the terminal measure. One difference is that the numeraire in the terminal measure depends on the initial forward rates. As we saw in (3.1), the pathwise method interchanges expectation and differentiation. If we decompose our deflated option price $g$ at exercise time $T_r$ into the undeflated value $c$ and numeraire $N$, we have from the fundamental theorem of asset pricing

$$
c(0) \frac{N(f(0))}{N(f(0))} = \mathbb{E} \left[ \frac{c(f(r))}{N(f(r))} \right],
$$

where the expectation is taken in the relevant martingale measure. We recall from (2.4) that under the terminal measure the initial value of the numeraire is given by

$$
N(0) = P(0, T_n) = P(0, T_0) \prod_{i=0}^{n-1} \frac{1}{1 + \tau_i f_i(0)}.
$$

When it comes to differentiating the payoff with respect to the initial forward rates we need to take their dependence in the numeraire into account. From the product rule

$$
\frac{\partial}{\partial f(0)} N(f(0)) \mathbb{E}[g(f(r))] = N(f(0)) \mathbb{E} \left[ \frac{\partial g(f(r))}{\partial f(0)} \right] + \mathbb{E}[g(f(r))] \frac{\partial N(f(0))}{\partial f(0)},
$$

\[\text{LIBOR market model Greeks}\]
where the derivative of the initial value of the numeraire is easily computed as
\[
\frac{\partial N(f(0))}{\partial f_j(0)} = \frac{-\tau_j}{1 + \tau_i f_i(0)} N(f(0)).
\]

The term we estimate using the pathwise method is
\[
\mathbb{E}\left[ \frac{\partial g(f(\tau))}{\partial f(0)} \right],
\]
so we subtract a portion of the final price from this to get the delta estimate
\[
\frac{\partial g}{\partial f_j(0)} = \mathbb{E}\left[ \frac{\partial g(f(\tau))}{\partial f_j(0)} \right] - \frac{\tau_j}{1 + \tau_j f_j(0)} \mathbb{E}[g(f(\tau))].
\]
Note that we do not have to make any adjustments to the pathwise delta in the spot measure as the initial value of the numeraire does not depend on the initial forward rates.

### 3.4.4 Terminal measure delta

Since the form of the drift in the terminal measure is different (the drift of forward rate \( i \) depends on the forward rates above \( i \) compared to the spot measure where the drift of forward rate \( i \) depends on non-reset rates including and before \( i \) the \( n \times n \) Jacobian matrix
\[
D(k) := \frac{\partial \log(f(k+1) + \alpha)}{\partial \log(f(k) + \alpha)},
\]
will have the structure of
\[
D = \begin{bmatrix}
1 & 1 & 1 & \cdots \\
1 & 1 & \times & \times & \times \\
1 & \times & \times & \cdots \\
1 & \times & \times & \times & \times \\
1 & \times & \times & \times & \times \\
\end{bmatrix}, \quad D^T = \begin{bmatrix}
1 & 1 & 1 & \cdots \\
1 & \times & \times & \cdots \\
1 & \times & \times & & \\
1 & \times & \times & \times & \\
\end{bmatrix}.
\]

Compare this to the form of \( D \) for the spot measure in Giles and Glasserman, [34], where it has the same structure of non-zero elements as \( D^T \) above. The entries of
the matrices $D$ are

$$\frac{\partial \log(f_i(k+1) + \alpha_i)}{\partial \log(f_j(k) + \alpha_i)} = \begin{cases} 1, & i = j, \\ -\frac{(f_i(k)+\alpha_i)\tau_i(1-a_j\tau_j)}{(1+\tau_j f_j(k))^2}a_i(k)a_j(k)^T, & \eta(k) < i < j, \\ 0, & \text{otherwise}. \end{cases}$$

Using this, we can derive the adjoint method, which for the log-Euler evolution gives

$$\tilde{V}_i(k) = \tilde{V}_i(k+1) - \frac{(f_i(k) + \alpha_i)\tau_i(1 - \alpha_i\tau_i)}{(1 + \tau_i f_i(k))^2}a_i(k) \sum_{j=\eta(k)}^{i-1} \tilde{V}_j(k+1)a_j(k)^T. \quad (3.15)$$

This is similar to the adjoint delta for the spot measure (3.7), with the main differences being the summation indices and the minus sign in the coefficient of the sum.

We do not derive the predictor-corrector delta, because the methodology is the same as for the spot measure. As we saw above the distinguishing features are the differently shaped Jacobian matrices and the negative coefficient of the drift terms.

### 3.4.5 Terminal measure vega

We do not derive the vega for the log-Euler evolution, but instead simply present its formula. The vega for row $i$ column $f$ of the pseudo-square root at time $T_k$ is computed recursively by

$$(S_{if}(k) - a_{if}(k) + Z_f(k+1))\tilde{V}_i(k+1) - \frac{(f_i(k) + \alpha_i)\tau_i}{1 + \tau_i f_i(k)} \sum_{j=\eta(k)}^{i-1} \tilde{V}_j(k+1)a_{jf}(k),$$

where the terms

$$S_{if}(k) = -\sum_{j=i+1}^{n-1} \frac{(f_j(k) + \alpha_j)\tau_j}{1 + \tau_j f_j(k)}a_{jf}(k),$$

are comparable to (3.11) and will have been computed in the efficient evolution of the forward rates.

We can derive the predictor-corrector vega in a similar way to the spot measure. The same sub-maps as (3.14) are used, with the only change being the formula for the drift. The derivatives of the sub-maps $H_{k,3}$ and $H_{k,1}$ are identical under the
3.4. Predictor-corrector discretisation

terminal measure, so we do not repeat them here. For $H_{k, 2}$ the derivatives are

$$\frac{\partial \hat{\mu}_i(k)}{\partial a_{jf}(k)} = \begin{cases} \hat{S}_{if}(k), & i = j \geq \eta(k), \\
a_{if}(k) \frac{\mu_i(k) + \alpha_j \tau_i}{1 + \tau_j f_j(k)}, & \eta(k) \leq i < j, \\
0, & \text{otherwise}. \end{cases}$$

For the derivative of the sub-map $H_{k, 0}$ the only difference with the terms above is that $\hat{S}_{if}(k)$ and $f_j(k)$ are replaced by $S_{if}(k)$ and $f_j(k)$. This gives an order $nD$ per step algorithm to compute the vegas using the predictor-corrector discretisation in the terminal measure.

3.4.6 Log-Euler and predictor-corrector Greeks comparison

In this section we compare the pathwise derivative estimates for log-Euler and predictor-corrector. We also examine how the choice of probability measure impacts the calculations. To do so two products are considered: an interest rate cap and a lockout Bermudan swaption. We want to compare the simulated deltas and vegas of the caplets underlying the cap to the analytic values from the Black formula. If either the log-Euler or predictor-corrector drift formulas were not approximations then we would only expect to see simulation error.

As a market scenario, we consider the case of yearly forward rates all equal to 5%, with displacements of 1.5% for each forward rate. Caplet volatilities are assumed to be flat and equal to 15%. For the cap, the first forward rate resets in one year’s time and the strike of each caplet is the same and equal to 7%. For the swaption, we set the correlation between each forward rate using the $L/\beta$ parametric form of (2.9) with $L = 0.5$ and $\beta = 0.1$. The strike of the swaption is also equal to 7%. We choose out-of-the-money strikes because the option’s value is less linear in volatility there and hence it is a more difficult test.

To simulate the pathwise Greeks we use a full-factor LMM with a randomised Sobol generator, see Section 2.5. Thirty two batches of $2^{18}$ paths ($2^{23}$ paths in total) are used to ensure that any errors are due to discretisation bias rather than simulation error. For the caps, we take one simulation step between each tenor date.

Table 3.2 shows the analytic delta and vega of each individual caplet calculated using the Black formula. Figure 3.1 shows the difference and two standard errors between the exact delta and the simulated pathwise deltas in the spot measure, with each of the ten deltas plotted on the x-axis. LE represents the log-Euler drift approximation, while PC represents the predictor-corrector drift approximation. The error for the last delta, for example, whose true value is 21.36% is 0.09% for log-Euler and 0.00% for predictor-corrector.
Table 3.2: Analytic Black-76 deltas and vegas for the ten caplets.

<table>
<thead>
<tr>
<th>forward rate</th>
<th>delta</th>
<th>vega</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_0$</td>
<td>1.85 %</td>
<td>0.54 %</td>
</tr>
<tr>
<td>$f_1$</td>
<td>8.59 %</td>
<td>1.62 %</td>
</tr>
<tr>
<td>$f_2$</td>
<td>13.08%</td>
<td>2.46 %</td>
</tr>
<tr>
<td>$f_3$</td>
<td>16.01%</td>
<td>3.08 %</td>
</tr>
<tr>
<td>$f_4$</td>
<td>17.96%</td>
<td>3.54 %</td>
</tr>
<tr>
<td>$f_5$</td>
<td>19.27%</td>
<td>3.89 %</td>
</tr>
<tr>
<td>$f_6$</td>
<td>20.16%</td>
<td>4.14 %</td>
</tr>
<tr>
<td>$f_7$</td>
<td>20.76%</td>
<td>4.33 %</td>
</tr>
<tr>
<td>$f_8$</td>
<td>21.14%</td>
<td>4.46 %</td>
</tr>
<tr>
<td>$f_9$</td>
<td>21.36%</td>
<td>4.54 %</td>
</tr>
</tbody>
</table>

Table 3.3: Relative error for deltas and vegas for the ten caplets in the spot measure using log-Euler (LE) and predictor-corrector (PC).

<table>
<thead>
<tr>
<th>forward rate</th>
<th>Delta</th>
<th>Vega</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LE</td>
<td>PC</td>
</tr>
<tr>
<td>$f_0$</td>
<td>0.61%</td>
<td>0.24%</td>
</tr>
<tr>
<td>$f_1$</td>
<td>0.46%</td>
<td>0.04%</td>
</tr>
<tr>
<td>$f_2$</td>
<td>0.44%</td>
<td>0.02%</td>
</tr>
<tr>
<td>$f_3$</td>
<td>0.48%</td>
<td>0.03%</td>
</tr>
<tr>
<td>$f_4$</td>
<td>0.56%</td>
<td>0.09%</td>
</tr>
<tr>
<td>$f_5$</td>
<td>0.54%</td>
<td>0.08%</td>
</tr>
<tr>
<td>$f_6$</td>
<td>0.50%</td>
<td>0.04%</td>
</tr>
<tr>
<td>$f_7$</td>
<td>0.49%</td>
<td>0.04%</td>
</tr>
<tr>
<td>$f_8$</td>
<td>0.51%</td>
<td>0.05%</td>
</tr>
<tr>
<td>$f_9$</td>
<td>0.42%</td>
<td>0.02%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LE</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_0$</td>
<td>2.06%</td>
<td>0.07%</td>
</tr>
<tr>
<td>$f_1$</td>
<td>1.42%</td>
<td>0.01%</td>
</tr>
<tr>
<td>$f_2$</td>
<td>1.31%</td>
<td>0.00%</td>
</tr>
<tr>
<td>$f_3$</td>
<td>1.30%</td>
<td>0.01%</td>
</tr>
<tr>
<td>$f_4$</td>
<td>1.32%</td>
<td>0.04%</td>
</tr>
<tr>
<td>$f_5$</td>
<td>1.28%</td>
<td>0.04%</td>
</tr>
<tr>
<td>$f_6$</td>
<td>1.23%</td>
<td>0.02%</td>
</tr>
<tr>
<td>$f_7$</td>
<td>1.17%</td>
<td>0.02%</td>
</tr>
<tr>
<td>$f_8$</td>
<td>1.11%</td>
<td>0.02%</td>
</tr>
<tr>
<td>$f_9$</td>
<td>0.98%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

From Figure 3.1 we notice that the predictor-corrector delta is very accurate for all caplets. Log-Euler, however, is not as accurate, with the error increasing for caplets with longer expiries. These errors are small when we consider the corresponding relative errors in Table 3.3, showing that the log-Euler delta estimates are sufficiently accurate. An estimate is considered acceptable if it has a relative error of less than 1%.

Figure 3.2 presents the error from the log-Euler and predictor-corrector vegas in the spot measure. Similarly with the deltas we see that predictor-corrector performs very well, producing almost no noticeable error. This is further highlighted when considering the relative error in Table 3.3. Log-Euler, however, produces a relative error greater than 1% for all but a solitary vega.

We now move on to consider the terminal measure. Figure 3.3 displays the difference between the true deltas and the simulated values. We see that log-Euler performs poorly for the first five caplet deltas, but then the error reduces for the
3.4. Predictor-corrector discretisation

Figure 3.1: Delta errors for out-of-the-money caplets in the spot measure using log-Euler (LE) and predictor-corrector (PC).

Figure 3.2: Vega errors for out-of-the-money caplets in the spot measure using log-Euler (LE) and predictor-corrector (PC).
last few deltas. Predictor-corrector, however, achieves accurate results for almost all estimates.

In terms of relative error, as shown in Table 3.4, we see that predictor-corrector is superior for all but two deltas. Even when predictor-corrector performs worse than log-Euler the relative error is still comfortably less than 1%.

![Delta error](image)

**Figure 3.3:** Delta errors for out-of-the-money caplets in the terminal measure using log-Euler (LE) and predictor-corrector (PC).

<table>
<thead>
<tr>
<th>forward rate</th>
<th>Delta LE</th>
<th>Delta PC</th>
<th>Vega LE</th>
<th>Vega PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_0$</td>
<td>-7.99 %</td>
<td>-1.18 %</td>
<td>-4.23 %</td>
<td>-0.47 %</td>
</tr>
<tr>
<td>$f_1$</td>
<td>-2.73 %</td>
<td>-0.39 %</td>
<td>-2.53 %</td>
<td>-0.22 %</td>
</tr>
<tr>
<td>$f_2$</td>
<td>-1.83 %</td>
<td>-0.29 %</td>
<td>-2.00 %</td>
<td>-0.13 %</td>
</tr>
<tr>
<td>$f_3$</td>
<td>-1.42 %</td>
<td>-0.24 %</td>
<td>-1.69 %</td>
<td>-0.02 %</td>
</tr>
<tr>
<td>$f_4$</td>
<td>-1.06 %</td>
<td>-0.10 %</td>
<td>-1.42 %</td>
<td>0.15 %</td>
</tr>
<tr>
<td>$f_5$</td>
<td>-0.73 %</td>
<td>-0.01 %</td>
<td>-1.18 %</td>
<td>0.32 %</td>
</tr>
<tr>
<td>$f_6$</td>
<td>-0.49 %</td>
<td>0.09 %</td>
<td>-0.96 %</td>
<td>0.50 %</td>
</tr>
<tr>
<td>$f_7$</td>
<td>-0.29 %</td>
<td>0.18 %</td>
<td>-0.76 %</td>
<td>0.70 %</td>
</tr>
<tr>
<td>$f_8$</td>
<td>-0.15 %</td>
<td>0.22 %</td>
<td>-0.67 %</td>
<td>0.80 %</td>
</tr>
<tr>
<td>$f_9$</td>
<td>-0.08 %</td>
<td>0.21 %</td>
<td>-0.73 %</td>
<td>0.77 %</td>
</tr>
</tbody>
</table>

**Table 3.4:** Relative error for deltas and vegas for the ten caplets in the terminal measure using log-Euler (LE) and predictor-corrector (PC).

Figure 3.4 plots the error from the vega estimates in the terminal measure. Given that part of the delta calculations are used to compute the vega, we would expect
the graph to look similar to Figure 3.3. If we consider the relative error, we see that in the same fashion as the spot measure, the log-Euler errors are mostly greater than 1% while for predictor-corrector they are much smaller. This is particularly true for the first five caplet vegas.

We now consider the lockout Bermudan swaptions. As an analytical formula does not exist under the LMM framework, we compute the true deltas and vegas using the log-Euler approximation with ten sub-steps between each tenor date. We compare this with log-Euler and predictor-corrector values obtained by taking one step between each tenor date. For the sake of brevity we only consider the spot measure.

To compute the exercise strategy we use a simple least-squares algorithm with $2^{14}$ first pass paths. Quadratic polynomials in the co-terminal swap rate and net present value of the swaption are employed as basis functions. We use the same exercise strategy (set of regression coefficients for the basis functions) to compute Greeks for the log-Euler, predictor-corrector and short stepped log-Euler simulations. This is to ensure that the differences between the Greeks are due to the discretisation schemes and not the exercise times. The prices of the lockout Bermudan swaptions are presented in Table 3.5.

<table>
<thead>
<tr>
<th>lockout period</th>
<th>short-stepped LE</th>
<th>LE</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 year</td>
<td>155.0</td>
<td>154.0</td>
<td>155.0</td>
</tr>
<tr>
<td>3 year</td>
<td>213.3</td>
<td>210.4</td>
<td>213.6</td>
</tr>
<tr>
<td>5 year</td>
<td>262.2</td>
<td>256.1</td>
<td>262.9</td>
</tr>
</tbody>
</table>

Table 3.5: Prices of the lockout Bermudan swaptions.

<table>
<thead>
<tr>
<th>forward rate</th>
<th>1 year</th>
<th>3 year</th>
<th>5 year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LE</td>
<td>PC</td>
<td>LE</td>
</tr>
<tr>
<td>$f_0$</td>
<td>-0.93 %</td>
<td>-0.16 %</td>
<td>4.08 %</td>
</tr>
<tr>
<td>$f_1$</td>
<td>2.41 %</td>
<td>0.05 %</td>
<td>2.93 %</td>
</tr>
<tr>
<td>$f_2$</td>
<td>1.47 %</td>
<td>0.15 %</td>
<td>2.27 %</td>
</tr>
<tr>
<td>$f_3$</td>
<td>1.03 %</td>
<td>0.16 %</td>
<td>1.75 %</td>
</tr>
<tr>
<td>$f_4$</td>
<td>0.76 %</td>
<td>0.00 %</td>
<td>1.47 %</td>
</tr>
<tr>
<td>$f_5$</td>
<td>0.63 %</td>
<td>0.04 %</td>
<td>1.21 %</td>
</tr>
<tr>
<td>$f_6$</td>
<td>0.53 %</td>
<td>0.03 %</td>
<td>0.98 %</td>
</tr>
<tr>
<td>$f_7$</td>
<td>0.37 %</td>
<td>0.01 %</td>
<td>0.78 %</td>
</tr>
<tr>
<td>$f_8$</td>
<td>0.30 %</td>
<td>0.02 %</td>
<td>0.67 %</td>
</tr>
<tr>
<td>$f_9$</td>
<td>0.25 %</td>
<td>0.06 %</td>
<td>0.56 %</td>
</tr>
</tbody>
</table>

Table 3.6: Relative error for the deltas of the different length lockout Bermudan swaptions in the spot measure using log-Euler (LE) and predictor-corrector (PC).

Table 3.6 shows the relative error of log-Euler and predictor-corrector deltas,
for three different length lockout periods. We notice that the relative error for log-Euler is always greater than predictor-corrector. Drawing our attention to the 5-year lockout Bermudan swaption we see that the relative error for log-Euler is practically greater than 1% for each delta, while for predictor-corrector it is significantly smaller.

<table>
<thead>
<tr>
<th>forward rate</th>
<th>1 year</th>
<th>3 year</th>
<th>5 year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LE</td>
<td>PC</td>
<td>LE</td>
</tr>
<tr>
<td>$f_0$</td>
<td>8.31%</td>
<td>-0.51%</td>
<td>4.39%</td>
</tr>
<tr>
<td>$f_1$</td>
<td>3.04%</td>
<td>-0.12%</td>
<td>4.50%</td>
</tr>
<tr>
<td>$f_2$</td>
<td>2.12%</td>
<td>-0.03%</td>
<td>3.65%</td>
</tr>
<tr>
<td>$f_3$</td>
<td>1.59%</td>
<td>-0.06%</td>
<td>3.02%</td>
</tr>
<tr>
<td>$f_4$</td>
<td>1.32%</td>
<td>-0.14%</td>
<td>2.63%</td>
</tr>
<tr>
<td>$f_5$</td>
<td>1.18%</td>
<td>-0.07%</td>
<td>2.34%</td>
</tr>
<tr>
<td>$f_6$</td>
<td>1.04%</td>
<td>-0.08%</td>
<td>2.07%</td>
</tr>
<tr>
<td>$f_7$</td>
<td>0.84%</td>
<td>-0.11%</td>
<td>1.79%</td>
</tr>
<tr>
<td>$f_8$</td>
<td>0.73%</td>
<td>-0.07%</td>
<td>1.61%</td>
</tr>
<tr>
<td>$f_9$</td>
<td>0.64%</td>
<td>-0.03%</td>
<td>1.43%</td>
</tr>
</tbody>
</table>

Table 3.7: Relative error for the vegas of the different length lockout Bermudan swaptions in the spot measure using log-Euler (LE) and predictor-corrector (PC).

Table 3.7 is similar to Table 3.6, however this time we are considering the vegas of the corresponding forward rates. We observe that log-Euler performs much worse than predictor-corrector. For a 5-year lockout period, the relative errors of the log-Euler vegas are between 3% and 6%, while for predictor-corrector they are always less than 1%. Examining a shorter lockout period of only 1 year we see that log-Euler produces significant errors and is much less accurate than predictor-corrector.

While these results are promising it is worthwhile considering the computational time for both methods. We have seen that predictor-corrector is the same order as the log-Euler scheme, but the leading coefficient will be different. Table 3.8 illustrates the time taken to compute the price, deltas and vegas for the 5-year lockout Bermudan swaption with $2^{14}$ paths using a five-factor LMM. A 3.16Ghz Intel Core 2 Duo PC with 4Gb of RAM was used, with single threaded C++ code.

<table>
<thead>
<tr>
<th>Method</th>
<th>Price</th>
<th>Deltas and Vegas</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE without sub-steps</td>
<td>0.28</td>
<td>0.13</td>
<td>0.41</td>
</tr>
<tr>
<td>PC without sub-steps</td>
<td>0.46</td>
<td>0.27</td>
<td>0.73</td>
</tr>
<tr>
<td>LE with 6-monthly steps</td>
<td>1.95</td>
<td>0.89</td>
<td>2.84</td>
</tr>
</tbody>
</table>

Table 3.8: Time taken (in seconds) to compute price and deltas and vegas for the 5-year lockout Bermudan swaption with $2^{14}$ paths.

The result in Table 3.8 that predictor-corrector takes approximately twice as long to compute deltas and vegas as log-Euler (0.27 seconds compared to 0.13 seconds) with the same number of steps is what we would expect. This is because the method
Figure 3.4: Vega errors for out-of-the-money caplets in the terminal measure using log-Euler (LE) and predictor-corrector (PC).

essentially performs two updates on the sensitivity vectors over each step (compare Jacobians from $H_{k,2}$ and $H_{k,0}$, with the log-Euler equations) that are similar to the update from one step of the log-Euler discretisation. While this seems much slower it is worth keeping in mind that computing $n$ deltas and $nD$ vegas takes less time than computing the price. Predictor-corrector is also much faster than the alternative of using log-Euler with 6-monthly time steps, which at 0.89 seconds took more than three times as long to compute the deltas and vegas.
Chapter 4

Markov-functional models

The aim of this chapter is to introduce the class of Markov-functional models and to demonstrate how to practically implement a model. Whilst there are many Markov-functional models in the literature, we focus on the separable LIBOR Markov-functional model introduced by Pietersz, Pelsser and Van Regenmortel, [70]. The reason for this is twofold. First, the model is similar to the LMM. Second, it is similar to other Markov-functional models, such as the model of Hunt, Kennedy and Pelsser, [45] as demonstrated for the one-factor case, by Bennett and Kennedy, [10].

This chapter is compiled in part from Pietersz, Pelsser and Van Regenmortel, [70], and Brace, [16]. However, we extend certain parts of the model and in particular present a spot measure implementation, which is not currently in the literature.

4.1 Why the LMM is not Markovian

The LMM is Markovian in the forward rates, but this is not useful from a numerical viewpoint, because the number of forward rates is often large. Unfortunately the LMM is not Markovian in a small number of factors because of two two reasons. The first being the state-dependent drift and the second being the way the time-dependence of the volatility affects each rate differently.

To simulate the forward rates to time $T_k$ we usually evolve to tenor dates $T_0, T_1, \ldots, T_k$, updating the drift at each date. Consider the equation for the drift (2.5), we see that a forward rate at time $T_k$ depends on a subset of the rates at time
48 Markov-functional models

$T_{k-1}$, which in turn depend on the rates at time $T_{k-2}$. This dependence continues all the way back to the initial forward rates. We can observe from this recursive relationship that knowing the initial rates is not enough to determine their value at a future time, we need to know the history of the process at each point where the drifts are recalculated.

The second issue with the LMM is the choice of volatility structures that are used in practice. Consider evolving the forward rates from time 0 to time $T_0$ and then to time $T_1$, ignoring drifts and correction terms we have from (2.2)

$$\log(f_i(T_1) + \alpha_i) = \log(f_i(0) + \alpha_i) + a_i(0)Z(1) + a_i(1)Z(2).$$

Assuming we are using a $D$-factor LMM, the forward rates at time $T_1$ will depend on $2D$ factors, $D$ from each $Z$ vector. This is because in general we cannot express the above equation using one random variable over the entire time interval, $[0, T_1)$, denoted by $Z(0,2)$,

$$\log(f_i(T_1) + \alpha_i) = \log(f_i(0) + \alpha_i) + a_i(0,2)Z(0,2), \quad (4.1)$$

unless the same time scaling is applied to all elements of the pseudo-square root matrices. Having the same time scaling applied to each forward rate volatility is equivalent to the instantaneous form

$$a_i(t) = \nu_i \sigma(t),$$

which is known as separable. If we consider the $abcd$ volatility (2.8), we cannot write this in the form above. If however, we make the restriction that $b = d = 0$ then

$$a_i(t) = ae^{c(T_i - t)} = ae^{T_i}e^{-ct}, \quad (4.2)$$

which has the time and forward rate dependence separated out. One can even prove that (4.2) is the only time-homogeneous separable volatility form. Separability is therefore a mixed blessing: we achieve low-dimensionality at the cost of time-homogeneity. This restricted volatility form can make the calibration procedure more complicated, but we can still fit the model to a range of market products, as we see in Section 4.4.

### 4.2 Markov-functional model

A general Markov-functional model consists of Markov factors, $X \in \mathbb{R}^F$, which are used to create a vector of state variables $\xi \in \mathbb{R}^n$. The functional form of the
model, which determines the relationship between $\xi$ and $X$, is typically chosen to fit market dynamics whilst retaining tractability. Once the functional form is chosen, the future values of the state variables are determined by the initial model inputs and the Markov factors. These state variables are then converted into the option price by the deflated payoff function $g$. A sketch of the model is

$$X \mapsto \xi \mapsto g.$$ (4.3)

Consider an option with expiry $T_n$. In a Markov-functional model, we use the Feynman-Kac formula to derive a PDE for its deflated value, which has the form

$$\frac{\partial g(t)}{\partial t} = Pg(t),$$ (4.4)

$$g_{T_n} = g(\xi, T_n),$$

where $P$ is the partial differential operator (which is determined by the model) and the terminal condition at expiry is given by the option’s payoff function. Since the model is Markovian the dimension of the PDE will be equal to $F$, which means that a practical implementation is possible when the number of factors is small.

An effective way to numerically solve the above PDE is to use the method of lines, [79], to convert it into a large system of ordinary differential equations (ODEs). We can then use the wealth of available literature to solve the system of ODEs. The well-known method of lines discretises the PDE in all dimensions except time. For example, assume we have the partial differential operator

$$P = \frac{\partial^2}{\partial x^2},$$

we construct a symmetric grid of possible $x$ values with (for ease of exposition) equal spacing between nodes of length $\Delta x$. We then substitute in a finite difference approximation for $g$. If central differencing is used we have

$$\frac{\partial g(\xi, t)}{\partial x_i} = \frac{g_{i-1}(\xi, t) - 2g_i(\xi, t) + g_{i+1}(\xi, t)}{\Delta x^2},$$

where $g_i(\xi, t)$ represents the deflated payoff function evaluated at the $i$th node. Let $m$ denote the number of nodes to the power $F$, then this generates an $m \times m$ matrix $A$ (tridiagonal in this example) and allows one to write the problem as a system of ODEs

$$\frac{du(t)}{dt} = Au(t) + b,$$ (4.5)

$$u(T_n) = g(\xi, T_n),$$
where \( b \in \mathbb{R}^m \) contains the boundary conditions that cannot fit into \( A \) and \( u \in \mathbb{R}^m \) is a vector of discretised \( g \) values. For ease of exposition we have assumed Dirichlet boundary conditions, so that \( b \) is not time dependent. This ODE is solved backwards in time creating an \( m \)-dimensional solution vector. The value of the option is given by the scalar function \( h \),

\[
h(u,0) = \begin{cases} 
u(0), & \text{at } \theta_0, \\ 0, & \text{otherwise}, \end{cases} \quad (4.6)
\]

which selects the price at the initial model parameters \( \theta_0 \) or origin (usually the middle point in the case of a symmetric grid).

For Bermudan type products an exercise strategy needs to be incorporated into the pricing phase. A common method is to initialise the PDE at one exercise date and solve back to the previous exercise date. This way the exercise strategy only enters through the terminal conditions. We construct the terminal condition at exercise time \( T_k \) by comparing the solution from the previous ODE (i.e. the continuation value) with the value upon immediate exercise. We define \( h \) at node \( j \) to be

\[
h_j(u, T_k) = \max \{ u_j(T_{k+}), g_j(T_k) \}, \quad (4.7)
\]

where \( u_j(T_{k+}) \) is the continuation value and \( g_j(T_k) \) is the deflated exercise value. This process is repeated backwards through the exercise dates until we reach time zero. We note that early exercise is considerably easier for a Markov-functional model than the LMM, compare equation (4.7) with Section 2.6.

### 4.3 Separable LIBOR Markov-functional model

We now present the model employed throughout the remainder of this thesis. We use an extended version of the \( F \)-factor separable LIBOR Markov-functional model, introduced in [70], with a more general volatility structure and also allow for skew in caplet volatilities. The model is based on a similar idea to the LMM of evolving discrete market observable forward rates, but we make a modification to the drift and restrict the volatility to separable structures.

The \( n \) state variables \( \xi \) in (4.3) are discrete forward rates \( f_0, \ldots, f_{n-1} \) with a corresponding tenor structure \( 0 < T_0 < T_1 < \ldots < T_n \), with \( \tau_i = T_{i+1} - T_i \), which is identical to the LMM. We assume that the forward rates have the following displaced log-normal dynamics

\[
\frac{d(f_i(t) + \alpha_i)}{f_i(t) + \alpha_i} = \mu_i(f, t)dt + \nu_i^T dX(t), \quad (4.8)
\]
where the $\alpha_i$’s are constant displacement coefficients and the forward rate specific volatility terms $\nu_i$ are $F$-dimensional deterministic vectors. The Markov factors $X(t)$ are defined by

$$dX(t) = C(t)dW(t),$$

with $X(0) = 0$ and $W(t)$ is a standard $F$-dimensional Brownian motion under the pricing measure. The matrix $C(t)$ scales time for all of the forward rates, allowing the evolution to be written as a single step over tenor dates, similar to (4.1). We assume that $C(t)$ is an $F \times F$ matrix that is constant across tenor dates $[T_i, T_{i+1})$. This is a more general form than the one used by Piterbarg, [73], which assumes $C(t)$ is upper triangular. Our separable volatility structure is therefore given by

$$\nu_i^T C(t).$$

Let $C(k)$ represent the matrix of constant values over $[T_{k-1}, T_k)$, with the convention that $T_{-1} = 0$. The log-forward rates are evolved across the step 0 to $T_k$ using

$$\log(f_i(T_k) + \alpha_i) = \log(f_i(0) + \alpha_i) + \mu_i(0) - \text{cov}_{ii}(0)/2 + \nu_i^T X(T_k),$$

where $\mu_i(0)$ is the integrated drift over the single evolution step $[0, T_k)$, $\text{cov}_{ii}(0)$ is the variance of the log-forward rate across the step and $X(T_k)$ is the integral of (4.9) over the period. This equation gives the functional form of the model. To ensure the Markovian condition holds we have made the modification that the forward rates are evolved to each time in a single step from time zero, which circumvents the drift issue discussed in Section 4.1. One possible problem with single stepping the forward rates is that the integral of the drift needs to be approximated over long intervals. This model therefore requires an accurate drift approximation or it will suffer from a large discretisation bias. We have found predictor-corrector to be sufficiently accurate.

The pricing PDE is given by the Feynman-Kac formula, with the partial differential operator

$$P = \sum_{i,j=1}^F q_{ij}(t) \partial_i \partial_j,$$

with $\partial_i = \frac{\partial}{\partial x_i}$, where $x_i$ represents the $i$th Markov factor. For the coefficients, $c_{ij}(t)$
is the element in the $i$th row of the $j$th column of $C(t)$ in (4.10) and

$$q_{ij}(t) = \frac{1}{2} \sum_{f=1}^{F} c_{if}(t)c_{jf}(t).$$

Since we are using piecewise constant matrices $C$, the PDE coefficients will be constant across tenor dates.

### 4.4 Calibration

A model’s popularity is a function of its calibration accuracy. Since the separable Markov-functional model has similarities to the LMM we can directly apply some of the calibration techniques, particularly the skew calibration of Section 2.4.3. For the volatility fit we need to be careful not to breach the separability condition. One technique that fits the model to $n$ caplets and $n$ coterminal swaptions is given in Chapter 7.5 of Brace, [16]. We give an outline of the technique for a one-factor model to demonstrate that separable volatility structures can be calibrated to the market. We start by defining

$$\Lambda_i := \int_0^{T_i} C(t) \, dt = \sum_{j=0}^{i-1} C(j)(T_{j-1} - T_j),$$

where the second equality is a consequence of the piecewise constant form of $C$. The variance of the caplets and coterminal swaptions in the model are given by

$$\text{var}_{C\text{plt}}^i T_i = \nu_i^2 \Lambda_i, \quad \text{(4.12)}$$

$$\text{var}_{\text{swpn}}^i T_i = \left( \sum_{k=j}^{n-1} U_k \nu_k \right)^2 \Lambda_i, \quad \text{(4.13)}$$

where $U_k$ is used to convert from caplet volatilities to swaption volatilities and is computed using the Hull and White approximation from Section 2.4.2. The aim is to fit these model quantities to their market traded equivalents. The routine starts with the last caplet/swaption (which coincide) and works backwards. We can easily fit the last caplet by setting

$$\Lambda_{n-1} = T_{n-1},$$

$$\nu_{n-1} = \sqrt{\text{var}_{C\text{plt}}^{n-1}}.$$
We now assume that we have fitted $\Lambda_k$ and $\nu_k$ for $k = j + 1, \ldots, n - 1$ and we demonstrate how to calibrate for step $j$. From (4.12) and (4.13), there are two equations and two unknowns. Divide (4.12) by (4.13), which removes $\Lambda_j$ and $T_j$ giving

$$\var_{C^\text{plt}}^j \left( U_j \nu_j + \sum_{l=j+1}^{n-1} U_l \nu_l \right)^2 = \nu_j^2 \var_{j}^\text{Swpn},$$

which is a quadratic in $\nu_j$. Taking the positive root produces the required term for $\nu_j$ and

$$\Lambda_j = \frac{\var_{C^\text{plt}}^j}{\nu_j^2} T_j.$$

Once we have solved for all of the $\nu_k$ and $\Lambda_k$ values, we back out the entries of $C(t)$ starting with

$$C(0) = \sqrt{\frac{\Lambda_0}{T_0 - 0}},$$

and then compute

$$C(k) = \sqrt{\frac{\Lambda_k - \Lambda_{k-1}}{\tau_k}},$$

for $k = 1, \ldots, n - 1$.

This calibration can fail at two points: if there is no single positive root to (4.14) and if the $\Lambda_k$ terms are not monotonically increasing. The first issue will arise if

$$\sqrt{\var_{j}^\text{Swpn}} < U_j \sqrt{\var_{C^\text{plt}}^i},$$

which is unlikely to occur with market data given that the caplet and swaption volatilities will be of a similar magnitude and the elements of $U_j$ are typically less than one.

The second issue will arise if swaption and caplet volatilities for the same expiry are largely different. This is unlikely with market data and will usually occur if a particular volatility is not liquidly traded and therefore not consistent with the other data.
To implement the Markov-functional model using a numerical PDE method we can design our program in a similar way to the LMM. There are, however, a few key differences. Instead of drawing random numbers to discretise the forward rates, we use a grid for the range of possible Markov factor values. We have that

$$X(T_k) = \int_0^{T_k} C(t)dW(t),$$

which has a multivariate normal distribution. We can therefore calculate the standard deviation of the Markov factors and construct a grid to cover a reasonable probability range. Brace, Chapter 10 of [16], suggests using a five to six standard deviation range for the grid. We have found five standard deviations to be sufficiently accurate, so we use it throughout the thesis.

Once the Markov factor grid has been constructed we are in a position to compute the forward rates for each node. Using these grid values the non-reset forward rates are computed from the functional form (4.11) with the predictor-corrector drift approximation. We note that we use the same grid throughout the life of the trade and when in the terminal measure, we use the same bond, $P(t, T_n)$ as numeraire.

We now construct the terminal condition, by calculating the deflated option price on each node with the discretised forward rates. At this point we use our numerical PDE method to solve back to the previous exercise date. The choice of numerical method is one of the most important decisions in ensuring we have an accurate price.

For the general volatility structure used with the separable LIBOR Markov-functional model we have mixed spatial derivatives of the form

$$\frac{\partial}{\partial x_i \partial x_j},$$

which makes solving the PDE more complicated, see Douglass and Gunn, [31]. Two methods that work well with mixed derivatives, and are used in this thesis, are the ADI technique introduced by Craig and Sneyd, [25], and the Krylov subspace method introduced by Niesen and Wright, [66].

In the case of a two-dimensional PDE the ADI method works by splitting the time-step procedure into two phases. Firstly, the PDE is solved implicitly for one Markov factor, while holding the other factor fixed, creating an intermediate step. Then the PDE is solved implicitly for the remaining factor, while holding the original factor fixed. Each individual operator produces a simple tridiagonal matrix, which is easily solved in a computer program. These tridiagonal systems are quick to solve, which makes the speed of the ADI method one of its biggest advantages. One issue
with the Craig and Sneyd ADI method is that it uses a constant step size to solve the PDE over a tenor period. This can be inefficient because it does not allow for larger steps over more stable regions.

The Krylov subspace method operates by directly approximating the solution of the ODE (4.5). It does this using Krylov subspace techniques to reduce the large matrix $A$ in (4.5), into a smaller, more manageable matrix of size $m$. Throughout this thesis we use $m = 30$, because it has been shown in general to be a reasonable size, see [66]. The solver is then fully adaptive and takes a variable time step based on accuracy criteria specified by the user and the stability of the problem. This leads to large step sizes over stable regions and smaller steps where the problem is less stable, making the solver both efficient and accurate. Niesen and Wright show that the Krylov subspace method is superior to the ADI technique for the Heston model [41]. From our experience, we also find the method to be superior for the separable Markov-functional model.

One difficulty for PDE pricing is path dependent options. The usual way to deal with such products is to use an auxiliary variable, which accounts for the possible values of the history of the process. This, however, will add an extra dimension to the problem possibly making the numerical method laborious. See Chapter 9.6 of [50] for a discussion of auxiliary variables for Asian options.

4.6 Separable LIBOR Markov-functional model in the spot measure

It is widely believed (see Remark 2 of Pietersz, Pelsser and Van Regenmortel, [70], Chapter 10.3 of Brace, [16] and Section 5.1 of Ng, [65]) that the separable LIBOR Markov-functional model needs to be implemented using the terminal measure. This is not ideal as we recall from Section 2.1 that the terminal measure LMM does not perform as well as the spot measure LMM.

So far we have not specified a measure, deferring the issue until now. The reason why the terminal measure is always used in the literature is because there are no problems long-stepping forward rates and the numeraire has the practical feature of being non path-dependent. The spot measure, unfortunately, does not naturally have these features. The two issues, therefore, with implementing a Markov-functional model in the spot measure are

- the forward rates need to be long-stepped across tenor dates, and
- the spot measure has a path dependent numeraire.

The astute insight made by Beveridge, [11], involves rewriting the expression for the
Markov-functional models

drift, which circumvents the first problem. In the spot measure the drift is expressed as

\[ \mu_i(t) = \sum_{j=\eta(t)}^{i} \frac{(f_j(t) + \alpha_j)\tau_j}{1 + \tau_j f_j(t)} \text{cov}_{ij}(t). \]

This can be rewritten without dependence on \( \eta(t) \),

\[ \mu_i(t) = \sum_{j=0}^{i} \frac{(f_j(t) + \alpha_j)\tau_j}{1 + \tau_j f_j(t)} \text{cov}_{ij}(t), \quad (4.15) \]

because the covariance of a reset forward rate is zero and so the terms do not contribute to the summation. This reformulation means we are now able to step across reset dates.

The second issue is the path dependency of the discretely-compounded money market numeraire. Recall that its value is given by

\[ P(t,T_{\eta(t)}) \prod_{j=0}^{\eta(t)-1} (1 + \tau_j f_j(T_j)), \]

which depends on all of the reset forward rates until time \( \eta(t) - 1 \) and is therefore path dependent. Since we are in a lattice framework, working backwards in time, we do not have this history. (We note that this is not a problem for the terminal measure, because the terminal zero-coupon bond (2.4) is completely specified by forward rates resetting at and after the current time.) One option to handle the path dependence is to construct a range of possible numeraire values by using an auxiliary variable. Cash flows are then deflated for each of these values and the PDE is solved. The problem with this is that it adds an extra dimension to the lattice and hence slows down the numerical implementation.

To avoid using an auxiliary variable we suggest discounting cash flows back one tenor date at a time, instead of converting into numeraire amounts. To demonstrate the idea, assume we have a cash flow at a particular grid point at time \( T_{k+1} \). We now need to get the expected cash flow valued at the previous reset date \( T_k \). First we solve the PDE across the time step, which gives the expected prices, but in time \( T_{k+1} \) amounts. These values need to be discounted back one period. Recalling that the forward rate \( f_k \) is for the interval \([T_k, T_{k+1})\) we can discount an expected cash flow back one step by multiplying by

\[ \frac{1}{1 + \tau_k f_k(T_k)}. \]
4.7 Comparison between spot and terminal measures

We do this for cash flows on all grid points, obtaining values in time $T_k$ amounts. The process is then repeated backwards until we have reached our starting point. By using this method we do not need an auxiliary variable and therefore do not suffer from the extra computational burden.

4.7 Comparison between spot and terminal measures

We aim to answer the practical question, which probability measure produces the most accurate price? To do so we compare the spot and terminal measures by pricing interest rate caps and double-digital options. We choose caps because they are instruments commonly traded in the market and double digitals because they are a good test as to how well our model fits the Black formula. The undiscounted price for a double-digital option that pays one when the forward rate sets below the lower strike $K_1$ or above the higher strike $K_2$ is

$$N(-d_2(K_1)) + N(d_2(K_2)),$$

where $N$ denotes the cumulative normal distribution function and $d_2$ is given by the Black formula. Pricing this product correctly is therefore a test of how well the model fits the market assumed forward rate distribution.

Since we can price both products analytically we are able to compare our numerical prices with an exact price. We use semi-annual forward rates, flat at 5%, with the first caplet and double digital expiring in 6-months time. Displacements are set to zero for each forward rate. A two-factor model is employed with the volatility structure

$$\nu_T^i = \begin{bmatrix} 0.10 & 0.08 \end{bmatrix},$$

where the forward rate specific volatility terms are the same for all rates and the time-dependent matrices are all equal to

$$C(k) = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

This equates to an annual implied Black volatility of approximately 21%. This simple structure was chosen to facilitate the reproduction of results.

Table 4.1 presents the difference between the cap prices computed using the Black formula and the Markov-functional model using the Krylov subspace method (with a spatial discretisation of 101 nodes in both dimensions). We compare both 10- and 20-rate caps using the PC and IPC drift approximations. We remark that
the terminal measure performs better than the spot measure for all of the caps irrespective of the drift approximation used.

<table>
<thead>
<tr>
<th></th>
<th>ITM</th>
<th>ATM</th>
<th>OTM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Terminal</td>
<td>Spot</td>
<td>Terminal</td>
</tr>
<tr>
<td>10 Rate</td>
<td>PC 0.04</td>
<td>0.85</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>IPC 0.04</td>
<td>0.86</td>
<td>0.17</td>
</tr>
<tr>
<td>20 Rate</td>
<td>PC 0.65</td>
<td>8.64</td>
<td>1.41</td>
</tr>
<tr>
<td></td>
<td>IPC 0.22</td>
<td>9.61</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of absolute price errors (in basis points) for ITM (strike 3%), ATM (strike 5%) and OTM (strike 7%) caps using PC and IPC drift approximations. Prices were computed using a PDE method.

To ensure that the errors in Table 4.1 are not only due to the numerical PDE method, we implemented the Markov-functional model using Monte Carlo. To compute the price we used a randomised Sobol quasi-random number generator with 16 batches of $2^{16}$ paths and Brownian bridging. The results are shown in Table 4.2. We do not present standard errors, because the prices have sufficiently converged, see Figure 4.2. We notice two contradictions with the previous table. The first is that the spot measure generally performs better than the terminal measure (this is consistent with the results from Beveridge, [11], for the LMM) and the second is that the IPC drift approximation works much better than the non iterative version (consistent with Joshi and Stacey, [53], for the LMM).

<table>
<thead>
<tr>
<th></th>
<th>ITM</th>
<th>ATM</th>
<th>OTM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Terminal</td>
<td>Spot</td>
<td>Terminal</td>
</tr>
<tr>
<td>10 Rate</td>
<td>PC 1.00</td>
<td>1.37</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>IPC 0.01</td>
<td>0.00</td>
<td>0.05</td>
</tr>
<tr>
<td>20 Rate</td>
<td>PC 18.92</td>
<td>14.25</td>
<td>21.58</td>
</tr>
<tr>
<td></td>
<td>IPC 0.67</td>
<td>0.01</td>
<td>1.13</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of absolute price errors (in basis points) for ITM (strike 3%), ATM (strike 5%) and OTM (strike 7%) Caps using PC and IPC drift approximations. Prices were computed using randomised quasi-Monte Carlo simulation.

One problem with studying interest rate caps is that we can have error cancellation by under pricing some caplets and over pricing others. We therefore focus on the caplets underlying one particular cap, the 20-rate OTM cap, for more conclusive results. This product is chosen because both the PDE and Monte Carlo prices have relatively large errors.

Figures 4.1 and 4.2 show the difference between the true prices and the numerical PDE and Monte Carlo prices respectively, for the 20 underlying caplets using the PC drift approximation. The actual prices range from 0.15 bps to 21.5 bps. Figure 4.2
also includes a two standard error range, which, because the prices have sufficiently converged, is barely visible on the graph. We see that the spot measure produces similar errors in the PDE and Monte Carlo implementations where it misprices the later caplets. The cause of this mispricing is due to the drift approximation. Consider the drift summation for the last forward rate, $f_{n-1}$; it contains every forward rate (see equation (4.15)). This drift term will therefore be relatively large and hence have a larger error than the drift of an earlier forward rate, which is composed of less terms.

We observe that the terminal measure performs well for the earlier and later caplets. This is because the absolute value of the drift terms of the corresponding forward rates are small. The drifts of the earlier forward rates are slight because the covariance term is small (due to the evolution step being short). The drifts of the later forward rates are modest because the summations contain few terms. We notice that the middle caplets have the largest errors, however the PDE method produces more accurate prices than the Monte Carlo method. This is because solving the PDE equation smooths cash flows across nodes, which reduces the impact of the boundary conditions (which is true in general for financial PDEs, see Chapter 4 of Tavella and Randall, [84]). Since the largest cash flows occur on and near the boundary, reducing their significance will improve the stability of the result. This avoids the issue of large cash flows increasing the variance of our results, which can happen within a Monte Carlo simulation.

**Figure 4.1:** Markov-functional model pricing errors (in basis points) for out-of-the-money caplets using a PDE method.
We now examine the pricing of double-digital options with strikes of 3% and 7%, where the true prices range from 42.2 bps for the first option to 1557 bps for the last option. Figures 4.3 and 4.4 mirror a similar result to the caplets in Figures 4.1 and 4.2. The larger absolute errors are caused by the product being more difficult to price, which also amplifies the noise in the numerical solution of the PDE. Once again we include a two standard error range for the Monte Carlo prices. Analogously to the caplets we notice that the terminal measure performs better for the PDE model while the spot measure performs better for the Monte Carlo implementation.

To quantify the performance of the two measures we compute the error measure,

\[ \sqrt{\sum_{i=1}^{20} (g_i - \hat{g}_i)^2}, \]

where \( g_i \) and \( \hat{g}_i \) are the Black and Markov-functional model prices of the \( i \)th products respectively. The errors are shown in Table 4.3.

Based on these results we recommend using the terminal measure for a PDE implementation, because it produces more accurate prices for the caps and double digitals. For a Monte Carlo implementation we find that the spot measure performs better, which is consistent with results for the LMM.
Figure 4.3: Markov-functional model pricing errors (in basis points) for double-digital options using a PDE method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Caplets</th>
<th>Double digitals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Terminal</td>
<td>Spot</td>
</tr>
<tr>
<td>PDE</td>
<td>0.4</td>
<td>3.1</td>
</tr>
<tr>
<td>Monte Carlo</td>
<td>4.8</td>
<td>5.3</td>
</tr>
</tbody>
</table>

Table 4.3: Comparison of the errors (in basis points) for the caplets and double-digital options, computed using equation (4.16).
Figure 4.4: Markov-functional model pricing errors and 2 standard errors (in basis points) for double-digital options using a quasi-Monte Carlo method.
Chapter 5

Control variate

The aim of this chapter is to effectively apply the separable Markov-functional model as a control variate for the displaced-diffusion LMM. The intention is to reduce the variance of an LMM simulation, which means that fewer paths can be run for equivalent accuracy, producing faster pricing.

The method we present works by computing prices and Greeks using two different implementations of the Markov-functional model: a lattice-based PDE; and a Monte Carlo which is highly correlated with the LMM. The PDE value is regarded as being precise and the difference is used to correct the LMM value. Of course, there are other ways to construct control variates, one could for example use simple contracts that can be analytically priced. The virtue of our approach is that it is product independent.

Piterbarg, [73], has previously suggested using a Markov-functional model as a control variate, but his method only works for prices and deltas, not vegas. Schlögl, [80], has also suggested using a Markov-functional model as control variate. Our objective here is to introduce a new methodology that is practical for all three. Whilst we introduce many innovations the essential improvement that allows for vega computations is that our calibration of the Markov-functional model varies smoothly with the calibration of the LMM.

This chapter is an extended version of the paper [29].
5.1 Markov-functional model as a control variate

To use the Markov-functional model as a control variate we value a product $V$ using

$$V_{control} = V_{LMM} - \zeta(V_{MC} - V_{PDE}),$$

where $V_{MC}$ and $V_{PDE}$ represent the deflated price in the Markov-functional model implemented via a Monte Carlo and PDE method respectively, and $\zeta$ is a constant value. Using different implementation methods for the Markov-functional model should still arrive at the same price, so the expected value is equal to that from the LMM

$$E[V_{control}] = E[V_{LMM}].$$

To see why the control variate works, consider the variance of the control price, given by

$$\text{Var}[V_{control}] = \text{Var}[V_{LMM}] + \zeta^2 \text{Var}[V_{MC}] - 2\zeta \text{Cov}[V_{LMM}, V_{MC}]. \quad (5.1)$$

This will have lower variance than the LMM price if

$$\zeta^2 \text{Var}[V_{MC}] < 2\zeta \text{Cov}[V_{LMM}, V_{MC}].$$

The variance reduction therefore depends on the correlation between the two model prices. The only differences between the LMM and the Markov-functional model, apart from the number of driving factors, is the volatility and drift restriction. As long as we can closely match the volatility structures and use a suitable drift approximation, the two models should be highly correlated resulting in considerable variance reduction.

The value of $\zeta$ which minimises the variance of (5.1) is

$$\zeta = \frac{\text{Cov}[V_{LMM}, V_{MC}]}{\text{Var}[V_{MC}]}.$$

To estimate $\zeta$ we could run a small number of simulations, calculate it using the above formula with the sample covariance and variance. To price we would then run a set of independent simulations using the estimated value of $\zeta$. Our goal, however, is to achieve a more accurate price faster, so instead of spending time performing preliminary estimation we take the simple approach of setting $\zeta = 1$. This is justified by our estimates of $\zeta$ for all examples considered in this chapter, see Section 5.5. For more details on control variates in general see Section 4.1 of Glasserman, [35].
5.1.1 Monte Carlo implementation

It is imperative that we carefully implement our Monte Carlo model. We want it to be as close as possible to that of the LMM to increase the variance reduction. This means that the same random numbers must be used for the first two factors when evolving rates.

To price a Bermudan product an exercise strategy is needed and here there are a few choices. We could:

- calculate the strategy for the Markov-functional Monte Carlo implementation
- use the exercise strategy derived from the lattice method
- use the strategy derived from the LMM, or
- use the same set of exercise times as the LMM.

There is a subtle difference between the last two choices. Applying the exercise strategy, in the form of a set of functions to compute the continuation value, may result in a slightly different exercise time between the two models. The variance reduction will be maximised when a product is exercised at the same time in both models as the cash flow amounts will be similar. Therefore, we recommend applying the LMM exercise times to the Markov-functional model.

5.1.2 Lattice implementation

Time spent solving the PDE directly reduces the time savings of the control variate method. We therefore need an efficient scheme and the Krylov subspace method meets this requirement.

Similarly with the Markov-functional Monte Carlo implementation we have a choice as to which exercise strategy to use. To ensure the price is not biased we need to have

\[ E(V_{MC}) = E(V_{PDE}), \]

which will hold if the same exercise strategy is used for both methods. However, applying the same exercise time from the LMM to the lattice method is nonsensical. As an alternative, we engage the exercise strategy from the LMM. This slight difference will result in a bias, however, our results show this bias to be very small, see Table 5.5.
5.2 Analytic calibration

The main innovation of this chapter lies in how we fit the Markov-functional model’s volatility structure to that of the LMM. We make both the evolution of the LMM and that of the fitted control variate vary smoothly with the calibration. For the LMM, a calibration is a covariance matrix for each step and the evolution is determined by a corresponding choice of pseudo-square root. We use the spectral decomposition discussed in Section 2.4.1. That is, if the covariance matrix, for step \( i \) has eigenvalues \( \lambda_f \) in decreasing order, with corresponding eigenvectors \( e_f \), then we take

\[
a_f = \sqrt{\lambda_f}e_f,
\]

to be column \( f \) of our pseudo-square root, \( A(i) \). As long as the eigenvalues are distinct, both they and the eigenvectors are analytic (and therefore smooth) functions of the matrix, see Andrew, Chu and Lancaster, [7]. This approach also maximises the impact of the first \( F \) factors.

For an \( F \)-dimensional Markov-functional model and a product that depends on \( n \) forward rates, there are \( nF + nF^2 \) volatility parameters to be fitted: the \( nF \) forward rate specific terms and \( nF^2 \) piecewise-constant time-dependent values. We will denote the pseudo-square roots arising from our separable fit as \( B(i) \).

One possible methodology, as in Piterbarg, [73], is to use a global least-squares numerical optimisation, however, there is no guarantee in that case that the matrix \( B(i) \) varies smoothly with \( A(i) \). Here we present a calibration method that is both analytic and stable. It has the features of using spectral methods to fit one matrix and inductively fitting the other matrices one by one.

For the remainder of this section we shall address the problem of fitting an \( F \)-dimensional Markov-functional model to a \( D \)-dimensional LMM, where \( F \leq D \). Naturally, we will only attempt to fit the first \( F \) columns of the matrices \( A(i) \). However, because we are using the pseudo-square root defined by spectral decomposition means that these will be the dominant factors. In fact, Lord and Pelsser, [60], found that up to 99% of the total yield curve variability can be accounted for in the first three factors.

Consider the evolution of the forward rates over \([0, T_0]\), with the corresponding pseudo-square roots \( A(0) \) and \( B(0) \). By looking at the stochastic differential equations for the forward rates in the LMM (2.1) and Markov-functional model (4.8) we see that if we have

\[
dX(t) = dW(t),
\]

by an appropriate choice of \( C \) values, then we can exactly calibrate the first \( F \)-
columns of $B(0)$ to $A(0)$, simply by setting

$$\nu_j^T = \begin{bmatrix} a_{j,1}(0) & \ldots & a_{j,F}(0) \end{bmatrix},$$

for $j = 0, \ldots, n - 1$ and with $C(0)$ equal to the identity matrix. Let $V$ denote the matrix with columns $\nu_j$ (which is equal to $B(0)$.)

Now that we have set all of the forward rate specific volatility terms by exactly fitting to the first $F$ columns of the first pseudo-square root, we need to calibrate the remaining $n - 1$ pseudo-square roots. Our freedom lies in the ability to pick the matrices $C(i)$.

For each $i$, we want to choose $C(i)$ to minimise the sum-of-squares norm of the difference

$$VC(i) - A(i),$$

including only the first $F$ columns of $A(i)$. Since the $k$th column of $C(i)$ only contributes to the $k$th column of $VC(i)$, we have a separate least-squares optimisation problem for each $k$ and $i$. If we let $a$ denote the target column and let $c$ denote the relevant column of $C(i)$ we have to minimise

$$Vc - a,$$

where $V$ is an $n \times F$ matrix, $c$ is an $F$-vector and $a$ is an $n$-vector. This can be solved using standard linear algebra techniques for solving an over-determined system. Since it is low-dimensional linear algebra, the calibration is performed effectively instantaneously within a computer program.

We could also perform the calibration starting with any evolution interval. Consider the evolution over $[T_{i-1}, T_i)$. We exactly fit the first $F$-columns of $A(i)$ by setting

$$\nu_j^T = \begin{bmatrix} a_{j,1}(i) & \ldots & a_{j,F}(i) \end{bmatrix},$$

with $C(i)$ equal to the identity matrix (note that some elements of $\nu_j$ will temporarily be set to zero as elements of $A(i)$ are zero due to reset forward rates). We perform a least-squares fit to the $n - i - 1$ pseudo-square roots after the evolution time $T_i$.

To fit the pseudo-square roots before $T_{i-1}$ we work backwards considering one evolution interval at a time. We compute the matrix $C(i - 1)$ by solving the system of equations, as above, including $\nu_j$ for $j = i, \ldots, n - 1$. Once $C(i - 1)$ has been computed we set $\nu_{i-1}$ so that the first $F$ columns of the $i - 1$th row of $A(i - 1)$ are exactly matched. We repeat, working backwards until all pseudo-square roots have been calibrated. This approach gives the flexibility of exactly matching the
first $F$-columns of a particular pseudo-square root.

The choice of pseudo-square root exactly fitted does affect the accuracy of the overall calibration. However, the calibration method is very fast so it is possible to iterate over all choices and select the best within a fraction of a second.

5.2.1 Calibration results

As an example of how accurate the calibration is, we fit the Markov-functional model to various volatility structures within the LMM. To gauge the accuracy of the calibration we use the statistic

$$\frac{\sqrt{\sum_{i,j,k}(a_{j,k}(i))^2} - \sqrt{\sum_{i,j,k}(a_{j,k}(i) - b_{j,k}(i))^2}}{\sqrt{\sum_{i,j,k}(a_{j,k}(i))^2}},$$

(5.2)

which is a measure of the amount of volatility the Markov-functional model captures. We consider calibrating to pseudo-square roots generated by taking the spectral decomposition of covariance matrices calculated using the $abcd$ volatility form of equation (2.8), with the correlation between each forward rate given by the $L\beta$ form (2.10).

Tables 5.1 - 5.3 show the calibration accuracy for two-, three- and five-factor LMMs fitted to a two-factor Markov-functional model. We fit the pseudo-square roots for different tenor products, all with semi-annual rates. For example, a product with a tenor of 5Y requires 10 pseudo-square roots. A variety of volatility scenarios for the LMM were considered, ranging from a very simple flat structure to more complicated forms. The scenarios are defined as $abcd\beta 1$ ($a = 0, b = 0, c = 0, d = 0.2, L = 0.5, \beta = 0.1$) giving flat volatility, $abcd\beta 2$ ($a = 0.05, b = 0.09, c = 0.44, d = 0.12, L = 0.5, \beta = 0.1$) representing a long lasting hump in the caplet volatilities, $abcd\beta 3$ ($a = -0.02, b = 0.3, c = 2, d = 0.15, L = 0.5, \beta = 0.1$) producing a hump peaking at about 1-year, and $abcd\beta 4$ ($a = -0.02, b = 0.3, c = 2, d = 0.15, L = 0.5, \beta = 0.2$) exhibiting a low correlation scenario.

The lower values in Tables 5.1 - 5.3 were obtained by choosing the least optimal pseudo-square root to exactly fit, while the upper values were obtained by calibrating to the optimal pseudo-square root. By optimal we mean maximising the statistic in equation (5.2). Table 5.4 shows which pseudo-square roots correspond to the best and worst fit in Table 5.1, where for example the top right entry 7-0 means that the worst and best calibration came from matching the pseudo-square root over the time step $[T_6, T_7]$ and $[0, T_0]$ respectively. We do not show the optimal pseudo-square root to fit for the three- and five-factor LMMs as the results were similar to the two-factor case.
5.2. Analytic calibration

<table>
<thead>
<tr>
<th>tenor</th>
<th>abcdβ1</th>
<th>abcdβ2</th>
<th>abcdβ3</th>
<th>abcdβ4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y</td>
<td>100-100 %</td>
<td>98-99 %</td>
<td>92-95 %</td>
<td>92-95 %</td>
</tr>
<tr>
<td>7Y</td>
<td>100-100 %</td>
<td>96-98 %</td>
<td>91-96 %</td>
<td>91-96 %</td>
</tr>
<tr>
<td>10Y</td>
<td>100-100 %</td>
<td>93-96 %</td>
<td>90-96 %</td>
<td>90-96 %</td>
</tr>
<tr>
<td>20Y</td>
<td>100-100 %</td>
<td>85-92 %</td>
<td>83-95 %</td>
<td>84-95 %</td>
</tr>
</tbody>
</table>

Table 5.1: Calibration accuracy range for a variety of volatility scenarios using the statistic (5.2) with a two-factor LMM.

Firstly we observe that the calibration works very well for the two-factor model, fitting the LMM pseudo-square roots very closely. Not surprisingly, two-factor flat volatility structures are perfectly matched (because a flat volatility structure is separable). Somewhat surprising is that the more complicated forms can still be calibrated very well, e.g. 82% of the volatility for a 20Y three-factor model with structure abcdβ4 can be captured.

<table>
<thead>
<tr>
<th>tenor</th>
<th>abcdβ1</th>
<th>abcdβ2</th>
<th>abcdβ3</th>
<th>abcdβ4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y</td>
<td>92-92 %</td>
<td>91-92 %</td>
<td>89-91 %</td>
<td>86-88 %</td>
</tr>
<tr>
<td>7Y</td>
<td>91-91 %</td>
<td>89-91 %</td>
<td>87-90 %</td>
<td>85-87 %</td>
</tr>
<tr>
<td>10Y</td>
<td>89-90 %</td>
<td>86-89 %</td>
<td>86-89 %</td>
<td>83-85 %</td>
</tr>
<tr>
<td>20Y</td>
<td>85-86 %</td>
<td>79-85 %</td>
<td>83-86 %</td>
<td>79-82 %</td>
</tr>
</tbody>
</table>

Table 5.2: Calibration accuracy range for a variety of volatility scenarios using the statistic (5.2) with a three-factor LMM.

As the number of factors in the LMM increases from two to three to five, the fit decreases. This is because a two-factor Markov-functional model cannot capture the components above the second column of the LMM pseudo-square root. Overall, however, the calibration accuracy is greater than 80% for all but a few cases, indicating that the Markov-functional model will be effective as a control variate.

Tables 5.1 - 5.3 highlight that the choice of which pseudo-square root to exactly match can make a difference in the accuracy of the calibration. From Table 5.4 we see that it is generally best to fit one of the earlier pseudo-square roots and the worst choice is to exactly fit a pseudo-square root approximately half way to maturity. In any case, the calibration method is analytic so it is possible to loop over all choices...
within a fraction of a second, making it easy to find the best fit.

### 5.3 Pricing

In this section we present prices and their standard errors using the control variate technique, before moving onto the powerful application of model vegas in the following section. We consider two products, Bermudan swaptions and cancellable inverse floater swaps. The Bermudan swaptions are struck at-the-money with the holder owning the right to enter into the swap. The inverse floater swaps have a strike of 18% and can be cancelled by the issuer at which point all future payments cease.

All forward rates are assumed to be semi-annual and flat, calculated using a continuously-compounded rate of 5%, i.e. $P(t,T) = e^{-0.05(T-t)}$. We use the volatility structure described above as $abcd/32$ as this represents what we consider a fairly difficult market scenario. All displacements are equal to zero, i.e. $\alpha_i = 0 \forall i$.

Both products start in 6-months time and can be called/cancelled on any of the reset dates. To calculate the exercise strategy we use the least-squares regression plus Andersen algorithm. The basis functions for the Bermudan swaptions are a constant and the quadratic polynomials of the intrinsic swaption value and the swap rate. For the cancellable inverse floater we use a constant, and quadratic polynomials in the current forward rate, the zero-coupon bond with the longest expiry and the coterminal swap rate. A Sobol quasi-random number generator with $2^{14}$ paths and Brownian bridging is used to determine the exercise strategy and then 10000 Mersenne Twister paths are used to determine the price. In the results below, we exactly fit to the pseudo-square root which provides the best global fit.

Table 5.5 shows prices, standard errors and the multiple of standard error reduction using the control variate for the Bermudan swaptions in the terminal measure. The method works very well, for a 5Y Bermudan swaption a standard error reduction of 22 to 248 times is achieved, which is an order of 400 to 60000 times speed up.

The speed up will be smaller than this in practice due to some time spent calculating the Markov-functional PDE and Monte Carlo prices, however, with an efficient

<table>
<thead>
<tr>
<th>tenor</th>
<th>$abcd/31$</th>
<th>$abcd/32$</th>
<th>$abcd/33$</th>
<th>$abcd/34$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y</td>
<td>-</td>
<td>8-1</td>
<td>7-0</td>
<td>7-0</td>
</tr>
<tr>
<td>7Y</td>
<td>-</td>
<td>11-1</td>
<td>11-0</td>
<td>12-0</td>
</tr>
<tr>
<td>10Y</td>
<td>-</td>
<td>15-2</td>
<td>17-1</td>
<td>18-1</td>
</tr>
<tr>
<td>20Y</td>
<td>-</td>
<td>30-2</td>
<td>23-4</td>
<td>23-4</td>
</tr>
</tbody>
</table>

Table 5.4: Pseudo-square roots exactly fitted corresponding to the two-factor calibration in Table 5.1
### Table 5.5: Prices and standard errors in basis points for different tenor Bermudan swaptions. 2-, 3- and 5-factor LMMs were used with the control variate.

<table>
<thead>
<tr>
<th></th>
<th>2 factor</th>
<th>3 factor</th>
<th>5 factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y LMM</td>
<td>221 (2.67)</td>
<td>220 (2.62)</td>
<td>219 (2.53)</td>
</tr>
<tr>
<td>Control</td>
<td>223 (0.01)</td>
<td>218 (0.09)</td>
<td>218 (0.12)</td>
</tr>
<tr>
<td>7Y LMM</td>
<td>321 (3.91)</td>
<td>325 (3.73)</td>
<td>321 (3.79)</td>
</tr>
<tr>
<td>Control</td>
<td>323 (0.03)</td>
<td>325 (0.17)</td>
<td>319 (0.21)</td>
</tr>
<tr>
<td>10Y LMM</td>
<td>479 (5.57)</td>
<td>468 (5.65)</td>
<td>468 (5.41)</td>
</tr>
<tr>
<td>Control</td>
<td>478 (0.07)</td>
<td>466 (0.33)</td>
<td>466 (0.40)</td>
</tr>
<tr>
<td>20Y LMM</td>
<td>830 (9.29)</td>
<td>803 (9.30)</td>
<td>812 (9.11)</td>
</tr>
<tr>
<td>Control</td>
<td>831 (0.32)</td>
<td>811 (0.97)</td>
<td>813 (1.15)</td>
</tr>
</tbody>
</table>

5.3. Pricing

A numerical scheme to calculate the PDE price the time savings will still be very significant. We did not focus on finding an efficient scheme to solve the PDE equation so we do not present timings here.

As we would expect the control variate works best for shorter dated products and when coupled with a low-factor LMM. This is because the LMM and Markov-functional models are more alike for small time frames and a similar number of factors, due to the volatility structures matching well (we can exactly fit one pseudo-square root, however, the further away from that pseudo-square root the lower the calibration accuracy). This is not to say that the control variate performs poorly for high factor longer dated products, we achieved a standard error reduction of 8 times for a five-factor 20Y Bermudan swaption, which is an order of 64 times speed up.

We note that there are slight discrepancies between the LMM and control prices in some examples, but these are within one standard error and are not considered significant.

The prices presented in Table 5.5 are computed using the Terminal measure. Since the terminal measure typically has a higher variance, the standard error reductions might not be significant if the spot measure is used. To address this issue Table 5.6 compares the standard errors of the Bermudan swaptions using both the terminal and spot measures.

We notice that the standard error for the LMM simulation is smaller in the spot measure than the terminal measure. Observe the 20Y five-factor product and we have values of 9.1 bps and 6.0 bps. This causes a smaller standard error reduction when compared with the terminal measure of 8 times as opposed to 6 times. What really matters, however, is the actual value of the error and here we see that the spot measure wins out with 0.9 bps in contrast to 1.2 bps. This result reinforces the effectiveness of the Markov-functional model as a control variate, irrespective of which measure is used.
Table 5.6: Standard errors in basis points for different tenor Bermudan swaptions using both terminal and spot measures. 2-, 3- and 5-factor LMMs were used with the control variate.

Table 5.7 displays the results for the inverse floater. We notice that the numbers are similar to those for the Bermudan swaption; significant standard error reductions are achieved. For a 20Y product priced with a five-factor LMM we achieve a standard error of 10 times less by using the control variate. This is encouraging as an inverse floater is an exotic instrument, which indicates that our method will be useful for an entire suite of products.

Table 5.7: Standard errors in basis points for different tenor cancellable inverse floater swaps using both terminal and spot measures. 2-, 3- and 5-factor LMMs were used with the control variate.
5.4 Vega

Pricing, while time consuming, is typically not as computationally intensive as Greek computations. If we are using a finite difference approach, then for each sensitivity we will need to run an extra set of simulations with the perturbed parameters. Given that the number of sensitivities required can be large, we need an efficient way of computing prices, otherwise running many pricing paths may simply be too time consuming to be practical. We do not use the more sophisticated pathwise adjoint approach in Chapter 3, because finite difference more clearly demonstrates the control variate idea.

To calculate the model vega we find the price using the control variate method, then perturb the LMM volatility (e.g. $d$ in the case of the $abcd$ volatility structure used here), recalibrate our Markov-functional model, which is performed quickly, and then compute the new price. The calibration methodology of fitting to the spectral pseudo-square roots will be stable for vega computations because small perturbations of the covariance matrix will cause small changes in the eigenvalues and eigenvectors (see [7]).

When calculating the model vega it is imperative that we use the same random numbers in both simulations. By doing this the standard errors for the unperturbed and perturbed simulations will be highly correlated and hence produce a smaller overall standard error. We also use the same set of exercise times in both perturbed and unperturbed simulations to reduce noise, see Section 3.3.

Denoting the perturbation by $\epsilon$ and the $i$th unperturbed and perturbed prices as $V_i(\theta)$ and $V_i(\theta + \epsilon)$ respectively, then the statistic we are interested in is

$$\text{model vega} = \frac{1}{N} \sum_{i=1}^{N} (V_i(\theta + \epsilon) - V_i(\theta)),$$

and its standard error, for $N$ simulations. We could use a central difference above, but instead we choose a one-sided difference as traders are often interested in what happens to their portfolio when volatilities increase by 1%.

5.4.1 Vega results

We present results for the computed model vegas corresponding to the Bermudan swaptions in Table 5.5, using the terminal measure. We do not consider the cancellable inverse floater swaps because the results are similar. Three different vega scenarios are presented, which are shown separately in Tables 5.8, 5.9 and 5.10. The presentation of the results is the same as before where the estimate, its standard error and multiple of standard error reduction is displayed.
Table 5.8: Model vegas and standard errors in basis points for the Bermudan swaptions priced in Table 5.5. All caplet volatilities were increased by 1%.

Table 5.9: Model vegas and standard errors in basis points for the Bermudan swaptions priced in Table 5.5. Three caplet volatilities in the middle of the product’s term structure were perturbed.

Table 5.10 considers the case where just one caplet volatility is perturbed. In each of the different tenor products we adjust the volatility corresponding to a caplet in the middle of the term structure. We see that even for a small change in the pseudo-square roots useful standard error reductions are achieved, e.g. 4 to 62 times for a 5Y Bermudan swaption.

There are two main reasons why the variance reductions for vega are not as large...
5.4. Vega

Table 5.10: Model vegas and standard errors in basis points for the Bermudan swaptions priced in Table 5.5. The middle caplet volatilities were perturbed i.e. those corresponding to the forward rates resetting at times $T_5$, $T_7$, $T_{10}$ and $T_{20}$ for the 5Y, 7Y, 10Y and 20Y products respectively.

<table>
<thead>
<tr>
<th>Product</th>
<th>2 factor</th>
<th>3 factor</th>
<th>5 factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y LMM</td>
<td>1.4 (2.0E-02)</td>
<td>1.4 (2.0E-02)</td>
<td>1.4 (2.1E-02)</td>
</tr>
<tr>
<td>Control</td>
<td>1.5 (3.2E-04)</td>
<td>1.5 (3.7E-03)</td>
<td>1.5 (4.8E-03)</td>
</tr>
<tr>
<td>7Y LMM</td>
<td>1.5 (2.2E-02)</td>
<td>1.5 (2.2E-02)</td>
<td>1.5 (2.4E-02)</td>
</tr>
<tr>
<td>Control</td>
<td>1.6 (5.5E-04)</td>
<td>1.6 (4.9E-03)</td>
<td>1.6 (6.3E-03)</td>
</tr>
<tr>
<td>10Y LMM</td>
<td>1.6 (2.3E-02)</td>
<td>1.6 (2.4E-02)</td>
<td>1.6 (2.4E-02)</td>
</tr>
<tr>
<td>Control</td>
<td>1.6 (1.1E-03)</td>
<td>1.7 (6.3E-03)</td>
<td>1.7 (7.8E-03)</td>
</tr>
<tr>
<td>20Y LMM</td>
<td>1.3 (2.0E-02)</td>
<td>1.4 (2.1E-02)</td>
<td>1.4 (2.3E-02)</td>
</tr>
<tr>
<td>Control</td>
<td>1.3 (2.3E-03)</td>
<td>1.4 (8.6E-03)</td>
<td>1.4 (1.1E-02)</td>
</tr>
</tbody>
</table>

as those for pricing. First, we are already using a variance reduction technique for the LMM vega by using the same random numbers to compute a perturbed price, $V^*$, and an unperturbed price, $V$. The vega’s variance

$$\text{Var}[V^*_i - V_i] = \text{Var}[V^*_i] + \text{Var}[V_i] - 2\text{Cov}[V^*_i, V_i],$$

is reduced, because using the same random numbers means that the covariance between the perturbed and unperturbed prices will be high.

Second, the contribution of the third and above factors is larger for the vega than the price. This is not to say that the first two factors do not dominate the vega calculation, but the higher factors still have some influence making the two-factor Markov-functional model slightly less effective for vega than pricing.

The vega results show that the calibration method introduced in Section 5.2 allows for a small perturbation in the LMM volatility structure to be translated into a small perturbation in the separable volatility structure. This makes the control variate effective not only for pricing, but for model vegas as well. The standard error reductions for vega are useful because a 5 times reduction leads to an order of 25 times faster calculation.

### 5.4.2 Alternative calibration

Assume we have used the calibration technique in Section 5.2 to fit the set of unperturbed pseudo-square root matrices $B(i)$ to $A(i)$ for $i = 0, \ldots, n - 1$. To calibrate the collection of perturbed pseudo-square root matrices $B(i)^*$ to those from the LMM, $A(i)^*$, we could use the same routine, that is we minimise the sum of squares differences between the elements of $B(i)^*$ and $A(i)^*$ using

$$[A(i)^* - B(i)^*]^2 \rightarrow \min. \quad (5.3)$$
Alternatively, we could find $B(i)^*$ such that

$$[(A(i)^* - B(i)^*) - (A(i) - B(i))]^2 \rightarrow \min,$$  
(5.4)

so that the differences between the unperturbed and perturbed separable matrices is similar to the differences between the unperturbed and perturbed non-separable matrices. Finding $B(i)^*$ is analogous to Section 5.2 so we do not present the details. We denote the first approach (5.3) as method 1 and the second (5.4) as method 2.

This second calibration technique is appealing because the vega is driven by the differences in the pseudo-square root matrices. If these differences are comparable, the LMM and Markov-functional model should behave similarly and therefore produce a reduced standard error.

In Table 5.11 we display the results using calibration method 1 and 2 when computing the vega for a single caplet, as in Table 5.10. This demonstrates that the alternative calibration methodology, method 2, performs the same or worse than method 1. Therefore, we recommend using method 1.

<table>
<thead>
<tr>
<th></th>
<th>2 factor</th>
<th>3 factor</th>
<th>5 factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y</td>
<td>method 1</td>
<td>62x</td>
<td>5x</td>
</tr>
<tr>
<td></td>
<td>method 2</td>
<td>30x</td>
<td>5x</td>
</tr>
<tr>
<td>7Y</td>
<td>method 1</td>
<td>40x</td>
<td>5x</td>
</tr>
<tr>
<td></td>
<td>method 2</td>
<td>21x</td>
<td>4x</td>
</tr>
<tr>
<td>10Y</td>
<td>method 1</td>
<td>21x</td>
<td>4x</td>
</tr>
<tr>
<td></td>
<td>method 2</td>
<td>12x</td>
<td>4x</td>
</tr>
<tr>
<td>20Y</td>
<td>method 1</td>
<td>8x</td>
<td>2x</td>
</tr>
<tr>
<td></td>
<td>method 2</td>
<td>8x</td>
<td>2x</td>
</tr>
</tbody>
</table>

Table 5.11: Standard error reductions when computing the vega in Table 5.10. Method 1 uses the calibration defined by equation (5.3), while method 2 uses (5.4).

5.5 Control variate $\zeta$

To justify using $\zeta = 1$ throughout this chapter, we now demonstrate how $\zeta$ can be calculated and present estimates for the prices and vegas previously considered. We want to estimate $\zeta$ in the control variate equation

$$V_{control} = Z_i - \zeta(Y_i - \bar{Y}),$$  
(5.5)

where $Z = V_{LMM}$, $Y = V_{MC}$ and $\bar{Y} = V_{PDE}$ for notational convenience. We can think of $\zeta$ as the slope of the least-squares regression line through the LMM and Markov-functional model prices. To minimise the variance of (5.5) using the sample
data, we use

\[ \hat{\zeta} = \frac{\sum_{i=1}^{N} (Y_i - \bar{Y})(Z_i - \bar{Z})}{\sum_{i=1}^{N} (Y_i - \bar{Y})^2}, \]

which has a standard error of

\[ SE(\hat{\zeta}) = \sqrt{\frac{\sum_{i=1}^{N} (Z_i - \bar{Z})^2 / (n - 2)}{\sum_{i=1}^{N} (Y_i - \bar{Y})^2}}. \]

For the Bermudan swaption prices in Table 5.5 and the three vega Tables 5.8-5.10, we compute the corresponding point estimate and standard error of \( \zeta \), where all of the pricing paths were used to compute it, i.e. \( N = 10000 \).

<table>
<thead>
<tr>
<th></th>
<th>2 factor</th>
<th>3 factor</th>
<th>5 factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y</td>
<td>1.00 (0.01)</td>
<td>1.00 (0.01)</td>
<td>1.00 (0.01)</td>
</tr>
<tr>
<td>7Y</td>
<td>1.01 (0.01)</td>
<td>1.01 (0.01)</td>
<td>1.01 (0.01)</td>
</tr>
<tr>
<td>10Y</td>
<td>1.01 (0.01)</td>
<td>1.01 (0.01)</td>
<td>1.01 (0.01)</td>
</tr>
<tr>
<td>20Y</td>
<td>1.03 (0.01)</td>
<td>1.03 (0.01)</td>
<td>1.02 (0.01)</td>
</tr>
</tbody>
</table>

Table 5.12: Estimate and standard error of \( \zeta \) in equation (5.5), corresponding to the pricing Table 5.5

<table>
<thead>
<tr>
<th></th>
<th>2 factor</th>
<th>3 factor</th>
<th>5 factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y</td>
<td>1.01 (0.01)</td>
<td>1.01 (0.01)</td>
<td>1.01 (0.01)</td>
</tr>
<tr>
<td>7Y</td>
<td>1.01 (0.01)</td>
<td>1.02 (0.01)</td>
<td>1.02 (0.01)</td>
</tr>
<tr>
<td>10Y</td>
<td>1.02 (0.01)</td>
<td>1.01 (0.01)</td>
<td>1.03 (0.01)</td>
</tr>
<tr>
<td>20Y</td>
<td>1.06 (0.01)</td>
<td>1.05 (0.01)</td>
<td>1.06 (0.01)</td>
</tr>
</tbody>
</table>

Table 5.13: Estimate and standard error of \( \zeta \) in equation (5.5), corresponding to the vega Table 5.8

<table>
<thead>
<tr>
<th></th>
<th>2 factor</th>
<th>3 factor</th>
<th>5 factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y</td>
<td>1.01 (0.01)</td>
<td>1.00 (0.01)</td>
<td>1.00 (0.01)</td>
</tr>
<tr>
<td>7Y</td>
<td>1.01 (0.01)</td>
<td>1.02 (0.01)</td>
<td>1.02 (0.01)</td>
</tr>
<tr>
<td>10Y</td>
<td>1.03 (0.01)</td>
<td>1.01 (0.01)</td>
<td>1.03 (0.01)</td>
</tr>
<tr>
<td>20Y</td>
<td>1.08 (0.01)</td>
<td>1.06 (0.01)</td>
<td>1.08 (0.01)</td>
</tr>
</tbody>
</table>

Table 5.14: Estimate and standard error of \( \zeta \) in equation (5.5), corresponding to the vega Table 5.9

As we can see the value of \( \hat{\zeta} \) is frequently within two standard errors of 1. It deviates from 1 as the length of the product increases, however it does not diverge very far.
Table 5.15: Estimate and standard error of $\zeta$ in equation (5.5), corresponding to the vega
Table 5.10

We computed the standard error reductions using the above values of $\hat{\zeta}$, instead of $\zeta = 1$, and found the results to be the same. We therefore recommend using a blanket value of $\zeta = 1$. This is useful from a computational perspective because we are not required to spend time computing $\hat{\zeta}$. 

<table>
<thead>
<tr>
<th></th>
<th>2 factor</th>
<th>3 factor</th>
<th>5 factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>5Y</td>
<td>1.01 (0.01)</td>
<td>1.00 (0.01)</td>
<td>1.00 (0.01)</td>
</tr>
<tr>
<td>7Y</td>
<td>1.01 (0.01)</td>
<td>1.02 (0.01)</td>
<td>1.02 (0.01)</td>
</tr>
<tr>
<td>10Y</td>
<td>1.03 (0.01)</td>
<td>1.04 (0.01)</td>
<td>1.03 (0.01)</td>
</tr>
<tr>
<td>20Y</td>
<td>1.08 (0.01)</td>
<td>1.06 (0.01)</td>
<td>1.07 (0.01)</td>
</tr>
</tbody>
</table>
Chapter 6

Markov-functional model

Greeks

This chapter demonstrates how to accurately and efficiently compute Greeks for Markov-functional models. We apply adjoint PDE techniques to compute the deltas, vegas and skew sensitivities, which can compute these terms in the same time as the price. Using adjoint PDEs in finance has been previously suggested, but resources are sparse. There is a presentation by Prideaux, [76], however it only considers the Black-Scholes model. The main contribution of this chapter is to introduce and derive the method for Markov-functional models, demonstrating all the details required for a practical implementation.

Throughout this chapter we will denote the sensitivity to the non-Markov terms as *mapping Greeks*. These are typically the sensitivities to the initial state variables and non time-dependent volatility terms, for example $f(0), \nu$ and $\alpha$ in the separable LIBOR model of Chapter 4. We define the *Markov Greeks* to be the sensitivity to the time-dependent volatility parameters, which are the $c_{ij}(t)$ terms in the separable LIBOR model.

As in Chapter 3 we make the assumption that the payoff function is Lipschitz continuous. We note that it may be possible to extend the method to payoff functions with a single discontinuity, using a similar approach to that of Chan and Joshi, [23].

This chapter is based on the paper [28].
6.1 PDE Greeks

6.1.1 Finite difference Greeks

One simple method to compute sensitivities is via finite difference, or bump and revalue, which was the method employed in Chapter 5. We compute the price with the initial market parameters, \( \theta_0 \), then perturb a single parameter by a small amount, \( \epsilon \) and recompute the price. The sensitivity is then given by

\[
\delta'(u(\theta_0), 0) = \frac{h(u(\theta_0 + \epsilon), 0) - h(u(\theta_0), 0)}{\epsilon},
\]

where \( h(u, 0) \) is the initial price of the option. The main benefit of this method is its ease of implementation. There are, however, many drawbacks (see, for example Piterbarg, [73]) the main ones being

- Careful choice of the perturbation size \( \epsilon \).
- At least one extra price needs to be computed for each extra sensitivity.

Choosing an appropriate perturbation can be difficult and depends on the magnitude of the parameter \( \theta \). Ideally we want to choose a small value to reduce the bias of the estimate, however a bump too small will lose the sensitivity in numerical noise.

For most financial models, particularly those used in interest rate markets, there are a large number of Greeks. For a medium-dated product the number can easily be greater than fifty. If we need to compute an extra fifty prices, then calculating the Greeks will be expensive, even for a fast model.

6.1.2 Computing Greeks directly

We can avoid the bias of the finite difference Greeks by computing them directly. Since we are considering Lipschitz-continuous payoff functions, we can differentiate and derive a PDE for the particular Greek of interest. Consider the sensitivity to a model parameter \( \theta \). We let

\[
\Delta_{\theta} := \frac{\partial g}{\partial \theta}.
\]

The idea is to apply this partial derivative to the pricing equation (4.4). If we are computing mapping Greeks then the partial differential operator will not contain \( \theta \)
and we have

$$\frac{\partial}{\partial \theta} \left( \frac{\partial g}{\partial t} = Pg \right) \Rightarrow \left( \frac{\partial^2 g}{\partial t \partial \theta} = \frac{\partial}{\partial \theta} Pg \right),$$

$$\Rightarrow \left( \frac{\partial \Delta \theta}{\partial t} = P \Delta \theta \right),$$

since partial differentiation commutes. We can use the method of lines to transform this PDE into an ODE and solve for the single Greek.

If we are computing Markov Greeks the PDE will explicitly contain $\theta$. We can still derive an equation for the Greek, however, the partial differential operator will change during differentiation.

The main benefit of this method is that the sensitivities are computed without bias. One problem, however, is that we need to solve an additional equation for each extra Greek. We can reuse some information, particularly for the mapping Greeks as the only change is the terminal condition, not the PDE itself. However, this requires adjustments to the particular lattice solver, which is not ideal.

### 6.1.3 Mapping adjoint Greeks

The adjoint PDE method computes all of the first order mapping Greeks, without bias, in approximately the same time as it takes to compute the price. It works by calculating an intermediate quantity first and then computes the sensitivities. We initially consider a single time horizon 0 and $T_n$, before moving on to multiple time horizons in the next sub-section, therefore for ease of notation let $h(u, 0) = h(u)$.

The following derivation is adapted from the more general case of differential algebraic equations presented by Cao, Li, Petzold and Serban, [21]. We want to find the adjoint of the large system of ODEs that computes the price of the option given by (4.5), where we ignore the vector of boundary conditions for ease of exposition. In any case, the spatial discretisation is usually large enough that the boundary conditions do not significantly affect the price, see Tavella and Randall, [84]. The solution to this system of ODEs is given by

$$u(t) = e^{At} u_{T_n}.$$

Rewriting this gives

$$\gamma(u, \theta) := e^{-At} u(t) - u_{T_n} = 0.$$  \hspace{1cm} (6.2)

We want to compute the sensitivity of the initial price, given by $h(u)$, to a mapping
parameter \( \theta \). To do so we introduce a Lagrange multiplier \( \lambda \)

\[
I(u, \theta) := h(u) - \lambda^T \gamma(u, \theta) = h(u),
\]

where the right equality comes from the definition of \( \gamma \). Here \( \lambda \) is not a free parameter in the usual Lagrange multiplier sense, but will be specified by the adjoint equation. The sensitivity of the price is now given by

\[
\frac{\partial I}{\partial \theta} = \frac{\partial h}{\partial u} \frac{\partial u}{\partial \theta} - \lambda^T \left( \frac{\partial \gamma}{\partial \theta} + \frac{\partial \gamma}{\partial u} \frac{\partial u}{\partial \theta} \right),
\]

\[
= -\lambda^T \frac{\partial \gamma}{\partial \theta} + \left( \frac{\partial h}{\partial u} - \lambda^T \frac{\partial \gamma}{\partial u} \right) \frac{\partial u}{\partial \theta}.
\]

The issue with computing the Greeks directly in the previous sub-section is that a new equation needs to be solved for each parameter \( \theta \). This problem also appears above, expressed as the partial derivative \( \partial u/\partial \theta \). We therefore choose the parameter \( \lambda \) to eliminate this term by setting

\[
\frac{\partial h}{\partial u} - \lambda^T \frac{\partial \gamma}{\partial u} = 0.
\]

Using (6.2) we can compute

\[
\frac{\partial \gamma}{\partial u} = e^{-At},
\]

which, after transposing, gives

\[
e^{-AT} \lambda(t) - \left( \frac{\partial h}{\partial u} \right)^T = 0.
\]

This equation is the solution of an ODE in the same way as (6.2) is. We can therefore specify the system of equations as

\[
\frac{d\lambda(t)}{dt} = A^T \lambda(t),
\]

\[
\lambda(0) = \frac{\partial h(u)}{\partial u}.
\]

This is the adjoint system, which we will use to compute the Greeks. This equation is solved forwards in time, since the function \( h \) is only defined at time 0. Its partial derivative, which gives the initial conditions, is

\[
\lambda(0) = \begin{cases} 1, & \text{at } \theta_0, \\ 0, & \text{otherwise}. \end{cases}
\]
The adjoint equation is no more difficult to solve than the pricing equation. We simply calculate forwards in time to get the vector solution $\lambda(T_n)$. To compute a Greek we put the adjoint solution back into (6.3)

$$\frac{\partial h}{\partial \theta} = -\lambda^T \frac{\partial \gamma}{\partial \theta} = -\lambda^T \left( \frac{\partial e^{-At}}{\partial \theta} u - \frac{\partial u_{T_n}}{\partial \theta} \right).$$

Since $A$ is a matrix the derivative of the first term appearing inside the brackets above is given by the general formula for matrix exponentials from [64]

$$\frac{\partial e^{-At}}{\partial \theta} = -\int_0^{T_n} e^{-As} \frac{\partial A}{\partial \theta} e^{-A(T_n-s)} ds,$$

which gives

$$\frac{\partial h}{\partial \theta} = \int_0^{T_n} \left( \lambda^T (T_n - s) \frac{\partial A}{\partial \theta} u(s) \right) ds + \lambda(T_n)^T \frac{\partial u_{T_n}}{\partial \theta}, \quad (6.6)$$

where $\partial u_{T_n}/\partial \theta$ is the derivative of the payoff function with respect to a parameter of interest at each of the nodes. For the mapping Greeks the term $\partial A/\partial \theta = 0$ and the expression is significantly simplified

$$\frac{\partial h}{\partial \theta} = \lambda(T_n)^T \frac{\partial u_{T_n}}{\partial \theta}.$$

This equation along with (6.4) are used to compute all of the mapping Greeks. The key point is that the adjoint equation (6.4) does not contain $\theta$ and only needs to be solved once. Each additional sensitivity then only requires an extra dot product, we do not need to solve another equation.

The adjoint method is numerically stable if the pricing scheme is stable. Since the adjoint system is solved forwards in time we can do a change of variable from $t$ to $T_n - t$ and the adjoint system becomes

$$\frac{d\lambda(T_n - t)}{dt} = A^T \lambda(T_n - t),$$

which is solved backwards in time. As $A$ is a square constant matrix, its eigenvalues are the same as the eigenvalues of its transpose. Therefore, the adjoint system will be stable if the pricing system is also stable. In the more general case of a non-constant matrix $A$ the adjoint system is also stable, see Theorem 4.1 in [21].

To derive the adjoint method we started with the ODE, which arose from spatial discretisation of the PDE. We note that there can be some differences with the
adjoint method derived directly from the PDE. These discrepancies, however, are small and will not lead to incorrect results for the Greeks, see Li and Petzold, [58].

6.1.4 Mapping adjoint Greeks for Bermudan products

The complexity of Bermudan products involves incorporating the exercise strategy into the adjoint equation. Analogously to Section 3.3, we use the same strategy for the price and the first order Greeks, because the sensitivity of the exercise strategy to a model parameter is only a second order effect.

We solve the adjoint PDE across exercise dates in the same way as we compute the price. The exercise strategy then only enters through the initial conditions, where we zero the nodes at points of exercise. We do this to ensure that the product is not exercised multiple times. The algorithm to compute the Greeks is

1. Perform the pricing routine storing the exercise times and computing and storing all of the pathwise derivatives at each node on each exercise date. That is, for one particular node at time $T_k$ we will have a vector containing elements

$$\frac{\partial g(T_k)}{\partial \theta},$$

of all model parameters of interest $\theta$ and a variable specifying whether we exercised at that date.

2. Initialise the adjoint equation at time zero using the same conditions as the single time step case (6.5). Solve the adjoint equation (6.4) to the first exercise date, $T_0$.

3. Compute and store the Greeks at time $T_0$ using the inner product of the adjoint solution and the stored payoff derivatives

$$\lambda(T_0)^T \frac{\partial u(T_0)}{\partial \theta},$$

for each element of $\theta$.

4. Using the stored exercise times from the pricing phase, initialise the adjoint equation at node $j$, time $T_0$ with

$$\lambda_j(T_0) = \begin{cases} 
\lambda_j(T_0^-), & \text{if not exercised at node } j, \\
0, & \text{if exercised at node } j,
\end{cases}$$

where $\lambda(T_0^-)$ is the solution of the previous adjoint equation. Solve for $\lambda$ to the next exercise date $T_1$. 
5. Compute the inner product

$$\lambda(T_1)^T \frac{\partial u(T_1)}{\partial \theta},$$

for each element of $\theta$ and add these values to the inner products computed in step 3.

6. Repeat steps 4 and 5 until the final exercise time $T_{n-1}$. The sensitivity of the option price to a parameter $\theta_j$ is then

$$\frac{\partial h}{\partial \theta_j} = \sum_{k=0}^{n-1} \lambda(T_k)^T \frac{\partial u(T_k)}{\partial \theta_j}.$$

One potential issue with the adjoint method is that at step 1 for each grid point at each exercise date a vector of derivatives needs to be stored. If there are, for example, 2500 grid points (a two-factor model with a $50 \times 50$ grid), 40 exercise dates and 100 sensitivities of interest then we need to store 10 million values. This is about 100 megabytes, which is a small amount for a modern computer. In any case, if memory is an issue we can re-evolve the stochastic process for the forward adjoint step and compute the pathwise derivatives as we need them in steps 3 and 5. We then only save the running sum in step 6, which requires a minimal amount of memory.

### 6.1.5 Markov Greeks

We have so far neglected the Markov Greeks, or the sensitivity to the variables which make up the Markov factors. For the case of the separable LIBOR Markov-functional model these are the volatility terms $c_{ij}(k)$ and we are interested in computing the vegas

$$\frac{\partial h(u)}{\partial c_{ij}(k)}.$$

The difference between calculating mapping and Markov vegas using the adjoint method is that the integral correction term in (6.6) will no longer be zero. To evaluate this integral we need to store both the adjoint terms $\lambda$ and the price terms $u$ for all nodes where the integrand is to be determined, which makes the Markov Greeks more memory intensive than the mapping Greeks.

Consider the example of piecewise constant Markov terms $C(k + 1)$ across the interval $[T_k, T_{k+1})$, which is the case for the LIBOR Markov-functional model in this thesis. To compute the sensitivity to these Markov terms we have three different
time intervals to consider, before $T_k$, across $[T_k, T_{k+1})$ and after $T_{k+1}$.

Over the region $[0, T_k)$ the derivative of $\partial A/\partial C(k+1)$ is zero and we can therefore use the same algorithm as the mapping Greeks in Section 6.1.4. That is we compute the summation in step 6 until time $T_k$.

Between reset times $T_k$ and $T_{k+1}$ the derivative of $\partial A/\partial C(k+1)$ will be non zero and we need to compute the integral correction term. To estimate this integral we use the following algorithm:

1. Ensure that the same time discretisation is used when numerically solving the adjoint and pricing PDEs. Store the vectors $\lambda(T_{k+1} - s)$ and $u(s)$ at each time step $s$.

2. Compute $\partial A/\partial C(k+1)$.

3. Evaluate $\lambda^T(T_{k+1} - s) \frac{\partial A}{\partial C(k+1)} u(s)$ for each time step $s$.

4. Calculate the integral above using a numerical integration technique.

The resulting integral and the term $\lambda(T_{k+1})^T \frac{\partial u(T_{k+1})}{\partial C(k+1)}$ is then added to the summation computed up to time $T_k$.

After time $T_{k+1}$ the integral correction term will be zero and we continue computing the vega using the algorithm in the mapping Greeks Section.

One way to ensure we use the same time discretisation in step 1 above is to use a fixed time stepping technique such as the ADI method discussed in Section 4.5. The integrand in step 3 will generally be well behaved because both $\lambda$ and $u$ are for realistic payoff functions and the term $\partial A/\partial C(k+1)$ essentially pulls out a linear multiple of $\lambda$. This means that we can use the trapezoidal rule to perform the numerical integration.

### 6.2 Markov-functional model derivatives

To apply the adjoint PDE Greeks to the separable LIBOR Markov-functional model we need to differentiate the discretisation scheme. The following model derivatives are calculated using the method in Sections 3.4.1 and 3.4.2, which decomposes the predictor-corrector discretisation into four sub-maps. We choose to focus on the terminal measure, because it produces better results than the spot measure, see Section 4.7.
6.2. Markov-functional model derivatives

6.2.1 Markov-functional model delta

Using the single-step predictor-corrector scheme in log coordinates the forward rates are evolved from time 0 to time $T_k$ using

$$\log(f_i(T_k) + \alpha_i) = \log(f_i(0) + \alpha_i) + \mu_i(0)^{PC} - \frac{\text{cov}_{ii}(0)}{2} + \nu^T X(T_k), \quad (6.7)$$

where $\mu_i(0)^{PC}$ is the integrated drift, $\text{cov}_{ii}(0)$ is the variance of the forward rate across the time step and

$$X(T_k) = \int_0^{T_k} C(t) dW(t).$$

To compute the delta we need to differentiate this discretisation scheme with respect to the initial forward rates since

$$\frac{\partial g(f(T_k))}{\partial f(0)} = \frac{\partial g(f(T_k))}{\partial f(T_k)} \frac{\partial f(T_k)}{\partial f(0)}.$$

We do so using the same technique as Section 3.4.1. This involves splitting up the evolution into a series of sub-maps, with each sub-map updating a vector. The four sub-maps $H_i$ compute the initial drifts, predicted rates, predicted drifts and then the evolved rates. Graphically we have

$$\begin{bmatrix} \log(f(0) + \alpha) \end{bmatrix} \xrightarrow{H_0} \begin{bmatrix} \log(f(0) + \alpha) \\ \mu(0) \end{bmatrix} \xrightarrow{H_1} \begin{bmatrix} \log(f(0) + \alpha) \\ \mu(0) \\ \hat{f}(0) \end{bmatrix} \xrightarrow{H_2} \begin{bmatrix} \log(f(0) + \alpha) \\ \mu(0) \\ \hat{\mu}(0) \end{bmatrix} \xrightarrow{H_3} \begin{bmatrix} \log(f(T_k) + \alpha) \end{bmatrix}. $$

We differentiate these sub-maps to compute the Jacobian matrices and then multiply backwards to time 0. The backwards multiplication can be performed with a computational order of $nF$. This is particularly useful for a Markov model as $F$ is usually small, typically 1 or 2.

6.2.2 Markov-functional model mapping vegas

To compute the mapping vegas using the adjoint method we calculate

$$\frac{\partial g(f(T_k))}{\partial \nu_{jf}}.$$
Since we are using a single-step discretisation we can express this sensitivity as
\[
\frac{\partial g(f(T_k))}{\partial \nu_{j\ell}} = \frac{\partial g(f(T_k))}{\partial f(T_k)} \frac{\partial f(T_k)}{\partial \nu_{j\ell}},
\]
where the derivative of the payoff function has already been computed during the calculation of the delta. All that remains is to differentiate the discretisation scheme. Before doing so we need to express the covariances (which appear in the drift (2.5) and drift correction (6.7)) in terms of the forward-rate-specific vectors. The covariance between rates \(i\) and \(j\) over the interval \([T_a, T_b]\), for \(b > a\), is
\[
\text{cov}_{ij}(T_a, T_b) = \sum_{l=a}^{b} [\nu_l^T C(l)] [\nu_l^T C(l)]^T.
\]
We now break-up the discretisation scheme into four sub-maps in the same way as computing the delta, however we now include the volatility terms
\[
\begin{pmatrix}
\nu \\
\log(f(0) + \alpha)
\end{pmatrix}
\xrightarrow{H_0}
\begin{pmatrix}
\nu \\
\log(f(0) + \alpha) \\
\mu(0)
\end{pmatrix}
\xrightarrow{H_1}
\begin{pmatrix}
\nu \\
\log(f(0) + \alpha) \\
\mu(0) \\
\hat{f}(0)
\end{pmatrix}
\xrightarrow{H_2}
\begin{pmatrix}
\nu \\
\log(f(0) + \alpha) \\
\mu(0) \\
\hat{\mu}(0)
\end{pmatrix}
\xrightarrow{H_3}
\begin{pmatrix}
\nu \\
\log(f(T_k) + \alpha)
\end{pmatrix}.
\]
Starting with the last sub-map (since we are working in a backwards direction) we differentiate to give
\[
\frac{\partial \log(f_i(T_k) + \alpha_i)}{\partial \nu_{j\ell}} = \begin{cases} 
-e_{ij}(0, T_k) + X_f(T_k), & \eta(T_k) \leq i = j, \\
0, & \text{otherwise},
\end{cases}
\]
where
\[
e_{ij}(0, T_k) = \sum_{l=0}^{T_k} [\nu_l^T C(l)] \text{row}(C(l), f)^T,
\]
with \(\text{row}(C(l), f)\) being the \(f\)th row of the matrix \(C(l)\). We will need to use these \(e_{ij}\) terms regularly and since they only involve initial inputs we can pre-compute them in the constructor of our computer program. Focusing on the sub-map \(H_2\) we
obtain

\[
\frac{\partial \hat{\mu}_i(0)}{\partial \nu_{jf}} = \begin{cases} 
- \sum_{l=i+1}^{n} (\hat{\mu}_l(0)+\alpha_l)_{\tau_l} e_{lj}(0,T_k), & \eta(T_k) \leq i = j, \\
- (\hat{\mu}_j(0)+\alpha_j)_{\tau_j} e_{ij}(0,T_k), & \eta(T_k) \leq i < j, \\
0, & \text{otherwise},
\end{cases}
\]

(6.8)

For the sub-map \(H_1\) we differentiate to give

\[
\frac{\partial \hat{f}_i(0)}{\partial \nu_{jf}} = \begin{cases} 
(\hat{f}_i(0) + \alpha_i) (-e_{if}(0,T_k) + X_f(T_k)), & \eta(T_k) \leq i = j, \\
0, & \text{otherwise},
\end{cases}
\]

For the final sub-map \(H_0\) the partial derivatives are the same as (6.8), but with \(\hat{f}\) replaced by \(f\). By taking advantage of the structure of the Jacobian matrices we can multiply the vega back to time zero in order \(nF\) calculations, in the same manner as Chapter 2.

### 6.2.3 Markov-functional model Markov vegas

To compute the sensitivity of the price to a Markov vega or \(c_{jq}(p)\) term we once again need to differentiate the discretisation scheme. For this subsection we will focus on a two-factor model for ease of exposition. Using the chain rule we have

\[
\frac{\partial g(f(T_k))}{\partial c_{jq}(p)} = \frac{\partial g(f(T_k))}{\partial f(T_k)} \frac{\partial f(T_k)}{\partial c_{jq}(p)}.
\]

We will split the discretisation scheme into the same four sub-maps as in the above mapping vegas section. Starting with the last sub-map \(H_3\) and recalling that \(Y_i(k) = \log(f_i(T_k) + \alpha_i)\), we have for \(p \leq k\)

\[
\frac{\partial Y_i(k)}{\partial c_{jq}(p)} = \begin{cases} 
-(\nu_0^2 c_{11}(p) + \nu_0 \nu_1 c_{21}(p)) + \nu_0 (W_1(T_{p+1}) - W_1(T_p)), & j = 1, q = 1, \\
-(\nu_0^2 c_{12}(p) + \nu_0 \nu_1 c_{22}(p)) + \nu_0 (W_2(T_{p+1}) - W_2(T_p)), & j = 1, q = 2, \\
-(\nu_1^2 c_{21}(p) + \nu_0 \nu_1 c_{11}(p)) + \nu_1 (W_1(T_{p+1}) - W_1(T_p)), & j = 2, q = 1, \\
-(\nu_1^2 c_{22}(p) + \nu_0 \nu_1 c_{12}(p)) + \nu_1 (W_2(T_{p+1}) - W_2(T_p)), & j = 2, q = 2,
\end{cases}
\]

(6.9)

where \(W_i(T_{p+1}) - W_i(T_p)\) is the increment of the \(i\)th Brownian motion over the interval \([T_p, T_{p+1})\), which arises from (4.9). Focusing our attention on the sub-map
\[ H_2 \text{ we obtain} \]
\[
\frac{\partial \hat{\mu}_i(0)}{\partial c_{jq}(p)} = \begin{cases} 
\sum_{l=i+1}^{n-1} \frac{(\hat{f}_l(0) + \alpha_l) \tau_l}{1 + \gamma_j f_l(0)} (2\nu_{j0}\nu_{l0} c_{11}(p) + (\nu_{j0}\nu_{l1} + \nu_{j1}\nu_{l0}) c_{21}(p)), & j = 1, q = 1, \\
\sum_{l=i+1}^{n-1} \frac{(\hat{f}_l(0) + \alpha_l) \tau_l}{1 + \gamma_j f_l(0)} (2\nu_{j1}\nu_{l1} c_{21}(p) + (\nu_{j0}\nu_{l1} + \nu_{j1}\nu_{l0}) c_{11}(p)), & j = 1, q = 2, \\
\sum_{l=i+1}^{n-1} \frac{(\hat{f}_l(0) + \alpha_l) \tau_l}{1 + \gamma_j f_l(0)} (2\nu_{j0}\nu_{l0} c_{12}(p) + (\nu_{j0}\nu_{l1} + \nu_{j1}\nu_{l0}) c_{22}(p)), & j = 2, q = 1, \\
\sum_{l=i+1}^{n-1} \frac{(\hat{f}_l(0) + \alpha_l) \tau_l}{1 + \gamma_j f_l(0)} (2\nu_{j1}\nu_{l1} c_{22}(p) + (\nu_{j0}\nu_{l1} + \nu_{j1}\nu_{l0}) c_{12}(p)), & j = 2, q = 2.
\end{cases}
\]

For the second sub-map \( H_1 \) we differentiate to give
\[
\frac{\partial \hat{f}_i(0)}{\partial c_{jq}(p)} = \begin{cases} 
(\hat{f}_i(0) + \alpha_i) (\cdot), & \eta(T_k) \leq i, \\
0, & \text{otherwise},
\end{cases}
\]
where the (\cdot) term is given by (6.9). For the final sub-map \( H_0 \) the derivative is the same as for the sub-map \( H_2 \), but with \( \hat{f} \) replaced by \( f \) so we do not repeat it here.

Using these derivatives it is possible to multiply the Markov vega terms back with a computational order of \( nF \), in a similar manner as the mapping vegas.

### 6.2.4 Markov-functional model skew sensitivity

To compute the sensitivity to a displacement coefficient \( \alpha_j \) we use the chain rule
\[
\frac{\partial g(f(T_k))}{\partial \alpha} = \frac{\partial g(f(T_k))}{\partial f(T_k)} \frac{\partial f(T_k)}{\partial \alpha}.
\]

To differentiate the discretisation scheme it is easier to work with the rates themselves instead of the log-rates. Using the following sub-maps
\[
\begin{bmatrix} \alpha \\ f(0) \end{bmatrix} \stackrel{H_0}{\rightarrow} \begin{bmatrix} \alpha \\ f(0) \\ \mu(0) \end{bmatrix} \stackrel{H_1}{\rightarrow} \begin{bmatrix} \alpha \\ f(0) \\ \mu(0) \\ \hat{f}(0) \end{bmatrix} \stackrel{H_2}{\rightarrow} \begin{bmatrix} \alpha \\ f(0) \\ \mu(0) \\ \hat{\mu}(0) \end{bmatrix} \stackrel{H_3}{\rightarrow} \begin{bmatrix} \alpha \\ f(T_k) \end{bmatrix},
\]
we compute
\[
\frac{\partial f_i(T_k)}{\partial \alpha_j} = \begin{cases} 
f_i(T_k) + \alpha_i - 1, & \eta(T_k) \leq i = j, \\
0, & \text{otherwise},
\end{cases}
\]
6.3. Numerical results

for \(H_3\). For the previous sub-map \(H_2\) we differentiate to give
\[
\frac{\partial \hat{\mu}_i(0)}{\partial \alpha_j} = \begin{cases} 
\frac{\tau_j}{1 + \tau_j f_j(0)} \text{cov}_{ij}(0), & \eta(T_k) \leq i < j, \\
0, & \text{otherwise.}
\end{cases}
\]

The derivative of the sub-map \(H_1\) is similar to that of \(H_3\), but with \(f\) replaced by \(\hat{f}\)
\[
\frac{\partial \hat{f}_i(T_k)}{\partial \alpha_j} = \begin{cases} 
\frac{\hat{f}_i(T_k) + \alpha_i}{f_i(0) + \alpha_i} - 1, & \eta(T_k) \leq i = j, \\
0, & \text{otherwise.}
\end{cases}
\]

Similarly the derivative of \(H_0\) is the same as \(H_2\) with \(\hat{f}\) replaced by \(f\). With these derivatives we are now in a position to calculate the sensitivity of the price with respect to the displacement terms. The calculation of the skew sensitivity can be computed in order \(nF\) operations.

6.3 Numerical results

In this section we demonstrate how well the adjoint method works for the separable LIBOR Markov-functional model. Three products are considered, an interest rate cap, a cancellable inverse floater swap and a callable Bermudan swaption, all with 10 year maturities. Semi-annual forward rates are used, flat at 5% with displacements of 1% for each forward rate. A strike of 7% is used for the cap, because an out-of-the-money product is less linear in volatility and it produces a more difficult test.

For the cancellable inverse floater we take the point of view of the issuer who receives the floating LIBOR rate and pays the coupon
\[(K - 2f_i(T_i))^+ \tau_i,\]
at time \(T_{i+1}\). The strike is 10%. Upon cancellation at any of the tenor dates \(T_0, \ldots, T_{n-1}\) no rebate is paid.

For the callable Bermudan swaption we model the product as an option to enter into an interest rate swap at any of the tenor dates above. The strike is set to 7%.

We use a two-factor model with the volatility structure
\[\nu_i^T = \begin{bmatrix} 0.10 & 0.08 \end{bmatrix},\]
where the forward rate specific volatility terms are the same for all forward rates.
and the time-dependent matrices are all equal to

\[ C(k) = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}. \]

Ignoring displacements, this equates to an annual implied Black volatility of approximately 21%. To solve the pricing and adjoint PDEs we use the Krylov subspace method, with a symmetric and equally spaced grid.

To examine the accuracy of the model and adjoint method Table 6.1 displays the exact sensitivities computed using the Black formula and the numerically computed values for the Markov-functional model (MFM) with 51 spatial nodes in each dimension. We present the deltas, half of the mapping vegas and the skew sensitivities. The prices of the individual caplets range from 0.43 bps to 28.3 bps, with the cap worth 356.6 bps. We see that the adjoint method performs well, computing most of the deltas very accurately, except for a few of the later forward rates. The vega and skew sensitivity estimates are also accurate, with a maximum absolute error of 0.2%.

The small discrepancies in the estimates come from two sources, the numerical PDE solution and the model. To eliminate the PDE errors we can use a better solver or a larger grid. We try 101 spatial nodes in each dimension for the results in Table 6.2. Here we observe that the estimates have improved, however the later deltas are still off by close to 1% in some cases. This inaccuracy is most likely due to model error, because we are single stepping the forward rates from time 0 to time \( T_k \) and the drift approximation is becoming inaccurate.

Table 6.3 displays some of the Markov vegas for the interest rate cap. The results are similar for the vega terms not shown, which were removed for ease of presentation. Instead of employing the Krylov subspace method to solve the pricing and adjoint PDEs we used the ADI method, because it features a constant step size making it easier to compute the integral correction term \( (6.6) \). A spatial grid of \( 101 \times 101 \) was used with 50 time steps between each tenor date. To compute the integral correction term we evaluated the integrand at all 50 time points and applied the trapezoidal rule.

We notice that the Markov vegas are computed accurately when compared to their true values from the Black formula. The early vegas have the largest error, but the absolute errors are 0.02% or less.

Table 6.4 shows the sensitivities of the cancellable inverse floater, whose computed value is 2856 bps. Since there is no analytic price we compare the adjoint and finite difference (FD) methods. We employ a double-sided finite difference with a perturbation of 1 bp to each of the model parameters. As we expect the adjoint
6.3. Numerical results

Table 6.1: Deltas, vegas and skew sensitivities for a cap, strike = 7%. A 51×51 spatial grid was used.

<table>
<thead>
<tr>
<th>Delta</th>
<th>Black</th>
<th>MFM</th>
<th>Vega</th>
<th>Black</th>
<th>MFM</th>
<th>Skew</th>
<th>Black</th>
<th>MFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₀</td>
<td>-0.4%</td>
<td>0.4%</td>
<td>ν₀,0</td>
<td>0.2%</td>
<td>0.3%</td>
<td>α₀</td>
<td>0.4%</td>
<td>0.6%</td>
</tr>
<tr>
<td>f₁</td>
<td>2.8%</td>
<td>3.1%</td>
<td>ν₁,₀</td>
<td>0.7%</td>
<td>0.7%</td>
<td>α₁</td>
<td>1.4%</td>
<td>1.6%</td>
</tr>
<tr>
<td>f₂</td>
<td>5.3%</td>
<td>4.9%</td>
<td>ν₂,₀</td>
<td>1.1%</td>
<td>1.1%</td>
<td>α₂</td>
<td>2.4%</td>
<td>2.3%</td>
</tr>
<tr>
<td>f₃</td>
<td>7.1%</td>
<td>7.2%</td>
<td>ν₃,₀</td>
<td>1.4%</td>
<td>1.5%</td>
<td>α₃</td>
<td>3.2%</td>
<td>3.2%</td>
</tr>
<tr>
<td>f₄</td>
<td>8.5%</td>
<td>8.5%</td>
<td>ν₄,₀</td>
<td>1.7%</td>
<td>1.8%</td>
<td>α₄</td>
<td>3.8%</td>
<td>3.8%</td>
</tr>
<tr>
<td>f₅</td>
<td>9.5%</td>
<td>9.0%</td>
<td>ν₅,₀</td>
<td>1.9%</td>
<td>2.0%</td>
<td>α₅</td>
<td>4.3%</td>
<td>4.2%</td>
</tr>
<tr>
<td>f₆</td>
<td>10.3%</td>
<td>10.4%</td>
<td>ν₆,₀</td>
<td>2.1%</td>
<td>2.3%</td>
<td>α₆</td>
<td>4.8%</td>
<td>4.8%</td>
</tr>
<tr>
<td>f₇</td>
<td>11.0%</td>
<td>10.7%</td>
<td>ν₇,₀</td>
<td>2.3%</td>
<td>2.4%</td>
<td>α₇</td>
<td>5.1%</td>
<td>5.1%</td>
</tr>
<tr>
<td>f₈</td>
<td>11.5%</td>
<td>11.4%</td>
<td>ν₈,₀</td>
<td>2.4%</td>
<td>2.6%</td>
<td>α₈</td>
<td>5.5%</td>
<td>5.5%</td>
</tr>
<tr>
<td>f₉</td>
<td>11.9%</td>
<td>11.7%</td>
<td>ν₉,₀</td>
<td>2.6%</td>
<td>2.7%</td>
<td>α₉</td>
<td>5.7%</td>
<td>5.7%</td>
</tr>
<tr>
<td>f₁₀</td>
<td>12.2%</td>
<td>12.3%</td>
<td>ν₁₀,₀</td>
<td>2.7%</td>
<td>2.9%</td>
<td>α₁₀</td>
<td>6.0%</td>
<td>6.0%</td>
</tr>
<tr>
<td>f₁₁</td>
<td>12.5%</td>
<td>12.1%</td>
<td>ν₁₁,₀</td>
<td>2.7%</td>
<td>2.9%</td>
<td>α₁₁</td>
<td>6.2%</td>
<td>6.1%</td>
</tr>
<tr>
<td>f₁₂</td>
<td>12.7%</td>
<td>12.5%</td>
<td>ν₁₂,₀</td>
<td>2.8%</td>
<td>3.0%</td>
<td>α₁₂</td>
<td>6.3%</td>
<td>6.3%</td>
</tr>
<tr>
<td>f₁₃</td>
<td>12.9%</td>
<td>12.7%</td>
<td>ν₁₃,₀</td>
<td>2.9%</td>
<td>3.1%</td>
<td>α₁₃</td>
<td>6.5%</td>
<td>6.5%</td>
</tr>
<tr>
<td>f₁₄</td>
<td>13.0%</td>
<td>12.3%</td>
<td>ν₁₄,₀</td>
<td>2.9%</td>
<td>3.1%</td>
<td>α₁₄</td>
<td>6.6%</td>
<td>6.5%</td>
</tr>
<tr>
<td>f₁₅</td>
<td>13.1%</td>
<td>12.4%</td>
<td>ν₁₅,₀</td>
<td>3.0%</td>
<td>3.1%</td>
<td>α₁₅</td>
<td>6.7%</td>
<td>6.6%</td>
</tr>
<tr>
<td>f₁₆</td>
<td>13.2%</td>
<td>12.5%</td>
<td>ν₁₆,₀</td>
<td>3.0%</td>
<td>3.1%</td>
<td>α₁₆</td>
<td>6.8%</td>
<td>6.7%</td>
</tr>
<tr>
<td>f₁₇</td>
<td>13.3%</td>
<td>12.4%</td>
<td>ν₁₇,₀</td>
<td>3.0%</td>
<td>3.1%</td>
<td>α₁₇</td>
<td>6.8%</td>
<td>6.7%</td>
</tr>
<tr>
<td>f₁₈</td>
<td>13.3%</td>
<td>12.4%</td>
<td>ν₁₈,₀</td>
<td>3.0%</td>
<td>3.1%</td>
<td>α₁₈</td>
<td>6.9%</td>
<td>6.8%</td>
</tr>
<tr>
<td>f₁₉</td>
<td>13.4%</td>
<td>12.6%</td>
<td>ν₁₉,₀</td>
<td>3.0%</td>
<td>3.1%</td>
<td>α₁₉</td>
<td>6.9%</td>
<td>6.9%</td>
</tr>
</tbody>
</table>

method matches the finite difference almost exactly. The small differences between some values are mostly likely caused by numerical noise in the solution of the PDE. The adjoint method will be more accurate than the finite difference approach as it does not suffer from bias introduced by the numerical bumping.

We do not present the results for the Bermudan swaptions at this stage, instead we present them in the following section.

It is worthwhile considering how long the Greek computations take in the adjoint approach, because this is where the method really shines. To compute the price and all of the 20 deltas, 40 mapping vegas and 20 skew sensitivities (80 Greeks in total) using a 51×51 grid took 1.0 seconds using single threaded C++ code on a desktop computer. Of this time, 0.6 seconds were spent computing all of the Greeks. For the case of a 101×101 grid the price and sensitivities were computed in 4.0 seconds with 2.2 seconds spent computing the Greeks. This demonstrates that we can compute all of the sensitivities in approximately the same time as it takes to compute only one Greek using finite difference. These results are for a two-factor model and it is therefore likely that a one-factor model would be significantly faster.

For the Markov vegas we used the slower ADI method to solve the pricing and

13.16GHz Intel CPU, with 4GB of RAM.
6.4 Market Greeks

To hedge an option, we need to know its sensitivity to changes in market traded instruments such as caplets and swaptions. The previous section, however, computed the option’s sensitivity to model parameters, which are not traded. We therefore need a way to convert model Greeks into market Greeks.

Table 6.2: Deltas, vegas and skew sensitivities for a cap, strike = 7%. A 101×101 spatial grid was used.
Table 6.3: Markov vegas for a cap, strike = 7%. A 101×101 spatial grid was used.

### 6.4. Market Greeks

<table>
<thead>
<tr>
<th>Vega</th>
<th>Black</th>
<th>MFM</th>
<th>Vega</th>
<th>Black</th>
<th>MFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_{12}(0))</td>
<td>0.18%</td>
<td>0.17%</td>
<td>(c_{21}(0))</td>
<td>0.25%</td>
<td>0.27%</td>
</tr>
<tr>
<td>(c_{12}(1))</td>
<td>0.17%</td>
<td>0.16%</td>
<td>(c_{21}(1))</td>
<td>0.24%</td>
<td>0.26%</td>
</tr>
<tr>
<td>(c_{12}(2))</td>
<td>0.16%</td>
<td>0.15%</td>
<td>(c_{21}(2))</td>
<td>0.23%</td>
<td>0.24%</td>
</tr>
<tr>
<td>(c_{12}(3))</td>
<td>0.15%</td>
<td>0.13%</td>
<td>(c_{21}(3))</td>
<td>0.21%</td>
<td>0.22%</td>
</tr>
<tr>
<td>(c_{12}(4))</td>
<td>0.13%</td>
<td>0.12%</td>
<td>(c_{21}(4))</td>
<td>0.19%</td>
<td>0.20%</td>
</tr>
<tr>
<td>(c_{12}(5))</td>
<td>0.12%</td>
<td>0.11%</td>
<td>(c_{21}(5))</td>
<td>0.17%</td>
<td>0.19%</td>
</tr>
<tr>
<td>(c_{12}(6))</td>
<td>0.11%</td>
<td>0.10%</td>
<td>(c_{21}(6))</td>
<td>0.16%</td>
<td>0.17%</td>
</tr>
<tr>
<td>(c_{12}(7))</td>
<td>0.10%</td>
<td>0.09%</td>
<td>(c_{21}(7))</td>
<td>0.14%</td>
<td>0.15%</td>
</tr>
<tr>
<td>(c_{12}(8))</td>
<td>0.09%</td>
<td>0.08%</td>
<td>(c_{21}(8))</td>
<td>0.13%</td>
<td>0.13%</td>
</tr>
<tr>
<td>(c_{12}(9))</td>
<td>0.08%</td>
<td>0.07%</td>
<td>(c_{21}(9))</td>
<td>0.11%</td>
<td>0.12%</td>
</tr>
<tr>
<td>(c_{12}(10))</td>
<td>0.07%</td>
<td>0.06%</td>
<td>(c_{21}(10))</td>
<td>0.10%</td>
<td>0.10%</td>
</tr>
<tr>
<td>(c_{12}(11))</td>
<td>0.06%</td>
<td>0.05%</td>
<td>(c_{21}(11))</td>
<td>0.09%</td>
<td>0.09%</td>
</tr>
<tr>
<td>(c_{12}(12))</td>
<td>0.05%</td>
<td>0.04%</td>
<td>(c_{21}(12))</td>
<td>0.07%</td>
<td>0.08%</td>
</tr>
<tr>
<td>(c_{12}(13))</td>
<td>0.04%</td>
<td>0.04%</td>
<td>(c_{21}(13))</td>
<td>0.06%</td>
<td>0.06%</td>
</tr>
<tr>
<td>(c_{12}(14))</td>
<td>0.04%</td>
<td>0.03%</td>
<td>(c_{21}(14))</td>
<td>0.05%</td>
<td>0.05%</td>
</tr>
<tr>
<td>(c_{12}(15))</td>
<td>0.03%</td>
<td>0.02%</td>
<td>(c_{21}(15))</td>
<td>0.04%</td>
<td>0.04%</td>
</tr>
<tr>
<td>(c_{12}(16))</td>
<td>0.02%</td>
<td>0.02%</td>
<td>(c_{21}(16))</td>
<td>0.03%</td>
<td>0.03%</td>
</tr>
<tr>
<td>(c_{12}(17))</td>
<td>0.02%</td>
<td>0.01%</td>
<td>(c_{21}(17))</td>
<td>0.02%</td>
<td>0.02%</td>
</tr>
<tr>
<td>(c_{12}(18))</td>
<td>0.01%</td>
<td>0.01%</td>
<td>(c_{21}(18))</td>
<td>0.02%</td>
<td>0.01%</td>
</tr>
<tr>
<td>(c_{12}(19))</td>
<td>0.01%</td>
<td>0.00%</td>
<td>(c_{21}(19))</td>
<td>0.01%</td>
<td>0.01%</td>
</tr>
</tbody>
</table>

6.4.1 Market vega

The idea is to find the linear combination of model vegas that corresponds to a market vega. For an \(F\)-factor model with \(n\) forward rates we can realistically expect to hedge with \(nF\) market instruments. In the case of a single-factor model these would most likely be \(n\) coterminal swaptions. For the two-factor model considered in this chapter they could be caplets as well as coterminal swaptions.

To calculate market vegas we adapt the method introduced by Joshi and Kwon, [52], which we now sketch. Let \(Y_1, Y_2, \ldots, Y_p\) denote market implied volatilities. We want to determine \(\partial g/\partial Y_i\) using the already computed model vegas. That is, we need to find a linear combination of model vegas so that only one market volatility increases by 1% and all other volatilities remain unchanged. Since these market volatilities are smooth functions of their model counterparts\(^2\), we can compute their partial derivatives

\[
y_i^T = \left( \frac{\partial Y_i}{\partial \psi_1}, \frac{\partial Y_i}{\partial \psi_2}, \ldots, \frac{\partial Y_i}{\partial \psi_r} \right),
\]

\(^2\)This is true for caplet volatilities and coterminal swaption volatilities when using the Hull and White swaption approximation in Section 2.4.2.
Table 6.4: Deltas, vegas and skew sensitivities for a cancellable inverse floater swap, strike = 10%. A 51x51 spatial grid was used for the finite difference (FD) and adjoint Greeks.

for a particular ordering of the r model volatility terms. To find the linear combination of model vegas, or equivalently the r-dimensional weights vector \( w_i \), we require

\[
y_i^T w_i = 1%,
\]

so that the inner product with the weights vector increases the market implied volatility by 1% and

\[
y_j^T w_i = 0, \quad (6.11)
\]

for \( j \neq i \) so that all other implied volatilities do not change. Concentrating on a two-factor model, we have r model vegas, but we are using \( 2n - 1 \) instruments to hedge (as the last caplet and coterminus swaption coincide). Since we would most likely have \( r > 2n - 1 \) we require an extra condition on the weights vector to uniquely determine it. We choose to impose

\[
w_i^T w_i \rightarrow \min,
\]
so that the minimal perturbation is found. Using these three conditions we can
determine the $2n - 1$ weight vectors as $w_i$ lies in the linear span of the projection of
$y_i$ onto the subspace orthogonal to (6.11) above, see [52]. The market vega is then
given by

$$\frac{\partial g}{\partial Y_i} = \sum_{j=i}^{r} w_{i,j} \frac{\partial g}{\partial \psi_j}.$$}

The routine will fail if the gradient $y_i$ is a linear combination of the other gradients
$y_j$ for $j \neq i$. If this happens then the product with implied volatility $Y_i$ would be
removed from the hedging portfolio.

We initially consider hedging a product using only the $2n$ mapping vegas as
these terms can be computed quicker than the Markov vegas. Figure 6.1 shows the
sensitivity of the cancellable inverse floater in Section 6.3 with respect to caplet
volatilities only. That is, we do not include coterminal swaptions in the hedging
portfolio. The horizontal axis represents caplets expiring at time $T_{k-1}$ and the
vertical axis displays the size of the vega. We notice that the vegas of the early
caplets are negative and then become positive. This is because the caplets are
trying to hedge both the put options on the LIBOR rate and the cancellability of
the swap.

![Caplet Vegas graph](image)

**Figure 6.1:** Sensitivity of the cancellable inverse floater swap with respect to caplet volatil-
ities (scaled by a factor of 100).

Figure 6.2 shows the vegas when only coterminal swaptions are considered, where
the horizontal axis represents swaptions expiring at time $T_{k-1}$ with maturity $T_n$.
Similarly with the caplets we see that the earlier swaptions account for the put optionality feature, while the later ones account for the cancellability. For this particular market setup, which effectively results in a one factor model, we could not include both caplets and coterminal swaptions in the hedging portfolio, because the caplets become redundant when the coterminal swaptions are included.

Figure 6.2: Sensitivity of the cancellable inverse floater swap with respect to coterminal swaption volatilities (scaled by a factor of 100).

For a more realistic market scenario we calibrate our two-factor Markov-functional model to a five-factor LMM, using the technique introduced in Section 5.2. The $i$th caplet volatility in the LMM is given by the $abcd$ volatility form with $a = -0.02, b = 0.3, c = 2, d = 0.2$. The correlation between rates is given by the $L\beta$ form with $L = 0.5$ and $\beta = 0.2$. The model parameters are given in Table 6.5 and the price is computed to be 2835 bps. We use the adjoint method to compute the mapping vegas and convert these into caplet and coterminal swaption vegas. The hedging portfolio consists of 20 caplets and 10 swaptions, using every second one, $\text{Swpn}(T_0, T_{20}), \text{Swpn}(T_2, T_{20}), \ldots, \text{Swpn}(T_{18}, T_{20})$.

Figure 6.3 shows both the caplet and coterminal swaption vegas. As we expect the caplets are hedging the put option feature while the swaptions hedge the cancellability.

We now consider the impact of the Markov vegas by including them into (6.10) along with the mapping vegas. We choose to include a parallel shift to both diagonal elements, $c_{11}$ and $c_{22}$ as well as a parallel shift to the off diagonal elements $c_{12}$ and
Table 6.5: Market volatilities used in Figure 6.3

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
f_i & \nu_0 & \nu_1 & [T_0, T_1] & c_{11} & c_{21} & c_{12} & c_{22} \\
\hline
f_0 & 0.14 & -0.33 & [0, T_0] & 0.79 & -0.13 & 0.44 & 0.52 \\
\hline
f_1 & 0.13 & -0.27 & [T_0, T_1] & 0.80 & -0.13 & 0.42 & 0.56 \\
\hline
f_2 & 0.13 & -0.25 & [T_1, T_2] & 0.81 & -0.13 & 0.41 & 0.59 \\
\hline
f_3 & 0.14 & -0.25 & [T_2, T_3] & 0.82 & -0.13 & 0.40 & 0.62 \\
\hline
f_4 & 0.14 & -0.24 & [T_3, T_4] & 0.84 & -0.12 & 0.37 & 0.65 \\
\hline
f_5 & 0.14 & -0.22 & [T_4, T_5] & 0.86 & -0.11 & 0.34 & 0.69 \\
\hline
f_6 & 0.15 & -0.20 & [T_5, T_6] & 0.88 & -0.10 & 0.31 & 0.73 \\
\hline
f_7 & 0.15 & -0.19 & [T_6, T_7] & 0.90 & -0.10 & 0.27 & 0.77 \\
\hline
f_8 & 0.16 & -0.18 & [T_7, T_8] & 0.92 & -0.08 & 0.22 & 0.83 \\
\hline
f_9 & 0.16 & -0.15 & [T_8, T_9] & 0.94 & -0.06 & 0.16 & 0.90 \\
\hline
f_{10} & 0.16 & -0.12 & [T_9, T_{10}] & 0.97 & -0.04 & 0.08 & 0.96 \\
\hline
f_{11} & 0.17 & -0.11 & [T_{10}, T_{11}] & 1.00 & 0.00 & 0.00 & 1.00 \\
\hline
f_{12} & 0.21 & -0.13 & [T_{11}, T_{12}] & 1.02 & 0.09 & -0.10 & 1.01 \\
\hline
f_{13} & 0.20 & -0.10 & [T_{12}, T_{13}] & 1.05 & 0.18 & -0.27 & 1.07 \\
\hline
f_{14} & 0.20 & -0.05 & [T_{13}, T_{14}] & 1.05 & 0.31 & -0.50 & 1.14 \\
\hline
f_{15} & 0.20 & 0.01 & [T_{14}, T_{15}] & 1.00 & 0.50 & -0.75 & 1.21 \\
\hline
f_{16} & 0.18 & 0.08 & [T_{15}, T_{16}] & 0.85 & 0.77 & -1.04 & 1.30 \\
\hline
f_{17} & 0.16 & 0.12 & [T_{16}, T_{17}] & 0.43 & 1.26 & -1.53 & 1.58 \\
\hline
f_{18} & 0.14 & 0.14 & [T_{17}, T_{18}] & -3.02 & 4.49 & 5.56 & -5.20 \\
\hline
f_{19} & 0.13 & 0.14 & [T_{18}, T_{19}] & 1.76 & 0.00 & 0.00 & 1.00 \\
\hline
\end{array}
\]

$c_{21}$, resulting in $2n$ extra vegas. These sensitivities were computed using the adjoint approach in Section 6.1.5.

Figure 6.4 is the same as Figure 6.3, however we see the changes when the Markov vegas are included. The main difference between the two hedging portfolios is the absolute value of the vegas. When the Markov vegas are included the absolute value of both the caplet and swaption vegas decrease, which is particularly evident for the earlier swaptions. Small absolute vegas are preferred in practice, because a lower bid/ask spread is then paid on the hedging portfolio. This means that including Markov vegas is beneficial, however they take time to compute, so a balance between speed and cost is required for a market implementation.

We also consider another cancellable inverse floater swap, which is identical to the previous one, but with a strike of 18%. Its value is computed to be 334.6 bps which indicates that it is likely to be cancelled earlier than the other inverse floater swap. Figure 6.5 displays the caplet and coterminal swaption vegas in the same way as Figure 6.4. Here we see that the products with the shortest expiries have the largest vegas, because of the likelihood of the product being cancelled early. Also of note is the difference when the Markov vegas are included, which increases the earlier caplet vegas for a reduction in the earlier swaption vegas, however the later caplet and swaption vegas show only minor changes.
Figure 6.3: Sensitivity of the cancellable inverse floater swap with respect to caplet and coterminal swaption volatilities (scaled by a factor of 100).

Figure 6.6 presents the vegas for the Bermudan swaption (whose value is computed to be 206.6 bps) with the mapping vegas used to create the graph shown in Table 6.6. Here we see that analogously with the inverse floater, the absolute values of the swaption vegas decrease when the Markov vegas are included. The absolute value of the caplet vegas are similar with and without the Markov vegas. In any case, it is likely that in a practical setting this product would be hedged using only coterminal swaptions.

6.4.2 Market skew

We have so far demonstrated how to hedge our exotic interest rate option to ATM volatilities. Since we are using a displaced-diffusion model we would also like to hedge the skew risk, using our computed sensitivities

\[
\frac{\partial g}{\partial \alpha}.
\]

Brace (Section 13.1 of [16]) demonstrates how to hedge the skew risk using two swaptions with the same expiry and different strikes for the LMM. This approach can be carried over directly to the separable LIBOR Markov-functional model, so we do not present the details.

Another slightly different approach is due to Joshi and Yang, [54], which focuses on the swap rate market model. This technique can also be applied to the separable
Figure 6.4: Sensitivity of the cancellable inverse floater swap (with strike = 10%) with respect to caplet and coterminal swaption volatilities, showing the difference caused by including the Markov vegas (scaled by a factor of 100).

Markov-functional model using similar ideas to Brace.

It is important to hedge the skew risk because it has been shown to have an impact on prices, see Pietersz and Pelsser, [69], for the particular case of Bermudan swaptions.
Figure 6.5: Sensitivity of the cancellable inverse floater swap (with strike = 18\%) with respect to caplet and coterminal swaption volatilities (scaled by a factor of 100).

Table 6.6: Deltas, vegas and skew sensitivities for a callable Bermudan swaption, strike = 7\%. A 101×101 spatial grid was used.
Figure 6.6: Sensitivity of the callable Bermudan swaption with respect to caplet and coterminal swaption volatilities, showing the difference of including the Markov vegas (scaled by a factor of 100).
Conclusion

The three main objects of study in this thesis are: (1) fast and accurate Greeks for the displaced-diffusion LIBOR market model; (2) a Markov-functional model as a control variate; and (3) adjoint PDE Greeks for Markov-functional models.

In Chapter 3 we extended the pathwise adjoint method to the displaced-diffusion LIBOR market model. In this framework we achieved speed improvements of 20 percent by using our new delta and vega expressions, which are derived with the log forward rates as state variables. With these efficiency gains we also introduced increased accuracy by extending the method to the predictor-corrector scheme. We demonstrated the improvements over the log-Euler approximation, which would lead to financial organisations more accurately calculating their risk positions.

Chapter 4 introduced the separable LIBOR Markov-functional model and presented a novel formulation in the spot measure. Contrary to the LMM, we found that the Markov-functional model often performed better in the terminal measure, particularly for PDE implementations. The main reason for this is that a numerical PDE method smooths large cash flows, reducing their impact.

A Markov-functional model was presented as a control variate for the LMM in Chapter 5. This is very effective at reducing the standard error of prices and vegas. The main reason for its effectiveness is a new accurate and analytic calibration method, which allows for small perturbations in the LMM volatility structure to be translated into small perturbations in the Markov-functional model’s structure. The standard error reductions are significant as they dramatically reduce the number of required simulations for a desired level of accuracy. This can greatly improve the speed of risk management systems for a large portfolio of trades.

Chapter 6 demonstrated how to use the adjoint PDE method to compute Greeks in Markov-functional models. We showed that it is both accurate and efficient. To compute the Greeks we need to solve an adjoint equation that is of similar complexity to the pricing equation. From this solution we are able to obtain all of the model sensitivities. This means that almost all Greeks can be computed in the same time as a single Greek using finite difference. The Greeks were then shown to be accurate by comparing results with the Black formula. We also demonstrated how the model
vegas can be converted into market vegas, providing evidence that the adjoint PDE method is useful for risk managing a book of interest rate derivatives.

We conclude with a statement regarding efficiency and accuracy. Separate divisions of a financial organisation require different features from their models. For example, a Front Office environment will require very accurate prices and Greeks, because they are offering products to the market and need to hedge any sold products. Market Risk, however, require more efficiency and can accept less accuracy, because they need to calculate the sensitivity of a large portfolio of trades to many different risk scenarios. It is therefore most likely that the predictor-corrector path-wise method will be more attractive to the Front Office, because of its accuracy improvements over the log-Euler method.

The control variate technique is appealing to both Front Office and Market Risk because it provides efficiency gains without a loss of accuracy. The adjoint PDE Greeks are at least as accurate as those produced by a finite-difference method, but the Markov-functional model is less accurate than the LMM. It is likely that the model will be attractive to the Front Office for shorter dated products (where it is most accurate), while the inaccuracy for longer dated products will be outweighed by its efficiency for a Market Risk environment.
Bibliography


[65] Ng, L. Local calibration of separable BGM models and finite difference pricing of LIBOR and CMS spread based exotics. *Wilmott* 1(2) (2009), 109–120.


Author/s:
Denson, Nicholas Andrew

Title:
Adjoint and PDE methods for pricing and risk management of exotic interest rate derivatives

Date:
2010

Citation:

Persistent Link:
http://hdl.handle.net/11343/35745

File Description:
Adjoint and PDE methods for pricing and risk management of exotic interest rate derivatives

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.