Non-Recombining Trees for Interest Rate Derivatives
A numerical investigation

Rory Wallace
Dissertation for M.Sc. Financial Mathematics 2003-2004
University of Warwick

(In association with Misys Risk Management Systems, London.)

August 28, 2006

Acknowledgements

I want to thank my supervisors, Stewart Hodges and Sebastian Van Strien of the University of Warwick, and Odile Hounkpatin of Misys Risk Management Systems, for their help and guidance through this project. I would also like to thank Craig Mounfield, formerly of Misys, who originally suggested the topic.

Abstract

I demonstrate how a non-recombining tree method can be used to value interest rate derivatives using the LIBOR Market Model framework. My approach is based on that of Peter Jäckel [Jäckel2000A]. I analyse its accuracy and practicality for pricing several standard derivative contracts, and compare it to alternative approaches such as Monte Carlo simulation. The method is shown to be relatively simple to implement, and practical for contracts with a limited number of exercise opportunities. It can have significant advantages over Monte Carlo simulation methods, but several potential problems are identified. I illustrate these problems and suggest possible solutions. Finally I attempt to extend the approach to model contracts with a large number of exercise opportunities by using approximation techniques.

Contents

Chapter 1

Introduction

The aim of this project is to investigate the use of the non-recombining tree method for valuing interest rate derivatives.

Interest rate modelling is a rich field. The "Market model" framework is popular, particularly the 'LIBOR Market Model' (also known as the BGM model, after the first authors to demonstrate it theoretically [BGM97]). This model is consistent with accepted market pricing formulae for standard instruments such as caps. It is also easy to explain its dynamics using martingale pricing theory.

The difficulty with BGM is that prices for many instruments cannot be found analytically. They must be estimated by simulation. An additional problem occurs when valuing instruments where the holder has a choice of exercise dates (known as 'American' or 'Bermudan' options). The value of such an option is its cost when the holder exercises it optimally. However the 'best' exercise strategy is difficult to find. Using a tree approach is a straightforward way to estimate it.

Recombining trees such as the binomial tree are very common, and were introduced by Sharpe and by Cox,Ross and Rubenstein in the 1970's - see [CRR79]. They are easy to understand and implement. A binomial tree is illustrated in Figure 3.1.

Recombining trees cannot easily be used for the BGM model. One reason for this is that interest rate volatilities are time-varying, so that constant

time steps would mean that the nodes of the tree would not meet. A more important reason is that the drifts of the underlying stochastic process are state dependent. This means that an 'up' move followed by a 'down' move, from a particular state, will not generally be the same as a 'down' followed by an 'up'. Various authors attempt to get around this problem by using variations on the tree such as a stochastic mesh method, or by varying the transition probabilities to force the nodes to recombine. However these methods can be complex, and are difficult to apply in multiple dimensions. More details follow in the literature review.

The most straightforward tree model for processes with state-dependent drifts is a 'non-recombining' or 'bushy' tree. Once the equations giving the drift and diffusion terms at each node are specified, the branching structure can be found relatively easily. Values of options with early exercise features can easily be found by moving backwards along the tree, and comparing the continuation value at each exercise decision with the exercise value. No interpolation of the exercise values, or parameterisation of the exercise strategy is required.

The main problem with using non-recombining trees is the 'curse of dimensionality'. As is shown in Chapter 6, using a large number of timesteps causes the number of nodes to grow exponentially. This rapidly causes computation times to become impractical. The three main ways to remedy this are

- Reduce the computation needed at each node
- Reduce the number of time steps
- Reduce the number of branches.

In following sections I will examine methods to do each of these.

There are other small problems with accuracy. These happen because we are attempting to sample a random underlying process in a structured way. If our choice of nodes happens not to sample the random process in an even way, our tree price may not converge to the expected value. This problem is illustrated in Chapter 7.

However the big advantage of using non-recombining trees is that it is a 'dumb' method. This means that once our tree configuration is decided, a given derivative contract can be 'plugged in' to the tree and valued without intervention, for example to decide an exercise strategy. This is in contrast to American Monte Carlo methods where a 'sensible' exercise strategy must be known, either calculated within the simulation, or externally specified.

Finally I will examine methods to obtain good estimates using a restricted number of time steps.

Chapter 2

Review of the LIBOR market model

In this section I summarise the LIBOR Market Model and introduce some terminology. This is required here as the whole of the approach I use relies on this model. A more rigorous and detailed account can be found in, for example, [HK2000] or [BM2001].

LIBOR stands for "London Inter-Bank Offer Rate"and corresponds to an almost-risk-free rate of interest paid between large banks on the London market.

The LIBOR (Forward) Market Model ("LFM") models the price processes of the traded assets in an economy. The economy is the riskless interest-rate and bond market. The traded assets are a finite number of discount bonds, each giving a payment on a single date. From these bonds we can derive traded interest rates or "Forward Rates".

A "Discount bond" or "Zero-coupon bond" is a fixed payment of 1 to be made at a future time T. $D(t,T)$ denotes the value at time t of this payment. The time T is also known as the "maturity date" of the bond.

The LIBOR rate $L_t[T,S]$ is the interest rate applicable at time t for lending money between dates T and S. Specifically, a standard unit deposit made at time T pays at maturity S an amount $1+\alpha[T,S]L_t[T,S]$. Note that this LIBOR rate is fixed at time T. Time T is known as the 'setting date'

of the rate $L[T,S]$. An agreement to lend money at LIBOR over a future period is known as a "Forward Rate Agreement" ("FRA"). Since these FRA's are traded we can observe $L_t[T,S]$ directly. $L_t[T,S]$ can also be expressed in terms of discount bonds maturing at times T and S via the relationship $L_t[T,S] = \frac{D_{TT}-D_{TS}}{\alpha_{[T,S]}D_{TS}}$.

The "Year Fraction" $\alpha_{[T,S]}$ between dates T and S denotes the conversion factor to be used when calculating the actual amount of interest to be paid for a given rate and time period. Generally we will work in years, and set the year fraction to be the actual fraction of a year between the two dates. We will not be concerned with other conventions for calculating year fractions although they can be important in practice. Where the intervals between rate reset dates are all equal, we can take α to be a constant. We denote the payment fraction corresponding to rate i by α_i.

We consider a fixed set of dates T_i, i=0..n, which span the period we are modelling, i.e. $T_0 = 0, T_n = T_{max}$. We denote $f_i(t)$ to be the "I'th Forward LIBOR at time t". This is equivalent to $L_t(T_i - 1, T_i)$. We see that f_i is a ratio of traded assets (discount bonds) divided by a particular numeraire (the discount bond payable at time T_i).

Now we have a vector $\mathbf{f} \equiv \{f_i\}$ which can be used to express the value of linear combinations of discount bonds. We assume that the market is complete, i.e. that this vector spans the economy. This will be the case since we are assuming that all asset price paths are continuous, and we are only concerned with payments occurring on our set of reference dates.

We now suppose that the distribution of each forward rate is log-normal, under its own measure. This means that, for some measure W_i,

$$df_i = \sigma_i(t) f_i dW_i \tag{2.1}$$

This is consistent with the Market's use of Black's formula to price caplets. Note that after a rate's setting date, it's volatility by definition becomes zero.

We assume that the Wiener processes $\{W_i\}, 1 <= i <= n$ are correlated, and that their correlation matrix is a function of time only. We denote this

correlation matrix by $\rho(t)$, where

$$dW_i(t)dW_j(t) = \rho(t)_{ij}dt \tag{2.2}$$

We now pick one particular asset to use as a numeraire. For now, we will choose the discount bond paying at time T_n. The martingale pricing principle is that $\frac{V_t}{N_t} = E^{\mathbb{N}}[\frac{V_T}{N_T}|\mathfrak{F}_t]$. So the forward rate corresponding to our numeraire bond has zero drift. If we have payments arising at different times, i.e. other forward rates, we cannot use the same pricing measure directly. Girsanov's theorem tells us that since all the measures are equivalent, we can find an $\mathfrak{F}_t$-predictable process such that $dW_j = dW_i - c(t)dt$.

It can be shown, for example in [BM2001, Section 6.3], that the calculation of the drift adjustment between the natural measure W_k for a rate, and the equivalent martingale measure W_i, is given by

$$df_k(t) = -\sigma_k(t)f_k(t)\sum_{k+1}^{i}\frac{\rho_{kj}\sigma_j(t)f_j(t)}{1+\alpha_j f_j(t)}dt + \sigma_k(t)f_k(t)dW_k(t) \tag{2.3}$$

We may choose to work under an equivalent martingale measure for any of our n fundamental traded assets, and the drift adjustment will take a similar form.

Note that the drift adjustment is state-dependent. This means that a closed-form solution for the joint distribution of the forward rates under any given measure cannot be found, so we must proceed by simulation.

(Note: The Market Model approach can also be applied by choosing the fundamental assets to be a set of interest rate swaps spanning the period in question. It is then known as the LIBOR Swap Market Model ("LSM"). This model can be useful for pricing swaps and swaptions. Conceptually it is similar to the LFM, but the calculations are more complicated. The LFM has the advantage of being easy to calibrate to caplets, and easier to simulate; It is the approach currently used by the industrial sponsor. I will not pursue the LSM approach further here.)

Chapter 3

Fundamentals of the Bushy Tree approach

3.1 Use of a tree

Before describing the non-recombining tree, I will briefly recap the idea of a tree, as used by [CRR79].

The tree shown in **??** models the evolution of a stock price S by allowing S to take discrete values on the nodes. The initial value of the stock is taken as the starting point of the tree. A range of time steps are chosen over the period of interest. The branching sizes and branch probabilities are chosen to match the mean and variance of the stock price over that period which are assumed known. Now since the discretised process on the tree has known branching structure, it is easy to traverse the tree and find the conditional expected value of any given future payoff. Expected values at the starting node, and optimal exercise strategies for American options, are simply found by backward recursion.

3.2 Non-Recombining trees

In chapter 2 we described the underlying stochastic differential equation. The bushy tree approach attempts to sample the space of all possible evolutions

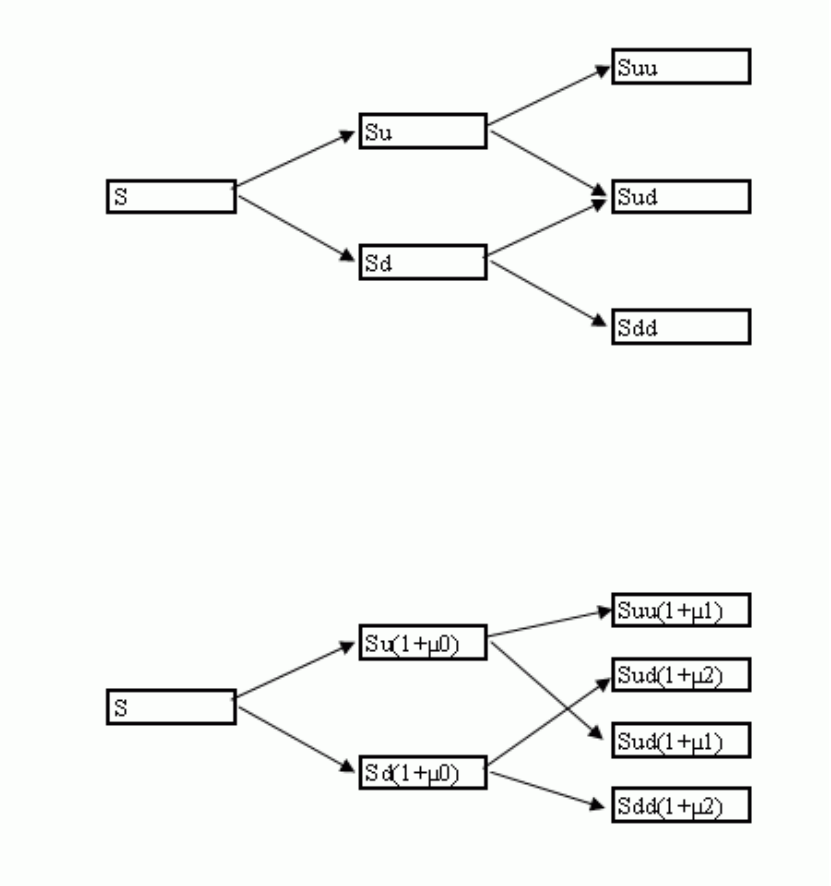

Figure 3.1: Example of bushy and non-bushy 2-branch trees

of this system in an even way.

Suppose we have a 2-dimensional system of uncorrelated Brownian motions, $\mathbf{W} = W_1, W_2$. We can simulate this on a tree over a time step $t, t + \Delta t$ by taking $\Delta W_1 = \pm\sqrt{\Delta t}$, $\Delta W_2 = \pm\sqrt{\Delta t}$. Figure 3.2 shows such a tree. It uses unequal step sizes to illustrate the fact that our tree will not generally recombine.

Applying this to the LIBOR market model uses exactly the same ideas; we step the underlying independent d-dimensional Brownian process; use this to obtain a correlated set of n increments for the process $\mathbf{W}$; use this $\{\Delta W_i\}$ to calculate an approximated evolution of $\mathbf{f}$ using a discretised version of 2.3.

The general case of the LFM has n correlated driving Brownian motions. (We could orthogonalize these to get up to n uncorrelated factors.) If we

keep all these factors and use the simple approach of bumping each factor up and down equally, we would need $2n$ branches at each node. As the LFM can have dimension up to 80 (quarterly interest rates for 20 years), we need to look at reducing the number of branches required.

This could be done by

- Reducing the number of factors used
- Reducing the number of branches per factor
- Combining branches

The easiest of these is to reduce the number of branches per factor. In 5.1 I will discuss this but figure 3.3 shows a simple example where 3 branches suffice for a 2-factor model.

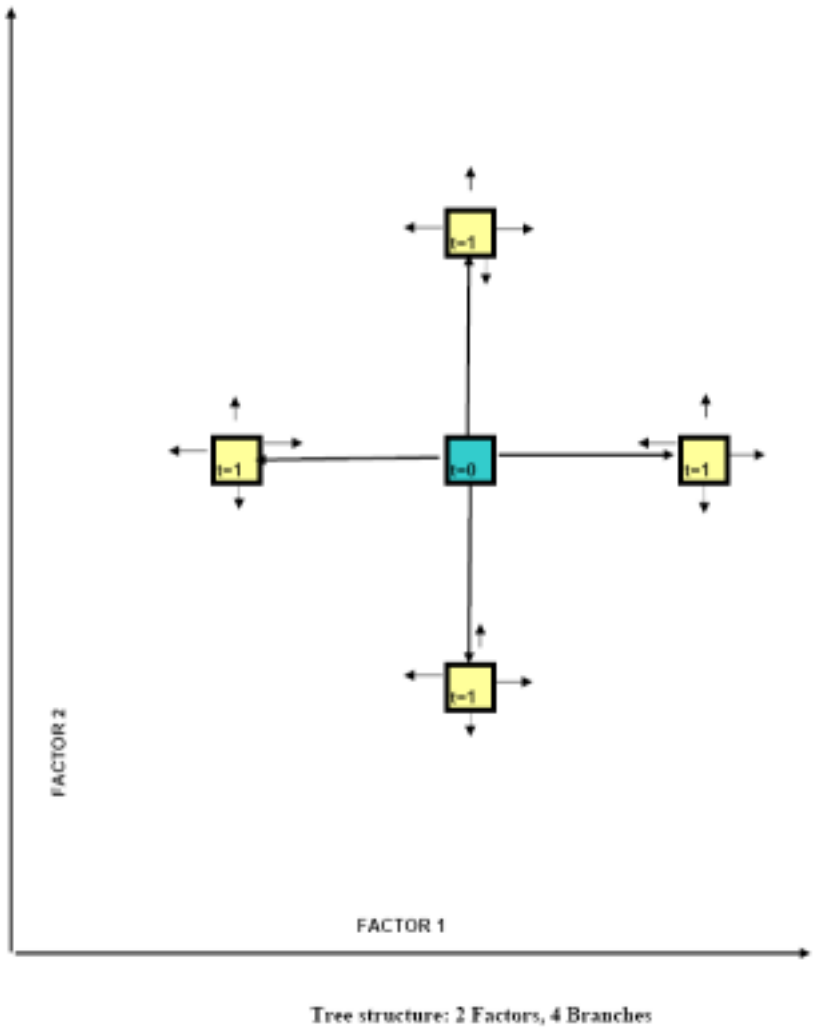

Figure 3.2: Example of 2-factor tree using 4 branches

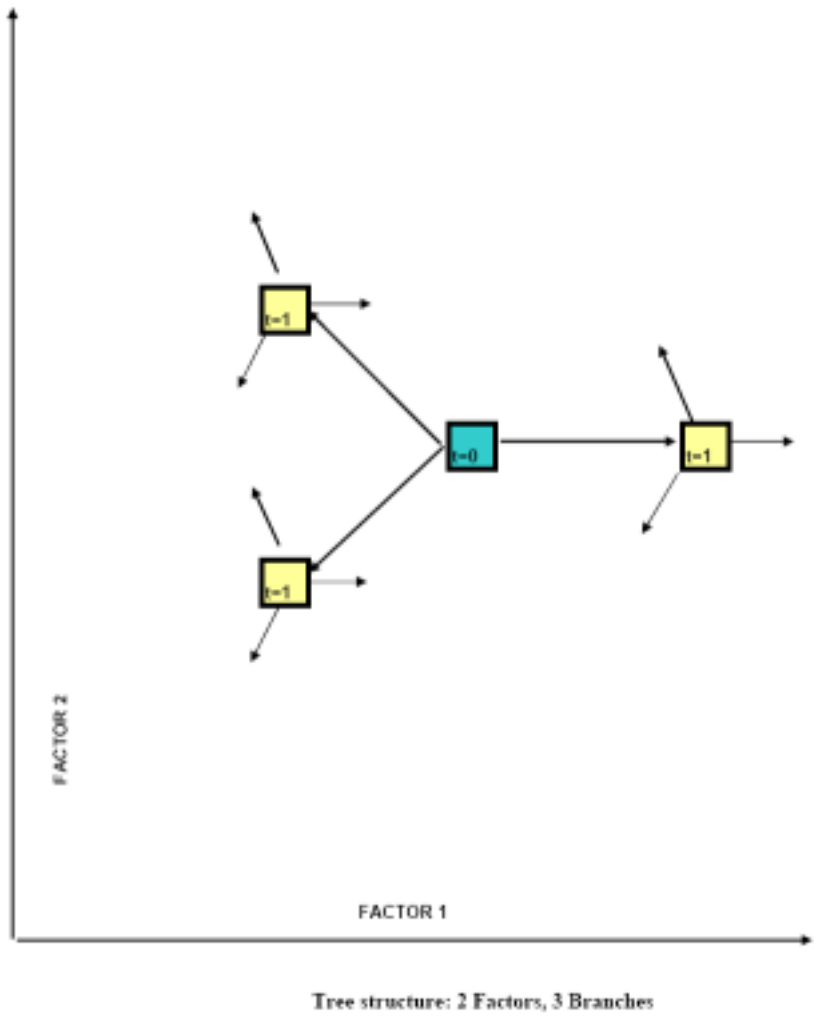

Figure 3.3: Example of 2-factor tree using 3 branches

Chapter 4

Literature review

4.1 Non-recombining tree methods

There are a small number of recent papers which discuss the use of a non-recombining tree with the LIBOR market model.

Tang & Lange [TL2001] propose a "non-exploding" bushy tree technique. Their procedure is similar to that of a regular bushy tree. However at each branching step, they discard a selection of new nodes. Their choice of which nodes to keep is done by matching the moments of an underlying state variable across the remaining nodes. The choice of state variable is related to the product being modelled, so for modelling an interest rate cap, for example, matching would be done on the short interest rate.

Their approach is complex in that it requires an interpolation to be done at each step. The pruning procedure is not entirely clear from their paper. The main benefit is that because not all branches are used, their tree is much faster. However the interpolation might need to be tweaked for each product.

McCarthy & Webber [McCW2002] use a non-recombining icosahedral lattice and use 2 interest rate models, one 3-factor HJM, and another is an affine model. They claim that their lattice method is significantly better than standard square branching methods.

Jäckel [Jäckel2000A] uses a similar approach to [McCW2002] in that he uses a more efficient branching structure. He explicitly shows how his ap-

proach can be used for an arbitrary number of factors, and suggests various accuracy improvements.

Based on discussions with my industrial supervisor, it was decided to go with Jäckel's approach, because it seemed clear how to implement it, and it had the flexibility to be adapted to various products and tree structures without changing the underlying method.

4.2 Other methods

There are a large variety of other methods that can be used for pricing in the market models. The standard method is Monte Carlo simulation which is straightforward to use for European products (i.e. those not involving a choice of exercise strategy). For American options, the optimal exercise strategy must also be chosen, making things more complicated. For a review of common methods, see, for example, [Gla2000, Chapter 8]. However the aim of this project was to look at the non-recombining tree approach so I will not discuss these much.

Chapter 5

Using the non-recombining tree

5.1 Branching method

Once the number of branches and factors for a tree has been decided, the tree can be constructed.

Clearly in a 1-factor model we need at least 2 branches. For a d-factor model we can take products of the branching structure on each factor to use 2^d branches. However this can be improved.

First consider how to branch a d-dimensional uncorrelated Brownian process. If we think of our branching coordinates for each factor as points in $\mathbb{R}^d$, and we take m points, then we want for each dimension,

$$\sum_{i=1}^{m} x_{i(.)} = 0 \tag{5.1}$$

and also

$$\sum_{i=1}^{m} (x_{i(.)})^2 = m \tag{5.2}$$

We also would like the points to be evenly distributed. We can choose $(d+1)$ points lying on the surface of a d-dimensional sphere by taking the vertices of a hypercube. For example in $\mathbb{R}^2$ we have an equilateral triangle, in $\mathbb{R}^3$ a tetrahedron, etc.

Jäckel refers to this selection of points as a "perfect simplex". Its coordi-

nates are given by

$$s_{ij}^{(d)} = -\sqrt{\frac{d+1}{(j+1)j}} \text{ for } j \geq i \tag{5.3}$$

$$s_{ij}^{(d)} = \sqrt{\frac{d+1}{(j+1)j}} \text{ for } j = i-1 \tag{5.4}$$

$$s_{ij}^{(d)} = 0 \text{ for } j < i-1 \tag{5.5}$$

For example, one perfect simplex in $\mathbb{R}^3$ is given by

$$\begin{pmatrix} -1.414 & -0.816 & -0.577 \\ 1.414 & -0.816 & -0.577 \\ 0 & 1.633 & -0.577 \\ 0 & 0 & 1.732 \end{pmatrix} \tag{5.6}$$

Clearly this could be better dispersed as 3 of the 4 z-coordinates of this simplex are identical.

5.1.1 Using extra branches

Where f factors are used, a minimum of $f+1$ branches must be used. Using b branches (where $b > f+1$) will improve precision. In this case, an optimal simplex S' of dimension $b-1$ should be found. The extra columns (from $f+1$ to b) of S' can be ignored when branching.

5.2 Optimal simplex alignment

Jäckel defines an "Optimal alignment" S' for a simplex S as the matrix S' which:

- Is found by a series of plane rotations of S
- Has

$$\sum_{1}^{[\frac{m+1}{2}]} (S'_{i,j} + S'_{m+2-i,j}) = 0 \text{ for } 1 \leq j \leq [\frac{m+1}{2}] \tag{5.7}$$

I don't think there is a closed form to find this optimal rotation. I have obtained it it by:

- Picking a random plane to rotate around.
- Rotating around this plane until the sum of squared deviations (over all dimensions) from 5.7 is minimized, using a one-dimensional minimization on the rotation angle (Brent's method).
- Repeat until convergence.

It is necessary to randomize the order of rotation to avoid convergence to local minima. For all dimensions up to twenty or so, convergence takes place within a few seconds and satisfies 5.7. It is unlikely in practice that more than this number of branches would be required.

For example, one optimal alignment found for $\mathbb{R}^3$ is given by

$$\begin{pmatrix} -0.899 & 1.092 & 1 \\ -1.092 & -0.899 & -1 \\ 1.092 & 0.899 & -1 \\ 0.899 & -1.092 & 1 \end{pmatrix} \tag{5.8}$$

The advantages of optimal alignment will be shown in section **??**, but essentially it is based on the idea of having an even spread on each factor.

5.3 Discretising the BGM equation

The BGM Equation (2.3) can be written as

$$df_k(t) = \mu(\mathbf{f}(t), t)dt + \sigma_k(t)f_k(t)dW_k(t) \tag{5.9}$$

Recalling that the drift term μ is state-dependent, and that we have chosen a numeraire.

Once the underlying process $\mathbf{W}$ has been discretised, the increments of $\mathbf{f}$ can be found. Some possible methods are:

5.3.1 (Forward) Euler method

This simply uses

$$\hat{f}_k(t+\Delta t) = \hat{f}_k(t) + \mu(\hat{\mathbf{f}}(t))\Delta t + \sigma_k \hat{f}_k(t)\Delta W_k \tag{5.10}$$

5.3.2 log-Euler method

This uses

$$\hat{f}_k(t+\Delta t) = \hat{f}_k(t)\exp([\mu(\hat{\mathbf{f}}(t)) - \frac{1}{2}\sigma_k^2]\Delta t + \sigma_k \hat{f}_k(t)\Delta W_k) \tag{5.11}$$

which is equivalent to applying an Euler scheme to log f_k. It has the advantage that f_k cannot become negative, and keeps f closer to being lognormal.

5.3.3 Pseudo-predictor-corrector method (Hybrid method)

Both previous methods estimate $\mu(\mathbf{f})$ over the interval $(t, t+\Delta t)$ based on its value at t. Since μ is state-dependent this approximation entails a small bias. For small step sizes this is commonly ignored. However [HJJ2001] propose an improved method. It is a 'hybrid' method which estimates the drift $\hat{\mu}(t)$ using 2.3, uses this to estimate the mean level of $\mathbf{f}(t+\Delta t)$, recalculates an estimated drift $\hat{\mu}(t+\Delta t)$, and finally uses the average $\tilde{\mu}(t, t+\Delta t) = \frac{1}{2}(\hat{\mu}(t)+\hat{\mu}(t+\Delta t))$ to recalculate $\mathbf{f}(t+\Delta t)$.

Since the drift calculations, which is the most time-consuming part of the discretization, are done twice, this method takes up to twice as long as the previous two. However [HJJ2001] shows that the reduction in bias provided by using this method ensure that it significantly outperforms standard Forward Euler methods. This is particularly noticeable over wider time steps. Since the non-recombining tree method will require a limited number of time steps (hence wider ones), this method is used in my simulations.

5.3.4 Terminal Decorrelation

In discretising (2.3), the correlation $\rho(t)(i,j)$ is involved. Rebonato ([Reb2002, Ch.6]) suggest that instead of using the instantaneous correlation at t, it is more accurate to substitute the 'terminal correlation' over $(t, t+\Delta t)$.

This only makes a difference when the individual volatility of forward rates is rapidly time-varying. In my simulations I did not find it made a significant difference, compared to other sources of error, so for simplicity I left it out.

5.4 Pricing using the tree

Remembering that we are using the martingale valuation method, and the law of iterated expectation, we obtain prices at each node of the tree by:

- Branching the forward rates onto each of the child nodes
- Calculating the expectation at each of the child nodes (in terms of the numeraire)
- Take the average over all direct children to get the expected value at that node
- If exercise is possible, comparing this value to the intrinsic exercise value

This gives a value at the top of the tree (which is in terms of our chosen numeraire) which then is multiplied by the initial numeraire value to give a cash amount.

Chapter 6

Tree performance

The running speed of a pricing algorithm is critical if it is to be used in practice. If prices are to be calculated for a market trading environment, they should be available within a few seconds. If they are needed for specialized (e.g. Over-The-Counter) products, or if they are required for risk management purposes, longer calculation times may be acceptable but should still be calculated within a few hours at the most.

For a Monte-Carlo type simulation, we know that the standard error of estimates decrease as $\frac{1}{\sqrt{N}}$. So the time required is dependent on the level of accuracy desired.

However the non-recombining tree method generally gives only a single number as a price estimate. The precision of this number may depend on how coarsely we have discretised the system. In the non-recombining tree case, this effectively means the number of branches at each node, and the number of time-steps at which we branch.

6.1 Typical running times

Before investigating how many nodes and branches are required, I will examine how many may be practical.

Figure 6.1 details estimated running times, in seconds, to price one option on a non-recombining tree.

The times here are based on 100,000 terminal nodes being calculated per second. This approximate speed was observed for a wide variety of tree sizes on the author's P.C.: (A mid-range AMD Athlon running at 2.0GHz). In practice, actual speeds will vary slightly depending on the exact tree and product setup. However the rapid growth in calculation times as the number of factors and branches are increased will remain.

Number of time steps \ Number of branches	2	3	4	5	6	7	8	9	10
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01
4	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.07	0.1
5	0.00	0.00	0.01	0.03	0.08	0.17	0.3	0.6	1.0
6	0.00	0.01	0.04	0.16	0.5	1.2	2.6	5.3	10.0
7	0.00	0.02	0.16	0.8	2.8	8.2	21.0	47.8	1.7 min
8	0.00	0.07	0.7	3.9	16.8	57.6	2.8 min	7.2 min	17 min
9	0.01	0.20	2.6	19.5	1.7 min	6.7 min	22 min	1.1 hr	2.8 hr
10	0.01	0.59	10.5	1.6 min	10 min	47 min	3.0 hr	10 hr	28 hr
12	0.04	5.3	2.8 min	41 min	6.0 hr	38 hr	*	*	*
15	0.3	2.4 min	3.0 hr	85 hr	*	*	*	*	*
18	2.6	1.1 hr	191 hr	*	*	*	*	*	*
20	10.5	9.7 hr	*	*	*	*	*	*	*
25	5.6 min	*	*	*	*	*	*	*	*
30	3.0 hr	*	*	*	*	*	*	*	*

Figure 6.1: Approximate tree running times (seconds)

Where a (*) appears, that combination is impractical, taking several days or weeks.

From the table it can be seen:

- With 2 branches (i.e. a 1-factor model), 20 timesteps can be done in 10 seconds, and 30 timesteps can be done in 3 hours.
- With 4 or 5 branches (allowing up to 4 factors), around 10 timesteps can be done in a minute.
- With more than 4 factors, the method becomes impractical, unless a very small number of exercise opportunities are allowed. However it may still be useful for benchmarking purposes. For example, an 8-factor, 10-timestep run could be carried out in 3 hours.

So to use this method in practice, we will have to either accept a low-factor model, or a limited number of exercise opportunities.

6.2 Performance issues

Even without changing the branching structure, there are several more possible ways to improve the speed of the non-recombining tree. These proposed methods may provide a constant factor of improvement at each node, so they do not avoid the dimensionality problem. However they are worth noting, as looking again carefully at these areas may prove worthwhile.

The implementation I have used is written in C++. While C++ is one of the fastest languages available, especially when all possible compiler optimisations are used, it may be possible to rewrite this code in a more efficient way. In particular, the code was written for clarity and some flexibility rather than optimised for any specific product.

The calculations done at each node of the tree, and so could profitably be optimised, relate to

- Calculating the updated forward rates (involving one or two drift recalculations).
- Recursively calculating the tree at each of the child nodes.
- Where exercise may be possible, calculating the exercise value in terms of the numeraire, and deciding whether to exercise or not.

6.2.1 faster updated rate calculations

For calculating the updated forward rates I used the 'hybrid' method discussed above. It is more common to use the forward Euler method which is up to twice as fast. However [HJJ2001] shows that the benefits of reduced bias outweigh the speed penalty. I believe that for the long time steps which may be required for a tree implementation, the bias of not using the hybrid method would be even more serious. Anyway, it would be easy to revert to the Forward Euler method if desired.

6.2.2 Drift calculations - Joshi's reduced-factor hint

Now looking at the method used to calculate the drifts at each node, I use the usual form of the B.G.M. equation which generates n random deviates based on $r \leq n$ factors, then proceeds to calculate drift adjustments for each rate based on a combination of these n rates. The drift adjustments therefore take $O(n^2)$ floating point operations on each node.

However Joshi [Joshi2002] has observed that, when working with a reduced-factor model, it is possible to rewrite the order of calculation to allow the use of $O(rn)$ multiplications. Clearly where $r << n$, as it would be when working with a 3-factor approximation for 40 quarterly rates out to 10 years, this would be a significant speed up.

I have not yet implemented this due to time constraints, but again it seems worth doing.

6.2.3 Exact martingale conditioning method - problems

In Jäckel's [Jäckel2000A] original paper, he proposes using the 'Exact Martingale Conditioning' method. The idea of this is that, since the ratio of any traded asset divided by the numeraire asset is a martingale, he adjusts the drifts so that the average value across each of the child branches equals the value at the parent branch. He finds this makes his tests approximately twice as quick. I implemented the method he provides, as a comparison to the Forward Euler method. While I did find some speedup, it was not by the same amount. However I did find that using the E.M.C. method gave incorrect values when payments were made on dates not the same as the natural payoff date of the numeraire. This is probably because I would need to re-derive the drifts in the E.M.C. method for this case, which I have not been able to do so far.

However if the E.M.C. method were chosen, the relevant drift adjustment coefficients would only need to be calculated in one location in the code, so this would be worth looking at again.

6.2.4 Calculation of child nodes

The branching structure using the simplex is precalculated, and the drift adjustments are determined as in 6.2.1. Then recursively calculating the value at each of the child nodes requires a new copy of the forward rates to be created and passed in to the children. Since this is done recursively, there are technical issues involved. In particular, a lot of new sets of forward rates are created and destroyed on the stack. This raises issues of memory allocation and efficiency.

Memory use is not a significant problem. Only one extended branch of the tree will be 'alive' at any given time. Once the child nodes have been calculated, all the parent node needs to do is keep track of the continuation value on each one of its direct children. (If complicated exercise strategies, for example for compound options, were allowed, slightly more information would be needed).

However the work involved in creating and destroying all these sets of forward rates as we move from one child node to the next is significant. In a first implementation of the tree I used the STL *vector* class. Replacing this with a more efficient C-style array wrapper class showed a huge improvement. If instead of working with the stack, a pre-allocated storage area were used for the currently active branch, the code could become more efficient still, at the expense of clarity and flexibility.

6.2.5 Multi-threading - easily possible but not done

Multi-threading could also be used to speed calculations. For a Monte Carlo simulation multi-threading is trivial: sample paths can be calculated on multiple processors at once, each one using a sufficiently different random number seed. The final results can then be averaged.

For the bushy tree method, distributed processing is only slightly more difficult. Once the initial setup of the tree is done, each child node need know nothing about its 'siblings'. All it needs is its own initial set of rates, and the branching structure which is common. So for a two-branch tree with two processors, the first processor could take one branch at time 0, the second

the other, and each would report back its final estimated value.

Again I have not implemented this but it is well worth bearing in mind, as I believe that many banks have 'processor farms' available whereby work can be distributed like this.

6.2.6 Precalculation

I have coded the tree so that it can value a collection of similar products using one run of the tree (for example, options differing only by their strikes). In this case it is important to precalculate as much as possible. For example, at each node, the zero-coupon bond prices can be calculated without reference to the individual contracts, so instead of valuing the contracts simply by passing in a collection of forward rates, it makes sense to calculate the bond values before looking at the individual contract payoffs. Depending on the contracts used, this precalculation could become more involved.

6.2.7 Valuing a portfolio

The tree simulation method has been set up so that a portfolio of options can be valued on the same tree. The time taken increases linearly with the number of options, each additional option takes only about 1% of the time required for the first one.

The only restriction on the portfolio is that all the contracts must be valued with respect to the same numeraire bond, so that the drift calculations can stay the same. In practice this means that the 'average' weighted payment time on each contract should be similar. (In theory (in the continuous case), the choice of numeraire does not make any difference, but as shown in [GZ2000] it does make a small difference.)

6.3 Pruning the tree

The times in figure 6.1 assume that the same number of branches and step sizes are used throughout the tree.

However it may be possible to reduce the branching required. As stated, **??** details one way to do this by pruning. Another way would be to first to determine an approximate exercise boundary using a low-factor model. Then use this to decide how finely to branch. If an option is deep in-the-money, it will be exercised early, and so there will be no need to recurse at that node. Similarly, near the exercise boundary we may wish to take more branches to increase precision. And far out-of-the-money, we can take wider time steps as the price will not be so sensitive.

Now all this is fine as an idea, but it is complex to implement and will not give a guaranteed speed improvement. I have not done it.

Chapter 7

Tree accuracy

In this section I look at the accuracy of the tree method for pricing a number of standard contracts. I will investigate how the accuracy of the price depends on the number of time steps and factors used.

For contracts where the price is known, I compare the estimated price to the exact analytic solution. For other contracts where the price is not known, or where the analytic price is not straightforward to derive, I compare prices to those found from a Monte Carlo simulation. Finally, for contracts with early exercise opportunities, where the true value cannot easily be found, even by straightforward Monte Carlo methods, I compare values against those found by Carr & Yang in [CY97]. (They used a stochastic mesh approach together with a Monte-Carlo test).

First I give details of the scenarios I use for testing. The first one is a simple one-factor model, the same as used by C&Y. This has the initial value of all forward rates at 10% p.a., and each rate has a flat annualised volatility of 20%. Since this is a one-factor model the correlations of all forward rates are 1.

The second scenario uses volatilities and forward rates derived from GBP cap and bond market data as at August 2004. The vol surface is obtained using the Rebonato caplet vol paramaterisation. The full-rank correlation structure is given by

$$\rho(i,j) = e^{-\beta|T_i - T_j|} \tag{7.1}$$

with $\beta = 0.1$. (This is assumed, not fitted). Varying numbers of factors are used. Plots of this scenario, and a full discussion of the reasons for the use of non-flat volatilities are included in Chapter 8.

I use the same definitions of products given in [CY97]. These are

- A caplet on forward rate f_i pays $\alpha_i(f_i - K)+$ at time T_i.
- A European (payer's) swaption can be exercised on its maturity date T. Its value if exercised is that of the floating rate of interest, less the strike, applied to a nominal principal, and paid at all interest rate payment dates from T until the swap end date M.
- A Bermudan swaption is similar to the European swaption except that it can be exercised at any rate payment date from now until its maturity date T. If it is exercised, the stream of interest payments starts at the following payment date and continues until the swap end date.

For caplets, the exact price can be found from Black's formula 8.2. For swaptions, Black's formula is commonly used. It is not theoretically exact to use Black's formula to value swaptions in the LFM, because the swap rate is essentially a combination of forward rates. Each forward rate is lognormal in its own measure, but when we work with one numeraire and combine the forward rates to get the swap rate, it is not exactly lognormal. However the discrepancy is small, and Black's formula is still very useful since it allows implied volatilities to be calculated for swaptions, from their prices.

One set of graphs shows the 'implied volatility smile' derived from the prices obtained. This is useful in getting a quick idea of how the prices compare over a range of strikes. It is known as a smile because of its typical curved shape for equities; theoretically the smile in the BGM model should be flat for caplets, or almost flat for other derivatives, and this is consistent with what I observe.

Another set of graphs shows the implied densities of the underlying. Following the results of Breeden and Litzenberger [BL78], the price of a call option on an underlying at a single date can be viewed as an integral with respect to a risk-neutral pricing density for its underlying. This density can

be found by taking the second derivative of the price of a call option with respect to strike, i.e. $f(S) = \frac{\partial^2 C}{\partial K^2}$, where f is normalized so that the area under its curve is 1. This idea is useful : it shows intuitively how the discretization inherent in the use of the tree might affect its accuracy.

The method used for comparison is a Monte Carlo simulation, based on discretising the BGM equation in the same way as used for the tree, and using the same time steps. Antithetic random sampling is used. (i.e. the simulation is run twice, with the second set of normal deviates being -1 times the first, and the average is taken.)

The Monte Carlo prices plotted are based on the average of 100,000*2 paths. The upper and lower price bounds are based on plus or minus three standard deviations, found using batched means. The dotted lines in figures 7.1 through 7.7 show the implied vols derived from these bounds on the prices.

7.0.1 Accuracy - caplets - number of branches

The first test is to see if the terminal distribution of a forward rate is simulated correctly using the simplest scenario (Scenario 1). I look at a caplet on the 5 year forward rate and take 10 time steps of 6 months each.

First I look at the implied volatility smile for this caplet. We know that since the caplet is an exponential martingale, the smile should be flat at 20%.

Figures 7.1, 7.2 and 7.3 show the smile for 2,3,and 4 branches. We can see:

- The implied volatility is approximately correct for options near the money.
- The maximum absolute error in implied volatility decreases rapidly as the number of branches are increased.
- There appears to be insufficient weight given to the extreme right of the distribution. I.e. there is a slight skew.
- There is a severe problem of 'oscillation' around the correct value. This is lessened as we increase the number of branches.

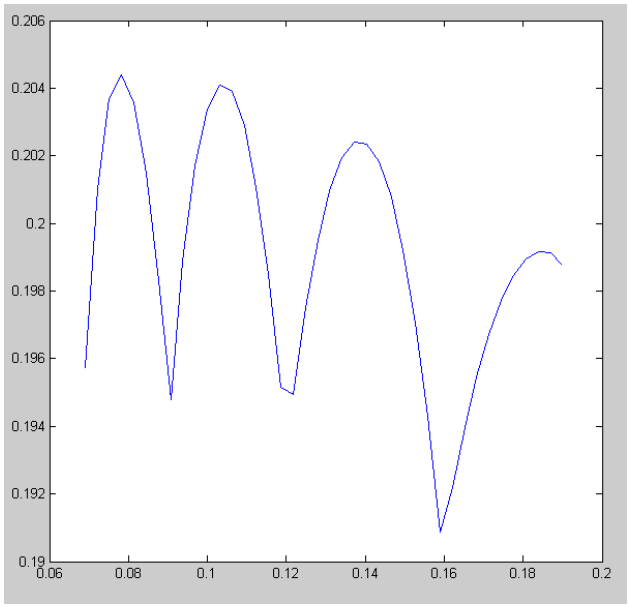

Figure 7.1: Implied Volatility - 5Y caplet - 2 branches per node

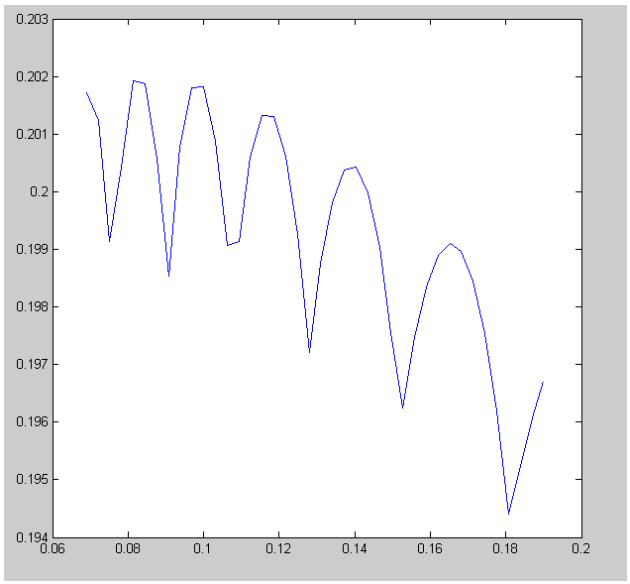

Figure 7.2: Implied Volatility - 5Y caplet - 3 branches per node

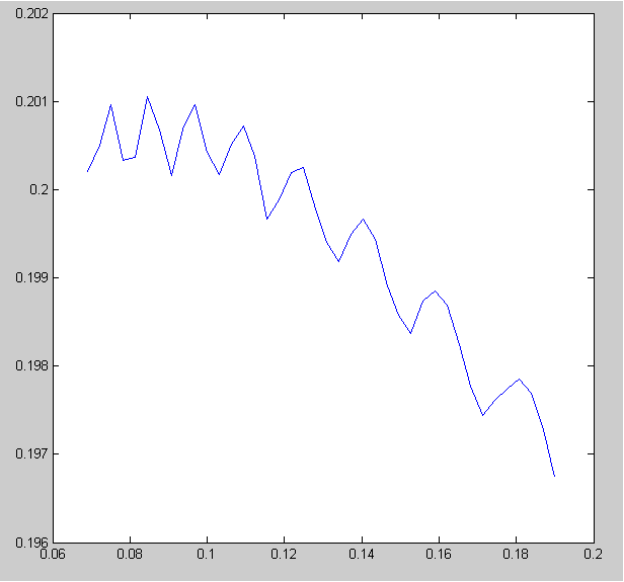

Figure 7.3: Implied Volatility - 5Y caplet - 4 branches per node

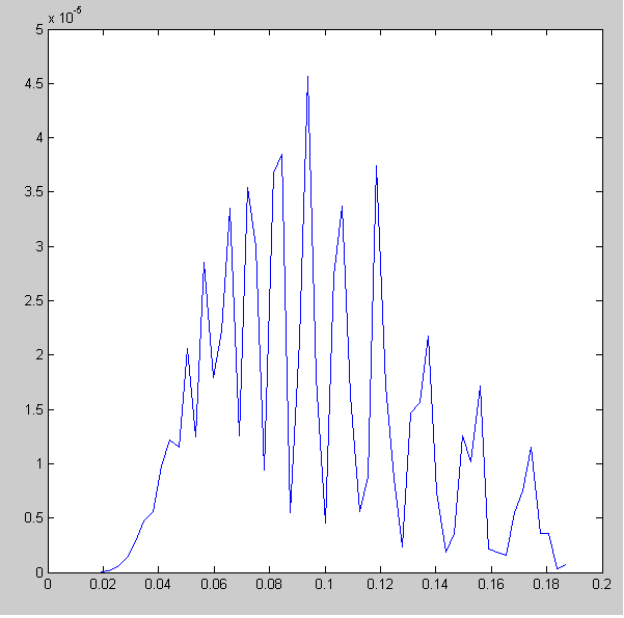

Figure 7.4: Implied Distribution - 5Y rate - 4 branches per node

To see what may be causing the oscillation, I look at the implied distribution of the 5 year rate in Figure **??**. This shows that, while the distribution appears approximately lognormal, there is a bunching effect.

This bunching effect is well known, it is caused by the nodes of the tree 'almost' recombining. For this reason I move on to Scenario 2 which has a more realistic time-varying volatility structure. (It would be possible to remedy the clustering by taking uneven time steps also. However this would involve extra complications and is unnecessary.)

7.0.2 Verification: Results for swaptions

I verify my results against those found in **??** to ensure that my method is approximately correct.

I use Scenario 1 with quarterly interest rates. I use the tree with 4 branches per node to value European and Bermudan swaptions each with (swap maturity,swap length) of (1 year,5 years) and of (3 years, 3 years).

Table 7.1: Bushy tree results versus Carr & Yang

Product	Strike	Carr/Yang	Bushy Tree	Relative difference
Euro 3X3	8%	0.047377	0.047336	-0.09%
Euro 3X3	10%	0.026246	0.026331	0.33%
Euro 3X3	12%	0.013629	0.013678	0.36%
Bermudan 3X3	8%	0.089425	0.089425	0.00%
Bermudan 3X3	10%	0.040717	0.040588	-0.32%
Bermudan 3X3	12%	0.018742	0.018483	-1.38%
Euro 1X5	8%	0.074844	0.074865	0.03%
Euro 1X5	10%	0.028138	0.028367	0.81%
Euro 1X5	12%	0.007553	0.007511	-0.56%
Bermudan 1X5	8%	0.08942	0.08942	0.00%
Bermudan 1X5	10%	0.03010	0.03032	0.72%
Bermudan 1X5	12%	0.007812	0.007675	-1.75%

Table 7.1 shows that the prices are approximately correct. All prices match to within 2%. The relative error is larger for out-of-the money options.

7.0.3 Moving to non-flat volatility

Figure 7.3 shows the strong effect of clustering, which makes it hard to identify possible bias in the tree. (This effect is even stronger in the 2-branch tree.). For this reason I move to Scenario 2.

I look at the price of European Swaptions obtained from the tree and compare them to Monte Carlo results. The rate correlations used for both methods are the same - based on a reduced-factor approximation of the correlation matrix given by 7.1.

I use 1-,2-,and 3- factor models with 2,3, and 4 branches respectively and look at a 5 X 5 swaption with semiannual interest rates. The at-the-money level is approximately 5.5 %.

Figures 7.5,7.6 and 7.7 show how the implied volatilies compare. The dotted lines in each case show upper and lower bounds based on 200,000 M.C. trials. ($\pm$ 3 standard errors).

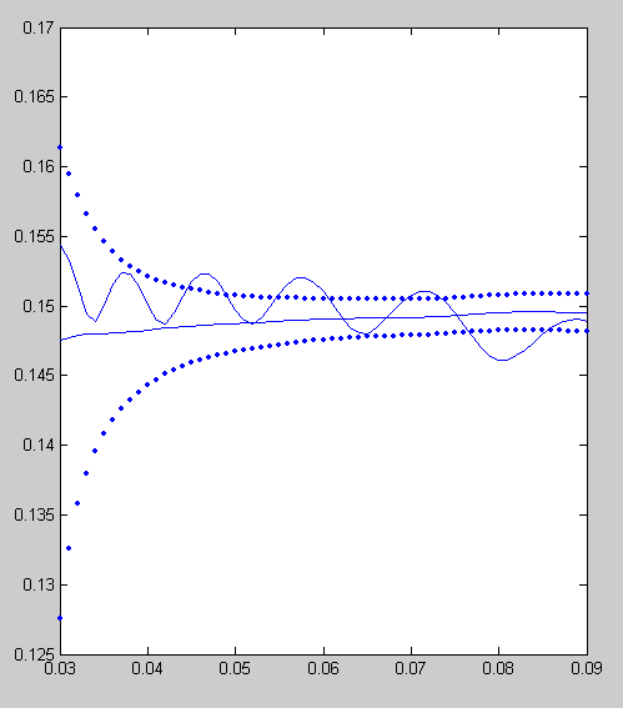

Figure 7.5: Implied Volatility - 5X5Y Euro swaption - 2 branches per node

I see that

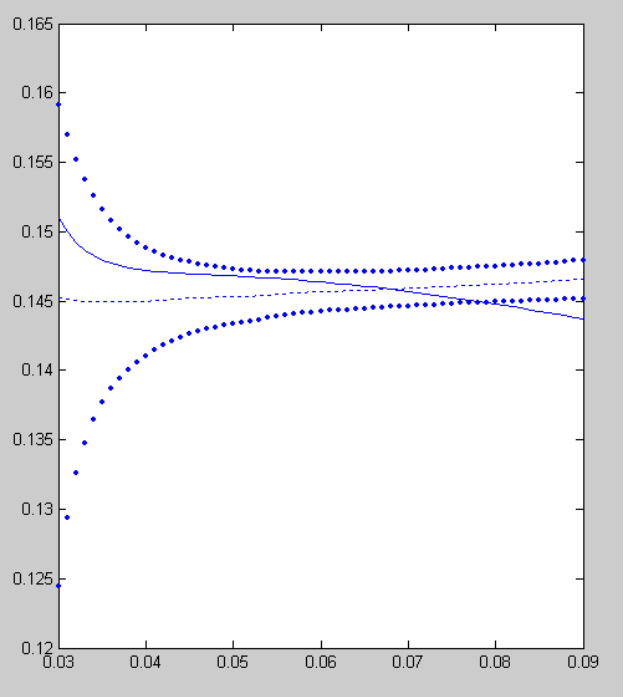

Figure 7.6: Implied Volatility - 5X5Y Euro swaption - 3 branches per node

- Even in the basic 2-branch case, the values from the tree are mostly within expected random error for the Monte Carlo method
- However, there seems to be a definite skew apparent for the tree. Lower-strike swaptions are overpriced and higher strike ones underpriced. However this effect is not significant near-the-money.

The skew occurs because the tree 'under-samples' in the tail. It might be possible to rectify this by applying an adjustment to the branching at high branch values, but I will not investigate this.

Figure 7.8 shows the implied distribution of the 5-year swap rate found from the 4-branch tree. This looks adequately smooth, in contrast to the flat-volatility case from Figure 7.4.

The time taken for each tree to run (pricing the whole range of strikes) is noted in Table 7.0.3. Notice that even the 4-branch tree with 10 steps takes only the same time as 100,000 Monte Carlo iterations.

(However, this comparison may be unfair to the Monte Carlo method.

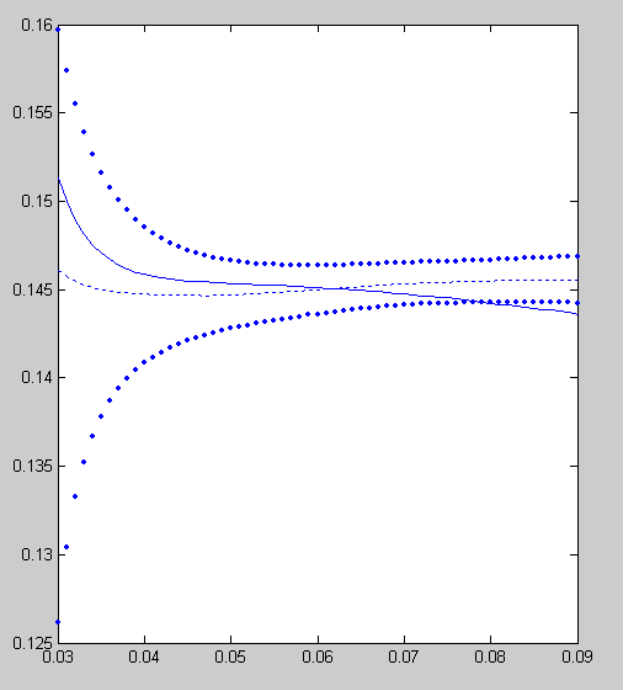

Figure 7.7: Implied Volatility - 5X5Y Euro swaption - 4 branches per node

Table 7.2: Actual running times for non-recombining tree

Method	Time
Monte-carlo (200,000 runs)	40 seconds
Tree (2 branches)	0.02 seconds
Tree (3 branches)	1.3 seconds
Tree (4 branches)	21.4 seconds

In practice low-discrepancy sequences would be used for Monte Carlo which might improve it's convergence speed.).

7.1 Improving tree accuracy

The original paper by Jäckel describes several methods which he has found to improve tree accuracy.

These are

- Optimal simplex alignment

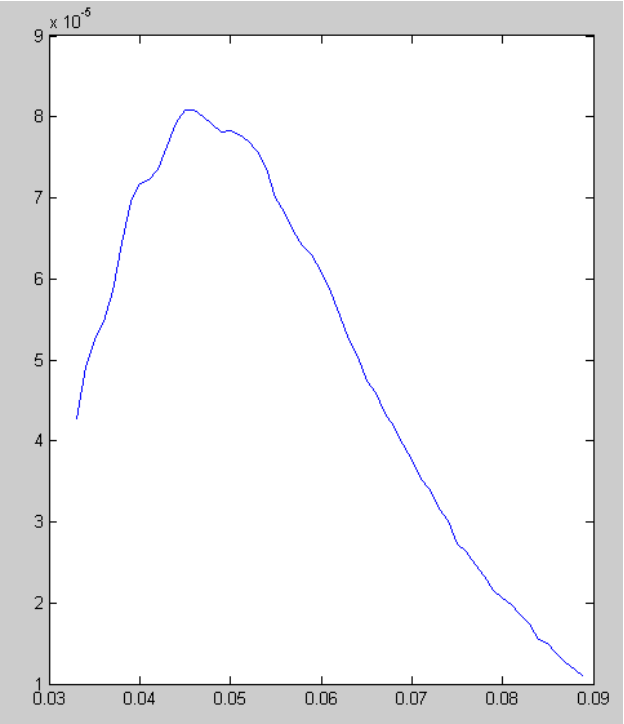

Figure 7.8: Implied Distribution at T=5 - 5Y swap rate - 4 branches per node

- Alternating simplex direction

Optimal simplex alignment is described in Section 5.2. Figure 3 of [Jäckel2000A] demonstrates a significant benefit of doing this. I did not find the difference to be dramatic but there is no harm doing it.

The "Alternating simplex direction" technique flips the direction of movement on each factor, on each alternate timestep. Again I did not see a dramatic improvement from using this method but have done it anyway.

Chapter 8

Volatility and Correlation effects

From Chapter 2 we see that the only free parameters in the LFM are the volatility and correlation structures for the forward rates. The initial level of forward rates $\mathbf{f}(0)$ can be seen in the market; The drifts of each rate fall out so as to make our chosen rate a martingale.

We want to decide on realistic volatility and correlation structures so that our structure closely fits the prices of traded assets, while remaining stable and practical to simulate.

For the non-recombining tree, spreading the volatility should mean that tree prices become more stable. Introducing correlation does not have such a good effect but I'll look at what difference it does make.

8.1 Volatility

8.1.1 Interest rate caps

An interest rate cap can be bought by a borrower to guarantee that the rate of interest he will have to pay on a loan never exceeds a given rate. A cap's payments are made at dates T_i, $0 < i \leq n$. At each date T_i the holder receives $(f_i - K)^+$ where K is the 'strike' of the cap. (Again f_i is the LIBOR payable on date T_i).

A cap can be broken down into its constituent caplets. A caplet is the payment of $(f_i - K)^+$ made at time T_i. Although caplets themselves are not traded, their prices can be determined from those of caps.

Since we are assuming that each forward rate is an exponential martingale in its own measure, caplets can be priced using Black's formula. i.e. if

$$\frac{df_i}{f_i} = \sigma_i(t)dW_t^{(i)} \tag{8.1}$$

then

$$V_t^{caplet} = \alpha_i D(t, T_i)[f_i(t)N(d_1) - KN(d_2)] \tag{8.2}$$

where $N(.)$ denotes the standard normal cumulative distribution function, and d_1 and d_2 are given in terms of the root-mean-square volatility of f_i between t and its setting time, T_{i-1}.

$$\tilde{\sigma}_i^2 = \frac{1}{T_{i-1} - t} \int_t^{T_{i-1}} \sigma_i^2(s)ds \tag{8.3}$$

$$d_1 = \frac{log(\frac{f_i(t)}{K})}{\tilde{\sigma}_i\sqrt{T_{i-1} - t}} + \frac{1}{2}\tilde{\sigma}_i\sqrt{T_{i-1} - t} \tag{8.4}$$

$$d_2 = \frac{log(\frac{f_i(t)}{K})}{\tilde{\sigma}_i\sqrt{T_{i-1} - t}} - \frac{1}{2}\tilde{\sigma}_i\sqrt{T_{i-1} - t} \tag{8.5}$$

Market practice is to quote prices for caps, not as currency amounts, but in implied volatilities. A separate implied volatility is quoted for each cap, depending on its length. From this, prices for the individual caplets can be found by the following process.

- Compute the price for a cap of length n by using the *same* implied volatility for each of its constituent caplets, and summing their prices.
- Compute the price for the cap of length $n - 1$ by using its respective volatility for each of the $(n - 1)$ caplets in it.
- Take the difference, which now allows you to determine the implied volatility for a caplet on rate f_n. (This involves inverting Black's equa-

tion using a root-finding method).

- Repeat this process until vols for all caplets have been found.

8.1.2 Rebonato's volatility paramaterisation function

Once the root-mean-square volatility $\tilde{\sigma}_i^2$ has been found for each forward rate i, this volatility can be redistributed in any way across the life of the forward rate, without affecting the price of caplets. If exotic options were traded on the forward rates (for example, an option on a forward rate which expired before that rate's setting date, paying $(f_i(s) - K)^+$ at time s, $t_{now} < s < T_{i-1}$), this redistribution could be done by referring to their prices. But these prices are not widely available so it is more common to fit a parameterized form to $\sigma_i(t)$.

It is also possible to work with the flat volatilities directly. In that case we would have $\sigma_i(t) = \tilde{\sigma}_i$ for all t. This does not seem sensible because it implies that the volatility of a forward rate is unaffected by its time to expiry. It seems more likely that rates extremely close to expiry are less volatile (e.g. because it is known that no price-sensitive events such as central bank meetings are due before their setting date), also that rates setting far in the future will be less volatile (perhaps because they are expected not to vary too far from their long-run mean).

The advantages of using a parameterized form are:

- The shape of the future term structure of volatility can be made to look similar to today.

- Summary information about, for example, the long-term expected volatility can be seen by inspecting the parameters.

- Analytic formulae exist for the covariances of any pair of rates over any desired time period (if the right type of parametrization is used).

I follow the approach described by Rebonato in [Reb2002, Ch.6,Ch.8]. He suggests a simple exponential form for the instantaneous volatility of a

forward rate. The volatility of each rate is a function of its time to expiry

$$\sigma_i(t) = g_i f(T_{i-1} - t) \tag{8.6}$$

where

$$f(s) = (a + bs)e^{-cs} + d \tag{8.7}$$

and $g(i)$ is a scaling factor which ensures that the equivalent Black volatility of each forward rate matches its target value. If this fitting process works well, g_i will be close to 1 for all forward rates.

This functional form for $f(s)$ is flexible, in that it allows for either a 'humped' or smooth instantaneous volatility. The slope, hump position, initial level, and final level can all be fitted (to a certain extent).

8.1.3 Fitting the volatility - A practical example

I first obtain the cap vols from market prices, then attempt to fit the paramaters a, b, c and d so that each $g(i)$ is as close to unity as possible. This fitting is done using the Levenberg-Marquardt minimization algorithm (see [Press et. al.1992]) which requires a sensible starting guess, but otherwise can be treated as a 'black box'.

I don't impose any constraints on the final parameters. As we will see, this can lead to unrealistic volatility surfaces, but caps are still priced correctly.

I have used two sets of recent market cap data to perform this fitting process with mixed results.

The graphs below show forward rate curves, and caplet implied volatilities derived from at-the-money cap prices, for annual periods from 1 year to 10 years. (Some of the interest rates may be slightly incorrect due to non-annual compounding - this will not affect the results significantly).

For the Euro (EURIBOR) rates, the yield curve is steeply upward sloping, and the caplet implied volatility curve shows a very steep peak at the 2-year point.

The GBP (LIBOR) rates do not have such a high peak, and the yield curve is flatter.

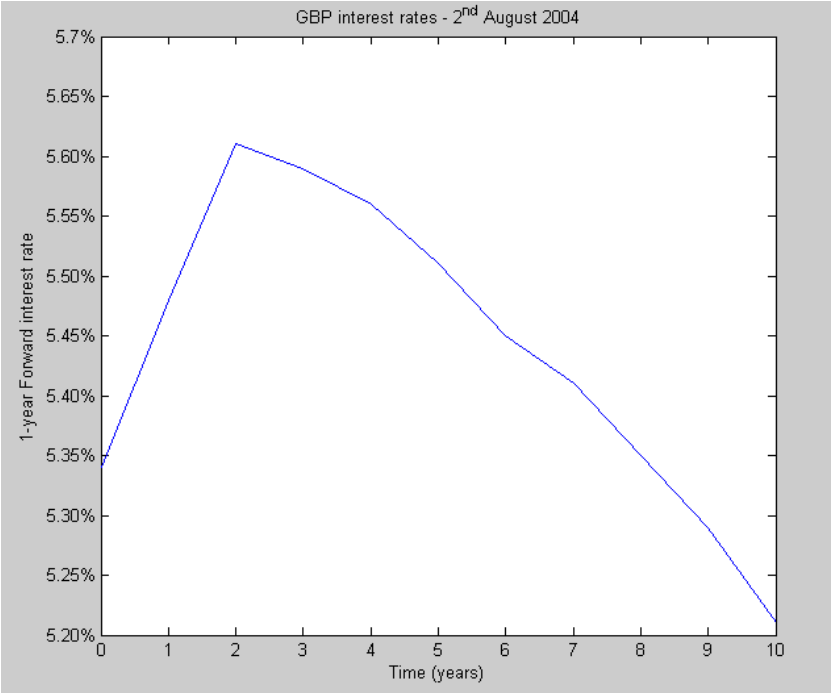

Figure 8.1: GBP forward rate curve

Figure ?? and ?? show forward rate volatility surfaces fitted using the functional form in 8.6 and 8.7. These graphs show how the volatility of each rate starts out low, then peaks as it approaches expiry, before falling again to a low level. The shape of this behaviour is similar for each rate. The height of its peak depends on the 'adjustment factor' for each rate; these adjustment factors are tabulated in figure [BLAH] together with a plot of the caplet volatilities from Rebonato's fitted form before rescaling.

The GBP vol surface looks quite possible, with long-dated forward rates having initial volatility of around 12%. However the EUR vol surface has strange behaviour. Long-dated forwards have initial volatility close to zero, which does not increase significantly until three or four years before their expiry.

This seems to be because the Rebonato functional form is not flexible enough to allow a high enough peak in the caplet implied volatility. This seems to be related to the the current low level of short-dated EUR rates; A similar problem was seen with USD rates from 2004. Both of these yield

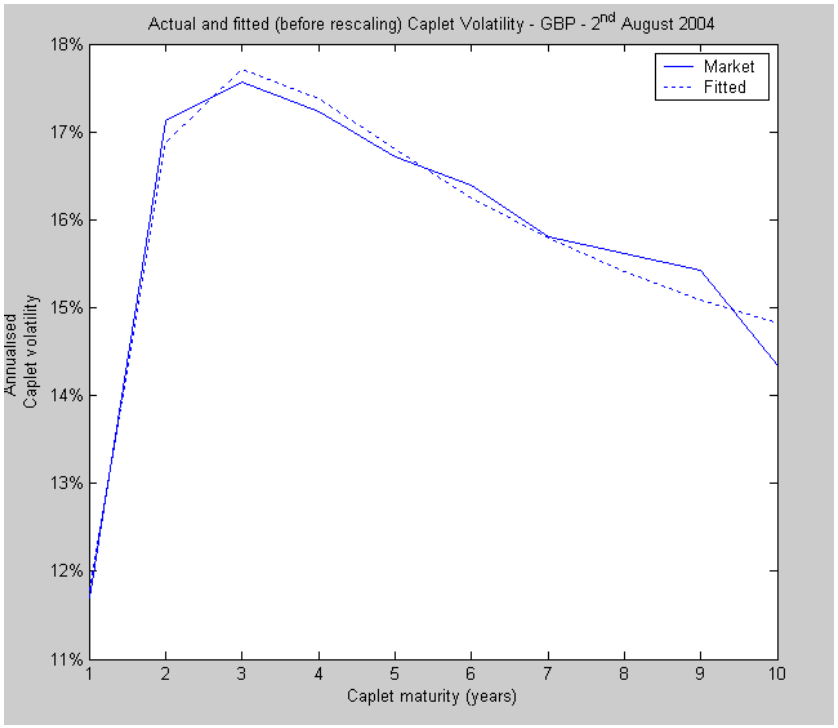

Figure 8.2: GBP Caplet volatility

curves slope steeply upwards. It may be that the log-normal distribution assumption for rates causes problems when rates are low. However this is rapidly getting out of the scope of this project.

For testing purposes with the non-recombining tree, I will stick to the GBP volatility surface which seems more realistic.

The adjuatment factor g_i required to ensure all caps are still priced correctly is shown for the GBP caps in Table 8.1.3. Since the fit is quite good, all adjustment factors are close to 1.0.

8.2 Correlation and factor dependence

8.2.1 Factor reduction in theory

Equation 2.3 describes the evolution of the forward rates in the general case, where we have n underlying Wiener processes for our n rates. The correlation

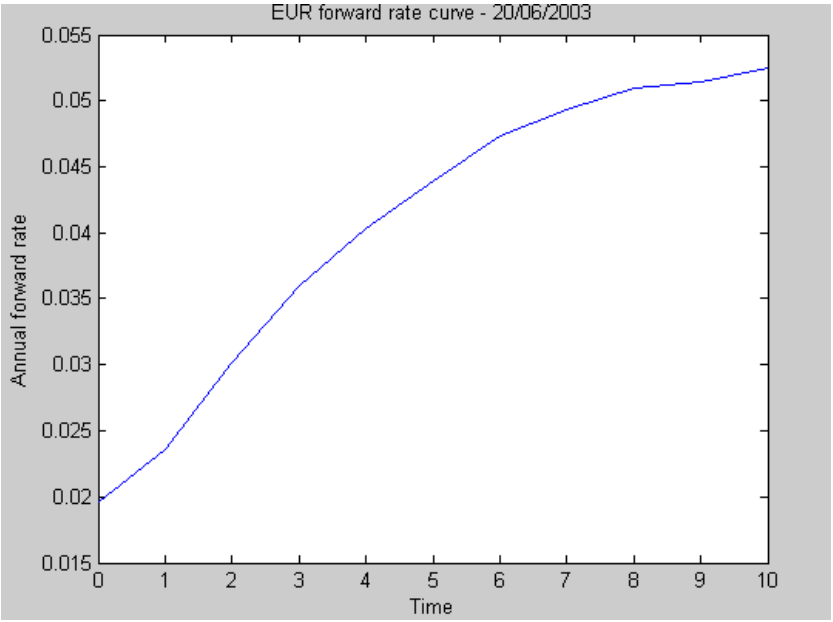

Figure 8.3: EUR forward rate curve

matrix for these processes ρ (from equation (2.2)) can have rank $r \leq n$.

Where ρ has rank $r < n$, it is possible to find a pseudo-square-root $\mathbf{Q}$ such that $\rho = \mathbf{Q}\mathbf{Q}^T$, where $\mathbf{Q}$ is of size n-by-r. $\mathbf{Q}$ can be found by various similar methods such as Cholesky decomposition or Symmetric Value decomposition.

Even where ρ is of full rank, it is possible to find an approximate matrix $\tilde{\mathbf{Q}}$ such that $\rho \approx \tilde{\rho} = \tilde{\mathbf{Q}}\tilde{\mathbf{Q}}^T$. This fitting of a target matrix $\tilde{\mathbf{Q}}$ can be carried out in a number of different ways, depending on what is wanted by a "close" fit.

The key reason for doing this is that it allows the use of an r-dimensional driving process for our simulation, thus allowing the use of a tree with fewer branches. If all factors are retained, it is not possible to sample evenly across each factor, and our tree method would lose its essential advantage of simplicity, which involves knowing the exact value of each factor at each node on the tree.

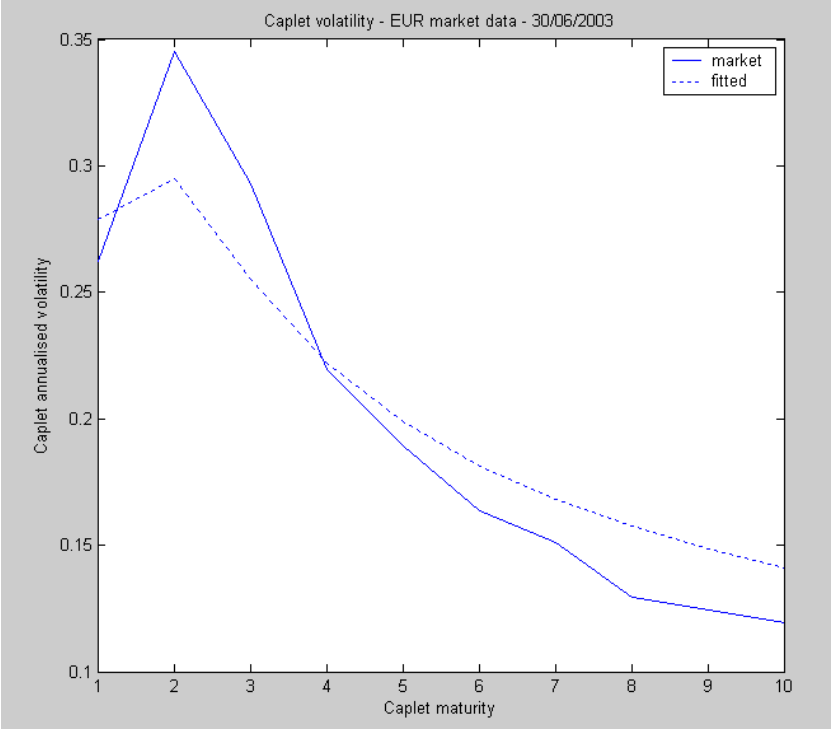

Figure 8.4: EUR Caplet volatility

8.2.2 Factor reduction in practice

[Reb99] reviews the issues involved in fitting a reduced-factor correlation matrix and proposes a method for finding the 'best' target, which requires one to

- Specify a 'quality-of-fit' function for possible matrices
- Use a geometric approach and vary a range of possible rotation angles so that the quality function is optimised.

However it states that the results from this method are quite similar to those found from a Principal Components analysis.

The Principal Components method works by:

- Orthogonalizing the existing correlation matrix
- Sort the eigenvectors of the "square-root" matrix by eigenvalue and keep the r largest

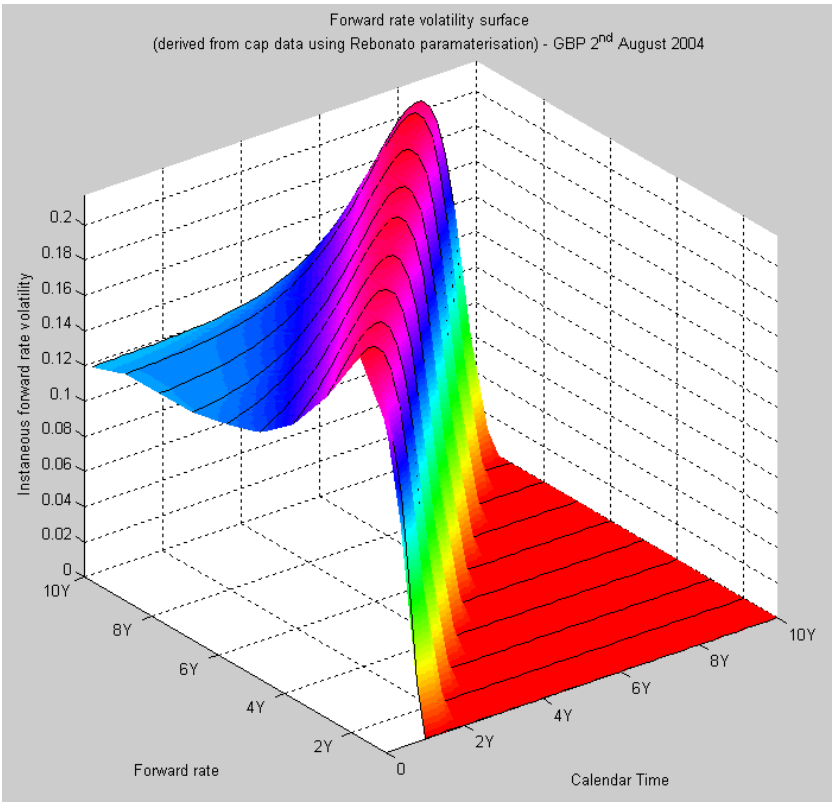

Figure 8.5: GBP forward volatility surface

- Rescaling the remaining eigenvectors so that the diagonal of the reduced correlation matrix remains unchanged

This is the method I have used, and it works reasonably well. It is not difficult computationally and it is easy to understand. However it shows up a few intrinsic drawbacks of factor reduction: It cannot match some of the common characteristics of correlation matrices using a small number of factors.

I now show the effect of factor reduction on correlation matrices themselves - I will describe the effect on simulated prices in a later section.

I have not got actual market correlation surfaces. From discussions with my supervisor, it seems that calibrating a market model to correlations is quite a tricky problem. It also seems that the level of swaption prices is not very sensitive to the exact shape of the correlation matrix. So I will use a

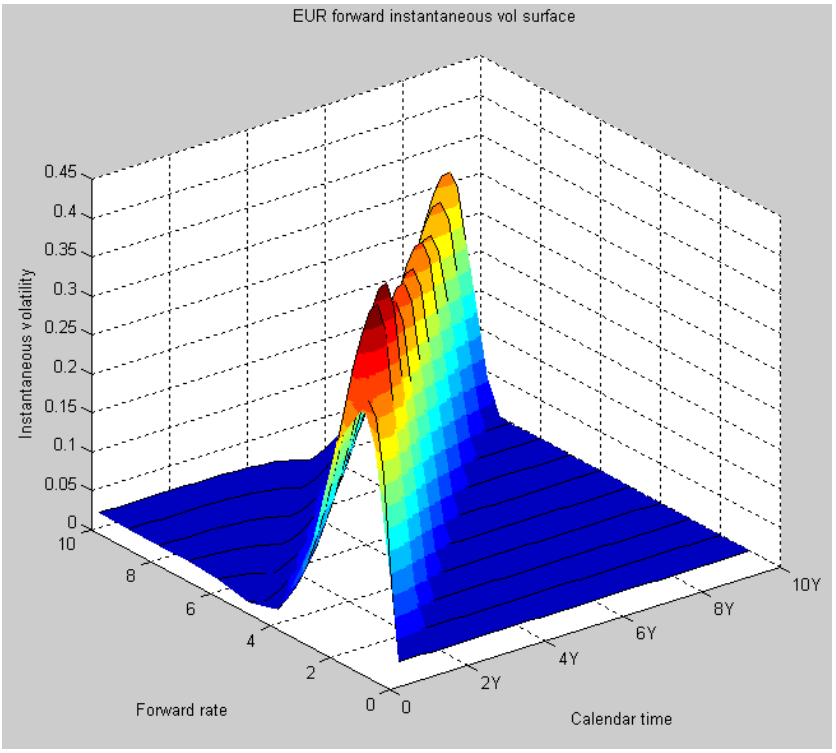

Figure 8.6: EUR forward volatility surface

simple functional form for the correlation that has 'nice' properties.

By definition the correlation of a forward rate with itself is 1. It seems reasonable that forward rates maturing at close dates will have high correlation, and that correlation will decay as their distance increases. It seems natural that the correlation matrix will be time-homogenous. This allows the decomposition to be done once; it also means that the covariance terms in the discretised drift equations become simpler.

Rebonato suggests a function of the form $\rho(i,j) = e^{-\beta|T_i - T_j|}$ is reasonable, with a typical value of $\beta = 0.1$.

8.2.3 Reducing the correlation matrix - example

Figure X shows a plot of a correlation matrix for rates out to 10 years using the structure in 7.1, together with its 3-factor approximation.

This shows how a factor-reduction using principal components analysis

Caplet	Adjustment
1Y	0.992
2Y	1.014
3Y	0.991
4Y	0.991
5Y	0.995
6Y	1.009
7Y	1.002
8Y	1.014
9Y	1.022
10Y	0.967

captures the broad shape of a correlation matrix. However there are features (such as convexity across the main diagonal) that it cannot capture. (I think this is an extreme example, because true market correlation matrices do not look as 'pointy' as the ideal one in Figure 8.7.

8.2.4 Reducing the covariance matrix instead?

Instead of reducing the correlation matrix once at the start, it would be possible to reduce the covariance matrix at each timestep. This has the advantage of having a closer fit to the true covariances of the remaining rates. From talking to my supervisor it seems more common to reduce the correlation matrix as this has nicer theoretical properties. I have not been able to find a good discussion of which method is better. In practice I found that the covariance-decomposition method had a visually better fit to the covariance matrix (as would be expected) but that the difference is small.

For testing I have left it as a correlation decomposition as this is more standard. It can be changed later if desired.

8.3 Factor dependence of prices

In Chapter 6 I showed that the choice of the number of factors to use dramatically affected running time.

I'll now look again at how the number of factors used affects estimated contract values.

There is some disagreement about whether a large number of factors are required to value swaptions correctly. Andersen & Andreasen in [AA2000] claim that Bermudan swaptions "change only moderately (and in fact typically *decrease* slightly) as the number of factors is increased from one two two". This article was written in response to Longstaff, Santa Clara, and Schwarz [LSS2000] who claimed that exercise strategies based on a single-factor model were significantly suboptimal.

For practical reasons, I will look at how the price of *European* swaptions depends on the number of factors used, using the same volatility and correlation structures as used elsewhere in this report - the 'GBP' scenario from Chapter 7. I saw similar results from using the tree with up to 4 factors.

Figure 8.8 shows the implied volatility of a 5 X 5 year European Swaption struck at 5.5% which is approximately at-the-money. The upper and lower confidence bounds are based on 200,000 paths of a Monte Carlo simulation. All the estimates use the same random seed so the shape of the curve should be due only to the number of factors used.

What it shows is that the volatility decreases significantly with the addition of a second factor. The effect of adding more factors is much smaller. Indeed each extra factor beyond the second makes a small difference, which is well within estimated standard error. However the plot does not 'level out' at all.

In the context of the non-recombining tree, this would indicate that, at least for vanilla instruments, adding a second factor is the most important to avoid bias. Adding a small number of extra factors does not seem worthwhile.

However for instruments which might depend on certain higher-order factors of yield-curve movements, for example depending explicitly on the difference between two adjoining rates, it may be desirable to use more factors to incorporate the possibility of these movements.

I did a similar test using another correlation structure, of the form $\rho(i,j) = e^{-\beta|T_i - T_j|^2}$ which looked more like market correlation structures seen in [Reb2002, Ch.6] - i.e. flatter across the main diagonal, and flatter in the corners - but

similar effects were seen.

Basically, a low-factor approximation to the covariance/correlation matrix, when we force it to have the same main diagonal, cannot have enough curvature to match the 'weighted average' covariance needed to price swaptions.

That was for European swaptions. I also looked at Bermudan swaptions to see if reducing the number of factors had the same effect.

Table 8.3 shows the same type of difference for the tree as for the Monte Carlo method, for European swaptions. The tree uses 4 branches and from 1 to 3 factors to price a 5-year European swaption expiring in 5 years. There is a significant difference going to 2 factors and a smaller difference going to 3.

Table 8.1: 5X5 European Swaption 1/2/3 factor comparison

Strike	2f/1f price	3f/2f price	Imp Vol 2f-1f	Imp Vol 3f-2f
4.0%	99.3%	99.8%	-0.40%	-0.14%
4.5%	98.9%	99.6%	-0.37%	-0.13%
5.0%	98.3%	99.3%	-0.36%	-0.14%
5.5%	97.5%	99.1%	-0.36%	-0.13%
6.0%	96.6%	98.7%	-0.36%	-0.13%
6.5%	95.5%	98.1%	-0.37%	-0.15%
7.0%	94.2%	97.9%	-0.38%	-0.13%
7.5%	92.7%	97.2%	-0.39%	-0.14%
8.0%	90.8%	96.2%	-0.42%	-0.16%

Table 8.3 shows the effect of adding a second and third factor for the equivalent Bermudan swaption. Now the difference from adding the third factor is less than 1% in price, for a wide range of strikes. This means that the Bermudan swaption appears less sensitive to the number of factors than the European one. As expected, the early exercise feature of the Bermudan swaption is reflected in it's higher price.

So it looks like there is a small bias in using a low-factor LFM model to price swaptions. This bias, however, is not large in comparison to the standard error from a reasonably-sized Monte Carlo run. However the error could be fudged by applying an adjustment to all the volatilities so that a

Table 8.2: 5X5 Bermudan Swaption 1/2/3 factor comparison

Strike	2f/1f price	3f/2f price	% of equivalent Euro.
4.0%	100.0%	100.0%	218%
4.5%	99.8%	100.3%	191%
5.0%	98.4%	99.9%	175%
5.5%	96.4%	100.5%	163%
6.0%	95.5%	99.8%	154%
6.5%	94.2%	98.8%	145%
7.0%	91.3%	100.0%	139%
7.5%	90.2%	98.4%	133%
8.0%	87.3%	97.5%	127%

desired set of swaptions were priced correctly, although this would mean that caplets were no longer priced correctly.

On the other hand, the speed and simplicity advantages of using the low factor model might mean that this get-around is acceptable. For example, if sensitivities to the main factors, and not exact pricing, was more important, this can be done easily.

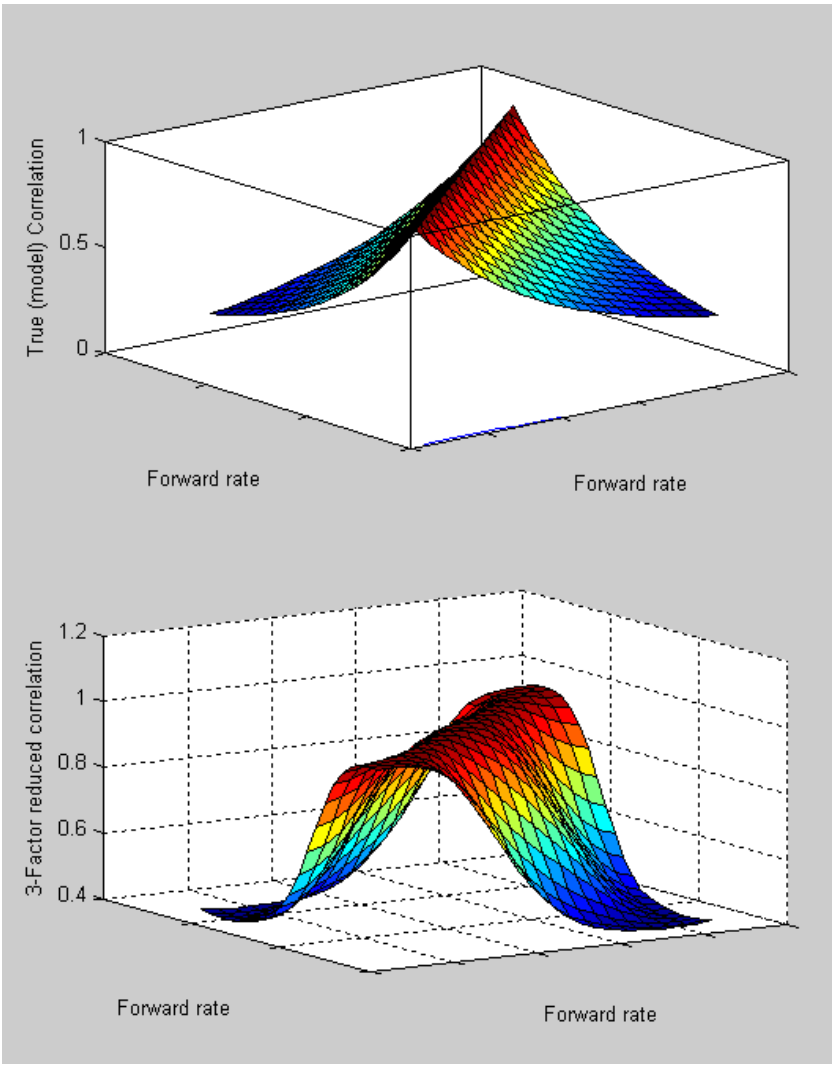

Figure 8.7: Factor reduction of theoretical correlation matrix using PCA

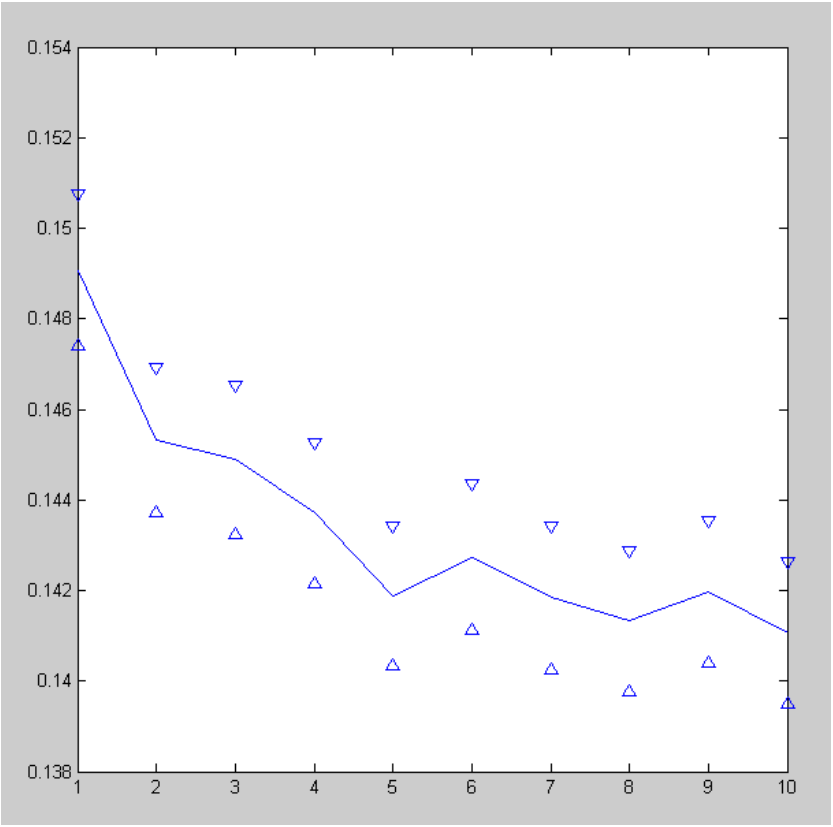

Figure 8.8: Impied volatility vs. Number of factors - ATM Swaption)

Chapter 9

Advantages of the non-recombining tree

In the introduction I briefly mentioned the main advantages of using a non-recombining tree method.

They are

- Easy to understand
- Relatively easy to construct and use
- Not requiring any customisation for different product types

In addition, they can be used to determine approximate exercise strategies, which can then be used in, for example, a Monte-Carlo simulation. Jäckel in [Jäckel2000B] gives a detailed example of how this can be done for Bermudan swaptions. Since the exercise decision at each node can be easily stored, and plotted against possible explanatory variables (such as the current swap rate), a reasonable strategy can be found by polynomial parameterisation of the exercise boundary.

In this project, I have not investigated this, but it seems a promising method.

Another possible use of the non-recombining tree method is *in conjunction* with a Monte Carlo method. For example, if a contract has 5 possible exercise

dates which are far into the future, say from 10 to 15 years, it would be quite feasible to

- Simulate forward curve using Monte-Carlo out to 10 years
- Use each of these 10-year states as the input to a separate tree which would be 5 levels deep
- Take expected values in the usual way

This would avoid some of the explosion effect seen in using a non-recombining tree out to long periods. The key advantage of the tree is in being able to recursively calculate an exercise strategy, but when no exercise opportunities exist, it may be wasteful to use a tree.

Chapter 10

Interpolation/Extrapolation

10.1 The need for approximation

As shown in Chapter 6, the non-recombining tree method is effectively impractical for more than about a dozen exercise opportunities, unless we restrict our model to a one-factor one, in which case up to about 30 points can be used.

However even 30 time points is insufficient to fully model several common contracts, for example, a 10-year Bermudan swaption with quarterly exercise dates.

Various possible approaches can be used for approximating such a contract by looking at coarser exercise times.

Due to lack of time, I have not developed this idea fully. However I wanted to flag it for the future as it could be useful.

The issues here are

- Branching our forward rates correctly over wider timesteps
- Interpolating the results of the coarse tree back to the fine tree

The adjustments required to simulate the LIBOR rates over non-standard time periods are a bit tricky. The original derivation of 2.3 was based on the assumption of no arbitrage existing between bonds payable at each of the LIBOR setting dates. Now if we skip over several intermediate dates, the

drift adjustment over the time step should be of a similar form. We must however be careful to adjust each of the covariance terms correctly, since the variance of a rate freezes after it is set. Compounding effects will also need to be taken into account. Brigo & Mercurio discuss this point in [BM2001, Section 6.17].

Any option X with a fine range of exercise times T_X can be approximated by a similar option X' with a coarser subset of exercise times $T'_X \subset T_X$. Clearly X' is worth less than X. Assuming the optimal exercise boundary for X is relatively flat, the convergence of value of X' to X should be fairly smooth.

Various authors including Broadie & Detemple [BD1996] attempt to model this convergence by fitting a polynomial curve to the estimated values over several coarse models. A recent review is provided in [CCS2002].

In the case of an interest rate option, the exercise boundary is less likely to be flat than, for example, an equity option, due to having a time-varying yield and volatility curve, so this convergence will not be as smooth. However coarser options may still provide a useful lower bound.

I have not rederived 2.3 for non-standard periods. However I have simulated prices for a Bermudan swaption by allowing exercise only on every n^{th} date. I look at a 6-year Bermudan swaption with 6-monthly rates, and show how the value differs for monitoring intervals of 1,2,3,4,6 and 12, over a range of strikes.

Figure 10.1 shows the convergence. For options (except those which will be exercised immediately) the ratio and convergence is surprisingly good.

It shows that using a monitoring interval of 2,3,4, or even 6 periods (up to 3 years), the ratio of values for at-the- or out-of-the money swaptions is broadly constant. The ratio increases steadily as the monitoring frequency is increased. However there is an amount of 'noise', and there are 'bumps' in the curves which move. This bumpiness would no doubt be increased should a coarser simulation structure (and not just a coarser set of monitoring times) be used.

Unfortunately due to time constraints I have not fitted a polynomial curve to quantify this convergence, although by looking at the graph above

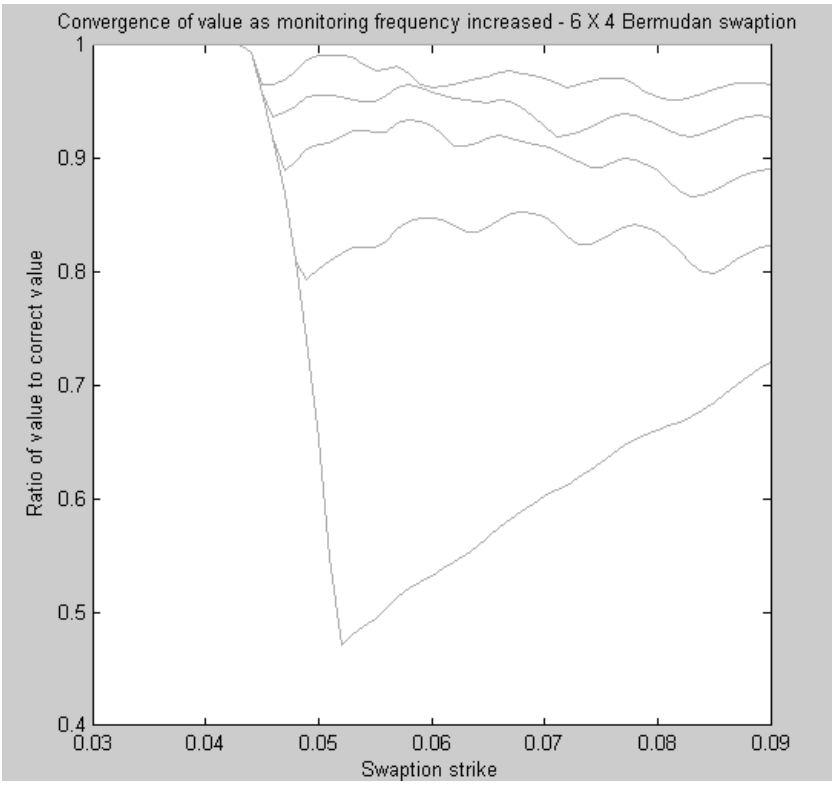

Figure 10.1: Convergence of Bermudan swaption price with monitoring frequency)

this looks very promising. However I think a 'rule-of-thumb' approach of taking the 4-period value and bumping it upwards by, say, 5%, might suffice. Or if we are interested in sensitivities rather than exact values, these may well take the same shape for the coarser model.

Chapter 11

Conclusion

The implementation of a basic non-recombining tree is relatively straightforward. It is also easy to understand as applied to the LIBOR market model. For pricing an arbitrary derivative contract, the tree method can easily be applied.

The running time of the tree is quite fast. However this is only the case for a small number of factors and exercise times. As a rule of thumb, 3 factors with 10 timesteps, or 1 factor with 30 timesteps, is practical.

I have shown it to match alternative methods such as Monte Carlo reasonably well. However some care must be taken in using it. This is mainly in ensuring that the prices and values are not 'jagged'. I have shown in one case that this can be done by one particular choice of volatility structure.

It seems necessary to use at least three branches to avoid clustering. It also seems important to use at least two factors to incorporate correlation effects. However adding more factors or branches has a less significant effect while making computation times very long. For this reason I suggest staying with a 2-factor, 3-branch tree.

There seems to be a definite 'skew' (compared to the theoretical LIBOR Market Model).when the bushy tree model is used. However this may not be a drawback in practice: Market prices themselves are quoted to include a volatility smile.

The tree does not seem to have a significant bias for at-the-money options.

I have not discussed the pricing of more exotic options using the tree. The same approach should work but the restrictions on fitting the correlation matrix may have more effect.

Approximated (or lower bounds for) prices for standard contracts with a large number of exercise dates can be found using the tree. However it is difficult to say how accurate this approximation would be. There are also some computational problems.

The use of a tree to find exercise boundaries in conjunction with a Monte-Carlo method to find accurate prices, as shown in [Jäckel2000B] seems promising.

In summary, as a 'quick-and-dirty' method, the non-recombining tree approach seems quite effective for vanilla swaptions.

Bibliography

[AA2000] L. Andersen, J. Andreasen, *Factor dependence of Bermudan swaptions: Fact or fiction* Gen Re working paper October 2000

[BGM97] Alan Brace, Dariusz Gatarek, and Marek Musiela, *The market model of interest rate dynamics*, Mathematical Finance 7 (1997), no. 2, 127-155.

[BL78] Breeden, D., and R. Litzenberger, *Prices of State Contingent Claims Implicit in Option Prices*, Journal of Business, 51, 1978

[BM2001] D. Brigo & F. Mercurio, *Interest Rate Models : Theory and Practice*, Springer, 2001.

[BD1996] Broadie, Mark & Detemple, Jerome, *American Option Valuation: New Bounds, Approximations, and a Comparison of Existing Methods*, Review of Financial Studies,Oxford University Press for Society for Financial Studies, vol. 9(4), pages 1211-50. 1996.

[CY97] Carr, P. and G.Yang, *Simulating Bermudan Interest Rate Derivatives* Working Paper, Bank of America / Numerix, 1997

[CCS2002] Chung, San-Lin, Chang, Chuang-Chang and Stapleton, Richard C., *Richardson Extrapolation Techniques for the*

Pricing of American-Style Options (March 2002). EFMA 2002 London Meetings

[CRR79] Cox, J., Ross, S., Rubenstein, M., *Option Pricing: A Simplified Approach* Journal of Financial Economics 7, p. 229-263, 1979

[Gla2000] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, Springer, 2000

[GZ2000] P. Glasserman and X. Zhao, *Arbitrage-Free Discretization of Lognormal Forward Libor and Swap Rate Models*, Finance and Stochastics 4:35-68, 2000

[HJJ2001] Hunter, C., P. Jackel, and M. Joshi, *Drift Approximations in a LIBOR Market Model*, QUARC working paper, 2001

[HK2000] Hunt, P.J. and Kennedy, J.E., *Financial Derivatives in Theory and Practice*, Wiley, 2000

[Jäckel2000A] Peter Jäckel, *Non-recombining trees for the pricing of interest rate derivatives in the BGM/J framework*, QUARC working paper, 2000

[Jäckel2000B] Peter Jäckel, *Monte Carlo in the BGM/J framework: Using a non-recombining tree to design a new method for Bermudan Swaptions*, QUARC working paper, 2000

[Joshi2002] Joshi, Mark, *Rapid Computation of Drifts in a reduced Factor LIBOR Market Model*, Wilmott Magazine, 2002

[LSS2000] F. Longstaff, E. Santa-Clara, E. Schwartz, *Throwing away a billion dollars: the cost of suboptimal exercise in the swaptions market*, UCLA working paper 2000

[McCW2002] L.A. McCarthy, N.J. Webber, *Pricing in three-factor models using icosahedral lattices* Journal of Computational Finance 5(2), 2001/2002

[Press et. al.1992] William H. Press , Brian P. Flannery , Saul A. Teukolsky , William T. Vetterling, *Numerical recipes in C: the art of scientific computing 2nd Edition*, Cambridge University Press, 1992

[Rad98A] Radhakrishnan, A.R., *Does Correlation Matter in Pricing Caps and Swaptions ?* NYU (September 1998).

[Rad98B] Radhakrishnan, A.R., *An Empirical Study of the Convergence Properties of the Non-recombining HJM Forward Rate Tree in Pricing Interest Rate Derivatives* NYU working paper, 1998.

[Reb2002] Rebonato, R. *Modern Pricing of Interest-Rate Derivatives*, Princeton U.P., 2002

[Reb99] Rebonato, R. *On the simultaneous calibration of multifactor lognormal interest rate models to Black volatilities and to the correlation matrix*, Journal of Computational Finance, Vol. 2 No. 4, 1999

[TL2001] Y. Tang & J. Lange, *A non-exploding Bushy Tree Technique and its Applications to the Multi-factor Interest Rate Market Model*, Journal of Computational Finance, Vol. 4 No. 4, Summer 2001.